2017

Deep Energy-Based Models for Structured Prediction

David Belanger

Recommended Citation
Belanger, David, "Deep Energy-Based Models for Structured Prediction" (2017). *Doctoral Dissertations*. 1030.
https://scholarworks.umass.edu/dissertations_2/1030

DEEP ENERGY-BASED MODELS FOR STRUCTURED PREDICTION

A Dissertation Presented

by

DAVID BELANGER

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

September 2017

College of Information and Computer Sciences

DEEP ENERGY-BASED MODELS FOR STRUCTURED PREDICTION

A Dissertation Presented

by

DAVID BELANGER

Approved as to style and content by:

Andrew K. McCallum, Chair

Justin Domke, Member

Marco Duarte, Member

Subhransu Maji, Member

Benjamin Marlin, Member

Alexander Rush, Member

James Allan, Chair
College of Information and Computer Sciences

ACKNOWLEDGMENTS

I am very fortunate to have been advised by Andrew McCallum. He has a keen eye for creative research problems, challenges us to think about the long-term impacts of our research, holds us to high standards, and ushers us into the broad research community. Many of the lessons I have learned from Andrew will not be implemented in software, but instead in my day-to-day interactions with my family, my peers, and my community.

When I first arrived in IESL, I was very fortunate to have Alexandre Passos and Sebastian Riedel as role models and collaborators. They taught me to develop skills in a breadth of machine learning topics, to read the Arxiv carefully, to take software engineering seriously, and to be generous with my time. Hopefully I was able to convey similar lessons to younger labmates.

A few other research friends deserve individual recognition: Luke Vilnis for being an endless supply of good ideas, Ari Kobren for his dedication to the lab and his creative research, Nick Monath for having an incredible work ethic, Arvind Neelakantan for keeping me up-to-date with the latest research trends, and Aaron Schein for being a tireless debater.

Overall, I have had the fortune of learning lessons from a tremendous group of people of IESL members over the years: Sam Anzaroot, Anton Bakalov, Trapit Bansal, Jinho Choi, Rajarshi Das, Laura Dietz, Dan Duckworth, Craig Greenberg, Pallika Kanani, Lorraine Li, Brian Martin, Arvind Neelakantan, Harshal Pandya, Pedram Rooshenas, Ben Roth, Jeevan Shankar, Sameer Singh, Michael Spector, Emma Strubell, Dung Thai, Pat Verga, Limin Yao, Mike Wick, and Jiaping Zheng.

Many thanks to the UMass machine learning professors that taught fun courses, gave me great research advice, served on my committee, and hung out with me at the climbing gym. These include Justin Domke, Akshay Krishnamurthy, Subhransu Maji, Ben Marlin, Brendan O'Connor, Dan Sheldon, and Hannah Wallach. I also had the pleasure of writing papers with Bill Freeman, Sham Kakade, and Tony Jebara, who exposed me to new machine learning perspectives.

UMass CICS is supported by a number of great administrative staff. Thanks to Leeanne Leclerc, Pam Mandler, Kate Moruzzi, and Lynn Yovina for being so patient and for enabling us to host many logistically-complicated events in the department.

I am tremendously grateful to be able to spend lots of my time with my family. My parents Ellen and Mike and my sister Rachel have shaped every detail of my life. Also, my dog Dino is an endless source of enthusiasm. Finally, during grad school I married my beautiful wife Debbi. She is generous, fun, committed, and an amazing role model for how to work hard and efficiently.

ABSTRACT

DEEP ENERGY-BASED MODELS FOR STRUCTURED PREDICTION

SEPTEMBER 2017

DAVID BELANGER

B.A., HARVARD UNIVERSITY

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Andrew K. McCallum

We introduce structured prediction energy networks (SPENs), a flexible framework for structured prediction. A deep architecture is used to define an energy function over candidate outputs and predictions are produced by gradient-based energy minimization. This deep energy captures dependencies between labels that would lead to intractable graphical models, and allows us to automatically discover discriminative features of the structured output. Furthermore, practitioners can explore a wide variety of energy function architectures without having to hand-design prediction and learning methods for each model. This is because all of our prediction and learning methods interact with the energy only via the standard interface for deep networks: forward and back-propagation. In a variety of applications, we find that we can obtain better accuracy using approximate minimization of non-convex deep energy functions than baseline models that employ simple energy functions for which exact minimization is tractable.

This thesis contributes methods for improving the speed, flexibility, and accuracy of SPENs. These include convex relaxations for discrete labeling problems, end-to-end training, where we back-propagate through the process of doing gradient-based prediction, sampling-based training, which helps explore output space, methods for regularizing SPENs such that gradient-based prediction converges quickly, and hybrid models that combine conditional random fields and SPENs.

TABLE OF CONTENTS

Page

CHAPTER

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 1

INTRODUCTION

1.1 Structured Prediction

In structured prediction, we seek to predict structured objects, which are essentially anything other than a scalar or categorical quantity. For example, these may be images, audio signals, sentences, sets of labels, database records, and graphs. In some applications, the structured object may be an intermediate representation produced by an artificial intelligence system when extracting information from observations. For example, downstream reasoning about the content of a news article may require predicting relational data about the individuals and events discussed in the text. In other applications, the structured object may be the output of a content generation system used as the interface between a computer and a user. For example, when a dialogue system responds to a user query, it may produce its response as a sentence containing multiple words, and this sentence may be further converted into an audio signal for a simulated person speaking the sentence.

Throughout the thesis, we will employ x as the input to a prediction problem and y as the structured output. In many applications, we could predict each subcomponent of y independently given x. However, this may have substantially lower accuracy than an approach that explicitly models the interactions among the subcomponents. Joint prediction of y poses computational challenges, however, as we must search in the exponentially-large space of candidate outputs. Structured prediction research focuses on posing models that both capture the data well and provide for tractable (approximate) search.

In recent years, deep neural networks have provided impressive accuracy improvements in a variety of applications. A particularly compelling aspect of deep networks is that representation learning can be performed easily by gradient descent. This replaces traditional methods where practitioners hand-design feature extraction functions using prior knowledge about the problem domain. In many structured prediction applications, deep networks are used to extract sophisticated representations for the inputs x, but practitioners use traditional methods for representing the interactions among components of the output y.

This thesis contributes methods for leveraging deep networks to learn representations for y. This provides a novel framework for structured prediction that supports a wide variety of high-performance models.

1.2 Considerations and Tradeoffs in Structured Prediction

When choosing among structured prediction techniques, practitioners consider the following factors:

1. **Expressivity:** The ability of the model to capture the underlying structure of the data. Expressive models have low approximation error.

2. **Parsimony:** The number of degrees of freedom of the model. Parsimonious models often generalize well, since they can be fit reliably on limited data.

3. **Certifiability:** Whether the method provides guarantees for either speed or exactness of prediction.

4. **Modularity:** Whether the model is able to share reusable components with other models, both in terms of learned parameters and software.

5. **Simplicity:** The ease of implementation and testing for the model.

6. **Tractability:** The computational complexity of prediction in the model.

The final factor creates many challenges. Since there are typically exponentially-many structured outputs, structured prediction is fundamentally a search problem, and different models admit prediction algorithms with varying levels of tractability.

The model selection process is often framed in terms of trading off bias vs. variance. In other words, how well the model can describe the underlying data (expressivity) vs. how vulnerable the model is to overfitting (parsimony). However, modularity and simplicity are extremely important factors in practice, to the point that the availability of open-source packages for certain algorithms has shaped the field in noticeable ways.

Performing model selection based on simplicity and modularity is not necessarily detrimental, however, as these are good engineering principles that help create reliable systems. Furthermore, it is unclear how important certifiability is in practice. For example, loopy belief propagation provides few guarantees for convergence or approximation error, but typically yields high-quality outputs. Namely, certifiability is very different than practical reliability.

Furthermore, just because a machine learning method provides user-facing simplicity and modularity does not mean that the method itself must be simplistic. Many popular paradigms in machine learning enable black-box interaction with the model: the user defines the model and sophisticated code handles learning and prediction in it. These include black-box variational inference (Nguyen & Bonilla, 2014; Ranganath et al., 2014; Kucukelbir et al., 2015; Mnih & Gregor, 2014; Rezende & Mohamed, 2015; Salimans et al., 2015), probabilistic programming (McCallum et al., 2009; Goodman et al., 2008; Goodman, 2013; Mansinghka et al., 2014; Tolpin et al., 2015), and deep learning libraries (Bergstra et al., 2011; Collobert et al., 2011; Jia et al., 2014; Abadi et al., 2016). Such approaches are attractive because practitioners can easily prototype and evaluate a variety of different models.

1.3 Approaches to Structured Prediction

A structured prediction technique has three components:

1. **Representation:** A definition of a model, which characterizes preferences for distinct candidate outputs.

2. **Learning:** How the parameters of the model are estimated on labeled data.

3. **Inference:** How to perform prediction using the model.

Here, as in many machine learning contexts, we use 'model' quite liberally. It may not be probabilistic, for example.

1.3.1 Feed-Forward Prediction

Suppose that our output y can be expressed as a collection of parts $\{y_1, \ldots, y_n\}$. The most simple structured prediction technique independently predicts each y_i using a separate model $g_i(x)$. Of course, this ignores the interactions among the different parts of y. On the other hand, in many problems these parts may be nearly conditionally independent given x. We can also share features across parts, such that each prediction is given by $g_i(F(x))$, where $F(\cdot)$ is shared. For example, when tagging tokens in a sentence with part of speech tags, we may extract features from the sentence using a bidirectional long short-term memory network (LSTM) and then predict each tag using a per-token logistic regression model with per-token features given by the hidden states of the LSTM.

Training and prediction in such models is straightforward, even when each function is expressed as a deep neural network. We refer to this approach as feed-forward because it is typically fast and prediction can be done with a single forward evaluation of F and each g_i.

1.3.2 Factorized Energy-Based Models

Alternatively, we can use an *energy-based model* (LeCun et al., 2006) to form predictions. Here, we implicitly define our input-to-output mapping as

$$\arg\min_y E_x(y). \tag{1.1}$$

$E_x(\cdot)$ is a scoring function that depends on x and assigns different values to different candidate structured outputs. As with feed-forward prediction, it can depend on x via hand-designed features or through a deep architecture.

The principal advantage of energy-based prediction vs. feed-forward prediction is that energy-based prediction explicitly models the interactions among the output parts. This is important when there are strong correlations among output parts that cannot be explained by x. An extreme example of this is when there are rigid constraints in the outputs, such as for dependency parsing: we cannot predict every dependency arc independently, since the task requires that the arcs form a tree. The tree constraint can be accounted for in an energy-based formulation, by predicting $\arg\min_{y \in \mathcal{T}} E_x(y)$, where $\mathcal{T}$ is the set of directed trees.

In order to render either exact or approximate optimization of (1.1) tractable, practitioners typically employ energy functions with some factorization structure that can be exploited to design efficient algorithms. See Sec. 2.3 for more details. *Factor graphs* assume that the energy decomposes as a sum of *factors*, terms that only depend on a subset of the components:

$$E_x(y) = \sum_{c \in C} E_x^c(y_c) \tag{1.2}$$

The factor structure provides opportunities for (approximate) energy minimization using a variety of optimization techniques. Unfortunately, these models typically exhibit poor tradeoffs between the size of the scope of the factor functions and the tractability of prediction, and prediction algorithms often need to be hand-designed to account for the specific factorization structure of the energy.

An important sub-class of factor graphs is *autoregressive energies* (Sec. 2.3.1.3). These assume a linear ordering on the N components of y such that

$$E_x(y) = \sum_{i=1}^{N} E_x^i(y_i \ , \ y_{<i}). \tag{1.3}$$

Here, $y_{<t}$ is the history of predictions preceeding component $\mathbf{y}_t$. These energies are often implemented using recurrent neural networks. The autoregressive structure enables efficient approximate energy minimization by searching in the space of prefixes of y using beam search or greedy search.

In these approaches, the ability to perform (approximate) energy minimization relies crucially on the factorization structure of the energy. In response, practitioners often use simple graph structures, but sophisticated deep features, since the functional form of the features does not affect the tractability of energy minimization with respect to y.

1.4 Structured Prediction Energy Networks

Deep learning has provided significant performance gains in a variety of application domains, largely because it enables automatic learning of sophisticated feature functions. To apply deep learning to structured prediction, prior work has mostly employed a commonly-used energy function structure, such as a linear-chain or grid factor graph (Sec. 2.3.1) and used a deep network for feature extraction. This is easy to implement, as long as the architecture for feature extraction supports back-propagation. On the other hand, it may impose an excessively strict inductive bias. Namely, practitioners are unable to use deep architectures to perform *structure learning*, representation learning that discovers the interactions among components of y.

In response, this thesis introduces introduces Structured Prediction Energy Networks (SPENs), where a deep architecture is used to extract features and also to encode the dependence of the energy on y.

Definition 1.4.1. A SPEN is an energy-based model for predicting y, given x, where y is continuous and we have an energy function $E_x(y)$ defined by a feed-forward neural network that takes both x and y as inputs and returns a scalar. To form predictions, the energy is minimized only with respect to y. The network has trainable parameters w and provides the following subroutines:

1. **Forward Propagation:** Given x and y, evaluate $E_x(y)$.

2. **Backward Propagation**: Given x and y, evaluate $\frac{d}{dy}E_x(y)$ and $\frac{d}{dw}E_x(y)$.

This is an extremely general definition that encompasses many instances of the models of the previous section. However, the focus of this thesis departs from the previous section by employing prediction and learning algorithms that do not rely on any factorization structure of the energy. Instead, we interact with it only via the standard interface for a deep network: forward and back-propagation. With this, we can perform energy minimization with respect to y using gradient descent.

Performing gradient-based prediction is advantageous because it is extremely generic. By not relying on any such factorization when choosing learning and prediction algorithms for SPENs, we can consider much broader families of deep energy functions. We do not need to specify the interaction structure in advance, but instead learn it automatically by fitting a deep network. This can capture sophisticated global interactions among components of y that are difficult to represent using a factorized energy. Essentially, much contemporary work applies deep networks to perform automatic feature learning for the input x to a prediction problem. Our work extends this to also perform automatic structure learning for the outputs y.

The SPEN energy function does not need to be a generic multi-layer perceptron. Instead, its functional form can be chosen by the user to capture known properties of the data. This can provide inductive bias that is particularly important in the limited data regime. For example, we can define the energy as the sum of local and global terms, where the global terms do not depend on x, and thus encode a learned prior over y. Of course, the downside of SPENs vs. alternative factorized energy-based models is that they provide few guarantees, particularly when employing non-convex energies.

1.5 Summary of Contributions

This thesis explores learning and prediction for energy-based models where the energy is given by a deep neural network. We focus on models where the energy is a black box that only provides forward and back-propagation, and thus we perform prediction by gradient descent. This generic approach enables exploration of a wide variety of model architectures and differs notably from popular instances of energy-based models such as factor graphs and autoregressive models, where tractability of (approximate) energy minimization depends crucially on the structure of the energy function. We explore a variety of training and prediction methods for a diverse selection of applications, including those with discrete outputs, where the SPEN is defined on the convex relaxation of the original problem. We hope to provide practitioners with the tools and insight necessary to apply SPENs to future problems.
Thesis Statement: *Accurate structured prediction can be achieved using gradient-based optimization of a learned energy function parametrized by deep neural network.*

1.5.1 Detailed Contributions

1. Definition of SPENs, example network architectures for a variety of applications (Chapter 3).

2. Gradient-based optimization algorithms for SPEN prediction (Chapter 4).
3. Structured SVM and end-to-end methods for training SPENs (Chapter 5).
4. Hybrid CRF-SPENs, where a SPEN predicts the clique marginals of a structured mean-field distribution. (Chapter 6).
5. Thorough discussion of related work (Chapter 7).
6. Investigation of SPEN properties and a comparison to CRFs on sequence and grid data (Chapter 8).
7. Applications of SPENs to the NLP problems of semantic role labeling, multi-label document classification, citation field extraction, and optical character recognition (Chapter 9).
8. Application of SPENs to image denoising (Chapter 10).
9. Prediction and learning methods for SPENs that help capture multi-modal output distributions. (Chapter 11).
10. Exploration of sampling-based large-margin methods with the goal of improving exploration vs. exploitation and accelerating learning (Chapter 12).
11. Conclusion and discussion of future work (Chapter 13).

1.6 Declaration of Previous Work

1.6.1 SPEN Papers

We introduced SPENs in

> Belanger, David and McCallum, Andrew. Structured Prediction Energy Networks. *International Conference on Machine Learning*, 2016

Follow-on work introducing improved learning methods appears in

> Belanger, D., Yang, B., and McCallum, A. End-To-End Learning for Structured Prediction Energy Networks. *International Conference on Machine Learning*, 2017

A special case of SPENs appeared first as

> Vilnis, Luke, Belanger, David, Sheldon, Daniel, and McCallum, Andrew. Bethe Projections for Non-Local Inference. *Conference on Uncertainty in Artificial Intelligence*, 2015 (with equal contribution between the first two authors)

1.6.2 Previous Work on Structured Prediction

Our development of SPENs builds on our prior work using convex and combinatorial optimization techniques to improve the speed and accuracy of strutured prediction.

> Belanger, David, Passos, Alexandre, Riedel, Sebastian, and McCallum, Andrew. Map Inference in Chains Using Column Generation. *Neural Information Processing Systems*, pp. 1844–1852, 2012

> Anzaroot, Sam, Passos, Alexandre, Belanger, David, and McCallum, Andrew. Learning Soft Linear Constraints with Application to Citation Field Extraction. *Association for Computational Linguistics*, 2014

> Belanger, David, Passos, Alexandre, Riedel, Sebastian, and McCallum, Andrew. Message Passing for Soft Constraint Dual Decomposition. *Conference on Uncertainty in Artificial Intelligence*, 2014 (with equal contribution between the first two authors)

> Tang, Kui, Ruozzi, Nicholas, Belanger, David, and Jebara, Tony. Bethe Learning of Conditional Random Fields Via Map Decoding. *Artificial Intelligence and Statistics*, 2015

CHAPTER 2

BACKGROUND

This chapter surveys a range of background material necessary for understanding the motivation and technical details of SPENs.

2.1 Notation

The following notation will be used throughout the thesis.

Symbol	**Meaning**
x	The input to a prediction problem.
y	The output of a prediction problem.
y	The continuous optimization variable for SPEN energy minimization.
w	The trainable parameters of a model that we learn.
$E_x(\bar{y})$	An energy function of $\bar{y}$, where the shape of the energy depends implicitly on inputs x.
$E(\bar{y}, F(x))$	Explicit energy function of $\bar{y}$ that depends on x by way of features $F(x)$.
θ	The natural parameters of an exponential family model.
μ	The expected sufficient statistics of an exponential family distribution.
$S(y)$	The sufficient statistics of an exponential family distribution.
Z	The partition function of a Gibbs distribution.
$H(P)$	The entropy of the probability distribution P.
$H_{\mathcal{B}}(\mu)$	The Bethe entropy of a Markov random field with marginals μ.
η	A step size employed in gradient descent.
$\langle \cdot, \cdot \rangle$	Standard inner product between vectors or tensors.
$\Delta(\cdot, \cdot)$	A cost function that measures the discrepancy between a prediction and the ground truth.

Table 2.1: Notation

2.2 Exponential Family Probability Distributions

An *exponental family probability distribution* takes the form

$$\mathbb{P}_\theta(y) = \frac{1}{Z} h(y) \exp(\theta^\top S(y)), \tag{2.1}$$

where $S(y)$ is a vector-valued function that maps y to a set of *sufficient statistics*, θ is a vector of *natural parameters*, $h(y)$ is a *base measure* that does not depend on θ, and Z_θ is a normalizing constant such that $P(y)$ sums to one. The normalizing constant is often called the *partition function.* For discrete y, we have:

$$Z_\theta = \sum_{y \in \mathcal{Y}} h(y) \exp(\theta^\top S(y)). \tag{2.2}$$

This is defined similarly, but with an integral, for continuous y. Going forward, we will use a summation for the sake of notational simplicity. We will also often omit the dependence of $\mathbb{P}_\theta$ and Z_θ on θ.

Many popular probability distributions are in the exponential family. See Tab. 2.2 for examples. Overall, this section focuses on uninvariate distributions, defined over scalars, integers, or categories. An *exponental family* is a a set of distributions for a given definition of $h(y)$ and $S(y)$.

Name	Density/Mass	Support	Sufficient Statistics	Natural Parameters
Gaussian	$\frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(\frac{-(y-\mu)^2)}{2\sigma^2}\right)$	$\mathbb{R}$	y, y^2	$\frac{\mu}{\sigma^2}$, $\frac{-1}{2\sigma^2}$
Poisson	$\frac{\lambda^y \exp(-\lambda)}{k!}$	$\mathbb{N}$	y	$\log(\lambda)$
Bernoulli	$p^y(1-p)^{1-y}$	$\{0, 1\}$	I[y = 1]	$\log(p)$
Categorical	$\prod_{k=1}^K p_k^{I[y=k]}$	$\{1, \ldots K\}$	{I[y = 1], . . . , I[y = K]}	$\{\log(p_1), \ldots, \log(p_K)\}$

Table 2.2: Examples of members of the exponential family. Here, $I[\cdot]$ is an indicator function for a predicate.

In the *minimal exponential family*, there does not exist a linear constraint that the sufficient statistics always satisfy. This is important for the identifiability of the model. A model is non-identifiable if two different values of θ yield the same distribution. In Tab. 2.2, all of the examples are minimal besides the categorical distribution. In the form we present it, the categorical distribution is not minimal because $1^\top S(y) = 1$ for every possible y, where 1 is a vector of all ones. This would be fixed by using only $K-1$ sufficient statistics, where we reconstruct $I[y = K]$ on the fly as $1 - \sum_{k=1}^{K-1} I[y = k]$.

2.2.1 Marginal Inference

Marginal inference computes the expected sufficient statistics, often denoted as μ:

$$\mu := \mathbb{E}_{\mathbb{P}}\left[S(y)\right]. \tag{2.3}$$

$$= \sum_y \mathbb{P}(y) S(y) \tag{2.4}$$

$$= \frac{1}{Z} \sum_y \exp(\theta^\top S(y)) S(y) \tag{2.5}$$

$$= \nabla_\theta \log \left(\sum_y \exp(\theta^\top S(y)) \right) \tag{2.6}$$

$$= \nabla_\theta \log Z \tag{2.7}$$

Note that $\log Z$ is a convex function of θ, since the log-sum-exp function is convex (Wainwright & Jordan, 2008). As a consequence, for elements of the minimal exponential family, there is a one-to-one correspondence between values of θ and values of $\nabla_\theta \log Z$. In other words, there is a one-to-one correspondence between the expected sufficient statistics of an exponential family distribution and its natural parameters.

This correspondence is useful because certain properties of $\mathbb{P}_\theta(y)$ are easier to reason about in terms of μ. For example, define the entropy of a distribution $\mathbb{P}(y)$ as

$$H(\mathbb{P}) = -\sum_y \mathbb{P}(y) \log \mathbb{P}(y) \tag{2.8}$$

We use the shortand $H(\mu)$ to refer to the entropy of the distribution that has marginals μ. With this, we can characterize marginal inference as the solution to an optimization problem:

$$\mu = \arg\max_{\mu' \in \mathcal{M}} \theta^\top \mu' + H(\mu'). \tag{2.9}$$

This follows immediately from the conjugate duality between the log-sum-exp and negative entropy functions (Wainwright & Jordan, 2008). The *marginal polytope* $\mathcal{M}$ is defined as the set of all vectors of expected sufficient statistics μ that are realizable from some set of natural parameters θ.

For simple univariate exponential family models, posing marginal inference as an optimization problem may seem un-necessarily complex. However, for structured distributions with exponentially-large support, exact marginal inference is intractable and (2.9) provides a foundation for designing approximate inference algorithms based on methods for approximate optimization.

Finally, throughout the thesis we will use the SoftMax function:

$$\text{SoftMax}(\theta_1, \ldots, \theta_n)[i] = \frac{\exp(\theta_i)}{\sum_j \exp(\theta_j)} \tag{2.10}$$

It inputs n values and outputs n values. Here, we provide the value of the softmax and a certain output index i. Note that SoftMax computes the expected sufficient statistics of the categorical distribution with natural parameters $\theta_1, \ldots, \theta_n$. Here, we do not employ the minimal parametrization for the distribution, and thus the SoftMax is not one-to-one. Namely, the output of the SoftMax is invariant to adding a constant to all of the inputs.

2.2.2 Maximum Likelihood Learning

A full discussion of methods for estimating the parameters of exponential family models is beyond the scope of this thesis. Here, we briefly explain *maximum likelihood learning* (MLE). Suppose we have M training examples $y_1, \ldots, y_M$. We seek to maximize the average log-likelihood of our data:

$$
\begin{aligned}
\text{LogLik}(\theta) &= \frac{1}{M}\sum_{i=1}^{M} \log \mathbb{P}_\theta(y_i) && (2.11)\\
&= \frac{1}{M}\sum_{i=1}^{M} \log \theta^\top S(y_i) - \log Z_\theta. && (2.12)
\end{aligned}
$$

When optimizing our likelihood with gradient-based methods, we have:

$$
\begin{aligned}
\nabla_\theta \text{LogLik}(\theta) &= \frac{1}{M}\sum_{i=1}^{M} \left[S(y_i) - \nabla_\theta \log Z_\theta\right]. && (2.13)\\
&= \mathbb{E}_{\tilde{\mathbb{P}}}[S(y)] - \mu. && (2.14)
\end{aligned}
$$

Here, we use $\tilde{\mathbb{P}}$ to denote the data distribution. The gradient (2.14) is conceptually attractive. It is the difference between the empirical expected sufficient statistics and the expected sufficient statistics of the distribution defined by the current value of θ. The gradient is zero when these expectations match. In fact, if we were to consider all possible distributions where the empirical and expected sufficient statistics match, and select the distribution with the maximum entropy, we would obtain exactly the exponential family distribution with parameters given by the MLE solution (Berger et al., 1996).

2.2.3 The Conditional Exponential Family

This thesis focuses on structured prediction, where we are given an x and seek to predict a y. A probabilistic formulation of such a task reasons about a conditional distribution $\mathbb{P}(y|x)$. A conditional exponential family distribution is simply a function from x to the natural parameters of an exponential family distribution:

$$
\mathbb{P}(y|x) = \frac{1}{Z_{x,w}} \exp(\theta(x)^\top S(y)) \qquad (2.15)
$$

We assume that $\theta(x) = G_w(x)$ is a parametrized function with learned parameters w. When $G_w(x)$ is differentiable with respect to w, then it is easy to extend the

MLE approach of Sec. 2.2.2 to the task of learning w. Given the gradient of the log likelihood with respect to θ, we can obtain the gradient of the log-likelihood with respect to w using the chain rule. We use the notation $Z_{x,w}$ to emphasize that the partition function depends both on x and on the parameters of the mapping from x to θ. Throughout the thesis, we will typically refer to the partition function simply as Z, however.

2.2.4 Likelihood Functions vs. Loss Functions

Consider a dataset $(x_1, y_1), \ldots, (x_n, y_n)$ of training examples for a prediction problem and a prediction function $\hat{y} = f_w(x)$ with trainable parameters w. Let $L(\hat{y}, y)$ be a loss function that measures the discrepancy between a prediction and the ground truth. In many machine learning tasks, we estimate w by minimizing the average loss:

$$\min_w \frac{1}{n} \sum_{i=1}^{n} L(f_w(x_i), y_i) \tag{2.16}$$

For a regression task, it is natural to employ the squared error $L(\hat{y}, y) = (\hat{y} - y)^2$. We can interpret this as the negative log likelihood of a normal distribution with mean given by $\hat{y}$ and a fixed variance. Here, we interpret $f_w(\cdot)$ as predicting the natural parameters of a probabilistic model. Alternatively, we can interpret the loss simply as a cost function that rewards $\hat{y}$ that are close to the ground truth. Here, $f_w(\cdot)$ returns a point estimate, not a the parameters of a distribution.

Consider a discrete problem where y can take on one of K values. We can define our loss function as the negative log-likelihood of a categorical distribution with a vector of natural parameters $a = [a_1, \ldots, a_n]$ that are predicted by $f_w(\cdot)$. The associated loss is

$$L(a, y) = -\log \mathrm{SoftMax}(a)_y, \tag{2.17}$$

where subscripting a vector by y refers to indexing a vector at the index given by the value of y. Alternatively, we can assign a non-probabilistic interpretation to this loss. Consider an arbitrary predictor that returns $\mu = [\mu_1, \ldots, \mu_n]$, a vector in the probability simplex on K things. In other words, each μ_i is positive and $\sum_i \mu_i = 1$. Then, we can define a non-probabilistic loss function

$$L(\{\mu_1, \ldots, \mu_n\}, y) = -\log \mu_y. \tag{2.18}$$

This is often known as the *cross entropy* loss. This distinction is important to understand because at various points in the thesis we employ non-probabilistic methods that directly predict elements of the K-simplex and use (2.18) as our training loss.

2.2.5 Model-Based Prediction

Given a conditional distribution $\mathbb{P}(y|x)$, there are mutiple ways to predict a value $\hat{y}$. The first, sometimes known as *MAP* inference, is to predict

$$\hat{y} = \arg\max_y \mathbb{P}(y|x). \tag{2.19}$$

MAP stands for maximum a-posteriori inference. However, it is often used as a decision rule (2.19) even in contexts where posterior inference is not being performed since no prior was imposed on y. It is also sometimes known as *most probable explanation* (MPE) inference. Note that in distributions such as a Gaussian, the mode and the mean are identical, so MAP inference can be seen as predicting the conditional expectation for y.

There are contexts in which MAP inference may be unadvisable, or even dangerous. Consider a problem where $\mathbb{P}(y|x)$ estimates the probability that a patient has a certain disease, given a set of lab results. Depending on the implications of false-negatives and false-positives, a doctor may recommend that the patient is treated for the disease even if the most likely outcome under $\mathbb{P}(y|x)$ is that the patient does not have it.

More generally, consider a risk function $R(y, y^*)$ that computes the cost associated with making prediction y when the true value is y^*. Since we do not know y^*, we integrate it out, using our estimated model $\mathbb{P}(y|x)$ to capture our uncertainty about its value. This leads to the following decision-theoretic objective:

$$\hat{y} = \arg\min_{y} \mathbb{E}_{\mathbb{P}(y^*|x)} R(y, y^*). \tag{2.20}$$

Consider a discrete prediction task, where $\mathbb{P}(y|x)$ is a categorical distribution. Here, we see that MAP inference corresponds to a choice of a 0-1 risk function, where we receive a cost of 0 for correct predictions that are exactly correct and 1 otherwise. In many structured predicition contexts, the 0-1 reward is undesirable because it does not differentiate among predictions that are partially correct, and thus we should use different prediction procedures.

Here, we have simplified the exposition by assuming that the output $\hat{y}$ we seek to predict and the distribution $\mathbb{P}(y^*|x)$ are over the same type of object. This does not need to be true in general, however. For example, $\mathbb{P}(y^*|x)$ could be defined over structured objects and we seek to predict a single scalar, as long as we have a risk $R(y, y^*)$ that accepts these types as arguments.

2.2.6 Gibbs Distributions

Exponential family distributions are instances of the broader family of *Gibbs distributions*, also known as *Boltzmann distributions*:

$$\mathbb{P}(y) = \frac{1}{Z} \exp(-E(y)). \tag{2.21}$$

Here, $E(y)$ is a general *energy function.* In a conditional Gibbs distribution, the energy function would depend on x as well as y. Note that the energy minimum is the most likely value for y. Exponential family distributions can be written as Gibbs distributions with energy $-\theta^\top S(y) - \log Z$.

Througout the thesis, we will use the notation

$$\mathbb{P}(y) \propto \exp(-E(y)). \tag{2.22}$$

This is a shorthand for (2.21), where we leave the partition function implicit. In certain contexts, it is also useful to introduce a temperature parameter τ:

$$\mathbb{P}(y) = \frac{1}{Z} \exp(-\frac{1}{\tau} E(y)) \tag{2.23}$$

As the temperature approaches zero, the distribution approaches a point mass on the energy minimum, or energy minima in the case of an energy with multiple local minima. As the temperature approaches infinity, the distribution approaches the uniform distribution. Also, note that we should not characterize exponential families as having linear energy functions. They are linear in the sufficient statistics, but the sufficient statistics may be a non-linear function of y. This is true, for example, for a Gaussian distribution.

2.3 Structured Energy Functions

We now turn to the case that y is a structured object. For example, it could be a generated image, a parse tree, etc. We assume that y has a collection of subcomponents, that we index as y_i. These correspond to individual pixels, edges in a parse tree, etc.

Factor graphs provide a useful formalism for representing energy functions over structured objects (Kschischang et al., 2001). Let C be a collection of subsets of the subcomponents of y. We have:

$$E(y) = \sum_{c \in C} E_c(y_c), \tag{2.24}$$

where each y_c is the value of y restricted to the set c and $E_c(y_c)$ is an arbitrary function. Each of the terms $E_c(y_c)$ is known as a *factor*.

Note that any energy function $E(y)$ can be written as a factor graph that includes a factor that is defined on all of y. However, generally practioners only refer to energy functions as factor graphs when the graph has non-trivial structure that can be exploited to perform efficient (approximate) energy minimization. In many cases, the tractability of energy minimization can be understood in terms of the graph's treewidth (Koller & Friedman, 2009).

2.3.1 Examples

Next, we present a few popular instances of factor graphs.

2.3.1.1 Linear-Chain Factor Graphs

Consider a sequence of discrete labels $y = y_1, \ldots, y_T$, where each y_t can take on one of K values. We define the energy for a chain-structured graph as:

$$E(y) = \sum_{t=1}^{T-1} A_t[y_t, y_{t+1}] \tag{2.25}$$

Here, the brackets indicate indexing a matrix by row and column indices. Note that the energy function only has terms that interact neighbors in the chain.

For this graph structure, exact energy minimization can be performed in $O(TK^2)$ time using the Viterbi algorithm, which performs dynamic programming (Viterbi, 1967).

For certain applications, it may be useful to reduce the number of free parameters by assuming that each A_t decouples into a local factor just for y_t and a pairwise factor, where the values of pairwise factor do not depend on t.

$$\sum_{t=1}^{T-1} U_t[y_t] + A[y_t, y_{t+1}]. \tag{2.26}$$

While (2.25) has $(T-1)K^2$ parameters, this only has $KT + K^2$.

2.3.1.2 Grid Factor Graphs

Next, we consider an $N \times M$ grid of labels indexed as $y_{i,j}$. We define a simple factor graph that interacts each label with its four immediate neighbors.

$$E(y) = \sum_{i=1}^{N} \sum_{j=1}^{M-1} A_{i,j}[y_{i,j}, y_{i,j+1}] + \sum_{i=1}^{N-1} \sum_{j=1}^{M} B_{i,j}[y_{i,j}, y_{i+1,j}]. \tag{2.27}$$

Here, the A matrices contain scores for horizontal edges in the grid and the B matrices are for vertical edges. For simplicity of notation, we ignore special indexing necessary for properly handling the boundaries. Again, we could have added explicit local factors, but these can be absorbed into the pairwise factors.

Energy minimization in graphs that have loops is NP-hard in general (Koller & Friedman, 2009). In practice, though, high quality approximate energy minimization can be often be performed using, for example, MCMC (Geman & Geman, 1984), message passing (Pearl, 1982; Yedidia et al., 2003), or graph cuts (Boykov & Kolmogorov, 2004).

2.3.1.3 Autoregressive Energies

Next, we assume a partial ordering of the N subcomponents of y, and use $y_{<i}$ to denote the set of subcomponents that are earlier than y_i in the ordering. We define an *autoregressive energy as*

$$E(y) = \sum_{i=1}^{N} E_i(y_i \ , \ y_{<i}). \tag{2.28}$$

Note that *any* probability distribution can factorized as follows:

$$\mathbb{P}(y_1, \ldots, y_N) = \prod_{i=1}^{N} \mathbb{P}(y_i | y_{<i}). \tag{2.29}$$

Consequently, autoregressive energies have been used for a number of successful deep density estimators (Larochelle & Murray, 2011; Larochelle & Lauly, 2012; Uria et al., 2014). Of course, the energy function does not need to be trained as a probabilistic model, and each factor does not need to correspond to a locally-normalized distribution.

An autoregressive energy corresponds to a fully-connected factor graph, since the final term in the sum above relies on all of y. As a result, exact energy minimization is typically intractable. However, approximate energy minimization can be performed using search in the space of prefixes of y. This has proven to be very useful, for example, on a variety of text generation tasks (Sutskever et al., 2014; Vinyals et al., 2014; Filippova et al., 2015; Venugopalan et al., 2015; Xu et al., 2015)

Given predicted values for a prefix, i.e., a set of subcomponents $y_{<i}$ for some $i > 1$, we have the energy function $E_i(y_i \ , \ y_{<i})$. We assume that this can be minimized with respect to y_i. For example, for discrete prediction problems $E_i(y_i \ , \ y_{<i})$ can be represented as a vector containing energy values for each of the values that y_i can take on. Minimizing the function can be done simply by scanning down the vector. Once we have predicted a value for y_i, we can iterate this process to predict y_{i+1}. This greedy search procedure can be extended to beam search, where we maintain a set of candidate prefixes.

Note that the linear-chain factor graph (2.25) corresponds to a first-order Markov model, where y_i only directly interacts with its immediate neighbors. However, (2.28) can depend on an arbitrarily-long history. Recurrent neural networks are useful for implementing autoregressive models, since they provide a compact means to summarize a variable-length history into a fixed-size vector.

This autoregressive approach can be applied to problems beyond those where y has a natural linear ordering. For example, we can define a partial ordering on pixels arranged in a 2-dimensional grid that sweeps from the top left to bottom right corner. Such an approach has been used for high-quality image generation (Theis & Bethge, 2015; van den Oord et al., 2016).

If we define our energy as the log of the factorized probability (2.29), then training by maximimum likelihood is straightforward, since each $\mathbb{P}(y_i|y_{<i})$ is a normalized univariate density. On the other hand, it is very challenging to estimate (2.28) without assuming that the corresponding Gibbs distribution is a product of locally-normalized distributions, or when we have missing data.

Finally, note that we are using the term 'autoregressive' quite generally. In many statistics applications, it refers to the case where the dependence of y_i on $y_{<i}$ comes by way of a linear function of a fixed-size window of $y_{<i}$.

2.3.2 Linear Parametrization for Discrete y

Next, we show how to express the energy of a factor graph over discrete y as a linear function of sufficient statistics of y. A similar approach could also be applied for some energy functions over continuous y.

We assume that each y_i can take on one of K values. Let y_i^o be a one-hot representation for y_i. In other words, it is a vector of length K that is zero except in the coordinate corresponding to the value of y_i where it is 1. The factor graph energy can be written as

$$\sum_{c \in C} E_c(y_c) = \sum_{c \in \mathcal{C}} \left\langle \theta_c, \bigotimes_{i \in c} y_i^o \right\rangle \tag{2.30}$$

$$= \sum_{c \in \mathcal{C}} \mathrm{vec}(\theta_c)^\top \mathrm{vec}(\bigotimes_{i \in c} y_i^o) \tag{2.31}$$

Here, $\bigotimes_{i \in c} y_i^o$ represents a repeated outer (tensor) product of the one-hot representations for each of the components y_i in the set c, $\langle \cdot, \cdot \rangle$ is the standard inner product on multi-dimensional tensors, and $\mathrm{vec}(\cdot)$ flattens a multi-dimensional tensor into a vector. It is fully general to use (2.30), since any function over discrete inputs can be defined as a lookup table with values for each of the possible inputs.

This linear parametrization is useful because it allows us to define exponential family distributions over structured objects (Sec. 2.3.4). Linearity enables a variety of non-probabilistic approaches as well, such as primal-dual methods for structured SVM (SSVM) learning (Taskar et al., 2004; Tsochantaridis et al., 2004) and MAP inference techniques based on LP relaxations (Globerson & Jaakkola, 2008; Rush et al., 2010; Sontag et al., 2011).

2.3.3 Conditional Factor Graphs

As in Sec. 2.2.3, it is straightforward to define a conditional factor graph, where the energy function is determined by some input variable x. For example, in our linear-chain factor graph (2.25), each matrix A_t can depend arbitrarily on x. This does not change the tractability of energy minimiztion with respect to y. In a variety of structured prediction applications, it has been fruitful to parametrize the dependence on x by way of a learned deep network. Throughout the thesis, we employ the notation $E_x(y)$ to refer to an energy function that conditions on x.

2.3.4 Markov Random Fields and Conditional Random Fields

Informally, a *Markov random field* (MRF) is defined as an exponential family distribution with a factor graph energy that is a linear function of sufficient statistics, as in (2.31). Many of the concepts from Sec. 2.2 for univariate distributions can be applied directly to MRFs. The main difference is that naive approaches to operations like computing Z are intractable, since there are exponentially-many possible y. For MRFs, the associated vector θ of natural parameters is often known as a vector of *log-potentials*.

Above, we present a informal constructive definition of an MRF, by defining a probabilistic model for a given energy function. The formal definition of an MRF operates in the opposite direction. At a high level, an MRF is defined as any joint distribution that obeys certain conditional independence relationships among sets of the subcomponents of y (Koller & Friedman, 2009). The Hammersly Clifford theorem guarantees that any positive distribution with such independence structure can be represented as a Gibbs distribution with a certain factor graph energy, where the factorization structure of the energy captures the conditional independence structure of the distribution. For the sake of this thesis, the informal definition above is sufficient.

There are a few general principles that can be used to design efficient MAP and marginal inference algorithms for MRFs. We recommend Koller & Friedman (2009) for more details. The first principle is that the graph structure can be exploited to efficiently perform *variable elimination*. Second, we can perform MCMC efficiently when the computation necessary to sample new values only considers small neighborhoods of the graph. Third, we can pose inference as constrained optimization problem and perform approximate optimization. Techniques such as mean-field inference and loopy belief propagation can be seen as approximate optimization algorithms for the dual form of marginal inference (2.9). Again, these leverage the graphical structure for efficiency.

A *conditional random field* (CRF) is simply a conditional MRF, i.e., a mapping from some input x to an MRF over y (LeCun et al., 1998; Lafferty et al., 2001). In other words, the natural parameters θ are a function of x. Note we do not train θ. We train the parameters of the function that map from x to θ.

In the exposition of Lafferty et al. (2001) and many follow-on papers, the energy is written as a function of both x and y. This is a mistake, as it suggests that the energy is used to define a joint distribution over x and y. It also often makes it difficult to account for arbitrary neural network mappings from x to θ. Also, note that CRFs are typically attributed to Lafferty et al. (2001), but many of the core technical contributions of the paper appeared earlier in LeCun et al. (1998).

Posing conditional factor graphs as CRFs is useful because it is natural to learn the parameters of the factor graph by maximizing the conditional likelihood of the data. Throughout the thesis, CRFs are used as a conceptual and experimental baseline to compare SPENs against. We do not present a full exposition on inference and learning algorithms for CRFs here, as we generally use off-the-shelf methods. See Sec. 8.4.1 for details on particular methods for training chain and grid-structured models that we have found to work well in practice. Also, as with the rest of the community, we will occasionally refer to any model with a linear factor graph energy as a CRF, even if it was not trained as a probabilistic model.

For structured models, the marginal polytope is the intersection exponentially-many linear constraints on μ, except for tree-structured factor graphs, where it can be described compactly. Various inference approaches perform constrained optimization over relaxations of the marginal polytope.

2.4 Frameworks for Structured Prediction

Next, we discuss the general families of approaches to structured prediction that are popular in the literature. For the sake of concreteness, we contrast approaches in terms of how they would be applied to *named entity recognition* (NER). Along the way, we will point out details relevant for alternative applications. See, for example, Tjong Kim Sang & De Meulder (2003) for a description of NER and how it is evaluated.

NER is an important preprocessing step for a variety of NLP tasks. Let $x = x_1, \ldots, x_n$ be a sequence of n word tokens. We seek to predict spans of tokens that correspond to named entities (people, locations, etc.). Each span can be one or more tokens long.

Each token x_i is associated with a discrete tag y_i. Since we seek to predict multi-word spans of tokens as named entities, we employ a set of possible per-token tags that is more complicated than just {none, person, location, etc.}. Specifically, we employ B-I-O tags (Ramshaw & Marcus, 1999). Here, if the set of entity types is {person, location}, we consider the tag set {O, B-person, I-person, B-location, I-location}. B stands for 'begin,' I stands for 'inside,' and O stands for 'outside.' For example, any span of tokens that corresponds to a person should start with B-person, and any subsequent tokens inside the span should be tagged as I-person. The O label is used for tokens that do not belong to a named entity.

BIO tagging for NER is a useful example for contrasting structured prediction approaches because not all methods will be able to enforce the constraint that an I tag cannot be preceeded by an O.

In all of following models, we assume that we have per-token feature vectors $F(x) = F_1(x), \ldots, F_n(x)$. These could either depend on x via a hand-engineered set of feature functions or by way of a learned deep network, such as a bi-directional LSTM (Graves et al., 2005), which is a particular *recurrent neural network* (RNN) architecture. For applications to NER, see, for example, Lample et al. (2016). By using an RNN for feature extraction, the features F_i may depend on far broader context than just the observation x_i.

We use the notation $E_x(y)$ to represent a conditional energy function. In practice, this would be implemented as function of two arguments $E(y, F(x))$, where we perform energy minimization only with respect to y.

2.4.1 Feed-Forward Prediction

A simple model for NER assumes that the energy factorizes as a sum of separate energy terms for each tag:

$$E_x(y) = \sum_i = W(y_i, F_i(x)) \tag{2.32}$$

Here, W is a function that is shared across the length of the sequence. It is useful to use the same W everywhere, as this reduces the number of learned parameters and also makes prediction invariant to certain transformations of the inputs. If we use this energy to define a Gibbs distribution, then the tags are conditionally independent given the features:

$$\mathbb{P}(y|x) = \prod_i \mathbb{P}(y_i|F_i(x)). \tag{2.33}$$

Whether we assume a probabilistic interpretation or not, prediction in this model is straightforward, as it decouples into the task of making independent per-tag predictions. It would be natural to predict each y_i by performing MAP inference with the energy $-\log \mathbb{P}(y_i|F_i(x))$. This is a trivial operation that computes the argmax of a vector. We call prediction in this model 'feed-forward' because it can be done non-iteratively and in parallel across the length of the sequence.

The restriction that the energy decouples over tags may not diminish performance much in practice, especially when we use sophisticated deep feature functions that are trained end-to-end. On the other hand, a principal disadvantage of this approach for NER is that we cannot guarantee that the outputs will be valid B-I-O when we perform independent predictions for each token, even though the model was trained on valid B-I-O data.

To perform probabilistic training, it would be natural to parametrize the conditional distribution as a multi-class logistic regression model $\mathbb{P}(y_i|F_i) = \text{SoftMax}(AF_i + b)$. Here, $AF_i + b$ can be seen as the natural parameters of a categorical distribution. MLE is easy because the loss decouples into the sum of negative log-likelihoods for per-tag models. Any learned parameters for the function that produces F from x can be updated using the chain rule. Alternatively, we could perform non-probabilistic training, where we do not pose the conditional distribution (2.33), but instead train a per-token multi-class classifier using, for example, a margin-based loss.

2.4.2 Energy-Based Prediction

In energy-based prediction, we first map from $F(x)$ to an energy function $E_x(y)$ and then perform (approximate) energy minimization with respect to y to yield a prediction.

Note that energy-based prediction is general enough to represent any possible mapping from x to y. Let $g(x)$ be an arbitrary prediction function. We can recover the behavior of g by using:

$$E_x(y) = \begin{cases} 0 \quad \text{if} \quad y = g(x) \\ 1 \text{ otherwise} \end{cases} \tag{2.34}$$

This may be a useless energy function in practice, though, since finding the energy minimum will be intractable. Furthermore, the energy function does not have nice neighborhood structure. Ideally, the energy $E_x(y)$ would be correlated with the quality of predicting y. In general, we assume that $E_x(y)$ is represented by a compact functional form that admits some (approximate) energy minimization technique that is faster and more reliable than random guessing.

The energy function may provide a parsimonious means to reward statistical regularities in the structured output y. Rather than assuming that different parts of y are conditionally-independent given f, we directly model correlations among them. We can even include a 'prior' term in the energy that does not depend on x at all. Overall, it is often easier for practitioners to specify domain knowledge about the properties of typical outputs using energy-based models than feed-forward methods.

In addition, energy-based prediction provides a natural framework for enforcing hard constraints by performing constrained energy minimization:

$$\hat{y} = \arg\min_{y \in \mathcal{Y}} E_x(y). \tag{2.35}$$

For example, $\mathcal{Y}$ can be the set of valid B-I-O sequences.

The principal disadvantage of energy-based prediction is that it is often more computationally expensive at test time than feed-forward approaches. Take, for example, a linear-chain factor graph (2.25) for NER. Energy minimization by the Viterbi algorithm is $O(KT^2)$, whereas prediction using the feed-forward method of the previous section requires T parallelizable operations that are each $O(K)$. Viterbi can be easily modified to guarantee that it always predicts valid B-I-O. For more complex factor graphs, such as grids, exact energy minimization may be intractable, and approximate minimization may still be expensive.

Of course, the feed-forward prediction process of the previous section can be interpreted as being energy-based, since the final step of performing independent MAP inference in each $\mathbb{P}(y_i|F_i)$ can be seen as solving an energy minimization problem. However, this is a trivial energy function that does not capture any direct interactions among outputs. On the other hand, we could use this exact energy function, but perform energy minimization using constrained Viterbi in order to output valid B-I-O. This would lose the speed advantages of feed-forward prediction, though.

There is a wide variety of techniques for training energy-based structured prediction models, and this thesis touches only on some of them. In Sec. 2.3.4 we briefly described how they can be trained to maximize the conditional likelihood. In Sec. 11.2 we provide further details for approximate MLE methods based on MCMC. In Sec. 7.11 we discuss alternative approaches to MLE for training Gibbs distributions, such as score matching (Hyvärinen, 2005). In Sec. 5.3 we describe structured SSVM learning. Finally, Sec. 2.4.3, Sec. 2.6 and Sec. 5.4 all explain methods for discriminatively training the energy parameters such that a particular energy minimization algorithm will work well.

2.4.3 Search-Based Prediction

Energy-based prediction can be seen as a search problem, where we search for the value of y that obtains the minimum energy. In some cases, particularly for autoregressive energies, it can be useful to employ classical search techniques, such as greedy search, beam search, or A* search. Here, we can perform search in the space of prefixes of y or local search in the space of complete outputs. This section highlights training methods that are specifically tailored to the properties of the search algorithm that will be used at test time.

As described in Sec. 2.3.1.3, many structured prediction tasks admit a natural partial ordering on the outputs, for which we can pose an autogregressive energy function. Here, the process of forming the structured output can be reduced to a sequence of univariate predictions of y_i given $y_{<i}$. This corresponds to greedy approximate energy minimization. At train time, it is easy to train this predictor, since we have $y_{<i}$ available. However, at test time we may make a prediction mistake. Since the model was trained only with correct predictions as $y_{<i}$, it may behave unreliably once a mistake has been made. A variety of training methods have been proposed to help avoid this pitfall (Daumé III & Marcu, 2005; Bordes et al., 2008;

Ross et al., 2011a; Chang et al., 2015; Ranzato et al., 2016; Gu et al., 2017; Bahdanau et al., 2017).

There are additional methods for training with respect to beam search (Daumé III & Marcu, 2005; Xu et al., 2007; Wiseman & Rush, 2016), A* search (Klein & Manning, 2003; Lewis et al., 2015; Lee et al., 2016) easy-first prediction (Stoyanov & Eisner, 2012), and search in the space of complete outputs (Doppa et al., 2014). In addition, we can directly train models such that they will perform well when used to filter the search space of other models (Weiss & Taskar, 2010; Rush & Petrov, 2012).

