April 2014

Deep Learning via Stacked Sparse Autoencoders for Automated Voxel-Wise Brain Parcellation Based on Functional Connectivity

Céline Gravelines
The University of Western Ontario

Supervisor
Mark Daley
The University of Western Ontario

Graduate Program in Computer Science

A thesis submitted in partial fulfillment of the requirements for the degree in Master of Science

DEEP LEARNING VIA STACKED SPARSE AUTOENCODERS FOR AUTOMATED VOXEL-WISE BRAIN PARCELLATION BASED ON FUNCTIONAL CONNECTIVITY

(Thesis format: Monograph)

by

Céline Gravelines

Graduate Program in Computer Science

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science

The School of Graduate and Postdoctoral Studies
The University of Western Ontario
London, Ontario, Canada

Abstract

Functional brain parcellation – the delineation of brain regions based on functional connectivity – is an active research area lacking an ideal subject-specific solution independent of anatomical composition, manual feature engineering, or heavily labelled examples. Deep learning is a cutting-edge area of machine learning on the forefront of current artificial intelligence developments. Specifically, autoencoders are artificial neural networks which can be stacked to form hierarchical sparse deep models from which high-level features are compressed, organized, and extracted, without labelled training data, allowing for unsupervised learning. This thesis presents a novel application of stacked sparse autoencoders to the problem of parcellating the brain based on its components' (voxels') functional connectivity, focusing on the medial parietal cortex. Various depths of autoencoders are investigated, yielding results of up to (68 ± 3)% accuracy compared with ground truth parcellations using Dice's coefficient. This data-driven functional parcellation technique offers promising growth to both the neuroimaging and machine learning communities.

Keywords

Deep Learning, Machine Learning, Unsupervised, Sparse, Stacked, Autoencoders, Brain Parcellation, Segmentation, fMRI, Functional Connectivity.

Acknowledgments

I wish to express my sincere thanks to my supervisor Mark Daley for encouraging me to pursue my Master's degree and for the constant guidance, advice, and support along the way.

Table of Contents

List of Tables

List of Figures

List of Appendices

Chapter 1

1 Introduction and Literature Review

Artificial intelligence has developed significantly in recent years as machine learning and classification algorithms have become more sophisticated and powerful. Currently, one of the most popular and promising approaches to machine learning is deep learning (Ng, Ngiam, Foo, Mai, & Suen, 2013). Deep learning involves learning the hierarchical structure of data by initially learning simple low-level features which are in turn used to successively build up more complex representations, capturing the underlying regularities of the data. Stacked sparse autoencoders are a type of deep network capable of achieving unsupervised learning – a type of machine learning algorithm which draws inferences from the input data and does not use labelled training examples. Sections 1.1 – 1.6 discuss the history and theory behind deep learning.

Functional brain parcellation is the task of delineating the brain based on functional connectivity. This active area of neuroscientific research lacks an ideal standard protocol as the current techniques assume unrealistic similarity of anatomic composition among subjects, depend on manual feature engineering, or heavily labelled examples (further discussed in section 1.7). Therefore, the motivation of this thesis will be to develop a solution for unsupervised brain parcellation using a deep network to automatically delineate brain regions based on functional connectivity.

1.1 History of Artificial Neural Networks

Research of artificial neural networks was developed for two distinct areas of study: to model biological processes in the brain, and to investigate the application of neural networks to artificial intelligence (AI). While there are parallels between the two domains, neural networks used in AI are generally simplified models of biological neural processing and the degree to which artificial neural networks imitate actual brain function is not fundamentally relevant to the machine learning processes presented.

Artificial neural networks were initially developed by McCulloch and Pitts in 1943. A neuron is used to represent a computational unit that takes some input values and produces an output, based on an activation function (McCulloch & Pitts, 1943). For example, a binary threshold activation function (commonly known as the Heaviside step function) is an all-or-nothing approach which will result in the neuron "firing" (outputting results) if the output value is 1, while it will not fire if the output value is 0. McCulloch and Pitts developed this neural model using electric circuits and proposed that modelling neurons using this binary threshold activation function mimics first order logic sentences. That is, providing a neuron with different combinations of 0 (false) and 1 (true) inputs can accurate represent AND, OR, NOT, NAND, and NOR statements at the output. However, it was later shown that the single model (one neuron) could solve neither the exclusive-or (XOR) nor exclusive-nor (XNOR) (Minsky & Papert, 1969).

In the late 1940s, Donald Hebb improved the neural network by proposing that neural pathways are strengthened the more they are used, pointing out that this concept is fundamental to the process of human learning and memory (Hebb, 1949). Thus, McCulloch and Pitts's neuron model was altered to account for this learning process. The solution involved assigning *non-identical weights* to each input. Consequently, an input of 1 may possess more or less weight, relative to the total threshold. Based on these new developments, Frank Rosenblatt introduced the perceptron, a neural model took n input values (a_1 to a_n) and n corresponding weights (w_1 to w_n) (Rosenblatt, 1962). Each of the input values is weighted and summed at the node which only "fires" (outputs) if the threshold value is reached. While the perceptron model offered hope for artificial neural networks, it was later shown that the perceptron could not be trained to recognize many classes of patterns as this single layer network was only capable of learning linearly separable patterns. Added the fact that XOR and XNOR statements could not be represented, the 1960s to 1990s saw a drastic decline of the use neural networks for practical machine learning (Larochelle, Bengio, Louradour, & Lamblin, 2009) as large single layer neural networks were inefficient and ineffective at learning tasks (Minsky & Papert, 1969).

1.2 Modern Artificial Neural Networks

Based on the perceptron, Figure 1 shows the simplest neural network consisting of just a single neuron (computational unit) that takes n input values, $x_1, x_2, \ldots, x_n$, and a bias intercept term which is a constant term (not included in the input) used to shift the activation function to the left or right.

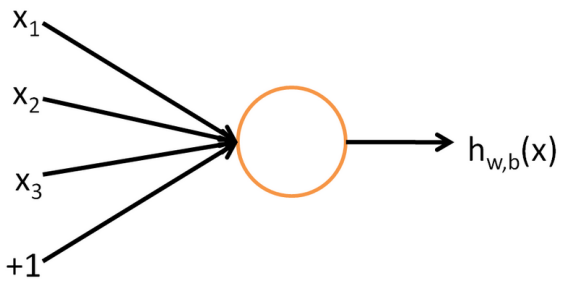

Figure 1: A single neuron with 3 input values, a bias term, and 1 output hypothesis value (Ng, Ngiam, Foo, Mai, & Suen, 2013)

Given an input, x, the network outputs a hypothesis $h_{w,b}(x)$ where W and b are weight and bias parameters which can be learned from the input data (Ng, Ngiam, Foo, Mai, & Suen, 2013). Thus, this input data acts as a training set to train the network to learn these parameters. This neuron's hypothesized output is defined as

$$h_{w,b}(x) = f\left(\sum_{i=1}^{n} W_i x_i + b\right) = a \qquad \text{Equation 1}$$

where W_i and x_i represent the weight connection and input to the i-th of n units, respectively. In addition, $f: \mathbb{R} \to \mathbb{R}$ is equal to the activation function, a, which will correspond to the sigmoid function used to scale the outputs to a range of [0,1]:

$$f(z) = \frac{1}{1 + e^{-z}} \qquad \text{Equation 2}$$

The sigmoid function is an alternative to the threshold function used in earlier literature.

A more complex neural network is formed by joining many neurons together, so that the output of one neuron can be the input to any another neuron in the network. These neurons can be organized into layers, as shown in Figure 2.

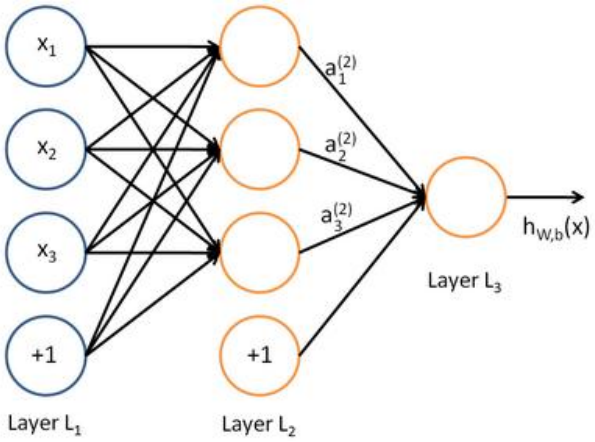

Figure 2: A neural network organized into 3 layers. L_1 represents the input layer, L_2 is a hidden layer, and L_3 is the output layer. (Ng, Ngiam, Foo, Mai, & Suen, 2013)

In Figure 2, the input values are also denoted as neuron-like units. The leftmost layer of the network is the input layer. The +1 bias unit corresponds to the intercept term. This network has 3 input units (not counting the bias). The rightmost layer is the output layer which happens to only have 1 output unit. Meanwhile, the middle layer is referred to as a hidden layer, as its values are not visible in the input or output data. There are 3 hidden units in this network. The connections between each unit (excluding the bias units) each represent a weight connection. A matrix, W^l, can be composed of all the weighted connections between units of the adjacent layers, l and $l + 1$. Therefore the parameters of the network are (W, b) where $(W, b) = (W^1, b^1, W^2, b^2)$ for the 3 layer model. The output of each unit in layer l is represented by an activation vector, a^l, which represents learned features from that layer, analogous to the hypothesis $h(x)$.

1.3 Deep Learning

When tasked with a problem to solve, humans often decompose the problem into smaller, easier-to-solve sub-problems at different levels of representation. Humans are able to

inadvertently exploit intuition and describe concepts in hierarchical ways, based on multiple levels of abstraction. For example, an individual seeks to identity an image. Taking the entire image into account, the individual looks specifically at the important features of the image. The individual sees a human form in the image, notices facial hair, body structure, and clothing and determines that the image is a man. That is, the individual has identified the image by breaking it down into smaller features, such as "has beard", "has broad shoulders", "is wearing a suit" and then determined a classification for the image. The problem is broken down on many levels. Without much conscious thought, humans look at much smaller features of images, such as lines, curves, and edges to determine the higher-level features. These numerous highly-varying, non-linear features organized into layers are what constitute a *deep* network (Bengio, Lamblin, Popovici, & Larochelle, 2007).

Deep learning generally refers to learning models which use feature hierarchies with many layers. Thus, a deep artificial neural network is a multi-layer network composed of input and output layers, in addition to numerous hidden layers between the input and output. Those hidden layers are composed of hidden (or "latent") units that can be used to describe underlying features of the data. Figure 3 depicts a common facial recognition task, in which the input layer represents the pixels of the image while the output is the corresponding identity (or classification) of the face, while the hidden layers can represent low-level features, such as edges and shapes, to high-level features, such as "big eyes" or "small nose". Learning the structure of a deep architecture aims to automatically discover these abstractions, from the lowest to highest levels. Favourable learning algorithms would be unsupervised, depending on minimal human effort, while allowing the network to discover these latent variables on its own, rather than requiring a pre-defined set of all possible abstractions. The ability to achieve this task while requiring little human input is particularly important for higher-level abstractions as humans are often unable to explicitly identify potential hidden, underlying factors of the raw input (Bengio, 2009). Thus, the power to automatically learn important underlying features fuels the popularity of deep architectures as the wide applications of deep machine learning become increasingly attainable.

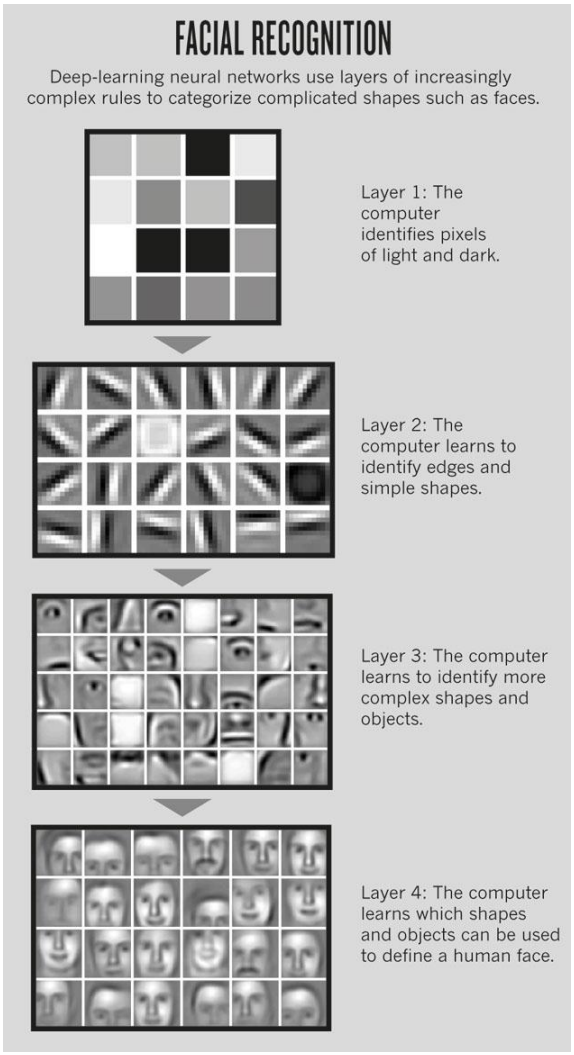

Figure 3: Facial recognition using a deep neural network, demonstrating the multiple levels of abstraction of the task, from low dimensional pixels to complex shapes and objects defining a human face. (Jones, 2014)

Deep networks were introduced in the 1980s in Fukushima's *Neocognitron* (Fukushima, 1980) which presented a hierarchical multi-layered neural network used for pattern recognition, such as the recognition and classification of handwritten characters. However, early deep multilayer networks were often believed to be too difficult to train and they were empirically found to be less effective than networks with only one or two hidden layers (Tesauro, 1992). Consequently, deep learning was not investigated much in machine learning literature.

Sepp Hochreiter's 1991 thesis identified the issue as "the vanishing gradient problem" which had led to this major failure of deep artificial neural network (Hochreiter, 1991). The problem stemmed from the fact that as a layer of neural network eventually learned a task reasonably well, the learned features were not successfully propagated to successive layers in the network. The deeper layers did not receive information in order to account for these new learned features. Backpropagation via gradient descent had become a crucial step of training deep networks (Werbos, 1974). However, due to the lack of computing power available at the time, it was thought to be too slow of an algorithm for practical training of neural networks, resulting in simple methods such as Support Vector Machines (SVMs) monopolizing the field (Mourão-Miranda, Bokde, Born, Hampel, & Stetter, 2005). SVMs are sufficient models for basic, linearly separable data, but lacked the capability of neural networks to learn for complex, non-linear data.

In 1992, Hochreiter's mentor, Jürgen Schmidhuber, attempted to solve the problem associated with deep networks by organizing a multi-level deep hierarchy which could be effectively pre-trained *one level at a time* via random initialization and unsupervised learning, followed by a supervised backpropagation pass for fine-tuning (Schmidhuber, 1992). This method allows each level of the hierarchy to learn a compressed representation of the input observation which is in turn fed into the next level as the successive input. The "vanishing gradient problem" was solved, but it was not until 2006 that deep learning regained and surpassed its original popularity.

By the mid-1990s to early 2000s, the standard learning strategy for deep neural networks often involved *randomly* initializing the weights of the network (pre-training), followed by backpropagation via gradient descent. However, this method has been empirically shown to find poor solutions for networks with multiple hidden layers (Larochelle, Bengio, Louradour, & Lamblin, 2009). The computational power to arrive at satisfactory results was still out of reach. As a result, shallow architectures continued to be the predominant structure for machine learning algorithms. These shallow architectures consist of only two-to-three levels of data-dependent computational elements. For example, kernel machines, such as Support Vector Machines (SVMs), and single-layer neural networks were popular learning algorithms that made use of shallow architectures.

However, it has been shown that deep architectures can be significantly more efficient (sometimes exponentially) than shallow architectures with respect to computational elements and parameters needed to fully represent some functions (Bengio, Lamblin, Popovici, & Larochelle, 2007). Shallow architectures are seriously flawed in their inefficiency regarding the number of computational units (hidden units), and consequently require a large supply of training examples (Bengio & LeCun, 2007). On the contrary, the non-linearity of deep architectures allows highly-varying functions to be represented compactly, requiring fewer parameters (Bengio, Lamblin, Popovici, & Larochelle, 2007). While it is not the case that deep architectures are always optimal over shallow architectures (Salakhutdinov & Murray, 2008), complex high-dimensional problems with sufficient data to capture the complexity are solved more efficiently and accurately when adopting a deep architecture for learning (Larochelle, Bengio, Louradour, & Lamblin, 2009; Lee, Laine, & Klein, 2011). Thus, determining efficient learning algorithms for deep architectures became a popular area of interest in the machine learning field.

While deep architectures were promising, the issue remained that many negative experimental results were suggesting that gradient-based training of randomly initialized supervised deep neural networks easily got stuck in local minima or plateaus (Bengio, Lamblin, Popovici, & Larochelle, 2007) and that it becomes increasingly difficult to find a good generalization as the architecture got deeper (more layers) (Larochelle, Bengio, Louradour, & Lamblin, 2009).

The algorithm popularized by Geoff Hinton in 2006 revolutionized deep learning as it employs a deep architecture and introduces a fast, greedy learning method used to construct multilayer directed networks, layer-by-layer (Hinton, Osindero, & Teh, 2006). As a result, this algorithm offers a solution to the issue of poor optimization which originally stemmed from random initialization of the network's parameters. The algorithm can quickly find a good set of parameters for the network, even in models with millions of parameters and many hidden layers. The training method also involves a fine-tuning component which is capable of learning a very successful generative model, outperforming discriminative methods and yielding state-of-the-art results of

classification of hand-written digits (Hinton, Osindero, & Teh, 2006). A major breakthrough of this generative model is that it can easily interpret the distributed representations in the hidden layers of the deep network.

1.4 Restricted Boltzmann Machines and Deep Belief Networks

The network used by Hinton is composed of several stacked restricted Boltzmann machines (RBMs). RBMs (originally named "Harmoniums") were first introduced in 1986 by Paul Smolensky. A single RBM consists of a layer of unconnected "visible" input units, x, that have undirected, symmetrical connections with another single layer of hidden units, h. The network is fully connected between the two layers, yet no units within the same layer are connected to one another, forming a bipartite graph (Smolensky, 1986). Each connection between units of the two layers has an associated weight that must be learned, represented by a weight matrix, W (Figure 4). Data can be generated from an RBM by initializing a random state in one of the layers and then performing alternating Gibbs sampling, a Markov Chain Monte Carlo algorithm which approximates the distribution based solely on previous states. Given the current states of the units in one layer, all the units of the other layer are then updated simultaneously. This process is repeated until the entire system is sampling from its equilibrium distribution (Hinton, Osindero, & Teh, 2006). RBMs are generative models, meaning that the training period results in a probability distribution of the training data being learned. When the model is later used for testing, it may encounter new, unfamiliar data, but the probability distribution can account for these previously unseen occurrences, yielding a likely, probabilistic output.

