2019-07-01

Adversarial Deep Neural Networks Effectively Remove Nonlinear Batch Effects from Gene-Expression Data

Jonathan Bryan Dayton
Brigham Young University

Adversarial Deep Neural Networks Effectively Remove Nonlinear Batch Effects from Gene-Expression Data

Jonathan Bryan Dayton

A thesis submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Master of Science

Stephen R. Piccolo, Chair
Mark Clement
David Wingate

Department of Biology

Brigham Young University

ABSTRACT

Adversarial Deep Neural Networks Effectively Remove Nonlinear Batch Effects from Gene-Expression Data

Jonathan Bryan Dayton
Department of Biology, BYU
Master of Science

Gene-expression profiling enables researchers to quantify transcription levels in cells, thus providing insight into functional mechanisms of diseases and other biological processes. However, because of the high dimensionality of these data and the sensitivity of measuring equipment, expression data often contains unwanted confounding effects that can skew analysis [1]. For example, collecting data in multiple runs causes nontrivial differences in the data (known as batch effects), known covariates that are not of interest to the study may have strong effects, and there may be large systemic effects when integrating multiple expression datasets. Additionally, many of these confounding effects represent higher-order interactions that may not be removable using existing techniques that identify linear patterns. We created Confounded to remove these effects from expression data. Confounded is an adversarial variational autoencoder that removes confounding effects while minimizing the amount of change to the input data. We tested the model on artificially constructed data and commonly used gene expression datasets and compared against other common batch adjustment algorithms. We also applied the model to remove cancer-type-specific signal from a pan-cancer expression dataset. Our software is publicly available at `https://github.com/jdayton3/Confounded`.

Keywords: batch effects, batch correction, gene expression, transcriptomics, deep learning, adversarial neural network, variational autoencoder

ACKNOWLEDGMENTS

Thanks to Dr. Stephen Piccolo, whose mentorship over the past four years has shaped my education much more than the classes I took. Thanks to Dr. Mark Clement and Dr. David Wingate for participating in my graduate committee and providing helpful feedback throughout my degree. Thanks to my office- and lab-mates, who let me bounce ideas off of them. Thanks to the BYU Department of Biology for funding my education, and specifically to Dr. Byron Adams for making the graduate experience more enjoyable, and to Gentri Glaittli for helping me to navigate the university requirements. Thanks to my grandmother, Jennette Hawkes, whose financial help came at a critical time in my education. Thanks to my parents, Minnie and Bryan Dayton, for teaching me the value of an education, and to my sisters Merinda, Louisa, Eliza, and Anna for their support. Thanks to Dr. Paul Horton, whose thorough peer review provided me with the idea for this project. Thanks to the researchers who generated the data for this project and who made it available.

Most of all, thanks to my wife, Zanna, for her love and support over the course of my graduate education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 Background

Gene expression data can be applied in many ways to advance our understanding of medicine and biology. For example, expression data has been applied to discover conserved genetic modules [2], to better understand the mechanisms of cardiovascular disease [3], to more accurately predict the clinical outcome in cancer patients [4], and to discover effective drugs for treating specific diseases [5]. Expression datasets are quite "wide," often containing tens of thousands of columns representing each gene in the human transcriptome. Because of the sensitive nature of these data (i.e. gene product levels in organisms may respond drastically to small environmental changes) and of the equipment used to quantify expression levels, expression data often contains unwanted confounding effects that can skew analysis. Three examples of this include (a) batch effects, (b) known covariates, and (c) dataset-level effects. (a) Batch effects occur when expression data are generated in multiple runs or multiple batches, and slight systemic differences occur during the different runs, such as different technicians operating the machine or slight temperature differences in the room. Batch effects are understood to have a nontrivial impact on high-throughput expression data [6]. In one study, researchers found that, contrary to previous knowledge, expression values from mice and humans clustered more closely by species than by tissue type [7]; however, referees showed in a rebuttal that when accounting for batch effects, these data actually clustered more closely by tissue type, as initially expected [8]. (b) Systemic bias can be even more pronounced when there are known covariates—for example, using data collected from different cancer types when cancer type is not of interest to the study [9]. (c) Finally, though batch effects within a dataset collected all by one lab at one time may be somewhat decreased experimentally by careful replication of experimental conditions, this is no longer possible when integrating different datasets; however, much better statistical significance can be achieved when analyzing larger datasets, so data integration is a key goal for transcriptomic analysis [10]. Each of these problems—batch effects, known covariates, and dataset integration—represents a case where data measurements are effected by some hidden variables and must be removed for effective analysis (see Figure 1).