2.5 Deep Learning

Througout the thesis, we employ deep neural networks to perform feature extraction, to perform feed-forward prediction, and to define energy functions. See Bengio et al. (2016) for a comprehensive overview of deep learning. Here, we provide a very high-level overview mainly for the sake of establishing terminology.

In deep learning, we generally define sophisticated neural networks as the composition of simple, easy-to-test building blocks. Suppose we have a module that receives h_0 as input and returns $h_1 = f_w(h_0)$, where w are the trainable parameters of the function f. Suppose h_0 is m-dimensional and h_1 is n-dimensional.

Define the $n \times m$ dimensional *Jacobian* matrix $\frac{dh_1}{dh_0}$, where the $i-j$th entry is given by $\frac{\partial h_1^{(i)}}{\partial h_0^{(j)}}$. Here, $h_1^{(i)}$ is the ith coordinate of the output h_1 and $h_0^{(j)}$ is the jth coordinate of the input h_0. Throughout the thesis, all Jacobians will have this orientation, where coordinates of the output index the rows. All gradients of a scalar-valued function with respect to a vector-valued input will be column vectors. If we wrote them as row vectors, our equations for gradient-based optimization throughout the thesis would be covered in transposes. We will not use separate notation for Jacobians, gradients, and scalar-valued derivatives, but the shape of the object will be able to be inferred from context.

Suppose we have a downstream scalar-valued loss L. In order to support both forward evaluation and gradient-based learning, our neural network module needs to implement the following:

1. **Forward** inputs h_0 and returns $h_1 = f(h_0)$.
2. **GradInput** inputs $\frac{dL}{dh_1}$ and returns $\frac{dL}{dh_0} = \frac{dL}{dh_1}^\top \frac{dh_1}{dh_0}$.
3. **GradParameters** inputs $\frac{dL}{dh_1}$ and returns $\frac{dL}{dw} = \frac{dL}{dh_1}^\top \frac{dh_1}{dw}$.

The GradInput and GradParameters methods are applications of the multi-variate chain rule. Their specific implementation will depend on the functional form of f_w.

If we construct a scalar-valued function $g(x)$ by composing modules that all obey this API, then we can evaluate it and differentiate it with respect to both x and its learned parameters. We use *forward-propagation* to refer to the evaluation of $g(x)$. Generally, we use *back-propagation* to refer to process of evaluating both GradInput and GradParameters. Typically it is more efficient to evaluate both of them jointly than each of them in isolation. However, in some situations we may only compute one of these. If $g(x)$ is defined in terms of sub-functions, then calls to Forward in $g(x)$ will call Forward in these sub-functions. Similarly, calls to GradInput and GradParameters will call these methods in the sub-functions.

In many cases, a neural network module will never actually instantiate the matrix $\frac{df(h_0)}{dh_0}$, but instead directly compute Jacobian-vector products such as $\frac{dL}{dh_1}^\top \frac{df(h_0)}{dh_0}$ directly. A simple example is when f_v is a coordinate-wise function, in which case $\frac{df(h_0)}{dh_0}$ is diagonal.

Back-propagation is an instance of *reverse-mode automatic differentiation.* This is used for differentiating a scalar-valued function that depends on many input variables. *Forward-mode automatic differentiation* is used to differentiate a function that inputs a scalar and outputs a vector. See Baydin & Pearlmutter (2014) for a useful overview of the history of automatic differentiation and its applications in machine learning.

We use *computation graph* to refer to the graph of dependencies between the sub-functions used to define a larger function. Say, for example, we have $g(x) = g_3(g_1(x), g_2(x))$. The computation graph is diamond-shaped. First, x is fed into both g_1 and g_2. Then, the outputs of these functions are merged and fed as input into g_3. A good neural network library will execute g_1 and g_2 in parallel.

This modular approach to defining functions, evaluating functions, and differentiating functions has helped accelerate deep learning research. Each of these reusable modules can be unit tested in isolation. Furthermore, the user does not need to do any calculations by hand in order to derive the high-level functions' gradients.

2.6 End-to-End Training of Unrolled Algorithms

For many learning methods, such as MLE and SSVM, computing a single gradient of the loss with respect to the parameters requires full energy minimization with respect to y. For energy functions where exact energy minimization is intractable, one could simply use the output of approximate energy minimization as a drop-in replacement for the energy minimum. However, this may have undesirable, unpredictable consquences for learning, since the relationship between the true energy minimum and the value returned by approximate energy minimization is typically unknown.

Instead, it can be useful in practice to adopt the *direct risk minimization* principle (Stoyanov et al., 2011), also known as *end-to-end* training, where the procedure to be used at test time is trained directly by gradient descent. Here, we choose a specific algorithm for approximate energy minimization and perform training such that the particular algorithm produces high-quality predictions. This is doable whenever the output of the algorithm is a differentiable function of its inputs (Tappen et al., 2007; Stoyanov et al., 2011; Ross et al., 2011b; Domke, 2013a; Kunisch & Pock, 2013; Hershey et al., 2014; Li & Zemel, 2014; on Uncertainty in Artificial Intelligence Zheng et al., 2015)..

These works perform end-to-end training for factor graph energies that are minimized by known iterative inference algorithms, such as mean-field inference and belief propagation. We use *unrolling* to refer to the process of taking an iterative procedure that is performed for T iterations and implementing it as a long computation graph with T blocks, where each block corresponds to a single inference iteration. Overall, this deep network obeys the same interface as a feed-forward predictor, and thus it can easily be trained by gradient descent, where we perform back-propagation through the unrolled implementation of the algorithm.

Note that the inference algorithm may have hyperparameters such as a step size. In practice, a significant benefit of end-to-end training is that it produces not just an energy function, but also a prediction algorithm that has been automatically tuned specifically to perform high-quality energy minimization. This contrasts with SSVM or MLE training, where the user needs to separately tune a test-time optimization procedure.

In Sec. 7.2, we discuss related work that unrolls gradient-based optimization of generic energy functions. This contrasts with works described above, which unroll inference algorithms that are carefully tailored to properties of the underlying factor graph.

2.7 Hamiltonian Monte Carlo Sampling

In Chapters 11 and 12, we employ *Hamiltonian Monte Carlo* (Neal et al., 2011) (HMC) sampling. This section provides general background on HMC.

HMC is a Markov chain Monte Carlo (MCMC) method for sampling from continuous densities of the form:

$$\mathbb{P}(y|x) \propto \exp(\frac{-1}{\tau}E_x(y)), \tag{2.36}$$

Generally, it mixes very well, since it leverages gradients of the density. It also permits the same black-box interaction with the energy function as gradient-based optimization. HMC introduces an auxiliary 'momentum' variable p, of the same size as y, and samples from the joint distribution over y and p by simulating Hamiltonian dynamics.

We first define the Hamiltonian $H(y, p) = U(y) + K(p)$, where $U(y) = E_x(y)$ is the potential energy and $K(p) = \frac{1}{2}p^\top p$ is the kinetic energy. This corresponds to the negative log density of the product distribution:

$$P(y, p|x) \propto \exp\left(\frac{-1}{\tau}E_x(y)\right) N(p; 0, I), \tag{2.37}$$

where $N(p; 0, I)$ is the density of a standard multivariate normal distribution.

The Hamiltonian dynamics are as follows:

$$\frac{dy}{dt} = \frac{dH}{dy} \tag{2.38}$$

$$\frac{dp}{dt} = -\frac{dH}{dp}. \tag{2.39}$$

In other words,

$$\frac{dy}{dt} = p \tag{2.40}$$

$$\frac{dp}{dt} = -\frac{d}{dy}E_x(y). \tag{2.41}$$

A key property of these dynamics is that they leave the value of the Hamiltonian unchanged. As a result, it is very powerful to use Hamiltonian dynamics as the proposal distribution for Metropolis-Hastings sampling, since the acceptance probability will always be 1.

Of course, we cannot simulate Hamiltonian dynamics in continuous time on a computer. Instead, we simulate them in discrete time using numerical simulation.

Let $p(t)$ and $y(t)$ be our estimates of the variables at timestep t. Neal et al. (2011) advocates using leapfrog integration:

$$\begin{aligned}
p(t+\frac{\epsilon}{2}) &= p(t) - \frac{\epsilon}{2}\frac{d}{dy}E_x(y(t)) && (2.42)\\
y(t+\epsilon) &= y(t) + p(t+\frac{\epsilon}{2}\epsilon) && (2.43)\\
p(t+\epsilon) &= p(t+\frac{\epsilon}{2}) - \frac{\epsilon}{2}\frac{d}{dy}E_x(y(t+\frac{\epsilon}{2})) && (2.44)
\end{aligned}$$

Here, ϵ is a step size. This discretization introduces errors and is not guaranteed to preserve the Hamiltonian. Therefore, the Metropolis-Hastings acceptance ratio may be less than one.

HMC has two key hyperparamters: the step size ϵ and the number of leapfrog steps T. Increasing ϵ or T will increase the mixing rate of sampling. However, it will increase the discretization error of numerical integration, which will decrease the acceptance rate. Conversely, we can achieve an acceptance rate of nearly 1.0 by taking a few small steps, but the resulting Markov chain will have high autocorrelation. These tradeoffs are notoriously difficult to tune, and they are worse for high-dimensional problems. The 'No-U-turn' sampler of Hoffman & Gelman (2014) can improve performance, and reduce the pain of tuning, by automatically setting T.

2.7.1 HMC Sampling for Simplex-Constrained Variables

Much of the focus of this thesis is on energy functions that are defined on the convex relaxation of discrete labeling problems. Here, y is subject to simplex constraints. Namely, if each output label can take on one of D values, each coordinate of y is an element of the probability simplex on D elements.

To sample these simplex-constrained variables, we reparametrize our distribution (2.36) such that it is defined over un-normalized logits l related to y by $y =$

SoftMax(l). Using the change-of-variables formula for probability densities, this yields:

$$\mathbb{P}(l|x) \propto \exp\left(\frac{-1}{\tau} E_x(\text{SoftMax}(l))\right) |J(\text{SoftMax}(l))|^{-1}, \tag{2.45}$$

where $|J(\text{SoftMax}(l))|^{-1}$ is the determinant Jacobian of the inverse SoftMax transformation. A naive implementation will yield a singular Jacobian, since SoftMax is not invertible. To maintain invertibility, we need to represent the associated categorical distribution as a minimal exponential family, as discussed at the end of Sec. 2.2. This has $D - 1$ free variables.

CHAPTER 3
STRUCTURED PREDICTION ENERGY NETWORKS

In developing SPENs, we have the following goals:

1. Develop learning and prediction algorithms that enable black-box interaction with the energy function only by way of forward and back-propagation.

2. Use deep architectures to do structure learning, i.e., learn discriminative features of the output variable y.

3. Achieve high performance on a variety of applications.

These lead to various research questions:

1. How reliable are SPENs with non-convex energy functions?

2. How can we use SPENs to perform discrete prediction?

3. How can we best learn SPENs such that gradient-based prediction works well in practice?

4. How can we use SPENs for problems where there are hard constraints on outputs?

5. How can we ensure that SPEN prediction is fast and has low memory requirements?

6. How can we use SPENs to capture uncertainty in our predictions?

3.1 SPENs for Discrete Prediction Problems

We achieve wide applicability of SPENs by employing gradient descent as our energy minimization technique. By requiring gradients, however, the SPEN energy must be defined on continuous inputs. Therefore, for tasks with discrete outputs we apply SPENs to a convex relaxation of the problem.

Consider, for example, the problem

$$\min_{y} \quad \mathrm{E}_{\mathrm{x}}(y) \quad \text{subject to} \qquad y \in \{0, \ldots, D-1\}^L. \tag{3.1}$$

In other words, there are L labels, each of which can take on one of D values. Problem (3.1) could be rendered tractable by assuming certain structure (e.g., a tree-structured factor graph) for the energy function $E_x(\cdot)$. Instead, we consider general $E_x(\cdot)$, but optimize over the convex hull of feasible y:

$$\min_{\bar{y}} \quad \mathrm{E}_{\mathrm{x}}(\bar{y}) \qquad\qquad \text{subject to} \qquad\qquad \bar{y} \in C_{L,D}, \tag{3.2}$$

where

$$C_{L,D} := \{\bar{y} \mid \bar{y} \in [0,1]^{L \times D}, \quad \sum_j \bar{y}_{ij} = 1 \; \forall i\}. \tag{3.3}$$

The distinction between y and $\bar{y}$ is important, as obtaining a valid y from $\bar{y}$ may require rounding, or some other method for mapping onto the discrete set of feasible output labelings. Most of our experiments using simple component-wise rounding. However, our semantic role labeling experiments in Sec. 9.1 employ a combinatorial solver to convert from a soft prediction $\bar{y}$ to a discrete y that obeys various hard constraints.

Remark 1. *In the remainder of the thesis, we always use $\bar{y}$ to denote the input to a SPEN energy function. This is to emphasize that it is a continuous quantity. For* *discrete prediction problems, $\bar{y}$ differs from the discrete output y that we ultimately seek to predict.*

In addition, the thesis presents various algorithms for discrete prediction with SPENs in terms of a two-dimensional prediction variable $\bar{y} \in C_{L,D}$ that is normalized in the second dimension. However, in various applications we employ prediction variables that have a different shape in practice. For example, when tagging the pixels of an image, we have 3-dimensional $\bar{y}$.

There are two main challenges when applying SPENs to discrete prediction problems with non-trivial combinatorial constraints on outputs. First, we need a method for converting from continuous to discrete predictions that guarantees that the discrete prediction is feasible. Second, we need to ensure that the combinatorial constrains are accounted for during the continuous optimization. Otherwise, the output of continuous optimization may be far from any feasible discrete prediction. Accounting for the combinatorial constraints as soft energy terms is difficult, however, as these may introduce energy barriers that prevent optimization from adequately traversing to high-quality values of $\bar{y}$.

Finally, note that in some contexts, we may not need to convert to a discrete output, even if the problem is defined over discrete objects. Following the general decision-theoretic framework for prediction in (2.20), we may use our continuous prediction as a vector of probabilities used to minimize expected risk.

3.2 Energy Function Subcomponents

In Def. 1.4.1, we define SPENs in their full generality, using a general energy function $E_x(\bar{y})$. In practice, we have found it very useful to define $E_x(\bar{y})$ as a collection of differentiable subnetworks. This decomposition provides a variety of opportunities for improving statistical and computational efficiency of SPENs and delinates abstraction barriers that are useful in a software implementation.

At first glance, such a decomposition may seem to violate the perspective of a 'black-box energy' that we have used throughout the thesis to argue for the desirability of SPENs. This is not true. When discussing the black-box nature of SPENs, we refer to the fact that SPEN prediction and learning algorithms do not depend on model-specific structure and only communicate with the energy function via forward and back-propagation. This does not mean, however, that our energy function is defined as a monolithic multi-layer perceptron. In fact, it may be defined as the sum of terms that the practioner carefully designs to model known properties of the data. Such a decomposition can provide important inductive bias for the model, allowing it to be fit accurately on limited data.

3.2.1 Feature Network and Energy Network

Throughout the thesis, we assume that $E_x(\bar{y})$ is constructed using two sub-networks.

1. The *feature network* $F(x)$ produces a feature representation for the input.

2. The *energy network* $E(\bar{y}, F(x))$ returns a scalar-valued energy value.

The energy network is a function of two inputs. At test time we minimize the energy only with respect to the first argument.

Explicitly defining the features as the output of a separate network is useful in practice for two reasons. First, we may pretrain $F(x)$ using some auxiliary, perhaps un-supervised, task. Second, at test time we only require the gradient of the energy with respect to $\bar{y}$. Therefore, we can evaluate $F(x)$ a single time, cache this value, and avoid performing any back-propagation through this subnetwork.

3.2.2 Initialization Network

We initialize gradient-based prediction using an *initialization network* $\text{Init}(\cdot)$, that we use to predict a feasible guess for $\bar{y} = \text{Init}(F(\mathbf{x}))$. This could either be a network with learned parameters or a trivial network that returns a constant value.

3.2.3 Local and Global Energy Terms

In most of our experiments, the energy network sums global and local terms:

$$E(\bar{y}\ , F(x)) = E^g(\bar{y}\ , F(\mathbf{x})) + \sum_i E^l(\bar{y}_i\ , F(\mathbf{x})). \tag{3.4}$$

Here, i indexes the components of $\bar{y}$. $E^g(\bar{y}\ \ , F(\mathbf{x}))$ is an arbitrary deep network that provides a global function that couples components together. The local term is analogous to the local factors in a factor graph. See Sec. 11.4.2 for a case where we do not use this decoupling. For simplicity of notation (and our implementation) we assume that the same features $F(\mathbf{x})$ are used by both terms.

Since the global term is fully general, the local term could have been absorbed into it, or we could have avoided using any local terms at all. However, explicitly defining local terms is useful in practice for a few reasons:

1. We can pretrain the features network $F(\cdot)$ by using the local terms to define a simple feed-forward predictor.
2. This feed-forward predictor can also be employed as an implementation of the Init$(\cdot)$ network.
3. We can help stabilize learning by first clamping the local terms for a few epochs while updating the global terms.

For general continuous output problems, we employ

$$E^l(\bar{y}_i, F(x)) = (\bar{y}_i - G_i(F(x))^2 \tag{3.5}$$

$$\text{Init}_i(F(x)) = G_i(F(x)) \tag{3.6}$$

For relaxations of discrete problems, we employ

$$E^l(\bar{y}_i, F(x)) = \bar{y}_i^\top G_i(F(x)) \tag{3.7}$$

$$\text{Init}_i(F(x)) = \text{SoftMax}(G_i(F(x)) \tag{3.8}$$

Here, $\text{Init}_i(\cdot)$ refers to the output of the initialization network for the i-the subcomponent. $G_i(\cdot)$ may be a learned network with extra parameters. The functional forms of the energy network and initialization network are based on exponential family distributions: Gaussian for general continuous problems and categorical for relaxations of discrete problems. We can pretrain our features $F(x)$ by training the feed-forward predictor $\text{Init}(F(x))$.

3.2.4 Dropout

Dropout (Srivastava et al., 2014) is a popular method for reducing overfitting in deep networks. Activations are randomly masked to 0 during the forward pass. This prevents co-adaptation of hidden units. Dropout can be applied naturally to our feature network. However, applying it to the energy network is more difficult. Typically, in dropout a new random pattern of zeros is sampled during each forward pass. However, with SPENs we call multiple forward-backward passes in the energy network to perform energy minimization. Therefore, we would need to keep the same random pattern for all of these evaluations. This presents implementation-level difficulties, and thus none of our experiments use dropout. However, it may be worth considering in future work.

3.3 Example Architectures for Particular Problems

We now provide concrete examples of SPEN architectures for various problems. Along the way, we draw parallels between SPEN architectures and analogous factor approaches to the problems.

We denote matrices in upper case and vectors in lower case. We use $g()$ to denote a coordinate-wise non-linearity function, and may use different non-linearities, such as sigmoids and rectified linear units (ReLUs), in different places.

3.3.1 Multi-Label Classification

Let x be an arbitrary input and let $y = \{y_1, \ldots, y_L\}$ be a collection of binary labels. Multiple labels may be true for a given x.

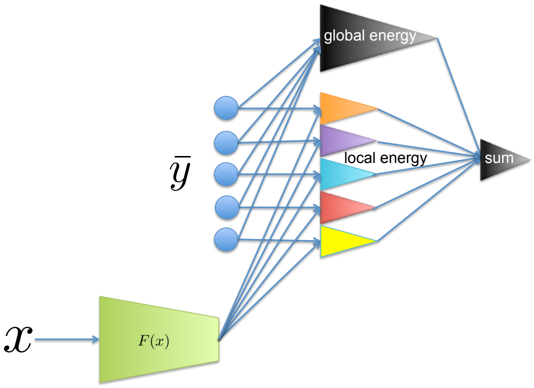


Figure 3.1: SPEN for Multi-Label Classification

Figure 3.1 depicts a SPEN architecture for this problem. For our feature network, we employ a simple multi-layer perceptron:

$$F(x) = g(A_2 g(A_1 x)). \tag{3.9}$$

The features are f-dimensional.

Our energy network is the sum of two terms. First, the *local energy network* scores $\bar{y}$ as the sum of L independent linear models for each label:

$$E_x^{\text{local}}(\bar{y}) = \sum_{i=1}^{L} \bar{y}_i b_i^\top F(x). \tag{3.10}$$

Here, each b_i is an f-dimensional vector of parameters for each label.

This score is added to the output of the *global energy network*, which scores configurations of $\bar{y}$ independent of x:

$$E_x^{\text{label}}(\bar{y}) = c_2^\top g(C_1\bar{y}). \tag{3.11}$$

In general, there is a tradeoff between using increasingly expressive energy networks and being more vulnerable to overfitting. In some of our experiments, we add another layer of depth to (3.11). It is also natural to use a global energy network that conditions on x, such as:

$$E_x^{\text{cond}}(\bar{y}) = c_2^\top g(C_1[\bar{y}, F(x)]). \tag{3.12}$$

For the sake of comparison, consider a fully-connected binary pairwise factor graph for multi-label classification. Again, we assume that the local factors depend on x, but that the values of the pairwise factors are independent of the x. Suppose that we were to apply $E_x(\cdot)$ directly to $y \in \{0,1\}^L$, rather than to the relaxation $\bar{y}$. Then, the factor graph's energy function could be expressed as the sum of a local term identical in functional form to (3.10) and a global term:

$$E_x^{\text{crf}}(y) = y^\top S_1 y. \tag{3.13}$$

3.3.2 Sequence Labeling

Consider an input sequence $x = \{x_1, \ldots, x_n\}$ of length L. We assume $F(x)$ returns an $L \times f$ matrix of per-timestep features. These can be computed using an arbitrary neural network, such as a convolutional network or a recurrent neural network.

Linear-chain factor graphs are popular models for sequence labeling. Since they are often trained to maximize the conditional log-likelihood, we will refer to them as CRFs. As in (2.26), the energy is:

$$\sum_{i=1}^{L} \psi_i^{F(x)}(y_i) + \psi_{i,i+1}^{F(x)}(y_i, y_{i+1}) \tag{3.14}$$

Here, ψ_i and $\psi_{i,i+1}$ are represented by tables of values that depend on $F(x)$. This notation is implicit in the notation going forward.

If we were to perform the $y \to \bar{y}$ convex relaxation we do for SPENs, then the CRF energy could be approximated as:

$$\sum_{i=1}^{L} \psi_i^\top \bar{y}_i + \bar{y}_i^\top \psi_{i,i+1} \bar{y}_{i+1} \tag{3.15}$$

This is an approximation because it performs a 'mean-field' approximation, where we do not explicitly reason about joint values of y_i and y_{i+1}.

With SPENs, we can extend (3.14) to be more general:

$$\sum_{i=1}^{L} E_i^{\text{local}}(\bar{y}_i) + E_i^{\text{pairwise}}(\bar{y}_i, \bar{y}_{i+1}) \tag{3.16}$$

This provides limited utility, however, as the tabular representation in (3.14) is also very general.

The difference between SPENs and CRFs is more important when we seek to represent energy functions with higher-order interactions, such as:

$$\sum_{i=1}^{L} E_i^{\text{local}}(\bar{y}_i) + E_i(\bar{y}_{i-2}, \bar{y}_{i-1}, \bar{y}_i, \bar{y}_{i+1}, \bar{y}_{i+2}) \tag{3.17}$$

Each E_i could be an arbitrary deep network.

An analogous high-order CRF is rarely used in practice because inference is difficult to implement, and the models are hard to fit with limited data. On the other hand, gradient-based SPEN inference is agnostic to the structure of the energy function: switching from (3.16) to (3.17) requires no change to the inference code and very minimal changes to the definition of the energy network. Of course, this comes at a price of losing guarantees of exact inference. As we increase the expressivity of the SPEN energy, it may become increasingly difficult to perform high-quality energy minimization in practice. Furthermore, a CRF is a proper probabilistic model that provides the opportunity to compute marginals for y, sample y, etc. using dynamic programming.

Finally, note that parameter tying provides an important source of inductive bias for linear-chain CRFs. Here, ψ_i and $\psi_{i,i+1}$ depend on a window of features $F(x)_{[i-c\,:\,i+c]}$ around index i and this functional dependence does not depend on the value of i. It is easy to inject similar inductive bias in a SPEN for sequence labeling by sharing parameters across the E_i functions by employing deep convolutional networks.

In Chap. 6 we discuss an extensions SPENs for chain-structured problems where the optimization variable explicitly represents pairwise relationships among adjacent tags.

3.3.3 Image Segmentation

Let x be a height × width × 3 input color image. We seek to predict an array of discrete pixel labels $y \in \{0, \ldots, D-1\}^{\text{height}\times\text{width}}$.

This problem can be posed easily as energy minimization in a grid-structured factor graph, where the dependence on x comes by way of deep convolutional network. These features may have broad receptive fields. However, the factor graph, i.e., the energy function over y, can only model local interactions between adjacent pixels.

Pracitioners have used more complex graphical models, but these require substantially more complex inference algorithms.

With SPENs, on the other hand, it is very easy to experiment with SPENs that capture sophisticated interactions between output labels at various lengthscales. Namely we can use a deep convolutional network for the energy function. This is a generalization of the architecture in the previous section, where the energy applies 2-dimensional convolutions to $\bar{y}$. No new algorithms need to be derived in order to accomodate these sophisticated models, as we can still interact with the energy as a black box that provides back-propagation.

3.3.4 Image Denoising

In contrast with image segmentation, image denoising can be posed as a problem with continuous output variables. For example, both x and $\bar{y}$ are height × width grayscale images. Here, the a similar approach as Section 3.3.3 can be employed. The principal difference is that we output continuous values $\bar{y}$ and do not need to perform any rounding.

It is interesting to choose a more model-based architecture, however. If we assume that the observed image x is a corrupted version of a latent image $\bar{y}$, where the noise is white with a known variance, then we can estimate $\bar{y}$ using MAP inference:

$$\bar{y}^* = \max_{\bar{y}} \log P(\bar{y}|x) = \max_{\bar{y}} \log P(\bar{y}) + \log P(x|\bar{y}) \tag{3.18}$$

Let λ be the inverse noise variance. Then, we use the SPEN to encode the MAP objective:

$$E(\bar{y}) + \lambda \|x - \bar{y}\|^2. \tag{3.19}$$

Here, $E(\bar{y})$ is a deep network that does not depend on x and returns the log prior likelihood of an image. The second term arises from the Gaussian noise model. $E(\bar{y})$ can be constructed in various ways. One advantage of using a SPEN over other approaches such as Markov Random Fields (Geman & Geman, 1984) or Fields of Experts (Roth & Black, 2005) is that we have the freedom to experiment with very sophisticated multi-resolution architectures for $E(\bar{y})$.

3.3.5 Link Prediction in a Graph

Let $x = \{x_1, \ldots, x_n\}$ be a collection of nodes with per-node feature vectors $g = \{g_1, \ldots, g_n\}$. We seek to predict edges among x. This can be formulated as an energy-based structured prediction with a matrix of binary output labels y_{ij}, where $y_{ij} = 1$ denotes that there is an edge (perhaps directed) from node i to node j.

We assume that $F(x)$ outputs an $n \times n \times f$ tensor containing an f-dimensional feature vector $F_{ij}(x)$ for every i-j pair, where F_{ij} is a function of g_i and g_j. We also assume that the functional dependence of F_{ij} on g_i and g_j is independent of i and j, such that the behavior of the model is invariant to permutation of the indices of the nodes.

Independent per-edge predictions can be made using a local energy:

$$E_x^{\text{local}}(\bar{y}) = \sum_{i=1}^{n} \sum_{j=1}^{n} w^\top F_{ij}(x). \tag{3.20}$$

Here, the parameter vector w does not depend on i or j. Again, this is to maintain permutation invariance.

To tie all of the predictions together, we define a feature vector R_i for each node i that depends on the presence (or absence) of predicted edges incident to i:

$$R_i(\bar{y}, F(x), g_i) = R\left(\text{SelectRow}(\bar{y}, i), \text{SelectRow}(F(x), i)\right). \tag{3.21}$$

SelectRow$(\cdot, i)$ slices the ith row of a matrix. Here, SelectRow$(\bar{y}, i)$ returns a vector in $[0,1]^n$ denoting the soft prediction that each edge incident to node i is on and SelectRow(F(x),i) returns an $n \times f$ matrix of features for every edge incident to i. The function R is an arbitrary deep network. This is capable of modeling non-linear interactions among the edges incident to a node. For example, we may have that two edges are mutually exclusive.

3.4 Representational Capacity

For MRFs, the interplay between the graph structure and the set of representable conditional distributions is well-understood (Koller & Friedman, 2009). However, characterizing the representational capacity of SPENs is more complex, as it depends on the general representational capacity of the deep architecture chosen.

Take, for example, our SPEN architecture for multi-label classification. In (3.11) and (3.12) , the product $C_1\bar{y}$ is a set of learned affine (linear + bias) measurements of the output. These capture salient features of the labels used to model their dependencies. By learning the *measurement matrix* C_1 from data, the practitioner imposes minimal assumptions a-priori on the interaction structure between the labels, but can model sophisticated interactions by feeding $C_1\bar{y}$ through a non-linear function. This has the additional benefit that the number of parameters to estimate grows linearly in L. This parsimony allows us to fit expressive models on limited data.

A quadratic dependence on L is unavoidable for factor graph approaches to the problem, unless we make strict assumptions about the interactions among the labels. In terms of statistical efficiency, the dependence of factor graphs on L is more complicated. A naive factor graph would have $O(L^2)$ parameters to estimate, which is prohibitive for small data and large L. On ther other hand, these issues can be circumvented using various parameter-tying schemes, such as a low-rank assumption (Srikumar & Manning, 2014; Jernite et al., 2015). While these may provide the opportunity to fit models on limited data, the expressivity of the interactions among the labels that these models can capture is fundamentally limited.

In the applications considered in this thesis, we build the SPEN on top of a high-performance feed-forward predictor implemented as a neural network. This predictor is used to pretrain our features, to initialize gradient-based energy minimization, and to provide the local energy terms for our energy network. Overall, our deep global energy terms provide a small, but complementary, contribution. It is unclear whether SPENs would fail in tasks where a high-quality local predictor is not available or where a SPEN energy that describes the data well has steep barriers that prevent gradient-based prediction from adequately exploring output space.

3.5 Speed

In general, we expect that feed-forward approaches for a given structured prediction problem will be faster than SPEN approaches. Of course, this may not be true. If the feed-forward network is extremely deep, then it will be slow. If the network for the SPEN energy is shallow and we only perform a few steps of iterative energy minimization, SPEN prediction will be fast. Often, the width of a network, i.e., the dimensionality of hidden layers, has a substantially lower impact on prediction speed than the depth, since associated operations, such as matrix multiplication, can often be parallelized across this dimension. Along these lines, one principal advantage of SPENs is that much of the computation may be parallelizable across the shape of the output (e.g., using convolutions for an image processing problem). Therefore, the scaling of prediction on a multi-core processor such as a GPU may be nearly constant. See Sec. 8.4.2.2 for experiments regarding this.

Finally, significant speed improvements can often be achieved using good low-level code. This has been instrumental to make RNNs feasible for production tasks such as machine translation (Wu et al., 2016). It is possible that SPEN speed could be improved, for example, by performing more arithmetic operations in place.

3.6 Input-Convex Neural Networks

Direct follow-on work to Belanger & McCallum (2016) appears in Amos et al. (2017), which introduces input-convex neural networks (ICNN). These are identical to SPENs, except that $E_x(\bar{y})$ is a convex function of $\bar{y}$, but not necessarily of x. All of the SPEN architectures considered in this thesis stack matrix matrix multiplications and non-decreasing coordinate-wise non-linearities. In Amos et al. (2017), the authors employ the same general energy function architecture, and achieve convexity with respect to $\bar{y}$ by constraining every learned parameter of the energy network to be positive. This sufficient condition is a consequence of the simple fact that "non-negative sums of convex functions are also convex and that the composition of a convex and convex non-decreasing function is also convex" (Boyd & Vandenberghe, 2004; Amos et al., 2017). Not every parameter needs to be positive. We only need to constrain any parameter that directly interacts with $\bar{y}$ or any of its descendents in the computation graph defined by the architecture. For example, like in our work, ICNNs express the dependence of the energy on x by way of an arbitrary feature network. This is not constrained, as computation of the features is upstream from $\bar{y}$ in the computation graph.

Convexity with respect to $\bar{y}$ is useful because global optimization of the energy function is feasible. This is useful at test time. It is also important at train time for structured SVM learning (Sec. 5.3), where prediction is used in the inner loop of learning. Amos et al. (2017) also demonstrate that *exact* inference can be performed using a linear program, if all of the non-linearities in the energy network are ReLUs. This is slow in practice, since it does not exploit the specific structure of the linear program. Instead they use a fast 'bundle entropy method' which converges very quickly in practice. We do not employ the bundle entropy method in our experiments, however, because it requires sophisticated custom code that prevents the black-box interaction with the energy function. It may be worth considering in future work, however, as it performs well in their experiments.

Many of our experiments contrast the performance of models trained with and without the ICNN constraint. While convexity of the energy is attractive, achieving convexity using the authors' particular ICNN constraint may be undesirable. For example, we find that enforcing positivity of the parameters severely hinders image denoising performance in Sec. 10.3. In these experiments, we can employ a particular energy network architecture that is always convex, regardless of the values of the parameters. That model performs well, but clamping the parameters of this architecture to be positive results in very poor performance.

If convexity can be achieved for certain values of the parameters, and convexity may be useful for achieving high-quality predictions, then why does fitting an unconstrained energy not automatically learn a convex energy whenever it would be useful? Unfortunately, this will likely not be true in practice due to the nature of the double-loop learning methods explored in this thesis. In all of them, some sort of (approximate) energy minimization is performed in order to obtain a single gradient of the loss with respect to the parameters. If this inner energy minimization is low-quality, then the learning signal used to update the parameters may be ineffective. Creating a framework that can reliably learn whether it is worth performing exact vs. approximate inference would be an interesting venue for future work.

CHAPTER 4

PREDICTION USING SPENS

This chapter presents a variety of optimization techniques and implementation-level details for energy minimization for SPENs. SPENs are flexible because (approximate) energy minimization can be performed using only a single subroutine from the energy function: evaluating $\frac{d}{d\bar{y}}E_x(\bar{y})$. SPEN prediction is 'easy' in the sense that practioners do not need to hand design specialized inference techniques for each model. We do not make any claims about the computational complexity of the overall gradient descent procedure, which may take many iterations to converge. Similarly, it is difficult to analyze the approximation error resulting from inexact minimization of a non-convex energy.

4.1 Gradient-Based Energy Minimization

Fig. 4.1 depicts a general computation graph for first-order gradient-based energy minimization. This employs the feature extraction and initialization networks introduced in the previous chapter.

A straightforward instance of Fig. 4.1 corresponds to performing T iterations of gradient descent with a per-iteration learning rate η_t:

$$\bar{y}_T = \bar{y}_0 - \sum_{t=0}^{T-1} \eta_t \frac{dE}{d\bar{y}}(\bar{y}_t) \tag{4.1}$$

Since this chapter is focused on minimization of the energy with respect to $\bar{y}$, our notation in various places removes the dependence of the energy on x and uses $E(\bar{y})$.

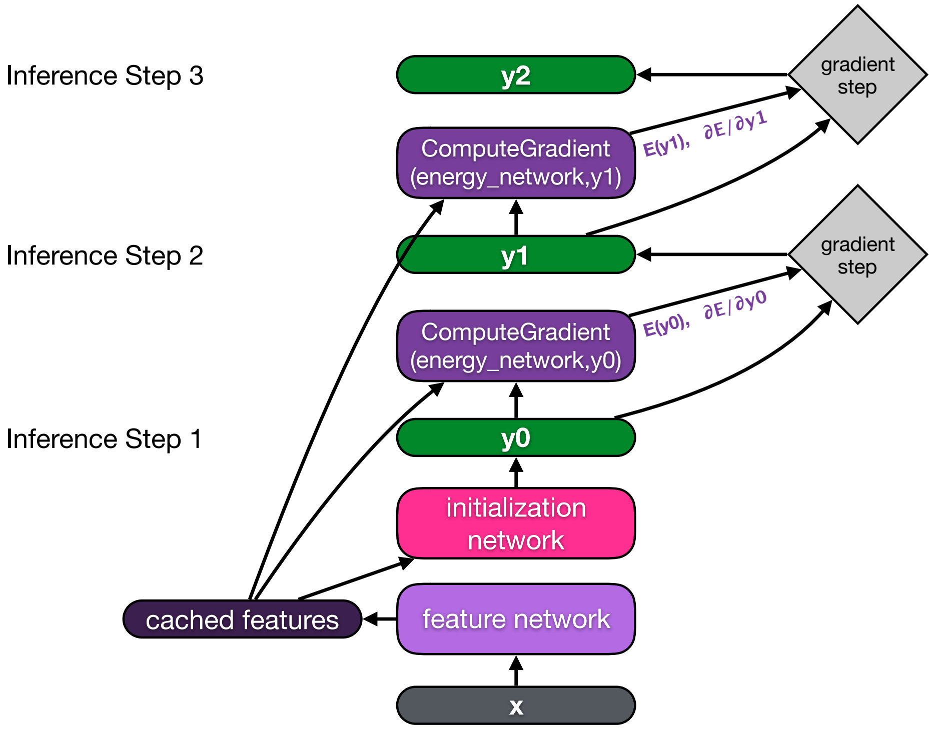


Figure 4.1: Computation Graph for Gradient-Based Prediction

Provided that we use a small enough step size and T is large enough, we will be able to reach a stationary point of the energy with respect to $\bar{y}$. In Sec. 4.3 we discuss extensions to (4.1).

The ComputeGradient module, given in Alg. 1, returns $\frac{dE}{d\bar{y}}(\bar{y})$ evaluated at the point y_t. Here, we use the explicit notation $E(\bar{y}, F(x))$ for the energy network introduced in Sec. 3.2.1. ComputeGradient only returns the derivative with respect to the first argument of the energy.

Using the ComputeGradient module requires non-standard interaction with a neural network library, as we compute derivatives in the forward pass of the network. This is defined in terms of the energy function's GradInput module, as defined at the end of Sec. 2.5.

Algorithm 1 Forward Pass of the ComputeGradient Module

```
**Input:** Differentiable Function $E(\bar{y}, f)$, Location $\{\bar{y}_0,\ f_0\}$
back_grad = 1
$\frac{dE}{d\bar{y}}, \frac{dE}{df}$ = E.GradInput($\{\bar{y}_0, f_0\}$,back_grad)
**Return:** $\frac{dE}{d\bar{y}}$
```

We can either always run T optimization iterations or we can check for a convergence criterion at every iteration and terminate if it is satisfied. An example termination criterion is:

$$\frac{\|\bar{y}_t - \bar{y}_{t-1}\|}{\|\bar{y}_t\|} < \text{tolerance}. \tag{4.2}$$

Note that gradient descent is a 'synchronous' algorithm: all coordinates of $\bar{y}$ are updated simultaneously, where the updates are computed with respect to the previous value of $\bar{y}$. For both belief propagation and mean-field inference in graphical models, synchronous updates yield worse results in practice. Instead, it is common to use a update schedule that sweeps across the graph, which can improve the speed of information flow. The analogue for gradient-based inference would be to do coordinate descent, or block-coordinate descent on the energy function. We do not consider this approach in our work, as it is not amenable to GPU-based computation. Section 4.4 discusses the computational complexity of SPEN prediction.

4.2 Discrete Prediction Problems

For SPENs defined on the convex relaxation of a discrete labeling problem, energy minimization requires optimization over the set $C_{L,D}$ defined in (3.3). Here, $\sum_j \bar{y}_{ij} = 1\ \forall j$ and each entry of $\bar{y}$ is non-negative. Once we perform this optimization, we need to convert our continuous prediction $\bar{y}$ to a discrete value y.

If the discrete structured output is subject to no more hard constraints than those defining $C_{L,D}$, we can form predictions using simple rounding, which takes the argmax along each row. We discuss handling additional constraints in Sec. 4.2.2.

Of course, there are perhaps better ways to obtain a value for y. We could, for example, perform branch-and-bound. Alternatively, we could employ a reranking approach to selecting y. Some process would produce a list of hypothesized y values. We would evaluate the SPEN energy on each of these, and take the configuration with the minimum energy. We do not investigate these extensions in this thesis, though they may be interesting venues for future work.

4.2.1 Optimization for Simplex-Constrained Objectives

The following subsections describe a variety of methods for performing a single step of a first-order optimization over $C_{L,D}$. In other words, each of the methods corresponds to a possible implementation of the gray boxes at the right of Figure 4.1. In all of these, we use the notation $g_t = \frac{dE}{d\bar{y}}(\bar{y}_t)$

See Sec. 5.4.3 for a discussion of the pros and cons of each of these methods when used in the inner loop of SPEN learning. We present experiments contrasting the approaches in Sec. 8.3.4.

4.2.1.1 Projected Gradient Descent

For $D = 2$, i.e., binary labeling problems, we can easily optimize directly over $[0, 1]^L$ using projected gradient descent:

$$\bar{y}_{t+1} = \text{Clip}_{0,1}\left(\bar{y}_t - \eta_t g_t\right), \tag{4.3}$$

where $\text{Clip}_{0,1}(x) = \max(\min(x, 1), 0)$. There are related methods for simplex-constrained projected gradient descent when $D > 2$ (Duchi et al., 2008).

4.2.1.2 Entropic Mirror Descent

We can directly optimize over $C_{L,D}$ using a version of entropic mirror descent that normalizes over each coordinate (Beck & Teboulle, 2003):

$$\bar{y}_{t+1} = \mathrm{SoftMax}\left(\bar{y}_t - \eta_t g_t\right). \tag{4.4}$$

Note that (4.4) maintains iterates that are strictly in the interior of $C_{L,D}$, i.e., no coordinate of $\bar{y}_t$ will ever be exactly 0 or exactly 1. This is useful because it allows to train using the cross entropy loss (2.18), which diverges for incorrect predictions that are exactly 0 or 1. Note that entropic mirror descent corresponds to projected gradient descent where distances are measured using the KL-divergence (Beck & Teboulle, 2003).

For binary problems ($D = 2$), we can represent elements of the probability simplex using a single number. Here, the updates have a particularly simple form:

$$\bar{y}_{t+1} = \sigma\left(\log(\bar{y}_t) - \log(1 - \bar{y}_t) - 2\eta_t g_t\right), \tag{4.5}$$

where $\sigma(\cdot)$ is the sigmoid function.

4.2.1.3 Unconstrained Gradient Descent by Reparametrization

Rather than performing constrained optimization over $C_{L,D}$, we can perform unconstrained optimization by optimizing ‘logits’ l, such that $\bar{y} = \mathrm{SoftMax}(l)$, where the softmax is taken over the second dimension. To do this, we simply add a SoftMax layer to the bottom of the energy network, so that we minimize $E_x(\mathrm{SoftMax}(l))$ with respect to l. This is analogous to the reparametrization method described in Sec. 2.7.1 for HMC sampling of simplex-constrained variables. Reparametrization simplifies optimization, since it does not require projection onto the constraint set.

4.2.2 Discrete Prediction Problems with Non-Local Constraints

For some structured prediction problems, the set of permissible outputs is more restrictive than $\{0, \dots, D-1\}^L$. For example, in dependency parsing, the predicted edges must form a tree and in Sec. 2.4 we must predict valid BIO sequences for NER.

We have two possible options to account for these constraints. First, we could modify the energy function defined over $\bar{y}$ such that it imposes very high energy to invalid configurations. We could use, for example, a log barrier function for the constraints. However, this will likely prevent gradient-based optimization from being able to traverse to high-quality values of $\bar{y}$. Alternatively, we could ignore this hard constraint, perform optimization, and then map back onto the constraint set post-hoc.

Consioder, for example, the case of dependency parsing. For a sentence of length n, there are n possible parse parents for each token (accounting for the root). We can optimize $\bar{y} \in C_{n,n}$, where all configurations are given finite energy. However, rounding from $\bar{y}$ to a valid tree is non-trivial. To account for the tree constraints, we can use some auxiliary post-processing to obtain y. For example, given $\bar{y} \in C_{n,n}$, we can run a maximum spanning tree (MST) algorithm to obtain a valid tree. Here, the MST algorithm would interpret $\bar{y}$ as a weighted adjacency matrix. A similar technique has been employed when performing minimum Bayes risk parsing (Titov & Henderson, 2006). First, marginal inference is performed to get a matrix of pairwise marginals. Next, an MST algorithm is run on the graph with edge weights given by the marginals.

Ideally, such a post-hoc projection would be largely unnecessary if we fit our model well. If the data always obeys certain constraints, and the energy function is expressive enough, then perhaps low energy $\bar{y}$ will always obey the constraints, or be very close to a vertex that does obey the constraints, such that simple rounding is sufficient. Of course this may not be true in practice.

See our semantic role labeling experiments (Sec. 9.1) for an exploration of these considerations. We perform rounding subject to various non-local constraints on outputs by solving a linear program. We find that many of these constraints were by energy minimization of a SPEN with a sophisticated global energy, and did not need to be fixed by post-processing.

4.3 Optimization Improvements

4.3.1 Momentum

In practice, many first-order optimization applications can be sped up using gradient descent with momentum (Polyak, 1964; Sutskever et al., 2013):

$$v_t = (1-\gamma)g_t + \gamma v_{t-1} \tag{4.6}$$

$$\bar{y}_t = \bar{y}_{t-1} - \eta_t v_t. \tag{4.7}$$

Here, the momentum variable v_t is similar to the momentum used in the Hamiltonian dynamics described in Sec. 2.7.

The use of v_t rather than g_t in (4.7) extends to the approaches of Section 4.2.1: we simply replace g_t in the various update formulae with v_t.

Momentum is very popular for optimizing the parameters of deep networks during training, but this presents a very different optimization problem than test-time inference in a SPEN. First, SPEN inference does not perform stochastic optimization: we compute exact gradients at every iteration. Second, we run SPEN inference for orders of magnitude fewer gradient steps than what is necessary to fit a deep network. For deep networks, it is popular to use $\gamma > 0.9$. We have found it useful to use substantially smaller values, such as 0.5.

4.3.2 Line Search

Rather than using a fixed step size η_t, we can adaptively choose the step size to guarantee that the objective decreases. Namely, we performing backtracking line search, where we pose an initial guess for η_t and then update $\eta_t = 0.5\eta_t$ until the step is satisfactory (Boyd & Vandenberghe, 2004). The most simple criterion is to check that the objective will be decreased:

$$E(\bar{y}_t - \eta_t g_t) < E(\bar{y}_t). \tag{4.8}$$

There are additional available criteria, for example the Armijo or Wolfe conditions, that depend on gradients of the energy. These may result in superior performance, and typically do not require much tuning.

Also, note that ideally our prediction procedure would not be sensitive to the scale of the energy. Namely, if we uniformly double $E(\cdot)$, then the energy minimum should be the same. However, doing so will interfere with optimization if we have a fixed step size. Using line search makes us more robust to changes in the scale of the energy function. Of course, this will not be entirely true in practice, since the line search method will have hyperparameters such as an initial step, a convergence threshold, etc. that may depend on the scale. In addition, we will encounter numerical errors if the scale is very small or very large.

4.3.3 Entropy Smoothing

When performing simplex-constrained optimization, we can smooth the objective with an entropy term:

$$\min_{\bar{y}} E_x(\bar{y}) + \lambda H(\bar{y}). \tag{4.9}$$

Here, we treat each row of $\bar{y} \in C_{L,D}$ as an independent categorical distribution, and thus $H(\bar{y})$ is the sum of the entropies for each row.

There are multiple reasons that using $\lambda > 0$ is desirable. First, for convex energy functions it provides a source of extra *strong convexity*, which improves the convergence rate of first-order methods (Bubeck, 2015). It is reasonable to expect that this would improve convergence for non-convex energies as well, since the basin of attraction around each local minimum is effectively defined by a convex function. Second, it can provide better-calibrated ‘soft’ predictions, which are useful when training with the cross entropy loss (2.18), or when we seek to threshold based on confidence values in downstream applications. Third, it is an ad-hoc way to make energy minimization behave like 'marginal inference:' in the exponential family, the objectives for MAP inference and marginal inference differ by an entropy term (Sec. 2.2).

Entropy smoothing is particularly easy when performing mirror descent, as we do not need to actually instantiate this entropy term. Instead, the update rule (4.4) can be modified to account for it analytically, using the update equation:

$$\bar{y}_{t+1} = \text{RowNormalize}\left(\frac{1}{1+\eta_t\lambda}\exp(\bar{y}_t - \eta_t g_t)\right). \tag{4.10}$$

Here, the RowNormalize function re-normalizes the rows of a matrix to sum to one. In other words, we have $\text{SoftMax}(\bar{y}) = \text{RowNormalize}\left(\exp(\bar{y})\right)$. We can derive (4.10) by modifying the steps in Beck & Teboulle (2003) that derive (4.4) in terms of KL-projected gradient descent to analytically handle the extra entropy term.

4.4 Computational Complexity

Comparing the computational complexity of gradient-based SPEN prediction to energy-based alternatives such as belief propagation (BP) in a factor graph is difficult, since both may take an unknown number of iterations to converge. On the other hand, it is natural to compare SPEN prediction to BP in terms of the computational complexity of a single iteration.

Consider the multi-label classification SPEN in Section 3.3.1. In the architectures (3.11) and (3.12), the energy first projects from $\bar{y}$ to a dense vector using the matrix C_1, where the number of rows is a hyperparameter. Therefore, the complexity of evaluating the energy function and its gradient is linear in L and D, and the user has a simple tuning parameter for choosing the expressivity of the model. When using a CRF for the problem, the user either needs to impose strict a-priori independence assumptions between labels or use a fully-connected graph with pairwise factors (Ghamrawi & McCallum, 2005; Finley & Joachims, 2008). Here, the cost of an inference algorithm such as BP is $O(L^2)$, as there are L^2 edges. Modeling higher-order interactions would result in extremely slow BP. On the other hand, the $O(L)$ SPEN can capture interactions of arbitrary arity among the labels in the deep energy applied to $C_1\bar{y}$.

Note that it is unreliable to analyze these algorithms in terms of their big-O behavior. First of all, many of the terms we consider for common structured prediction applications, such as the length of a sentence, are fairly small. Second, algorithms should be evaluated in terms of how parallelizable they are. This is especially important given the availability of GPUs, which have thousands of cores. Consider, for example, prediction using a chain-structured factor graph vs. prediction using the SPEN (3.17) with a convolutional energy network. The computational complexity of the Viterbi algorithm for the factor graph is $O(LD^2)$, where D is the cardinality of each tag, and L is the length of the sequence. The algorithm scans serially across the sequence. On the other hand, all operations for energy minimization in the SPEN can be parallelized across the length of the sentence. Therefore, even though the Viterbi algorithm only requires two passes along the sequence to do exact optimization, it might be slower than a SPEN. Of course, the Viterbi algorithm performs exact energy minimization, whereas gradient-based SPEN inference would not.

Overall, iterative SPEN prediction will almost certainly be slower than feed-forward structured prediction approaches. The only way that feed-forward approaches are slower is if they require substantially more sophisticated feature computation.

CHAPTER 5

LEARNING SPENS

Now we discuss a collection of techniques for learning the parameters of the network $E_x(\bar{y})$. Note that Chapters 11 and 12 explore additional learning methods that are based on sampling.