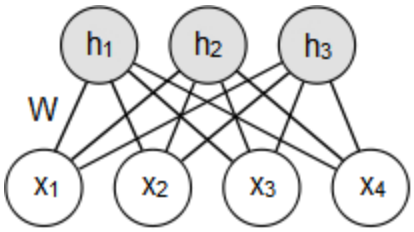

Figure 4: A restricted Boltzmann Machine with weighted connections between a layer of 4 visible input units, *x*, and a layer of 3 latent hidden units, *h*.

Hinton's proposed learning algorithm acts on several RBMs stacked on top of one another to form a generative model called a Deep Belief Network (DBN). The leftmost layers (or "lower layers" in some graphical representations) of the network are able to extract low-level features from the input, x. These lower level features are fed into the rightmost layers (or "upper" levels) which represent more abstract features and concepts which can explain the input observation. That is, the model initially learns simple concepts which it uses to build more abstract, higher-order concepts (Hinton, Osindero, & Teh, 2006; Larochelle, Bengio, Louradour, & Lamblin, 2009). To achieve this learning, the algorithm involves greedily pre-training one layer at a time, using unsupervised learning at each level in order to preserve information from the input, followed by a supervised fine-tuning pass through the entire network (Bengio, Lamblin, Popovici, & Larochelle, 2007) Once the stack of RBMs has been learned, the entire stacked DBN can be viewed as a single probabilistic model.

This algorithm is particularly interesting as it provides a method for high-level representations to be learned from low-level data, depending mostly on a large supply of unlabelled input and limited labelled data which is used to slightly fine-tune the model for the current task. This revolutionary learning algorithm for deep architectures eliminates the problem caused by random initialization leading to poor optimization solutions. This greedy layer-wise training strategy is a crucial tool for improved optimizations as it initializes the weights in a region near a good local minimum (greedily), resulting in internal representations that are high-level abstractions of the input, leading to better generalizations (Bengio, Lamblin, Popovici, & Larochelle, 2007).

The learning strategy is greedy in that optimization occurs by initializing the parameters of each layer near a good local minimum, independent of the other layers. Adjustments to the strength each neuron-like unit depends only on the states of the pre-synaptic and post-synaptic neurons (Hinton, Osindero, & Teh, 2006). The greedy layer-wise algorithm involves first learning the parameters between the first 2 layers, while assuming all the other parameters in the remainder of the network are frozen (tied; unchanging). Once the first set of parameters is learned, the algorithm freezes all layers except the parameters between the next 2 layers. This process continues until the parameters between every layer in the network are optimized.

There is no exact solution to the number of hidden layers required to create an optimal deep network, nor for the number of hidden units per layer. The most common approach is to validate the model by training and testing on various numbers of hidden layers and units. The optimal number of hidden layers in the network tends to depend on the nature of the input data, specifically how it is organized hierarchically in various levels of abstraction (Heaton, 2008). The decision of units used is crucial to the success of the model as it greatly influences the outcome. Too few hidden units results in underfitting, where the model is unable to adequately detect the structure in complex datasets. Meanwhile, too many hidden units increase the amount of training time and cause overfitting, where the information contained in the training set is not enough to train all the neurons (Heaton, 2008). Various pieces of literature provide different "rules of thumb" regarding the number of hidden units per layer where each layer should have 2/3 the number of input units (Heaton, 2008), between the number of input and output units (Blum & Rivest, 1992), twice the number of input units (Berry & Linoff, 1997), or 1/30 of the number of training cases (Heaton, 2008). However, there is no generally accepted support for these suggestions, so validation with test cases is frequently used to crudely determine the optimal number for the particular dataset.

Deep networks have developed as tools for a wide variety of applications including classification tasks (Bengio, Lamblin, Popovici, & Larochelle, 2007; Larochelle, Bengio, Louradour, & Lamblin, 2009), regression (Salakhutdinov & Hinton, 2008), dimensionality reduction (Hinton & Salakhutdinov, 2006), robotics (Hadsell, Erkan,

Sermanet, Scoffier, Muller, & LeCun, 2008), and natural language processing (Collobert & J. Weston, 2008).

Thus, the original idea of deep learning was not wrong. The earlier setbacks associated with deep architectures were results of not using enough data and being limited in computational power. Previous deep neural networks did not successfully learn because they were not given enough time. However, pre-training allows the network to build a model that is likely to succeed (rather than starting from randomly initialized parameters), resulting in less time required by the network to learn.

1.5 Training a Deep Model

Hinton, et al. (2006) provide an algorithm to pre-train each layer of the DBN using an unsupervised approach. This greedy layer-wise unsupervised learning algorithm first involves training the lower layer of the model with an unsupervised learning algorithm which yields some initial set of parameters for that first layer of the network. That output from the first layer is a reduced representation of the input. This output then acts as the input for the following layer which is similarly trained, resulting in initial parameters for that layer. Again, the output from this second layer is used as the input for the next layer until the parameters for each layer are initialized. The overall output of the network is delivered as the final activation vector.

Following this unsupervised pre-training phase of stacked layers, the entire network can then be fine-tuned in the opposite direction using backpropagation in this supervised learning phase. Backpropagation works by continually readjusting the weights of the connections between units of the network. The readjustment maintains the goal of minimizing the difference between the actual output vector of the network (the network's activation) and the desired output vector (the true target value). As the weights are adjusted, the internal hidden units better represent important features of the task domain which are not explicitly part of the input or output values. This ability to capture useful new features from the data makes backpropagation a more successful algorithm than

earlier, simpler techniques such as the perceptron's convergence procedure (Rumelhart, Hinton, & Williams, 1986).

1.6 Autoencoders

In artificial neural networks, backpropagation was developed for improved representation learning. The process of backpropagation involves readjusting the weights, depending the on the existence of some expected output. In the 1980s, autoencoders (also called "auto-associators") were introduced in order to perform backpropagation without a teacher (Bourlard & Kamp, 1988). That is, autoencoders offer a method of automatically learning features from unlabelled data, allowing for unsupervised learning.

An autoencoder is an artificial neural network that is able to be trained in a fully unsupervised way. In previous neural networks, labelled data were required to act as training examples essential to the backpropagation fine-tuning pass as those labels were used to readjust the parameters. However, autoencoders provide the opportunity to learn without this dependence on labelled data. An autoencoder neural network performs backpropagation by setting the target output values equal to the input values, and thus the autoencoder is trained to minimize the discrepancy between the data and *its reconstruction* (that is, the difference between the actual output vector and the expected output vector where that expected output is the same as the input vector). As a result, autoencoders are able to learn without a teacher.

An autoencoder consists of three or more layers: an input layer; some number of hidden layers which forms the encoding; and an output layer whose units correspond to the input layer. Hinton et al. defined an autoencoder as a nonlinear generalization of principal components analysis (PCA) which uses an adaptive, multilayer "encoder" network to transform the high-dimensional input data into a low-dimensional code, while a similar "decoder" network is able to recover the data from the code (Hinton & Salakhutdinov, 2006). That is, an autoencoder network with the same number of hidden units as input and output units would be a (linear) PCA model.

Since the outputs of the network are equal to the input, the autoencoder's goal is to learn an approximation of the identity function (Ng, Ngiam, Foo, Mai, & Suen, 2013). While this may seem like a trivial learning task, placing constraints on the network can reveal interesting structure of the data. An example of a constraint on the network is a limitation to the number of hidden units in the hidden layer, thus forcing the network to learn a compressed representation of the input. This method allows for the discovery of internal representations of the data that rely on fewer intermediate features. For example, for a facial recognition task, each pixel of the image may be represented at the input layer. That data is compressed in the hidden layer into features such as "small mouth" or "wide eyes." That is, the input data of the face can be described using less data than is actually given in the image. That compressed data can then be uncompressed in order to re-represent the input data at the output layer, allowing the facial image to be reconstructed entirely from the learned features.

Rather than limiting the number of hidden units, an alternative constraint to the network could be the *sparsity* of hidden units that are activated. Sparsity is a useful constraint when the number of hidden units is large (even larger than the number of input values) that can allow for the discovery of interesting structure of the data (Ng, Ngiam, Foo, Mai, & Suen, 2013). A sparse autoencoder has very few neurons that are active. A neuron in an artificial neural network is informally considered "active" (or "firing") if its output value is close to 1, while it is considered "inactive" if its output value is close to 0. The concept of creating a sparse autoencoder involves constraining the neurons to be inactive most of the time (Ng, Ngiam, Foo, Mai, & Suen, 2013). As a result, even with many hidden units, the data is constrained, forcing the network to learn the important features of the data in order to reconstruct it.

With deep architectures, learning-feature hierarchies are formed when lower-level features are learned and used to compose higher levels of the hierarchy. Similar to how RBMs are stacked to form DBNs, this deep approach can be extended to non-linear autoencoders to form a stacked autoencoder network (Bengio, Lamblin, Popovici, & Larochelle, 2007; Larochelle, Bengio, Louradour, & Lamblin, 2009). A stacked sparse autoencoder is a neural network composed of multiple layers of sparse autoencoders in

which the outputs of each layer is fed into the inputs of the successive layer (Ng, Ngiam, Foo, Mai, & Suen, 2013). However, optimizing the weights of autoencoders is a challenging task. Large initial weights cause autoencoders to find poor local minima, while small initial weights result in tiny gradients in the early layers, proving it infeasible to train many-layered autoencoders in this condition (Hinton & Salakhutdinov, 2006). However, if the initial weights are already close to a good solution, optimization techniques, such as gradient descent, work well. It follows that employing a similar "greedy layer-wise" learning algorithm (which was used for DBNs) to stacked autoencoders is an effective method of pre-training the network of a deep autoencoder (Hinton & Salakhutdinov, 2006).

To see the structural composition of this stacked autoencoder, first consider the single layer autoencoder network in Figure 5. Let $W_{i,j}^{l}$ represents the weighted connection between the unit j of layer l, and unit i of layer $l+1$, while b_i^l is the bias associated with unit i in the layer $l+1$. These values ($W_{i,j}^{l}\ and\ b_i^l$) are organized in the matrices for each unit and layer of the network. Considering a single independent layer model (i.e.: a single autoencoder of the stack, composed of an input, output, and hidden layer), the W^1 matrix is composed of the weighted connection between the input data and the hidden units, while W^2 contains the weighted connection between the hidden units and the output. Similarly, b^1 represents the biases from the bias unit in the input layer to each hidden unit, while b^2 represents the bias from the bias unit in the hidden layer to the output layer. That is, each single layer module has a set of parameters $(W, b) = W^1, b^1, W^2, b^2$ representing the weights and biases from the input units to the hidden units, and from the hidden units to the output units, as shown in Figure 5.

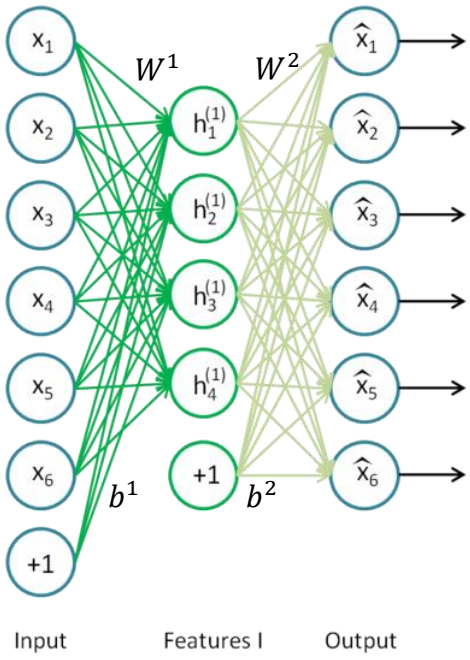

Figure 5: A single layer autoencoder which will be the first in the stack. The weighted connections from the input layer, x, join to the hidden layer, h, represented by a matrix, W^1, and the weights from the hidden layer, h, to the output units, $\hat{x}$, (the representation of the input) are stored in the matrix W^2 (Ng, Ngiam, Foo, Mai, & Suen, 2013).

Note that the output units of the single autoencoder in Figure 5 will not actually present in the *stacked* autoencoder. They are simply used for the training of the single layer. Rather, the activation vector which represents the features detected from the hidden layer are used as input to the following autoencoder which is added to the stack, as seen in Figure 6. That is, the hidden units of the first autoencoder can be considered the visible input units to the next autoencoder. As expected, the output units of the second autoencoder in the stack are a representation of the hidden units of the first autoencoder (i.e.: the current autoencoder's input units). Recalling that each single autoencoder has a set of parameters $(W, b) = W^1, b^1, W^2, b^2$ representing the weights and biases from the input units to the hidden units, and from the hidden units to the output units, let $W^{k,1}, b^{k,1}, W^{k,2}, b^{k,2}$ represent the parameters W^1, b^1, W^2, b^2 for the k-th autoencoder.

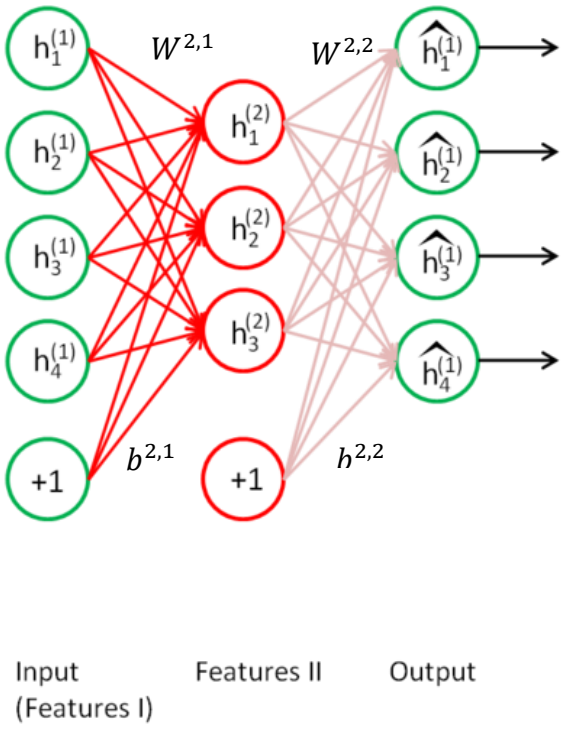

Figure 6: The second autoencoder in the stack. The output from the first autoencoder's hidden feature layer (Figure 5) is used as input to the second autoencoder (Ng, Ngiam, Foo, Mai, & Suen, 2013).

To perform this greedy layer-wise pre-training, consider a stacked autoencoder composed to n_l layers. The stacked autoencoder can be greedily pre-trained in order to initialize the parameters of this deep network, in a similar way as the DBN was trained: train the first layer using the raw input to obtain parameters $W^{1,1}, b^{1,1}, W^{1,2}, b^{1,2}$ for the first autoencoder in the stack, while the rest of the parameters in the remainder of the network remain fixed. With these initialized parameters, the raw input can be transformed into a vector, a^2, consisting of the activations (learned features) of the hidden units. The autoencoder is able to map the input directly to the hidden layer using a parameterized closed-form equation called an *encoder* (Ng, Ngiam, Foo, Mai, & Suen, 2013): Noting that a^1 represents the initial input to the network, *x*, the activation vector of a layer, *l*, is given as

$$a^l = f(z^l) \qquad \text{Equation 3}$$

where $f(\cdot)$ is the sigmoid function, while the step is run forward to acquire the following variable such that

$$z^{l+1} = W^{l,1}a^{l} + b^{l,1}. \qquad \text{Equation 4}$$

In other words, z^{l+1} denotes the total input to the layer $l + 1$, including the bias, which is calculated based on the previous activation vector. This *encoding step* is thus able to transform the high-dimensional input data into a lower-dimensional code.

The *decoding steps* are similarly used to map from these learned features from the hidden space back to the reconstruction of that input. The decoder is a parameterized closed-form equation used to "undo" the encoding function where

$$a^{n+1} = f(z^{n+1}) \qquad \text{Equation 5}$$

$$z^{n+l+1} = W^{n-l,2}a^{n+l} + b^{n-l,2}. \qquad \text{Equation 6}$$

Thus the decoder recovers the high-dimensional data from the code in order to create the reconstruction of the input (Ng, Ngiam, Foo, Mai, & Suen, 2013).

This output activation vector, a^2, can then be used as input to train the second layer, yielding the parameters $W^{2,1}, b^{2,1}, W^{2,2}, b^{2,2}$, and thus a new set of features given by the activation vector is provided. This procedure is repeated for all subsequent layers of the network, as the output of each new layer serves as input into the next layer, while the remaining layers are frozen. Finally, the output activation vector a^{n_l} from the final layer n_l gives the activation of the deepest layer of hidden units which, for the autoencoder, is the reconstruction of the input vector.

1.7 Brain Parcellation

While the fixed anatomical structure of the brain has been thoroughly studied and documented, the functional connectivity of the brain is a dynamic, elusive, and less-understood domain. Unlike other organs, such as the heart, which have dynamic structure

with well-defined functions, the brain's static structural networks gives rise to dynamic functional connectivity which performs a vast array of tasks, facilitating perception, cognition, and actions (Park & Friston, 2013). The mapping of functional connectivity through the analysis of resting-state fMRI was first demonstrated in 1995 (Biswal, Zerrin Yetkin, Haughton, & Hyde, 1995), leading neuroscientists to examine the relationship between resting-state connectivity and the functional organization of the brain, associated with processes such as language, motor skills, and memory. To better understand how such diverse, complex function can arise from the brain's static neuronal architecture, researchers aim to accurately *parcellate* individual brains based on its functional connectivity.

Brain parcellation refers to the delineation of structural and functional regions of the brain (Lee, Laine, & Klein, 2011). It is an important and challenging task to establish these correspondences across structural and functional brain images. The ability to parcel out functional brain regions non-invasively would enhance the quality of various experimental investigations, leading to more accurate across-subject comparisons of independent functional regions (Cohen, et al., 2008). Successful brain parcellation provides great benefits to clinical neuroscience and cognitive psychology as the relations between structure and function become better defined.

Parcellation can be performed by analyzing functional images of the brain. Functional neuroimaging allows researchers to study the brain's dynamic functional relationships in real-time, based on the current state of the brain's activity, rather than simply the fixed anatomical structure of the brain. Current modes of acquiring functional neuroimaging include fMRI, PET (invasive), fNIRS, EEG and MEG (poor spatial resolution).