Several methods exist for removing batch effects from gene expression datasets. Two commonly used methods are ComBat [11] and SVA [12]. ComBat uses an empirical Bayes method to estimate batch effect parameters and then uses linear regression to remove the effects, and SVA uses singular value decomposition to model batch effects which can then be accounted for in statistical analyses. Since both of these methods use linear methods to model confounding effects, they are not designed to account for nonlinear effects, such as the cascading changes in gene pathways in response to environmental stressors [13] (e.g. gene X transcripts increase and gene Y transcripts decrease in response to rising temperature, and since both are now past a certain threshold, gene Z transcripts also increase, and so on). As machine learning becomes more common in biological research, these nonlinear confounding effects become more troublesome since many machine learning algorithms can successfully identify complex interactions between variables. For example, advances in neural networks have introduced new ways to account for higher-order, nonlinear relationships in data [14]. These networks have proven effective in removing irrelevant, domain-specific signal in credit rating, online reviews, and image recognition tasks [15]. Several recent studies [16, 17, 18] have applied neural networks to batch effects; however, several limitations complicate their usability on many real-world datasets: they require that the input data only contains two batches, that the batches are sufficiently large (we received errors when testing with a subset of MNIST where n=100), and that the batches are balanced. These requirements rarely hold in existing datasets; for example, the bladderbatch dataset used in the R sva package [19, 20] has 5 batches, only 57 samples, and between 4 and 19 samples per batch. Additionally, the metrics these recent studies have used to validate results don't quantitatively test whether complex interactions still remain in the data.

Artificial neural networks are a machine learning tool inspired by the way human brains function; input values pass through layers of linear and nonlinear functions, the final output values are measured against objectives, and the layers of functions are adjusted to bring the outputs closer to the objectives. This process is repeated until the outputs are sufficiently close to

their targets [21]. Research has shown that neural networks are effective in working with gene expression data; for example, neural networks have been applied to detect cancer and identify critical cancer genes [22], to infer gene-expression values from just the values of a few "landmark genes" [23], to extract biologically relevant latent spaces in RNA-Seq data [24], to reduce the dimensions of single-cell RNA-Seq data [25], to identify drug-repurposing targets using transcriptomic data [26], and to generate realistic synthetic biomedical data for other scientific studies [27]. Autoencoders are a type of neural network that encode and then reconstruct their input, and their traditional objective function is to construct the output to be as similar to the input as possible [28]. Neural networks have historically decreased in effectiveness when working with data from multiple research domains [29], in part because they may learn based on dataset-specific confounding effects (e.g. which researcher collected the data) instead of learning based on practically meaningful causal effects (e.g. which gene is consistently upregulated in a disease) [30]. Recently, researchers have experimented with discouraging neural networks from learning based on domain-specific information. They have accomplished this by splitting a network into two "adversarial" sub-networks with two competing objective functions: 1. to learn as much as possible about the input data and 2. to forget any patterns related to unimportant information [29, 31]. In this way, the input data is preserved as well as possible while removing unimportant information from the data. Louizos, et al. [15] used this type of adversarial dual objective function with a variational autoencoder and successfully removed domain-based variability in credit score, financial savings, and hospital admittance datasets.

In this study, we present Confounded, an adversarial autoencoder that identifies and removes confounding effects. We test the hypothesis that using an adversarial neural network can correct for confounding effects more completely than previous tools do. We also explore the extent to which confounders still remain in different datasets after adjustment with various algorithms, and we present a framework to assess the extent to which confounding effects remain after adjustment using various classification algorithms.

2 Methods

All our code has been made publicly available at `https://github.com/jdayton3/Confounded`, and all our data are available as described below.

2.1 Network Structure

We used an adversarial autoencoder network to model and remove the confounding effects. We structured this network in two parts: a variational autoencoder [15] to replicate the input (expression) data and a discriminator (also known as a classifier) to detect remaining confounding effects in the autoencoder's output. By penalizing the autoencoder for the discriminator's success, the autoencoder subnetwork learned over the course of training to output the expression data with confounding effects minimized. We implemented the neural network in TensorFlow 1.11.0 [**?**] with Python 3.6 [32]. All layers in the network were fully connected and all activation functions were Rectified Linear Units (ReLUs) [33] except the final layers in the autoencoder and the discriminator, which used the sigmoid function.