5.1 General Setup

All of the learning methods discussed in this chapter return a gradient $\frac{dL}{dw}$, where w is a vector of the parameters of the energy and feature networks and L is some loss function. The outer optimization over w can be done using any popular technique for stochastic optimization of deep networks. We recommend Adam (Kingma & Ba, 2015b).

It may be useful to initialize the parameters of the feature network by first training them using a simple local classification loss, ignoring any interactions between components of y. Furthermore, for problems with very limited training data, we have found that overfitting can be lessened by keeping the feature network's parameters fixed when training the energy network parameters.

5.2 Learning using the Implicit Function Theorem

The *implicit function theorem* offers a framework for directly differentiatiang the loss with respect to the energy function's parameters (Foo et al., 2008; Samuel & Tappen, 2009). See Domke (2012) for an overview.

Let w be the trained parameters of the energy function. For a given ground truth output $\bar{y}_g$ define $L(\bar{y}, \bar{y}_g)$ to be the loss associated with predicting $\bar{y}$. Let $\bar{y}_o$ be the output of energy minimization. For a given training example $(x, \bar{y}_g)$ we have:

$$\frac{dL}{dw} = -\frac{\partial^2 E_x}{\partial w \partial \bar{y}^\top} \left(\frac{\partial^2 E_x}{\partial \bar{y} \partial \bar{y}^\top} \right)^{-1} \frac{dL}{d\bar{y}} \Big|_{\bar{y}=\bar{y}_o} \tag{5.1}$$

While a naive implementation requires inverting Hessians, one can solve the product of an inverse Hessian and a vector using conjugate gradients, which can leverage the techniques discussed in Sec. 5.4.1 for approximating a Hessian-vector product.

Our experiments do not consider the method, as prior work has suggested that high performance requires exact energy minimization and many conjugate gradient iterations (Domke, 2012). On the other hand, the approach is conceptually appealing, and should be considered for future research.

5.3 Structured SVM Learning

For many energy-based structured prediction models, the practitioner is able to interact with the model in only two ways: (1) evaluate the model's energy on a given value of y, and (2) minimize the energy with respect to the y. This occurs, for example, when predicting combinatorial structures such as bipartite matchings and graph cuts. A popular technique in these settings is the structured support vector machine (SSVM) (Taskar et al., 2004; Tsochantaridis et al., 2004).

If we assume (incorrectly) that our SPEN energy minimization is not subject to optimization errors, then (1) and (2) apply to SPENs and it is straightforward to train using an SSVM loss. This ignores errors resulting from the potential non-convexity of $E_x(\bar{y})$ or the relaxation from y to $\bar{y}$ for discrete problems. However, such an assumption is a reasonable way to construct an approximate learning procedure.

Define $\Delta(\bar{y}_o, \bar{y}_g)$ to be an error function between a prediction $\bar{y}_o$ and the ground truth $\bar{y}_g$. We assume that it is non-negative and that $\Delta(\bar{y}_g, \bar{y}_g) = 0$.

Let $[\cdot]_+ = \max(0, \cdot)$. For a given training example $(x, \bar{y}_g)$, the SSVM loss is:

$$L_{\text{SSVM}} = \max_{\bar{y}} \left[\Delta(\bar{y}, \bar{y}_g) - E_x(\bar{y}) + E_x(\bar{y}_g)\right]_+ . \tag{5.2}$$

Here, the $[\cdot]_+$ function is redundant when performing exact energy minimization. We require it, however, because gradient descent only performs approximate minimization of the non-convex energy. Note that the signs in (5.2) differ from the convention employed in many other papers, since we pose prediction as minimizing $E_x(\cdot)$.

We seek to differentiate L_{SSVM} with respect to the trainable parameters w of the energy function. Due to both the max and $[\cdot]_+$ operators, the loss is not differentiable. However, we can obtain a subgradient of the loss and perform stochastic subgradient descent. Subgradients are defined only for convex functions. However, many deep learning practitioners have successfully used locally-defined subgradients for non-convex loss functions. Danskin's theorem states that the subgradient of a max of convex functions is the convex hull of the set of subgradients of all of the convex functions that achieve the maximum. See Boyd & Vandenberghe (2004) for a useful overview of subgradient descent.

For subgradient descent, it is sufficient to choose a single element of the set of subgradients of the loss and treat this as if it was a gradient of the loss. We employ:

$$\frac{dL_{\text{SSVM}}}{w} = I\left[-E_x(\bar{y}_o) + E_x(\bar{y}_g) \geq E_x(\bar{y}_o) - \Delta(\bar{y}_o, \bar{y}_g)\right] \left(\frac{d}{dw}E_x(\bar{y}_g) - \frac{d}{dw}E_x(\bar{y}_o)\right), \tag{5.3}$$

where $I[\cdot]$ is the indicator function for a predicate and $\bar{y}_o$ is the output of *loss-augmented inference*:

$$\bar{y}_o = \arg\min_{\bar{y}} \left(-\Delta(y_i, y) + E_x(y)\right). \tag{5.4}$$

If multiple values of $\bar{y}$ take on the minimum value in (5.4), we select $\bar{y}_o$ by simply sampling from among these at random.

Note that the parameter gradient is zero whenever $-E_x(\bar{y}) + E_x(\bar{y}_g) \leq E_x(\bar{y}) - \Delta(\bar{y}, \bar{y}_g)\ \forall \bar{y} \neq \bar{y}_g$, i.e., the energy of the ground truth is separated from the energy of all other possible configurations by a sufficient margin. Otherwise, the parameter gradient update (which steps in the negative gradient direction) has a simple form: it pushes up on the energy of y_o and pushes down on the energy of the ground truth.

We require that Δ is not a discrete function such as the Hamming loss, but instead a differentiable surrogate loss, such as the squared loss or cross entropy loss (2.18), that we can define on continuous $\bar{y}$. Since Δ is differentiable, it is straightforward to perform loss-augmented inference using the same gradient-based energy minimization techniques we employ for test-time prediction.

Overall, we have found SSVM training to be unreliable in situations where exact energy minimization is intractable. If loss augmented inference is performed poorly, then we may fail to discover margin violations that exist. When no margin violations are discovered, the model parameters are not updated, even if they are low quality.

On the other hand, training factor graphs using an SSVM loss is conceptually more attractive than training SPENs. In loopy graphical models, it is tractable to solve the LP relaxation of MAP inference using graph-cuts or message passing techniques, e.g., (Boykov & Kolmogorov, 2004; Globerson & Jaakkola, 2008). Using the LP relaxation instead of exact MAP inference in the inner loop of CRF SSVM learning is fairly benign, since it is guaranteed to over-generate margin violations in (5.2) (Kulesza & Pereira, 2007; Finley & Joachims, 2008).

Structured perceptron learning (SP) (Collins, 2002) is an alternative to SSVM learning that is popular among NLP practitioners. It can be obtained simply by setting Δ to zero above. This is attractive because we can perform standard energy minimization, not loss-augmented energy minimization during training. The associated loss seeks to ensure that the ground truth has the lowest energy configuration, but it does not enforce that configurations' energies are separated by a margin. We found it difficiult to apply SP training to SPENs. With SSVM learning, the energy function takes on a characteristic scale, due to the scale defined by Δ. This not true for SP. As a result, tuning gradient-based energy minimization is difficult in practice. Typically SP training is applied to models such as linear-chain factor graphs where hyperparameter-free algorithms based on dynamic programming exist for exact energy minimization.

5.4 End-to-End Learning

Next, we apply the direct risk minimization principle of Sec. 2.6 to SPENs. In other words, we train SPENs 'end-to-end.' Here, we construct the predicted value $\bar{y}_p$ as a *differentiable* function of x. We perform gradient-based learning by using a standard differentiable loss function $L(\bar{y}_p, \bar{y}_g)$, where y_g is the ground truth value for y. With this, we can do learning by back-propagating through the process of doing gradient-based prediction. Such an approach was introduced in Domke (2012). Previously, it was applied to time series imputation in Brakel et al. (2013).

Even though L has the same semantics as the Δ function used above, we use different notation in order to emphasize their different roles. In this section, L is used as a differentiable loss for penalizing discrepancies between predictions and the ground tuth. For SSVM learning, a more complicated loss function is defined by wrapping Δ in a hinge loss term that enforces soft margin constraints between all configurations in the ground truth.

For discrete prediction problems, where the SPEN is defined on a convex relaxation, our test-time procedure rounds from $\bar{y}$ to y. However, during train time our loss is imposed on a predicted soft value $\bar{y}_p$. This is analogous to many other models in machine learning, such as logistic regression, where the decision rule (ie to output a 0 or a 1) is different than the model output (a probability estimate between 0 and 1). See Sec. 2.2.4 for further discussion. Though we may train with the cross entropy loss (2.18), we do not appeal to any interpretation as a probabilistic model. In other words, we do not maximize the likelihood of our data; we are simply penalizing a convex surrogate for the 0-1 loss.

To apply the direct risk minimization principle to SPENs, we need to express SPEN inference as a differentiable computation graph. This is presented in Fig. 4.1, where the implementation of the 'gradient step' module depends on which optimization algorithm from Sec. 4.2.1 is employed. By unrolling the iterative procedure over time, we observe that prediction is very similar to a recurrent neural network (RNN). In both, the update function from one timestep to the next is shared across timesteps. For the SPEN, the energy network provides this sharing, as the same energy network is used at all steps of gradient descent.

Fig. 5.1 presents the backwards computation graph for differentiating gradient-based SPEN inference. This simply reverses the arrows from Fig. 4.1. This computation graph is a graphical representation for the dependencies between modules for the back-propagation method introduced in Sec. 2.5. It takes as input the gradient of the loss L with respect to the output of the network (here, y_2) and outputs the gradient of the loss with respect to the parameters of each of the modules in the network. Computing this requires calling back-propagation in each of the boxes in the figure with the topological order defined by the arrows. Many of these boxes are defined compositionally in terms of smaller building blocks, so back-propagation in each box may require a number of back-propagation calls in sub-modules.

Consider simple gradient descent with a fixed learning rate η. The associated gradient step module takes $\bar{y}_t$ and $g_t = \frac{dE_x(\bar{y}_t)}{d\bar{y}_t}$ as input and returns $\bar{y}_t - \eta g_t$. Given $\frac{dL}{dy_{t+1}}$, the GradInput step of back-propagation in the gradient step module returns $\frac{dL}{dy_t} = \frac{dL}{dy_{t+1}}$ and $\frac{dL}{dg_t} = -\eta \frac{dL}{dg_{t+1}}$. If we treat η as a trainable parameter, the GradParameters step returns $\frac{dL}{d\eta} = -\frac{dL}{dg_{t+1}}$. If η is not a trainable parameter, then the GradParameters method is empty.

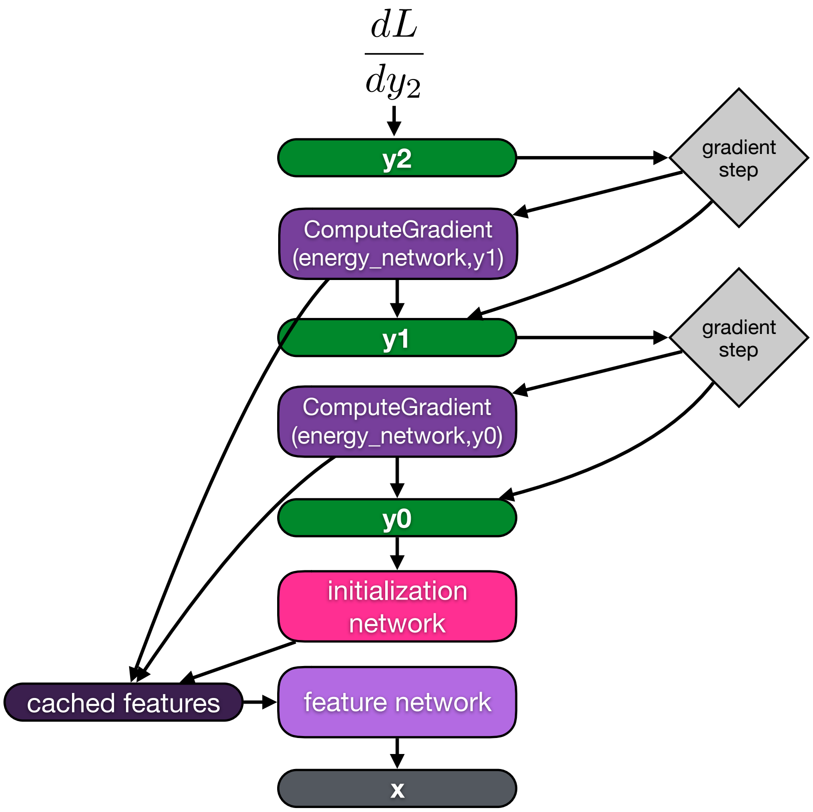


Figure 5.1: Backwards Graph for End-to-End Learning. See Sec. 5.4.1 for details on how to back-propagate through the ComputeGradient module. The other modules are built in terms of elementary operations in a neural network libarary, and thus back-propagation is straightforward.

Note that each ComputeGradient module has a pointer to the same energy network. This energy network has trainable parameters that we would like to compute the gradient of the loss with respect to. Since the same energy network appears in multiple places, it is important to maintain an accumulator that aggregates all of these contributions to the parameter gradient.

All of the energy minimization algorithms in Sec. 4.2.1 involve simple operations native to a deep learning library, and thus we can back-propagate easily through the gradient step modules using automatic differentiation. Of the optimization improvements listed in Sec. 4.3, entropy smoothing and momentum are simple differentiable operations. Sec. 5.4.1 demonstrates how to back-propagation through the ComputeGradient module module. In the backwards pass, the ‘cached features’ module adds up all of the gradients of the loss with respect to the features from the different evaluations of the energy network. This total gradient is then fed into the features network for further back-propagation.

Gradient-based optimization is differentiable with respect to the parametrization of the energy function if the energy function is twice differentiable with respect to $\bar{y}$, i.e., its partial derivative with respect to $\bar{y}$ differentiable. For this reason, we avoid using ReLU units in our energy functions and instead use the SoftPlus function, which is a smoothed version of ReLU without a kink at 0. We use a temperature of 25 for the SoftPlus, so that its shape is very similar to a ReLU. With temperature τ, this takes the form $\text{SoftPlus}_\tau(s) = \frac{1}{\tau}\log\left(1+\exp(\tau x)\right)$.

Overall, end-to-end learning provides a number of opportunities that are unavailable for SSVM learning:

1. We can train a parametrized initialization network.

2. We can directly train for the scenario where we only have the budget for a small number of gradient steps at test time (Sec. 5.4.5). This will hopefully help learn an energy surface such that gradient descent arrives in a high-quality region quickly.

3. SPEN prediction is differentiable with respect to many of the inference hyperparameters, such as the learning rate and momentum constant. Therefore, we can treat these as learned parameters that are tuned jointly with the learned energy function (for Computational Linguisticsaurin et al., 2015).

5.4.1 Differentiating the ComputeGradient Module

This section describes how to differentiate the ComputeGradient module. Here, it is useful to we write the energy as an explicit function $E(\bar{y}, f)$ of two inputs. The

GradInput and GradParameters methods for the module are given in Algorithms 2 and 3 respectively.

Let z be the output of ComputeGradient computed in the forward pass evaluated at the location $\{\bar{y}_0,\ f_0\}$ (Alg. 1). In other words, $z = \frac{dE}{d\bar{y}}$ evaluated at $f = f_0$ and $\bar{y} = \bar{y}_0$.

During back-propagation, the GradInput method for the ComputeGradient module receives the column vector $\frac{dL}{dz}$, where L is the downstream loss, and it seeks to compute $\frac{dL}{dz}^\top \frac{dz}{df_0}$ and $\frac{dL}{dz}^\top \frac{dz}{d\bar{y}_0}$. Similarly, the GradParameters method computes $\frac{dL}{dz}^\top \frac{dz}{dw}$, where w is the parameter vector for $E(f, \bar{y})$. Note that here, and the rest of the thesis, we abuse notation slightly by writing $\frac{\partial E}{\partial \bar{y}}$ as $\frac{dE}{d\bar{y}}$, even when treating E as an explicit function $E(\bar{y}, f)$ of two variables. We use the notation $\frac{dE}{d\bar{y}}$ to be consistent with other parts of the thesis focused on test-time energy minimization, where the energy is treated as a function $E_x(\bar{y})$ of a single input.

Next, we derive a method for efficiently approximating the vector-Jacobian product $\frac{dL}{dz}^\top \frac{dz}{dw}$. The other back-propagation terms can be computed similarly. We seek to compute

$$\frac{dL}{dz}^\top \frac{dz}{dw} = \frac{dL}{dz}^\top \left(\frac{d}{dw} z \right) = \frac{dL}{dz}^\top \frac{d}{dw} \frac{dE}{d\bar{y}}, \tag{5.5}$$

A naive approach would instantiate the Hessian matrix $H = \frac{dE}{dw d\bar{y}}$. Here, and going forward, we adopt the convention for Hessians that $H_{ij} = \frac{dE}{dw_j d\bar{y}_i}$. Explicitly computing this matrix would unmanageable in terms of size, and it also violates our black-box interaction with the energy, where we only assume a gradient subroutine. There are a few ways to compute Hessian-vector products without instantiating the Hessian. First, the technique of Pearlmutter (1994) provides exact computation, but it requires non-trivial changes to code for back-propagation in E. Alternatively, we can use a neural network library where the gradient is itself computed using a computation graph that supports forward and back-propagation. This is only doable when certain functions are used to define the energy. Finally, we can use the finite difference approximation employed by Domke (2012). This is easy to implement and allows us to maintain black-box interaction with the energy. Our experiments use finite differences, since the neural network library we use does not naturally support the first two options.

Next, we derive the finite-difference method of Domke (2012) in terms of an energy function that takes two arguments $\{\bar{y}, f\}$. Recall from Sec. 2.5 that back-propagation computes Jacobian-vector products. In this derivation, we leverage the fact that a Jacobian-vector product is equivalent to a directional derivative. Let h be an arbitrary differentiable function from $\mathbb{R}^m \to \mathbb{R}^n$ and let $v \in \mathbb{R}^m$ be the direction we seek to take the directional derivative in. Then,

$$\frac{dh(\bar{y}_0)}{d\bar{y}} v = \lim_{r \to 0} \frac{1}{r} \left(h(\bar{y}_0 + rv) - h(\bar{y}_0) \right). \tag{5.6}$$

Here, the Jacobian matrix $\frac{dh(\bar{y}_0)}{d\bar{y}}$ is $n \times m$. Next, we extend this expression to a function $h(\bar{y}, f)$ of two variables (and again abuse notation for partial derivatives). The first input $\bar{y}$ is m-dimensional. We do not differentiate with respect to the second argument.

$$\frac{dh(\bar{y}_0, f_0)}{d\bar{y}} v = \lim_{r \to 0} \frac{1}{r} \left(h(\bar{y}_0 + rv, f_0) - h(\bar{y}_0, f_0) \right). \tag{5.7}$$

Finally, set h to $\frac{dE}{dw}$, a function that returns a column vector, and $v = \frac{dL}{dz}$. Then, $\frac{d}{d\bar{y}} \frac{dE(\bar{y}_0, f_0)}{dw}$ is a matrix where the rows index coordinates of w and the columns index coordinates of $\bar{y}$. We have:

$$\left(\frac{d}{d\bar{y}} \frac{dE(\bar{y}_0, f_0)}{dw} \right) \frac{dL}{dz} = \lim_{r \to 0} \frac{1}{r} \left(\frac{dE(\bar{y}_0 + r\frac{dL}{dz}, f_0)}{dw} - \frac{dE(\bar{y}_0, f_0)}{dw} \right). \tag{5.8}$$

The left hand side is precisely the transpose of (5.5). The approximation's accuracy is $O(r)$, and we can make r as small as we want. However, this is subject to numerical underflow considerations. An improved $O(r^2)$ approximation can be obtained using

$$\frac{d}{d\bar{y}}\frac{dE}{dw}(\bar{y}_0, f_0)\frac{dL}{dz} = \lim_{r\to 0}\frac{1}{2r}\left(\frac{dE(\bar{y}_0 + r\frac{dL}{dz}, f_0)}{dw} - \frac{dE(\bar{y}_0 - r\frac{dL}{dz}, f_0)}{dw}\right). \tag{5.9}$$

Algorithm 2 GradInput Method for the ComputeGradient Module

```
**Input:** Function $E(\bar{y}, f)$, Inputs $\{\bar{y}_0,\ f_0\}$ to $E$, backwards gradient $b_0$, finite difference step size $\epsilon$
$l = \|b_0\|$
$b_n = (1/l)b$
$\bar{y}_+ = \bar{y}_0 + \epsilon b_n$
$\frac{dE}{df_+}, \frac{dE}{d\bar{y}_+} = E.\text{GradInput}(\{\bar{y}_+, f_0\}, b_n)$
$\bar{y}_- = \bar{y}_0 - \epsilon b_n$
$\frac{dE}{df_-}, \frac{dE}{d\bar{y}_-} = E.\text{GradInput}(\{\bar{y}_-, f_0\}, b_n)$
**Return:** $\frac{l}{2\epsilon}\left(\frac{dE}{d\bar{y}_+} - \frac{dE}{d\bar{y}_-}\right), \frac{l}{2\epsilon}\left(\frac{dE}{df_+} - \frac{dE}{df_-}\right)$
```

Algorithm 3 GradParameters Method for the ComputeGradient Module

```
**Input:** Function $E(\bar{y}, f)$, Inputs $\{\bar{y}_0,\ f_0\ \}$ to $E$, backwards gradient $b_0$, finite difference step size $\epsilon$
$l = \|b_0\|$
$b_n = (1/l)b$
$\bar{y}_+ = \bar{y}_0 + \epsilon b_n$
$\frac{dE}{dw_+}, \frac{dE}{d\bar{y}_+} = E.\text{GradParameters}(\{\bar{y}_+, f_0\}, b_n)$
$\bar{y}_- = \bar{y}_0 - \epsilon b_n$
$\frac{dE}{dw_-}, \frac{dE}{d\bar{y}_-} = E.\text{GradParameters}(\{\bar{y}_-, f_0\}, b_n)$
**Return:** $\frac{l}{2\epsilon}\left(\frac{dE}{dw_+} - \frac{dE}{dw_-}\right)$
```

5.4.2 Line Search

We have performed experiments where we employ backtracking line search, as shown in Algorithm 4, in our unrolled optimizer. Here, the step length α is a function of y_0, the direction g and the parameters of the function E. We seek to design this function to be differentiable in its arguments.

Note that the α returned by line search is a piece-wise constant function of these inputs. In other words, for infinitesimal changes in the inputs, the return value does not change. At these points, the gradient of α with respect to everything is 0. At the discontinuities, which occur where $E(y_0) = E(y + \alpha g)$, we choose to set the gradient equal to 0 as well.

This heuristic method works well in practice, but may have strange failure modes. None of our experimental results in later chapters use it. However, we think it would be worth exploring in future work. We expect there is some variant of line search that would be actually differentiable (or at least sub-differentiable).

Algorithm 4 Basic Backtracking Line Search

```
**Input:** Function $E(y)$, point $\bar{y}_0$, direction g, initial step $\alpha_0$
**Output:** Step length $\alpha$ such that $E(\bar{y}_0 + \alpha g) < E(y_0)$.
$\alpha = \alpha_0$
**while** $E(\bar{y}_0) > E(\bar{y}_0 + \alpha g)$ **do**
    $\alpha \leftarrow \frac{\alpha}{2}$
**end while**
return $\alpha$
```

5.4.3 Avoiding Vanishing Gradients

Our unrolled optimization algorithm is a deep network that is subject to the *vanishing gradient problem* (Hochreiter et al., 2001a). Here, the gradient of the loss with respect to early layers of the network is extremely weak during learning. This is a consequence of saturating non-linearities. Consider a sigmoid non-linearity. For inputs that are large in magnitude, the output of the sigmoid is nearly exactly 0 or 1, and small changes in the input yields essentially no change in the output. Therefore, gradient-based optimization will fail to update the weights for layers below a saturated sigmoid.

In Sec. 4.2.1 we discuss a variety of methods for performing energy minimization subject to simplex constraints, which arises for SPENs defined on the convex relaxation of a discrete prediction problem. In the context of end-to-end learning it is important to understand whether unrolling these optimization methods will yield a network that is vulnerable to vanishing gradients. For notational simplicity, this section considers an optimization problem defined over the simple binary probability simplex.

For Euclidean projected gradient descent (Sec. 4.2.1.1), we have:

$$\bar{y}_{t+1} = \mathrm{Clip}_{0,1}\left[\bar{y}_t - \eta_t \nabla E_{\mathbf{x}}(\bar{y}_t)\right]. \tag{5.10}$$

This will yield extreme vanishing gradients, since back-propagation through the projection will yield 0 gradients whenever $\mathbf{y}_t - \eta_t \nabla E_{\mathbf{x}}(\mathbf{y}_t) \notin [0, 1]$, i.e., whenever the gradient step takes the iterate outside of the constraint set.

For entropic mirror descent (Sec. 4.2.1.2), the updates resemble a vanilla RNN:

$$\bar{y}_{t+1} = \sigma\left(\log(\bar{y}_t) - \log(1 - \bar{y}_t) - 2\eta_t g_t\right). \tag{5.11}$$

While such models are known for suffering from vanishing gradients, practioners are often able to successfully train them in practice. Therefore, we should consider unrolled entropic mirror descent in practice.

When we reparametrize the optimization to be unconstrained (Sec. 4.2.1.3), our updates are of the form:

$$l_{t+1} = l_t - \eta_t \nabla E_{\mathbf{x}}\left(\mathrm{SoftMax}(l_t)\right). \tag{5.12}$$

Here, the update from l_t to l_{t+1} is the identity plus a correction term. Consequently, the gradient of the loss with respect to l_t will be as least as strong as the gradient of the loss with respect to l_{t+1}. Deep architectures that use additive updates, such as LSTMs (Hochreiter & Schmidhuber, 1997), highway networks (Srivastava et al., 2015) and residual networks (He et al., 2016) have proven to be very high-performance in practice, in part because they are easy to train because they suffer less from vanishing gradients.

5.4.4 Dynamically-Unrolled Inference

In Fig. 1, we unroll gradient-based inference for 3 iterations. In general, we can unroll for T iterations. Alternatively, we can run inference until gradient descent converges, i.e., when condition (4.2) is satisfied. Here, the computation graph is dynamically shaped, since the number of iterations varies across data cases. However, we can still perform back-propagation through this graph by lazily unrolling it on a per-case basis. This may be useful when performing end-to-end learning because we know that the inner optimization reaches a fixed point. On the other hand, if we give the unrolled optimizer an unlimited budget of iterations to converge, we may learn a model such that optimization is extremely slow in practice.

5.4.5 Training to Make Inference Converge Quickly

One advantage of training with a fixed number of iterations T is that we learn to accommodate a limited computational budget. On the other hand, it is hard for the user to know a-priori what an appropriate choice of T is. If T is too large, the training may produce a model where test-time prediction takes longer to converge than it needed to.

In response, it may be helpful to choose a large T and change the training objective such that prediction is explicitly encouraged to converge quickly. We can add the following term to our training objective:

$$\lambda \sum_{t=1}^{T} \|\bar{y}_{t+1} - \bar{y}_t\|. \tag{5.13}$$

When using dynamic unrolling, we terminate prediction whenever $\|y_{t+1} - y_t\|$ is less than a threshold. Thus, by encouraging this quantity to be small in (5.13), we encourage it to converge quickly.

Though some of our experiments use this method, it may be superior in future work to instead explicitly encourage the gradients to be small:

$$\lambda \sum_{t=1}^{T} \|g_t\|., \tag{5.14}$$

where $g_t = \frac{d}{d\bar{y}} E_x(\bar{y}_t)$.

An alternative way to encourage rapid optimization is to define our loss function as a sum of losses on every iterate $\bar{y}_t$, rather than only the final one. Let $L(\bar{y}_t, \bar{y}_g)$ be a differentiable loss between an iterate and the ground truth. We employ

$$L = \frac{1}{T} \sum_{t=1}^{T} w_t L(\bar{y}_t, \bar{y}_g), \tag{5.15}$$

where $w_t = \frac{1}{T-t+1}$. This encourages the model to achieve high-quality predictions early. It has the additional benefit that it reduces vanishing gradients, since a new loss term is introduced at every timestep. It is not strictly necessary to use the weights w_t. However, we have found it useful in practice.

5.4.6 Untying Energy Networks Across Iterations

Unrolling gradient descent produces a recurrent neural network. The update at each timestep is parametrized by the energy function, since each timestep's update corresponds to a step in the direction of the energy function gradient. Like most RNNs, the network for unrolled gradient descent has tied parameters across time, since the same energy function is used at each iteration of gradient descent.

One natural extension of a SPEN would be to use a different energy function at each step in the RNN. This does not correspond to gradient-based optimization, as it would be iterative optimization of a changing objective function. However, it may provide a convenient source of modeling flexibility. Untying parameters across time has been shown to improve performance for deep unrolling of belief propagation and mean-field inference (Hershey et al., 2014). In the early steps, the model may make simple local updates to $\bar{y}$ that are 'obvious' given x. Then, a different energy function is used to reconcile non-local interactions.

5.4.7 Reducing Memory Overhead

Basic implementations of end-to-end learning with unrolled gradient descent require substantially more memory than an implementation of gradient-based energy minimization to be used at test time. This is because we need to store the intermediate state of computation performed in the forward pass in order to do back-propagation for learning.

Consider the update rule $\bar{y}_t = \bar{y}_{t-1} - \eta \nabla E(\bar{y}_{t-1})$. At test time, this can be done in place, using $\bar{y} \leftarrow \bar{y} - \eta \nabla E(\bar{y})$. On the other hand, at train time we need to save each $\bar{y}_t$ separately, since these will be necessary in the backwards pass. A naive implementation would also save the intermediate state obtained inside the energy function when evaluating $\nabla E(\bar{y}_{t-1})$.

For T steps of unrolled gradient descent, the memory requirements are not just T times worse, but cT, where c is some constant that depends on the particular optimization algorithm used. For example, if we perform the simple updates $\bar{y}_t = \bar{y}_{t-1} - \eta \nabla E(\bar{y}_{t-1})$, then $c = 2$ because we need to store the values of both $\bar{y}_{t-1}$ and $\nabla E(\bar{y}_{t-1})$. Depending on details of the implementation, c will be at least 3 if we use momentum.

One method for reducing memory overhead is to use an optimization algorithm that supports 'reversible dynamics' (for Computational Linguisticsaurin et al., 2015). Here, given the optimization state at timestep $t + 1$, we can reconstruct the optimization state at timestep t. Therefore, during back-propagation we can construct the required intermediate state on-the-fly and then delete it. This is available for gradient descent with momentum (but not simple gradient descent) (for Computational Linguisticsaurin et al., 2015). It also can be applied to mean-field and belief propagation inference in MRFs (Domke, 2013a). Optimization is fundamentally an information destroying process, as many initialization points will get mapped to the same optimum. See for Computational Linguisticsaurin et al. (2015) for a discussion of how to avoid numerical underflow when reversing the dynamics.

An alternative, more generic, approach to reducing memory overhead is to reconstruct the intermediate state on-the-fly during back-propagation by performing extra forward-evaluations of parts of the network (Zweig & Padmanabhan, 2000; Lewis, 2003; Chen et al., 2016; Gruslys et al., 2016). Say, for example, that we only store $\bar{y}_t$ for $t \leq \frac{T}{2}$. Whenever we require a value of $\bar{y}_t$ at $t > \frac{T}{2}$ during back-propagation, we can recompute it by performing forward-propagation in the sub-network that defines the relationship between $\bar{y}_{\frac{T}{2}}$ and $\bar{y}_t$ for $t \geq \frac{T}{2}+1$. This trick can be applied recursively using divide-and-conquer. Overall we only require $O(\sqrt{T})$ more memory than an implementation of forward-propagation that does all computations in place and does not save any intermediate state. It introduces a factor of 2 in terms of computational cost.

In our experiments, we do not employ these tricks. Instead, we use a simple implementation detail that is good enough to allow our models to fit on large GPUs for reasonable values of T. We interact with the energy network ‘statelessly.’ Namely, we checkpoint all inputs and outputs to forward and back-propagation in the energy network, but throw away its internal state whenever we use it. The internal state is reconstructed on the fly during back-propagation. This allows us to use a single instance of the energy network, where all ComputeGradient modules have a pointer to it. A naive implementation would have required T copies of the energy network for each of the T ComputeGradient modules, where the T copies have different memory allocations for storing their internal state, but share parameters.

This final detail is the primary reason we use the Torch library (Collobert et al., 2011) rather than better-tested alternatives such as Tensorflow (Abadi et al., 2016). Torch allows for lower-level interaction with the computation graph, where it is easy to manage custom rules, such as our stateless use of the energy network, to be used in forward and back-propagation.

5.5 End-to-End vs. SSVM Learning

Overall, we have found end-to-end learning much more user-friendly than SSVM learning. Often, particularly for non-convex energies, it also results in better performance. There are many reasons why this may be true, however, and it is difficult to disentangle them. In general, it is difficult to characterize the inherent performance of a learning method when the method depends on many hyperparameters and design decisions. If the method performs well, but only for very specific values of these choices, is it good or bad?

With SSVM learning, the energy function is an independent object from the algorithm used to minimize it. This means that it is important to hand-tune the optimization method such that it works well for a given energy function. However, over the course of learning the shape and scale of the energy function may be changing dramatically. Consequently, choosing good hyperparameters for energy minimization is difficult. This makes model selection difficult, as our ability to even accurately estimate the quality of a given energy function requires a grid search over multiple hyperparameters. In contrast end-to-end learning yields both an energy function and a specific energy minimization algorithm.

It is worth noting, however, that the set of available energy minimization methods for SSVM learning is much broader than those we can unroll for end-to-end learning, since we do not require differentiability. Furthermore, the optimization algorithm's updates can be done in-place during SSVM learning, since we do not require the intermediate optimization iterates. In future work, it is possible that SSVM learning could be greatly improved by using more sophisticated optimization methods, including those that use random restarts for non-conved energies.

An additional advantage of SSVM learning is that it evaluates the energy function at the ground truth and explicitly enforces that the ground truth has low energy. In chapters 11 and 12 we explore additional learning methods that have this property. In contrast, end-to-end learning only interacts with the ground truth by way of the loss function L and the energy is only evaluated at the iterates traversed by unrolled optimization. This seems like a wasted opportunity to inject very specific supervision about how the energy function should be shaped. On the other hand, the updates that SSVM learning performs simply push its values up and down, whereas end-to-end learning shapes gradients of the energy to point in certain directions. In Sec. 12.6, we further juxtapose these updates.

End-to-end learning is useful because we can always obtain a valid gradient of the training loss with respect to the parameters. Even when the unrolled optimization performs low-quality energy minimization, back-propagation still gives an exact parameter gradient of our surrogate loss. However, for SSVM learning, the gradients may be extremely low quality approximations when inexact energy minimization is performed. In addition, for similar reasons, end-to-end learning can recover better from bad choices of the hyperparameters chosen for the unrolled optimizer since we can treat these as learnable parameters.

Finally, suppose that we had a magical subroutine that always returned the exact energy minimum. Should we train the energy with end-to-end or SSVM learning? Many of the issues with SSVM training, e.g., that it does not give a test-time prediction procedure, would not be present in this scenario. Both objectives are optimized when the ground truth is the energy minimum. However, the energy function may not be expressive enough to guarantee that the ground truth for all of the training instances is the energy minumym. In fact, to prevent overfitting we would likely want to avoid this regime. It is unclear which method would perform best given a limit-capacity energy.

CHAPTER 6
HYBRID CRF-SPENS

For SPENs defined on the convex relaxation of a discrete prediction problem, prediction is similar to fully-factored mean-field inference in an MRF, since in both we maintain a probability vector for each output variable. This chapter explores an extension of SPENs where prediction is analogous to *structured mean field* (Saul & Jordan, 1996). Here, energy function is defined over a set of variables that explictly represent pairwise probabilities of outputs.

The subject of this chapter first appeared, with equal contribution from first two authors, as Vilnis et al. (2015).

Going forward, we will call the structured prediction technique of Vilnis et al. (2015) a CRF-SPEN, for reasons that will become apparent in the next section. The key contribution of the paper is a proximal-gradient technique for performing energy minimization over the structured mean field constraint set. Specifically, we use the Bethe entropy as a distance generating function for non-Euclidean proximal gradient descent. This is useful because the associated proximal step can be computed efficiently using the sum-product algorithm for a chain-structured MRF.

Experiments applying SPEN-CRFs to citation field extraction and optical character recognition can be found in Sec. 9.3.

6.1 Background

See Sec. 2.2 for background on exponential family probability distributions. Let y be the discrete output we seek to predict. Recall that the marginal polytope of a distribution is defined as the convex hull of the set of all vectors of sufficient statistics $S(y)$ that are realizable for some y. Marginal inference in an MRF computes the expected sufficient statistics. In keeping with MRF terminology, we refer to the scope of subcomponents of y in a given factor as a *clique*.

We define our energy function over a continuous variable μ that is constrained to be in the marginal polytope $\mathcal{M}$ of a chain-structured MRF of length L. In other words, μ is the vector of expected sufficient statistics for the linear parametrization of the factor graph energy (2.25). Specifically, μ is a concatenation of a set of clique marginals $\mu_{t,t+1} \in [0,1]^{D \times D}$ that are subject to the following normalization and marginalization constraints:

$$\sum_{i,j} \mu_{t,t+1}(i,j) = 1 \; \forall t \tag{6.1}$$

$$\sum_{i} \mu_{t,t+1}(i,j) = \sum_{k} \mu_{t+1,t+2}(j,k) \; \forall t, j. \tag{6.2}$$

The first constraint ensures that the pairwise marginals are normalized. The second constraint enforces that marginals for neighboring cliques agree on their overlap. Rather than using the notation $\bar{y}$ for these, as we have in previous chapters, we employ μ, in keeping with conventions employed in the MRF inference literature. The vector of node marginals is an element of the set $C_{L,D}$, as defined in (3.3), and is related to the pairwise marginals by the identity:

$$\mu_t(i) = \sum_{j} \mu_{t,t+1}(i,j). \tag{6.3}$$

In (2.9), we establish that marginal inference can be performed by solving an optimization problem:

$$\min_{\mu \in \mathcal{M}} -\theta(x)^\top \mu - H(\mu). \tag{6.4}$$

Here, $\theta(\cdot)$ is a mapping from x to the log-potentials of a chain-structured MRF (Sec. 2.3.1.1 and Sec. 2.3.4). We use $H(\mu)$ as shorthand notation for the entropy of the joint distribution over y with marginals μ (and μ has been flattened into a vector). For our chain-structured MRF, however, this entropy can be written as an explicit function of μ:

$$H_{\mathcal{B}}(\mu) = \sum_{t=1}^{L} H_0(\mu_{t,t+1}) - \sum_{t=2}^{L-1} H_0(\mu_t), \tag{6.5}$$

where H_0 is the standard entropy of a probability vector: $H_0(p) = -\sum_i p[i] \log p[i]$. Going forward, we use the subscript $\mathcal{B}$ for our entropy to emphasize that we are treating it as an explicit function of μ and because (6.5) is an instance of the *Bethe Entropy* (Bethe, 1935; Yedidia et al., 2003).

6.2 CRF-SPENs

In the SPEN architecture 3.16 for sequence data, we use a combination of local terms and a global energy term that couples all of the labels together. In a CRF-SPEN, we also use a global energy term, but replace the local terms with the energy associated with marginal inference in a chain-structured CRF:

$$E_x(\mu) = -\theta(x)^\top \mu - H_{\mathcal{B}}(\mu) + G_x(\mu). \tag{6.6}$$

The global scoring function $G_x(\cdot)$ may depend on x arbitrarily. Going forward, our notation sometimes omits the dependence of θ and G on x. Note that if we omit the G term, this energy function corresponds to marginal inference in an MRF with log-potentials $-\theta$. We refer to this MRF as the *base model*.

As with the other SPEN models of this thesis, we assume that the only available interaction with the energy function $G(\cdot)$ is via forward and back propagation. In response, our goal is to design energy minimization methods for solving

$$\mu^* = \arg\min_{\mu \in \mathcal{M}} -\theta(x)^\top \mu - H_{\mathcal{B}}(\mu) + G(\mu) \tag{6.7}$$

by leveraging our ability to solve (6.4) efficiently and our ability to obtain gradients of $G(\mu)$.

6.3 Variational Inference Interpretation

This section provides two complementary interpretations of (6.7) as performing variational inference in certain classes of probability distributions over y. They yield precisely the same variational expression. However, one is useful because it helps motivate a principled test-time prediction procedure (Sec. 6.4.1), while the second helps characterize our proposed learning algorithm as variational EM (Sec. 6.4.2).

6.3.1 Dual Representation for μ^*

Proposition 1. *For fixed θ and G, the output μ^* of minimizing the SPEN-CRF objective* (6.7) *is equivalent to the output of standard inference* (6.4) *in an MRF with the same clique structure as our base model, but with shifted log-potentials:*

$$\tilde{\theta} = \theta - \nabla G(\mu^*). \tag{6.8}$$

Proof. Forming a Lagrangian for (6.7), the stationarity conditions with respect to the variable μ are:

$$0 = -(\theta - \nabla G(\mu^*)) - \nabla H_{\mathcal{B}}(\mu^*) + \nabla_\mu C(\mu, \lambda), \tag{6.9}$$

where $C(\mu, \lambda)$ are collected terms relating to the marginal polytope constraints. The proposition follows because (6.9) is the same as the stationarity conditions for

$$\mu^* = \underset{\mu \in \mathcal{M}}{\arg\min} - \langle \theta - \nabla G(\mu^*), \mu \rangle - H_{\mathcal{B}}(\mu). \ \square \tag{6.10}$$

Therefore, after obtaining the solution μ^* to the structured mean-field problem (6.7), we can either reason about properties of μ^* directly, or we can reason about properties of the joint distribution over y given by an MRF with parameters $\tilde{\theta}$ given by (6.8).

6.3.2 Energy Minimization as Variational Inference

Next, we characterize μ^*, the solution to (6.7), as a structured mean-field approximation to a complex joint distribution:

$$P_c(y) = (1/Z_{\theta,G}) P_\theta(y) P_G(y). \tag{6.11}$$

We assume that isolated marginal inference in $P_\theta(y)$ is tractable, as this is our base distribution. However, $P_G(y)$ is an alternative structured distribution over y for which we do not have an efficient inference algorithm. Like $P_\theta(y)$, it depends on y by way of the sufficient statistics $S(y)$. However, this dependence may be non-linear. In particular, we assume $P_G(y) \propto \exp(G(S(y)))$, where $G(\cdot)$ is a convex function. Above, $Z_{\theta,G}$ is the normalizing constant of the combined distribution. Note that if G was linear, inference in both $P_G(y)$ and $P_c(y)$ would be tractable, since the distribution would decompose over the same cliques as $P_\theta(y)$. Since (6.11) is intractable to reason about in general, we approximate it with a variational $Q(y)$.

We select $Q(y)$ by minimizing an approximation to its KL divergence to the true distribution. We have:

$$KL(Q(y)||P_c(y)) = -H(Q) - \mathbb{E}_Q[\langle \theta, S(y) \rangle] + \mathbb{E}_Q[G(S(y))] \tag{6.12}$$

$$\approx -H(Q) - \langle \theta, \mu(Q) \rangle + G(\mu(Q)). \tag{6.13}$$

Here, for the distribution $Q(y)$, we define its expected sufficient statistics as $\mu(Q) = \mathbb{E}_{Q(y)} S(y)$.

Note that (6.13) is equivalent to (6.7). Note that the surrogate we minimize is a *lower* bound to (6.12), as $\mathbb{E}_Q[G(S(y))] \geq G(\mu(Q))$, by Jensen's inequality the convexity of G, and the linearity of the term with θ. This differs from many variational inference approaches that minimize an upper bound. Minimizing a lower bound of course provides no guarantees.

So far, we have made no assumptions about the parametrization or factorization structure of the distribution Q. However, we can prove that the minimizing (6.13) yields a distribution with convenient structure.

Proposition 2. *Minimizing* (6.13) *over the set of all possible distributions* $Q(y)$*, yields an MRF with exactly the same clique structure as* P_θ *and parameters* $\tilde{\theta} = \theta - \nabla G(\mu^*)$*, as in Prop. 1.*

Proof. Let q_y denote the probability under Q of a given joint configuration y. There are exponentially many such q_y. Define $H(Q)$ as the entropy on the simplex $-\sum_y q_y \log(q_y)$. Since Q minimizes (6.13), we have the following stationarity conditions for every q_y:

$$\frac{d}{dq_y}\left[-H(Q) - \langle \theta, \mu(Q) \rangle + G(\mu(Q))\right] + \lambda = 0 \tag{6.14}$$

$$1 + \log(q_y) - \langle \theta, S(y) \rangle + \frac{d}{dq_y} G(\mu(Q)) + \lambda = 0. \tag{6.15}$$

Here, λ is a dual variable for the constraint $\sum_y q_y = 1$. For the middle term, we invoked the identity $\langle \theta, \mu(Q) \rangle = \sum_y q_y \langle \theta, S(y) \rangle$.

Rearranging and applying the chain rule for the third term, we have:

$$q_y = (1/Z) P_\theta(y) \exp\left(\left\langle -\nabla G(\mu(Q)), \frac{d}{dq_y}\mu(Q) \right\rangle\right), \tag{6.16}$$

$$= (1/Z) P_\theta(y) \exp\left(\langle -\nabla G(\mu(Q)), S(y) \rangle\right) \tag{6.17}$$

where Z is a normalizing constant obtained from the stationarity conditions for λ. At optimality, we have $\mu(Q) = \mu^*$. □

Often, structured mean-field inference is derived in terms of posing a constraint set for distributions Q, and minimizing the KL divergence between the true distribution and this constraint set. Here, we arrive at structured mean field using an alternative perspective. We approximate the KL divergence in such a way that performing unconstrained optimization over all possible distributions Q yields the same result as if we had done constrained ourselves to the set of distributions represented by an MRF with the sufficient statistics of the base distribution.

6.4 Prediction and Learning

The results of the previous section provide a means to design model-based prediction and learning algorithms. These methods assume a subroutine for solving the energy minimization problem (6.7). Optimization approaches are provided in Sec. 6.5.

6.4.1 Prediction

Proposition 1 suggests a simple model-based method for transforming from a 'soft' output μ to a discrete prediction y. First, we find the variational distribution over y parametrized as an MRF with parameter $\tilde{\theta}$. Then, we perform MAP inference in this MRF. MAP could be performed, for example, using the Viterbi algorithm.

Conveniently, our inference technique in Section 6.5 iteratively estimates $\tilde{\theta}$ on the fly, namely via the dual iterate θ_t in Alg. 6. The approach is summarized in Alg. 5.

This model-based approach contrasts with the rounding approach we use in other chapters, where we obtain y by locally maximizing nodes' marginals. For problems with rigid constraints on feasible sequences of outputs, such as for BIO tagging (Sec. 2.4), node-wise rounding would not guarantee valid predictions. On the other hand, these can be enforced when we do MAP in the base model.

Algorithm 5 Predicting discrete outputs using a SPEN-CRF.

```
**Input:** $x$
$\theta \leftarrow \theta(x)$ # Condition base distribution on $x$
$G(\cdot) \leftarrow G_x(\cdot)$ # Condition global energy function on $x$

$\mu^* \leftarrow \arg\min_{\mu \in \mathcal{M}} \theta(x)^\top \mu - H_{\mathcal{B}}(\mu) + G(\mu)$ # Energy minimization (e.g., Alg. 6)
$\tilde{\theta} \leftarrow \theta - \nabla G(\mu^*)$ # Dual view on $\mu^*$ (Prop. 1)
$y^* \leftarrow \text{MAP}\left(\tilde{\theta}\right)$ # e.g., Viterbi algorithm
**Return:** $y^*$
```

6.4.2 Learning

Above, we condition on a given value of x and employ $\theta = \theta(x)$ and $G(\cdot) = G_x(\cdot)$. Next, we assume that this conditioning comes by way of differentiable functions of x, with trainable parameters w_b and w_g, respectively, and provide methods for learning these parameters.

Consider a set of training examples (y_i, x_i). We would like to learn our parameters by maximizing the conditional likelihood of our data under the model (6.11):

$$P_c(y_i|x_i) = (1/Z_{\theta,G})P_\theta(y)P_G(y). \tag{6.18}$$

However, evaluating this likelihood is intractable. Instead we maximize a surrogate likelihood obtained from a variational approximation of (6.18).

Let μ_i be the output of energy minimization (6.7) with $\theta = \theta(x_i)$ and $G(\cdot) = G_{x_i}(\cdot)$ and define $Q(y_i; \mu_i)$ to be the distribution over y_i resulting from applying Prop.1. Namely, $Q(y_i; \mu_i)$ is an MRF with parameters

$$\tilde{\theta}_i = \theta(x_i) - \nabla_\mu G_{x_i}(\mu_i). \tag{6.19}$$

From Prop. 2, we have that $Q(y_i; \mu_i)$ is an approximation of (6.18). With this, we have the surrogate likelihood:

$$L(w_b, w_g; \mu_i) = \log Q(y_i; \mu_i). \tag{6.20}$$

We employ the notation $Q(y_i; \mu_i)$ to highlight the role of μ_i: for a given (y_i, x_i) pair, the family of variational distributions over y_i is indexed by possible values of μ_i. Our overall training objective is obtained by summing (6.20) over our training examples.

There are two principal methods for minimizing a surrogate likelihood of the form (6.20). First, we can perform a version of variational EM, where we alternate between updating μ_i for each element of our training set and updating the parameters w_b and w_g. Second, we can express each μ_i as a differentiable function of w_b and w_g and directly differentiate the right hand size of (6.20).

The second approach is doable using similar methods to what we use for end-to-end learning of SPENs, since μ_i is obtained by gradient-based energy minimization. However, our experiments instead employ the first approach because it is simple, works well, and is easy to implement.

At each iteration of learning, we sample a single x_i, y_i pair, obtain the optimal μ_i using energy minimization (E step), and update our parameters using the gradient of the surrogate likelihood (M step). In variational EM, the M step often performs full maximization. Here, this would consist of multiple gradient steps with respect to the parameters. We avoid this, however, as this will break the property that $\mu\left(Q(y;\mu_i)\right) = \mu_i$, which is necessary when obtaining gradients of the surrogate likelihood with respect to the parameters.

Since $Q(y;\mu_i)$ is an MRF with log-potentials $\tilde{\theta}_i = \theta(x_i) - \nabla_\mu G_{x_i}$, the gradient of the surrogate likelihood with respect to w_b is the standard CRF likelihood gradient:

$$\frac{d}{dw_b}L(w_b, w_g; \mu_i) = \frac{d}{d\tilde{\theta}_i}L(w_b, w_g; \mu_i)\frac{d\tilde{\theta}_i}{d\theta_i}\frac{d\theta_i}{dw_b} \tag{6.21}$$

$$= \left(S(y_i) - \mu_i\right)\frac{d\theta_i}{dw_b} \tag{6.22}$$

Similarly, for w_g, the parameters of $G_x(\cdot)$, We have:

$$\frac{d}{dw_g}L(w_b, w_g; \mu_i) = -\frac{d}{d\tilde{\theta}_i}L(w_b, w_g; \mu_i)\frac{d}{dw_g}\frac{d}{d\mu}G_{x_i}(\mu_i) \tag{6.23}$$

$$= \left(S(y_i) - \mu_i\right)\frac{d}{dw_g}\frac{d}{d\mu}G_{x_i}(\mu_i). \tag{6.24}$$

The experiments of Vilnis et al. (2015) consider architectures for G that are simple enough such that this second order derivative can be computed symbolically.