Functional magnetic resonance imaging (fMRI) permits researchers to associate functional activity with specific neuroanatomical regions of the brain with a high degree of spatial resolution. This non-invasive tool detects changes in blood oxygenation levels and changes in blood flow resulting from neural activity. When a brain region is more active, it consumes more oxygen, resulting in increased blood flow to the active area (Devlin, 2008). As a result, fMRI are able to map parts of the brain which are involved in

a particular mental process. The 3-dimensional brain image is composed of units called voxels, each representing a cube of brain tissue. A voxel is a 3-dimensional building block of a 3-dimensional fMRI volume, analogous to a pixel being a 2-dimensional building block of a 2-dimensional image on a screen. Advances in fMRI analysis have led to the discovery of biomarkers and diagnostic indicators for a variety of psychological and neurodegenerative disorders (Henley, Bates, & Tabriz, 2005). The ability to accurately label various brain regions based on function is imperative to understanding the relationships between brain function and structural appearance. A full understanding of these correspondences can be applied to understanding the underlying causes of various metabolic diseases and fine-scale lesions in the brain, potentially leading to major advancements in treating and curing diseases such as Alzheimer's disease (Henley, Bates, & Tabriz, 2005).

Many experiments have been developed which use functional neuroimaging to identify brain regions associated with various cognitive processes. Oftentimes, these investigations study which brain areas are activated while some task is being performed (relative to other tasks). However, these experiments are based on *a priori* hypotheses since those experiments are designed for the purpose of acquiring this knowledge (Mourão-Miranda, Bokde, Born, Hampel, & Stetter, 2005). Ideally, a natural, *a posteriori* method of searching for functional localization can be developed and implemented for general use.

There currently exists no standard protocol for such parcellation of the brain into discrete regions (Bohland, Bokil, Allen, & Mitra, 2009). A variety of software parcellation packages are available, many of which depend on a labelled brain atlas which associate physical structure with function. However, each atlas is often developed using a parcellation protocol which has only been applied to a single individual or it is simply an amalgamation of various brains (Lee, Laine, & Klein, 2011), thus these software packages are often inconsistent as they offer different descriptions of neuroanatomical organization. An alternate approach to parcellation involves researchers co-registering brain images to each other, using a template or labelled brain atlas of the same imaging modality. However, these registration methods assume image similarity is a stand-in for

anatomical similarly which is often not the case (Lee, Laine, & Klein, 2011). Recent large-scale efforts have been established to manually apply standard brain parcellation techniques to many different brain images (www.braincolor.org/protocols), but manual parcellation is tedious, time-consuming, and heavily dependent on expertise the area (Lee, Laine, & Klein, 2011). Automatic brain parcellation is a significantly more reasonable approach than manual, yet the intrinsic variability of the human brain makes the task of defining consistent correspondences across brains a challenging one for both humans and computers (Lee, Laine, & Klein, 2011).

The ability to parcellate functional regions of the brain without relying on standard mappings allows for subject-specific parcellation, providing better diagnoses and understanding of unhealthy or abnormal brains. For example, stroke patients suffer permanent damage resulting in lesions in the brain and they subsequently lose the ability to perform corresponding functions which were originally attributed to the now-damaged area. However, some patients are able to recover the lost function over months or years. While the irreversible damage is still present, this re-learning of the function indicates that another area of the brain has learned to compensate and provide the functionally lost (Schmah, Hinton, Zemel, Small, & Strother, 2008). Therefore, a subject-specific functional parcellation technique which relies only on the functional connectivity (and not anatomic structure) would allow specialists to see the area of the brain providing the recovering function.

To help alleviate the challenges stemming from subject specific parcellations, automated parcellation schemes have been developed which do not depend on the use of a brain atlas. Automatic learning techniques such as Support Vector Machines (Mourão-Miranda, Bokde, Born, Hampel, & Stetter, 2005), Principal Component Analysis (Zhong, Wang, Lu, Zhang, Jiao, & Liu, 2009), independent component analysis (McKeown & Sejnowski, 1998) or single-layer neural networks are used to learn the appropriate neuroanatomical categorization of fMRI image data. However, a major drawback of current automatic brain parcellation strategies is that they typically depend on algorithms with shallow architectures which have been shown to be limited and non-optimal when

learning complex high-dimensional features (Bengio & LeCun, 2007). In addition, these techniques heavily rely on labelled training examples data for supervised learning.

Data-driven clustering algorithms offer another approach to partition the brain into distinct functional regions. Kelly et al. (2012) present a voxel-wise classification of brain imaging data into natural subsets (clusters) based on intrinsic functional connectivity using clustering algorithms. The algorithm used assigns each unit of data (each voxel) to the same cluster, given that the voxel is more similar to voxels assigned to that particular cluster than any other cluster (Kelly, et al., 2012). The similarity between all pairs of voxels of intrinsic functional connectivity patterns used for this partition can be quantified using η^2. The η^2 metric is used to measure effect size, which represents the strength of the relationship between two variables. When used to evaluate the similarity (as a ratio of variance) of each voxel against every other voxel, the value for each pair ranges between 0 (no agreement) and 1 (identical). Cohen et al. (2008) introduced the η^2 statistic for the purpose of delineation based on functional connectivity, suggesting that it provides a better measure of similarity between two images than spatial correlation because it accounts for differences in scaling and offset between images, while correlation does not take these factors into account (Cohen, et al., 2008). The η^2 matrix can be calculated as

$$\eta^2 = 1 - \frac{\sum_{i=1}^{n}[(a_i - m_i)^2 + (b_i - m_i)^2]}{\sum_{i=1}^{n}[(a_i - M)^2 + (b_i - M)^2]} \qquad \text{Equation 7}$$

where a_i and b_i are the values of the voxel at location i in the maps a and b (i.e.: a correlation matrix), m_i is the mean value of a_i and b_i (i.e.: $(a_i + b_i)/2$), and M is the grand total mean of across all voxels. The k-means clustering algorithm uses this η^2 statistic to measure similarity in order to partition each voxel into a cluster which represents a functional brain region. The algorithm was used to delineate the functional areas of the brain's insula – a component of the brain used for tasks including sensory, somatic, cognitive, and emotional processing (Kelly, et al., 2012).

Most current automatic partitioning techniques continue to depend on heuristic manual feature engineering (Lee, Laine, & Klein, 2011) which is costly, error-prone, and impractically dependent on human expertise in the problem domain, thus compromising the generalizability of the model. For example, while k-means clustering is relatively effective approach to partitioning, it depends on human expertise and interpretation of the problem domain in order to select a k-value to represent the number of clusters a priori. K-means clustering discriminates and is biased towards the training data set. For example, consider a k-means algorithm set to organize the data into 5 clusters. If an input that should (ideally) be partitioned into some 6^{th} cluster is encountered, it will be classified incorrectly. K-means clustering can be biased in this way and it often does not generalize well, especially for more complex data sets. Additionally, k-means clustering is very sensitive to initial conditions, thus with different starting conditions, the same k-means algorithm may result in different partitions of the same data set. Furthermore, while deep learning can be regarded as high-level feature-extraction, k-means clusters can be a feature-extractor as well. However, k-means is conceptually equivalent to a single layer of a deep network and thus suffers the disadvantages associated with shallow architectures.

In general, the drawbacks of current automated brain parcellation are that they 1) employ algorithms with shallow architectures, 2) they are based on heuristic manual feature engineering, or 3) they assume the validity of the underlying feature model (Lee, Laine, & Klein, 2011). Lee et al. (2011) have attempted to account for these shortcomings by approaching the task of brain parcellation via deep learning. Specifically, convolutional deep neural networks are used: Training of the deep network is initiated by performing a forward propagation pass. Two-dimensional fMRI images were broken up into 28 by 28 image sites where context-aware feature learning is used to obtain the parcellation labels with the greatest probability, composing the convolutional layers of the network. The output of these convolutional layers are fed into simple subsampling (pooling) layers in order to reduce the computational expense of the model, while introducing scale invariance of the learned features. The output nodes of the network are then compared with expected labels provided by human experts. The errors can be backpropagated

through the network to iteratively fine-tune the parameters. The parcellation is validated by the Dice coefficient,$D_{c,}$ which is used to measure the agreement between the expected output (provided by experts) and the learned brain parcels. The score ranges from 0 to 1 where 1 indicates a perfect agreement. In general, D_c was low for small brain parcels when compared to larger brain structures, indicating less accurate parcellation of finer functional areas. The low parcellation performance and the high inter-subject variability were attributed to the limited training set used for the study (Lee, Laine, & Klein, 2011). Regardless, the architecture, learning process, and consequent results indicate that deep learning is a viable tool for successful two-dimensional brain parcellation.

The method presented in Chapter 2 proposes a promising approach for automated voxel-wise brain parcellation using stacked sparse autoencoders for unsupervised learning of *three-dimensional* fMRI data. Semi-supervised softmax regression will be used for voxel classification based on ground-truth labels of brain regions. This technique is implemented under the hypothesis that the patterns of connectivity underlying the fMRI data are of high dimension with features organized complexly in multiple levels of abstraction which can be extracted most effectively by a nonlinear deep autoencoder. The implementation details and parcellation results will be presented in Chapter 3, while Chapter 4 will serve as a final discussion and conclusion of the thesis's findings, weaknesses and significance.

Chapter 2

2 Methodology

The design and methodology of developing the presented solution to automated brain parcellation via deep learning will be outlined in this chapter. Beginning with raw resting-state fMRI data, a sparse deep autoencoder will be implemented in the Python programming language, learning to classify neural voxels based only on their functional connectivity with one another, without regard for their anatomic relations. A general summary of the algorithm for brain parcellation is outlined in the following 6 steps which will be further investigated in this chapter: 2.1) Prepare the data, 2.2) Pre-train the autoencoder, 2.3) Train the softmax classifier, 2.4) Fine-tune the entire network, 2.5) Parcellate new data, and 2.6) Visualize the parcellation.

2.1 Prepare the Data

The fMRI data of 10 different subjects were used for the training of the autoencoder. The size of this dataset allows for feasible runtime, but further investigation of this approach would benefit from a larger sample size. For each subject, the raw preprocessed data is first loaded into a xifti file format – a format developed for storing neuroimaging data with metadata for use with Python and C programs. The subject's functional and anatomical brain images are also loaded into the program to provide reference for the xifti file. From that file, a correlation matrix is produced using Python's NumPy library, associating the time course of each brain voxel against every other voxel. Thus a single row represents the similarity between a single voxel and every other voxel. Consider the case with n voxels, $V_i, V_2, \dots, V_n$. An $n \times n$ matrix can be formed, composed of Pearson product-moment correlation coefficients, r, such that the coefficient of V_i, V_j is at position i, j in the matrix. The coefficient is defined as the covariance of two variables X, Y (i.e.: V_i, V_j), divided by the product of their standard deviations. That is,

$$r = \frac{cov(X,Y)}{\sigma_x \sigma_y} = \frac{E\left[(x - \mu_x)(y - \mu_y)\right]}{\sigma_x \sigma_y} \qquad \text{Equation 8}$$

where cov is the covariance, σ_x is the standard deviation of X, μ_x is the mean of X, and E is the expectation (the expected outcome given the probability of each potential outcome). The coefficients range from [-1,1] such that 0 indicates no correlation, 1 indicates perfect positive correlation, and -1 indicates negative correlation between the voxels. The matrix is symmetric since the correlation between voxels V_i, V_j equals the correlation between voxels V_j, V_i. Additionally, the diagonal of the matrix is always 1 since there is perfect correlation between a voxel's time-series and itself.

From a row-wise comparison of the matrix, similarities among rows indicate stronger global associations of connectivity. To measure these associations, this correlation matrix of n voxels will be converted to an $n \times n$ η^2 matrix, as seen in Figure 7. (Note that "n" is the alphabetical character referring to the number of voxels, while "η" is the Greek character "*eta*" described as follows) while The η^2 statistic (Equation 4) describes the strength of the similarity between the voxels, as it represents the ratio of variance between each voxel and all the other voxels. Like the correlation matrix, the η^2 matrix will be symmetric where a single row represents the strength of the connectivity between a single voxel and each other voxel.

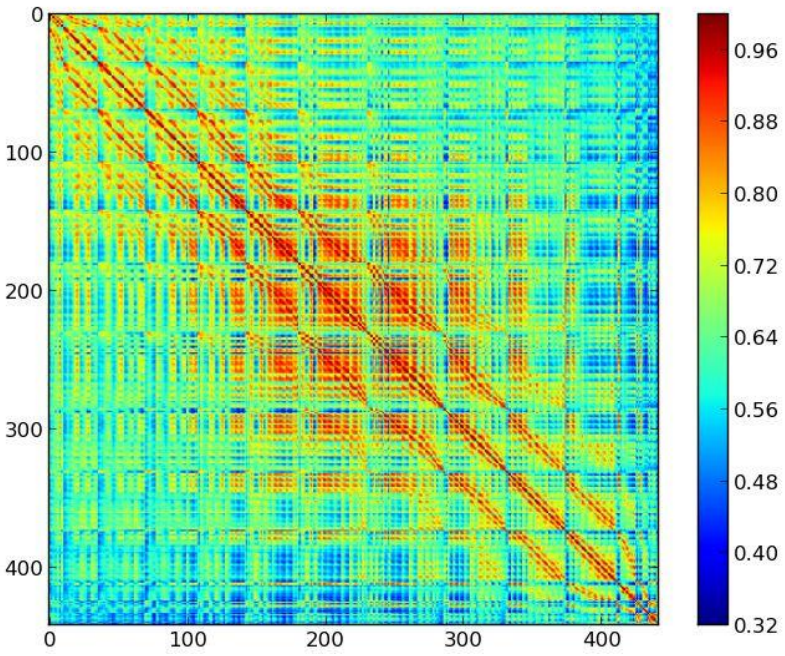

Figure 7: A 442×442 η^2 matrix depicting the global patterns of functional connectivity among the 442 voxels from the fMRI data. One row of the matrix represents one voxel's connectivity with each of the 442 voxels representing in the columns. The closer η^2 is to 1 (red/darker), the stronger the association.

Using such a matrix allows for the analysis of global correlation patterns among voxels. Rather than just investigating the interaction between each two voxels, the matrix allows for the investigation of how each voxel relates to the rest of the voxels as a whole. Therefore, it is the entire normalized η^2 matrix that serves as the initial low-level features of input fed into the stacked autoencoder. Each row of the matrix is a vector acting as a single input (visible) unit into the network from which the inherent latent factors will be extracted, activating the hidden layers of the network.

The whole brain can be investigated using the data or, if a particular area is of interest, a mask can be applied to the η^2 data, allowing the learning procedure to only act on that brain area. While less information is learned if an area is masked out, the problem becomes much smaller and easier to manage, resulting in a faster runtime. For this reason, this thesis will be focusing on the functional connectivity of the medial parietal cortex – an area of the brain associated with sensorimotor integration, regulating the

relationship between the sensory and motor systems for tasks such as hand-eye coordination. Therefore, only the data associated with this region is considered as the remainder of the data is masked out and ignored.

Once the fMRI data is in the suitable form of input data (a normalized η^2 matrix), the deep sparse autoencoder and softmax classifier will be trained and used for parcellation of the brain region's voxels. The final program is dynamic, allowing the user to input new fMRI data while specifying the desired number of hidden layers and hidden units per layer in the autoencoder. The remainder of this chapter outlines the procedure followed to achieve this goal.

2.2 Pre-train the autoencoder

Previous deep learning approaches used random initialization of parameters before training a network. This was a major downfall of previous deep neural networks as a vast amount of time was required for the networks to learn, making the technique infeasible for practical learning. However, Hinton's more recent method involved greedy layer-wise pre-training of each layer by initializing the parameters near a local minimum (Hinton & Salakhutdinov, Reducing the dimensionality of data with neural networks, 2006). By pre-training each layer, less time is required to learn the optimal parameters and the model becomes significantly more computationally powerful. Recall that with a stacked autoencoder, each subset of adjacent layers can be viewed as a single-layer autoencoder. Therefore the output activation vector from the first autoencoder in the stack is then used as input to the second autoencoder in the stack. Pre-training involves setting the parameters between adjacent single-layer autoencoders independent of the other autoencoders in the network. As one set of parameters is being determined (e.g.: the parameters between layers 1 and 2), the remaining parameters of the network (between layers 2 and 3, layers 3 and 4, etc.) remained fixed. Recall that the parameters of the network are (W, b) where W and b are matrices representing the weighted connections and biases, respectively, joining the units of adjacent layers of network.

The following process outlines the method for pre-training each set of parameters between layers of the network, while also computing the activation features extracted from each unit of that hidden layer. This activation vector output of each layer is in turn used as input for the pre-training of the following layer in the stacked autoencoder. This process demonstrates that the greedy layer-wise pre-training makes use of the same algorithm applied multiple times, once to each layer of the network, until the entire network has been pre-trained with optimized parameters.

Given a network of n_l layers, the pre-training will be performed on each layer independently. Thus, for each layer, l, in the network:

2.2.1) Initialize the parameters (W, b) to small near-zero values

2.2.2) Optimize the parameters by minimizing the cost function, $J_{sparse}(W, b)$

2.2.3) Calculate the output activation vector, a^l, and set it as the input for layer $l + 1$. If $l = n_l$ (i.e.: the last layer), a^{n_l}, is stored for input to the softmax model

The following subsections elaborate of these pre-training steps for a single layer of the autoencoder.

2.2.1 Initialize parameters

The goal of the pre-training is to minimize the cost, $J_{sparse}(W, b)$, as a function of W and b in order to set the parameters in a good neighbourhood for further training. In order to do so, these parameters at each layer must be initialized to some small, random, near-zero value. It is crucial that these parameters are not simply originating at values of 0 as such an initialization would result in all the hidden layer units learning the same function of the input (Ng, Ngiam, Foo, Mai, & Suen, 2013).

2.2.2 Optimize parameters

2.2.2.1 Optimization Method

The cost function is minimized using the Limited memory BFGS (L-BFGS) optimization algorithm provided in SciPy's optimization package. BFGS is a quasi-Newton optimization method. Newton optimization methods require the gradient of partial

derivatives of the function, as well as the inverse Hessian matrix (matrix of second order partial derivatives) – a large, costly, and complex factor to store and compute, making it infeasible for large optimization tasks. However, quasi-Newton techniques overcome this obstacle by instead using an *approximation* of the inverse Hessian matrix, which the method can extract from the first partial derivatives. Regardless, even the quasi-Newton BFGS method's approximation of dense inverse Hessian requires significant memory resources. Therefore, the Limited memory BFGS method offers a solution to this limitation by only using minimal vectors to represent the matrix implicitly, rather than fully calculating and storing the full inverse Hessian in memory (Galen & Jianfeng, 2007).