2.1.1 Autoencoder

We implemented the variational autoencoder [15] described by Géron [34, Chapter 15]. This network has 2 hidden encoding layers and 2 decoding layers, each of size 500. The code size is 20. Each hidden layer is activated with the Exponential Linear Unit (ELU) function [35]. It is trained with the Adam optimizer [36] on reconstruction loss (sigmoid cross entropy) combined with latent loss (Kullback-Leibler, or KL, divergence [37]).

2.1.2 Discriminator

We trained the discriminator to determine the original batch of the autoencoder's output. The discriminator subnetwork consists of an input layer; four fully connected hidden layers of sizes

1024, 512, 512, and 128, respectively; and an output layer sized based on the number of batches. In order to combat overfitting and improve training, we also added 50%-probability dropout [38] (which prevents overfitting by dropping a random subset of layer inputs in each training iteration) and batch normalization [39] (which helps training by smoothing out the optimization landscape [40]) to each layer of the discriminator. These additions seemed to reduce overfitting in the discriminator.

2.1.3 Loss functions

We trained the network using three loss functions. First, we calculated the autoencoder's loss (L_A) by summing the reconstruction loss (sigmoid cross entropy between the autoencoder's input and output) and the latent loss (KL divergence [37] of the code layer). Second, we calculated the discriminator's loss (L_D) as sigmoid cross entropy between its output and a one-hot encoding of the samples' batch labels. Finally, we also trained the autoencoder layers on a combination of the two previous losses,

$$L_{dual} = L_A - \lambda(L_D) \tag{1}$$

The λ value represents a tradeoff parameter for tuning the network's tendency for more faithfully replicating the input or for more completely removing confounding effects. A higher λ value indicates that the network should remove confounding effects more aggressively, whereas a lower value indicates that the network should instead favor faithfully reconstructing the input data. We did not optimize L_A directly; instead we trained the autoencoder by optimizing L_{dual}.

2.1.4 Training

In all cases, we trained the network using the Adam Optimizer [36] with a training rate of 0.0001 for 10,000 iterations on mini-batches of size 100. In each iteration, we optimized on both L_D and L_{dual}. When optimizing L_D (i.e. training the discriminator), we froze the autoencoder's weights, and vice versa. We trained the network on a 2017 Dell XPS 15 9560 with an 8-core Intel

i7-7700HQ CPU and 16 GB of RAM. For each dataset, training typically took roughly 30 minutes to complete, including the time taken to load the input into memory and to save the output to disk.

2.2 Datasets

In order to test both theoretical and practical differences between Confounded and previous methods, we compared them for a variety of datasets of varying sizes and type of data measured.

2.2.1 MNIST

The MNIST digits dataset [41] is a database of images of handwritten digits that are size-normalized and centered. It contains 60,000 training images and 10,000 test images. We used MNIST so we could visually assess how well the true signal (in this case, the shape and digit of each handwritten digit image) was preserved after batch adjustment. In order to use this dataset, we flattened each 28 by 28 image into a 1D vector of size 784 and put each in a CSV file along with the accompanying digit information. We limited our dataset to only the 10,000 test images. Although convolutional layers are typically used when working with image data, we only used fully connected layers even for this image dataset. In this way, we show that the autoencoder is still able to find and represent spatial relationships without explicitly defining spatial relationships in the model while testing the same network we use on expression data, where no spatial relationships are inherent.

Because there is no batch information in the MNIST digits dataset, we had to simulate nonlinear confonding effects. To do so, we wrote a Python script to take the MNIST data in, apply a nonlinear effect, and output the adjusted data. We applied nonlinear effects by iteratively realizing vectors of normally distributed values, multiplying and adding these vectors to the "expression" vectors, and applying nonlinearity to the adjusted vector. We split the image data into two batches while keeping the batches balanced (5,000 images for each batch) and including

the same number of each digit in each batch. We applied the same random vectors to each image in a batch. Finally, we added random noise to each image in order to prevent images in a batch from being overly similar to each other.

2.2.2 Bladderbatch

The bladderbatch dataset is a microarray transcriptomic expression dataset from a study of patients with bladder cancer [42]. It has been made available as an R package [19] and is used in the documentation of the sva R package [20] to illustrate how to batch-adjust using ComBat. It contains expression values for 57 tissue samples with and without bladder cancer across 5 unbalanced batches. The dataset has a cancer status (cancerous vs. normal tissue) column, which we used for "true class" classification, and a batch column. Because bladderbatch is such a small dataset in terms of typical deep learning datasets, we selected it as a way to test whether our network was overfitting.