Finally, in general this learning algorithm is not guaranteed to converge, since we use alternating updates and each update does not guarantee monotonic improvements to the surrogate likelihood. In practice, however, terminating after a fixed number of iterations yields models that generalize well.

6.5 Energy Minimization

This section focuse on the case that $G_x(\mu)$ is convex in μ, in order to establish guarantees of exactness and convergence rates for gradient-based energy minimization. However, the method can be applied naturally to non-convex energies, and doing so provides performance improvements in some of our experiments.

We now present an approach to solving (6.7) using non-Euclidean projected gradient methods, which require access to a procedure for marginal inference in the base distribution (which we term the *marginal oracle*), as well as access to the gradient of the energy function G.

6.5.1 Convex Optimization Background

Before presenting our algorithms, we review several definitions from convex analysis (Rockafellar, 1997).

We call a function φ *σ-strongly convex* with respect to a norm $\|\cdot\|_P$, if for all $x, y \in \mathrm{dom}(\varphi)$,

$$\varphi(y) \geq \varphi(x) + \nabla\varphi(x)^T(y-x) + \frac{\sigma}{2}\|y-x\|_P^2.$$

Proposition 3 (e.g. Beck & Teboulle (2003))**.** *The negative entropy function $-H(x) = \sum_i x_i \log x_i$ is 1-strongly convex with respect to the 1-norm $\|\cdot\|_1$ over the interior of the probability simplex (restricting dom(H) to the interior of the simplex).*

Given a smooth and strongly convex function φ, we can also define an associated generalized (asymmetric) distance measure called the *Bregman divergence* (Bregman, 1967) generated by φ,

$$B_\varphi(x, x_0) = \varphi(x) - \varphi(x_0) - \langle \nabla\varphi(x_0), x - x_0 \rangle .$$

For example, the KL divergence is the Bregman divergence associated to the negative entropy function, and the squared Euclidean distance is its own associated divergence.

Composite minimization (Passty, 1979) is a family of techniques for minimizing functions of the form $h = f + R$, where we have an oracle that allows us to compute minimizations over R in closed form. Problems of this form are often solved using *proximal gradient* descent, which minimize $h(x)$ over some convex set X using:

$$x_{t+1} = \operatorname*{arg\,min}_{x \in X} \;\; \langle \nabla f(x_t), x \rangle + \frac{1}{2\eta_t}\|x - x_t\|_2^2 + R(x),$$

Algorithm 6 Bethe-RDA

```
Input: parameters $\theta$, energy function $G(\mu)$
set $\theta_0 = \theta$
set $\mu_0$ to prox-center MARGINAL-ORACLE($\theta_0$)
$\bar{g}_0 = 0$
repeat
    $\beta_t =$ constant $\geq 0$
    $\bar{g}_t = \frac{t-1}{t}\bar{g}_{t-1} + \frac{1}{t}\nabla L(\mu_t)$
    $\theta_t = \theta - \frac{t}{t+\beta_t}\bar{g}_t$
    $\mu_t =$ MARGINAL-ORACLE($\theta_t$)
until CONVERGED($\mu_t, \mu_{t-1}$)
```

for some decreasing sequence of learning rates η_t. Note that because of the requirement $x \in X$, proximal gradient generalizes projected gradient descent – since unconstrained minimization might take us out of the feasible region X, computing the update requires projecting onto X.

However, there is no reason to use the squared Euclidean distance when computing our updates and performing the projection. In fact, the squared term can be replaced by any Bregman divergence. This family of algorithms includes the *mirror descent* and *dual averaging* algorithms (Beck & Teboulle, 2003; Nesterov, 2009).

We base our projected inference algorithms on *regularized dual averaging* (RDA) (Xiao, 2010). The updates are:

$$x_{t+1} = \arg\min_{x \in X} \ \langle \bar{g}_t, x \rangle + \frac{\beta_t}{t}\varphi(x) + R(x), \tag{6.25}$$

where $\bar{g}_t = \frac{1}{t}\sum_k^t \nabla f(x_k)$ is the average gradient of f encountered so far. One benefit of RDA is that it does not require the use of a learning rate parameter ($\beta_t = 0$) when R is strongly convex. RDA can be interpreted as doing a projection onto X using the Bregman divergence generated by the strongly convex function $\varphi + R$.

6.5.2 Minimizing the CRF-SPEN Energy

These non-Euclidean proximal algorithms are especially helpful when we are unable to compute a projection in terms of Euclidean distance, but can do so using a different Bregman divergence. We will show that this is exactly the case for SPEN-CRFs: the marginal oracle allows us to project in terms of KL divergence. It remains to show that Bethe entropy $-H_{\mathcal{B}}$ is strongly convex, so we can use it in RDA.

Proposition 4. *For trees with n nodes, the negative Bethe entropy function $-H_{\mathcal{B}}$ is $\frac{1}{2}(2n-1)^{-2}$-strongly convex with respect to the 2-norm over the interior of the marginal polytope $\mathcal{M}$.*

Proof. Consequence of Lemma 1 in Fu & Banerjee (2013).

With these definitions in hand, we present Bethe-RDA projected inference Algorithm 6. This algorithm corresponds to instantiating (6.25) with $R = -H_{\mathcal{B}} - \langle \theta, \mu \rangle$ and $\varphi = -H_{\mathcal{B}}$. Note the simplicity of the algorithm when choosing $\beta_t = 0$. It is intuitively appealing that the algorithm amounts to no more than calling our marginal inference oracle with iteratively modified natural parameters.

Proposition 5. *For convex energy functions and convex $-H_{\mathcal{B}}$, the sequence of primal averages of Algorithm 6 converges to the optimum of the variational objective* (6.7) *with suboptimality of $O(\frac{ln(t)}{t})$ at time t.*

Proof. This follows from Theorem 3 of Xiao (2010) along with the strong convexity of $-H_{\mathcal{B}}$. □

If we have more structure in G, specifically a Lipschitz-continuous gradient, we can modify the algorithm to use Nesterov's acceleration technique and achieve a convergence of $O(\frac{1}{t^2})$. Additionally, in practice these problems need not be solved to optimality at test time, since small the MAP output will be insensitive to small differences in the predicted μ.

6.6 Discussion

Since the CRF-SPEN has terms that explicitly model joint configurations of y_t and y_{t+1}, it may be better at modeling data with very strong correlations between adjacent timesteps than a SPEN that makes a fully-factorized mean-field assumption. In problems subject to rigid constraints on outputs, such as those employed when performing text chunking as B-I-O tagging Ramshaw & Marcus (1999), our MAP inference-based approach for converting the output of energy minimization to a discrete output can also guarantee that all outputs are valid B-I-O sequences.

On the other hand, CRF-SPENs introduce computational overhead, since the forward-backward algorithm is used in the inner loop of energy minimization. This step cannot be parallelized across the length of the sequence. Overall, it is unclear if the extra computational cost and extra cost of implementation for CRF-SPENs may not be worth the expressivity they provide.

CRF-SPENs build a SPEN on top of a probabilistic model for which we have an efficient marginal inference routine, as marginal inference is required for our proximal-gradient method. Besides chain-structured graphical models, there a variety of other models where we can perform exact marginal inference in polynomial time. These include, for example, first-order arc-factored dependency parses (McDonald & Satta, 2007; Koo et al., 2007), first-order arc-factored projective dependency parses (Eisner, 1996), and determinental point processes (Kulesza & Taskar, 2011).

In future work, it may be worth considering extensions of CRF-SPENs to problems where exact marginal inference is intractable, such as for grids. A key modeling advantage of CRF-SPENs is that the energy function is defined over optimization variables that capture pairwise relationships. This is challenging, however, since these optimization variables are constrained to be consistent on variables in their overlap. One way to avoid these issues would be to maintain pairwise marginals for nonoverlapping cliques. Here, optimization would only be subject to local normalization constraints.

Optimization of a loopy CRF-SPEN with overlapping cliques would be challenging, but doable. Many approximate inference methods for loopy models, especially those based on message passing, attack the associated dual problem, where messages correspond to dual variables for consistency constraints. We would need to use a primal-dual method, based on Lagrangian relaxation, that maintains both clique-wise primal iterates and dual variables for the consistency constraints. An ad-hoc approach to this saddle-point problem would be to take alternating steps to update the primal variables and the dual variables. An additional challenge of CRF-SPENs for loopy models is that we would need a reliable MAP method for doing model-based rounding.

CHAPTER 7

RELATED WORK

7.1 Gradient-Based Prediction of Neural Network Energies

Our use of back-progation to perform gradient-based prediction differs from most deep learning applications, where it is used to update the network parameters. However, gradient-based prediction has been useful in a variety of deep learning applications, including *siamese networks* (Bromley et al., 1993), methods for generating adversarial examples (Szegedy et al., 2014; Goodfellow et al., 2015), methods for embedding documents as dense vectors (Le & Mikolov, 2014), and successful techniques for image generation and texture synthesis (Mordvintsev et al., 2015; Gatys et al., 2015a,b).

7.2 Meta-Learning and Learning to Optimize

When we estimate the parameters of SPENs, we are learning how to parametrize an optimization problem: given x, we construct an optimization problem over $\bar{y}$. Essentially, SPEN learning learns a function that predicts *what* to optimize.

End-to-end learning for gradient-based optimization first appeared in Domke (2012). The method was applied for time series imputation in Brakel et al. (2013). It was used for training generative adversarial networks in Metz et al. (2017) and for hyperparameter tuning in for Computational Linguisticsaurin et al. (2015). See Greff et al. (2017) and Gregor & LeCun (2010) for a discussion regarding the connections between network architectures and certain unrolled iterative estimation algorithms.

An alternative line of work learns a function that predicts *how* to optimize (Hochreiter et al., 2001b; Andrychowicz et al., 2016; Chen et al., 2017; Wichrowska et al., 2017). Here, each 'training example' is a collection of input-output pairs that have been split into a train set and a test set. The goal is to learn a meta model that estimates parameters of a task-specific model on the train set, such that the task-specific model performs well on the test set. The meta model is a parametrized learning algorithm. The simplest example would be gradient descent where the step size is treated as a learned parameter. These works consider more sophisticated learning algorithms, however, that model the iterative learning procedure using a long-short term memory network (Hochreiter & Schmidhuber, 1997). They demonstrate a certain degree of generalization: the meta learner can successfully learn on new datasets that have different charactersitics than the datasets that the meta learner was trained on. For example, it can learn task-specific models that have substantially more parameters than the task-specific models used when training the meta learner (Andrychowicz et al., 2016).

Note that this distinction between what to optimize and how to optimize becomes substantially less clear in Section 5.4, where we pose a method for jointly learning the parameters of the energy function and the hyperparameters used for gradient-based optimization used at test time.

A related line of work considers end-to-end learning for tasks that consist of two stages: (a) fit a model to data, and (b) use the model to perform further decision making and planning. A traditional approach would be to perform (a) using classical approaches such as maximum-likelihood learning, and then do (b). It may be useful, however, to instead directly estimate the model such that it is most helpful for the downstream use-case (b). This is possible when we have annotation for what the output of (b) should be and both (a) and (b) are differentiable (Chen et al., 2015; Tamar et al., 2016; Silver et al., 2017; Donti et al., 2017; Amos & Zico Kolter, 2017).

7.3 Neural Networks with Additive Updates

Deep architectures that use additive updates, such as LSTMs (Hochreiter & Schmidhuber, 1997), highway networks (Srivastava et al., 2015), gated recurrent units (Chung et al., 2014), and residual networks (He et al., 2016) are very popular in deep learning applications. They incrementally build up a representation additively. The decision of what to add is a complicated non-linear function, but the actual updates to the hidden state are linear. This is useful because it reduces vanishing gradients at train time. For recurrent networks it also helps the model capture long term dependencies between inputs and outputs.

Gradient descent is an additive operation: the final iterate is a weighted sum of the initial iterate and gradients of the energy. In practice, we often only unroll a few steps of gradient descent. Here, it is unlikely that the unrolled optimizer is actually doing full energy minimization. With this in mind, it may be useful to view SPENs as a residual network with a very specific parameter tying scheme.

7.4 Mean-Field Inference

Mean field variational inference approximates a complex joint distribution $P(y)$ by using a tractable distribution $Q(y)$. $Q(y)$ may be a product of univariate distributions or a structured distribution, such as a chain-structured MRF (Jordan et al., 1999). Typically, Q is selected by minimizing $KL(Q||P)$.

Consider fully-factored mean-field for a problem where each of the L components y_i is binary. Q can be represented in terms of a vector of per-component marginals $\mu_1, \ldots, \mu_L$. Here, μ_i is analogous to the $\bar{y}_i$ variable used for SPENs defined on the convex relaxation of a discrete problem.

Under the fully-factored Q distribution, we assume that components y_i and y_j are independent. Therefore, the 2×2 matrix representing their joint distribution is given by the rank-one matrix $\mu_i \mu_j^\top$. This applies to higher-order factors as well. Let y_i^o be a one-hot representation for y_i. Consider a general factor graph with energy:

$$-\sum_{c \in \mathcal{C}} \left\langle \boldsymbol{\theta}_c, \bigotimes_{i \in c} y_i^o \right\rangle . \tag{7.1}$$

Here, $\bigotimes_{i \in c} y_i^o$ represents a repeated outer (tensor) product of the one-hot representations for each of the components y_i in the factor with scope c. Then, under Q, the expected energy is

$$-\sum_{c \in \mathcal{C}} \left\langle \boldsymbol{\theta}_c, \bigotimes_{i \in c} \mu_i \right\rangle . \tag{7.2}$$

Note that if each factor was of size 1 or 2, then (7.2) would be a (not-necessarily convex) quadratic form.

The user does not directly choose to construct the mean-field energy as a multilinear form (7.2). Instead, this functional form is a consequence of decisions that he or she made about the structures of the distributions Q and P. With SPENs, we take an alternative, more pragmatic approach: the user specifically parametrizes the functional form of the energy minimization problem, instead of posing a probabilistic model for which energy minimization corresponds to variational inference. This provides the opportunity to employ a substantially broader family of energy functions.

7.5 Black-Box Variational Inference

Lately, the community has embraced 'black box' inference in probabilistic models. Here, generic tools are provided such that the user can explore a wide range of models while maintaining the ability to do inference. Furthermore, many model-specific details for how to perform inference are addressed under the hood. This allows the user to focus on model selection. Such techniques are available for variational inference in directed graphical models (Nguyen & Bonilla, 2014; Ranganath et al., 2014;

Kucukelbir et al., 2015) and also ‘deep generative models,’ where transformations between latent variables and from latent variables to observations are given by deep networks (Mnih & Gregor, 2014; Rezende & Mohamed, 2015; Salimans et al., 2015). Black-box interaction with the model is possible because of certain restrictions on acceptable models, on the functional form of variational approximations to the posterior, and by using the ‘reparametrization trick,’ where randomness in the model is decomposed into model-independent white noise and parametrized deterministic functions (Williams, 1992), or the *score function* estimator, which directly differentiates an expectation with respect to a parametrized distribution (Williams, 1992). Like SPENs, they also often interface with the model only via a gradient routine (gradient of the likelihood). HMC-based inference in the popular STAN package (Carpenter et al., 2016) relies on a similar reparametrization as our transformation in Sec. 4.2.1.3 to un-constrained optimization of logits.

7.6 Dual Decomposition

Dual decomposition is a popular method for performing MAP inference in complex factor graphs by leveraging repeated calls to MAP in tractable submodels (Komodakis et al., 2007; Rush et al., 2010; Sontag et al., 2011). The family of models solvable with dual decomposition is limited, however, because the terms that link the submodels must be expressible as linear constraints that factorize in the same way as the submodels in which MAP is tractable. Dual decomposition can also be used to derive MPLP, a convergent alternative to max-product belief propagation for solving the LP relaxation for MAP inference (Globerson & Jaakkola, 2008).

Similar techniques based on the alternating direction method of multipliers can be adapted for marginal inference, in problems where marginal inference in submodels is tractable (Ravikumar et al., 2010; Domke, 2011; Martins et al., 2011a; Fu & Banerjee, 2013).

7.7 Directly Learning Models that Iteratively Refine Outputs

Many of the above structured prediction methods start with an initial guesses for a prediction and then iteratively refine it in order to make the final prediction. For example, in mean-field inference we use a factorized probability distribution over outputs that is iteratively updated using coordinate descent on the mean field energy. In gradient-based prediction, we represent the output as a single continuous vector and update it using gradients of a differentiable energy function. In dual decomposition, we maintain primal variables (predictions), but update these by shifting our dual variables and redoing MAP. In all of these cases, the updates are derived from an iterative optimization method for an associated energy function.

An alternative line of work directly learns an iterative method, without an associated energy function. Here, for example, we have some update rule $\bar{y}_{t+1} = g_w(\bar{y}_t, x)$ with trainable parameters w. A natural way to train this is to minimize the total loss $\sum_{t=1}^{T} L(\bar{y}_t, \bar{y}^*)$, as in (5.15), where $\bar{y}^*$ is the ground truth and L is a loss function (Newell et al., 2016; Li et al., 2016; Belagiannis & Zisserman, 2016; Strubell et al., 2017). This assumes that the target for each intermediate iterate should be the ground truth. In Carreira et al. (2016), the g network is trained as a multi-variate regression task, by defining a trajectory for intermediate $\bar{y}_t$ using linear interpolation between a starting position and the ground truth. In Sec. 7.12, we interpret this method as an instance of imitation learning.

The difference between the update rule $\bar{y}_{t+1} = g_w(\bar{y}_t, x)$ above and the gradient descent update rule $\bar{y}_{t+1} = \bar{y}_t - \eta \nabla E_x(\bar{y}_t)$ is subtle, especially if g_w is implemented using a residual architecture (Sec. 7.3). In one, we learn a displacement field directly. In the other, the displacement field is defined as the gradient of an energy. It is unclear whether this distinction is important, but our suspicion is that the latter approach is more parsimonious and better able to handle constrained problems. Better understanding the relationship between these approaches is an important topic for future work.

7.8 Posterior Regularization

Posterior regularization (PR) (Ganchev et al., 2010), learning from measurements (LFM) Liang et al. (2009) , and generalized expectations (GE) (Mann & McCallum, 2010), are a family of closely-related techniques for performing unsupervised or semi-supervised learning of a conditional distribution $P_{\boldsymbol{\theta}}(y|x)$ or a generative model $P_{\boldsymbol{\theta}}(x|y)$ using expectation-maximization (EM), where the E-step for latent variables y does not come directly from inference in the model, but instead from projection onto a set of expectations obeying global regularity properties. In PR and GE, this yields a projection objective of the same general form (6.7) as we use for SPEN-CRFs. Here, the G terms come from a Lagrangian relaxation of regularity constraints, with paramters that correspond to dual variables for the constraints. Originally, PR employed linear constraints on marginals, but He et al. (2013) extend the framework to arbitrary convex differentiable functions. Similarly, in LFM such an inference problem arises because we perform posterior inference assuming that the observations $\mathbf{y}$ have been corrupted under some noise model. Tarlow & Zemel (2012) also present a method for learning with certain forms of non-local losses in a max-margin framework.

Our goals are very different than the above learning methods. We do not impose non-local terms in order to regularize our learning process or allow it to cope with minimal annotation. Instead, we use them to increase the expressivity of our model.

7.9 Graphical Models with Global Factors

Often, the complexity of inference in factor graphs scales exponentially in the graph's treewidth (Koller & Friedman, 2009). This may prevent practitioners from using global factors that couple many output variables together. One of our principal motivations for developing SPENs is to provide models where the cost of prediction scales better with the expressivity of the global interactions. See Sec. 3.4, for example, for a discussion of the differences in the dependence of prediction cost on the problem size for SPENs vs. factor graphs applied to multi-label classification.

An alternative to SPENs for using global scoring functions is to constrain the global factors to have specific combinatorial structure that can be exploited to design efficient algorithms. For example, consider a factor that enforces the constraint that only one of N variables is true (Martins et al., 2015), or that K of N variables are true (Swersky et al., 2012). In this case the messages passed by belief propagation in and out of the factor can be computed efficiently, in time that is not exponential in the number of variables. Such a technique can be applied for factors that impose bipartite matchings (Duchi et al., 2007), enforce tree structures (Smith & Eisner, 2008; Martins et al., 2015), interact large segments of pixels (Kohli et al., 2008; Russell et al., 2009), or enforce cardinalities and orderings among outputs (Tarlow et al., 2010).

In these methods, only certain global factor structures can be used. On the other hand, these factors can be handled efficiently and analytically. With SPENs, we are much more flexible in defining global terms, but we can only use generic gradient-based methods. It is possible that gradient descent will not be able to traverse a jagged energy landscape defined by complex global factors.

7.10 Deep Boltzmann Machines and Deep Belief Networks

A probabilistic interpretation of SPENs is that we fit the conditional distribution $\mathbb{P}(\bar{y}|x) \propto \exp(-E_x(\bar{y}))$. For the purposes of this section, it is sufficient to assume that that the energy network has a single hidden layer h and $\bar{y}$ is written as a one-dimensional vector. In addition, the dependence on x is unimportant, and thus we consider the un-conditional distribution

$$\mathbb{P}(\bar{y}) \propto \exp(-E(\bar{y})). \tag{7.3}$$

Deep Boltzmann machines (DBM) are instances of a family of deep un-directed graphical models (Smolensky, 1986; Freund & Haussler, 1992; Marks & Movellan, 2001; Hinton, 2002; Welling et al., 2004). A DBM consists of multiple stacked *restricted Boltzmann machines* (RBMs). Like the simple SPEN architecture described above, an RBM has a vector of hidden activations h. However, here h is a latent random variable. The joint distribution over $\bar{y}$ and h is given by:

$$\mathbb{P}(\bar{y}, h) \propto \exp(\bar{y}^\top A h). \tag{7.4}$$

This corresponds to an MRF where the associated factor graph has connections between the components of $\bar{y}$ and h and vice-versa, but no direct connections among the components of $\bar{y}$ or among h.

We have the following marginal distribution over $\bar{y}$:

$$\mathbb{P}(\bar{y}) \propto \int dh \exp(\bar{y}^\top A h). \tag{7.5}$$

Typically, reasoning directly about the marginal distribution is intractable. Instead, its properties are approximated by approximate inference in the joint distribution, often using mean-field inference or MCMC. Due to the bipartite graph structure, block Gibbs sampling is particularly convenient.

The roles of the hidden layers h in RBMs and SPENs are qualitatively different. In RBMs, h is an explicit random variable that is coupled to $\bar{y}$ by way of a joint distribution. On the other hand, in SPENs h is a deterministic function of $\bar{y}$. For an RBM, the joint (7.4) may have a significantly more parsimonious parametrization than a model that directly parametrizes the marginal distribution (7.5). In addition, by treating h as a random variable, it may be easier to capture multi-modal distributions over $\bar{y}$. For SPENs, on the other hand, evaluating the marginal probability is significantly more straightforward. In DBMs, it is procedurally easy to sample $\bar{y}$ using Gibbs sampling. However, perhaps this won't mix as well as HMC-based sampling for SPENs energies. On the other hand, HMC is often difficult to tune.

Learning both DBMs and SPENs with maximum-likelihood is difficult, due to the intractability of their distributions' partition functions. In Sec. 11.2 we discuss methods for sampling-based approximate maximum-likelihood learning. The methods are typically applied to models with discrete $\bar{y}$, but can be adapted to continuous $\bar{y}$ (Mnih & Hinton, 2005; Hinton et al., 2006a; Ngiam et al., 2011).

It is important to contrast DBMs with *deep belief networks* (DBNs), which are directed graphical models (Neal, 1992; Dayan et al., 1995). Again, we assume here that the model has a single hidden layer h, but in practice it may be much deeper. The joint distribution is given by:

$$\mathbb{P}(\bar{y}, h) = \mathbb{P}(\bar{y}|h)\mathbb{P}(h). \tag{7.6}$$

In a sigmoid belief network (Neal, 1992), for example, each component $\bar{y}_i$ is binary and has conditional distribution

$$\mathbb{P}(\bar{y}_i = 1|h) = \sigma(w_i^\top h) \tag{7.7}$$

In deeper DBNs, each hidden layer is a vector-valued random variable where each coordinate is either 0 or 1 and is sampled given the layer above it using a distribution of the form (7.6). The distribution over the top layer is either some tractable distribution, or it could be given by an RBM (Hinton et al., 2006b). Many of the first applications of variational inference in machine learning were for performing approximate learning in DBNs (Saul et al., 1996; Jaakkola et al., 1996; Bishop et al., 1998).

DBNs have been successfully used to perform unsupervised pretraining for deep feature extraction in domains such as image recognition (Lee et al., 2009) and speech recogniton (Hinton et al., 2012). DBNs do not pose an explicit energy function and computing the marginal probability of $\bar{y}$ is intractable. On the other hand, sampling is straightforward.

To achieve high performance with DBMs and DBNs, it is often important to perform layer-wise training (Hinton et al., 2006b; Ranzato et al., 2006; Torralba et al., 2008; Salakhutdinov & Hinton, 2009; Nair & Hinton, 2009). Here, the depth of the model is successfully increased, where all parameters besides the most recently introduced are held fixed.

7.11 Avoiding the Partition Function when Training Probabilistic Energy-Based Models

This section presents a variety of alternatives to MLE for estimating probabilistic energy-based models. MLE has optimal *statistical efficiency*, in the sense that its estimates are unbiased and achieve the minimum obtainable variance about the true data-generating parameters (Cramér, 1947; Rao, 1945). However, MLE is often unobtainable for structured distributions because the partition function involves an intractable sum or integral. Therefore, it is often necessary to trade off statistical efficiency with computational efficiency.

Consider $\bar{y}$ to be a length-m vector. If it is not a vector, we can reshape it such that it is. For probability distribution $\mathbb{P}(\bar{y})$, the score function is defined as $\frac{d \log \mathbb{P}(\bar{y})}{d\bar{y}}$. Let $\tilde{\mathbb{P}}(\bar{y})$ be the data distribution and let $\mathbb{P}_w(\bar{y}) \propto \exp(-E(y))$ be a learned distribution with parameters w.

Score matching (Hyvärinen, 2005) seeks to minimize:

$$J(w) = \mathbb{E}_{\tilde{\mathbb{P}}(\bar{y})} \left\| \frac{d \log \tilde{\mathbb{P}}(\bar{y})}{d\bar{y}} - \frac{d \log \mathbb{P}_w(\bar{y})}{d\bar{y}} \right\|^2. \tag{7.8}$$

This is the expected squared distance between the score functions of the data distribution and the model distribution, where the expectation is taken with respect to the data distribution. The expectation with respect to $\tilde{\mathbb{P}}(\bar{y})$ can be approximated using a sample average. Unfortunately, the objective is intractable, however, since we do not have functional form for the data distribution that allows us to compute $\frac{d \log \tilde{\mathbb{P}}(\bar{y})}{d\bar{y}}$. Subject to various technical conditions, minimizing this objective is equivalent to minimizing:

$$J(w) = \mathbb{E}_{\mathbb{P}_w(\bar{y})} \left[\left\| \frac{d \log \mathbb{P}_w(\bar{y})}{d\bar{y}} \right\|^2 + \sum_{i=1}^{m} \frac{d \log \mathbb{P}_w(\bar{y})}{d\bar{y}_i} \right]. \tag{7.9}$$

This can be optimized in practice, as it only involves the score function for the model distribution. In most applications, the gradient terms in (7.9) can be obtained in closed form. For an energy-based model with an arbitrary neural-network energy, we can differentiate (7.9) using the methods discussed in Sec. 5.4.1 for efficiently computing Hessian-vector products.

It may be useful to perform score matching between the model distribution and a smoothed version of the data distribution (Kingma & LeCun, 2010; Vincent, 2011; Swersky et al., 2011). Consider a Gaussian Parzen density estimator with bandwidth λ:

$$\tilde{\mathbb{P}}_\lambda(\bar{y}) = \int d\bar{y}' N(\bar{y}; \bar{y}', \lambda^2) \tilde{\mathbb{P}}(\bar{y}'). \tag{7.10}$$

For a finite dataset, this distribution corresponds to a Gaussian mixture, with centers at each of the datapoints.

For the case that the model distribution $\mathbb{P}_w(\bar{y})$ is given by an energy-based model with a certain 2-layer energy function, it can be shown that learning this distribution by score matching is equivalent to learning a denoising autoencoder (Vincent, 2011). Let $G(\bar{y})$ be a feed-forward network that maps $\bar{y}$ to a hidden representation and then back out to a reconstruction of the input. We penalize reconstructions using their mean-squared-error from the input. For a given input $\bar{y}$, the denoising autoencoder loss is

$$\mathbb{E}_{\epsilon \sim N(0,\lambda)} \left\| G(\bar{y} + \epsilon) - \bar{y} \right\|^2 . \tag{7.11}$$

Using the score-matching perspective, we see that G is the score function a learned distribution. In other words G is the gradient of the log-density with respect to the inputs. This provides an interesting perspective on denoising autoencoder learning: rather than learning an energy function and differentiating it, we can define the energy function implicitly, by way of a feed-forward network that directly computes the gradient of the log-density. This approach was recently applied to learning deep energy-based models for anomaly detection (Zhai et al., 2016). Finally, with this implicit definition, it is natural to sample from the density using MCMC, which is employed by *generative stochastic networks* (Bengio et al., 2014).

The technical conditions required by score matching may not apply in practice (Hyvärinen, 2005). Furthermore, the method is statistically consistent, but this does not guarantee that it will perform reliably for finite-sized small training sets. We have not experimented with score matching for training SPENs, but it may be a fruitful opportunity for future research.

Noise Contrastive Estimation (Gutmann & Hyvärinen, 2010) (NCE) provides another alternative to MLE for training probabilistic models. Let $\mathbb{P}_n(\bar{y})$ be a noise distribution that we can efficiently sample from and evaluate the density of. Consider a mixture distribution defined by sampling from the data distribution $\tilde{\mathbb{P}}(\bar{y})$ with probability $\frac{1}{K}$ and from the noise distribution with probability $\frac{K-1}{K}$. Define the binary random variable D to be equal to 1 if if $\bar{y}$ was sampled from the data distribution. For a given value of $\bar{y}$, we have the posterior probability:

$$\mathbb{P}(D=1|\bar{y}) = \frac{\tilde{\mathbb{P}}(\bar{y})}{\tilde{\mathbb{P}}(\bar{y}) + K\mathbb{P}_n(\bar{y})}. \tag{7.12}$$

Next, we replace $\tilde{\mathbb{P}}(\bar{y})$ with the learned model distribution:

$$\mathbb{P}(D=1|\bar{y}) = \frac{\mathbb{P}_w(\bar{y})}{\mathbb{P}_w(\bar{y}) + K\mathbb{P}_n(\bar{y})} = \sigma(\log \mathbb{P}_w(\bar{y}) - \log K\mathbb{P}_n(\bar{y})). \tag{7.13}$$

So far, the likelihood (7.13) is still intractable, as evaluating $\mathbb{P}_w(\bar{y})$ requires evaluating its partition function. However, it is shown in Gutmann & Hyvärinen (2010) that it is sufficient to pose $\mathbb{P}_w(\bar{y})$ as an un-normalized density, as the optimal density will in fact be normalized. This is true whenever the true data generating distribution is in the model family, i.e., there exists a w^* such that the data was generated by $\mathbb{P}_{w^*}(\bar{y})$. Therefore, we can approximate our likelihood as $\sigma(E(\bar{y}) - \log K\mathbb{P}_n(\bar{y}))$. This trick has been useful for speeding up learning for energy-based language models (Mnih & Teh, 2012; Mnih & Kavukcuoglu, 2013).

NCE is statistically consistent and asymptotically normal. It also approaches the MLE objective as $K \longrightarrow \infty$ (Gutmann & Hyvärinen, 2010). *Negative sampling* (Mikolov et al., 2013) is a simple special case of NCE that is not necessarily statistically consistent (Dyer, 2014).

Pseudolikelihood (Besag, 1975) approximates MLE of the joint distribution $\mathbb{P}_w(\bar{y})$ by instead performing MLE for a collection of univariate conditional distributions. We assume that $\bar{y}$ is a collection of components $\bar{y}_1, \ldots, \bar{y}_N$ and use $\bar{y}_{\neg i}$ to be the set of all components besides $\bar{y}_i$. For a given observed $\bar{y}$ we have the pseudolikelihood:

$$\prod_i \mathbb{P}_w(\bar{y}_i|\bar{y}_{\neg i}) \tag{7.14}$$

Each conditional distribution $\mathbb{P}_w(\bar{y}_i|\bar{y}_{\neg i})$ is assumed to have a tractable partition function, since it requires summing only over the support of the single component $\bar{y}_i$.

The factor graph structure defines a set of conditional independence relationships among the components. Therefore, in factor graphs with small factors, $\mathbb{P}_w(\bar{y}_i|\bar{y}_{\neg i})$ can be evaluated using a small subset of $\bar{y}_{\neg i}$. For SPENs, however, we do not assume any such factorization structure, and thus evaluating (7.14) would require many complete forward evaluations of $E(y)$.

Finally, suppose that the partition function was guaranteed to be 1, and thus $\mathbb{P}_w(\bar{y}) = \exp(-E(y))$. Maximizing the likelihood of this distribution is straightforward. An interesting line of work poses a probabilistic model $\mathbb{P}_w(\bar{y}) = \frac{1}{Z}\exp(-E(y))$ and interleaves two types of gradient updates: updates that ignore the partition function and seek to maximize the likelihood of $\mathbb{P}_w(\bar{y}) = \exp(-E(y))$, and updates that penalize $(Z-1)^2$ (Devlin et al., 2014; Andreas et al., 2015). Learning can be accelerated by performing the second set of updates less frequently than the first.

7.12 Value-Based Reinforcement Learning

At various points in this thesis, we assign different semantics to the energy function $E_x(\bar{y})$. For example, it can be the specification of a probabilistic model $\mathbb{P}(y|x) \propto \exp(-E_x(y))$ or the specification of a decision rule $\bar{y}^* = \arg\min_y E_x(y)$. These lead to different learning algorithms. Next, we discusses how the energy function could also be interpreted as a *value function* in reinforcement learning (RL). This provides an interesting direction for future work.

RL is broad family of methods for learning policies for *sequential decision making*, where an agent performs a sequence of actions. The agent collects rewards as it interacts with an environment that is potentially stochastic and only partially observable. A full exposition on the fundamentals of RL is beyond the scope of this thesis. We recommend Sutton & Barto (1998) for an approachable introduction.

Let $\pi(a|s)$ be our policy, which is a distribution of actions at a given state. A value function $V_\pi(s)$ computes the expected future reward obtained by starting in state s and taking actions according to π. The state-action value function $Q_\pi(s,a)$ computes the expected future reward conditional on taking action a in state s, and then following π. In many cases, the optimal policy π^* is deterministic, and thus has the property that $\pi^*(s) = \arg\max_a Q_{\pi^*}(s,a)$. Often, learning is based on some generalization of policy iteration, where we alternate between updating Q and π.

The policy π can either be defined implicitly, by way of greedy maximization of the Q function at a given state, or it can be defined directly as a parametrized distribution $\pi_\theta(a|s)$ with learned parameters. For the second approach, we can learn θ to maximize the expected future reward using actor-critic methods (Barto et al., 1983; Konda & Borkar, 1999). Here, the Q function, which estimates expected future reward, is used as an auxiliary function (the critic) simply to guide the policy (the actor) towards choosing high-quality actions When expected future reward is not estimated by a critic, but instead by a Monte-Carlo approximation obtained by executing the policy, we obtain methods such as REINFORCE (Williams, 1992).

In future work, it may be useful to treat iterative optimization of the SPEN energy as a sequential decision making problem. Here, the energy function would obey the semantics of a negative value function, i.e., that it estimates expected future cost. With this, it would be natural to define our policy as taking a single gradient step on the value function. It would be unnecessary to estimate a state-action function $Q(s,a)$, since we can locally maximize expected future reward using a single gradient step.

One of the core considerations of RL is balancing exploration and exploitation. Using RL methods would provide a easier methods for incorporating exploration than end-to-end learning. Using RL to train SPENs would be similar to how discrete factor graphs are trained using RL in Rohanimanesh et al. (2009).

Steps in this direction were recently taken by Gygli et al. (2017). They identify the parallel between SPENs and value functions, but they do not estimate the energy using RL methods. Instead, they estimate the energy using a regression criterion. For a given training instance x_i, $\bar{y}_i$ let $\Delta(\bar{y}, \bar{y}_i)$ be the discrepancy between an arbitrary prediction $\bar{y}$ and the ground truth. Let $Q(\bar{y})$ be an arbitrary distribution. They seek to minimize $\mathbb{E}_Q[(E_{x_i}(\bar{y}) - \Delta(\bar{y}, \bar{y}_i))^2]$. This is useful because $\Delta(\bar{y}, \bar{y}_i)$ is minimized at $\bar{y} = \bar{y}_i$.

Finally, note that RL is not the only technique to solve such problems. For example, when we have examples available for high-quality sequences of actions, we can perform imitation learning (Schaal, 1999; Abbeel & Ng, 2004; Ratliff et al., 2006; Silver et al., 2008; Argall et al., 2009; Chernova & Veloso, 2009; Ross & Bagnell, 2010). The method of Carreira et al. (2016), described at the end of Sec. 7.1 can be seen as an instance of imitation learning. Here, the example sequences are not explicitly annotated, but are generated synthetically by linearly interpolating between the initial guess $\bar{y}_0$ and the ground truth.

CHAPTER 8

EMPIRICAL EXPLORATION OF SPEN PROPERTIES

This chapter presents experiments designed to provide some intuition for the general properties of SPENs, which will hopefully help future practitioners when developing new SPEN applications. Our experiments are divided into four categories:

1. (Sec. 8.3) Experiments used for analyzing the impact of certain SPEN design decisions for end-to-end learning. These helped us select configurations that were used in other chapters in this thesis.

2. (Section 8.4) Experiments that contrast SPENs with CRFs and feed-forward predictors in terms of speed and accuracy.

3. (Section 8.5) Experiments that analyze the performance and reliability of SSVM vs. end-to-end training for SPENs.

4. (Section 8.6) Experiments demonstrating that SPENs could be used to provide interpretable structured learning.

All of the experiments consider discrete labeling problems, where the SPEN is defined on a convex relaxation of the original problem. Since this approach reduces discrete prediction to continuous optimization, the analysis of this chapter would also apply to problems with continuous outputs.

8.1 Data

First, we describe the data used for the experiments in this chapter.

8.1.1 Sequences

We consider a sequence of observations $x = \{x_1, \ldots, x_L\}$, where each x_i is drawn from a 6-dimensional spherical Gaussian with unit variance. To generate a sequence of observations $y = \{y_1, \ldots, y_L\}$, we first construct a linear-chain CRF model, which yields a joint distribution over y, given x. We can efficiently draw exact samples from this distribution by forward filtering and backward sampling in the model.

With our model, we form predictions by *minimum Bayes risk decoding* (MBR decoding), with respect to the Hamming error Goel & Byrne (2000). This is a decision-theoretically optimal method for forming predictions if our risk function is the Hamming error (Sec. 2.2.5). MBR decoding first peforms marginal inference in the model using the forward-backward algorithm, and then assigns each variable to the value that maximizes its marginals. For a given set of model parameters and test data drawn from the model with those parameters, we define the *Bayes accuracy* of the data as the Hamming accuracy of MBR decoding. This acts as a useful performance upper bound, and we generally evaluate other methods in terms of their *relative Bayes accuracy*, the relative difference between the Bayes accuracy and their accuracy. Specifically, if the Bayes accuracy is a and the alternative method's accuracy is b, we compute $1 - \frac{a-b}{a}$.

To stress test SPENs, we intentionally design the parameters of our CRFs such that SPENs will struggle to fit the data. We hand-tune various hyperparameters, such as the relative scales of the local and pairwise factors, such that our random CRF models have two key properties:

- First, we want a local classifier to have low relative Bayes accuracy when fit to data drawn from the model. Otherwise, we may obtain good performance simply by using an expressive local energy, rather than exploiting a global energy network. This is achieved in part by generating models with large-magnitude pairwise factors relative to the local factors. We further restrict the power of our local factors, both in the CRFs used to generate the data and any models we fit to the data by using a convolution width of 1 in our feature networks. Across our 20 random problems, we find that the relative Bayes accuracy of a local classifier is 65%.

- Second, we want approximate mean-field marginal inference in the CRF to yield a low-quality approximation of the exact marginals. Since SPEN inference resembles mean-field inference, data drawn from a model with this property may be hard to fit with a SPEN. Across our 20 problems, we find that MBR prediction with mean-field marginals differs on 38% of the predicted labels vs. MBR prediction with exact marginals.

The CRF extracts features from x using a width-1 convolution that is passed through a ReLU. While these features could of course be more expressive, using a width of 1 helps achieve the first desideratum above. The weights of the local factors and the pairwise factors of the model are a linear function of these localized per-timestep features. For both of these linear functions, we tie parameters across the length of the sequence. Each y_i has a domain size of 8 and the sequences are of length 12. We create 20 distinct datasets consisting of a train set of 5K sequences and a test set of 10K sequences.

In Sec. 8.4.2.1, we directly contrast the performance of SPENs and CRFs on this data. Since the data is drawn from a CRF and the factors were designed such that mean-field inference may perform poorly, this presents a very challenging baseline. However, it provides useful controlled experiments where we can isolate the effects of various design decisions, such as the energy minimization method used vs. the parametrization of the energy function.

8.1.2 Grids

A classic application of grid-shaped structured prediction problems is when we have an input RGB image $x \in [0, 1]^{h \times w \times 3}$ and seek to assign binary per-pixel labels $y \in 0, 1^{h \times w}$. We perform image segmentation on the Weizmann horses dataset (Borenstein & Ullman, 2008). This isn't synthetic data, but it is considered a toy dataset by the vision community. On the other hand, it has been used as a benchmark problem in a large number of papers on graphical models. Our results are not directly comparable to previously-published results on the dataset. First, the raw images are not all the same size. In order to support GPU-based computation, we downsample them to all be 64 pixels high and 80 pixels wide. Furthermore, the original dataset does not provide a proper validation set for tuning hyperparameters. We use 150 images for training and 88 each for validation and testing. Sec. 8.4.3 provides empirical results demonstrating that our results using CRFs on the modified dataset are very similar to published results on the original dataset.

8.2 SPEN Architectures and Training

8.2.1 Sequences

Our feature network is identical to that of the CRF introduced in Sec. 8.1.1, and our local energy terms are identical to the local factors of the CRF.

Our global energy network concatenates the 25 per-timestep features from the feature network and concatenates these with the per-timestep soft predictions $\bar{y}$. We then apply a width-k temporal convolution that maps to h hidden features, feed the output through a SoftPlus at temperature 25, and then produce a per-timestep energy using a local linear transformation for each timestep. The total global energy is the sum of these terms. We can vary k, the kernel width of the first convolution. With $k = 2$, the expressivity of the model is similar to a linear-chain CRF. However, it is easy to consider much higher-order interactions simply by increasing k.

8.2.2 Grids

Our feature network consists of a 7×7 convolution that maps from 3 input channels to 25 hidden features, a ReLU, a 1×1 convolution that maps from 25 hidden features to 25 new hidden features, and then finally a ReLU. Here, a 1×1 convolution applies a linear transformation independently at grid location. Overall, this archtecture produces a 25-dimensional feature vector for each grid location. We first train this network to optimize the performance of a feed-forward predictor, which applies a linear classifier to these features. The feature network for all other configurations are initialized using these parameters.

The global energy network of our SPEN first extracts h $k \times k$ convolutional features of the output $\bar{y}$. It then feeds these through a SoftPlus layer, and then concatenates them with the per-pixel features of x computed by the feature network. This per-pixel representation is then mapped to a per-pixel energy using a linear transformation. The final image-level energy is defined as the sum of the per-pixel energies. The SoftPlus is at temperature 25. For most experiments, we use $h = 25$.

Note that if $k = 3$, the receptive field of the SPEN energy is similar to a grid-structured CRF with pairwise factors: the energy function only explicitly captures the interactions between a pixel and its immediate neighbors. They have different expressivity, however, as the SPEN has one term that jointly scores 9 pixels (the center pixel and its eight neighbors), whereas the CRF captures their interactions using the sum of 4 separate edge factors. Here, the CRF also fails to explicitly capture interactions between pixels that are diagonal neighbors. Also, while the SPEN easily supports using larger k, inference in a CRF with high-order factors is very cumbersome.

8.2.3 Training

All models are trained using the Adam optimizer (Kingma & Ba, 2015a) with a learning rate of 0.001. For instances where gradients are particularly noisy, we set Adam's ϵ parameter to 1e-6. Otherwise, we use default settings. The weights for all convolutions are initialized using the method of Glorot & Bengio (2010). We typically clip our gradients to have norm no greater than 1.0.

We have found that the easiest way to avoid overfitting is to perform early stopping, where we select a snapshot from training that performed best on held-out data. An alternative approach would be to tune the expressivity of the feature and energy networks, by varying the dimensionality of their hidden states. This introduces two new hyperparameters to perform a grid search over, whereas early stopping does not introduce any new hyperparameters. When performing end-to-end learning, we always minimize the average cross entropy loss of the soft predictions output by gradient-based energy minimization.

8.3 Experiments for Engineering End-to-End SPENs

8.3.1 The Unreasonable Effectiveness of End-to-End Learning

As a warmup, we first demonstrate a simple case where end-to-end learning is a very effective way to discriminatively train the parameters of an unrolled prediction procedure. This experiment trains a CRF, not a SPEN. This is because it is easy to generate data from the conditional distribution defined by a CRF, but not a SPEN. We expect the general conclusions from this experiment apply to SPENs as well.

For each of the 20 models used to generate our sequence tagging data, we contrast the relative Bayes accuracy of two different prediction approaches. The **True-MF** approach performs approximate MBR decoding in the true CRF model, where approximate marginals are obtained using 15 iterations of mean-field inference. The **E2E-MF** approach unrolls the same mean-field inference method used before, but this time it is applied to a new CRF with parameters that we train from scratch. Here, the parameters are directly trained (on the training data) to minimize the cross entropy loss of the approximate marginals output by mean-field. Details of the mean-field method and how to learn it end-to-end are provided in Sec. 8.4.1.

The results of our experiment are summarized in Table. 8.1. Remarkably, substantially better performance can be obtained using **E2E-MF**, which trains a model from scratch, despite the fact that the **True-MF** utilizes the true parameters that were used to generate the data.

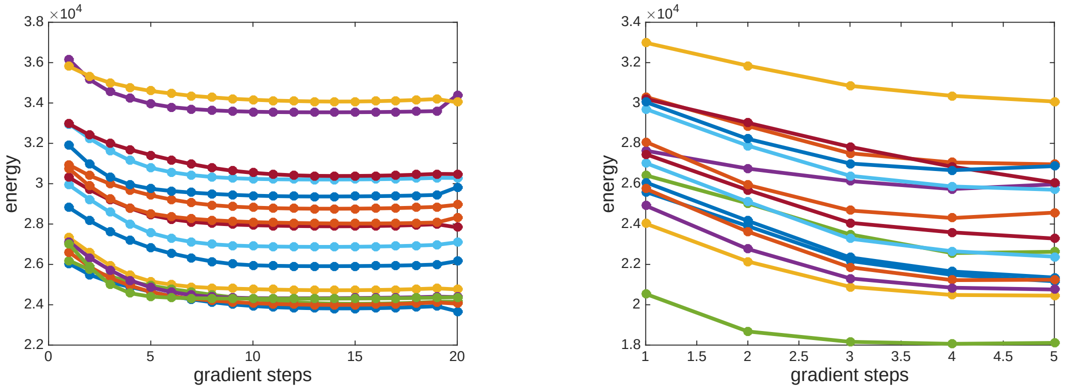
	True-MF	**E2E-MF**
relative Bayes accuracy	79.1 ± 7.3	91.6 ± 2.6

Table 8.1: Contrasting the performance of approximate MBR prediction using mean-field inference with the parameters of the true CRF used to generate the data vs. a new CRF that is trained end-to-end to optimize the performance of the same inference procedure. Surprisingly, we find that substantially better performance can be achieved using the second method.

8.3.2 Gradient-Based Energy Minimization

In Fig. 8.1, we plot the objective functions for test time SPEN energy minimization for a few different examples in the horses test set. We append the energy of the ground truth as the rightmost point in each curve.

For curves where the curve jumps up at the last step, this corresponds to the case where the predicted energy is lower than the ground truth energy. This is a modeling error. For curves that turn down at the end, this corresponds to the case that the predicted energy was higher than the ground truth energy. This is a test-time optimization error. In (a), we unroll for 18 gradient steps. In (b), we only employ 3 gradient steps. Truncating (a) to only 3 steps would result in poor performance. However, (b) is able to achieve good performance because the model is explicitly trained such that 3 steps of gradient descent will perform well.



(a) 18 steps

(b) 3 steps

Figure 8.1: Test-time energy minimization for test examples in the horses dataset. The optimization trajectory is appended by a final point containing the value of the energy at the ground truth configuration.

Next, in Figure 8.2 we contrast how peaked a SPEN's predictions are. Since the SPEN is not a proper probabilistic model, it may over-estimate its confidence, yielding soft predictions $\bar{y}$ that are nearly 0-1. On the horses data we compute the 'peakedness' of the model's soft prediction. This is 1.0 minus the average distance from each $\bar{y}_i$ to the boundary of $[0, 1]$. Peakedness of 1.0 means that the model always predicts either 0 or 1. The peakedness would be 0.75 if the predictions were distributed uniformly. We contrast the peakedness of the SPEN with the peakedness of BP marginal inference in a CRF model trained on the data. Note that the SPEN was trained end-to-end and includes an entropy smoothing term that rewards high-entropy outputs. Overall, we are surprised to find that they have extremely similar peakedness. Perhaps the difference in the approaches and energy functions and unrolled optimization is less important than the loss used to train the model, which was the same for both.

	SPEN	**CRF w/ BP**
Peakedness %	97.1	97.2

Table 8.2: Contrasting how peaked the predictions are with gradient-based prediction from a SPEN vs. BP inference in a CRF on the horses data.

8.3.3 Learning to Optimize Quickly

Next, we present experiments evaluating the methods introduced in Sec. 5.4.5 for explicitly regularizing the model such that gradient-based prediction converges quickly. We can either employ the term (5.13) that penalizes the distance between consecutive iterates or we can use the loss function (5.15) that computes a weighted average of the loss at all of the intermediate iterates of optimzation.

These methods are applied to the horses data in Fig. 8.2. On the left, we plot test-time energy minimization curves for our baseline system, which does not use (5.13) or (5.15). In the middle, we use (5.13), but not (5.15). The energy is flat until the very end, where it is minimized quickly. This exposes a weakness of the regularizer (5.13): it encourages the iterates to move quickly from their initialization point to the final prediction, but it doesn't actually encourage this to happen at the beginning of optimization. On the right, we combine both (5.13) and (5.15). This yields the desired optimization behavior. Note that the difference in test accuracy between these systems is small, but non-trivial. From left to right, we have 92.4, 92.2, and 91.1. This is because we placed an a large weight on the regularizer (5.13) in order to magnify the effect of these methods.

Overall, we do not encourage practitioners to use the methods discussed in this section. A significantly easier way to achieve fast, high-quality test time optimization is to simply choose a smaller value of T. There is also a confounding factor when using larger T: these models are harder to train end-to-end. This phenomenon is explored in the next section.