Stochastic gradient descent (SGD) is a simple alternative optimization technique commonly used in training. However, it not only requires addition fine-tuning of parameters (a learning rate), but it has been shown to be outperformed by L-BFGS when training large datasets. In particular, when comparing the two approaches for sparse autoencoder optimization, L-BFGS has demonstrated faster and more stable training (Le, Ngiam, Coates, Lahiri, Prochnow, & Ng, 2011). L-BFGS is a sophisticated, yet easy-to-use off-the-shelf optimization method that offers fast, reliable results, while requiring only the function to optimize and its gradient. For these reasons, L-BFGS is used throughout the project for optimization.

2.2.2.2 Cost Function

Starting with randomly assigned, near-zero values for the parameters, pre-training is done by training each layer individually. Thus as the first layer is pre-trained to find some optimal parameters, the remaining layers remain fixed. The input data of the entire model is used as the input data to the first layer. This data is used to determine the parameters which minimize the cost function of the sparse autoencoder, $J_{sparse}(W, b)$, and the corresponding derivatives of $J_{sparse}(W, b)$ with respect to the parameters (W, b).

The $J_{sparse}(W, b)$ cost function is the sum of three components: a) the average sum-of-squares error term, b) the weight decay term, and c) the sparsity penalty. The following subsections build the complete equation needed for the stacked sparse autoencoder.

a) Average Sum-of-Squares Error Term

The first term of $J_{sparse}(W, b)$ is the average sum-of-squares error term. This term measures the discrepancy between the observed data and the expected/estimated data. Given a set of m training examples, $\{(x^1, y^1), \ldots, (x^m, y^m)\}$, where x^i is the i-th input and y^i is its corresponding label, a neural network can be trained using L-BFGS optimization. The cost function of a single training example (x, y) can initially be defined as

$$J(W, b; x, y) = \frac{1}{2}\left\|h_{W,b}(x^i) - y^i\right\|^2 \qquad \text{Equation 9}$$

where $h_{W,b}$ is the hypothesized output of the particular unit, indicating the likelihood that the neuron will be firing. Specifically, $h_{w,b}(x) = f(\sum_{i=1}^{n} W_i x_i + b) = a$ (Equation 1), as calculated in the encoding step of the autoencoder. Recall that $f(\cdot)$ is the sigmoid function $f(z) = \frac{1}{1+e^{-z}}$ used to scale the output within the range [0,1]. Meanwhile, $\left\|h_{W,b}(x^i) - y^i\right\|$ is the Euclidean distance between $h_{W,b}$ and y. Thus to define the mean sum-of-squares error over all m training examples, the cost function starts out as

$$J_{sparse}(W, b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left\|h_{W,b}(x^i) - y^i\right\|^2\right]. \qquad \text{Equation 10}$$

b) Weight Decay Term

The second term of the cost function is the weight decay term. Also called a regularization term, a weight decay term is used to add a penalty to the error function. This term is useful for decreasing the magnitudes of the weight, thus minimizing the risk of overfitting. Without this penalty, large weights can cause excessive variance to the output (Geman, Bienenstock, & Doursat, 1992). Therefore the weight decay term is used to regularize the weights by decreasing their magnitude.

The weight decay parameter, λ, is used to control the relative importance of the weight decay term of the cost function. Too small of a λ will tend to overfit the data, while too

large values of λ likely underfits the data, both leading the poor predictions. The value is optimized using a grid search. As a result of adding the weight decay term, the cost function is now

$$J_{sparse}(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{1}{2}\left\|h_{W,b}(x^i) - y^i\right\|^2\right] + \frac{\lambda}{2}\sum_{l=1}^{n_l-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}\left(W_{ji}^{l}\right)^2 \qquad \text{Equation 11}$$

where n_l is total number of layers in the network and s_l is the number of hidden units in the layer.

c) Sparsity Penalty Term

While a_j^2 represents the output activation of the *j*-th hidden unit of the first autoencoder in the stack, let $a_j^2(x)$ explicitly represent the output activation of that same hidden unit given some particular input x.

The sparse functional connectivity of the brain has been justified through observation as neurological findings have demonstrated that neurological processes usually only directly interact with few other brain regions (Huang, et al., 2009). Therefore, under the reasonable assumption that the underlying data from the fMRI is sparse, a sparsity constraint is placed on the network, limiting the activation of the hidden units with the goal of discovering the underlying structure of the data, regardless of the number of hidden units used. To establish the sparsity constraint on the autoencoder, $\hat{\rho}$ represents the average activation of a particular hidden unit, resulting from all the inputs to that unit. That is,

$$\hat{\rho}_j = \frac{1}{m}\sum_{i=1}^{m}\left[a_j^2(x^i)\right] \qquad \text{Equation 12}$$

where $\hat{\rho}_j$represents the average activation of the hidden unit *j* averaged over the set of *m* training examples. Further, the constraint is set such that $\hat{\rho}_j = \rho$ where ρ is the sparsity parameter of the entire network, the value of which is typically close to 0 so that the distribution of activations is highly peaked at zero in order to maintain overall sparsely

active neurons in the network. For example, if ρ = 0.05, most of the hidden units must have activations close to 0 in order to maintain an *average* activation of each hidden unit in the network near 0.05, the desired overall sparsity.

To achieve this requirement of equality between $\hat{\rho}_j$and ρ, an additional penalty term based on Kullback-Leibler divergence is added to the cost function, used to penalize $\hat{\rho}_j$ when deviating significantly from ρ (Hinton, A Practical Guide to Training Restricted Boltzmann Machines, 2010). KL-divergence measures the difference between two probability distributions. Specifically,

$$\sum_{j=1}^{s_l} KL(\hat{\rho} \,||\, \rho) = \sum_{j=1}^{s_l} \left(\rho \log \frac{\rho}{\hat{\rho}_j} + (1-\rho) \log \frac{1-\rho}{1-\hat{\rho}} \right) \qquad \text{Equation 13}$$

is a measure of the information lost when ρ is used to approximate $\hat{\rho}$, while summing over all the s_l hidden units in the layer.

Thus, the term enforcing sparse activation is added to the cost function, penalizing the units of the autoencoder that are active:

$$J_{sparse}(W,b) = \left[\frac{1}{m} \sum_{i=1}^{m} \frac{1}{2} \left\| h_{W,b}(x^i) - y^i \right\|^2 \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} \left(W_{ji}^{l} \right)^2 + \beta \sum_{j=1}^{s_l} KL(\hat{\rho} \,||\, \rho) \qquad \text{Equation 14}$$

where β is the sparsity parameter, used to control the relative weight of the sparsity penalty term.

2.2.2.3 Gradient

Backpropagation is used to determine the gradient vector of the partial derivatives of the function. Common backpropagation procedures require some expected results to compare with the actual results, and are therefore reliant on supervision. However, recall that the fundamental concept behind autoencoders dictates that the output of the network is a reconstruction of the input, where the hypothesized output is approximated by the input (i.e.: $h(x) \approx x$).

The gradient of the overall cost function, $J_{sparse}(W,b)$, is determined from the following partial deriviatives (noting the weight decay term is only applicable with respect to W, not b):

$$\frac{\delta}{\delta W_{ij}^{l}} J_{sparse}(W,b) = \left[\frac{1}{m}\sum_{i=1}^{m}\frac{\delta}{\delta W_{ij}^{l}} J(W,b;x^{i},y^{i})\right] + \lambda W_{ij}^{l} \qquad \text{Equation 15}$$

$$\frac{\delta}{\delta b_{i}^{l}} J_{sparse}(W,b) = \frac{1}{m}\sum_{i=1}^{m}\frac{\delta}{\delta b_{i}^{l}} J\left(W,b;x^{i},y^{i}\right) \qquad \text{Equation 16}$$

where the partial derivatives of the cost function, $J_{sparse}\left(W,b;x^{i},y^{i}\right)$, with respect to a single training example, (x,y), are given as

$$\frac{\delta}{\delta W_{ij}^{l}} J_{sparse}\left(W,b;x^{i},y^{i}\right) = a_{j}^{l}\delta^{l+1} \qquad \text{Equation 17}$$

$$\frac{\delta}{\delta b_{i}^{l}} J_{sparse}\left(W,b;x^{i},y^{i}\right) = \delta_{i}^{l+1}. \qquad \text{Equation 18}$$

These partial derivatives of individual examples are determined via backpropagation. The output activation value of each node in the layer l, a_{i}^{l} is calculated using the encoding equation with the current parameters and then compared with the expected output (i.e.: a reconstruction of the input value since it is an autoencoder) and an error term, δ_{i}^{l}, is calculated to represent the amount that the node i contributed to the discrepancies between the actual and expected output.

The error term is first calculated at the last layer, then propagated backwards through the preceding layers. For the last layer in the network, when $l = n_{l}$, the error term is defined as

$$\delta_{i}^{n_{l}} = \frac{\delta}{\delta z_{i}^{n_{l}}}\frac{1}{2}\left\|y - h_{W,b}(x)\right\|^{2} = -\left(y_{i} - a_{i}^{n_{l}}\right)\cdot f'(z_{i}^{n_{l}}) \qquad \text{Equation 19}$$

Note that if $f(z) = \frac{1}{1+e^{-z}}$ is the sigmoid function, the derivative is given by

$f'(z) = f(z)(1 - f(z))$. Since the activation of a network is defined as $a_i^l = f(z_i^l)$, then $f'(z_i^l) = a_i^l(1 - a_i^l)$.

For each remaining layer, l, in the network (from $l = n_l - 1, n_l - 2, \ldots, 2$), the error term is based on the error of layer $l + 1$ (the succeeding layer) and takes sparsity into account, defined as

$$\delta_i^l = \left(\left(\sum_{j=1}^{s_{l+1}} W_{ji}^l \, \delta^{l+1} \right) + \beta \left(-\frac{\rho}{\hat{\rho}} + \frac{1-\rho}{1-\hat{\rho}} \right) \right) \cdot f'(z^l) \qquad \text{Equation 20}$$

Now that the cost function and partial derivatives which make up the gradient vector have been determined (Equations 15 and 16), the L-BFGS algorithm is run and the parameters are updated.

2.2.3 Compute the activation vector

Once the cost function for the current layer has been minimized and the optimal parameters are returned, those parameters of the current layer are then fed into the feed-forward encoding steps (Equations 3 and 4), thus extracting the activation feature output of that layer, scaled to a value within the range of [0,1] by the sigmoid function.

In order to stack each single layer autoencoder to form a deep autoencoder, the output of the preceding layer is in turn used as the input for the following layer of the autoencoder. So while the raw data is used as input into the first layer of the network, the input for each following layer is now set to be the value of the activation vector of the previous layer. The above pre-training process (parameter initialization, optimization, and encoding of activation vectors) continues until the entire network of n_l layers has been pre-trained. Finally, the activation of the last layer in the network, a^{n_l}, will be calculated, fundamental in achieving the ultimate goal of this project – to parcellate voxels into functional brain regions.

2.3 Train the softmax classifier

After having trained each layer of the network on the unlabelled data, the parameters are now starting at a better location in parameter space than if they had been randomly initialized – thus accounting for a fundamental flaw in previous deep networks. In addition, the final output of the network, a^{n_l}, has been calculated – a feature vector which provides a reconstruction of the input with respect to its high-level features. These features can be fed into a classifier in order to perform classification of the sparse stacked autoencoder's input value. Continuing from the stacked autoencoder being constructed in Figures 5 and 6, Figure 8 illustrates the concept of the last hidden layer's activation vector serving as raw input to the softmax classifier. Note that the softmax classifier is not considered an additional layer in the network.

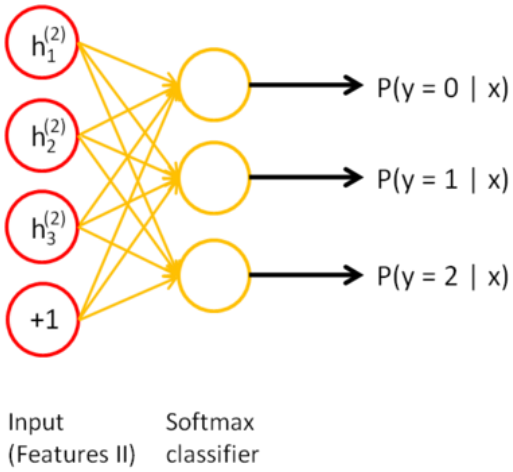

Figure 8: Continuing to build the stacked autoencoder from Figures 5 and 6, the activation vector resulting from the last hidden layer in the network is used as input to the softmax classifier which determines the probability of each possible label (Ng, Ngiam, Foo, Mai, & Suen, 2013).

In general, a classifier can be defined as a function that receives values of various features from training examples (independent variables) and provides an output which predicts the class or category that each training example belongs to (dependent variables) (Pereira, Mitchell, & Botvi, 2009). In the case of this project, each row of the η^2 matrix

represents a comparison of a voxel's global pattern of functional connectivity with those of all other voxels, which serves as input to the network. The goal of the classifier being trained is to segment the voxels based on their patterns of functional connectivity.

Logistic regression is a common supervised classification model used to classify data into one of two possible categories. However, brain segmentation is much more complex and there exist many potential functional brain regions in which a voxel may reside. Therefore, a softmax regression model (also known as "multinomial logistic regression" – which generalizes logistic regression) will be used for this multi-class classification, allowing each voxel to be classified into one of many possible function brain regions. Since these regression models require labelled data, the softmax classifier will be training using a limited number of labelled training examples in the form of ground truth labels provided by two neuroanatomist at the Rotman Research Institute at Baycrest Hospital in Toronto. As a result, the classification step is neither supervised nor unsupervised, but rather *semi-supervised*. The labels used are well-defined within their functional regions, thus not compromising the accuracy of the classification.

Given a limited set of m labelled training examples, $\{(x^1, y^1), \dots, (x^m, y^m)\}$, where x^i is the i-th input and y^i is its corresponding label/classification, a softmax regression model dictates that y^i can take on any one of k possible values (where k is the total number of potential classifications), such that $y \epsilon \{1, 2, \dots, k\}$. By contrast, binary classification with logistic regression limits the label y such that $y \epsilon \{0,1\}$. The softmax model performs the classification by predicting the probabilities of the k possible outcomes given the inputs.

For convenience, consider an alternate notation for representing the parameters of the network. For each layer, l, let θ_l represent a single long, one-dimensional vector consisting of the flattened parameter matrices of (W, b) concatenated together. Furthermore, let θ represent a matrix stacking $\theta_1, \theta_2, \dots, \theta_{n_l}$ thus storing all parameters of the network in a single variable.

The cost function associated with binary logistic regression is defined by the following equation:

$$J_{logistic}(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=0}^{1} 1\left\{y^i = j\right\}\log p\left(y^i = j \,\middle|\, x^i;\ \theta\right)\right] \qquad \text{Equation 21}$$

where $1\{\cdot\}$ is an indicator function. The function evaluates to 1 if $\{\cdot\}$ is true, otherwise it evaluates to 0 if $\{\cdot\}$ is false. That is, 1{true} = 1 (e.g.: 1{1+1 = 2} = 1) and 1{false} = 0 (e.g.: 1{1+1=3} = 0). For the purpose of this project, the indicator function will represent the correspondence with standard ground truth labels. Thus if the output corresponds to the ground truth label, the term indicator function evaluates to 1.

Note that when dealing with multi-class softmax regression, the function must sum over all possible k values (i.e.: from $j = 1, 2, \ldots, k$ rather than $j = 0, 1$). In addition, softmax regression holds that given some input x, the probability of the unit i's label taking on the each of the potential k values ($j = 1, 2, \ldots, k$) is represented as

$$p\left(y^i = j \mid x^i;\ \theta\right) = \frac{e^{\theta_j^T x^i}}{\sum_{l=1}^{k} e_l^T x^i} \qquad \text{Equation 22}$$

In continuing to develop the softmax regression cost function, the problem of over-parameterization must be addressed: the function allows multiple parameter settings to exist that result in the same output. Therefore, a weight decay term will be added to the cost function, penalizing excessively large values of the parameters, thus preventing overfitting.

As a result of these changes to the logistic regression cost function, the softmax regression cost function is formally defined as

$$J_{softmax}(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{j=1}^{k} 1\{y^i = j\}\log\frac{e^{\theta_j^T x^i}}{\sum_{l=1}^{k} e_l^T x^i}\right] + \frac{\lambda}{2}\left[\sum_{i=1}^{k}\sum_{j=0}^{n}\theta_{ij}^2\right]. \qquad \text{Equation 23}$$

To implement softmax regression, this function will be optimized via L-BFGS. The gradient is

$$\nabla_{\theta_j} J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[x^i\big(1\{y^i = j\} - p\big(y^i = j|x^i;\ \theta\big)\big)] + \ \lambda\theta_j. \qquad \text{Equation 24}$$

Now given the cost function and gradient, the final activation vector output from the last layer of the stacked autoencoder determined during pre-training, a^{n_l}, is used as input to the softmax classifier (Figure 9). The cost function is minimized using the L-BFGS optimization algorithm to train the classifier and further improve the parameters.

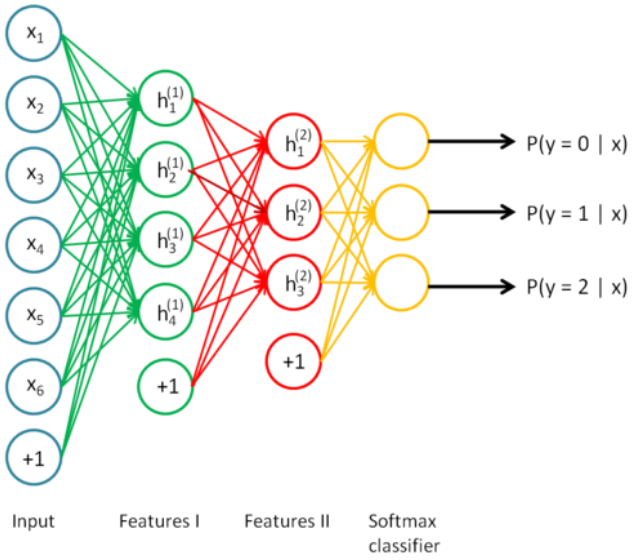

Figure 9: The entire stacked autoencoder with two hidden layers, viewed as a single model capable of classification. The output activations of the second (and final) hidden layer are input into the softmax classifier (Ng, Ngiam, Foo, Mai, & Suen, 2013).