2.2.3 GSE37199

The GSE37199 dataset contains Affymetrix microarray gene-expression data from patients with advanced castration-resistant prostate cancer [43]. We accessed a version of this dataset from `http://doi.org/10.17605/OSF.IO/SSK3T` that was tidied as part of a curated compendium of human transcriptional biomarker data [44]. It contains expression values for 93 tissue samples categorized as either "advanced castration resistant" or "good prognosis." We used this cancer status variable as the "true class" for classification. It has two types of batch variables: "plate" and "centre." We adjusted against the "plate" variable because it was more balanced than "centre" (with counts of $\{43, 50\}$ compared to $\{27, 66\}$). The GSE37199 dataset represents a slightly larger dataset than bladderbatch, with only two batches that are closer to being balanced (with batch counts of $\{4, 5, 11, 18, 19\}$ and $\{43, 50\}$, respectively).

2.2.4 TCGA Pan-cancer Data

The Cancer Genome Atlas (TCGA) Pan-Cancer project produced expression data for thousands of tumors across many cancer types [45]. In a previous study [9], we classified this dataset based on the presence or absence of mutations in several known cancer genes. We attempted to adjust for the confounding effect of cancer type using ComBat prior to classification. However, we found that a strong nonlinear signal could still be identified by the Random Forests algorithm after adjustment. Here, we used the same version of the dataset that we tidied in this previous study (available at `https://osf.io/7xjdn/`). This dataset has RNA-Seq expression values for 9,365 samples across 25 distinct cancer types. We used this dataset as a way to test whether Confounded works on RNA-Seq data and to test whether we could remove confounding effects that ComBat cannot remove.

2.3 Comparison to other methods

We compared our method to two other batch adjusters: a scale adjuster and ComBat [11]. We implemented the scale adjuster in the R programming language, version 3.6.0 [46] using RStudio version 1.2.1194 [47]. It adjusts the data by linearly expanding or contracting each batch so all batches have the same range. We used the ComBat implementation from the R sva package [20] with some modifications to allow it to work on columns without variance in the MNIST dataset. We initially intended to test against the SVA [12] method but concluded that SVA is more suited for producing surrogate variables for further statistical research rather than removing those variables from the data. There are a number of other methods for batch adjustment that do and do not use deep neural networks (for example, see [12, 48, 16, 17]). Unfortunately, most of these methods lack a common interface and common assumptions that input datasets must meet. For these reasons, we compared Confounded only to the two methods listed above. Future batch adjustment research may benefit from standardization of input formats, user interfaces, and validation datasets.

2.4 Statistics and Metrics

2.4.1 Mean squared error

Mean squared error (MSE) is a measure of how much two vectors or matrices deviate from one another. It is commonly used as a loss value in autoencoders to make the network minimize the difference between the input and output values. We wanted to see how well Confounded and other batch correction software maintain patterns in the input data as measured by MSE.

2.4.2 Maximum mean discrepancy

In a recent paper, Shaham, et al. [16] used neural networks to remove batch effects. Instead of constraining the autoencoder to remove batch effects based on a discriminator, these researchers trained their network to minimize maximum mean discrepancy (MMD) between batches in an embedded layer of their network. We calculated MMD using the same formula as Shaham, et al. to determine whether batches looked like they came from the same distribution after adjustment. For the kernel, we used the Gaussian kernel between two batches as implemented in `sklearn.metrics.pairwise.rbf_kernel` [49]. In cases where there were more than two batches, we averaged all pairwise MMD values to calculate an overall MMD.

2.4.3 Classification accuracy

In order to determine (a) whether batch can still be identified post-adjustment and (b) how well class-related signal is maintained after adjustment, we determined classification accuracy based on batch and "true class" labels using several machine learning classifiers.

We used four classifiers from the scikit-learn 0.19.1 Python library [49] in order to classify on batch and true class before and after adjustment: Naive Bayes [50], Random Forests [51], k-Nearest Neighbors [52], and SVM [53] with a radial basis kernel. Table 1 details which columns were used for training.