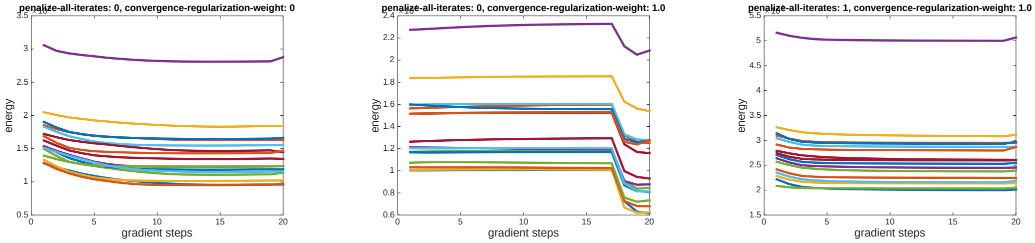


Figure 8.2: Exploring methods for explicitly training such that gradient-based energy minimization converges quickly. Left: baseline end-to-end training on the horses data. Middle: penalize the distance between consecutive iterates using (5.13). Right: penalize consecutive iterates and minimize the average loss of all optimization iterates, rather than the final iterate, using the technique in (5.15).

8.3.4 Depth of the Unrolled Network, Vanishing Gradients, Exploding Gradients, and Gradient Clipping

Next, we empirically validate our claim from Sec. 5.4.3 that unrolling mirror descent for simplex-constrained optimization produces vanishing gradients, while unrolling unconstrained optimization of logits does not. See Fig. 8.3 and its caption for an explanation.

Define g_t as $\nabla E_x(\bar{y}_t)$. Since we use a single energy network with tied parameters for all t, the gradient of the loss with respect to the parameters of the energy network will be a weighted sum of gradients evaluated at each t, with weights given by $\|\frac{d\text{Loss}}{dg_t}\|$. This means that our parameter gradients will be influenced substantially more by later iterations of unrolled optimizations that earlier optimization.

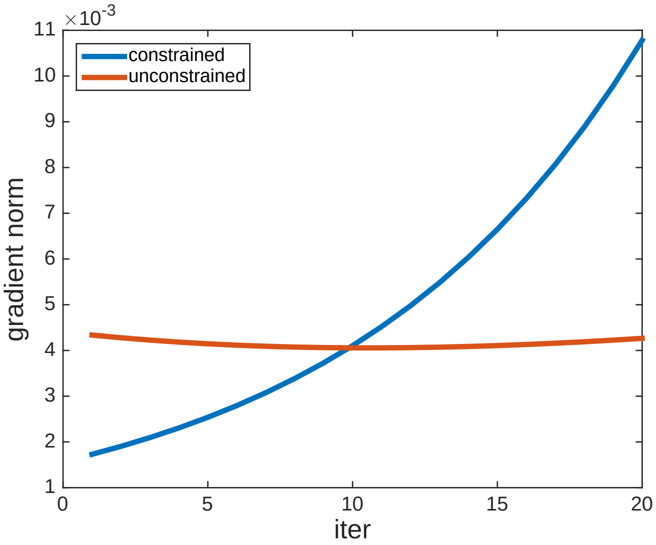
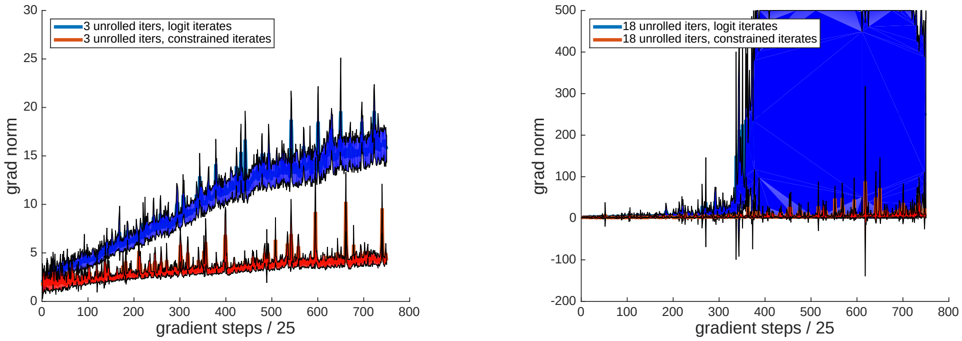


Figure 8.3: Demonstrating that unrolling entropic mirror descent yields severe vanishing gradients. On the chain data, we train SPENs with 20 unrolled iterations of either simplex-constrained optimization or unconstrained optimization of logits. Let $\bar{y}_t$ be the intermediate iterate at step t and g_t be the gradient of the energy with respect to y_t. We compute the average norm of $\frac{d\text{Loss}}{dg_t}$, and average across the iterations of training. We find that mirror descent yields yields gradients that decay significantly faster. In other words, the gradient of the loss with respect to g_t is significantly higher for large t. This may prevent us from learning a high-quality energy function, since it fails to account for the impact of the energy on the early steps of energy minimization.

In other applications of RNNs, such as for training language models, the vanishing gradient problem is detrimental because it prevents a model from learning long-term dependencies. However, here the uneven weighting between early and later iterations may actually be desirable. Note that the early iterations of unrolled optimization simply need to point in the general direction of the ground truth, whereas later iterations may be used to resolve more fine-grained details of the structured output. This may explain why we have found that high-quality models can be trained by unrolling either optimization method.



(a) 3 iterations

(b) 18 iterations

Figure 8.4: Analyzing the effect of unrolling unconstrained gradient descent on logits vs. simplex-constrained optimization using mirror descent. In (a) and (b) we plot a sliding window average of the norm of the gradient with respect to the parameters over the course of training a model. Error bars (in blue and red) are for a single standard deviation. In some cases, they are extremely large, denoting gradients with very high variance. Fig. (a) unrolls for 3 iterations, while (b) unrolls for 18 iterations. In (a), unconstrained optimization has larger gradients, but the noisiness of the gradients is on the same scale as for constrained optimizaton. In (b), the unconstrained optimization has gradient norms that are both large on average and very high variance. I

Fig. 8.3 establishes that we may encounter vanishing gradients. However, exploding gradients, may pose a larger threat to training. In Fig. 8.4 and Fig. 8.5, we find that unrolling for a large number of iterations results in more violatile gradients,

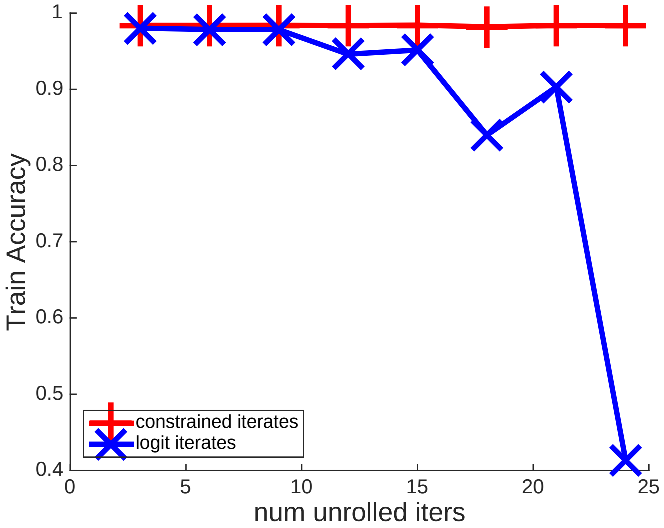


Figure 8.5: Relative Bayes accuracy vs. the number of unrolled iterations. We find that unconstrained optimization is unstable when unrolled for many iterations.

employing unconstrained logit iterates results in higher-variance gradient norms, and high-variance gradients result in lower-quality models. See the figures' captions for details. All of our experiments were done with single-precision floats. It's possible that some of this instability results from numerical overflow, which could perhaps be alleviated by using new GPU kernels that support double precision. Unfortunately these are not compatible with the deep learning library we use.

The left of Fig. 8.6 displays the impact of gradient clipping on training a model with $k = 3$ on the horses data. Here, during training each parameter gradient is renormalized such that its norm is no bigger than c (if the norm is less than c, the gradient is left unchanged). We consider $c = 1.0, 5.0, 10.0$. Overall, clipping to 1.0 yields both faster optimization and a better final value. This is counterintuitive to us because when using 1.0, 100% of the parameter gradients have norm greater than 1.0. This means that learning places equal weight on each gradient, since they are all renormalized to unit norm. Of course, performance on held-out data is not perfectly correlated with train-time loss, especially when we are susceptible to overfitting. We find that using gradient clipping of 10.0 yields superior test accuracy.

Overall, gradient clipping is a poorly-understood element of the deep learning practitioner's bag of tricks. It corresponds to trust-region optimization: gradient descent is restricted from moving too far in a given iteration, since each gradient is restricted in norm. However, it is very ad-hoc to uniformly scale the entire parameter gradient to be within a certain ball. This is because the gradient norm depends on the number of parameters, and different coordinates of the gradient correspond to qualitatively different parameters (e.g., bias terms vs. elements of a convolution kernel). We encourage future work that more systematically applies trust region methods to deep learning training.

Next, we consider the impact of automatically learning a separate step size for each iteration of gradient-based energy minimization. This improves the expressivity of the model and eliminates the need to select step sizes using an expensive outer loop of grid search. The learned step sizes are given on the right of Fig. 8.6 for a model trained on the horses data with 20 unrolled iterations. When we unroll unconstrained optimization of logits, the learned learning rates are nearly monotonically increasing. For unrolled mirror descent, the learning rates also increase at the end, but are more flat overall. For both, there are various reasons why the model may place more

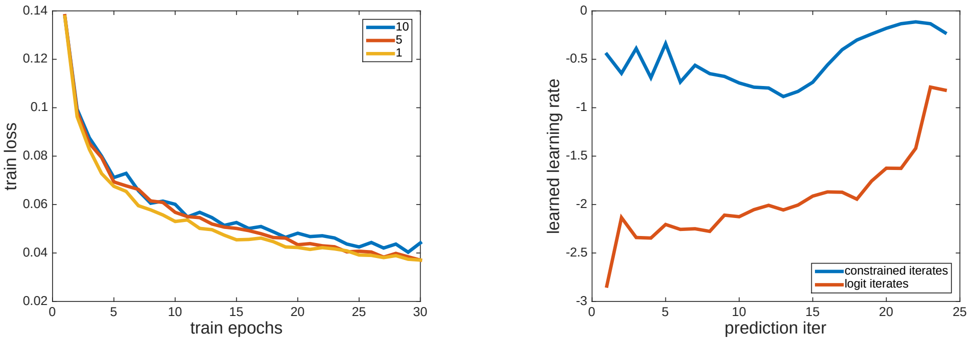


Figure 8.6: Left: Effect of gradient clipping on the dynamics of learning. Right: Learned per-iteration step sizes for unrolled energy minimization.

weight on later iterations of gradient descent. First, as the iterates get closer to the the energy minimum, the norm of the gradients gets smaller. Using a larger step size may be necessary to overcome this.

Finally, note that we do not perform any experiments where we perform test-time energy minimization using a different number of gradient descent iterations than were used during end-to-end learning. Such an approach was shown to be useful, for example, in Domke (2013a). Overall, we have found it useful to learn per-iteration step sizes. In this case, it is difficult to choose what the step size should be for iterations not used during training. One of the principal advantages in practice of end-to-end learning is that it directly provides a test-time prediction procedure with no hyperparameters than need to be tuned.

8.3.5 Input-Convex Neural Networks

As described in Sec. 3.6, we can constrain our energy to be convex with respect to $\bar{y}$ by restricting all weights in the energy that interact with $\bar{y}$ to be positive. We place no restrictions on bias terms. We use ICNN to refer to this constraint imposed on a SPEN.

Recall that the ICNN constraint is a particular method for achieving convexity. Positivity of the parameters is not a necessary condition, however. Positivity may impose an excessively strong restriction that hinders the expressivity of the model and the dynamics of gradient-based learning.

In Table 8.3 we contrast the performance of a SPEN trained with and without the ICNN constraint on our sequence tagging data. We find that the constraint substantially reduces performance. Table 8.4 presents a similar phenomenon for our grid data. Here, we find that imposing the ICNN constraint eliminates situations where the predicted energy is greater than the energy at the ground truth. However, the model is low quality.

In Fig. 8.7, we plot test-time energy minimization curves for SPENs with and without the ICNN constraint. We find that the curves for the ICNN are extremely flat. This suggests that the model is not learning particularly well. There are two regimes in which we may not learn well. First, all of the SoftPlus non-linearities of the energy could saturate, so gradients of the energy are nearly zero. Second, all of the inputs to the SoftPlus could be large and positive, such that it behaves as the identity function. In this case, the energy network is a linear function, which provides limited expressivity in addition to the local energy network. Perhaps performance could be improved by changing the parameter initialization scheme such that we better avoid these regimes.

See the ends of Sec. 9.1.3 and Sec. 10.3 for discussions of the impact of the ICNN constraint on the performance of SPENs for semantic role labeling and image denoising, respectively. Overall, we find that enforcing convexity never helps.

ICNN	**yes**	**no**
Relative Bayes Accuracy	68.3 ± 8.6	80.1 ± 5.4

Table 8.3: Average relative Bayes accuracy for SPENs trained with and without the ICNN constraint on our 20 sequence tagging problems.

ICNN	**no**	**yes**
Test Accuracy	91.8	90.1
Opt Error Rate	33.7	0

Table 8.4: Effect of the ICNN constraint on performance on the horses data for a SPEN with kernel width 3 in the energy network. A test-time optimization error occurs when the ground truth has a lower energy than the predicted value. ICNN prevents these errors, but it results in low-quality predictions.

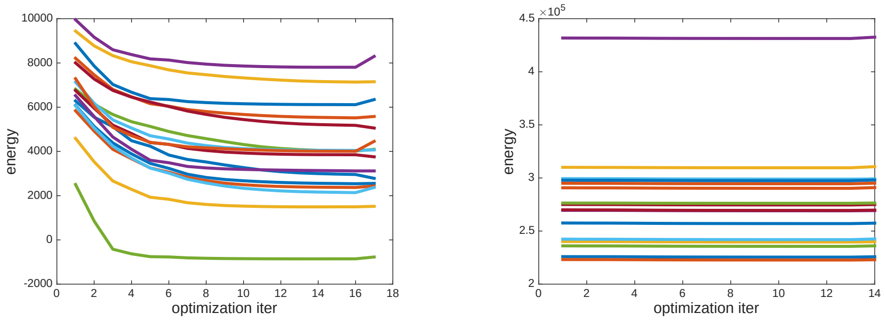
8.4 SPENs vs. CRFs

We consider the following CRF prediction methods.

1. **Exact Maximum Likelihood (MLE)** is available for chain-structured CRFs, where the partition function is computed using dynamic programming. Prediction is performed using exact MBR decoding.

2. **Mean Field (MF)** uses unrolled synchronous mean-field inference. The predictor is unrolled into a computation graph and trained end-to-end, as discussed in Section 2.6.

3. **Gradient-Based Mean Field (GMF)** is the same as **MF**, but we backpropagate through unrolled gradient-based optimization of the mean-field objective using the same techniques used for optimizing SPEN energies.



(a) Non-Convex

(b) Convex

Figure 8.7: Test-time energy minimization for SPENs trained with and without the restriction that they are input-convex. As before, we append the energy of the ground truth as the rightmost point in each curve. In (a), we find that energy minimization for non-convex energies sometimes returns a value greater than the energy of the ground truth. In (b), this never occurs. However, the energy barely changes between the beginning and the end.

4. **Belief Propagation (BP)** As discussed in Section 2.6, we can unroll belief propagation and perform end-to-end learning. We only use this for grids, as it is identical to the exact maximum likelihood method for chains.

As alternatives, we consider

1. **Structured Prediction Energy Network (SPEN)**
2. **Feed-Forward (FF)** uses the same features as the other models, but applies independent logistic regression models for each label.

Going forward, we will employ *configuration* to refer to the combination of a model (e.g., a SPEN or a CRF), coupled with a method for training it (e.g., MF). The above configurations can be compared along multiple axes:

1. **FF vs. Energy-Based Approaches** considers whether joint prediction of y is actually useful, or high accuracy can be achieved simply by predictions that are conditionally independent using high-quality features.

2. **MF** vs. **GMF** considers whether end-to-end learning using gradient-based mean-field is comparable to end-to-end learning using synchronous mean field for a CRF. This isolates the first of the two differences between CRFs and SPENs: the form of the end-to-end learning vs. the form of the energy function.

3. **SPEN** vs. $\{\mathbf{BP}, \mathbf{MF}\}$ considers the importance of the expressivity of the energy function. For SPENs, our energy terms can directly couple together large sets of output variables, by using wide convolutions in the energy network. The analogous receptive field is smaller for the CRFs we consider.

4. **BP** vs. $\{\mathbf{MF}, \mathbf{SPEN}\}$ considers the importance maintaining intermediate values for pairs of output variables vs. maintaining only node-wise quantities. This comparison can also be used to evaluate the impact of inference with a message passing schedule, rather than the synchronously updating all prediction variables at once.

Details of our training procedure for the CRF models are given for grids in the next section. These can be adopted to chains in a straightforward manner.

8.4.1 End-to-End CRF Learning

We next describe concrete methods for training grid CRFs using the principal of direct risk minimization from Section 2.6. We consider unrolled mean-field and loopy belief propagation. The mean-field method can be specialized to chain-structured problems easily. Exact belief propagation is straightforward for chains, so one would not use the method of this section for chains.

The **MF** configuration unrolls synchronous coordinate descent mean field inference into a feed forward computation graph. For each pixel i, j, iteration t of inference maintains a vector $\bar{y}^{(t)}_{i,j} \in [0,1]^2$ that sums to one. The coordinates of this vector contain the beliefs that pixel i, j takes on states 0 and 1, respectively. One update of mean-field is as follows:

$$\bar{y}^{(t+1)}_{i,j} = \text{SoftMax}\left(\psi^{\texttt{left}}_{ij}\bar{y}^{(t)}_{(i,j-1)} + \psi^{\texttt{right}}_{ij}\bar{y}^{(t)}_{(i,j+1)} + \psi^{\texttt{above}}_{ij}\bar{y}^{(t)}_{(i-1,j)} + \psi^{\texttt{below}}_{ij}\bar{y}^{(t)}_{(i+1,j)}\right) \quad (8.1)$$

Here, $\psi^{\texttt{left}}_{ij}$ is the 2×2 matrix of log potentials for the horizontal edge between pixel i, j and the pixel to the left of it. The other potentials are define similarly. We assume the matrices in (8.1) are properly transposed such that it is appropriate to always perform right multiplication. The energy from all local factors has been absorbed into the values of the edge potentials. This update consists of basic linear algebra and a per-pixel softmax. Therefore, it can be accelerated easily on a GPU if we perform updates to all pixels synchronously in parallel. In practice, our implementation differs slightly from (8.1) because we do all computation in log space.

The **GMF** configuration unrolls gradient-based prediction of the same mean-field energy function that **MF** minimizes. This employs the same training method as **SPEN**.

The **BP** configuration unrolls loopy belief propagation into a fixed computation graph that supports backprop. The implementation is more complex, as it does not update beliefs for the entire image in parallel, but instead employs a schedule. Each pixel has four messages entering it: `above`, `below`, `left`, and `right`. Each iteration t of inference updates all of the `above` messages, and then all of the `below` messages, and so on. Each set of messages is updated in a recurrent fashion. For example, the `above` messages are updated for the topmost row of pixels first, and then updated for the second row of pixels, and so on. This is a non-traditional message passing schedule that is designed to exploit vectorized computation.

We next describe how we update the `above` messages. The others are updated analogously. Here, w distinct messages, corresponding to a single row, are updated in parallel for the $h \times w$ image. Additionally, these updates are parallelized across a minibatch of images. The update equation is:

$$\texttt{above}^{(t+1)}_{i+1,j} = \text{SoftMax}\left(\psi^{\texttt{left}}_{ij}\texttt{left}^{(t)}_{i,j} + \psi^{\texttt{right}}_{i,j}\texttt{right}^{(t)}_{i,j} + \psi^{\texttt{above}}_{i,j}\texttt{above}^{(t+1)}_{i,j}\right) \tag{8.2}$$

Here, the $\texttt{above}^{(t+1)}_{i+1,j}$ message is the downwards message from i, j to $(i{+}1), j$. It enters $i + 1$ from above. The appearance of $\texttt{above}^{(t+1)}_{i+1,j}$ on the right hand side, rather than $\texttt{above}^{(t)}_{i+1,j}$, is the reason that the updates are done in sweeps across the grid rather than in parallel. Again, in practice the computation is done in log space. We do not use any message damping.

Note that we train all of our CRFs by minimizing the cross entropy loss of predicted node marginals, but we could have performed alternative methods that more explicitly cast the CRF as probabilistic model. For example, we could treat the node and edge marginals computed by **BP** as the gradient of the Bethe approximation to the partition function (Yedidia et al., 2003). However, we only consider the node-level cross entropy, as this was shown to perform the best on the Weizmann horses in Domke (2013a). Furthermore, this is a reasonable way to directly train for the accuracy of our models, since we evaluate with node-level Hamming loss.

Next, we provide experiments designed to validate that the above CRF configurations provide a strong baseline for comparing against SPENs. First in Table 8.5 we justify that our inference implementation is correct and that it performs similarly to alternative methods with different message passing schedules. Second, in Table 8.6 we compare the results on the Weizmann horses from Domke (2013a) to the outputs of our methods to demonstrate that end-to-end learning with our particular inference algorithms can yield high-quality CRFs. A direct comparison to Domke (2013a) is not possible, since the experiments differ in details of the unrolled inference, the size of the images considered, and the functional form of the features used, but the overall results are similar. See the figures' captions for more details.

Method	# iters	BP	Junction Tree
MF	5	0.055	0.054
MF	25	0.051	0.051
BP	5	0.00013	0.0014
BP	25	1.7e-07	0.0014

Table 8.5: Justifying our choice of update schedules for **MF** and **BP** inference by comparing accuracy to the outputs of the widely-used libDAI library (Mooij, 2010), which employs alternative updates. Accuracy is computed as the ℓ_∞ norm between node marginals (smaller is better). We report average accuracy on 18 5×5 grid MRFs, where each value of the potentials is drawn independently from $N(0, 2)$. Rows indicate the method we use. The final two columns indicate which libDAI method was employed. Junction Tree computes the true marginals. BP are marginals computed using loopy belief propagation, but with a different schedule than ours. Overall, our inference is very accurate.

	FF	**MF**	**BP**
Train	90.5	92.2	93.6
Test	87.5	89.4	90.7

	FF	**MF**	**BP**
Train	92.2	95.8	97.9
Dev	88.0	90.1	93.0
Test	88.2	89.6	91.9

Table 8.6: Justifying our choice of CRF inference algorithms and our modifications of the Weizmann horses dataset by comparing end-to-end learning results from Domke (2013a) (left) to our methods (right). The Domke (2013a) results unroll 40 iterations, but obtain similar results with 10, and are trained with the same loss as our methods. Unlike us, they unroll TRW message passing for **BP**. We report Hamming accuracy. The improved performance of our methods over those of Domke (2013a) is likely due to the use of deep convolutional features.

Finally, note that our mean-field approach differs from convention, in that it updates all beliefs in parallel. This is to provide enhanced performance on a GPU. Such updates may perform low quality energy minimization, however, and may not even be convergent. To understand the performance effect of such updates, we contrast **MF** and **GMF** on our synthetic sequence tagging data. **GMF** performs proper gradient-based minimization of the mean-field energy, whereas **MF** performs fixed point iteration that is not even guaranteed to converge. For each of the 20 test problems, we run each inference method using the model used to generate the data. We compute the average difference between the outputs of **MF** and **GMF**, averaged across 125 examples per test problem. We find that the mean absolute difference between node marginals is only 0.06 on average.

8.4.2 Chain-Structured Problems

8.4.2.1 Accuracy

Most of this thesis focuses on situations where SPENs outperform baseline models, including the next section on grid-structured CRFs. In many of those cases, it is difficult to identify what accounts for SPENs' superior performance. By evaluating on data drawn from linear-chain CRFs in this section, we are able to isolate the importance of various SPEN details. This comes with the cost that the results of this section are largely negative for SPENs. Unsurprisingly, SPENs are outperformed by various CRF methods, since the data is generated by CRFs.

MLE	FF	MF	GMF	SPEN
95.8 ± 1.6	65.1± 9.4	91.6 ± 2.6	89.6 ± 2.9	80.1 ± 5.4

Table 8.7: Relative Bayes accuracy percentage for various methods on the synthetic data drawn from a CRF. Performance is averaged over 20 different learning problems.

Our results are presented in Table 8.7. First, observe that the relative Bayes accuracy for **MLE** is 95.8%, not 100%. This is because we fit the model on limited samples from the true underlying distribution. This provides a upper bound for achievable performance using CRFs on this data. Next, note that the performance of **FF** is poor. This is by construction: we generate data with strong coupling between output variables, and we restrict the feature network to use width-1 convolutions. Since the exact **FF** architecture acts as the local terms for the other models, any performance improvement of the other structured prediction techniques must have been achieved by capturing the interactions among outputs. Next, **MF** performs worse than **MLE**. This is because **MLE** performs MBR decoding with exact marginal inference, whereas **MF** employs approximate marginal inference. Note that **GMF** performs similarly to **MF**. It seems that the use of a mean-field energy function is more damaging than the particular optimization algorithm used to minimize it.

Finally, unfortunately **SPEN** performs worse than the other methods, but better than **FF**. We found that the **SPEN** performance was fairly robust to various hyperparameters, such as whether we optimize simplex-constrained iterates or unconstrained logit iterates.

The controlled experiments established in the previous section illuminate the source of SPENs' poor performance on this data:

- In **MF** vs. **GMF**, the configurations are identical except for the unrolled optimization method they train end-to-end. This suggests that both coordinate descent and gradient descent are similarly effective optimization methods for the mean field energy, and that both of these are similarly easy to learn end-to-end. In other words, the effects of vanishing/exploding gradients are similar.

- In **GMF** vs. **SPEN**, we unroll the same optimization method, but employ a different functional form for the non-local terms in the energy network. The considerable accuracy difference suggests that this choice of architecture is crucial. Consider a sequence of length 2 and domain size D, where the prediction variable is broken into $\bar{y}_1$ and $\bar{y}_2$. Both are elements of the probability simplex on D elements. The global energy network of the **SPEN** is:

  $$c^\top \text{SoftPlus}\left(B\text{SoftPlus}\left(A_1\bar{y}_1 + A_2\bar{y}_2\right)\right), \tag{8.3}$$

  for appropriately sized matrices and vectors. The analogous term for **MF** is:

$$\bar{y}_1^\top A \bar{y}_2. \tag{8.4}$$

Perhaps **GMF** outperforms **SPEN** because (8.4) is a more parsimonious or easier-to-train approximation of the energy function of the CRF that generated the data. It has terms that explicitly interact components of $\bar{y}_1$ and $\bar{y}_2$, whereas in (8.3) relies on mapping $\bar{y}_1$ and $\bar{y}_2$ to a feature representation, adding these features, and scoring this using a non-linear function. The additive, rather than multiplicative, interaction in the first layer may be problematic.

Note that we experimented with a simpler version of (8.3) that removes the intermediate hidden layer, but the performance was worse.

Overall, we would need to perform similar controlled experiments on more problem domains in order to draw general conclusions about the importance of these various factors for SPEN performance.

8.4.2.2 Speed

Next we compare the speed of inference in a chain-structured CRF vs. a SPEN. For a CRF, the computational complexity of the Viterbi and sum-product algorithms is linear in L, the length of the sequence. On the other hand, the per-iteration complexity of **MF** prediction in a CRF or prediction in a **SPEN** does not depend on L, since the updates can be parallelized. However, we need to perform T iterations of the updates.

In Figure 8.8 we vary L and plot the average number of seconds required to predict on a single batch. For **MF**, we employ $T = 10$ iterations and for **SPEN** we employ $T = 15$. These are typical values used in a variety of experiments. All computations were performed on a GPU.

We find that that the time cost of Viterbi and sum-product is linear in L, but cost of **SPEN** prediction does not depend on L. Unfortunately, in our implementation the time cost of **MF** also increases linearly with L. This is a symptom of software-level issues. The torch bindings to the modern CUDNN libraries are limited, and one of the operations necessary for **MF** (a batch matrix multiply) is not yet available. We do not use any matrix multiplication library calls in our log-space sum-product implementation.

Overall, various implementation-level details can have a tremendous impact on the speed of prediction. For example, we perform sum-product in log space in order to avoid numerical underflow. However, this introduces considerable overhead, as we need to exponentiate many values in the inner loop. In many situations, it would have been numerically safe to avoid log space entirely.

8.4.3 Grid-Structured Problems

It is important to contrast SPENs and CRFs on problems where exact inference in CRFs is intractable. We consider a grid-structured CRF with data-dependent factors between every pixel and its immediate neighbors. Each factor is parametrized by is a 2×2 table of values, where each entry is a linear function of the concatenation of the feature vectors for the two pixels involved. We separately tie the parameters of these linear functions for all horizontal edges and for all vertical edges. All CRF configurations can be used to produce a 'soft prediction' between 0 and 1 for every pixel. As with our SPENs, we train all CRFs by minimizing the average pixel-level cross entropy. We also pretrain our features by maximizing the performance of a local classifier.

In order to avoid tremendous memory overhead when unrolling **MF** and **BP** for many iterations, Domke (2013a) avoids storing values of the messages and beliefs at intermediate iterations of backprop. Instead, these are reconstructed on the fly during back-propagation. We have found that this is unnecessary for our application because high quality predictions can be obtained using fewer than 10 inference iterations.

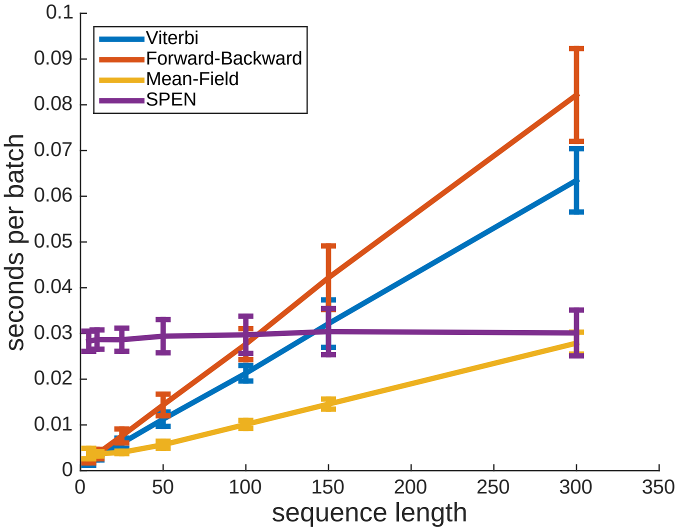


Figure 8.8: Comparison of the average per-batch runtime of prediction in a linear-chain CRF vs. a SPEN, as a function of the length of the sequence. Each step of SPEN inference is easily parallelized across the length of the sequence, whereas the forward-backward and Viterbi algorithms require sequential computation. Mean-field inference in a CRF should have constant runtime, like a SPEN, but details of the libraries used in our implementation prevent this.

Table 8.8 compares the performance of **SPEN**, **MF**, and **BP** on the horses data. **SPEN**-k uses a receptive field size of k for the first convolutional layer in the energy network. All hyperparameters were selected to maximize the performance on the Dev data.

	SPEN-3	**SPEN**-5	**SPEN**-7	**MF**	**BP**
Train	97.4	97.8	97.8	95.8	97.6
Dev	92.8	93.1	93.8	90.8	93.0
Test	91.8	91.9	92.3	89.6	91.9

Table 8.8: Comparing ther performance of SPENs and CRFs on the Weizmann horses dataset. **SPEN**-k uses a filter width of k in the first layer of the energy network.

EXACT	**LP**	**BP**	**MF**	**SPEN**
0.798 ± 0.05	0.795 ± 0.05	0.757 ± 0.06	0.77 ± 0.02	0.8 ± 0.03

Table 8.9: Hamming accuracy for different prediction methods, which are used both during SSVM training and at test time, using the setup of Finley & Joachims (2008) on the Yeast dataset. SPENs perform comparably to EXACT and LP, which provide stronger guarantees for SSVM training.

Here, a SPEN and a CRF can differ in two key ways. First, for **SPEN**-5 and **SPEN**-7, the energy function is more expressive than the CRF. For **SPEN**-3, it is similar. Second, the configurations maintain different representations when reasoning about candidate outputs. Both **SPEN** and **MF** maintain node-wise quantities. However, **BP** maintains quantities between pairs of nodes. For image segmentation, explicitly reasoning about joint configurations between adjacent pixels may be important because the problem relies crucially on edge detection.

We find that **SPEN**-3 outperforms **MF**, despite having similar representational capacity. On the other hand, **BP** outperforms **SPEN**-3. We hypothesize that this is because **BP** maintains edge-wise quantities during inference.

We find that increasing the receptive field of the **SPEN** energy network yields small improvements in performance, since the higher-capacity energy networks are vulnerable to overfitting. The horses dataset isn't large enough to use these expressive networks to their full potential.

8.4.4 Multi-Label Classification

Due to scalability considerations, most prior applications of CRFs to multi-label classification have been restricted to problems with substantially fewer output labels than those considered in Belanger & McCallum (2016). In Table 8.9, we consider the 14-label yeast dataset (Elisseeff & Weston, 2001), which is the largest problem fit using a CRF in Finley & Joachims (2008) and Meshi et al. (2010).

Finley & Joachims (2008) analyze the effects of inexact prediction on SSVM training and on test-time prediction. Table 8.9 considers exact prediction using an ILP solver (**EXACT**, loopy belief propagation **BP**, solving the LP relaxation (**LP**), an adaptation of the end-to-end **MF** approach described in the previous section, and **SPEN**, where the same prediction technique is used at train and test time. Note that here **BP** is not trained end-to-end. Instead max-product BP is used within SSVM learning. All results, besides **SPEN** and **MF**, are from Finley & Joachims (2008). The **SPEN** and **MF** use linear feature networks. We report Hamming accuracy, using 10-fold cross validation.

We use Table 8.9 to make two arguments. First, it demonstrates that training **MF** end-to-end can be a high-quality alternative to other CRF methods. Second, a key argument of Finley & Joachims (2008) is that SSVM training is more effective when the train-time inference method will not under-generate margin violations. Here, **BP** and **SPEN**, which both approximately minimize a non-convex inference objective, have such a vulnerability, whereas **LP** does not, since solving the LP relaxation provides a lower bound on the true solution to the value of (5.4). Since **SPEN** performs similarly to **EXACT** and **LP**, this suggests that perhaps the effect of inexact prediction is more benign for **SPEN** than for **BP**. However, **SPEN** exhibits alternative expressive power to pairwise CRFs, and thus it is difficult to fully isolate the effect of SSVM training on accuracy.

8.5 SSVM vs. End-to-End Learning

All of the experiments in Belanger & McCallum (2016) employ SSVM learning. Overall, we have found that the method behaves unreliably when loss-augmented inference is intractable, and this has motivated our exploration of alternative training techniques. On the other hand, SSVM learning is conceptually straightforward, is easy to implement, and requires less memory overhead than end-to-end learning.

This section provides a couple of experiments using SSVM training. Additional relevant experiments are provided in Chap. 12.

Method	**E2E**	**E2E**	**SSVM**	**SSVM**
ICNN	**no**	**yes**	**no**	**yes**
Train	97.4	95.6	93.0	95.5
Dev	92.8	91.0	89.3	91.1
Test	91.7	90.1	89.2	89.9

Table 8.10: Comparing end-to-end and SSVM learning for **SPEN**-3 and **ICNN**-3 configurations.

Table 8.10 compares end-to-end and SSVM learning on the horses dataset. We train the **SPEN**-3 configuration, along with **ICNN**-3. These are identical, except for the fact that the energy function of **ICNN**-3 is convex with respect to $\bar{y}$. We find that end-to-end training outperforms SSVM training, enforcing convexity hurts the performance of the model, and SSVM training performs poorly with a non-convex energy.

Figure 8.9 plots an approximation of the SSVM train loss for **SPEN**-3 and **ICNN**-3. It is an approximation because we evaluate the loss not using the global optimum of loss-augmented inference, but instead the output of our approximate energy minimization procedure. See the caption for additional discussion.

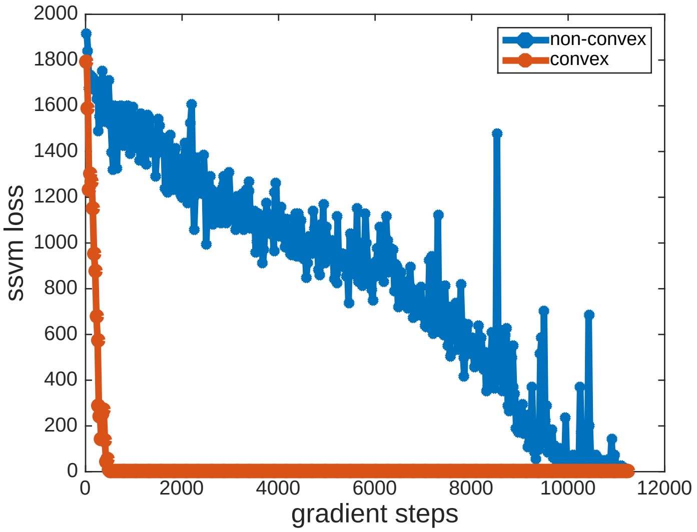


Figure 8.9: Approximate SSVM training loss when training **SPEN**-3 and **ICNN**-3, i.e., when using with non-convex and convex energies. The true SSVM loss is defined in terms of a potentially intractable energy minimization problem. Here, we define the approximate SSVM loss by using the output of approximate energy minimization as a drop-in replacement for the actual energy minimum. As a consequence, the SSVM loss may be zero, even if there are margin violations, when the energy minimization procedure fails to find them. Here, this happens when we have a non-convex loss. The SSVM loss quickly reaches zero, despite the fact that the learned model is low-quality.

8.6 Structure Learning Using SPENs

With SPENs, we can pose structure learning as feature learning in a deep network. This section presents experiments on synthetic multi-label classification data designed to illuminate the potential for SPENs to provide interpretable structure learning.

Namely, we demonstrate that the label measurement matrix, C_1 in the global energy network (3.11), provides a useful tool for analyzing the structure of the dependencies among the labels. The experiments also highlight that SPENs can excel in regimes of limited training data, due to their superior parsimony compared to analogous feed-forward approaches.

To generate data, we first draw a data matrix X with 64 features, with each entry drawn from $N(0,1)$. Then, we generate a 64 x 16 weights matrix A, again from $N(0,1)$. Then, we construct $Z = XA$ and split the 16 columns of Z into 4 consecutive blocks. For each block, we set $Y_{ij} = 1$ if Z_{ij} is the maximum entry in its row-wise block, and 0 otherwise. We seek a model with predictions that reliably obey these within-block exclusivity constraints.

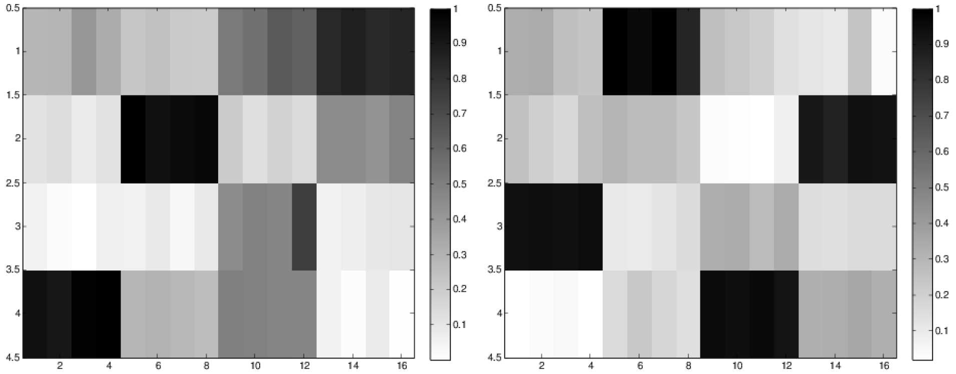
Our architecture uses a global energy network that does not depend on x and a linear function for the feature network. The global energy network has 2 layers and 4 hidden units.

Figure 8.10 depicts block structure in the learned measurement matrix. Measurements that place equal weight on every element in a block can be used to detect violations of the mutual exclusivity constraints characteristic of the data generating process. The choice of network architecture can significantly affect the interpretability of the measurement matrix, however. When using ReLU, which acts as the identity for positive activations, violations of the data constraints can be detected by taking linear combinations of the measurements (a), since multiple hidden units place large weight on some labels. On the other hand, since applying HardTanh to measurements saturates from above, the network learns to utilize each measurement individually. This yields slightly more distinct block structure in (b) than in (a).

Next, in Table 8.11 we compare: a linear classifier, a 3-Layer ReLU MLP with hidden units of size 64 and 16, and our SPEN (with HardTanh activations). Using fewer hidden units in the MLP results in substantially poorer performance.

# train examples	Linear	3-Layer MLP	SPEN
1.5k	80.0	81.6	**91.5**
15k	81.8	96.3	**96.7**

Table 8.11: Comparing performance (F1) on the synthetic task with block-structured mutual exclusivity between labels. Due to its parsimonious parametrization, the SPEN succeeds with limited data. With more data, the MLP performs comparably, suggesting that even rigid constraints among labels can be predicted in a feed-forward fashion using a sufficiently expressive architecture.



(a) ReLU

(b) HardTanh

Figure 8.10: Learned SPEN measurement matrices on synthetic data containing mutual exclusivity of labels within size-4 blocks, for two different choices of nonlinearity in the global energy network. 16 Labels on horizontal axis and 4 hidden units on vertical axis.

We find that the SPEN consistently outperforms the MLP, particularly when training on only 1.5k examples. In the limited data regime, their difference is because the MLP has 5x more parameters, since we use a simple linear feature network in the SPEN. We also inject domain knowledge about the constraint structure when designing the global energy network's architecture.

Next, observe that for 15k examples the performance of the MLP and SPEN are comparable. Initially, we hypothesized that the mutual exclusivity constraints of the labels could not be satisfied by a feed-forward predictor, and that reconciling their interactions would require an iterative procedure. However, it seems that a large, expressive MLP can learn an accurate predictor for this data when presented with lots of examples.

CHAPTER 9
SPENS FOR NATURAL LANGUAGE PROCESSING

This chapter presents a collection of experiments on various NLP tasks where we seek to predict discrete output. They are chosen to highlight different aspects of SPENs' strengths and weaknesses. In all tasks, SPEN performance either achieves the state of the art or is competitive with it.

First, Sec. 9.1 considers semantic role labeling (SRL). The task is challenging for SPENs because the outputs are subject to rigid, non-local constraints. On the other hand, it is clear that there are interesting statistical regularities of outputs that cannot be captured by popular baseline models but could be captured by a global SPEN energy.

After that, Sec. 9.2 considers multi-label classification for tagging documents. Here, we are given no structure a-priori for the interactions among output labels. This is a natural use-case for SPENs because we can perform automatic structure learning by fitting a global energy. We are able to scale to problems with a large number of labels, where existing factor graph approaches would be intractable.

Sec. 9.3 presents applications of SPEN-CRFs to the task of citation field extraction. We are given a string corresponding to a citation in the references or bibliography section of a research paper and need to segment it into fields corresponding to authors, title, venue, date, etc. This is a useful task for evaluating SPEN-CRFs because the outputs are subject to hard constraints on allowed label sequences, similar to the BIO tags explained in Sec. 2.4. There are also global regularities of outputs. For example, author fields tend to occur before title fields.

Finally, Sec. 9.4 contrasts the performance of SPEN-CRFs with high-order CRFs on a popular optical character recognition benchmark. These experiments demonstrate the importance of design decisions for the energy function architecture.

9.1 Semantic Role Labeling

SRL predicts the semantic structure of predicates and arguments in sentences (Gildea & Jurafsky, 2002). For example, in the sentence "I want to buy a car," the verbs "want" and "buy" are two predicates, and "I" is an argument that refers to the wanter and buyer, "to buy a car" is the thing wanted, and "a car" is the thing bought. Formally, given a set of predicates p in a sentence x and a set of candidate argument spans a, we assign a discrete semantic role y to each pair of a predicate and an argument, where y can be either a pre-defined role label or an empty label.

Existing work imposes hard constraints on y. The objective is to minimize the energy:

$$\min_{y} E(y\,; x, p, a) \text{ s.t. } y \in \mathcal{Y}(x, p, a), \tag{9.1}$$

where $\mathcal{Y}(x, p, a)$ is set of feasible joint role assignments. This constrained optimization problem can be solved using integer linear programming (ILP) (Punyakanok et al., 2008) or its LP relaxation (Das et al., 2012). These methods rely on the output of local classifiers that were trained without regard for the structural constraints. More recently, Täckström et al. (2015) account for the constraint structure using dynamic programming at train time. FitzGerald et al. (2015) extends this using neural network features and show improved results.

There are two principal families of hard constraints describing the set $\mathcal{Y}(x, p, a)$. First, for a given predicate the arguments that attach to it cannot overlap. Second, for a given predicate, at most one argument can be take on each of the available 'core roles' (Punyakanok et al., 2008). Core roles are basic relations, like the relationships between the agent and an action or a patient and an action.

9.1.1 Data and Preprocessing and Baselines

We consider the CoNLL 2005 shared task data (Carreras & Màrquez, 2005), with standard data splits and official evaluation scripts. We apply similar preprocessing as Täckström et al. (2015). This includes part-of-speech tagging, dependency parsing, and using the parse to generate candidate arguments.

Our baseline is an arc-factored model for the conditional probability of the predicate-argument arc labels:

$$\mathbb{P}(y|x,p,a) = \Pi_i \mathbb{P}(y_i|x,p,a). \tag{9.2}$$

where $\mathbb{P}(y_i|x,p,a) \propto \exp\big(g(y_i,x,p,a)\big)$. Here, each conditional distribution is given by a logistic regression model. We compute $g(y_i,x,p,a)$ using a multi-layer perceptron (MLP) similar to FitzGerald et al. (2015). Its inputs are discrete features extracted from the argument span and the predicate (including words, pos tags, and syntactic dependents), and the dependency path and distance between the argument and the predicate. These features are transformed to a 300-dimensional representation linearly, where the embeddings of word types are initialized using newswire embeddings from Mikolov et al. (2013). We map from 300 dimensions to 250 to 47 (the number of semantic roles in CoNLL) using linear transformations separated by tanh layers. We apply dropout to the embedding layer with rate 0.5, and train using Adam with default settings (Kingma & Ba, 2015a) to minimize the per-arc cross entropy loss.

When using the negative log of (9.2) as an energy in (9.1), there are variety of methods for finding the optimal $y \in \mathcal{Y}(x,p,a)$. First, we can employ simple heuristics for locally resolving constraint violation. The **Local + H** configuration uses (9.2) and these. We can instead use the AD^3 message passing algorithm (Martins et al., 2011b) to solve the LP relaxation of this constrained problem. We use **Local + AD^3** to refer to this configuration. Since the LP relaxation does not guarantee feasible outputs, we post-process the AD^3 output using the same heuristics as **Local + H**.

9.1.2 SPEN Model

We employ a pretrained version of (9.2) to provide the local energy term of a SPEN. This is augmented with global terms that couple the outputs together.

The SPEN performs continuous optimization over the relaxed set $\bar{y}_i \in S_D$ for each discrete label y_i, where D is the number of possible roles and S_d is the probability simplex on D elements. The preprocessing generates sparse predicate-argument candidates, but we optimize over the complete bipartite graph between predicates and arguments to support vectorization. We have $\bar{y} \in S_D^{n \times m}$, where n and m are the max number of predicates and arguments. Invalid arcs, i.e, those that were filtered by preprocessing, are constrained to the empty label.

9.1.2.1 Global Energy Terms

From the pre-trained model (9.2), we define f_r as the predicate-argument arc features, We also have predicate features f_p and argument feature f_a, given by the average word embedding of the token spans. The hidden layers of any MLP described below are 50-dimensional. Each MLP is two layers, with a SoftPlus in the middle.

Let $\bar{y}_p \in S_D^m$ be the sub-tensor of $\bar{y}$ for a given predicate p and let $z_p = \sum_k \bar{y}_p[:,k] \in [0,1]^m$, where $z_p[a]$ is the total amount of mass assigned to the arc between predicate p and argument a, obtained by summing over possible labels. We also define $m_p = \sum_k \bar{y}_p[k,:] \in \mathbb{R}_+^A$. This is a length-A vector containing how much total mass of each arc label is assigned to predicate p. Finally, define $s_r = \sum_k \bar{y}[:,:,k]$. This is the total mass assigned to arc r, obtained by summing over the possible labels that the arc can take on.

The global energy is defined by the sum of the following terms. The first energy term scores the set of arguments attached to each predicate. It computes a weighted average of the features f_a for the arguments assigned to predicate p, with weights given by z_p. It then concatenates this with f_p, and passes the result through an MLP that returns a single number. The total energy is the sum of the MLP output for every predicate. The second energy term scores the labels of the arcs attached to each predicate. We concatenate f_p with m_p and pass this through an MLP as above. The third energy term models how many arguments a predicate should take on. For each predicate, we predict how many arguments should attach to it, using a linear function applied to f_p. The energy is set to the squared difference between this and the total mass attached to the predicate under $\bar{y}$, which is given by $\sum_k m_p[k]$. The fourth energy term averages m_p over all p and applies an MLP to the result. The fifth term computes a weighted average of the arc features f_r, with weights given by s_r and also applies an MLP to the result. The last two terms capture general topical coherence of the prediction.

9.1.2.2 Constraint Enforcement

As with Täckström et al. (2015), we seek to account for constraints $\mathcal{Y}(x, p, a)$ during both inference and learning, rather than only imposing them via post-processing. Therefore, we include additional energy terms that encourages membership in $\mathcal{Y}(x, p, a)$ using twice-differentiable soft constraints that can be applied to $\bar{y}$. All of the constraints in $\mathcal{Y}(x, p, a)$ express that certain arcs cannot co-occur. For example, two arguments cannot attach to the same predicate if the arguments correspond to spans of tokens that overlap. Consider general binary variables a and b with corresponding relaxations $\bar{a}, \bar{b} \in [0, 1]$. We convert the constraint $\neg(a \wedge b)$ into an energy function $\alpha\text{SoftPlus}(\bar{a} + \bar{b} - 1)$, where α is a learned parameter.

We consider $\mathbf{SPEN + H}$ and $\mathbf{SPEN + AD^3}$, which employ heuristics or AD3 to enforce the output constraints. Rather than applying these methods to the continuous probabilities output by the model (9.2), we use the output of energy minimization.

Model	Dev (WSJ)	Test (WSJ)	Test (Brown)
Local + H	78.0	79.7	69.7
Local + AD^3	78.2	80.0	69.9
SPEN + H	79.0	80.7	69.3
SPEN + AD^3	**79.0**	**80.7**	69.4
Täckström (Local)	77.9	79.3	70.2
Täckström (Structured)	78.6	79.9	**71.3**
FitzGerald (Local)	78.4	79.4	70.9
FitzGerald (Structured)	78.3	79.4	71.2

Table 9.1: SRL Results (F1)

9.1.3 Results and Discussion

Table 9.1 contains results on the CoNLL 2005 WSJ dev and test sets and the Brown test set. We compare the **SPEN** systems and **Local** systems and the best non-ensemble systems of Täckström et al. (2015) and FitzGerald et al. (2015), which have similar overall setups as us. Note that Zhou & Xu (2015) obtain slightly better performance using alternative methods.

For these, 'Local' refers to fitting (9.2) without regard for the output constraints, whereas 'Structured' explicitly considers them during training. We select our SPEN configuration by maximizing performance of **SPEN + AD**^{3} on the dev data. Our best system unrolls for 10 iterations, trains per-iteration learning rates, uses no momentum, and unrolls (5.12).

Overall, **SPEN + AD**^{3} performs the best of all systems on the WSJ test data. We expect our diminished performance on the Brown test set is due to overfitting. The Brown set is not from the same source as the train, dev, and test WSJ data. SPENs are more susceptible to overfitting because the expressive global term introduces many parameters.