2.4 Fine-tune the stacked autoencoder

A fine-tuning pass is commonly used in training to improve the performance of the stacked autoencoder. While layer-wise pre-training is used for finding the features of the network, fine-tuning is used to slightly modify the features of the network in order to adjust the boundaries between the classification classes (Hinton, 2007). To implement fine-tuning of the entire network, all the layers must be treated as a single model. A single iteration of fine-tuning improves all the weights of the stacked autoencoder, at every level. A similar technique can be used for this fine-tuning step as was used to pre-

train the parameters by minimizing the $J_{sparse}(W, b)$ function by implementing a similar backpropagation step to get the gradients. Using the latest parameters discovered during classification, fine-tuning involves performing L-BFGS optimization with the goal of reducing the error between the actual output of the softmax model and the expected output (i.e.: the reconstruction of the input). The same cost function and gradient defined in Equations 23 and 24 are used, respectively, and the parameters are updated for the final time.

2.5 Parcellate new fMRI data

Now that both the network and the classifier are fully trained with putatively optimal parameters, entirely new data can be fed into the stacked autoencoder for classification provided by new subjects. While fMRI data from 10 subjects was used to *train* the autoencoder, the fMRI data from 2 subjects were retained for *testing*. For each test, 1 of the 2 test subjects was randomly selected for the new parcellation. Note that the training data was not used for testing. The new subject's fMRI data is processed into a normalized η^2 matrix to represent the global patterns of functional connectivity among voxels using the approach presented in Section 2.1. Starting from this low level data, higher level features will be extracted at each new layer of the autoencoder. Finally, the trained softmax classifier outputs a vector of digital labels where each row of the η^2 matrix (each voxel) yields a corresponding label. This vector of labels can then be re-associated with the anatomical structure of the brain to depict the parcellation.

2.6 Visualize the parcellation

NIfTI (Neuroimaging Informatics Technology Initiative) is a data format in which the classification of each voxel can be assigned to its anatomical counterpart in brain space. The 3D NIfTI image resulting from the predicted labels will colour-code voxels such that voxels belonging to the same functional brain region will the same colour. An anatomical 3D image of the brain may underlay the labels to illustrate the parcellation directly on the brain's structure. Alternatively, the NIfTI image of the predicted labels may be compared with a NIfTI image created from the ground truth labels that are commonly accepted

among experts. This comparison allows for the general visual analysis of the success of the functional brain parcellation via unsupervised deep learning.

Chapter 3 presents the findings of implementing this stacked autoencoder for automated brain parcellation. The results will be analyzed and compared with expected functional segmentations, measuring the accuracy and success of this solution.

Chapter 3

3 Implementation and Results

3.1 Preprocessing

The deep sparse autoencoder was trained on fMRI data from 10 different subjects. The dataset contains resting-state fMRI scans (and associated anatomical scans) for 10 right-handed adults (5 male) ranging in age from 21 to 35 years. None of these participants had a history of neurological or psychiatric illness. MRI data were acquired with a 3 T Siemens TimTrio MRI scanner using a 32-channel head coil. Functional volumes consisted of 36 slices acquired parallel to the ACPC axis using an interleaved slice acquisition order and an echo-planar imaging pulse sequence (TR = 2000 ms, TE = 30 ms, flip angle = 78° , 64 × 64 matrix, 21.1 × 21.1 cm FOV, 3 × 3 × 3 mm voxel resolution). A total of 300 functional volumes were collected from each participant. In addition, a high-resolution anatomical scan (192 slices, 256 × 256 matrix, 21.1 × 21.1 cm FOV, 1 × 1 × 1 mm voxel resolution MP-RAGE pulse sequence) was acquired from each participant to assist in visualizing the results of functional analyses and aid in preprocessing.

The use of 10 subject samples allowed for a variable unlabelled training set, while the sample size was small enough to be computationally feasible in training and testing hundreds of autoencoders with various parameters. However, further investigates of this technique would likely benefit from a larger dataset if the resources are available. The study focuses specifically on the functional parcellation of the medial parietal cortex, therefore a mask is applied to the fMRI information to isolate that region of interest. Therefore the input data is composed of the intersection of the total fMRI data with the mask, ignoring the rest of the brain. By doing so, less of the brain is segmented, but the algorithm runs faster on the drastically reduced number of voxels – a requirement given the limited resources and the need for a combinatorial parameter search. The medial parietal cortex, composed of 442 voxels, will be segmented by the deep autoencoder.

Thus a single η^2 matrix for one subject is a 442×442 matrix where each row represents the global patterns of functional connectivity between that voxel and all other voxels of the medial parietal cortex. A total of 10 η^2 matrices are used as unlabelled training sets for the autoencoder to optimize its parameters during training. Once training is completed, the autoencoder can be used to parcellate new fMRI data, the resulting output of which will be a 1×442 array of labels corresponding with the 442 voxels of the subject's medial parietal cortex in the common functional reference space.

3.2 Implementation Details

The training of the deep autoencoder is performed multiple times, using a range of hidden layers and a range of hidden units per layer for each run. Consider an autoencoder with n hidden layers. Each of the n hidden layers are trained and tested with a range of hidden units from 200 to 1000, in increments of 200, such that each combination is used. That is, since there are 5 possible numbers of hidden units per layer (200, 400, 600, 800, 1000), there is a total of 5^n autoencoders trained and tested for an autoencoder with n hidden layers. As outlined in chapter 1, the choice of number of hidden units per layer is not well established as it ranges from suggestions of two-thirds to twice the number of input units. However, these rules of thumb are not heavily supported either empirically or theoretically, so many tests are performed with varying numbers of hidden units.

Deep autoencoders with 2, 3, and 4 hidden layers will be investigated using this process. Networks with 5 and 6 hidden layers are of interest, but infeasible to test using the same strategy as the addition of more layers (of hundreds of units) drastically increases the training time ($5^5 = 3125$ and $5^6 = 15625$ runs of training and testing would be required). Therefore, networks with 5 and 6 hidden layers are tested with 200, 600, and 1000 units per hidden layer (a total of $3^5 = 243$ and $3^6 = 729$ runs, respectively).

While the parameters of the neural network are represented as (W, b), the training of the network also depends on the *hyperparameters* λ, ρ, and β which represent the weight decay parameter, sparsity parameter, and weight of the sparsity penalty of network's cost function, respectively. The values of the hyperparameters were determined independently for networks with each number of hidden layers. A grid search was used, keeping all

hyperparameters constant expect the one being optimized. Table 1 shows the final optimal values discovered for these hyperparameteres.

Table 1: The hyperparameters used for training the autoencoders of various depths.

Number of Hidden Layers	λ	ρ	β
2	0.00003	0.040	0.3
3	0.00005	0.035	0.5
4	0.00001	0.045	0.3
5	0.00001	0.045	0.1
6	0.00003	0.040	0.5

In addition to the deep autoencoders trained and tested, an autoencoder with 1 hidden layer was implemented to provide a comparison of the deep model with a model similar to PCA. As introduced in Chapter 1, an autoencoder with a single linear hidden layer behaves similarly to PCA. The thesis investigates deep autoencoders under the hypothesis that the functional connectivity of the voxels captured in the fMRI data is nonlinear, therefore the results from this PCA model will provide insight as to the nature of the input used and whether a deep learning approach is worth the significant computational cost (training time and computational resources) when compared to a linear approach.

3.3 Results

As outlined in Chapter 1, there does not exist a standard method of parcellating functional brain regions into discrete regions nor is there a gold standard parcellation to use for comparisons, thus limiting the quantitative analysis of the resulting parcellation. However, for the purpose of investigating the success of the deep autoencoder, the learned labels which represent the grouping of voxels are compared with a set ground truth, commonly accepted labels, which were verified by two neuroanatomists, to check for validity.

The Dice coefficient is a statistic to measure the spatial overlap (similarity) of two samples, commonly used for segmentation evaluation in medical imaging (Lee, Laine, & Klein, 2011). Using this metric to evaluate the algorithm will allow for simple comparisons with other studies and alternative approaches. The Dice coefficient, D_c, is determined by the following equation:

$$D_c = \frac{2\,|\,A\ \cup B|}{|A| + |B|} \qquad \text{Equation 25}$$

where A and B are the two sets being compared, $|\cdot|$ indicates the size of the set, and the operator ∪ represents the union of its operands. In the case of the parcellation technique being examined, the two sets are the ground truth labels and the parcellation labels acquired from the algorithm. The value of D_c ranges between 0 (no overlap) and 1 (perfect agreement).

The Dice coefficient is calculated for each parcellation resulting from each trained autoencoder with every combination of hidden units per layer. Appendices A - D show tables of the resulting D_c values for each autoencoder trained and tested with 2 – 5 hidden layers. The most successful parcellation with D_c = 0.678 was achieved with 5 hidden layers of 600, 600, 1000, 1000, 200 units per respective layer.

Investigating the trend of parcellation accuracy with respect to the number of hidden layers of the autoencoder, Figure 10 shows the D_c value for all the runs, grouped by number of hidden layers in the autoencoder, organized in a violin plot with a boxplot overlaid. A violin plot is similar to a box plot, but it roughly shows the probability density of the data at various values. That is, the figure shows the distribution of D_c values for all tests for each number of hidden layers. The wider the plot, the greater proportion of runs with that particular D_c were recorded.

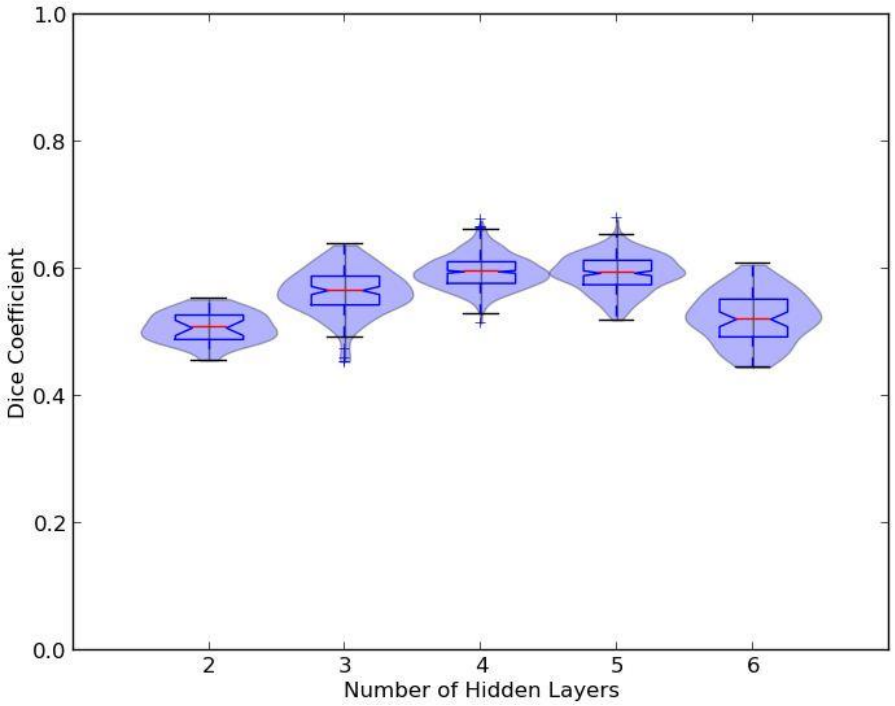

Figure 10: Violin and box plots depicting the distribution of Dice coefficients from parcellations trained and tested on autoencoders with 2 – 6 hidden layers.

From the graph, it is evident that the parcellations were most accurate when 4 and 5 hidden layers were used in the autoencoder. At the 6th layer, the accuracy declines. As shown in the graph, the distribution of D_c for autoencoders of the various depths is approximately normal. The minimum, maximum, and mean values are summarized in Table 2. The standard error of the mean of D_c for each number of hidden layers, $\Delta D_{c_{mean}}$, is calculated as $\Delta D_{c_{mean}} = \frac{\sigma}{\sqrt{N}}$ where σ is the standard deviation and N is the number of runs performed for that particular number of hidden layers.

Table 2: The minimum, maximum, mean, and standard deviation of the Dice coefficients for parcellations implemented with autoencoders of various depths.

Number of Hidden Layers	$D_{c_{min}}$	$D_{c_{max}}$	$D_{c_{mean}} \pm \Delta D_{c_{mean}}$	σ
2	0.455	0.552	0.508 ± 0.005	0.023
3	0.452	0.638	0.563 ± 0.003	0.037
4	0.514	0.676	0.595 ± 0.001	0.026
5	0.518	0.679	0.592 ± 0.002	0.029
6	0.443	0.607	0.523 ± 0.005	0.039

Running ANOVA on the results yields a p-value of 3.3×10^{-10} between all groups of hidden layers. Such a small p-value implies statistical significance between the accuracy of the segmentations resulting from the various depths of the autoencoder.

The linear shallow autoencoder implemented to model PCA had a single hidden layer of 442 hidden units. Upon optimizing the parameters, the most accurate single layer parcellation is described by a Dice coefficient of 0.467.

Focusing on the deep autoencoder, Figure 11 shows a sample of parcellation results of the medial parietal cortex projected back into brain space. All voxels of the same colour represent a segmentation of voxels belonging to the same functional region. Recall that the parcellations are performed based on no anatomical dependencies – only the functional connectivity among the voxels is considered as input for the segmentation. Each row of three images in the figure represents one segmentation, viewed in three dimensions. The first segmentation (the first row) depicts the ground truth delineation based on the labels verified by neurologists, while the following rows show the most accurate segmentation resulting from 3, 4, and 5 hidden layers, respectively.

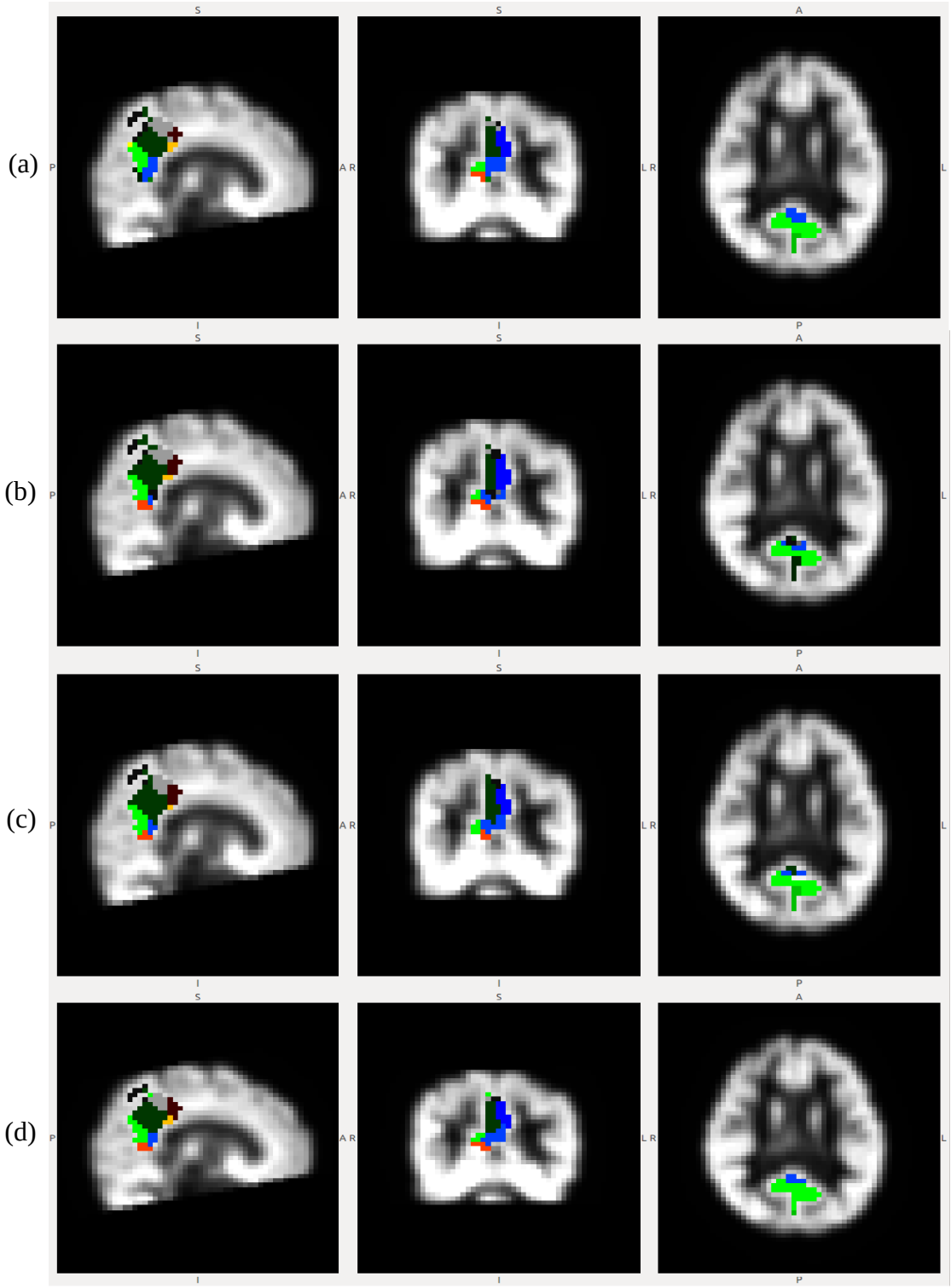

Figure 11: Each row depicts 3 dimensions of a single parcellation of the medial parietal cortex: (a) displays the ground truth parcellation verified by anatomists; autoencoders trained and tested with (b) 3 hidden layers, D_c = 0.638 ± 0.037, (c) 4 hidden layers D_c = 0.676 ± 0.026, (d) 5 hidden layers, D_c = 0.679 ± 0.029.

While Figure 11 (above) shows the parcellations in the context of the whole brain, Figure 12 (below) shows close-up views of for the ground truth parcellation and the most accurate parcellation resulting from 5 layers, allowing for easier visual comparison. In other words, Figure 12 depicts zoomed in views of Figure 11 (a) and (d).

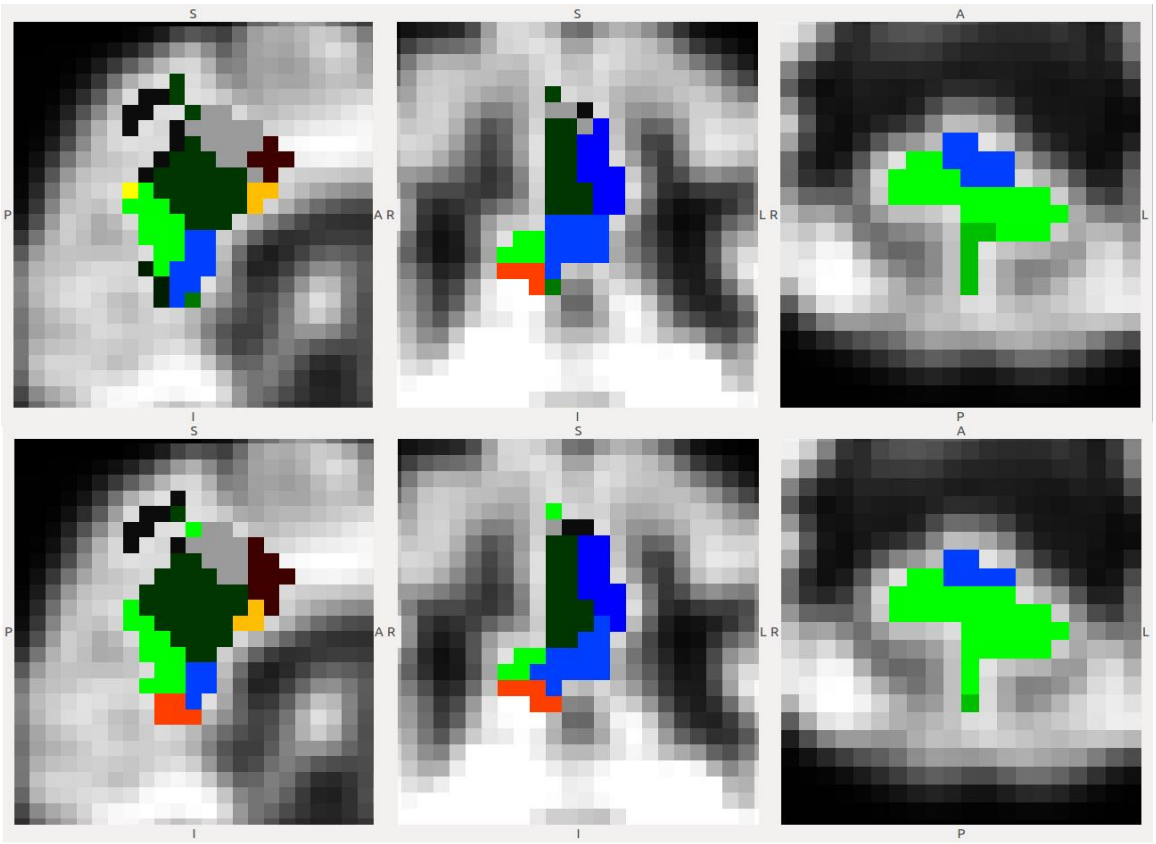

Figure 12: Zoomed-in parcellations of the medial parietal cortex where (a) shows the ground truth parcellation based on expert verified labels of functional regions in 3 dimensions and (b) shows a parcellation acquired from a deep autoencoder with 5 hidden layers. The spatial overlap between the two parcellations is described by D_c = 0.679 ± 0.029.

4 Discussion and Conclusion

This thesis represents a first step in developing and exploring deep sparse autoencoders for voxel-wise automated parcellation of functional brain regions. Functional brain parcellation is the task of delineating regions of the brain, based on the functional connectivity of the components within the brain (voxels). To perform this task, an artificial neural network called a deep autoencoder is constructed and trained with unsupervised learning. It works by minimizing the reconstruction error between the input data and the reconstruction of that input data at the output, in an unsupervised manner.

The process of training and testing the autoencoder for parcellation begins by converting a subject's raw fMRI data to a normalized η^2 matrix, where each row represents a voxel's global patterns of functional connectivity with other voxels. Therefore, each row serves as an input feature vector to the autoencoder from which higher level features will be extracted, enabling the classification of each voxel to a functional region. The deep autoencoder is composed of several single-layered autoencoders stacked together, where the features discovered at each layer are able to progressively represent higher-level features of the input data as the output of one becomes the input to the next. By pre-training each set of adjacent layers of the network independently, a good solution is found near the local minima prior to the rest of the training, making deep learning a significantly more feasible pursuit than otherwise thought. Sparsity is also introduced, limiting the overall activation of the network, thus modelling the expected sparse connectivity of the brain. The autoencoder uses a semi-supervised softmax regression model to segment the voxels based on potential ground truth functional regions, before a final backpropagating fine-tuning pass is performed. As a result, the putatively optimal parameters of the network are discovered and new fMRI data (converted to an η^2 matrix) can be fed into the autoencoder, resulting in a parcellation of the voxels based solely on their functional connectivity.

4.1 Discussion

The most accurate parcellations were performed by autoencoders with 4-5 hidden layers, thus supporting the hypothesis that the patterns of functional connectivity underlying the fMRI are not linear, but rather of high dimension and complexly organized at multiple levels of abstraction. The deep autoencoder, in comparison to the linear PCA approach, was able to extract those high level features in order to train the classifier and build resulting parcellations. Furthermore, once the autoencoder consisted of over 5 hidden layers, it is likely that the model suffered from overfitting, leading to less accurate parcellations.

The study performed by Lee et al. (2011) which implemented deep learning in the form of a convolutional network to two-dimensional fMRI images performed mean segmentations with Dice coefficients of 0.85 ± 0.04 and 0.73 ± 0.04 (for two slightly different approaches). The segmentations using the autoencoder implemented in this thesis with 5 hidden layers had a mean Dice coefficient of 0.592 ± 0.002 and a maximum of 0.68 ± 0.03 (i.e.: (68 ± 3)% accuracy compared with the ground truth) – modest, but respectable values demonstrating the potential of the approach. The fact that the deep learning parcellation technique implemented with autoencoders provides these promising results with no anatomical dependencies for training is an important step for the fields of both neuroscience and applied machine learning.

4.2 Contributions

Currently, there does not exist a single acknowledged standard method for automatic brain parcellation of functional regions. Existing protocols rely heavily on static labelled brain atlases (which assume significant anatomical similarity among brains), manual feature engineering (which are tedious and prone to human error), or use classification methods which do not scale well to fMRI's complex data sets (such as k-means clustering). This thesis presents a novel solution to automated functional parcellation of three-dimensional fMRI data via deep autoencoders. The ability to accurately parcel

functional regions automatically enhances the quality of further experimental investigations, as more accurate segmentations can be discovered and investigated.

In addition, a major contribution of the parcellation technique presented is that, unlike many alternative approaches, the project presented does not rely on the anatomical structure of the data to delineate the brain space. That is, the physical locations of the voxels do not play a role in the resulting delineation. Rather, only the degree of functional connectivity among voxels discovered in the raw fMRI data is used to train the autoencoder for the segmentation discovered. Therefore this research offers new insights to both the neuroscience and machine learning communities.

In a practical setting, an automated unsupervised method of parcellation permits neurologists to apply the parcellation technique directly to their patients. After an fMRI scan, the preprocessed data can be fed into the program developed. From there, the data are converted to a normalized eta-squared matrix which is used as input to the trained stacked autoencoder, resulting in a segmentation of the voxels in the brain. This technique requires neither human input nor machine learning expertise from the user of the program. As a result, the user will have access to a customized delineated brain image, based on specific functional connectivity of the subject, rather than a brain image segmented by simply associating standard structural regions with functions. The segmentation technique can be used in clinical neuroscience and cognitive psychology to better under the brain's connectivity and perhaps provide better diagnoses and treatment to individuals with neurodegenerative diseases such as Alzheimer's. Subtle abnormalities in functional connectivity can be indicative of Alzheimer’s disease years before clinical diagnoses (Sheline & Raichle, 2013). Therefore, the parcellations depicted from this deteriorating functional connectivity may serve as an instrument for early detection of the disease. The research has the potential to provide the medical community with a feasible, automated way of modelling the relationship between structure and function in the brain that can be consistently and repeatedly applied to different subjects of varying health. It is the intent that the current research effort will contribute to a more accurate and precise mapping of function within the brain. Through the advances of this data-driven machine learning technique, this methodology may be extend beyond the neuroimaging

community and benefit communities in any domain which can be modelled as a network, eliminating the need for guesswork and prior knowledge.

In addition, this thesis contributes to the machine learning research community a novel application of a deep autoencoder, stacked for the classification of functional brain regions. Applying this approach to a task as large in scale and complex as the human brain will undoubtedly prove informative for the growth of the practical deep learning. Furthermore, deep learning is a relatively young research area. Successful application of techniques from unsupervised deep learning to the important, real-world problem of brain parcellation will help to validate what has been, for much of its history, a largely supervised enterprise.

4.3 Threats to Validity

A criticism of the parcellations performed is that for each test, the deep autoencoder was consistently trained using the fMRI data of the medial parietal cortex from only 10 subjects. This sample size was chosen as a balance between fair representation and efficiency. Nevertheless, the sample size used is relatively small. While using more subjects' data would have been infeasible (time-wise) for the number of tests run (individual tests – out of thousands – ranged from approximately 2 minutes – 1.5 hours running on 8 processors, depending on the size of the autoencoder), additional data would provide more reputable training data. Therefore, the accuracy of the test runs likely suffered as a result of this limited training set.

In addition, deep networks are notoriously difficult to train due to the difficult optimization of the algorithm’s hyperparameters. Although the training was performed from empirically “good” hyperparameters, it is possible that the optimization algorithm got stuck in poor local optima, preventing the most effective parameters from being used. Furthermore, the hyperparameters could not be optimized for every combination of number of hidden units per layer, so even if the best hyperparameters were found via optimization, they are not necessarily ideal for every test.

A fundamental challenge of this project stems from the lack of gold standard with which the results of the parcellation can be compared for analysis. A segmentation derived via k-means which was approved by anatomists is used as the “ground truth” for the sake of validating the parcellation, but it is crucial to acknowledge that the brain is a dynamic organ, and somewhat unique for each individual. A major motivation of this study is to find an alternative to applying a single parcellation to multiple subjects, so the reliance on it for comparisons compromises the measure of the method’s ability to capture subject-specific functional variability. The ground truth labels are determined from healthy subjects, and therefore do not take abnormalities in connectivity into consideration. As a result, while the ground truth labels offer confidence to the measure of parcellation success, the labels used for validation do not provide ideal confirmation of the subject-specific accuracy. Nevertheless, this investigation has demonstrated that a deep learning approach to the problem is both feasible and meritorious of further study.

4.4 Future Work

To further develop this approach, it would be prudent to attempt the parcellation with more training data than from 10 subjects. While more subject data can be acquired experimentally, the NIH Human Connectome Project data offers a large database of highly reliable neuroimaging data as the Project aims to map anatomical and functional connectivity within the brain (The Human Connectome Project, 2014). The vast number of subject data provided from this database would result in a more credible conclusion as a large sample size can be used – hundreds of different brains can be used for training and validation of the parcellation technique. In addition, parcellation of the entire brain – not just the medial parietal cortex – would provide insight into the accuracy of the approach at a larger scale.

Additionally, since there is no gold standard functional segmentation of the brain, the ground truth labels used for validation are not necessarily accurate for each specific subject. Objective and accurate evaluation of any segmentation method is crucial for a technique to be accepted in practice. Therefore, a prudent step for improved validation would be to manually segment each subject's brain based on functional connectivity. Of

course, while a motivation of this investigation is to eliminate the need for this tedious, time-consuming task, a comparison to a *subject-specific* manual segmentation may provide additional insight into the true accuracy of the deep autoencoder.

Another future step in improving this parcellation technique would be to implement the deep autoencoder on a GPU. The algorithm implemented makes heavy use of matrix multiplication and element-wise operations. Such calculations associated with deep learning have been found to be 10 – 30 times faster when implemented on a GPU compared with common CPUs (Bergstra, Bastien, Turian, Pascanu, & Delalleau, 2010). By implementing the deep autoencoder in this way, the optimization of parameters (and hyperparameters) will be significantly more efficient, allowing for more accurate parcellations in less time.

Furthermore, a future step for improving this parcellation technique would be to implement a more efficient search of the hyperparameters. While the most widely used strategies make use of a manual and grid searches, Bergstra and Bengio (2012) propose that random searches for hyperparameters are most efficient and effective, as this method of optimization does not treat each hyperparameter with equal importance, therefore not wasting resources optimizing a value which does not play an crucial role to the overall function.

In closing, the future directions of this work may expand upon this novel and promising approach which serves as an important initial step in developing a deep learning approach for voxel-wise functional brain parcellation.

References

Bengio, Y. (2009). Learning deep architectures for AI. *Foundations and trends in Machine Learning*, 1-127.

Bengio, Y., & LeCun, Y. (2007). Scaling learning algorithms towards AI. *Large-Scale Kernel Machines*, 34.

Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (2007). Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 153.

Bergstra, J., & Bengio, Y. (2012). Random Search for Hyper-Parameter Optimization. *Journal of Machine Learning Research*, 281-305.

Bergstra, J., Bastien, F., Turian, J., Pascanu, R., & Delalleau, O. (2010). *Deep Learning on GPUs with Theano.* Montreal: Universite de Montreal.

Berry, M. J., & Linoff, G. (1997). *Data mining techniques.* John Wiley & Sons, Inc.

Biswal, B., Zerrin Yetkin, F., Haughton, V. M., & Hyde, J. S. (1995). Functional connectivity in the motor cortex of resting human brain using echo-planar mri. *Magnetic resonance in medicine, 34*(4), 537-541.

Blum, A. L., & Rivest, R. L. (1992). Training a 3-node neural network is NP-complete. *Neural Networks*, 117-127.

Bohland, J. W., Bokil, H., Allen, C. B., & Mitra, P. P. (2009). The brain atlas concordance problem: quantitative comparison of anatomical parcellations. *PLoS One, 4*(9), e7200.

Bourlard, H., & Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biological cybernetics, 59*(4-5), 291-294.

Cohen, A. L., Fair, D. A., Dosenbach, N. U., Miezin, F. M., Dierker, D., Van Essen, D. C., et al. (2008). Defining functional areas in individual human brains using resting functional connectivity MRI. *Neuroimage, 41*(1), 45-57.

Collobert, R., & J. Weston, J. (2008). A unified architecture for natural language. *The Twenty-fifth International Conference on Machine Learning* (pp. 160–167). Helsinki: AMC.

Devlin, H. (2008). *What is Functional Magnetic Resonance Imaging (fMRI)?* Retrieved from PsychCentral: http://psychcentral.com/lib/what-is-functional-magnetic-resonance-imaging-fmri

Fukushima, K. (1980). Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics, 36*(4), 193-202.

Galen, A., & Jianfeng, G. (2007). Scalable Training of L1-Regularized Log-Linear Models. *International Conference on Machine Learning.*

Geman, S., Bienenstock, E., & Doursat, R. (1992). Neural Networks and the Bias/Variance Dilemma. *Neural Computation*, 1-58.

Hadsell, R., Erkan, A., Sermanet, P., Scoffier, M., Muller, U., & LeCun, Y. (2008). Deep belief net learning in a long-range vision system for autonomous offroad. *Intelligent Robots and Systems*, 628–633.

Heaton, J. (2008). *Introduction to Neural Networks for Java, Second Edition.* St. Louis: Heaton Research, Inc.

Hebb, D. O. (1949). *The Organization of Behavior.* New York: Wiley.

Henley, S., Bates, G., & Tabriz, S. (2005). Biomarkers for neurodegenerative diseases. *Current Opinion in Neurology, 18*(6), 698-705.

Hinton, G. E. (2007). Learning multiple layers of representation. *Trends in cognitive sciences*, 428-434.

Hinton, G. E. (2010). *A Practical Guide to Training Restricted Boltzmann Machines.* University of Toronto.

Hinton, G. E., & Salakhutdinov, R. R. (2006). Reducing the dimensionality of data with neural networks. *Science, 313*(5786), 504-507.

Hinton, G. E., Osindero, S., & Teh, Y. W. (2006). A fast learning algorithm for deep belief nets. *Neural computation, 18*(7), 1527-1554.

Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen Netzen. *Diploma thesis*. Institut f. Informatik, Technische Univ. Munich.

Huang, S., Li, J., Sun, L., Liu, J., Wu, T., Chen, K., et al. (2009). Learning Brain Connectivity of Alzheimer's Disease from Neuroimaging Data. *NIPS*, 808-816.

Jones, N. (2014). Computer science: The learning machines. *Nature*, 146–148.

Kelly, C., Toro, R., Di Martino, A., Cox, C. L., Bellec, P., Castellanos, F. X., et al. (2012). A convergent functional architecture of the insula emerges across imaging modalities. *Neuroimage, 61*(4), 1129-1142.

Larochelle, H., Bengio, Y., Louradour, J., & Lamblin, P. (2009). Exploring strategies for training deep neural networks. *The Journal of Machine Learning Research*, 1-40.

Le, Q., Ngiam, J., Coates, A., Lahiri, A., Prochnow, B., & Ng, A. (2011). On optimization methods for deep learning. *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, (pp. 265-272).

Lee, N., Laine, A. F., & Klein, A. (2011). Towards a deep learning approach to brain parcellation. *International Symposium on Biomedical Imaging* (pp. 321-324). Chicago, IL: IEEE.

McCulloch, W., & Pitts, W. (1943). A Logical Calculus of IDeas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics, 5*(4), 115 - 133.

McKeown, M. J., & Sejnowski, T. J. (1998). Independent component analysis of fMRI data: examining the assumptions. *Human brain mapping*, 368-372.

Minsky, M. L., & Papert, S. A. (1969). *Perceptrons.* Cambridge, MA: MIT Press.

Mourão-Miranda, J., Bokde, A. L., Born, C., Hampel, H., & Stetter, M. (2005). Classifying brain states and determining the discriminating activation patterns: support vector machine on functional MRI data. *Neuroimage, 28*(4), 980-995.

Ng, A., Ngiam, J., Foo, C., Mai, Y., & Suen, C. (2013, April 7). *UFLDL Tutorial*. Retrieved from Stanford Deep Learning: http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial

Park, H., & Friston, K. (2013). Structural and Functional Brain Networks: From Connections to Cognition. *Science, 342*(6158).

Pereira, F., Mitchell, T., & Botvi, M. (2009). Machine learning classifiers and fMRI: a tutorial overview. *Neuroimage*, 199-209.

Rosenblatt, F. (1962). *Principles of neurodynamics.* New York: Spartan.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1986). Learning representations by back-propagating errors. *Nature, 323*(6088), 533-536.

Salakhutdinov, R. R., & Hinton, G. E. (2008). Using deep belief nets to learn covariance. In J. K. Platt, *Advances in Neural Information Processing* (pp. 1249–1256). Cambridge, MA: MIT Press.

Salakhutdinov, R., & Murray, I. (2008). On the quantitative analysis of deep belief networks. *International Conference on Machine Learning.* Helsinki.

Schmah, T., Hinton, G. E., Zemel, R., Small, S., & Strother, S. (2008). Generative versus discriminative training of RBMs for classification of fMRI images. *NIPS*, 1409-1416.

Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, *4*(2), 234-242.

Sheline, Y. I., & Raichle, M. E. (2013). Resting state functional connectivity in preclinical Alzheimer's disease. *Biological psychiatry*, 340-347.

Shin, H. C., Orton, M. R., Collins, D. J., Doran, S. J., & Leach, M. O. (2013). Stacked Autoencoders for Unsupervised Feature Learning and Multiple Organ Detection in a Pilot Study Using 4D Patient Data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1930-1943.