Note that **SPEN + AD**^{3} and **SPEN + H** performs identically, whereas **LOCAL + AD**^{3} and **LOCAL + H** do not. This is because our learned global energy encourages constraint satisfaction during gradient-based optimization of $\mathbf{y}$. Using the method of Amos et al. (2017) for restricting the energy to be convex wrt $\mathbf{y}$, we obtain 80.3 on the test set. When taking so few gradient steps, it is hard to understand the impact of convex energies.

9.2 Multi-Label Document Classification

In multi-label classification (MLC), we are given an arbitrary input x and seek to predict a collection of labels $y \in \{0, 1\}^L$. Often, we are given no topology among the labels in advance. This contrasts, for example, with a time series, where y_i is typically more correlated with y_{i+1} than y_{i+10}.

When using a factor graph for MLC, we need to either make a strict assumption about the labels' interactions, or model at least L^2 terms (Ghamrawi & McCallum, 2005; Finley & Joachims, 2008; Meshi et al., 2010; Petterson & Caetano, 2011) , unless we make strict assumptions about labels' dependencies (Read et al., 2011; Niculescu-Mizil & Abbasnejad, 2015). This is prohibitive for large L. In contrast, the general SPEN architecture for MLC given in Section 3.3.1 provides linear scaling in L. Specifically, the per-iteration computational complexity of prediction is linear in L, as is the number of parameters to estimate.

The experiments in Belanger & McCallum (2016) are all on MLC. The problem is well-suited to demonstrating the capabilities of SPENs, since it requires structure learning, which we can do automatically when fitting the SPEN.

9.2.1 Related Work

The most simple multi-label classification approach is to independently predict each label y_i using a separate classifier. This can perform poorly, particularly when certain labels are rare or some are highly correlated. Modeling improvements use max-margin or ranking losses that directly address the multi-label structure (Elisseeff & Weston, 2001; Godbole & Sarawagi, 2004; Bucak et al., 2009).

Correlations between labels can be modeled explicitly using models with low-dimensional embeddings of labels (Ji & Ye, 2009; Cabral et al., 2011; Yu et al., 2014; Bhatia et al., 2015). This can be achieved, for example, by using low-rank parameter matrices. In the SPEN framework, such a model would consist of a linear feature network (3.9) of the form $F(x) = A_1 x$, where A_1 has fewer rows than there are target labels, and no global energy network. While the prediction cost of such methods grows linearly with L, these models have limited expressivity, and cannot capture strict structural constraints among labels, such as mutual exclusivity and implicature. By using a non-linear multi-layer perceptron (MLP) for the feature network with hidden layers of lower dimensionality than the input, we are able to capture similar low-dimensional structure, but also capture interactions between outputs. In our experiments, MLP is a competitive baseline that has been under-explored in prior work.

Our parametrization of the global energy network (3.11) in terms of linear measurements of the labels is inspired by prior approaches using compressed sensing and error-correcting codes for multi-label classification (Hsu et al., 2009; Hariharan et al., 2010; Kapoor et al., 2012). However, these rely on assumptions about the sparsity of the true labels or prior knowledge about label interactions, and often do not learn the measurement matrix from data. We do not assume that the labels are sparse. Instead, we assume their interaction can be parametrized by a deep network applied to a set of linear measurements of the labels.

9.2.2 Experiments

Table 9.3 compares SPENs to a variety of high-performing baselines on a selection of standard multi-label classification tasks. The experiments in this section are presented as they appear in Belanger & McCallum (2016). They all use SSVM learning.

	#labels	#features	# train	% true labels
Bibtex	159	1836	4880	2.40
Delicious	983	500	12920	19.02
Bookmarks	208	2150	60000	2.03

Table 9.2: Properties of the datasets.

	BR	LR	MLP	DMF	SPEN
Bibtex	37.2	39.0	38.9	40.0	**42.2**
Bookmarks	30.7	31.0	33.8	33.1	**34.4**
Delicious	26.5	35.3	**37.8**	34.2	37.5

Table 9.3: Comparison of various methods on 3 standard datasets in terms of F1 (larger is better).

Dataset sizes, etc. are described in Table 9.2. We contrast SPENs with BR: independent per-label logistic regression (a.k.a. the 'binary relevance model'); MLP: multi-layer perceptron with ReLU non-linearities trained with per-label logistic loss, i.e., the feature network equation (3.9) coupled with the local energy network equation (3.10); and LR: the low-rank-weights method of Yu et al. (2014). BR and LR results, are from Lin et al. (2014). The local energy of the SPEN is identical to the MLP.

We also compare to deep mean field (DMF) (Sec. 8.4.1). We consider 5 iterations of mean-field inference in a fully connected pairwise CRF with data-dependent pairwise factors, and perform end-to-end maximum training to minimize the cross entropy loss of the predicted node marginals. Local factors are identical to the MLP classifier, and their parameters are clamped to reduce overfitting (unlike any of the other methods, the DMF has L^2 parameters). Note that we only obtained high performance by using pretrained unary factors from the MLP. Without this, accuracy was about half that of Table 9.3.

We report the example averaged (macro average) F1 measure. For Bibtex and Delicious, we tune hyperparameters by pooling the train and test data and sampling without replacement to make a split of the same size as the original. For Bookmarks, we use the same train-dev-test split as Lin et al. (2014).We seleced 15 linear measurements (rows of C_1 in (3.11)) for Bookmarks and Bibtex, and 5 for Delicious. For SPENs, we obtain predictions by rounding $\bar{y}_i$ above a threshold tuned on held-out data.

There are multiple key results in Table 9.3. First, SPENs are competitive compared to all of the other methods, including DMF, a structured prediction technique. While DMF scales computationally to moderate scales, since it is vectorized and can be run efficiently on a GPU, it cannot scale statistically, since the pairwise factors have so many parameters. As a result, we found it difficult to avoid overfitting with DMF on the Bookmarks and Delicious datasets. In fact, the best performance is obtained by using the MLP unary factors and ignoring pairwise terms. Second, MLP, a technique that has not been treated as a baseline in recent literature, is surprisingly accurate as well. Finally, the MLP outperformed SPEN on the Delicious dataset. Here, we found that accurate prediction requires well-calibrated soft predictions to be combined with a confidence threshold. The MLP, which is trained with logistic regression, is better at predicting soft predictions than SPENs, which are trained with a margin loss. To obtain the SPEN result for Delicious in Table 9.3, we need to smooth the test-time prediction problem with extra entropy terms to obtain softer predictions.

Many multi-label classification methods approach learning as a missing data problem. Here, the training labels y are assumed to be correct only when $y_i = 1$. When $y_i = 0$, they are treated as missing data, whose values can be imputed using assumptions about the rank (Lin et al., 2014) or sparsity (Bucak et al., 2011; Agrawal et al., 2013) of the matrix of training labels. For certain multi-label tasks, such modeling is useful because only positive labels are annotated. For example, the approach of (Lin et al., 2014) achieves 44.2 on the Bibtex dataset, outperforming our method, but only 33.3 on Delicious, substantially worse than the MLP or SPEN. Missing data modeling is orthogonal to the modeling of SPENs, and future work could combine missing data techniques with SPENs.

9.3 SPEN-CRFs for Sequence Tagging

9.3.1 Citation Extraction

Model	F1
Our Baseline CRF	94.47
CRF-SPEN	95.47
Baseline CRF of Anzaroot et al. (2014)	94.41
Soft-DD (Anzaroot et al., 2014)	95.39

Table 9.4: Comparison of F1 scores on Citation Extraction dataset for a CRF-SPEN vs. the specialized global factor graph of Anzaroot et al. (2014) (Soft-DD). Both variants learn global regularities that significantly improve performance. The difference between the performance of Soft-DD and CRF-SPEN ois insignificant.

We next apply SPEN-CRFs to the NLP task of performing text field segmentation on the UMass citation dataset (Anzaroot & McCallum, 2013). It contains 1456 training, 366 testing, and 659 development strings of citations from research papers, segmented into fields (author, title, etc.).

Our modeling approach, closely follows Anzaroot et al. (2014), who extract segmentations using a linear-chain segmentation model, to which they add a large set of 'soft' linear global regularity constraints. Let y be a candidate discrete labeling and let $S(y)$ be the sufficient statistics for a linear-chain factor graph (Sec. 2.3.2).

Imagine, for example, that we constrain predicted segmentations to have no more predicted last names than first names. The numbers of first and last names can be computed by linear measurements $a_{\text{first}}^\top S(y)$ and $a_{\text{last}}^\top S(y)$, respectively. A hard constraint on y would enforce $a_{\text{first}}^\top S(y) - a_{\text{last}}^\top S(y) = 0$. This is relaxed in Anzaroot et al. (2014) to a penalty term

$$c\ell_h \left(a_{\text{first}}^\top S(\bar{y}) - a_{\text{last}}^\top S(\bar{y})\right) \tag{9.3}$$

that is added to the MAP inference objective for the chain-structured base model, where $\ell_h(x) = \max(1 - x, 0)$ is a hinge function. For multiple soft constraints, the overall prediction problem is

$$\arg\min_y \langle \theta, S(y) \rangle + \sum_j c_j \ell_h \left(a_j^\top S(y)\right), \tag{9.4}$$

where θ are the natural parameters of the underlying linear-chain MRF (θ depends on x). In Anzaroot et al. (2014), the authors use a dual decomposition style algorithm for solving (9.4), that crucially relies on the specific structure of the hinge terms ℓ_h. They learn the c_j for hundreds of 'soft constraints' using a perceptron-style algorithm.

To adapt this model as a SPEN-CRF, we consider the same set of measurement vectors a_j, but impose non-local terms that act on marginals μ rather than on the sufficient statistics $S(y)$ of a discrete prediction. Further, we use smoothed hinge functions, aka a SoftPlus, which improve the convergence rate of energy minimization (Rennie, 2005). We find the variational distribution by solving the marginal inference version of (9.4), an instance of a SPEN-CRF energy (6.6):

$$\arg\min_\mu \langle \theta, \mu \rangle - H_{\mathcal{B}}(\mu) + \sum_j c_j \ell_h \left(a_j^\top \mu\right), \tag{9.5}$$

As in Anzaroot et al. (2014), we first learn chain the parameters for the functional dependence of θ on x on the training set. Then, we learn the c_j parameters on the development set, using the method of Sec. 6.4.2, and tune hyperparameters for development set performance. At both train and test time, we ignore any terms in (9.5) for which $c_j < 0$.

We present our results in Tab. 9.4, measuring segment-level F1. We can see that our baseline chain has slightly higher accuracy than the baseline approach of Anzaroot et al. (2014), possibly due to optimization differences. Our SPEN-CRF matches and very slightly beats their soft dual decomposition (Soft-DD) procedure. This is especially impressive because they employ a specialized linear-programming solver and learning algorithm adapted to the task of MAP inference under hinge-loss soft constraints, whereas we simply plug in our general learning and inference algorithms for CRF-SPENs – applicable to any set of energy functions.

Our comparable performance provides experimental evidence for our intuition that preferences about MAP configurations can be expressed as functions of expectations. Anzaroot et al. (2014) solve a penalized MAP problem directly, while our prediction algorithm first finds a distribution satisfying these preferences, and then performs standard MAP inference in that distribution.

9.4 Handwriting Recognition

We next apply CRF-SPENs to the widely-used handwriting recognition dataset of Taskar et al. (2004). It contains images of 6877 handwritten words with an average length of 8 characters per word. Each character is represented as a 16-by-8 binary image. We follow the setup of Weiss & Taskar (2010), splitting the data into 10 equally sized folds, using 9 for training and one to test. We report the cross-validation results across all 10 folds.

The *structured prediction cascades* method of Weiss & Taskar (2010) achieves high performance on this dataset by using a factor graph that models extremely high-order factors of characters (up to 6-grams). Inference in such a model would typically be intractably slow, but their cascadr method prunes the search space for the high-order model using confidence scores from low-order models. Their results are reproduced in Tab. 9.5. The excellent performance of these high-order models is consequence of the fact that the data contains only 55 unique words, written by 150 different people. Once the model has access to enough higher-order context, the problem becomes much easier to solve.

With this in mind, we design two non-convex, non-local energy functions. These energies are intended to regularize our predictions to lie close to known elements of the vocabulary. Our base model is a standard linear-chain CRF with image features on the nodes, and data-independent bigram edge factors. Let $U(\mu) = \sum_n \mu_n$ be a function that takes the concatenated vector of node and edge marginals and sums up all of the node marginals, giving the global unigram expected sufficient statistics. Let $\{u_i\} = \{U(\mu(y_i))\}$ indicate the set of all such unique vectors when applying U to the train set empirical sufficient statistics for each data case y_i. Simply, this gives 55 vectors u_i of length 26 containing the unigram counts for each unique word in the train set.

Our intuition is that we would like to be able to "nudge" the results of inference in our chain model by pulling the inferred $U(\mu)$ to be close to one of these global statistics vectors. In response, we add the following non-convex non-local energy function to the model:

$$G^u_x(\mu) = a(x) \min_i \|u_i - U(\mu)\|_1. \tag{9.6}$$

We learn two variants of this model, which differently parametrize the dependence of the scaling factor $a(x)$ on x. The first does not depend on x and treats a as a constant. The second applies a linear function to a global representation $f(x)$: concretely, we approximate the radial basis function *kernel mean map* (MM) (Smola et al., 2007) using random Fourier features (RFF) (Rahimi & Recht, 2007). This simply involves multiplying each image feature vector in the sequence by a random matrix with 1000 rows, applying a pointwise non-linearity, and setting $f(x)$ to the average of these vectors. In essence, this corresponds to a one-layer multi-layer perceptron with non-learned weights.

N-Grams	2	3	4	5	6
Accuracy	85.02	96.20	97.21	98.27	98.54

Table 9.5: Character-wise accuracy of Structured Prediction Cascades (Weiss & Taskar, 2010) on OCR dataset.

Model	Accuracy
2-gram factor graph (base model)	84.93
G_x^u	94.01
G_x^u (MM)	94.96
G_x^w	98.26
G_x^w (MM)	98.83
55-Class Classifier (MM)	86.06

Table 9.6: Character-wise accuracy of our baselines, and models using learned non-local energies on Handwriting Recognition dataset.

Results of these experiments can be seen in Table 9.6. Adding the non-local energy brings our performance well above the baseline bigram chain model, and our training procedure is able to give substantially better performance using the G_x^u (MM) energy, where the scale of the global energy depends on x.

Note that energy G_x^u, based on unigram sufficient statistics, is not able to capture the relative ordering of letters in the vocabulary words, which the structured prediction cascades models do capture. This motivates us to consider another energy function that is sensitive to order. Let $\{w_i\} = \{\mu_n(y_i)\}$ be the set of unique vectors of concatenated node marginal statistics for the train set. This gives 55 vectors of length $l_i * 26$, where l_i is the length of the ith distinct train word. Next, we define a different energy function to add to our base chain model:

$$L_x^w(\mu) = a(x) \min_i \|w_i - \mu\|_1. \tag{9.7}$$

Once again we implement versions of $a(x)$ that are either a constant or depend on x via MM features. As noted in Weiss & Taskar (2010), giving the model access to this level of high-order structure in the data makes the inference problem extremely easy. Our model outperforms the best structured prediction cascades results, and we note again an improvement from using the featurized over the non-featurized global energy function.

Of course, since the dataset has only 55 actual labels, and some of those are not valid for different input sequences due to length mismatches, this is arguably a classification problem as much as a structured prediction problem. To address this, we create another baseline, which is a constrained 55-class logistic regression classifier (constrained to only allow choosing output classes with appropriate lengths given the input). We use our same global mean-map features from the G_x^* (MM) variants of the structured model and report these results in Tab. 9.6. We also tune the number of random Fourier features as a hyperparameter to give the classifier as much expressive power as possible. As we can see, the performance is still significantly below the best structured models, indicating that the interplay between local and global structure is important. ‘

CHAPTER 10
SPENS FOR IMAGE DENOSING

In this chapter, we use SPENs to denoise grayscale images. The task is a natural application of SPENs because the output is fundamentally continuous, and thus we do not need to convert the output of energy minimization to a discrete prediction. Our experiments compare methods along two axes: the expressivity of the energy function employed and the sophistication of the unrolled optimizer that is trained end-to-end. Specifically, we contrast energies that are simple and convex with general energies given by a deep network. We also contrast an optimization method that is specifically designed to efficiently leverage the factorization structure of the simple energy with a generic first-order optimization method. For our particular task, we find that it is more effective in practice to use an expressive deep energy, even if this prevents us from using sophisticated optimizers.

10.1 Denoising by MAP Inference

Let $x \in [0,1]^{w \times h}$ be an observed grayscale image. We assume that it is a noisy realization of a latent clean image $\bar{y} \in [0,1]^{w \times h}$, which we estimate using MAP inference. Consider a Gaussian noise model with variance σ^2 and a prior $\mathbb{P}(\bar{y})$. The associated energy function is:

$$E_x(\bar{y}) = \|\bar{y} - x\|_2^2 - \sigma^2 \log \mathbb{P}(\bar{y}). \tag{10.1}$$

Here, the feature network is the identity. The first term corresponds to a local energy term and second is a global energy that depends on $\bar{y}$ but not x. Depending on how we evaluate predictions, it may actually be optimal to predict the posterior mean, rather than the MAP value. However, this chapter focuses on the framework of MAP-based prediction as it yields more natural approximation algorithms for SPENs. In addition, the two approaches may be very similar if the posterior is highly concentrated.

There are three general families for the prior. First, it can be hard-coded, using assumptions about the smoothness of images. Second, it can be learned by approximate density estimation. Third, given a collection of $\{x, \bar{y}\}$ pairs, we can perform supervised learning, where the prior's parameters are discriminatively trained such that the output of a particular algorithm for mimimizing (10.1) is high-quality. End-to-end learning has proven to be highly succesful for the third approach (Tappen et al., 2007; Barbu, 2009; Schmidt et al., 2010; Sun & Tappen, 2011; Domke, 2012; Wang et al., 2016), and thus it is important to evaluate the methods of this paper on the task.

Much of the existing work on end-to-end training for denoising considers some form of a field-of-experts (FOE) prior (Roth & Black, 2005). We consider an ℓ_1 version, which assigns high probability to images with sparse activations from K learned filters:

$$\mathbb{P}(\bar{y}) \propto \exp\left(-\sum_k \|(f_k * \bar{y})\|_1\right). \tag{10.2}$$

Wang et al. (2016) perform end-to-end learning for (10.2), by unrolling proximal gradient methods that analytically handle the non-differentiable ℓ_1 term. Note that (10.2) is convex with respect to $\bar{y}$.

This paper assumes we only have black-box interaction with the energy. In response, we alter (10.2) such that it is twice differentiable, so that we can unroll generic first-order optimization methods. The non-differentiability of (10.2) results from the $\|\cdot\|_1$ term, which is a sum of terms passed through absolute value functions. We approximate (10.2) by leveraging a SoftPlus with a temperature of 25, replacing the absolute value function $|s|$ with:

$$\text{SoftAbs}(s) = 0.5\ \text{SoftPlus}(s) + 0.5\ \text{SoftPlus}(-s). \tag{10.3}$$

The principal advantage of learning algorithms that are not hand-crafted to the problem structure is that they provide the opportunity to employ more expressive energies. In response, we also consider a deeper prior, given by:

$$\mathbb{P}(\bar{y}) \propto \exp\left(-\text{DNN}(\bar{y})\right). \tag{10.4}$$

Here, $\text{DNN}(\bar{y})$ is a general deep convolutional network that takes an image and returns a number. The architecture in our experiments consists of a $7 \times 7 \times 32$ convolution, a SoftPlus, another $7 \times 7 \times 32$ convolution, a SoftPlus, a $1 \times 1 \times 1$ convolution, and finally spatial average pooling. A $1 \times 1 \times 1$ convolution is identical to applying a linear transformation to the features at each pixel that maps each pixel to a single number. The method of Wang et al. (2016) cannot handle this prior.

10.2 Experimental Setup

We evaluate on the 7-Scenes dataset (Newcombe et al., 2011), where we seek to denoise depth measurements from a Kinect sensor. Our data processing and hyperparameters are designed to replicate the setup of Wang et al. (2016), who demonstrate state-of-the art results for energy-minimization-based denoising on the dataset. Table 10.1 summarizes our results for a selection of configurations. Example outputs are given in Figure 10.1. We train using random 96 × 128 crops from 200 images of one scene. We report PSNR for 5500 images from the other 6 scenes.

We expect that the conclusions of our depth denoising experiments would carry over to the task of denoising 3-channel natural images. In Wang et al. (2016), the authors successfully use the same architecture for both tasks.

10.3 Results and Discussion

Our experiments here focus on end-to-end learning. See Sec. 12.4 for additional experiments on the 7-Scenes data using alternative SPEN learning methods, including SSVM.

The first row of Table 10.1 presents various baselines. **BM3D** is a widely-used non-parametric method (Dabov et al., 2007). **FilterForest** adaptively selects denoising filters for each location (Fanello et al., 2014). **ProximalNet** is the system of Wang et al. (2016). **FOE-20** is an attempt to replicate **ProximalNet** using end-to-end SPEN learning. We unroll 20 steps of gradient descent with momentum 0.75 and use the modification in (10.3). Note it performs similarly to **ProximalNet**, which unrolls 5 iterations of sophisticated optimization. If we train a feed-forward convnet, using the same architecture as our DNN prior, but without spatial pooling, we obtain 37.0.

The next set of results considers improved instances of the FOE model. First, **FOE-20+** is identical to **FOE-20**, except that it employs the average loss (5.15), uses a momentum constant of 0.25, and treats the learning rates η_t as trainable parameters. We find that this results in both better performance and faster convergence. Of course, we could achieve fast convergence by simply setting T to be small. In response, we consider **FOE-3**. This only unrolls for $T = 3$ iterations and obtains superior performance.

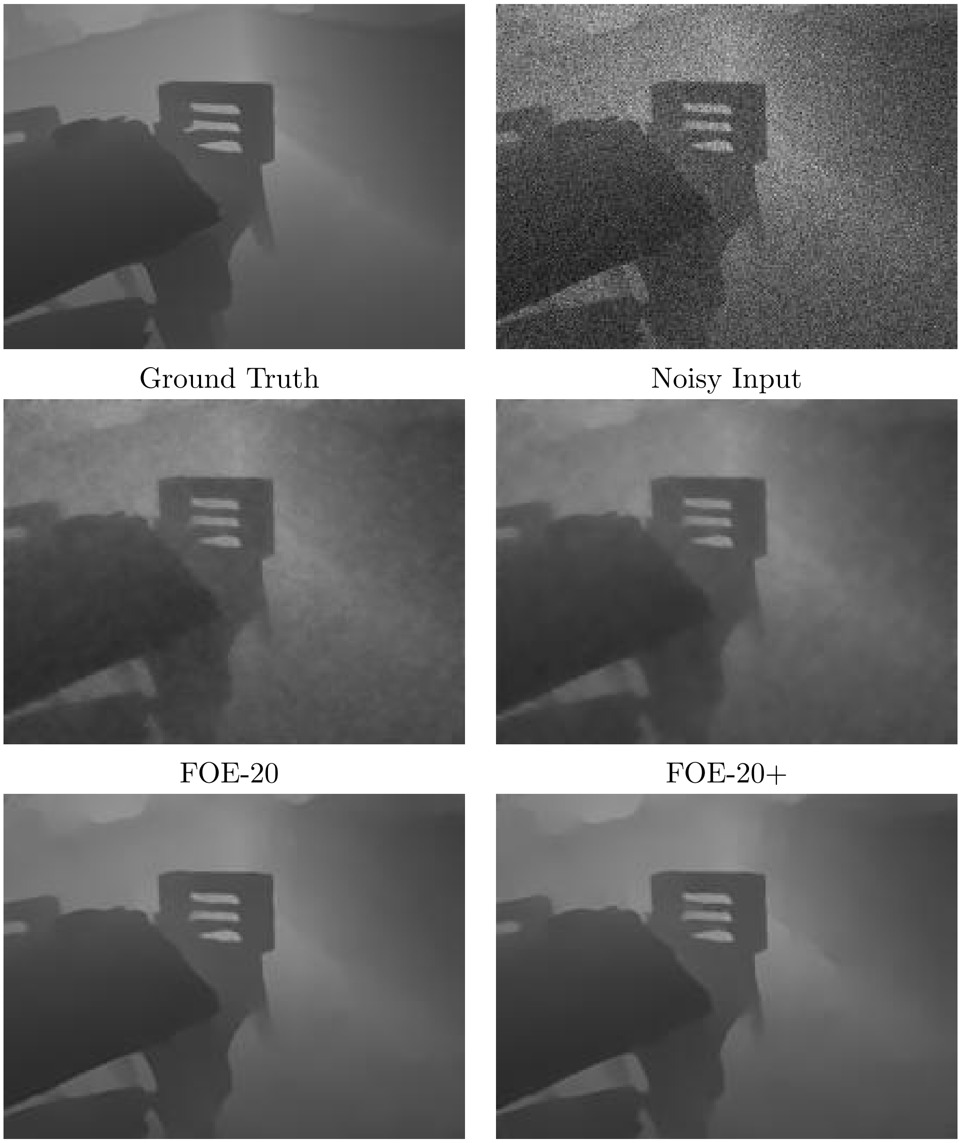
The final two results are with the DNN prior (10.4). **DeepPrior-20** unrolls 20 steps of gradient descent with a momentum constant of 0.25. The gain in performance is substantial, especially considering that a PSNR of 30 can be obtained with elementary signal processing. Similar to **FOE-3** vs. **FOE-20+**, we experience a modest performance gain using **DeepPrior-3**, which only unrolls for 3 gradient steps but is otherwise identical.

BM3D	**FilterForest**	**ProximalNet**	**FOE-20**
35.46	35.63	36.31	36.41
FOE-20+	**FOE-3**	**DeepPrior-20**	**DeepPrior-3**
37.34	37.65	40.3	40.4

Table 10.1: Denoising Results (PSNR)

In general, it is superior to only unroll for a few iterations. One possible reason is that the shallow depth of the unrolled architecture is easier to train. Truncated optimization with respect to $\bar{y}$ may also provide an interesting prior over outputs (Duvenaud et al., 2016), which can be particularly useful because the Gaussian noise model assumed in (10.1) is not characteristic of the data collection process (Wang et al., 2016). This is consistent with the observation of (Wang et al., 2014) that better energy minimization for FOE models may not improve PSNR. Also, note that unrolling for 20 iterations often results in over-smoothed outputs for all of the configurations. This performance is also well-motivated by theory if we only unroll for a single iteration, due to connections between denoising autoencoders, score matching, and energy-based models (Sec. 7.11).

Finally, note that we were unable to achieve reasonable performance when enforcing the ICNN constraint (Sec. 3.6), which restricts all of the parameters of the convolutions to be positive. Unfortuately, this condition prevents the filters in the low levels of the SPEN energy to act as edge detectors and as filters that encourage pixels' values to be close to the average of their neighbors. Both of these are important for high-quality denoising. The FOE energy is convex, even if we do not constraint the parameters to be positive. We can achieve good performance with an FOE, but not with a ICNN-constrained FOE. In future work, it may be fruitful to employ alternative methods to the ICNN constraint for enforcing convexity.



Ground Truth

Noisy Input

FOE-20

FOE-20+

DeepPrior-20

DeepPrior-3

Figure 10.1: Example Denoising Outputs

CHAPTER 11

CAPTURING UNCERTAINTY WITH SPENS

When learning SPENs, there are multiple sources of uncertainty we need to account for:

- **Uncertainty during Learning:** Our energy function might not capture the data well, especially at the beginning of learning. Therefore, deterministically following gradients of the energy may not efficiently explore high-quality output regions.

- **Uncertainty about Predictions:** Our prediction problem may be fundamentally multi-modal, where there are multiple values of $\bar{y}$ that are likely given x. In many applications, it can be useful to predict multiple values, rather than a single point, in order to account for the diversity of likely values.

To address both of these sources of uncertainty, it is important to use randomized methods at train and test time that explicitly explore the space of outputs and to train using a loss function that rewards models that capture the multiple modes.

In the previous chapters, we always predict a single output $\bar{y}$ using SPEN energy minimization. Furthermore, when we train end-to-end using a loss function such as the mean squared error (MSE), our loss is minimized when we predict the conditional expectation of $\mathbb{P}(\bar{y}|x)$. This may be inappropriate when the data's conditional distribution $\mathbb{P}(\bar{y}|x)$ is multi-modal.

Consider, for example, the 2-dimensional distribution in Fig. 11.1. We assume that this contains draws from the data's true conditional distribution $\mathbb{P}(\bar{y}|x)$ for some

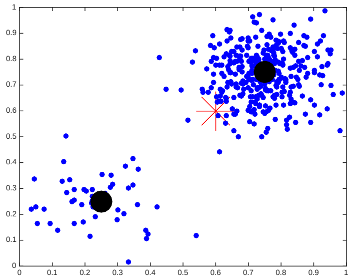


Figure 11.1: Example data drawn from a GMM (blue), cluster centers (black), and point that minimizes the mean-squared error to the data (red).

x. Here, the data is clustered around two modes, and the top right cluster is more likely than the bottom left one. The MSE will be minimized if we predict the red star, even though this value has low density under the data distribution. This is because fitting a model with the MSE implicitly assumes a uni-modal Gaussian distribution for the data.

See Sec. 2.2.4 for a discussion regarding the distinction between training to maximize the conditional likelihood of the data vs. training to minimize the loss of predictions. In the first, the model takes x and returns the parameters θ for a parametrized conditional distribution $\mathbb{P}_\theta(\bar{y}|x)$. In the second, the model takes x and predicts a point estimate for $\bar{y}$. The distinction is particularly important when $\mathbb{P}_\theta(\bar{y}|x)$ integrates out latent variables. Note that we could capture the data quite accurately in Fig. 11.1 if given x we predicted the parameters of a Gaussian mixture model (GMM).

Conditional density estimation of $\mathbb{P}_\theta(\bar{y}|x)$ is often intractable, but it may also be unnecessary. In many applications, the practitioner does not require access to the full conditional distribution. Instead, he or she is only interested in the values of $\bar{y}$ that are likely. For Fig. 11.1, for example, it would be sufficient to only return the cluster centers (in black). With this, he or she would not be able to make an accurate point estimate of $\bar{y}$, but would be able to conclude with confidence that $\bar{y}$ is close to either $c_1 = [0.25, 0.25]$ or $c_2 = [0.75, 0.75]$. Therefore, we could view this is a prediction problem where the predictor returns the set $\{c_1, c_2\}$.

Predictors that return sets of diverse predictions appear in a variety of machine learning contexts. For example, in information retrieval it is often important to return a list of results. Say the query is 'jaguar.' It is useful to return results both about cars and jungle cats in order to ensure that at least one relevant document is provided to the user.

We need to take care when training SPENs so that we are able to use the energy to accurately predict diverse sets of outputs, as the available training methods either employ the conditional likelihood perspective or the loss minimization perpective. For example, in end-to-end learning we use the energy function to define a prediction procedure, and train this procedure to minimize the loss of a point estimate. If we use end-to-end training with the MSE to fit the data in Fig. 11.1, it will return an energy function that has its minimum at the red star. As a result, this energy is useless if we seek to predict diverse sets of likely outputs. On the other hand, we can directly maximize the conditional likelihood of the Gibbs distribution

$$\mathbb{P}(\bar{y}|x) \propto \exp(-E_x(\bar{y})). \tag{11.1}$$

The resulting energy will have multiple local minima whenever the data is multimodal.

The goal of this chapter is to bridge the gap between conditional likelihood maximization and end-to-end approaches for training SPENs. Likelihood maximization is undesirable because exact approaches are typically intractable and approximations often require MCMC methods that are slow and difficult to tune. End-to-end learning is attractive because it returns not just an energy function, but also an automatically tuned prediction algorithm tailored to the properties of the energy.

In response, we develop a method for end-to-end training of a randomized predictor that returns diverse sets of likely outputs. This adapts the approach of (Guzman-Rivera et al., 2012) for training a fixed ensemble of independent predictors and can also be seen as a non-probabilistic method for fitting the randomized optimum model of Tarlow et al. (2012). Our experiments contrast the end-to-end approach with conditional density estimation of (11.1) using MCMC-based approximate maximum likelihood training. We find that the first approach is more straightforward, is substantially easier to tune, and may be a useful modeling technique for future SPEN applications.

11.1 Evaluating Set-Valued Predictors

Our experiments in Sec 11.4.1 evaluate our models in terms of their ability to identify the multiple modes of the data. Consider a predictor that inputs x and returns a set of predictions $\bar{y}_1, \ldots, \bar{y}_K$. Let $\bar{y}^*$ be the ground truth. Let $\Delta(\bar{y}, \bar{y}^*)$ be a cost function that measures the difference between a prediction and the ground truth. We define the K-oracle cost (Guzman-Rivera et al., 2012) of the samples as:

$$\Delta_K(\bar{y}_1, \ldots, \bar{y}_K, \bar{y}^*) = \min_k \Delta(\bar{y}_k, \bar{y}^*). \tag{11.2}$$

This is low whenever any of the predictors is close to $\bar{y}^*$. When $\mathbb{P}(\bar{y}|x)$ is multi-modal, the loss will be low only when the set of predictions is diverse and covers the high-density regions of $\mathbb{P}(\bar{y}|x)$. In many applications, (11.2) is precisely the quantity of interest when evaluating test-time performance. For example in information retrieval, we may be interested in recall at K.

In future work, it may be interesting to employ loss functions that explicitly encouraging diversity among the predictions (Kulesza & Taskar, 2010; Guzman-Rivera et al., 2014). Our approach could also be used as a way to train the scoring model and sampler used for approximate minimum Bayes risk prediction (Premachandran et al., 2014).

11.2 Sampling-Based Approximate Maximum Likelihood

First, we consider approximate conditional likelihood estimation of the Gibbs distribution (11.1). For the sake of notational simplicity, this section considers the un-conditional Gibbs distribution:

$$\mathbb{P}(\bar{y}) \propto \exp(-E(\bar{y})). \tag{11.3}$$

The negative log likelihood of a given training sample $\bar{y}_i$ under (11.3) is:

$$L(\bar{y}_i) = -E(\bar{y}_i) + \log \int_{\mathcal{Y}} d\bar{y} \exp\left(-E(\bar{y})\right) \tag{11.4}$$

Here, the integral is taken over the set $\mathcal{Y}$ of permissible $\bar{y}$, and we assume the integral, also known as the partition function, to be finite. Unfortunately, this integral is intractable for all but the simplest models.

The *contrastive divergence* (CD) learning method (Hinton, 2002) uses a single sample $\bar{y}_s$ to provide a Monte Carlo approximation of the contribution of the second term to the maximum likelihood gradient:

$$\nabla L(\bar{y}_i) \approx -\nabla E(\bar{y}_i) + \nabla E(\bar{y}_s) \tag{11.5}$$

In CD, we do not seek independent samples from (11.1), but instead samples that are both likely under (11.1) and in the neighborhood of the ground truth $\bar{y}_i$. Random exploration of this neighborhood is performed using just a couple steps of MCMC on (11.3), where the sampling is initialized at the ground truth. Performance may be improved by collecting multiple $\bar{y}_s$ using parallel MCMC chains. When samples are collected using HMC, and the energy function is parametrized by a deep network, contrastive divergence is also known as *contrastive back-propagation* (Mnih & Hinton, 2005; Hinton et al., 2006a; Ngiam et al., 2011).

Since each MCMC trajectory for CD is reset to the ground truth, it does not explore far from the observed data. Therefore, training may make small holes for the observed data points and do nothing to shape the overall energy function. In general, CD is not a valid stochastic approximation method for minimizing (11.4).

Instead of CD, we can perform actual stochastic MLE training (Younes, 1989). This is also known as *persistent contrastive divergence* (Tieleman, 2008). It is very similar to CD, except that we do not reset the MCMC chain at the ground truth every time a sample is collected. As long as the parameters of $E(\cdot)$ are updated using a sufficently small step size, $\bar{y}_s$ can be regarded as a sample from (11.3). With this, (11.5) is a valid stochastic gradient (i.e., it's correct in expectation). When $E(\cdot)$ is convex in its learned parameters, this method will converge to the true MLE solution, subject to various conditions on the choice of step size schedule and how well-behaved $E(\cdot)$ is (Swersky et al., 2010).

Alternatively, we can directly approximate the log partition function using a few particles:

$$\log \int_{\mathcal{Y}} d\bar{y} \exp\left(-E_{x_i}(\bar{y})\right) \approx \log \sum_{k=1}^{N} \exp\left(-E(\bar{y}_k)\right). \tag{11.6}$$

This may be a reasonable assumption when the density is highly concentrated in a small region. The approach is popular in many NLP applications, such as semantic parsing (Liang et al., 2011), where low-energy samples are gathered using approximate k-best MAP inference methods such as beam search. Applying this method to continuous problems presents a variety of challenges, however. For example, it is difficult to guarantee that the set of low-energy particles is diverse and not infinitessimally close to each other.

Overall, MLE updates have a simple form: they push down the energy of the ground truth and pull up the energy of configurations other than the ground truth. One potential issue with this approach is that it may not properly shape the energy landscape in regions away from the ground truth such that they can be traversed by gradient-based prediction. In particular, if the energy network is very high capacity, the global optimum will be obtained when the energy is flat for all values of $\bar{y}$ that are not observed and very small for ground truth configurations.

This thesis focuses on the regime where the energy function is a black box that only provides subroutines for forward and back-propagation. Here, it is natural to sample from (11.3) using HMC. However, when the energy function has known structure, it may be preferrable to employ fast-mixing alternatives such as slice sampling (Neal, 2003).

After learning the energy function using MLE, we can predict diverse sets of likely outputs by by sampling from (11.1) at a low temperature.

11.3 End-to-End Learning for Randomized Set-Valued Predictors

While density estimation is conceptually attractive, it may not yield good K-oracle cost (11.2) in practice. First of all, it does not directly provide a test-time prediction procedure, and thus we will need to separately tune our prediction procedure. Second, the energy function has limited expressivity, and its capacity may be wasted fitting the distribution in low-density regions.

This section proposes an alternative approach, where we directly minimize the K-oracle cost (11.2) end-to-end. Given an energy function $E_x(\cdot)$, let $\mathbb{P}_R(\bar{y}; E)$ be a distribution over randomized predictions. This distribution may be completely different than the Gibbs distribution (11.1) induced by the energy.

Furthermore, we assume that $\mathbb{P}_R(\bar{y}; E)$ supports the *reparametrization trick* (Kingma & Welling, 2014), where samples can be obtained by evaluating a differentiable, deterministic function $M(x, \epsilon)$ that depends on x and an external source of randomness $\epsilon \sim \mathbb{P}_\epsilon$. The advantage of the reparametrization trick is that the distribution $\mathbb{P}_\epsilon$ we take an expectation with respect to his not part of our learned model. As a result, we can use straightforward stochastic gradient training.

With this, the expected K-oracle cost (11.2) can be written as:

$$\mathbb{E}_{\bar{y}_k \sim \mathbb{P}_R(\bar{y}\ E)} \Delta_K(\bar{y}_1, \ldots, \bar{y}_K, \bar{y}^*) = \mathbb{E}_{\epsilon_1, \ldots, \epsilon_K \sim \mathbb{P}_\epsilon} \Delta(M(x, \epsilon_1), \ldots, M(x, \epsilon_1), \bar{y}^*) \qquad (11.7)$$

It is straightforward to compute a stochastic subgradient of the right hand side. We first sample K random values ϵ_k and evaluate the loss $\Delta(M(x, \epsilon_k), \bar{y}^*)$ for each. Then, we set ϵ to whichever ϵ_k yielded the lowest loss and peform back-propagation in $\Delta(M(x, \epsilon), \bar{y}^*)$. It may seem that this approach wastes lots of computation, since only one of the K forward evaluations is used to actually update the parameters. However, it would introduce considerable overhead to use a loss function that places non-zero gradients on all samples, as we would need to back-propagate K separate times.

For a given energy function, there are many ways to define a randomized sampling procedure that supports the reparametrization trick. First, we can unroll a fixed number of steps of HMC for the distribution (11.1), where we ignore the accept-reject step (Salimans et al., 2015). Similarly, we could unroll sampling by Langevin dynamics (Welling & Teh, 2011). Our experiments consider an even more simple randomized predictor: we randomly sample the location for $\bar{y}$ where deterministic gradient-based energy minimization is initialized. Here, we can use the exact code used for end-to-end learning in previous chapters. Such learning seeks to shape the energy function such that it has multiple basins of attraction, with a local optimum at each of the data's modes.

11.4 Experiments

11.4.1 Experimental Setup

We evaluate our methods on synthetic low-dimensional data with distinct cluster structure. We use synthetic data, rather than the real-world problems of other chapters, for a few reasons: (a) by using 2-dimensional outputs, we can easily visualize all learned energy functions and inspect whether they have multiple local minima, (b) to create a regime where baseline end-to-end learning trained to minimize MSE would perform very poorly, and (c) stochastic maximum likelihood will be competitive. Since HMC is very difficult to tune in high dimensions, MLE will be a stronger baseline for low-dimensional problems.

Our data is sampled from a GMM with 2 components:

$$\mathbb{P}(\bar{y}|x) = \frac{1}{2}N(\bar{y}; \mu_1, \sigma^2) + \frac{1}{2}N(\bar{y}; \mu_2, \sigma^2), \tag{11.8}$$

where $N(\bar{y}; \mu, \sigma^2)$ is a the density for a multi-variate Gaussian with mean vector μ and covariance $\sigma^2 I$. We employ$\mu_1 = A_1 x$ and $\mu_2 = A_2 x$. We generate data by sampling a 5-dimensional x from $N(0, 1)$ and then sampling $\bar{y}$. We generate our data set such that for each x we include 25 $(x, \bar{y})$ pairs, where $\bar{y}$ is drawn from $\mathbb{P}(\bar{y}|x)$. This is useful because for a given x we can visualize both the data associated with it and also the conditional energy function $E_x(\bar{y})$.

We choose the data-generating parameters such that samples nearly always occur within $[0, 1]^2$. When performing MCMC-based learning, we explicitly constrain the HMC iterates to this set using the reparametrization method of Sec. 2.7.1. For end-to-end learning, we do not constrain the iterates. At test time, we perform unconstrained optimization over $\mathbb{R}^2$ for all configurations.

We use MLE to refer to a training configuration that performs stochastic maximum likelihood (persistent contrastive divergence). We have found this to be substantially more reliable than the alternative approximate methods discussed in Sec. 11.2. For energies fit using MLE, we form predictions at test time using the same sampling approach used by the randomized unrolled optimizer we train end-to-end. Here, we sample an initial point, and then proceed with deterministic gradient descent. For the MLE energies, we find that the predictive performance is slightly improvded by using backtracking line search during test-time energy minimization.

For MLE training, we first burn in HMC using 25 leapfrog trajectories for each x-y pair. Then, we take 10 additional leapfrog trajectories, where each consists of 5 steps. For each sample, we compute a gradient with respect to the model parameters. Our HMC step size is incremented/decremented on the fly such that proposals are accepted approximately 75% of the time. For end-to-end learning, we unroll for 10 gradient steps and treat our per-step learning rates as trainable parameters.

We use E2E-8 to refer to an application of the end-to-end method of Sec. 11.3 with 8 random samples. We unroll 10 steps of optimization with no momentum and treat our per-step learning rate as a trainable parameter.

11.4.2 SPEN Architectures

We employ two different architectures when fitting our GMM data. The first, which we refer to as a GMM energy hard-codes the functional form of the conditional density of the true data-generating process:

$$E_x(y) = \log \sum_{i=1,2} \frac{1}{2} N(\bar{y}; \mu_i, \sigma_i^2) \tag{11.9}$$

$$\mu_i = A_i x + b_i \tag{11.10}$$

Here, the trainable parameters are weights A_i, biases b_i, and σ_i. This architecture can be seen as a *mixture density network* (Bishop, 1994). Note that we are not actually doing density estimation for a GMM: since sampling is constrained to $[0,1]^L$, the density we learn is actually a GMM density multiplied by an indicator function for whether $\bar{y}$ is in $[0,1]^L$. We found that the learning dynamics for this energy are vulnerable in practice to collapsing modes: if the two mixture components ever coincide, learning will not be able to tease them apart afterwards. When this happens, gradient-based learning will not be able to learn a model that fits the data well. We avoid this by initializing the biases b_i such that the two components are well-separated at the start of learning.

Second, we employ a generic deep network energy, which we refer to as the MLP energy. First, consider an MLP defined just on y.

$$E(y) = a_3^\top g(A_2 g(A_1 y + b_1) + b_2). \tag{11.11}$$

Here, g is a coordinate-wise non-linearity. When training by MLE, we use a ReLU, and when performing end-to-end learning, we use a SoftPlus. Let h be a hyperparameter governing the size of our hidden states. Then, A_1 is a $h \times 2$ matrix, b_1 b_2 are length-h vectors of biases, A_2 is a $h \times h$ matrix, and a_3 is a length-h vector.

We have found that (11.11) works well for un-conditional estimation of data drawn from a GMM. To extend this to be a a conditional energy function, we have a few options. First, we can concatenate y and x as inputs to (11.11). We have found that this does not work particularly well in practice.

Instead, we have found it significantly better to employ a *hypernetwork*, where a subnetwork predicts the weights to be used in the main network. See Ha et al. (2017) for a thorough overview of the history of architectures and learning methods for hypernetworks. We employ an architecture that is identical to (11.11), except that the weights and biases in the first layer of the MLP are a function of x.

$$E_x(y) = a_3^\top g(A_2 g(A_1(x) y + b_1(x)) + b_2), \tag{11.12}$$

$$\tag{11.13}$$

where $A_1(x)$ is a matrix defined by contracting a learned 3-dimensional tensor with x. Similarly, $b_1(x)$ is a vector obtained by contracting a learned 2-dimensional tensor with x.

11.4.3 Results

First, in Fig. 11.2 we plot energy functions learned by MLE. The columns correspond to different values of x. For each x, our dataset contains multiple samples for $\bar{y}$. These samples are red dots in the figures. The color scale of the figures is such that yellow is high energy and dark blue is low energy. In the top row, we fit a GMM energy function and in the bottom row we fit an MLP energy. In general, we found our learned MLP energies to be uni-modal. They generally capture the envelope of the data, but incorrectly assign low energy to the entire region between the two clusters of data.

Next, in Fig. 11.3 we perform the same experiment, but estimate our energy function using end-to-end minimization of the 8-oracle loss. The black lines are trajectories taken by our learned randomized optimizer. For both the GMM and MLP energies, the trajectories reliably go to each of the two modes. Note that the unrolled optimizer never performs exact energy minimization. It seems that it is actually beneficial to perfom inexact minimization, since this provides more diversity in the set of 8 samples, which will lead to a lower 8-oracle loss. Otherwise, all of the samples would be at one of the two energy minima.

In Tab. 11.1 we evaluate our methods in terms of their 8-Oracle performance. We contrast MLE vs. E2E-8 training and MLP vs. GMM energy functions. Remarkably, our best performance is obtained using an MLP energy with E2E-8 training. We struggle to achieve reasonable performance using an MLP energy with MLE training, as the learned energy is typically unimodal (see bottom row of Tab. 11.2). As a result, randomized prediction always predicts a point that is mid-way between the two

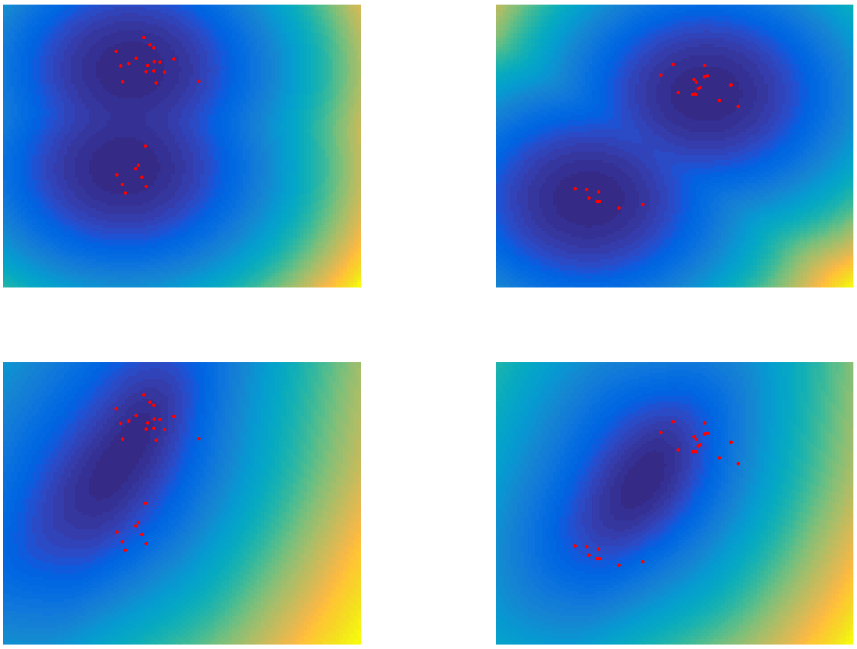


Figure 11.2: Learned energy functions from stochastic MLE training. Each column uses a different value of x, from which we draw multiple samples of y from a conditional GMM. Top row: GMM energy function. Bottom row: MLP energy function.

clusters. In general, we found E2E-8 training substantially easier to tune than HMC-based stochastic MLE. We are unsure why we generally achieve worse performance with the GMM energy vs. the MLP energy. Perhaps the quadratic GMM energy is too steep, and this hinders adequate exploration.

Next, we include one final experiment that highlights the sensitivity of end-to-end learning to hyperparameters of the unrolled optimizer. In Fig. 11.4, we consider the performance of E2E-5 training for the GMM energy function. On the left, we unroll gradient descent with momentum. The learned energy is a low-quality approximation of the true energy. We include only a single example optimization trajectory, as they are in general quite long and complex. Note that the final iterate is remarkably close to one of the cluster centers, despite the fact that the iterate is not remotely close to the energy minimum. This is because the energy minimization procedure is learned

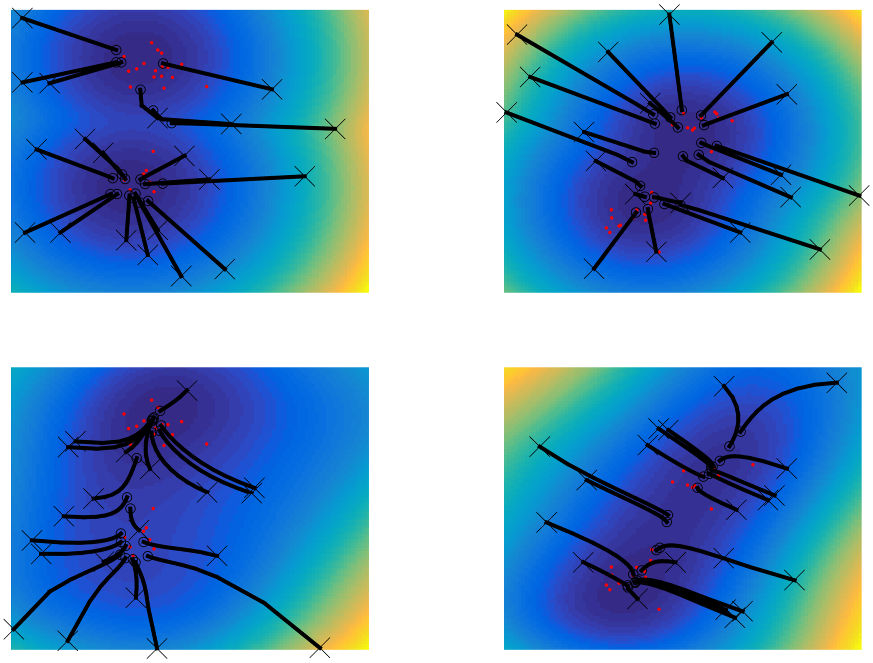


Figure 11.3: Energy functions learned by end-to-end minimization of 8-Oracle loss. Each column uses a different value of x, from which we draw multiple samples of y from a conditional GMM. Top row: GMM energy function. Bottom row: MLP energy function. Black lines are trajectories taken by our learned randomized optimizer.

end-to-end: the model incurs low loss if the final iterate is far from a mode, not if the energy minimum is far from a mode.

On the right, we apply the same procedure, but do not use any momentum in the unrolled optimizer. We find that energy function much more reliably fits the data and that GD trajectories end in local optima. We include this not as experimental evidence suggesting that momentum is bad in general, but to emphasize that learning can be extremely sensitive to hyperparameters.

11.5 Discussion

This chapter presents multiple methods for capturing uncertainty with energy-based prediction. Ideally, we would apply these every time we train a SPEN, not just for applications where we evaluate in terms of K-oracle or where we know in advance

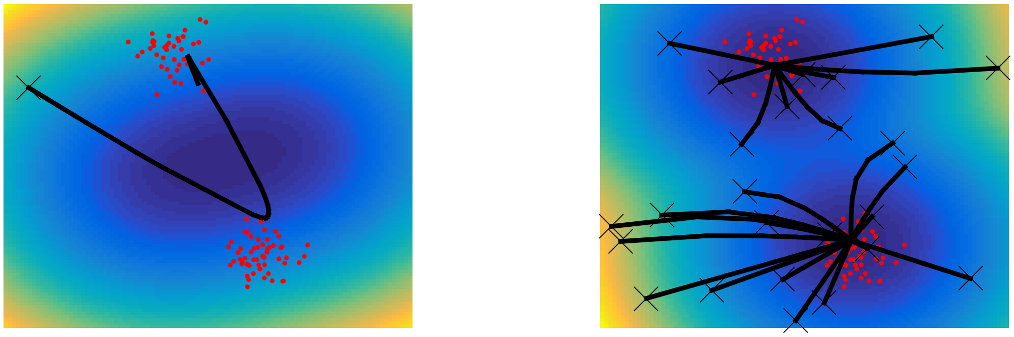


Figure 11.4: E2E-5 training of a GMM energy function. Left: unrolled optimization includes a momentum constant of 0.8. Right: no momentum used in unrolled optimization. Black: trajectory taken by the learned gradient-based optimizer. On the left, a high-quality prediction is made even though the learned energy function does not describe the true energy of the underlying data-generation process.

Training Method	**MLE**	**MLE**	**E2E-8**	**E2E-8**
Architecture	**GMM**	**MLP**	**GMM**	**MLP**
8-Oracle cost	0.007	0.040	0.004	0.002

Table 11.1: 8-Oracle cost using various training methods and energy function architectures.

that the output distribution has well-separated modes. However, it would require additional work to be able to reliably apply these methods to high-dimensional problems. Stochastic MLE is challenging because sampling is difficult for high-dimensional problems, especially when we simultaneously consider many energy functions, corresponding to different x. It may be useful to use better adaptive HMC methods (Girolami & Calderhead, 2011; Hoffman & Gelman, 2014). E2E-K has the undesirable property that the energy does not get penalized when some, but not all, trajectories lead to low quality outputs. We expect performance could be improved by employing reinforcement learning methods for training. See Sec. 7.12 for further discussion.

Some of the visualizations of this chapter reveal important general properties of SPENs. For example, on the left of Fig. 11.2 non-convergent GD with too big of a step size leads to a good final point, but the trajectory is chaotic. It certainly does not correspond to energy minimization. We expect the gradient of the GD trajectory with respect to the learned parameters is more unreliable here than for a GD trajectory that is convergent. This pathology may affect many of the experiments of this thesis, but we are not able to identify this issue by visual inspection. In future work, it may be important to better ensure that the unrolled optimizer is tuned to the shape and scale of the energy function at hand. One solution would be to use a deep network to predict for each x the hyperparameters for an optimizer. These could include step sizes as well as a preconditioner.

CHAPTER 12

SAMPLING-BASED LARGE-MARGIN LEARNING FOR SPENS

It may appear that the end-to-end learning is the gold standard for learning SPENs, since it directly optimizes the performance of gradient-based prediction. It has a few noteworthy drawbacks, however:

1. **Exploration-Exploitation:** The unrolled optimization of end-to-end learning always seeks to minimize the current estimate of the energy function. However, the energy function may not describe the data well, and gradient-based optimization may not efficiently explore high-quality regions of output space. Exploration may be important even for problems without the multi-modal structure explored in the previous chapter.

2. **Memory:** End-to-end learning requires saving the intermediate state for all computations in the forward pass during unrolled optimization, so that we can perform back-propagation. This introduces significant memory overhead, unless we use sophisticated back-propagation tricks that are either problem-structure-specific or introduce additional computation (Sec. 5.4.7).

3. **Speed:** End-to-end learning is an expensive double-loop approach: we must perform repeated steps of optimization with respect to $\bar{y}$ in order to obtain a single gradient of the loss with respect to the energy function's parameters.

This chapter explores the viability of using sampling-based large-margin training with SSVM and SampleRank (Wick et al., 2011) losses to adress these drawbacks.

We choose these methods because they naturally provide for exploration and because sampling can be done with limited memory overhead. An additional motivation is that SampleRank was developed in our research group, and we would like to extend it to new models.

Overall, our primary challenge is to design methods that both learn quickly and achieve high-accuracy predictions. The computational bottleneck for our methods is sampling. This is true for both gradient-based sampling using Hamiltonian Monte Carlo (HMC) and deterministic ‘sampling’ using gradient descent (GD). In response, we employ a method for accelerating learning that uses a single trajectory of GD/HMC to define multiple training examples that we use to update our model’s parameters.

Our experiments provide some successes and some failures. We have one key success: a simple modification to SSVM training that significantly improves its performance, such that it rivals end-to-end learning on some problems. Standard SSVM training defines a parameter gradient using a hinge loss applied to the ground truth and the output of loss-augmented inference. We instead define our loss as the sum of hinge losses at all of the points along the trajectory taken by gradient-based loss-augmented inference.

On the other hand, we find it difficult to tune HMC such that it is both stable and performs adequate exploration. This is a consequence of the fact that we consider high dimensional problems and treat the energy function as a black box, despite the fact that its shape and scale is changing over the course of learning.

In addition, we struggle to achieve a learning method that is both faster than end-to-end learning and achieves comparable performance. In Sec. 12.6, we discuss the qualitative differences between the updates that end-to-end learning and SSVM/SampleRank learning provide to the energy function. One reason that end-to-end learning may be faster is that it provides richer feedback to the energy function, by pushing the gradient of the energy to point in a certain direction. This contrasts with SSVM/SampleRank learning which only pushes up and down on the values of the energy.

12.1 Problem Formulation

In Sec. 5.3 we introduce the SSVM loss:

$$\sum_{\{x_i,\bar{y}_i\}} \max_{\bar{y}} \left[\Delta(\bar{y}_i,\bar{y}) - E_{x_i}(\bar{y}) + E_{x_i}(\bar{y}_i)\right]_+ . \tag{12.1}$$

Here, $\Delta(\bar{y}_i,\bar{y})$ is a user-defined task-specific cost function.

We can interpret $\left[\Delta(\bar{y}_i,\bar{y}) - E_{x_i}(\bar{y}) + E_{x_i}(\bar{y}_i)\right]_+$ as a convex relaxation of the constraint $E_{x_i}(\bar{y}) > E_{x_i}(\bar{y}_i) + \Delta(\bar{y}_i,\bar{y})$, i.e., that the energy of configuration $\bar{y}$ is greater than the energy of the ground truth configuration $\bar{y}_i$ by a margin $\Delta(\bar{y}_i,\bar{y})$.

If all of the constraints are satisfiable, then minimizing the maximum constraint violation (12.1) is equivalent to minimizing the average constraint violation:

$$\sum_{\{x_i,\bar{y}_i\}} \frac{1}{|\mathcal{Y}|} \sum_{\bar{y}} \left[\Delta(\bar{y}_i,\bar{y}) - E_{x_i}(\bar{y}) + E_{x_i}(\bar{y}_i)\right]_+ . \tag{12.2}$$

Of course, if the constraints are not satisfiable, then these formulations will have different optima. Here, $\bar{y}$ is continuous for SPENs, and thus we should employ an integral rather than a sum. We use summation notation going forward, however, for simplicity. Here, $|\mathcal{Y}|$ is a normalizing constant equal to the size of the set of feasible $\bar{y}$.

The principal advantage of the max-violation formulation (12.1) is that evaluating a subgradient of the loss can be performed by solving an inner optimization problem, whereas computing a subgradient of the averge-violation formulation (12.2) requires a potentially-intractable sum. On the other hand, solving the inner problem for (12.1) may be difficult, especially for non-convex energy functions, while the average-violation formulation can be minimized using inexpensive stochastic approximation. A stochastic sub gradient of (12.2) can be obtained simply by sampling a configuration $\bar{y}$ uniformly at random.

Unfortunately, uniform sampling may be extremely inefficient for learning, since the subgradient is 0 at any point $\bar{y}$ where the margin constraint is satisfied. Alternatively, we can change the actual objective function (12.2) to be an expectation with respect to a non-uniform distribution:

$$\sum_{\{x_i, \bar{y}_i\}} \mathbb{E}_{\bar{y} \sim \mathbb{P}_L(\bar{y}|x_i)} \left[\Delta(\bar{y}_i, \bar{y}) - E_{x_i}(\bar{y}) + E_{x_i}(\bar{y}_i)\right]_+ . \tag{12.3}$$

Here, $\mathbb{P}_L(\bar{y}|x_i)$ is our non-uniform sampling distribution, and (12.3) can be minimized easily using stochastic subgradient descent. We assume that $\mathbb{P}_L(\bar{y}|x_i)$ does not depend on the parameters of the model that we are training. Otherwise, differentiating (12.3) would be complicated. In many of the approaches below, $\mathbb{P}_L(\bar{y}|x_i)$ is defined in terms of the current energy function, however. We simply ignore this dependence when deriving our gradients.

Changing the loss function to be with respect to some $\mathbb{P}_L(\bar{y}|x_i)$ may actually be advantageous. For example, in many situations the constraints will not actually be satisfiable, given the limited capacity of the network parameterizing the energy function. Therefore, by constructing $\mathbb{P}_L(\bar{y}|x_i)$, the user specifies which constraints are most important. For example, it may be useful to focus on satisfying constraints in the region that would be explored by gradient-based prediction.

We can recover traditional SSVM training if we employ

$$\mathbb{P}_L(\bar{y}|x) = \mathrm{I}\left[\bar{y} = \arg\min_{\bar{y}'} \left(\Delta(\bar{y}_i, \bar{y}') - E_{x_i}(\bar{y}')\right)\right] \tag{12.4}$$

Here, $\mathrm{I}[\cdot]$ is the indicator function for an event. In cases where the argmin function returns a set, we would replace the indicator function with the uniform distribution over this set.

Alternatively, we can sample from the Gibbs distribution:

$$\mathbb{P}_L(\bar{y}|x) \propto \exp(\frac{-1}{\tau} E_x(\bar{y})), \tag{12.5}$$

where τ is a temperature parameter. By decreasing τ, we encourage our samples to concentrate around energy minima. Note that (12.4) is the zero-temperature limit of the loss-augmented Gibbs distribution

$$\mathbb{P}_L(\bar{y}|x) \propto \exp(\frac{-1}{\tau}\left(E_x(\bar{y}) + \Delta(\bar{y}_i, \bar{y}')\right)). \tag{12.6}$$

Sampling from this distribution at a non-zero temperature may help discover margin violations when employing non-convex energies. Such a distribution was first employed for softmax-margin training (Gimpel & Smith, 2010), which focused on problems where sampling is not necessary, since exact marginal inference is tractable.

Besides varying the sampling distribution, we can also alter the overall set of constraints we seek to enforce. The SampleRank loss of Wick et al. (2011) is similar to the SSVM loss, but it seeks to enforce margin constraints between all pairs of configurations, rather than between a single configuration and the ground truth. See Sec. 12.2.2 for more details.

12.2 Efficient Learning

A principal goal of this chapter is to obtain learning procedures that are more computationally efficient than end-to-end training. One must be careful when designing sampling-based learning methods, however, as the cost of collecting a sample may rival the cost of the inner energy minimization step in end-to-end learning.

SPENs are defined over continuous inputs, and thus we must sample from continuous densities. Here, it is natural to sample using Hamiltonian Monte Carlo (HMC). Unfortunately, the complexity of HMC sampling from a distribution like (12.5) is similar to the complexity of gradient-based energy maximization of the distribution's log-density. This is not surprising, as both techniques use gradient methods to traverse to high-probability, i.e., low-energy, regions. Overall, sampling in SPENs is fundamentally different than sampling in popular instances of factor graphs, where the conditional independence structure provides for efficient MCMC steps that do not need to recompute the energy of $\bar{y}$ from scratch.

Note that HMC can be done with limited memory overhead, by performing all updates to the sampled variables in place. This is analogous to how test-time energy minimization by GD can be done with less memory overhead than an unrolled GD predictor that supports back-propagation (Sec. 5.4.7).

To alleviate the the computational limitation of continuous sampling, we propose gathering multiple correlated samples $\bar{y}^{(1)}, \ldots, \bar{y}^{(N)}$, such as from consecutive steps of MCMC. Then, we define a margin-based loss at each of these points. This is similar to how stochastic MLE (Sec. 11.2) constructs multiple parameter gradients using a single MCMC chain.

Overall, our learning method has the following modules:

1. **Sec. 12.2.1:** A procedure for collecting a set of samples $\{\bar{y}^{(1)}, \ldots, \bar{y}^{(N)}\}$ from some distribution $\mathbb{P}_L(\bar{y}|x_i)$.

2. **Sec. 12.2.2:** A loss function that takes individual samples, or pairs of samples, and computes the value and gradient of a margin-based training loss.

3. **Sec. 12.2.1:** A method for aggregating the gradients of the loss at each of the samples and updating the parameters of the energy function.

12.2.1 Gathering Trajectories of Samples

As identified in the previous section, we can decrease the cost of training by obtaining a set $\{\bar{y}^{(1)}, \ldots, \bar{y}^{(N)}\}$ of correlated samples from our distribution $\mathbb{P}_L(\bar{y}|x_i)$ at once. To do this, we propose constructing a trajectory, e.g., an MCMC chain, in the space of $\bar{y}$, and returning N evenly-spaced points along this trajectory.

For the determinstic distribution (12.4) defined by loss-augmented energy minimization, we define the trajectory as the path taken by gradient-based optimization. Similarly, for HMC-based sampling from distributions such as (12.5), we use a single trajectory of leapfrog integration. See Sec. 2.7 for details.

Finally, we can perform *loss-rewarded sampling.* This employs a distribution obtained by flipping the sign of the cost term in (12.6). In SSVM training, loss-augmented inference rewards bad predictions, since it is used to discover margin violations. With loss-rewarded sampling, we reward good predictions. This can be seen as a form of imitation learning, where training employs sampling that is guided by information about the ground truth that is unavailable at test time. As with many applications of imitation learning, it is natural to use this extra guidance only at the beginning of learning.

Overall, we do not consider alternative methods for inexpensive sampling, such as a random walk, as it is unclear that the samples will ever traverse to high-quality regions for our high-dimensional problems of interest.

For discrete prediction problems, we could have performed MCMC on the discrete representation directly, and avoided any convex relaxation. However, it would be very difficult to achieve efficient MCMC for such a problem. Gibbs sampling is intractable, as multiple forward evaluations of the energy network would be necessary to obtain the conditional distribution used to flip a single output variable. HMC is an instance of Metropolis-Hastings sampling, where the proposals are produced by simulating Hamiltonian dynamics. We could instead perform Metropolis-Hastings with a proposal distribution that is cheap to evaluate and works for discrete variables. However, it would be difficult to design a proposal distribution such that the acceptance ratio is reasonable for high-dimensional problems. In addition, computing each acceptance ratio would still require a forward evaluation of the energy network. This contrasts with a sparse factor graph, where we can leverage the factorization structure to compute the acceptance ratio by only evaluating a few local terms.

12.2.2 Loss Functions

We define *SampleSVM* training as the proposed sampling-based method for minimizing (12.3).

We also consider the SampleRank loss, which defines margin constraints on pairs of configurations (Wick et al., 2011). For a given x_i, let $\bar{y}_0$ and $\bar{y}_1$ be arbitrary configurations and let $\bar{y}_i$ be the ground truth associated with x_i. We define the asymmetric discrepancy function

$$D_i(\bar{y}_0, \bar{y}_1) = \Delta(\bar{y}_0, \bar{y}_i) - \Delta(\bar{y}_1, \bar{y}_i). \tag{12.7}$$

This measures how much closer $\bar{y}_0$ is to the ground truth than $\bar{y}_1$. SampleRank enforces the constraint that the energy difference between $\bar{y}_0$ and $\bar{y}_1$ is greater than their discrepancy. Given two samples $\bar{y}_0$ and $\bar{y}_1$, we assume without loss of generality that $\Delta(\bar{y}_0, \bar{y}_i) \geq \Delta(\bar{y}_1, \bar{y}_i)$. In other words, $\bar{y}_1$ is a more accurate output than $\bar{y}_0$. With this, we define the max-violation formulation of the SampleRank loss as:

$$\sum_{\{x_i, \bar{y}_i\}} \max_{\bar{y}_0, \bar{y}_1} \left[D_i(\bar{y}_0, \bar{y}_1) + E_{x_i}(\bar{y}_1) - E_{x_i}(\bar{y}_0) \right]_+ . \tag{12.8}$$

Similarly, the non-uniform average-violation version of SampleRank is:

$$\sum_{\{x_i, \bar{y}_i\}} \mathbb{E}_{\bar{y}_0, \bar{y}_1 \sim \mathbb{P}_L(\bar{y}|x_i)} \left[D_i(\bar{y}_0, \bar{y}_1) + E_{x_i}(\bar{y}_1) - E_{x_i}(\bar{y}_0) \right]_+ . \tag{12.9}$$

Note that the SampleRank authors derive their method in terms of the objective function (12.8), rather than (12.9). However, their objective is minimized with stochastic approximation, where $\bar{y}_0$ and $\bar{y}_1$ are sampled from some distribution. This does not make any sense. The authors should have introduced an objective function of the form (12.9) that explicitly identifies the distribution from which constraints are sampled.

SampleRank was originally developed for large, sparse factor graphs, where the conditional independence structure of the model allows for efficient MCMC sampling. In Wick et al. (2011), $\bar{y}_0$ and $\bar{y}_1$ are consecutive states of a Markov chain that only differ in the settings of a few variables. Given the model's factorization structure, evaluation of $D_i(\bar{y}_0, \bar{y}_1)$, and evaluation of the gradient of the loss with respect to the energy function's parameters requires information only from a small neighborhood around variables that have changed. This is not available when training SPENs with SampleRank, however, as the energy function is a black-box object with no known factorization structure.

To maintain the similarity between our use of SampleRank and how it is employed by Wick et al. (2011), we choose $\bar{y}_0$ and $\bar{y}_1$ to be consecutive samples from a trajectory obtained using the methods of the previous section. If our trajectory is of length N, our loss is defined on $N - 1$ pairs. We also propose a novel approach, SampleRank-G that introduces N additional pairs, where each new pair consists of a sample from the trajectory and the ground truth. Note that when comparing a point to the ground truth, the SampleRank loss is identical to the SampleSVM loss. SampleRank-G would have been inefficient for Wick et al. (2011), since much of the efficiency of the method relies on $\bar{y}_0$ and $\bar{y}_1$ being consecutive MCMC states. For SPENs, however, none of these tricks are available, so comparing to the ground truth has the same cost as comparing consecutive samples.

Finally, note that in Wick et al. (2011) the authors do not use the discrepancy function during sampling. This differs significantly from SSVM training, where loss-augmented inference solves an optimization problem with an objective that is shifted by $\Delta(\cdot, \cdot)$. For SampleRank training of SPENs, we require that the discrepancy function is defined on the domain of for the continuous variable $\bar{y}$. We do not require that it is differentiable, however.

12.2.3 Interleaving Inference and Learning

In Sec. 12.2, we advocate for using trajectories of correlated samples as a way to circumvent the computational cost of sampling. This provides N points that can be used to define a large-margin loss. For SampleSVM, we compute the gradient of the loss with respect to the parameters of the energy function at N points. For SampleRank, we have $N - 1$ gradients, and for SampleRank-G we have $2N - 1$ gradients.

We propose two general methods for updating the parameters of our energy function. The *Avg-Grad* method obtains a single parameter gradient by averaging over all gradients from the trajectory. In contrast, the *Interleave-Grad* method updates the parameters on the fly while sampling a trajectory, like we do for stochastic MLE (Sec. 11.2). As soon as we have a new $\bar{y}^{(t)}$ available, we compute the gradient of the relevant terms for our large-margin loss and update the parameters with a small learning rate. Here, HMC sampling, or gradient-based energy minimization, is interacting with an evolving energy function. This interleaving approach was employed in Wick et al. (2011).

Avg-Grad may accelerate training because we obtain a lower-variance stochastic gradient by averaging. Interleave-Grad may accelerate training because we perform more parameter updates. We consider both in our experiments.

As described in Def. 1.4.1, the dependence of our energy network on x comes by way of features $F(x)$. When performing sampling with respect to $\bar{y}$, $F(x)$ only needs to be computed once. However, when employing Interleave-Grad, we need to recompute the features at every step, since the parameters of $F(x)$ have been updated. Similarly, for loss functions that evaluate the energy at the ground truth, we need to recompute the ground truth energy at every step. Depending on complexity of the architectures of the feature and energy networks, this extra computational overhead may diminish any speed gains available from interleaving inference and learning for SPENs.

12.3 Details

- One of our primary goals is to achieve faster learning than end-to-end learning. Rather than measuring accuracy vs. wall-clock time, we measure accuracy vs. the number of training examples considered. This avoids various implementation-level details that would confound the comparison. Using the number of training examples considered is different than the number of gradient steps taken, since for Interleave-Grad we take N gradient steps for a given training example.

- We always accept HMC proposals, rather than evaluating an acceptance ratio. Acceptance probabilities less than one occur because approximate leapfrog integration only appromiates the true Hamiltonian dynamics. This approximation is worse in high dimensions and for large step sizes. We have found it extremely challenging to achieve reasonable acceptance probabilities for real-world problems that are higher-dimensional than the synthetic GMM problems in the previous section. An alternative approach would have been to use an extremely small step size, but this would require using a large number of computationally-expensive leapfrog steps.

- Before performing learning with HMC-based trajectories, we first perform learning with GD trajectories and select a step size that yields the best performance. Let η_G be the step size that works well for GD and let η_τ be the step size we use for HMC sampling at temperature τ. We set $\eta_\tau = \sqrt{\tau \eta_G}$, so that the characteristic scale of the per-sep updates to $\bar{y}$ for HMC is equal to that of GD.

- When sampling on the probability simplex using the method of Sec. 2.7.1 we include the Jacobian term in the energy for sampling, but do not employ it at test time. This is because we seek to find the most likely normalized iterate, not the most likely un-normalized logit.

- It is extremely important to appropriately set the scales of various terms. Consider a problem where $E_x(\bar{y})$ is a composed of a sum L terms. This occurs, for example, in our image denoising application, where the final layer of both the local and global energy terms is a summation of per-pixel energy values. Here, the total energy should be set to the sum of these per-pixel energies, rather than the average, so that the general per-pixel dynamics of test-time gradient descent will not be effected by the size of the image. With this, it is important that any cost function Δ used with our large-margin methods is also computed as a sum, instead of an average, such that the energy and the cost are on the same scale. Third, the overall margin-based loss function, should be divided by L. This ensures that the scale of the gradient of the loss with respect to the parameters will not depend on L.

12.4 Image Denoising Experiments

First, we consider the performance of our methods for image denoising on the 7-Scenes dataset (Newcombe et al., 2011), using the same SPEN architectures employed in the experiments of Chapter 10. The goal of these experiments is to provide controlled experiments that demonstrate the impact of the various design decisions discussed for sampling-based large-margin learning. Overall, we have found that different tasks may require very different settings of these hyperparameters. When prototyping new applications with SPENs, practitioners should explore the questions put forth in the following experiments on their own data. In general, these learning methods are substantially more brittle than end-to-end learning, and need to be tuned carefully.

Denoising presents a few key challenges for using our methods. First, noise-less images lay on a low-dimensional manifold, and the performance of our learning methods may be sensitive to our ability to sample on or near this manifold. Second, as our experiments in Sec. 10.3 suggest, the particular parametrization of our energy function (which corresponds to MAP inference) has an undesirable characteristic: a few steps of energy minimization can yield high-quality predictions, but taking many steps may yield over-smoothed outputs. This presents a conceptual hurdle, as it is unclear whether we should perform full energy minimization in the inner loop of large-margin learning, or instead employ truncated optimization at train time. Overall, we have found that the latter approach is signficantly better. However, this means that some of the conclusions of this section may not generalize to other problems where full energy minimization is best.

12.4.1 SampleSVM training of Field-of-Experts Model

First, we consider learning the ℓ_1 field-of-experts prior introduced in (10.2). This is a useful application for evaluating SSVM-based learning methods, since the energy function is convex with respect to $\bar{y}$. We consider using either HMC or GD-based trajectories, where both methods employ the loss-augmented energy function (12.6).

We have found it most effective to use the same number of gradient steps at test time as employed when collecting trajectories at train time. All of our experiments use 5 steps. Therefore, if we perform Avg-Grad with $N = 5$, this means that each sample is separated by a single gradient step.

In Table 12.1, we compare the performance of using $N = 1$ vs. $N = 5$ with either Avg-Grad or Interleave-Grad updates. Note that using $N = 1$ with GD trajectories is equivalent to standard SSVM learning (except for the fact that we perform truncated optimization with respect to $\bar{y}$). We find that $N = 5$ with Avg-Grad is best, and performs at the same level as our best field-of-experts model trained using end-to-end learning (see Tab. 10.1).

These results are further investigated in the left figure of Fig. 12.1, where we contrast the curves of the training objective and test accuracy over the course of training for the $N = 1$ and $N = 5$ w/ AvgGrad configurations. We find that the gradient averaging provides faster, more stable learning. We expect this is due to two reasons: (a) averaging produces lower-variance stochastic gradients, and (b) the effective size of the gradients decreases over time for Avg-Grad. This is because the Avg-Grad gradient is the average of gradients computed along the trajectory of loss-augmented inference. As training progresses and the model gets better, points at the end of the trajectory may become more likely to violate the margin constraints than points at the beginning. Therefore, the number of terms in the average that contribute non-zero gradients may get smaller. Perhaps we could close the gap between these two configurations if we carefully tuned a learning rate decay schedule for $N = 1$ training.

In the right figure of Fig. 12.1, we consider how test-time performance increases over time for $N = 1$ and $N = 5$ with AvgGrad training, and compare this to end-to-end training. Unfortunately, we find that end-to-end training learns faster (in terms of the number of training examples considered) and is more stable. Using $N = 5$ is superior to $N = 1$, though. Next, in Tab. 12.2 we compare training with trajectories collected with GD vs. HMC. We struggle overall to achieve high performance using

HMC-based trajectories, since the scale and shape of the energy changes dramatically over the course of learning. In future work, it may be useful to pursue methods that adaptively tune HMC on the fly (Girolami & Calderhead, 2011; Hoffman & Gelman, 2014).

Finally, we consider whether Interleave-Grad can be used to provide a useful speedup for training. Due to the noise introduced by updating the energy function on the fly, it is necessary to use a smaller learning rate for Interleave-Grad than for Avg-Grad. In Fig. 12.2, we contrast the performance of Avg-Grad with a learning rate of 10e-4 with Interleave-Grad with a learning rate of either 0.5e-4 or 2.5e-4. We find that using 0.5e-4 with Interleave-Grad produces performance and training stability that rivals that of Avg-Grad. However, in order to achieve this we must use a learning rate that is so small that any potential speedup to learning from updating the energy on the fly is eliminated.

Num Samples	**1**	**5**	**5**
Grad Accumulation	n/a	Avg-Grad	Interleave-Grad
PSNR	37.1	37.7	37.2

Table 12.1: Comparing the performance of various SampleSVM training configurations with trajectories collected using deterministic loss-augmented inference. For comparison, we achieve 37.7 using end-to-end training.

Sampling	**GD**	**HMC**-0.001	**HMC**-0.01
PSNR	37.6	36.5	33.5

Table 12.2: Performance of $N = 5$ Avg-Grad training using different methods to collect trajectories. They also employ the same loss-augmented energy function. HMC-τ employs a single trajectory of leapfrog integration at a temperature of τ.

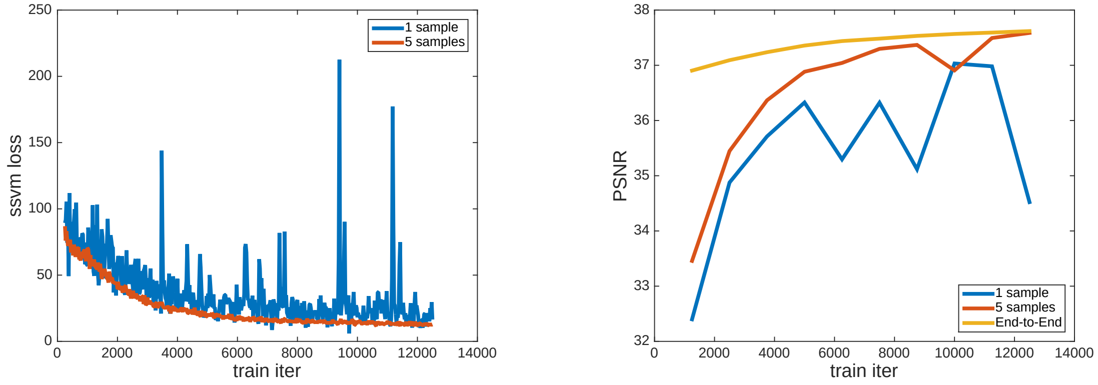


Figure 12.1: Left: train loss for SampleSVM training using $N = 1$ vs. $N = 5$. Right: Test PSNR over the course of learning for SampleSVM training with $N = 1$ and $N = 5$ vs. end-to-end learning.

Sampling	**GD**	**GD**	**HMC**-0.1	**HMC**-0.1
Grad Accumulation	Avg-Grad	Interleave-Grad	Avg-Grad	Interleave-Grad
PSNR	36.8	36.0	35.9	36.0

Table 12.3: Performance of SampleRank-G using a field-of-experts model with various methods for collecting trajectories and accumulating gradients.

12.4.2 SampleRank training of Field-of-Experts Model

Next, we evaluate the performance of SampleRank-based training on the denoising data. In Tab. 12.3, we vary the method for collecting trajectories (GD vs. HMC) and the method for accumulating gradients (Avg-Grad vs. Interleave Grad). As before, we find that using GD-based trajectories with Avg-Grad performs best.

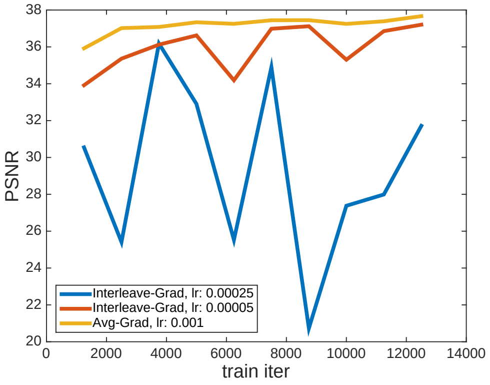


Figure 12.2: Test PSNR for $N = 5$ the course of training for Avg-Grad and Interleave-Grad with two different learning rates. In order to achieve stable learning with Interleave-Grad, we must use such a small learning rate that any potential speed advantages vs. Avg-Grad are eliminated.

Note these results employ the Samplerank-G loss, as defined in Sec. 12.2.2, and that we could not get competitive results using the original version of SampleRank presented in Wick et al. (2011), where no margin constraints between samples and the ground truth are imposed. Our best performance was 35.4. Whenever a pairwise margin constraint is violated between samples, SampleRank and SampleRank-G push down the energy of one of the samples. Unfortunately, this sample may not be on the image manifold. With SampleRank-G, we also push down the energy of the ground truth, which is on the image manifold, and this may be crucial for performance. In fact, if we ignore all margin constraints besides between points and the ground truth, we recover the SampleSVM method explored in the previous section, which outperforms SampleRank-G.

In the left figure of Fig. 12.3 we compare the loss for SampleRank-G vs. SampleRank over the course of training. Overall, the loss for SampleRank is much noisier. When SampleRank has low loss, this is not because it is necessarily a high-quality model. Since the samples are collected by traversing the current energy function, there is no guarantee that the sampling explores high-quality outputs. Even in these low-quality regions, the margin constraints between pairs of samples could be satisfied.

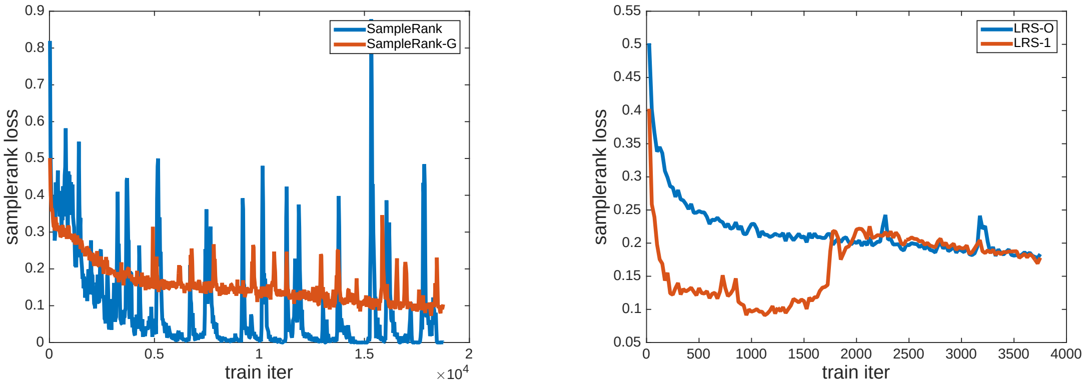


Figure 12.3: Left: Loss curves for SampleRank vs. SampleRank-G. Right: loss curves for SampleRank-G training when using loss-rewarded sampling or not.

Finally, in the right figure of Fig. 12.3 we consider the impact of using loss-rewarded sampling, which we introduce at the end of Sec. 12.2.1. Given ground truth $\bar{y}_i$, at training iteration t we employ the energy function $-w_t\Delta(\bar{y}_i, \bar{y}) - E_{x_i}(\bar{y})$, where $w_t = \max(0, 1 - \frac{t}{T_H})$. Here, T_H is a time horizon over which we decay w_t. Our experiment uses $T_H = 2000$. We find that the SampleRank-G loss decreases more quickly if we use loss-rewarded sampling than if we do not. However, this reduction in loss seems to come because of how we bias our samples, not because we learn a good model quicker. It provides no better test-set performance overall, or in the early iterations of training.

12.4.3 Sampling-Based Training of DeepPrior Model

Now, we use our methods to train the DeepPrior SPEN introduced in Sec. 10.3. Here, the energy function is non-convex with respect to $\bar{y}$.

In Tab. 12.4, we apply SampleSVM learning. We vary the technique used for gathering trajectories, along with the number of samples N per trajectory to use. GD with $N = 1$ corresponds to traditional SSVM learning. We find that using $N = 5$ with GD performs significantly better than $N = 1$, achieving performance comparable to end-to-end training. We were not able to achieve reasonable performance using HMC-based trajectories except at a low temperature. For deep, non-convex energies, it may be particularly important to tune the HMC hyperparameters on the fly.

Num Samples	1	5	5	5	n/a
Sampling	**GD**	**GD**	**HMC**-0.001	**HMC**-0.01	**End-to-End**
PSNR	38.7	40.2	40.1	34.0	40.4

Table 12.4: Performance of SampleSVM learning DeepPrior.

In Tab. 12.5 we consider the performance of Samplerank-G. As with the field-of-experts model, samplerank does not perform as well as SampleSVM learning. We also struggle to achieve high-quality models when using HMC-based trajectories, except if we use an extremely low temperature for sampling.

Sampling	**GD**	**HMC**-0.001	**HMC**-0.01	**End-to-End**
PSNR	38.5	38.2	36.5	40.4

Table 12.5: Performance of SampleRank-G using a DeepPrior model with various methods for collecting trajectories.

12.5 Image Segmentation Experiments

Next, we apply our sampling-based large-margin learning methods to the Weizmann horses dataset (Borenstein & Ullman, 2008), using the SPEN architectures introduced in Chapter 8. We evaluate development set accuracy using a SPEN with a kernel width of 5 in the energy network. See Sec. 8.2.2 for a discussion of our architecture. We employ the squared-loss between $\bar{y}$ and a one-hot representation for the ground truth $\bar{y}^*$ as Δ.

Num Samples	5	5	5	n/a
Sampling	**GD**	**HMC**-0.01	**HMC**-0.1	**End-to-End**
Accuracy	91.6	93.2	92.2	93.1

Table 12.6: Performance of SampleSVM on the Weizmann horses data.

In Tab. 12.6, we consider the performance of SampleSVM learning. Here, we find that we can achieve performance that is comparable to to end-to-end learning using HMC-based trajectories. It is unclear why sampling-based learning works well for this problem and not for denoising in the previous section. We expect part of the reason is that the sampling for denoising does not reliably sample on or near the image manifold. However, for the horses data, there is no clear manifold structure for $\bar{y}$.

Next, in Tab. 12.7 we consider the performance of SampleRank on the same data. The table contrasts multiple methods for collecting trajectories. Overall, we find that HMC generally performs better, and that we can match the performance of end-to-end learning. Note that the best performance we can achieve with SampleRank-G is 90.7. We are unsure why this is true.

Sampling	**GD**	**HMC**-0.01	**HMC**-0.1	**End-to-End**
Accuracy	91.7	93.2	92.2	93.1

Table 12.7: Performance of SampleRank on the Weizmann horses data.

12.6 Discussion

This chapter explores the viability of using sampling-based large-margin methods for fast, accurate training SPENs. We introduce a collection of methods that make certain configurations of SampleSSVM and SampleRank training achieve similar performance to end-to-end learning. However, we are not able to achieve our goals of faster training or better performance. Faster training is difficult because sampling continuous variables with arbitrary deep energy functions requires similar computation as gradient-based energy minimization. We experiment with a method to interleave sampling and updates to the model parameters, but this is only stable when we use a small enough step size that it is slower than end-to-end learning. Our goal of achieving better accuracy by using sampling to do better exploration is also not attained. This is because adaptively tuning HMC on the fly to perform well for high-dimensional black-box energy functions is very difficult.

Ironically, the easiest-to-implement and most-reliable approach does not require random sampling at all: we simply define our SSVM loss not just between the ground truth and the solution to loss-augmented inference, but between the ground truth and every point along the trajectory taken by gradient-based loss-augmented inference. This is easy to implement and may of future interest for SPEN applications.

Above, we mention computational challenges for getting SSVM and SampleRank learning to be faster than end-to-end learning. The remainder of this section is devoted to discussing a key qualitative difference between these training methods that might also dictate how fast they can train. Specifically, SampleSVM and SampleRank learning may require lots of gradient steps because the quality of the feedback that the training loss provides is lower-quality than that of end-to-end learning.

First, we consider SSVM learning. Let $\hat{y}$ be the output of loss-augmented inference, let y^* be the ground truth, let $\Delta(\hat{y}, y^*)$ be a cost function, and let w be the trainable parameters of the energy function. We assume that a margin violation occurs between $\hat{y}$ and y^*, ie $E(\hat{y}) < E(y^*) + \Delta(\hat{y}, y^*)$. Otherwise, the gradient of the loss with respect to w would be zero. The parameter gradient takes the simple form of pushing the energy up for the incorrect prediction $\hat{y}$ and pushing the energy down for the ground truth:

$$\frac{dL}{dw} = \left.\frac{dE(y)}{dw}\right|_{y=\hat{y}} - \left.\frac{dE(y)}{dw}\right|_{y=y^*}. \tag{12.10}$$

Consider the special case that the energy function is linear in sufficient statistics $S(y)$ (Sec. 2.3.2), as it would be in a CRF. Here, we have $E(y) = w^\top S(y)$ and $\frac{dL}{dw} = S(\hat{y}) - S(y^*)$. Any term that is identical between $\hat{y}$ and y^* will cancel out, and thus the parameter gradient will focus on the specific mistakes made by $\hat{y}$.

However, this property does not hold for arbitrary energy functions $E(\cdot)$. Consequently, the parameter gradient will not explicitly target which aspects of the structured output $\hat{y}$ need to be fixed. With such limited feedback, large-margin learning for SPENs may require many parameter updates.

Next, we consider end-to-end learning. For simplicity, we assume that our cost is the squared error $L(\hat{y}, y^*) = \frac{1}{2}\|\hat{y} - y^*\|^2$ and that our unrolled predictor performs a single step of gradient descent:

$$\hat{y} = y_0 - \eta \nabla E(y_0). \tag{12.11}$$

Define g as $\nabla E(y_0)$. With this, we have:

$$\frac{dL}{dw} = \frac{dL}{d\hat{y}} \frac{d\hat{y}}{dg} \frac{d^2 E(y)}{dydw} \tag{12.12}$$

$$\frac{dL}{dw} = -\eta \frac{dL}{d\hat{y}} \frac{d^2 E(y)}{dydw} \tag{12.13}$$

$$\frac{dL}{dw} = -\eta(\hat{y} - y^*) \left. \frac{d^2 E(y)}{dydw} \right|_{y=\hat{y}} \tag{12.14}$$

Here, $\hat{y} - y^*$ is the displacement vector between the prediction and the ground truth. As a result, the parameter gradient (12.14) updates the energy function such that its gradient with respect to y is more in line with this displacement. This provides substantially different feedback to the energy function than SSVM learning. Here, we directly shape the gradient field of the energy such that it points in the right direction, whereas SSVM learning only pushes up and down on values of the nergy. This may be very important for fast learning and for supporting high-quality gradient-based prediction.

Finally, note that end-to-end learning never evaluates the energy function at the ground truth. It instead evaluates a loss function that compares the output of energy minimization to the ground truth and uses this for back-propagation. This seems like a wasted opportunity, as know where the ground truth is, and could directly ensure that it has low energy. On the other hand, SSVM learning does exactly this. In future work, it may important to design a learning method for SPENs that has the benefits of both approaches.

CHAPTER 13
CONCLUSION

13.1 Summary of Contributions

This thesis explores a wide variety of training methods and applications for SPENs. Our goal has been to advocate for the general usefulness of deep energy-based models and to expose important design decisions necessary to get SPENs to work well in practice. Hopefully, SPENs will become a useful contribution to the dialog on how to best incorporate deep networks in structured prediction.

We depart from many previous structured prediction works by treating our energy function as a black-box that only provides forward and back-propagation. This allows us to employ a broad range of energy functions and to use general-purpose learning and prediction code. In addition, while our prediction and learning methods interact with the energy function as a black-box, the practitioner is free to design the functional form of the energy to capture prior knowledge about the problem domain.

We select the applications for our experiments primarily because they allow us to perform controlled experiments that isolate the effects of various factors on SPEN performance. These include the loss function used for training the energy, the optimization algorithm used for energy minimization, the functional form of the energy, the method used for rounding from a continuous to discrete prediction, and the convexity of the energy. These reveal informative details for designing new SPEN applications.

13.2 Future Work

SPENs are a novel framework for structured prediction that present a variety of desirable properties. On the other hand, their speed and accuracy need to be improved before they can be deployed in large-scale applications. The first section below emphasizes a few details of our work that are generally useful and should be maintained in further research. After that, we provide a list of specific suggestions for improving SPENs.

13.2.1 SPENs Details to Retain for Future Work

End-to-end SPEN learning (Sec. 5.4), i.e., unrolling a particular energy minimization algorithm and training it by gradient descent, is very effective in practice. Not only can it provide high-quality models, but it is substantially more user-friendly, since learning directly returns a prediction algorithm. Otherwise, we have to separately tune an optimization procedure to be used at test time. This has the added benefit that it is easy to learn an energy function such that it can be optimized quickly at test time.

The general principal of using deep network to do representation learning for the outputs of a structured prediction problem is good. On the other hand, it is difficult to fully utilize the capability of deep global energy functions, since labeled datasets for structured prediction are often quite small, and thus SPENs are vulnerable to overfitting. The architecture in our SRL experiments (Sec. 9.1) hard-codes prior knowledge about constraints the data must satisfy and about important interactions among components of $\bar{y}$. Such an approach may be useful in many applications going forward.

It can be useful to perform stage-wise training of different parts of a SPEN. Our decomposition of the energy into local and global terms (Sec. 3.2.3) is useful because it enables us to pre-train our features $F(x)$ using a feed-forward predictor defined by the local terms. By explicitly defining $F(x)$, instead of absorbing it into the energy network, we are also able to achieve fast energy minimization with respect to y by caching the features.

Our method for capturing multi-modal output distributions using end-to-end training of a randomized predictor (Sec. 11.3) is easier to implement and tune than methods that perform multi-modal density estimation. Being able to capture uncertainty in predictions is very useful in a variety of applications. By defining a randomized predictor simply by selecting a random initial point, we also avoid introducing any new hyperparameters.

Finally, we have found it very helpful to inspect the various diagnostics employed in Sec. 8.3 for tuning SPENs. Plotting the curves for test-time energy minimization helps us identify problems where the energy minimization is non-convergent or low-quality. Plotting the norm of the gradient over the course of learning helps us understand if we should be clipping our gradients differently. Plotting the the loss function over the course of learning helps us tune our learning rate.

13.2.2 Concrete Suggestions for Future Work

This section enumerates a collection of methods that would be good projects for future structured prediction research. Many of these are in response to characteristics of SPENs that we have found disappointing in practice.

Overall, using a deep network to score candidate outputs is a good idea. However, it may be useful to employ such a scoring function in alternative prediction procedures that avoid gradient-based energy minimization. First of all, gradient-based prediction is often slow for us in practice. It is unclear how much SPEN prediction could be sped up using carefully optimized code. In addition, it is difficult to apply to discrete prediction problems, as we must first optimize a continuous relaxation, and then convert this to a discrete ouput. A simple alternative approach would be to use the

SPEN to score a relatively small list of candidate outputs that are generated by some auxiliary model. Here, prediction would simply return the element of the list with the lowest energy. In addition, there may be methods where we could employ the deep SPEN energy only at train time, as a differentiable loss function used for training a fast feed-forward predictor.

In addition, future work should unroll optimization methods that better adapt to the geometry of the specific energy function to be minimized. Two different values of x may yield energy functions $E_x(\cdot)$ that are qualitatively different. Furthermore, we have found that the general scale of the energy changes dramatically over the course of learning. When we employ an optimization method that is not properly tuned to the energy, gradient descent may be chaotic (Fig. 11.2), in which case the learning signal provided by end-to-end learning would be unreliable. It would be interesting, for example, to use a learned model that directly predicts the optimizer's hyperparameters as a function of x. One could also develop a differentiable approximation of line search (Sec. 4.3.2). In addition, it may be useful to develop regularization methods that enforce that the energy always has the same general scale or that it is well-behaved (strongly-convex, low condition number, etc.). It may also be useful to design alternatives to the ICNN constraint that enforce convexity, but are less restrictive. Alternatively, we could employ a regularization method that enforces convexity as a soft constraint.

In designing our energy network architectures, it is often clear what a reasonable functional form for the dependence of $E_x(\bar{y})$ on $\bar{y}$ should be. However, it is less clear how this should depend on x. For example, in many architectures, we simply concatenate $\bar{y}$ and the features $F(x)$. Going forward, it may be useful to further explore the hypernetwork approach discussed in Sec. 11.4.2. Here, we employ an architecture only over $\bar{y}$, but the parameters of the network depend on x. This is conceptually similar to how a CRF is defined as a mapping from x to the natural parameters of an MRF.

Traditional approaches to learning loopy factor graphs are often double-loop algorithms, where an iterative method for MAP or marginal inference needs to be called in order to compute a single gradient of the loss with respect to the parameters. This is slow in practice and also sensitive to truncated optimization of the inner problem. An attractive line of work flattens these updates, such that alternating steps can be taken on the parameters and the optimization variables for the inference problem (Meshi et al., 2010; Domke, 2013b; Bach et al., 2015; Hazan et al., 2016). This is typically obtained by dualizing the inner inference problem such that learning is a joint minimization problem of parameters and messages, i.e., dual variables. The sampling-based learning methods in Chapters 11 and 12 also interleave parameter updates and updates to $\bar{y}$. However, the overall learning problem is not a joint optimization of trainable parameters w and $\bar{y}$. In future work, it may be fruitful to develop alternative SPEN learning methods that are not based on sampling but do have this interleaving property.

In Sec. 7.12, we outline how SPENs could be trained using reinforcement learning. This would allow us to directly optimize the performance of gradient-based prediction, while avoiding the pitfalls of end-to-end learning, which is vulnerable to vanishing gradients, has high memory requirements, and can not accomodate non-differentiable loss functions.

SPENs perform representation learning for $\bar{y}$, but the iterative energy minimization is done with respect to y itself. It may be preferrable to instead perform the optimization in a different representation that is directly learned such that gradient-based optimization is well-behaved. The challenge of such an approach is that we need a differentiable method for converting the optimized abstract representation to a value of $\bar{y}$. On the other hand, this technique may improve our ability to explore a diverse set of candidate outputs.

Finally, SPENs seem most important in the limited data regime, as the practitioner can introduce useful inductive bias by choosing the architecture of the energy network. Overall, though, it is unclear why a feedforward network cannot match the parsimony of a SPEN. Similarly, it is unclear what constraints on outputs can and cannot be enforced by simple feed-forward prediction. In addition, feed-forward and energy-based approaches admit different loss functions, each of which may have different associated sample complexity. We should pursue a better understanding of the regimes where energy-based prediction is most effective vs. alternative approaches, as this would help the community direct its research efforts.

BIBLIOGRAPHY

Abadi, Martın, Agarwal, Ashish, Barham, Paul, Brevdo, Eugene, Chen, Zhifeng, Citro, Craig, Corrado, Greg S, Davis, Andy, Dean, Jeffrey, Devin, Matthieu, et al. Tensorflow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. *arXiv preprint arXiv:1603.04467*, 2016.

Abbeel, Pieter and Ng, Andrew Y. Apprenticeship Learning Via Inverse Reinforcement Learning. *International Conference on Machine Learning*, pp. 1, 2004.

Agrawal, Rahul, Gupta, Archit, Prabhu, Yashoteja, and Varma, Manik. Multi-Label Learning with Millions of Labels: Recommending Advertiser Bid Phrases for Web Pages. *International Conference on World Wide Web*, 2013.

Amos, B. and Zico Kolter, J. OptNet: Differentiable Optimization As a Layer in Neural Networks. *International Conference on Machine Learning*, 2017.

Amos, Brandon, Xu, Lei, and Kolter, J Zico. Input-Convex Deep Networks. *International Conference on Machine Learning*, 2017.

Andreas, Jacob, Rabinovich, Maxim, Klein, Dan, and Jordan, Michael I. On the Accuracy of Self-Normalized Log-Linear Models. *Neural Information Processing Systems*, 2015.

Andrychowicz, Marcin, Denil, Misha, Gomez, Sergio, Hoffman, Matthew W, Pfau, David, Schaul, Tom, and de Freitas, Nando. Learning to Learn by Gradient Descent by Gradient Descent. *Neural Information Processing Systems*, 2016.

Anzaroot, Sam and McCallum, Andrew. A New Dataset for Fine-Grained Citation Field Extraction. *International Conference on Machine Learning Workshop on Peer Reviewing and Publishing Models*, 2013.