Smolensky, P. (1986). Chapter 6: Information Processing in Dynamical Systems: Foundations of Harmony Theory. In D. E. Rumelhart, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition* (pp. 194 - 281). Cambridge, MA: MIT Press.

Tesauro, G. (1992). Practical issues in temporal difference learning. *Machine Learning*, 257–277.

The Human Connectome Project. (2014). *The National Institute of Health Human Connectome Project*. Retrieved from Laboratory of Neuro Imaging and Martinos Center for Biomedical Imaging: http://www.humanconnectomeproject.org/

Werbos, P. J. (1974). Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences. *PhD Thesis*. Harvard University.

Zhong, Y., Wang, H., Lu, G., Zhang, Z., Jiao, Q., & Liu, Y. (2009). Detecting functional connectivity in fMRI using PCA and regression analysis. *Brain topography*, 134-144.

Appendices

Appendix A: Dice coefficients comparing parcellations acquired from autoencoders trained and tested with 2 hidden layers versus the ground truth parcellation provided by anatomists. The **bolded** row represents the parcellation with the maximum accuracy.

Units per Hidden Layer		D_c
200	200	0.515747
200	400	0.538371
200	600	0.522534
200	800	0.49086
200	1000	0.506697
400	200	0.540633
400	400	0.508959
400	600	0.506697
400	800	0.529321
400	1000	0.533846
600	200	0.454661
600	400	0.486335
600	600	0.486335
600	800	0.529321
600	**1000**	**0.551946**
800	200	0.506697
800	400	0.527059
800	600	0.49991
800	800	0.479548
800	1000	0.486335
1000	200	0.502172
1000	400	0.488597
1000	600	0.495385
1000	800	0.522534
1000	1000	0.479548

Appendix B: Dice coefficients comparing parcellations acquired from autoencoders trained and tested with 3 hidden layers versus the ground truth parcellation provided by anatomists. The **bolded** row represents the parcellation with the maximum accuracy.

Units per Hidden Layer			D_C
200	200	200	0.52262
200	200	400	0.45249
200	200	600	0.54751
200	200	800	0.51584
200	200	1000	0.61086
200	400	200	0.51584
200	400	400	0.45928
200	400	600	0.51584
200	400	800	0.50452
200	400	1000	0.52036
200	600	200	0.49095
200	600	400	0.53846
200	600	600	0.47285
200	600	800	0.51584
200	600	1000	0.45249
200	800	200	0.54977
200	800	400	0.57014
200	800	600	0.57919
200	800	800	0.5905
200	800	1000	0.57014
200	1000	200	0.59729
200	1000	400	0.50905
200	1000	600	0.52941
200	1000	800	0.53167
200	1000	1000	0.51131
400	200	200	0.56561
400	200	400	0.56787
400	200	600	0.58371
400	200	800	0.56787
400	200	1000	0.58824
400	400	200	0.58145
400	400	400	0.55656
400	400	600	0.55656
400	400	800	0.59276
400	400	1000	0.56787
400	600	200	0.53394
400	600	400	0.6086
400	600	600	0.60407
400	600	800	0.61086
400	600	1000	0.55656
400	800	200	0.55204
400	800	400	0.54977
400	800	600	0.57692
400	800	800	0.53846
400	800	1000	0.60633
400	1000	200	0.5905
400	1000	400	0.5543
400	1000	600	0.58145
400	1000	800	0.55656
400	**1000**	**1000**	**0.63801**
600	200	200	0.52489
600	200	400	0.57014
600	200	600	0.58824
600	200	800	0.60407
600	200	1000	0.55656
600	400	200	0.62896
600	400	400	0.54751
600	400	600	0.55656
600	400	800	0.50452
600	400	1000	0.54751
600	600	200	0.56787
600	600	400	0.57014
600	600	600	0.54751
600	600	800	0.58824
600	600	1000	0.6086
600	800	200	0.63122
600	800	400	0.57466
600	800	600	0.54977
600	800	800	0.54299

600	800	1000	0.61086
600	1000	200	0.57466
600	1000	400	0.57692
600	1000	600	0.54072
600	1000	800	0.58371
600	1000	1000	0.62443
800	200	200	0.54751
800	200	400	0.55656
800	200	600	0.5362
800	200	800	0.60407
800	200	1000	0.57466
800	400	200	0.53394
800	400	400	0.55656
800	400	600	0.58597
800	400	800	0.57466
800	400	1000	0.61991
800	600	200	0.54299
800	600	400	0.55204
800	600	600	0.55882
800	600	800	0.6086
800	600	1000	0.58145
800	800	200	0.53846
800	800	400	0.5543
800	800	600	0.61991
800	800	800	0.59955
800	800	1000	0.57466
800	1000	200	0.58597
800	1000	400	0.57014

800	1000	600	0.60633
800	1000	800	0.62896
800	1000	1000	0.60407
1000	200	200	0.5362
1000	200	400	0.53846
1000	200	600	0.5905
1000	200	800	0.62443
1000	200	1000	0.5543
1000	400	200	0.55656
1000	400	400	0.58145
1000	400	600	0.61312
1000	400	800	0.5543
1000	400	1000	0.57692
1000	600	200	0.58597
1000	600	400	0.56335
1000	600	600	0.52262
1000	600	800	0.5905
1000	600	1000	0.5181
1000	800	200	0.56787
1000	800	400	0.53846
1000	800	600	0.54525
1000	800	800	0.54751
1000	800	1000	0.57692
1000	1000	200	0.54751
1000	1000	400	0.54751
1000	1000	600	0.60181
1000	1000	800	0.55204
1000	1000	1000	0.58145

Appendix C: Dice coefficients comparing parcellations acquired from autoencoders trained and tested with 4 hidden layers versus the ground truth parcellation provided by anatomists. The **bolded** row represents the parcellation with the maximum accuracy.

Units per Hidden Layer				D_c
200	200	200	200	0.579186
200	200	200	400	0.597285
200	200	200	600	0.631222
200	200	200	800	0.617647
200	200	200	1000	0.579186
200	200	400	200	0.58371
200	200	400	400	0.59276
200	200	400	600	0.556561
200	200	400	800	0.665158
200	200	400	1000	0.606335
200	200	600	200	0.638009
200	200	600	400	0.556561
200	200	600	600	0.576923
200	200	600	800	0.599548
200	200	600	1000	0.595023
200	200	800	200	0.588235
200	200	800	400	0.658371
200	200	800	600	0.61991
200	200	800	800	0.613122
200	200	800	1000	0.665158
200	200	1000	200	0.608597
200	200	1000	400	0.597285
200	200	1000	600	0.595023
200	200	1000	800	0.588235
200	200	1000	1000	0.631222
200	400	200	200	0.606335
200	400	200	400	0.565611
200	400	200	600	0.556561
200	400	200	800	0.633484
200	400	200	1000	0.567873
200	400	400	200	0.624434
200	**400**	**400**	**400**	**0.676471**
200	400	400	600	0.61086
200	400	400	800	0.58371
200	400	400	1000	0.61086
200	400	600	200	0.60181
200	400	600	400	0.563348
200	400	600	600	0.635747
200	400	600	800	0.581448
200	400	600	1000	0.565611
200	400	800	200	0.561086
200	400	800	400	0.638009
200	400	800	600	0.622172
200	400	800	800	0.606335
200	400	800	1000	0.599548
200	400	1000	200	0.61086
200	400	1000	400	0.615385
200	400	1000	600	0.595023
200	400	1000	800	0.608597
200	400	1000	1000	0.597285
200	600	200	200	0.59276
200	600	200	400	0.628959
200	600	200	600	0.581448
200	600	200	800	0.617647
200	600	200	1000	0.608597
200	600	400	200	0.647059
200	600	400	400	0.61991
200	600	400	600	0.561086
200	600	400	800	0.635747
200	600	400	1000	0.585973
200	600	600	200	0.613122
200	600	600	400	0.606335
200	600	600	600	0.649321
200	600	600	800	0.58371
200	600	600	1000	0.633484
200	600	800	200	0.581448
200	600	800	400	0.61086
200	600	800	600	0.570136
200	600	800	800	0.651584
200	600	800	1000	0.60181
200	600	1000	200	0.590498
200	600	1000	400	0.617647
200	600	1000	600	0.597285

200	600	1000	800	0.581448
200	600	1000	1000	0.651584
200	800	200	200	0.60181
200	800	200	400	0.576923
200	800	200	600	0.574661
200	800	200	800	0.588235
200	800	200	1000	0.61086
200	800	400	200	0.631222
200	800	400	400	0.628959
200	800	400	600	0.606335
200	800	400	800	0.617647
200	800	400	1000	0.599548
200	800	600	200	0.628959
200	800	600	400	0.595023
200	800	600	600	0.606335
200	800	600	800	0.613122
200	800	600	1000	0.613122
200	800	800	200	0.585973
200	800	800	400	0.608597
200	800	800	600	0.606335
200	800	800	800	0.640271
200	800	800	1000	0.649321
200	800	1000	200	0.579186
200	800	1000	400	0.613122
200	800	1000	600	0.576923
200	800	1000	800	0.633484
200	800	1000	1000	0.61086
200	1000	200	200	0.61086
200	1000	200	400	0.595023
200	1000	200	600	0.617647
200	1000	200	800	0.61086
200	1000	200	1000	0.59276
200	1000	400	200	0.635747
200	1000	400	400	0.658371
200	1000	400	600	0.552036
200	1000	400	800	0.631222
200	1000	400	1000	0.58371
200	1000	600	200	0.585973
200	1000	600	400	0.61991
200	1000	600	600	0.626697
200	1000	600	800	0.58371

200	1000	600	1000	0.60181
200	1000	800	200	0.638009
200	1000	800	400	0.595023
200	1000	800	600	0.588235
200	1000	800	800	0.563348
200	1000	800	1000	0.642534
200	1000	1000	200	0.61991
200	1000	1000	400	0.665158
200	1000	1000	600	0.574661
200	1000	1000	800	0.622172
200	1000	1000	1000	0.581448
400	200	200	200	0.622172
400	200	200	400	0.653846
400	200	200	600	0.606335
400	200	200	800	0.570136
400	200	200	1000	0.61086
400	200	400	200	0.606335
400	200	400	400	0.613122
400	200	400	600	0.613122
400	200	400	800	0.60181
400	200	400	1000	0.604072
400	200	600	200	0.574661
400	200	600	400	0.631222
400	200	600	600	0.585973
400	200	600	800	0.608597
400	200	600	1000	0.570136
400	200	800	200	0.585973
400	200	800	400	0.61991
400	200	800	600	0.588235
400	200	800	800	0.631222
400	200	800	1000	0.626697
400	200	1000	200	0.588235
400	200	1000	400	0.58371
400	200	1000	600	0.58371
400	200	1000	800	0.662896
400	200	1000	1000	0.624434
400	400	200	200	0.563348
400	400	200	400	0.572398
400	400	200	600	0.579186
400	400	200	800	0.608597
400	400	200	1000	0.631222

400	400	400	200	0.570136
400	400	400	400	0.622172
400	400	400	600	0.567873
400	400	400	800	0.59276
400	400	400	1000	0.61086
400	400	600	200	0.581448
400	400	600	400	0.595023
400	400	600	600	0.61086
400	400	600	800	0.597285
400	400	600	1000	0.61086
400	400	800	200	0.597285
400	400	800	400	0.581448
400	400	800	600	0.597285
400	400	800	800	0.638009
400	400	800	1000	0.595023
400	400	1000	200	0.595023
400	400	1000	400	0.540724
400	400	1000	600	0.585973
400	400	1000	800	0.642534
400	400	1000	1000	0.635747
400	600	200	200	0.626697
400	600	200	400	0.640271
400	600	200	600	0.595023
400	600	200	800	0.599548
400	600	200	1000	0.576923
400	600	400	200	0.595023
400	600	400	400	0.61086
400	600	400	600	0.608597
400	600	400	800	0.588235
400	600	400	1000	0.59276
400	600	600	200	0.656109
400	600	600	400	0.60181
400	600	600	600	0.644796
400	600	600	800	0.617647
400	600	600	1000	0.588235
400	600	800	200	0.61086
400	600	800	400	0.647059
400	600	800	600	0.599548
400	600	800	800	0.581448
400	600	800	1000	0.628959
400	600	1000	200	0.613122
400	600	1000	400	0.595023
400	600	1000	600	0.572398
400	600	1000	800	0.615385
400	600	1000	1000	0.570136
400	800	200	200	0.595023
400	800	200	400	0.581448
400	800	200	600	0.576923
400	800	200	800	0.595023
400	800	200	1000	0.61991
400	800	400	200	0.633484
400	800	400	400	0.615385
400	800	400	600	0.617647
400	800	400	800	0.574661
400	800	400	1000	0.570136
400	800	600	200	0.608597
400	800	600	400	0.599548
400	800	600	600	0.608597
400	800	600	800	0.628959
400	800	600	1000	0.61991
400	800	800	200	0.558824
400	800	800	400	0.588235
400	800	800	600	0.597285
400	800	800	800	0.597285
400	800	800	1000	0.595023
400	800	1000	200	0.576923
400	800	1000	400	0.615385
400	800	1000	600	0.617647
400	800	1000	800	0.60181
400	800	1000	1000	0.615385
400	1000	200	200	0.59276
400	1000	200	400	0.615385
400	1000	200	600	0.565611
400	1000	200	800	0.590498
400	1000	200	1000	0.579186
400	1000	400	200	0.60181
400	1000	400	400	0.606335
400	1000	400	600	0.649321
400	1000	400	800	0.61086
400	1000	400	1000	0.574661
400	1000	600	200	0.628959
400	1000	600	400	0.597285

400	1000	600	600	0.576923
400	1000	600	800	0.61991
400	1000	600	1000	0.660633
400	1000	800	200	0.606335
400	1000	800	400	0.626697
400	1000	800	600	0.615385
400	1000	800	800	0.61991
400	1000	800	1000	0.61086
400	1000	1000	200	0.585973
400	1000	1000	400	0.617647
400	1000	1000	600	0.588235
400	1000	1000	800	0.638009
400	1000	1000	1000	0.635747
600	200	200	200	0.561086
600	200	200	400	0.590498
600	200	200	600	0.597285
600	200	200	800	0.599548
600	200	200	1000	0.576923
600	200	400	200	0.576923
600	200	400	400	0.576923
600	200	400	600	0.613122
600	200	400	800	0.585973
600	200	400	1000	0.581448
600	200	600	200	0.597285
600	200	600	400	0.606335
600	200	600	600	0.613122
600	200	600	800	0.585973
600	200	600	1000	0.579186
600	200	800	200	0.61086
600	200	800	400	0.588235
600	200	800	600	0.579186
600	200	800	800	0.595023
600	200	800	1000	0.597285
600	200	1000	200	0.597285
600	200	1000	400	0.613122
600	200	1000	600	0.588235
600	200	1000	800	0.633484
600	200	1000	1000	0.590498
600	400	200	200	0.570136
600	400	200	400	0.572398
600	400	200	600	0.59276
600	400	200	800	0.617647
600	400	200	1000	0.590498
600	400	400	200	0.585973
600	400	400	400	0.61991
600	400	400	600	0.581448
600	400	400	800	0.626697
600	400	400	1000	0.597285
600	400	600	200	0.563348
600	400	600	400	0.588235
600	400	600	600	0.633484
600	400	600	800	0.567873
600	400	600	1000	0.595023
600	400	800	200	0.613122
600	400	800	400	0.576923
600	400	800	600	0.58371
600	400	800	800	0.565611
600	400	800	1000	0.572398
600	400	1000	200	0.581448
600	400	1000	400	0.638009
600	400	1000	600	0.563348
600	400	1000	800	0.576923
600	400	1000	1000	0.606335
600	600	200	200	0.552036
600	600	200	400	0.635747
600	600	200	600	0.59276
600	600	200	800	0.597285
600	600	200	1000	0.615385
600	600	400	200	0.613122
600	600	400	400	0.595023
600	600	400	600	0.597285
600	600	400	800	0.585973
600	600	400	1000	0.604072
600	600	600	200	0.585973
600	600	600	400	0.595023
600	600	600	600	0.563348
600	600	600	800	0.588235
600	600	600	1000	0.579186
600	600	800	200	0.606335
600	600	800	400	0.599548
600	600	800	600	0.581448
600	600	800	800	0.60181

600	600	800	1000	0.633484
600	600	1000	200	0.628959
600	600	1000	400	0.626697
600	600	1000	600	0.608597
600	600	1000	800	0.581448
600	600	1000	1000	0.58371
600	800	200	200	0.615385
600	800	200	400	0.631222
600	800	200	600	0.647059
600	800	200	800	0.574661
600	800	200	1000	0.595023
600	800	400	200	0.554299
600	800	400	400	0.61991
600	800	400	600	0.549774
600	800	400	800	0.622172
600	800	400	1000	0.574661
600	800	600	200	0.581448
600	800	600	400	0.615385
600	800	600	600	0.579186
600	800	600	800	0.58371
600	800	600	1000	0.565611
600	800	800	200	0.613122
600	800	800	400	0.576923
600	800	800	600	0.658371
600	800	800	800	0.585973
600	800	800	1000	0.613122
600	800	1000	200	0.59276
600	800	1000	400	0.563348
600	800	1000	600	0.579186
600	800	1000	800	0.599548
600	800	1000	1000	0.590498
600	1000	200	200	0.604072
600	1000	200	400	0.579186
600	1000	200	600	0.565611
600	1000	200	800	0.563348
600	1000	200	1000	0.585973
600	1000	400	200	0.624434
600	1000	400	400	0.574661
600	1000	400	600	0.579186
600	1000	400	800	0.597285
600	1000	400	1000	0.588235

600	1000	600	200	0.558824
600	1000	600	400	0.604072
600	1000	600	600	0.622172
600	1000	600	800	0.558824
600	1000	600	1000	0.572398
600	1000	800	200	0.574661
600	1000	800	400	0.579186
600	1000	800	600	0.597285
600	1000	800	800	0.595023
600	1000	800	1000	0.60181
600	1000	1000	200	0.58371
600	1000	1000	400	0.576923
600	1000	1000	600	0.633484
600	1000	1000	800	0.61086
600	1000	1000	1000	0.574661
800	200	200	200	0.563348
800	200	200	400	0.547511
800	200	200	600	0.570136
800	200	200	800	0.647059
800	200	200	1000	0.572398
800	200	400	200	0.558824
800	200	400	400	0.608597
800	200	400	600	0.597285
800	200	400	800	0.536199
800	200	400	1000	0.615385
800	200	600	200	0.561086
800	200	600	400	0.608597
800	200	600	600	0.58371
800	200	600	800	0.572398
800	200	600	1000	0.565611
800	200	800	200	0.58371
800	200	800	400	0.567873
800	200	800	600	0.595023
800	200	800	800	0.595023
800	200	800	1000	0.549774
800	200	1000	200	0.563348
800	200	1000	400	0.597285
800	200	1000	600	0.61086
800	200	1000	800	0.581448
800	200	1000	1000	0.595023
800	400	200	200	0.574661