Anzaroot, Sam, Passos, Alexandre, Belanger, David, and McCallum, Andrew. Learning Soft Linear Constraints with Application to Citation Field Extraction. *Association for Computational Linguistics*, 2014.

Argall, Brenna D, Chernova, Sonia, Veloso, Manuela, and Browning, Brett. A Survey of Robot Learning from Demonstration. *Robotics and autonomous systems*, 57(5): 469–483, 2009.

Bach, Stephen H., Huang, Bert, Boyd-Graber, Jordan, and Getoor, Lise. Paired-Dual Learning for Fast Training of Latent Variable Hinge-Loss MRFs. *International Conference on Machine Learning*, 2015.

Bahdanau, Dzmitry, Brakel, Philemon, Xu, Kelvin, Goyal, Anirudh, Lowe, Ryan, Pineau, Joelle, Courville, Aaron, and Bengio, Yoshua. An Actor-Critic Algorithm for Sequence Prediction. *International Conference on Learning Representations*, 2017.

Barbu, Adrian. Training an Active Random Field for Real-Time Image Denoising. *IEEE Transactions on Image Processing*, 18(11):2451–2462, 2009.

Barto, Andrew G, Sutton, Richard S, and Anderson, Charles W. Neuronlike Adaptive Elements That Can Solve Difficult Learning Control Problems. *IEEE transactions on systems, man, and cybernetics*, pp. 834–846, 1983.

Baydin, Atilim Gunes and Pearlmutter, Barak A. Automatic Differentiation of Algorithms for Machine Learning. *arXiv preprint arXiv:1404.7456*, 2014.

Beck, Amir and Teboulle, Marc. Mirror Descent and Nonlinear Projected Subgradient Methods for Convex Optimization. *Operations Research Letters*, 31(3), 2003.

Belagiannis, Vasileios and Zisserman, Andrew. Recurrent human pose estimation. *arXiv preprint arXiv:1605.02914*, 2016.

Belanger, D., Yang, B., and McCallum, A. End-To-End Learning for Structured Prediction Energy Networks. *International Conference on Machine Learning*, 2017.

Belanger, David and McCallum, Andrew. Structured Prediction Energy Networks. *International Conference on Machine Learning*, 2016.

Belanger, David, Passos, Alexandre, Riedel, Sebastian, and McCallum, Andrew. Map Inference in Chains Using Column Generation. *Neural Information Processing Systems*, pp. 1844–1852, 2012.

Belanger, David, Passos, Alexandre, Riedel, Sebastian, and McCallum, Andrew. Message Passing for Soft Constraint Dual Decomposition. *Conference on Uncertainty in Artificial Intelligence*, 2014.

Bengio, Yoshua, Laufer, Eric, Alain, Guillaume, and Yosinski, Jason. Deep Generative Stochastic Networks Trainable by Backprop. *International Conference on Machine Learning*, pp. 226–234, 2014.

Bengio, Yoshua, Goodfellow, Ian J, and Courville, Aaron. Deep Learning. *MIT Press*, 2016.

Berger, Adam L, Pietra, Vincent J Della, and Pietra, Stephen A Della. A Maximum Entropy Approach to Natural Language Processing. *Computational linguistics*, 22 (1):39–71, 1996.

Bergstra, James, Bastien, Frédéric, Breuleux, Olivier, Lamblin, Pascal, Pascanu, Razvan, Delalleau, Olivier, Desjardins, Guillaume, Warde-Farley, David, Goodfellow, Ian, Bergeron, Arnaud, et al. Theano: Deep Learning on Gpus with Python. *Neural Information Processing Systems 2011, BigLearning Workshop, Granada, Spain*, 2011.

Besag, Julian. Statistical Analysis of Non-Lattice Data. *The statistician*, pp. 179–195, 1975.

Bethe, Hans A. Statistical Theory of Superlattices. *Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences*, 150(871):552–575, 1935.

Bhatia, Kush, Jain, Himanshu, Kar, Purushottam, Jain, Prateek, and Varma, Manik. Locally Non-Linear Embeddings for Extreme Multi-Label Learning. *CoRR*, abs/1507.02743, 2015.

Bishop, Christopher M. Mixture Density Networks. *Tech. Rep. NCRG/94/004, Neural Computing Research Group*, 1994.

Bishop, Christopher M, Lawrence, Neil, Jaakkola, Tommi, and Jordan, Michael I. Approximating Posterior Distributions in Belief Networks Using Mixtures. *Neural Information Processing Systems*, pp. 416–422, 1998.

Bordes, Antoine, Usunier, Nicolas, and Bottou, Léon. Sequence Labelling SVMs Trained in One Pass. *Machine Learning and Knowledge Discovery in Databases*, pp. 146–161, 2008.

Borenstein, Eran and Ullman, Shimon. Combined Top-Down/bottom-up Segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, 30(12): 2109–2125, 2008.

Boyd, Stephen and Vandenberghe, Lieven. *Convex Optimization.* Cambridge university press, 2004.

Boykov, Yuri and Kolmogorov, Vladimir. An Experimental Comparison of Min-Cut/max-Flow Algorithms for Energy Minimization in Vision. *Pattern Analysis and Machine Intelligence*, 26(9):1124–1137, 2004.

Brakel, Philémon, Stroobandt, Dirk, and Schrauwen, Benjamin. Training Energy-Based Models for Time-Series Imputation. *Journal of Machine Learning Research*, 14, 2013.

Bregman, Lev M. The Relaxation Method of Finding the Common Point of Convex Sets and Its Application to the Solution of Problems in Convex Programming. *USSR computational mathematics and mathematical physics*, 7(3):200–217, 1967.

Bromley, Jane, Bentz, James W, Bottou, Léon, Guyon, Isabelle, LeCun, Yann, Moore, Cliff, Säckinger, Eduard, and Shah, Roopak. Signature Verification Using a Siamese Time Delay Neural Network. *Pattern Recognition and Artificial Intelligence*, 7(04), 1993.

Bubeck, Sébastien. Convex Optimization: Algorithms and Complexity. *Found. Trends Mach. Learn.*, 8(3-4):231–357, November 2015. ISSN 1935-8237. doi: 10.1561/2200000050. URL `http://dx.doi.org/10.1561/2200000050`.

Bucak, Serhat S, Mallapragada, Pavan Kumar, Jin, Rong, and Jain, Anil K. Efficient Multi-Label Ranking for Multi-Class Learning: Application to Object Recognition. *International Conference on Computer Vision*, 2009.

Bucak, Serhat Selcuk, Jin, Rong, and Jain, Anil K. Multi-Label Learning with Incomplete Class Assignments. *Computer Vision and Pattern Recognition*, 2011.

Cabral, Ricardo S, Torre, Fernando, Costeira, João P, and Bernardino, Alexandre. Matrix Completion for Multi-Label Image Classification. *Neural Information Processing Systems*, 2011.

Carpenter, Bob, Gelman, Andrew, Hoffman, Matt, Lee, Daniel, Goodrich, Ben, Betancourt, Michael, Brubaker, Michael A, Guo, Jiqiang, Li, Peter, and Riddell, Allen. Stan: A Probabilistic Programming Language. *Journal of Statistical Software*, 20, 2016.

Carreira, Joao, Agrawal, Pulkit, Fragkiadaki, Katerina, and Malik, Jitendra. Human Pose Estimation with Iterative Error Feedback. *Computer Vision and Pattern Recognition*, 2016.

Carreras, Xavier and Màrquez, Lluís. Introduction to the CoNLL-2005 Shared Task: Semantic Role Labeling. *CoNLL*, 2005.

Chang, Kai-Wei, Krishnamurthy, Akshay, Agarwal, Alekh, Daume, Hal, and Langford, John. Learning to Search Better Than Your Teacher. *International Conference on Machine Learning*, pp. 2058–2066, 2015.

Chen, Jianshu, He, Ji, Shen, Yelong, Xiao, Lin, He, Xiaodong, Gao, Jianfeng, Song, Xinying, and Deng, Li. End-To-End Learning of Latent Dirichlet Allocation by Mirror-Descent Back Propagation. *Neural Information Processing Systems*, 2015.

Chen, Tianqi, Xu, Bing, Zhang, Chiyuan, and Guestrin, Carlos. Training Deep Nets with Sublinear Memory Cost. *arXiv preprint arXiv:1604.06174*, 2016.

Chen, Yutian, Hoffman, Matthew W, Colmenarejo, Sergio Gomez, Denil, Misha, Lillicrap, Timothy P, and de Freitas, Nando. Learning to learn without gradient descent by gradient descent. *International Conference on Machine Learning*, 2017.

Chernova, Sonia and Veloso, Manuela. Interactive Policy Learning Through Confidence-Based Autonomy. *Journal of Artificial Intelligence Research*, 34(1): 1, 2009.

Chung, Junyoung, Gulcehre, Caglar, Cho, KyungHyun, and Bengio, Yoshua. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *arXiv preprint arXiv:1412.3555*, 2014.

Collins, Michael. Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. *Proceedings of the Association for Computational Linguistics-02 Conference on Empirical Methods in Natural Language Processing-Volume 10*, pp. 1–8, 2002.

Collobert, R., Kavukcuoglu, K., and Farabet, C. Torch7: A Matlab-like Environment for Machine Learning. *BigLearn, Neural Information Processing Systems Workshop*, 2011.

Cramér, Harald. Mathematical Methods of Statistics, 1947.

Dabov, Kostadin, Foi, Alessandro, Katkovnik, Vladimir, and Egiazarian, Karen. Image Denoising by Sparse 3-D Transform-Domain Collaborative Filtering. *IEEE Transactions on image processing*, 16(8):2080–2095, 2007.

Das, Dipanjan, Martins, André FT, and Smith, Noah A. An Exact Dual Decomposition Algorithm for Shallow Semantic Parsing with Constraints. *Conference on Lexical and Computational Semantics*, 2012.

Daumé III, Hal and Marcu, Daniel. Learning As Search Optimization: Approximate Large Margin Methods for Structured Prediction. *International Conference on Machine Learning*, pp. 169–176, 2005.

Dayan, Peter, Hinton, Geoffrey E, Neal, Radford M, and Zemel, Richard S. The Helmholtz Machine. *Neural computation*, 7(5):889–904, 1995.

Devlin, Jacob, Zbib, Rabih, Huang, Zhongqiang, Lamar, Thomas, Schwartz, Richard M, and Makhoul, John. Fast and Robust Neural Network Joint Models for Statistical Machine Translation. *Association for Computational Linguistics (1)*, pp. 1370–1380, 2014.

Domke, Justin. Dual Decomposition for Marginal Inference. *Conference on Artificial Intelligence*, 2011.

Domke, Justin. Generic Methods for Optimization-Based Modeling. *International Conference on Artificial Intelligence and Statistics*, 22:318–326, 2012.

Domke, Justin. Learning Graphical Model Parameters with Approximate Marginal Inference. *Pattern Analysis and Machine Intelligence*, 2013a.

Domke, Justin. Structured Learning Via Logistic Regression. *Neural Information Processing Systems*, pp. 647–655, 2013b.

Donti, P. L., Amos, B., and Zico Kolter, J. Task-Based End-To-End Model Learning. *ArXiv e-prints*, March 2017.

Doppa, Janardhan Rao, Fern, Alan, and Tadepalli, Prasad. Structured Prediction Via Output Space Search. *The Journal of Machine Learning Research*, 15(1):1317–1350, 2014.

Duchi, John, Tarlow, Daniel, Elidan, Gal, and Koller, Daphne. Using Combinatorial Optimization Within Max-Product Belief Propagation. *Neural Information Processing Systems*, 19:369, 2007.

Duchi, John, Shalev-Shwartz, Shai, Singer, Yoram, and Chandra, Tushar. Efficient Projections onto the L 1-Ball for Learning in High Dimensions. *International Conference on Machine Learning*, 2008.

Duvenaud, David, MAssociation for Computational Linguisticsaurin, Dougal, and Adams, Ryan P. Early Stopping As Nonparametric Variational Inference. *International Conference on Artificial Intelligence and Statistics*, 2016.

Dyer, Chris. Notes on Noise Contrastive Estimation and Negative Sampling. *arXiv preprint arXiv:1410.8251*, 2014.

Eisner, Jason M. Three New Probabilistic Models for Dependency Parsing: An Exploration. *Proceedings of the 16th Conference on Computational Linguistics-Volume 1*, pp. 340–345, 1996.

Elisseeff, André and Weston, Jason. A Kernel Method for Multi-Labelled Classification. *Neural Information Processing Systems*, 2001.

Fanello, Sean Ryan, Keskin, Cem, Kohli, Pushmeet, Izadi, Shahram, Shotton, Jamie, Criminisi, Antonio, Pattacini, Ugo, and Paek, Tim. Filter Forests for Learning Data-Dependent Convolutional Kernels. *Computer Vision and Pattern Recognition*, 2014.

Filippova, Katja, Alfonseca, Enrique, Colmenares, Carlos A, Kaiser, Lukasz, and Vinyals, Oriol. Sentence Compression by Deletion with LSTMs. *Empirical Methods in Natural Language Processing*, pp. 360–368, 2015.

Finley, Thomas and Joachims, Thorsten. Training Structural SVMs When Exact Inference Is Intractable. *International Conference on Machine Learning*, 2008.

FitzGerald, Nicholas, Täckström, Oscar, Ganchev, Kuzman, and Das, Dipanjan. Semantic Role Labeling with Neural Network Factors. *Empirical Methods in Natural Language Processing*, pp. 960–970, 2015.

Foo, Chuan-sheng, Do, Chuong B, and Ng, Andrew Y. Efficient Multiple Hyperparameter Learning for Log-Linear Models. *Neural Information Processing Systems*, pp. 377–384, 2008.

for Computational Linguisticsaurin, Dougal MAssociation, Duvenaud, David, and Adams, Ryan P. Gradient-Based Hyperparameter Optimization Through Reversible Learning. *International Conference on Machine Learning*, July 2015.

Freund, Yoav and Haussler, David. Unsupervised Learning of Distributions of Binary Vectors Using Two Layer Networks. *Neural Information Processing Systems*, 1992.

Fu, Qiang and Banerjee, Huahua Wang Arindam. Bethe-ADMM for Tree Decomposition Based Parallel MAP Inference. *Conference on Uncertainty in Artificial Intelligence*, 2013.

Ganchev, Kuzman, Graça, Joao, Gillenwater, Jennifer, and Taskar, Ben. Posterior Regularization for Structured Latent Variable Models. *Journal of Machine Learning Research*, 99:2001–2049, 2010.

Gatys, Leon A., Ecker, Alexander S., and Bethge, Matthias. A Neural Algorithm of Artistic Style. *CoRR*, abs/1508.06576, 2015a.

Gatys, Leon A., Ecker, Alexander S., and Bethge, Matthias. Texture Synthesis Using Convolutional Neural Networks. *Neural Information Processing Systems*, 2015b.

Geman, Stuart and Geman, Donald. Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images. *IEEE Transactions on pattern analysis and machine intelligence*, pp. 721–741, 1984.

Ghamrawi, Nadia and McCallum, Andrew. Collective Multi-Label Classification. *ACM International Conference on Information and Knowledge Management*, 2005.

Gildea, Daniel and Jurafsky, Daniel. Automatic Labeling of Semantic Roles. *Computational linguistics*, 28(3):245–288, 2002.

Gimpel, Kevin and Smith, Noah A. Softmax-Margin CRFs: Training Log-Linear Models with Cost Functions. *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pp. 733–736, 2010.

Girolami, Mark and Calderhead, Ben. Riemann Manifold Langevin and Hamiltonian Monte Carlo Methods. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 73(2):123–214, 2011.

Globerson, Amir and Jaakkola, Tommi S. Fixing Max-Product: Convergent Message Passing Algorithms for MAP LP-Relaxations. *Neural Information Processing Systems*, 2008.

Glorot, Xavier and Bengio, Yoshua. Understanding the Difficulty of Training Deep Feedforward Neural Networks. *International Conference on Artificial Intelligence and Statistics*, 9:249–256, 2010.

Godbole, Shantanu and Sarawagi, Sunita. Discriminative Methods for Multi-Labeled Classification. In *Advances in Knowledge Discovery and Data Mining*, pp. 22–30. Springer, 2004.

Goel, Vaibhava and Byrne, William J. Minimum Bayes-Risk Automatic Speech Recognition. *Computer Speech & Language*, 14(2):115–135, 2000.

Goodfellow, Ian J, Shlens, Jonathon, and Szegedy, Christian. Explaining and Harnessing Adversarial Examples. *International Conference on Learning Representations*, 2015.

Goodman, Noah, Mansinghka, Vikash, Roy, Daniel M, Bonawitz, Keith, and Tenenbaum, Joshua B. Church: A Language for Generative Models. *Uncertainty in Artificial Intelligence*, 2008.

Goodman, Noah D. The Principles and Practice of Probabilistic Programming. *ACM SIGPLAN Notices*, 48(1):399–402, 2013.

Graves, Alex, Fernández, Santiago, and Schmidhuber, Jürgen. Bidirectional LSTM Networks for Improved Phoneme Classification and Recognition. *Artificial Neural Networks: Formal Models and Their Applications–ICANN 2005*, pp. 753–753, 2005.

Greff, Klaus, Srivastava, Rupesh K, and Schmidhuber, Jürgen. Highway and Residual Networks Learn Unrolled Iterative Estimation. *International Conference on Learning Representations*, 2017.

Gregor, Karol and LeCun, Yann. Learning Fast Approximations of Sparse Coding. *International Conference on Machine Learning*, 2010.

Gruslys, Audrunas, Munos, Remi, Danihelka, Ivo, Lanctot, Marc, and Graves, Alex. Memory-Efficient Backpropagation Through Time. *Neural Information Processing Systems*, pp. 4125–4133, 2016.

Gu, Jiatao, Cho, Kyunghyun, and Li, Victor OK. Trainable Greedy Decoding for Neural Machine Translation. *arXiv preprint arXiv:1702.02429*, 2017.

Gutmann, Michael and Hyvärinen, Aapo. Noise-Contrastive Estimation: A New Estimation Principle for Unnormalized Statistical Models. *International Conference on Artificial Intelligence and Statistics*, 2010.

Guzman-Rivera, Abner, Batra, Dhruv, and Kohli, Pushmeet. Multiple Choice Learning: Learning to Produce Multiple Structured Outputs. *Neural Information Processing Systems*, pp. 1799–1807, 2012.

Guzman-Rivera, Abner, Kohli, Pushmeet, Batra, Dhruv, and Rutenbar, Rob. Efficiently Enforcing Diversity in Multi-Output Structured Prediction. *Artificial Intelligence and Statistics*, pp. 284–292, 2014.

Gygli, Michael, Norouzi, Mohammad, and Angelova, Anelia. Deep Value Networks Learn to Evaluate and Iteratively Refine Structured Outputs. *International Conference on Machine Learning*, 2017.

Ha, David, Dai, Andrew, and Le, Quoc. HyperNetworks. *International Conference on Learning Representations*, 2017.

Hariharan, Bharath, Zelnik-Manor, Lihi, Varma, Manik, and Vishwanathan, Svn. Large Scale Max-Margin Multi-Label Classification with Priors. *International Conference on Machine Learning*, 2010.

Hazan, Tamir, Schwing, Alexander G, and Urtasun, Raquel. Blending Learning and Inference in Conditional Random Fields. *Journal of Machine Learning Research*, 17:1–22, 2016.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep Residual Learning for Image Recognition. *Computer Vision and Pattern Recognition*, 2016.

He, L., Gillenwater, J., and Taskar, B. Graph-Based Posterior Regularization for Semi-Supervised Structured Prediction. *CoNLL*, 2013.

Hershey, John R, Roux, Jonathan Le, and Weninger, Felix. Deep Unfolding: Model-Based Inspiration of Novel Deep Architectures. *arXiv preprint arXiv:1409.2574*, 2014.

Hinton, Geoffrey, Osindero, Simon, Welling, Max, and Teh, Yee-Whye. Unsupervised Discovery of Nonlinear Structure Using Contrastive Backpropagation. *Cognitive science*, 30(4):725–731, 2006a.

Hinton, Geoffrey, Deng, Li, Yu, Dong, Dahl, George E, Mohamed, Abdel-rahman, Jaitly, Navdeep, Senior, Andrew, Vanhoucke, Vincent, Nguyen, Patrick, Sainath, Tara N, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. *IEEE Signal Processing Magazine*, 29 (6):82–97, 2012.

Hinton, Geoffrey E. Training Products of Experts by Minimizing Contrastive Divergence. *Neural computation*, 14(8):1771–1800, 2002.

Hinton, Geoffrey E, Osindero, Simon, and Teh, Yee-Whye. A Fast Learning Algorithm for Deep Belief Nets. *Neural computation*, 18(7):1527–1554, 2006b.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long Short-Term Memory. *Neural computation*, 1997.

Hochreiter, Sepp, Bengio, Yoshua, Frasconi, Paolo, and Schmidhuber, Jürgen. Gradient Flow in Recurrent Nets: The Difficulty of Learning Long-Term Dependencies, 2001a.

Hochreiter, Sepp, Younger, A Steven, and Conwell, Peter R. Learning to Learn Using Gradient Descent. *International Conference on Artificial Neural Networks*, pp. 87–94, 2001b.

Hoffman, Matthew D and Gelman, Andrew. The No-U-Turn Sampler: Adaptively Setting Path Lengths in Hamiltonian Monte Carlo. *Journal of Machine Learning Research*, 15(1):1593–1623, 2014.

Hsu, Daniel, Kakade, Sham, Langford, John, and Zhang, Tong. Multi-Label Prediction Via Compressed Sensing. *Neural Information Processing Systems*, 2009.

Hyvärinen, Aapo. Estimation of Non-Normalized Statistical Models by Score Matching. *Journal of Machine Learning Research*, 6(Apr):695–709, 2005.

Jaakkola, Tommi, Saul, Lawrence K, and Jordan, Michael I. Fast Learning by Bounding Likelihoods in Sigmoid Type Belief Networks. *Neural Information Processing Systems*, pp. 528–534, 1996.

Jernite, Yacine, Rush, Alexander M., and Sontag, David. A Fast Variational Approach for Learning Markov Random Field Language Models. *International Conference on Machine Learning*, 2015.

Ji, Shuiwang and Ye, Jieping. Linear Dimensionality Reduction for Multi-Label Classification. *IJCAI*, 9:1077–1082, 2009.

Jia, Yangqing, Shelhamer, Evan, Donahue, Jeff, Karayev, Sergey, Long, Jonathan, Girshick, Ross, Guadarrama, Sergio, and Darrell, Trevor. Caffe: Convolutional Architecture for Fast Feature Embedding. *Proceedings of the 22nd ACM International Conference on Multimedia*, pp. 675–678, 2014.

Jordan, Michael I, Ghahramani, Zoubin, Jaakkola, Tommi S, and Saul, Lawrence K. An Introduction to Variational Methods for Graphical Models. *Machine learning*, 37(2):183–233, 1999.

Kapoor, Ashish, Viswanathan, Raajay, and Jain, Prateek. Multilabel Classification Using Bayesian Compressed Sensing. *Neural Information Processing Systems*, 2012.

Kingma, Diederik and Ba, Jimmy. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 2015a.

Kingma, Diederik P. and Ba, Jimmy. Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations*, 2015b.

Kingma, Diederik P and Welling, Max. Auto-Encoding Variational Bayes. *International Conference on Learning Representations*, 2014.

Kingma, D.P. and LeCun, Yann. Regularized Estimation of Image Statistics by Score Matching. *Neural Information Processing Systems*, 23:1126–1134, 2010.

Klein, Dan and Manning, Christopher D. A Parsing: Fast Exact Viterbi Parse Selection. *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pp. 40–47, 2003.

Kohli, Pushmeet, Ladicky, L'ubor, and Torr, Philip HS. Robust Higher Order Potentials for Enforcing Label Consistency. *Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

Koller, Daphne and Friedman, Nir. *Probabilistic Graphical Models: Principles and Techniques*. MIT press, 2009.

Komodakis, Nikos, Paragios, Nikos, and Tziritas, Georgios. MRF Optimization Via Dual Decomposition: Message-Passing Revisited. *IEEE ICCV*, 2007.

Konda, Vijaymohan R and Borkar, Vivek S. Actor-Critic–Type Learning Algorithms for Markov Decision Processes. *SIAM Journal on control and Optimization*, 38(1): 94–123, 1999.

Koo, Terry, Globerson, Amir, Carreras Pérez, Xavier, and Collins, Michael. Structured Prediction Models Via the Matrix-Tree Theorem. *Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (Empirical Methods in Natural Language Processing-CoNLL)*, pp. 141–150, 2007.

Kschischang, Frank R, Frey, Brendan J, and Loeliger, H-A. Factor Graphs and the Sum-Product Algorithm. *IEEE Transactions on information theory*, 47(2):498–519, 2001.

Kucukelbir, Alp, Ranganath, Rajesh, Gelman, Andrew, and Blei, David. Automatic Variational Inference in Stan. *Neural Information Processing Systems*, pp. 568–576, 2015.

Kulesza, Alex and Pereira, Fernando. Structured Learning with Approximate Inference. *Neural Information Processing Systems*, 2007.

Kulesza, Alex and Taskar, Ben. Structured determinantal point processes. *Advances in neural information processing systems*, pp. 1171–1179, 2010.

Kulesza, Alex and Taskar, Ben. Learning Determinantal Point Processes. *Conference on Uncertainty in Artificial Intelligence*, 2011.

Kunisch, Karl and Pock, Thomas. A Bilevel Optimization Approach for Parameter Learning in Variational Models. *SIAM Journal on Imaging Sciences*, 6(2):938–983, 2013.

Lafferty, John D, McCallum, Andrew, and Pereira, Fernando CN. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. *International Conference on Machine Learning*, 2001.

Lample, Guillaume, Ballesteros, Miguel, Kawakami, Kazuya, Subramanian, Sandeep, and Dyer, Chris. Neural Architectures for Named Entity Recognition. *Proc. NAAssociation for Computational Linguistics-HLT*, 2016.

Larochelle, Hugo and Lauly, Stanislas. A Neural Autoregressive Topic Model. *Neural Information Processing Systems*, pp. 2708–2716, 2012.

Larochelle, Hugo and Murray, Iain. The Neural Autoregressive Distribution Estimator. *International Conference on Artificial Intelligence and Statistics*, 1:2, 2011.

Le, Quoc and Mikolov, Tomas. Distributed Representations of Sentences and Documents. *International Conference on Machine Learning*, 2014.

LeCun, Yann, Bottou, Léon, Bengio, Yoshua, and Haffner, Patrick. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

LeCun, Yann, Chopra, Sumit, Hadsell, Raia, Ranzato, M, and Huang, F. A Tutorial on Energy-Based Learning. *Predicting Structured Data*, 1, 2006.

Lee, Honglak, Grosse, Roger, Ranganath, Rajesh, and Ng, Andrew Y. Convolutional Deep Belief Networks for Scalable Unsupervised Learning of Hierarchical Representations. *International Conference on Machine Learning*, pp. 609–616, 2009.

Lee, Kenton, Lewis, Mike, and Zettlemoyer, Luke. Global Neural CCG Parsing with Optimality Guarantees. *Empirical Methods in Natural Language Processing*, 2016.

Lewis, Bil. Debugging Backwards in Time. *arXiv preprint cs/0310016*, 2003.

Lewis, Mike, He, Luheng, and Zettlemoyer, Luke. Joint A* CCG Parsing and Semantic Role Labelling. *Empirical Methods in Natural Language Processing*, pp. 1444–1454, 2015.

Li, Ke, Hariharan, Bharath, and Malik, Jitendra. Iterative instance segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3659–3667, 2016.

Li, Yujia and Zemel, Richard S. Mean-Field Networks. *International Conference on Machine Learning Workshop on Learning Tractable Probabilistic Models*, 2014.

Liang, Percy, Jordan, Michael I, and Klein, Dan. Learning from Measurements in Exponential Families. *International Conference on Machine Learning*, 2009.

Liang, Percy, Jordan, Michael I, and Klein, Dan. Learning Dependency-Based Compositional Semantics. *Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pp. 590–599, 2011.

Lin, Victoria (Xi), Singh, Sameer, He, Luheng, Taskar, Ben, and Zettlemoyer, Luke. Multi-Label Learning with Posterior Regularization. *Neural Information Processing Systems Workshop on Modern Machine Learning and NLP*, 2014.

Mann, Gideon S and McCallum, Andrew. Generalized Expectation Criteria for Semi-Supervised Learning with Weakly Labeled Data. *Journal of Machine Learning Research*, 11:955–984, 2010.

Mansinghka, Vikash, Selsam, Daniel, and Perov, Yura. Venture: A Higher-Order Probabilistic Programming Platform with Programmable Inference. *arXiv preprint arXiv:1404.0099*, 2014.

Marks, Tim K and Movellan, Javier R. Diffusion Networks, Products of Experts, and Factor Analysis. *Proc. Int. Conf. on Independent Component Analysis*, pp. 481–485, 2001.

Martins, André, Figueiredo, Mário, Aguiar, Pedro, Smith, Noah A, and Xing, Eric P. An Augmented Lagrangian Approach to Constrained MAP Inference. *International Conference on Machine Learning*, 2011a.

Martins, André FT, Figeuiredo, Mario AT, Aguiar, Pedro MQ, Smith, Noah A, and Xing, Eric P. An Augmented Lagrangian Approach to Constrained MAP Inference. *International Conference on Machine Learning*, 2011b.

Martins, André FT, Figueiredo, Mário AT, Aguiar, Pedro MQ, Smith, Noah A, and Xing, Eric P. AD3: Alternating Directions Dual Decomposition for MAP Inference in Graphical Models. *Journal of Machine Learning Research*, 16:495–545, 2015.

McCallum, Andrew, Schultz, Karl, and Singh, Sameer. Factorie: Probabilistic Programming Via Imperatively Defined Factor Graphs. *Neural Information Processing Systems*, pp. 1249–1257, 2009.

McDonald, Ryan and Satta, Giorgio. On the Complexity of Non-Projective Data-Driven Dependency Parsing. *Proceedings of the 10th International Conference on Parsing Technologies*, pp. 121–132, 2007.

Meshi, Ofer, Sontag, David, Globerson, Amir, and Jaakkola, Tommi S. Learning Efficiently with Approximate Inference Via Dual Losses. *International Conference on Machine Learning*, 2010.

Metz, Luke, Poole, Ben, Pfau, David, and Sohl-Dickstein, Jascha. Unrolled Generative Adversarial Networks. *International Conference on Learning Representations*, 2017.

Mikolov, Tomas, Sutskever, Ilya, Chen, Kai, Corrado, Greg S, and Dean, Jeff. Distributed Representations of Words and Phrases and Their Compositionality. *Neural Information Processing Systems*, 2013.

Mnih, Andriy and Gregor, Karol. Neural Variational Inference and Learning in Belief Networks. *International Conference on Machine Learning*, 2014.

Mnih, Andriy and Hinton, Geoffrey. Learning Nonlinear Constraints with Contrastive Backpropagation. *IJCNN*, 2005.

Mnih, Andriy and Kavukcuoglu, Koray. Learning Word Embeddings Efficiently with Noise-Contrastive Estimation. *Neural Information Processing Systems*, pp. 2265–2273, 2013.

Mnih, Andriy and Teh, Yee Whye. A Fast and Simple Algorithm for Training Neural Probabilistic Language Models. *International Conference on Machine Learning*, 2012.

Mooij, Joris M. LibDAI: A Free and Open Source C++ Library for Discrete Approximate Inference in Graphical Models. *Journal of Machine Learning Research*, 11: 2169–2173, August 2010.

Mordvintsev, Alexander, Olah, Christopher, and Tyka, Mike. Inceptionism: Going Deeper into Neural Networks. research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html, 2015.

Nair, Vinod and Hinton, Geoffrey E. 3D Object Recognition with Deep Belief Nets. *Neural Information Processing Systems*, pp. 1339–1347, 2009.

Neal, Radford M. Connectionist Learning of Belief Networks. *Artificial intelligence*, 56(1):71–113, 1992.

Neal, Radford M. Slice Sampling. *Annals of statistics*, pp. 705–741, 2003.

Neal, Radford M et al. MCMC Using Hamiltonian Dynamics. *Handbook of Markov Chain Monte Carlo*, 2:113–162, 2011.

Nesterov, Yurii. Primal-Dual Subgradient Methods for Convex Problems. *Mathematical programming*, 120(1):221–259, 2009.

Newcombe, Richard A, Izadi, Shahram, Hilliges, Otmar, Molyneaux, David, Kim, David, Davison, Andrew J, Kohi, Pushmeet, Shotton, Jamie, Hodges, Steve, and Fitzgibbon, Andrew. KinectFusion: Real-Time Dense Surface Mapping and Tracking. *IEEE International Symposium on Mixed and Augmented Reality*, 2011.

Newell, Alejandro, Yang, Kaiyu, and Deng, Jia. Stacked hourglass networks for human pose estimation. *European Conference on Computer Vision*, 2016.

Ngiam, Jiquan, Chen, Zhenghao, Koh, Pang W, and Ng, Andrew Y. Learning Deep Energy Models. *International Conference on Machine Learning*, 2011.

Nguyen, Trung V and Bonilla, Edwin V. Automated Variational Inference for Gaussian Process Models. *Neural Information Processing Systems*, pp. 1404–1412, 2014.

Niculescu-Mizil, Alexandru and Abbasnejad, Ehsan. Label Filters for Large Scale Multilabel Classification. *International Conference on Machine Learning 2015 Workshop on Extreme Classification*, 2015.

on Uncertainty in Artificial Intelligence Zheng, ShConference, Jayasumana, Sadeep, Romera-Paredes, Bernardino, Vineet, Vibhav, Su, Zhizhong, Du, Dalong, Huang, Chang, and Torr, Philip. Conditional Random Fields As Recurrent Neural Networks. *International Conference on Computer Vision (ICCV)*, 2015.

Passty, Gregory B. Ergodic Convergence to a Zero of the Sum of Monotone Operators in Hilbert Space. *Journal of Mathematical Analysis and Applications*, 72(2):383 – 390, 1979.

Pearl, Judea. *Reverend Bayes on Inference Engines: A Distributed Hierarchical Approach*. Cognitive Systems Laboratory, School of Engineering and Applied Science, University of California, Los Angeles, 1982.

Pearlmutter, Barak A. Fast Exact Multiplication by the Hessian. *Neural computation*, 6(1):147–160, 1994.

Petterson, James and Caetano, Tibério S. Submodular Multi-Label Learning. *Neural Information Processing Systems*, 2011.

Polyak, Boris T. Some Methods of Speeding up the Convergence of Iteration Methods. *USSR Computational Mathematics and Mathematical Physics*, 4(5):1–17, 1964.

Premachandran, Vittal, Tarlow, Daniel, and Batra, Dhruv. Empirical Minimum Bayes Risk Prediction: How to Extract an Extra Few% Performance from Vision Models with Just Three More Parameters. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1043–1050, 2014.

Punyakanok, Vasin, Roth, Dan, and Yih, Wen-tau. The Importance of Syntactic Parsing and Inference in Semantic Role Labeling. *Computational Linguistics*, 34, 2008.

Rahimi, Ali and Recht, Benjamin. Random Features for Large-Scale Kernel Machines. *Neural Information Processing Systems*, 2007.

Ramshaw, Lance A and Marcus, Mitchell P. Text Chunking Using Transformation-Based Learning. In *Natural Language Processing Using Very Large Corpora*, pp. 157–176. Springer, 1999.

Ranganath, Rajesh, Gerrish, Sean, and Blei, David M. Black Box Variational Inference. *International Conference on Artificial Intelligence and Statistics*, pp. 814–822, 2014.

Ranzato, Marc'Aurelio, Poultney, Christopher, Chopra, Sumit, and LeCun, Yann. Efficient Learning of Sparse Representations with an Energy-Based Model. *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pp. 1137–1144, 2006.

Ranzato, Marc'Aurelio, Chopra, Sumit, Auli, Michael, and Zaremba, Wojciech. Sequence Level Training with Recurrent Neural Networks. *International Conference on Learning Representations*, 2016.

Rao, C Radhakrishna. Information and Accuracy Attainable in the Estimation of Statistical Parameters. *Bull. Calcutta Math. Soc*, 37(3):81–91, 1945.

Ratliff, Nathan, Bradley, David, Bagnell, J Andrew, and Chestnutt, Joel. Boosting Structured Prediction for Imitation Learning. *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pp. 1153–1160, 2006.

Ravikumar, Pradeep, Agarwal, Alekh, and Wainwright, Martin J. Message-Passing for Graph-Structured Linear Programs: Proximal Methods and Rounding Schemes. *Journal of Machine Learning Research*, 11:1043–1080, 2010.

Read, Jesse, Pfahringer, Bernhard, Holmes, Geoff, and Frank, Eibe. Classifier Chains for Multi-Label Classification. *Machine learning*, 85(3):333–359, 2011.

Rennie, Jason DM. Smooth Hinge Classification, 2005.

Rezende, Danilo Jimenez and Mohamed, Shakir. Variational Inference with Normalizing Flows. *International Conference on Machine Learning*, 2015.

Rockafellar, R Tyrell. *Convex Analysis*, volume 28. Princeton University Press, 1997.

Rohanimanesh, Khashayar, Singh, Sameer, McCallum, Andrew, and Black, Michael J. Training Factor Graphs with Reinforcement Learning for Efficient Map Inference. *Neural Information Processing Systems*, pp. 2044–2052, 2009.

Ross, Stéphane and Bagnell, Drew. Efficient Reductions for Imitation Learning. *International Conference on Artificial Intelligence and Statistics*, 2010.

Ross, Stéphane, Gordon, Geoffrey J, and Bagnell, Drew. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *International Conference on Artificial Intelligence and Statistics*, 2011a.

Ross, Stephane, Munoz, Daniel, Hebert, Martial, and Bagnell, J Andrew. Learning message-passing inference machines for structured prediction. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 2737–2744. IEEE, 2011b.

Roth, Stefan and Black, Michael J. Fields of Experts: A Framework for Learning Image Priors. *Computer Vision and Pattern Recognition*, 2005.

Rush, Alexander M and Petrov, Slav. Vine Pruning for Efficient Multi-Pass Dependency Parsing. *Proceedings of the 2012 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp. 498–507, 2012.

Rush, Alexander M, Sontag, David, Collins, Michael, and Jaakkola, Tommi. On Dual Decomposition and Linear Programming Relaxations for Natural Language Processing. *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 1–11, 2010.

Russell, Chris, Kohli, Pushmeet, Torr, Philip HS, et al. Associative Hierarchical Crfs for Object Class Image Segmentation. *Computer Vision, 2009 IEEE 12th International Conference On*, pp. 739–746, 2009.

Salakhutdinov, Ruslan and Hinton, Geoffrey. Deep Boltzmann Machines. *Artificial Intelligence and Statistics*, pp. 448–455, 2009.

Salimans, Tim, Kingma, Diederik P, Welling, Max, et al. Markov Chain Monte Carlo and Variational Inference: Bridging the Gap. *International Conference on Machine Learning*, pp. 1218–1226, 2015.

Samuel, Kegan GG and Tappen, Marshall F. Learning Optimized Map Estimates in Continuously-Valued Mrf Models. *Computer Vision and Pattern Recognition*, 2009.

Saul, Lawrence K and Jordan, Michael I. Exploiting Tractable Substructures in Intractable Networks. *Neural Information Processing Systems*, pp. 486–492, 1996.

Saul, Lawrence K, Jaakkola, Tommi, and Jordan, Michael I. Mean Field Theory for Sigmoid Belief Networks. *Journal of artificial intelligence research*, 4(1):61–76, 1996.

Schaal, Stefan. Is Imitation Learning the Route to Humanoid Robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

Schmidt, Uwe, Gao, Qi, and Roth, Stefan. A Generative Perspective on Mrfs in Low-Level Vision. *Computer Vision and Pattern Recognition*, 2010.

Silver, D., van Hasselt, H., Hessel, M., Schaul, T., Guez, A., Harley, T., Dulac-Arnold, G., Reichert, D., Rabinowitz, N., Barreto, A., and Degris, T. the Predictron: End-To-End Learning and Planning. *International Conference on Machine Learning*, 2017.

Silver, David, Bagnell, James, and Stentz, Anthony. High Performance Outdoor Navigation from Overhead Data Using Imitation Learning. *Robotics: Science and Systems IV, Zurich, Switzerland*, 2008.

Smith, David A and Eisner, Jason. Dependency Parsing by Belief Propagation. *Empirical Methods in Natural Language Processing*, pp. 145–156, 2008.

Smola, Alex, Gretton, Arthur, Song, Le, and Schölkopf, Bernhard. A Hilbert Space Embedding for Distributions. *Algorithmic Learning Theory*, pp. 13–31, 2007.

Smolensky, P. Foundations of Harmony Theory: Cognitive Dynamical Systems and the Subsymbolic Theory of Information Processing. *Parallel distributed processing: Explorations in the microstructure of cognition*, 1:191–281, 1986.

Sontag, David, Globerson, Amir, and Jaakkola, Tommi. Introduction to Dual Decomposition for Inference. *Optimization for Machine Learning*, 1:219–254, 2011.

Srikumar, Vivek and Manning, Christopher D. Learning Distributed Representations for Structured Output Prediction. *Neural Information Processing Systems*, 2014.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Srivastava, Rupesh K, Greff, Klaus, and Schmidhuber, Jürgen. Training Very Deep Networks. *Neural Information Processing Systems*, 2015.

Stoyanov, Veselin and Eisner, Jason. Easy-First Coreference Resolution. *COLING*, pp. 2519–2534, 2012.

Stoyanov, Veselin, Ropson, Alexander, and Eisner, Jason. Empirical Risk Minimization of Graphical Model Parameters Given Approximate Inference, Decoding, and Model Structure. *International Conference on Artificial Intelligence and Statistics*, 2011.

Strubell, Emma, Verga, Patrick, Belanger, David, and McCallum, Andrew. Fast and Accurate Sequence Labeling with Iterated Dilated Convolutions. *arXiv preprint arXiv:1702.02098*, 2017.

Sun, Jian and Tappen, Marshall F. Learning Non-Local Range Markov Random Field for Image Restoration. *Computer Vision and Pattern Recognition*, 2011.

Sutskever, I, Martens, J, Dahl, George, and Hinton, Geoffery. On the Importance of Momentum and Initialization in Deep Learning. *30th International Conference on Machine Learning*, 2013.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to Sequence Learning with Neural Networks. *Neural Information Processing Systems*, 2014.

Sutton, Richard S and Barto, Andrew G. *Reinforcement Learning: An Introduction.* MIT press Cambridge, 1998.

Swersky, Kevin, Chen, Bo, Marlin, Ben, and De Freitas, Nando. A Tutorial on Stochastic Approximation Algorithms for Training Restricted Boltzmann Machines and Deep Belief Nets. *Information Theory and Applications Workshop (ITA), 2010*, pp. 1–10, 2010.

Swersky, Kevin, Buchman, David, Freitas, Nando D, Marlin, Benjamin M, et al. On Autoencoders and Score Matching for Energy Based Models. *International Conference on Machine Learning*, pp. 1201–1208, 2011.

Swersky, Kevin, Tarlow, Daniel, Adams, Ryan P, Zemel, Richard S, and Frey, Brendan J. Probabilistic N-Choose-K Models for Classification and Ranking. *Neural Information Processing Systems*, pp. 3059–3067, 2012.

Szegedy, Christian, Zaremba, Wojciech, Sutskever, Ilya, Bruna, Joan, Erhan, Dumitru, Goodfellow, Ian, and Fergus, Rob. Intriguing Properties of Neural Networks. *International Conference on Learning Representations*, 2014.

Täckström, Oscar, Ganchev, Kuzman, and Das, Dipanjan. Efficient Inference and Structured Learning for Semantic Role Labeling. *Transactions of the Association for Computational Linguistics*, 2015.

Tamar, Aviv, Levine, Sergey, Abbeel, Pieter, WU, YI, and Thomas, Garrett. Value Iteration Networks. *Neural Information Processing Systems*, pp. 2146–2154, 2016.

Tang, Kui, Ruozzi, Nicholas, Belanger, David, and Jebara, Tony. Bethe Learning of Conditional Random Fields Via Map Decoding. *Artificial Intelligence and Statistics*, 2015.

Tappen, Marshall F, Liu, Ce, Adelson, Edward H, and Freeman, William T. Learning Gaussian Conditional Random Fields for Low-Level Vision. *Computer Vision and Pattern Recognition*, 2007.

Tarlow, Daniel and Zemel, Richard S. Structured Output Learning with High Order Loss Functions. *International Conference on Artificial Intelligence and Statistics*, pp. 1212–1220, 2012.

Tarlow, Daniel, Givoni, Inmar E, and Zemel, Richard S. HOP-MAP: Efficient Message Passing with High Order Potentials. *International Conference on Artificial Intelligence and Statistics*, 5:6, 2010.

Tarlow, Daniel, Adams, Ryan, and Zemel, Richard. Randomized Optimum Models for Structured Prediction. *Artificial Intelligence and Statistics*, pp. 1221–1229, 2012.

Taskar, B., Guestrin, C., and Koller, D. Max-Margin Markov Networks. *Neural Information Processing Systems*, 2004.

Theis, Lucas and Bethge, Matthias. Generative Image Modeling Using Spatial LSTMs. *Neural Information Processing Systems*, pp. 1927–1935, 2015.

Tieleman, Tijmen. Training Restricted Boltzmann Machines Using Approximations to the Likelihood Gradient. *International Conference on Machine Learning*, pp. 1064–1071, 2008.

Titov, Ivan and Henderson, James. Bayes Risk Minimization in Natural Language Parsing. *University of Geneva technical report*, 2006.

Tjong Kim Sang, Erik F and De Meulder, Fien. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAAssociation for Computational Linguistics 2003-Volume 4*, pp. 142–147, 2003.

Tolpin, David, van de Meent, Jan-Willem, and Wood, Frank. Probabilistic Programming in Anglican. *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pp. 308–311, 2015.

Torralba, Antonio, Fergus, Rob, and Weiss, Yair. Small Codes and Large Image Databases for Recognition. *Computer Vision and Pattern Recognition, 2008. Computer Vision and Pattern Recognition 2008. IEEE Conference On*, pp. 1–8, 2008.

Tsochantaridis, Ioannis, Hofmann, Thomas, Joachims, Thorsten, and Altun, Yasemin. Support Vector Machine Learning for Interdependent and Structured Output Spaces. *International Conference on Machine Learning*, 2004.

Uria, Benigno, Murray, Iain, and Larochelle, Hugo. A Deep and Tractable Density Estimator. *International Conference on Machine Learning*, pp. 467–475, 2014.

van den Oord, Aaron, Kalchbrenner, Nal, Espeholt, Lasse, Vinyals, Oriol, Graves, Alex, et al. Conditional Image Generation with Pixelcnn Decoders. *Neural Information Processing Systems*, pp. 4790–4798, 2016.

Venugopalan, Subhashini, Rohrbach, Marcus, Donahue, Jeffrey, Mooney, Raymond, Darrell, Trevor, and Saenko, Kate. Sequence to Sequence-Video to Text. *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4534–4542, 2015.

Vilnis, Luke, Belanger, David, Sheldon, Daniel, and McCallum, Andrew. Bethe Projections for Non-Local Inference. *Conference on Uncertainty in Artificial Intelligence*, 2015.

Vincent, Pascal. A Connection Between Score Matching and Denoising Autoencoders. *Neural Computation*, 2011.

Vinyals, Oriol, Kaiser, Lukasz, Koo, Terry, Petrov, Slav, Sutskever, Ilya, and Hinton, Geoffrey. Grammar As a Foreign Language. *CoRR.*, 2014.

Viterbi, Andrew. Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm. *IEEE transactions on Information Theory*, 13(2): 260–269, 1967.

Wainwright, Martin J and Jordan, Michael I. Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends® in Machine Learning*, 1 (1-2):1–305, 2008.

Wang, Shenlong, Schwing, Alex, and Urtasun, Raquel. Efficient Inference of Continuous Markov Random Fields with Polynomial Potentials. *Neural Information Processing Systems*, 2014.

Wang, Shenlong, Fidler, Sanja, and Urtasun, Raquel. Proximal Deep Structured Models. *Neural Information Processing Systems*, 2016.

Weiss, David J and Taskar, Benjamin. Structured Prediction Cascades. *International Conference on Artificial Intelligence and Statistics*, pp. 916–923, 2010.

Welling, Max and Teh, Yee W. Bayesian Learning Via Stochastic Gradient Langevin Dynamics. *International Conference on Machine Learning*, pp. 681–688, 2011.

Welling, Max, Rosen-Zvi, Michal, and Hinton, Geoffrey E. Exponential Family Harmoniums with an Application to Information Retrieval. *Neural Information Processing Systems*, 4:1481–1488, 2004.

Wichrowska, Olga, Maheswaranathan, Niru, Hoffman, Matthew W, Colmenarejo, Sergio Gomez, Denil, Misha, de Freitas, Nando, and Sohl-Dickstein, Jascha. Learned Optimizers That Scale and Generalize. *International Conference on Machine Learning*, 2017.

Wick, Michael, Rohanimanesh, Khashayar, Bellare, Kedar, Culotta, Aron, and McCallum, Andrew. Samplerank: Training Factor Graphs with Atomic Gradients. *International Conference on Machine Learning*, pp. 777–784, 2011.

Williams, Ronald J. Simple Statistical Gradient-Following Algorithms for Connectionist Reinforcement Learning. *Machine learning*, 8(3-4):229–256, 1992.

Wiseman, Sam and Rush, Alexander M. Sequence-To-Sequence Learning As Beam-Search Optimization. *Empirical Methods in Natural Language Processing*, 2016.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V, Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, et al. Google's Neural Machine Translation System: Bridging the Gap Between Human and Machine Translation. *arXiv preprint arXiv:1609.08144*, 2016.

Xiao, Lin. Dual Averaging Methods for Regularized Stochastic Learning and Online Optimization. *Journal of Machine Learning Research*, 11:2543–2596, 2010.

Xu, Kelvin, Ba, Jimmy, Kiros, Ryan, Cho, Kyunghyun, Courville, Aaron, Salakhudinov, Ruslan, Zemel, Rich, and Bengio, Yoshua. Show, Attend and Tell: Neural Image Caption Generation with Visual Attention. *International Conference on Machine Learning*, pp. 2048–2057, 2015.

Xu, Yuehua, Fern, Alan, and Yoon, Sung Wook. Discriminative Learning of Beam-Search Heuristics for Planning. *IJCAI*, pp. 2041–2046, 2007.

Yedidia, Jonathan S, Freeman, William T, and Weiss, Yair. Understanding Belief Propagation and Its Generalizations. *Exploring artificial intelligence in the new millennium*, 8:236–239, 2003.

Younes, Laurent. Parametric Inference for Imperfectly Observed Gibbsian Fields. *Probability theory and related fields*, 82(4):625–645, 1989.

Yu, Hsiang-Fu, Jain, Prateek, Kar, Purushottam, and Dhillon, Inderjit S. Large-Scale Multi-Label Learning with Missing Labels. *International Conference on Machine Learning*, 2014.

Zhai, Shuangfei, Cheng, Yu, Lu, Weining, and Zhang, Zhongfei. Deep Structured Energy Based Models for Anomaly Detection. *International Conference on Machine Learning*, pp. 19–24, 2016.

Zhou, Jie and Xu, Wei. End-To-End Learning of Semantic Role Labeling Using Recurrent Neural Networks. *Association for Computational Linguistics*, pp. 1127–1137, 2015.

Zweig, Geoffrey and Padmanabhan, Mukund. Exact Alpha-Beta Computation in Logarithmic Space with Application to MAP Word Graph Construction. *Sixth International Conference on Spoken Language Processing*, 2000.