800	400	200	400	0.554299
800	400	200	600	0.595023
800	400	200	800	0.613122
800	400	200	1000	0.581448
800	400	400	200	0.567873
800	400	400	400	0.58371
800	400	400	600	0.558824
800	400	400	800	0.590498
800	400	400	1000	0.581448
800	400	600	200	0.567873
800	400	600	400	0.570136
800	400	600	600	0.552036
800	400	600	800	0.585973
800	400	600	1000	0.538462
800	400	800	200	0.563348
800	400	800	400	0.558824
800	400	800	600	0.563348
800	400	800	800	0.61086
800	400	800	1000	0.59276
800	400	1000	200	0.599548
800	400	1000	400	0.606335
800	400	1000	600	0.588235
800	400	1000	800	0.567873
800	400	1000	1000	0.581448
800	600	200	200	0.631222
800	600	200	400	0.599548
800	600	200	600	0.574661
800	600	200	800	0.626697
800	600	200	1000	0.554299
800	600	400	200	0.513575
800	600	400	400	0.61991
800	600	400	600	0.572398
800	600	400	800	0.59276
800	600	400	1000	0.617647
800	600	600	200	0.631222
800	600	600	400	0.558824
800	600	600	600	0.576923
800	600	600	800	0.599548
800	600	600	1000	0.606335
800	600	800	200	0.570136
800	600	800	400	0.570136

800	600	800	600	0.595023
800	600	800	800	0.626697
800	600	800	1000	0.59276
800	600	1000	200	0.61086
800	600	1000	400	0.581448
800	600	1000	600	0.608597
800	600	1000	800	0.59276
800	600	1000	1000	0.581448
800	800	200	200	0.579186
800	800	200	400	0.61086
800	800	200	600	0.558824
800	800	200	800	0.585973
800	800	200	1000	0.552036
800	800	400	200	0.585973
800	800	400	400	0.558824
800	800	400	600	0.60181
800	800	400	800	0.572398
800	800	400	1000	0.567873
800	800	600	200	0.558824
800	800	600	400	0.58371
800	800	600	600	0.576923
800	800	600	800	0.59276
800	800	600	1000	0.576923
800	800	800	200	0.59276
800	800	800	400	0.588235
800	800	800	600	0.599548
800	800	800	800	0.638009
800	800	800	1000	0.608597
800	800	1000	200	0.588235
800	800	1000	400	0.60181
800	800	1000	600	0.599548
800	800	1000	800	0.579186
800	800	1000	1000	0.574661
800	1000	200	200	0.624434
800	1000	200	400	0.547511
800	1000	200	600	0.563348
800	1000	200	800	0.60181
800	1000	200	1000	0.565611
800	1000	400	200	0.547511
800	1000	400	400	0.604072
800	1000	400	600	0.581448

800	1000	400	800	0.590498
800	1000	400	1000	0.599548
800	1000	600	200	0.588235
800	1000	600	400	0.59276
800	1000	600	600	0.567873
800	1000	600	800	0.570136
800	1000	600	1000	0.595023
800	1000	800	200	0.60181
800	1000	800	400	0.540724
800	1000	800	600	0.59276
800	1000	800	800	0.585973
800	1000	800	1000	0.567873
800	1000	1000	200	0.638009
800	1000	1000	400	0.574661
800	1000	1000	600	0.565611
800	1000	1000	800	0.579186
800	1000	1000	1000	0.556561
1000	200	200	200	0.585973
1000	200	200	400	0.581448
1000	200	200	600	0.552036
1000	200	200	800	0.613122
1000	200	200	1000	0.58371
1000	200	400	200	0.540724
1000	200	400	400	0.597285
1000	200	400	600	0.644796
1000	200	400	800	0.565611
1000	200	400	1000	0.58371
1000	200	600	200	0.540724
1000	200	600	400	0.552036
1000	200	600	600	0.581448
1000	200	600	800	0.576923
1000	200	600	1000	0.588235
1000	200	800	200	0.640271
1000	200	800	400	0.572398
1000	200	800	600	0.565611
1000	200	800	800	0.567873
1000	200	800	1000	0.570136
1000	200	1000	200	0.576923
1000	200	1000	400	0.552036
1000	200	1000	600	0.590498
1000	200	1000	800	0.606335
1000	200	1000	1000	0.615385
1000	400	200	200	0.549774
1000	400	200	400	0.556561
1000	400	200	600	0.60181
1000	400	200	800	0.563348
1000	400	200	1000	0.643222
1000	400	400	200	0.570824
1000	400	400	400	0.588923
1000	400	400	600	0.611548
1000	400	400	800	0.618335
1000	400	400	1000	0.582136
1000	400	600	200	0.582136
1000	400	600	400	0.607023
1000	400	600	600	0.638697
1000	400	600	800	0.60476
1000	400	600	1000	0.62286
1000	400	800	200	0.593448
1000	400	800	400	0.620597
1000	400	800	600	0.60476
1000	400	800	800	0.593448
1000	400	800	1000	0.591186
1000	400	1000	200	0.62286
1000	400	1000	400	0.570824
1000	400	1000	600	0.597973
1000	400	1000	800	0.564036
1000	400	1000	1000	0.597973
1000	600	200	200	0.570824
1000	600	200	400	0.61381
1000	600	200	600	0.584398
1000	600	200	800	0.579873
1000	600	200	1000	0.570824
1000	600	400	200	0.59571
1000	600	400	400	0.588923
1000	600	400	600	0.60476
1000	600	400	800	0.588923
1000	600	400	1000	0.60476
1000	600	600	200	0.578498
1000	600	600	400	0.587548
1000	600	600	600	0.576235
1000	600	600	800	0.58076
1000	600	600	1000	0.555873

1000	600	800	200	0.558136
1000	600	800	400	0.583023
1000	600	800	600	0.58981
1000	600	800	800	0.528724
1000	600	800	1000	0.58076
1000	600	1000	200	0.573973
1000	600	1000	400	0.555873
1000	600	1000	600	0.626009
1000	600	1000	800	0.562661
1000	600	1000	1000	0.553611
1000	800	200	200	0.567186
1000	800	200	400	0.544561
1000	800	200	600	0.596597
1000	800	200	800	0.576235
1000	800	200	1000	0.58981
1000	800	400	200	0.587548
1000	800	400	400	0.567186
1000	800	400	600	0.562661
1000	800	400	800	0.612434
1000	800	400	1000	0.535511
1000	800	600	200	0.551348
1000	800	600	400	0.58981
1000	800	600	600	0.553611
1000	800	600	800	0.535511
1000	800	600	1000	0.592072
1000	800	800	200	0.569448
1000	800	800	400	0.606647
1000	800	800	600	0.586285
1000	800	800	800	0.570448
1000	800	800	1000	0.640584

1000	800	1000	200	0.59081
1000	800	1000	400	0.565923
1000	800	1000	600	0.563661
1000	800	1000	800	0.577235
1000	800	1000	1000	0.59986
1000	1000	200	200	0.620222
1000	1000	200	400	0.617959
1000	1000	200	600	0.595335
1000	1000	200	800	0.606647
1000	1000	200	1000	0.588548
1000	1000	400	200	0.617959
1000	1000	400	400	0.584023
1000	1000	400	600	0.595335
1000	1000	400	800	0.602122
1000	1000	400	1000	0.602122
1000	1000	600	200	0.574973
1000	1000	600	400	0.597597
1000	1000	600	600	0.595335
1000	1000	600	800	0.629271
1000	1000	600	1000	0.638321
1000	1000	800	200	0.568186
1000	1000	800	400	0.602122
1000	1000	800	600	0.565923
1000	1000	800	800	0.622484
1000	1000	800	1000	0.59986
1000	1000	1000	200	0.59986
1000	1000	1000	400	0.584023
1000	1000	1000	600	0.606647
1000	1000	1000	800	0.59986
1000	1000	1000	1000	0.58176

Appendix D: Dice coefficients comparing parcellations acquired from autoencoders trained and tested with 5 hidden layers versus the ground truth parcellation provided by anatomists. The **bolded** row represents the parcellation with the maximum accuracy.

Units per Hidden Layer					D_c
200	200	200	200	200	0.631222
200	200	200	200	600	0.529412
200	200	200	200	1000	0.597285
200	200	200	600	200	0.599548
200	200	200	600	600	0.613122
200	200	200	600	1000	0.604072
200	200	200	1000	200	0.59276
200	200	200	1000	600	0.617647
200	200	200	1000	1000	0.579186
200	200	600	200	200	0.588235
200	200	600	200	600	0.615385
200	200	600	200	1000	0.572398
200	200	600	600	200	0.606335
200	200	600	600	600	0.61086
200	200	600	600	1000	0.626697
200	200	600	1000	200	0.567873
200	200	600	1000	600	0.549774
200	200	600	1000	1000	0.529412
200	200	1000	200	200	0.585973
200	200	1000	200	600	0.622172
200	200	1000	200	1000	0.628959
200	200	1000	600	200	0.538462
200	200	1000	600	600	0.633484
200	200	1000	600	1000	0.59276
200	200	1000	1000	200	0.613122
200	200	1000	1000	600	0.61991
200	200	1000	1000	1000	0.624434
200	600	200	200	200	0.61086
200	600	200	200	600	0.59276
200	600	200	200	1000	0.563348
200	600	200	600	200	0.570136
200	600	200	600	600	0.579186
200	600	200	600	1000	0.58371
200	600	200	1000	200	0.595023
200	600	200	1000	600	0.570136
200	600	200	1000	1000	0.624434
200	600	600	200	200	0.597285
200	600	600	200	600	0.615385
200	600	600	200	1000	0.633484
200	600	600	600	200	0.581448
200	600	600	600	600	0.576923
200	600	600	600	1000	0.570136
200	600	600	1000	200	0.61991
200	600	600	1000	600	0.628959
200	600	600	1000	1000	0.615385
200	600	1000	200	200	0.599548
200	600	1000	200	600	0.542986
200	600	1000	200	1000	0.60181
200	600	1000	600	200	0.604072
200	600	1000	600	600	0.590498
200	600	1000	600	1000	0.651584
200	600	1000	1000	200	0.631222
200	600	1000	1000	600	0.61991
200	600	1000	1000	1000	0.538462
200	1000	200	200	200	0.59276
200	1000	200	200	600	0.624434
200	1000	200	200	1000	0.561086
200	1000	200	600	200	0.590498
200	1000	200	600	600	0.563348
200	1000	200	600	1000	0.570136
200	1000	200	1000	200	0.558824
200	1000	200	1000	600	0.581448
200	1000	200	1000	1000	0.59276
200	1000	600	200	200	0.631222
200	1000	600	200	600	0.585973
200	1000	600	200	1000	0.624434
200	1000	600	600	200	0.527149
200	1000	600	600	600	0.613122
200	1000	600	600	1000	0.59276
200	1000	600	1000	200	0.595023
200	1000	600	1000	600	0.628959

200	1000	600	1000	1000	0.570136
200	1000	1000	200	200	0.651584
200	1000	1000	200	600	0.558824
200	1000	1000	200	1000	0.540724
200	1000	1000	600	200	0.561086
200	1000	1000	600	600	0.640271
200	1000	1000	600	1000	0.617647
200	1000	1000	1000	200	0.617647
200	1000	1000	1000	600	0.549774
200	1000	1000	1000	1000	0.613122
600	200	200	200	200	0.585973
600	200	200	200	600	0.565611
600	200	200	200	1000	0.604072
600	200	200	600	200	0.608597
600	200	200	600	600	0.628959
600	200	200	600	1000	0.613122
600	200	200	1000	200	0.606335
600	200	200	1000	600	0.61991
600	200	200	1000	1000	0.590498
600	200	600	200	200	0.628959
600	200	600	200	600	0.579186
600	200	600	200	1000	0.549774
600	200	600	600	200	0.599548
600	200	600	600	600	0.595023
600	200	600	600	1000	0.558824
600	200	600	1000	200	0.633484
600	200	600	1000	600	0.556561
600	200	600	1000	1000	0.590498
600	200	1000	200	200	0.579186
600	200	1000	200	600	0.622172
600	200	1000	200	1000	0.606335
600	200	1000	600	200	0.563348
600	200	1000	600	600	0.572398
600	200	1000	600	1000	0.576923
600	200	1000	1000	200	0.590498
600	200	1000	1000	600	0.579186
600	200	1000	1000	1000	0.617647
600	600	200	200	200	0.628959
600	600	200	200	600	0.5181
600	600	200	200	1000	0.570136
600	600	200	600	200	0.595023

600	600	200	600	600	0.606335
600	600	200	600	1000	0.588235
600	600	200	1000	200	0.581448
600	600	200	1000	600	0.58371
600	600	200	1000	1000	0.597285
600	600	600	200	200	0.536199
600	600	600	200	600	0.588235
600	600	600	200	1000	0.640271
600	600	600	600	200	0.624434
600	600	600	600	600	0.597285
600	600	600	600	1000	0.59276
600	600	600	1000	200	0.613122
600	600	600	1000	600	0.61086
600	600	600	1000	1000	0.554299
600	600	1000	200	200	0.61086
600	600	1000	200	600	0.558824
600	600	1000	200	1000	0.588235
600	600	1000	600	200	0.617647
600	600	1000	600	600	0.58371
600	600	1000	600	1000	0.647059
600	**600**	**1000**	**1000**	**200**	**0.678733**
600	600	1000	1000	600	0.61086
600	600	1000	1000	1000	0.638009
600	1000	200	200	200	0.59276
600	1000	200	200	600	0.624434
600	1000	200	200	1000	0.554299
600	1000	200	600	200	0.604072
600	1000	200	600	600	0.579186
600	1000	200	600	1000	0.567873
600	1000	200	1000	200	0.61086
600	1000	200	1000	600	0.595023
600	1000	200	1000	1000	0.59276
600	1000	600	200	200	0.531674
600	1000	600	200	600	0.635747
600	1000	600	200	1000	0.574661
600	1000	600	600	200	0.604072
600	1000	600	600	600	0.574661
600	1000	600	600	1000	0.59276
600	1000	600	1000	200	0.574661
600	1000	600	1000	600	0.581448
600	1000	600	1000	1000	0.608597

600	1000	1000	200	200	0.613122
600	1000	1000	200	600	0.613122
600	1000	1000	200	1000	0.606335
600	1000	1000	600	200	0.613122
600	1000	1000	600	600	0.59276
600	1000	1000	600	1000	0.588235
600	1000	1000	1000	200	0.622172
600	1000	1000	1000	600	0.624434
600	1000	1000	1000	1000	0.588235
1000	200	200	200	200	0.58371
1000	200	200	200	600	0.547511
1000	200	200	200	1000	0.633484
1000	200	200	600	200	0.561086
1000	200	200	600	600	0.552036
1000	200	200	600	1000	0.545249
1000	200	200	1000	200	0.617647
1000	200	200	1000	600	0.617647
1000	200	200	1000	1000	0.622172
1000	200	600	200	200	0.59276
1000	200	600	200	600	0.599548
1000	200	600	200	1000	0.522624
1000	200	600	600	200	0.558824
1000	200	600	600	600	0.552036
1000	200	600	600	1000	0.590498
1000	200	600	1000	200	0.558824
1000	200	600	1000	600	0.60181
1000	200	600	1000	1000	0.590498
1000	200	1000	200	200	0.549774
1000	200	1000	200	600	0.58371
1000	200	1000	200	1000	0.588235
1000	200	1000	600	200	0.574661
1000	200	1000	600	600	0.597285
1000	200	1000	600	1000	0.565611
1000	200	1000	1000	200	0.606335
1000	200	1000	1000	600	0.604072
1000	200	1000	1000	1000	0.60181
1000	600	200	200	200	0.595023
1000	600	200	200	600	0.590498
1000	600	200	200	1000	0.558824
1000	600	200	600	200	0.579186
1000	600	200	600	600	0.58371

1000	600	200	600	1000	0.576923
1000	600	200	1000	200	0.545249
1000	600	200	1000	600	0.527149
1000	600	200	1000	1000	0.597285
1000	600	600	200	200	0.554299
1000	600	600	200	600	0.574661
1000	600	600	200	1000	0.599548
1000	600	600	600	200	0.60181
1000	600	600	600	600	0.599548
1000	600	600	600	1000	0.585973
1000	600	600	1000	200	0.631222
1000	600	600	1000	600	0.628959
1000	600	600	1000	1000	0.640271
1000	600	1000	200	200	0.588235
1000	600	1000	200	600	0.59276
1000	600	1000	200	1000	0.608597
1000	600	1000	600	200	0.556561
1000	600	1000	600	600	0.558824
1000	600	1000	600	1000	0.579186
1000	600	1000	1000	200	0.608597
1000	600	1000	1000	600	0.626697
1000	600	1000	1000	1000	0.545249
1000	1000	200	200	200	0.588235
1000	1000	200	200	600	0.531674
1000	1000	200	200	1000	0.595023
1000	1000	200	600	200	0.61991
1000	1000	200	600	600	0.581448
1000	1000	200	600	1000	0.595023
1000	1000	200	1000	200	0.570136
1000	1000	200	1000	600	0.570136
1000	1000	200	1000	1000	0.613122
1000	1000	600	200	200	0.590498
1000	1000	600	200	600	0.608597
1000	1000	600	200	1000	0.597285
1000	1000	600	600	200	0.597285
1000	1000	600	600	600	0.622172
1000	1000	600	600	1000	0.61086
1000	1000	600	1000	200	0.563348
1000	1000	600	1000	600	0.613122
1000	1000	600	1000	1000	0.588235
1000	1000	1000	200	200	0.545249

1000	1000	1000	200	600	0.581448
1000	1000	1000	200	1000	0.617647
1000	1000	1000	600	200	0.536199
1000	1000	1000	600	600	0.626697

1000	1000	1000	600	1000	0.590498
1000	1000	1000	1000	200	0.585973
1000	1000	1000	1000	600	0.651584
1000	1000	1000	1000	1000	0.617647

Curriculum Vitae

Name:	Céline Gravelines
Post-secondary Education and Degrees:	The University of Western Ontario London, Ontario, Canada 2008-2012 Honors B.Sc.
Honours and Awards:	Google Anita Borg Memorial Scholarship Finalist 2012 Western Graduate Research Scholarship 2012-2013
Related Work Experience	Teaching Assistant The University of Western Ontario 2012-2013