We calculated the average of classification accuracies for four-fold cross-validation repeated three times. We interpret lower accuracy for batch classification as meaning that the batch is removed more effectively. We also interpret higher true class classification as meaning that the important signal is not lost during the process of adjustment. Therefore, given output data from the ideal batch adjuster, batch classification would be no better than random for any classification algorithm, and true class accuracy would be no lower than accuracy for the unadjusted data.

3 Results

In this study, we created Confounded, an adversarial variational autoencoder neural network, to remove nonlinear batch and confounding effects from expression data that may not be accounted for by traditional linear methods. We compared Confounded to a scaling method and to ComBat [11] using various metrics. The scaling method performed consistently worse across the qualitative and quantitative evaluations that we performed; ComBat and Confounded performed relatively well overall, but each of these algorithms excelled in different types of scenarios, which we illustrate below.

3.1 Confounded removes nonlinear confounding effects that other adjusters miss

To compare Confounded to other batch adjustment methods, we compared PCA and t-SNE plots along with MSE, MMD, and classifier batch prediction accuracy (using various classifiers).

PCA and t-SNE plots seem to show a decrease in separability after adjustment with various methods for the GSE37199 dataset (see Figures 4 and 5). However, previous research has shown that these plots are not completely trustworthy in representing nonlinear effects [9]. In the PCA plot, Confounded appears to maintain a similar distribution to the unadjusted data, indicating that the underlying distribution has been faithfully reproduced by the networks. The t-SNE plot shows that the data post-adjustment by Confounded and ComBat appear to cluster less tightly by batch than the unadjusted and scale-adjusted data. This may indicate an effective removal of nonlinear

effects in both cases.

Confounded shows mixed success with the MSE and MMD metrics. With MSE, Confounded outperformed the scale adjuster in 3 of the 4 datasets but scored drastically worse on the MNIST dataset, with scores listed in Table 2 (see also Figure 6). With MMD, Confounded outperformed the scale adjuster again in 3 of the 4 datasets and tied the scale adjuster on the TCGA dataset, with scores listed in Table 3 (see also Figure 7). With both metrics, Confounded consistently performed somewhat worse than ComBat.

We would expect that after batch adjustment by an ideal adjuster, batch would no longer be detectable by any machine learning classifier. Using the batch classification accuracy metric, Confounded seems to outperform other adjusters on larger datasets, whereas ComBat and Confounded seem to perform about the same on smaller datasets (see Figure 8). With both the bladderbatch and GSE37199 datasets, batch classification accuracy decreases well below baseline after batch adjustment with ComBat for all classifiers we tested (see Table 4). Interestingly, batch accuracy also decreases drastically for the MNIST and TCGA datasets, but only for the Naive Bayes classifier. This may be due to two factors: both ComBat and Naive Bayes use Bayesian methods, so ComBat may specifically remove the effects that Naive Bayes identifies; and Naive Bayes does not find patterns based on interactions between variables. Although Naive Bayes is no longer able to identify confounding effects in the data after ComBat-adjustment, Random Forests (which does use interactions between variables) still has a very high accuracy for MNIST and an increased accuracy for TCGA. In contrast, after adjustment by Confounded, the Random Forests algorithm's accuracy decreases more than with any other adjuster for both the MNIST and TCGA datasets. This indicates that while ComBat's performance may work at least as well as Confounded for smaller expression datasets, Confounded may work better with larger datasets.

With the larger datasets in particular, Confounded outperforms the other adjusters. On the MNIST dataset, Random Forests is able to detect batch with perfect or near-perfect accuracy after

adjustment with the scale adjuster and ComBat, but the highest batch classification accuracy after adjustment by Confounded is Naive Bayes, with an accuracy of 68.8%. With TCGA, both the scale adjuster and ComBat drastically increase Random Forests' batch classification accuracy from 87.6% to 96.3% and 97.1% respectively, whereas Confounded decreases the accuracy to 8.8%.

3.2 Class-related signal is still detectable after adjustment by Confounded

With the smaller datasets, Confounded seems to keep true class information roughly as well as ComBat, (with Random Forests, Bladderbatch: 74.3 for ComBat% and 72.1% for Confounded, GSE37199: 60.4% for ComBat and 69.0% for Confounded; see Figure 9 and Table 5). For the Bladderbatch dataset, true class accuracy is much lower after adjusting with any algorithm, indicating that cancer status and batch may not be independent.

With the larger datasets, Confounded's true class accuracy consistently decreases below the accuracy of other adjusters. A look at the MNIST digits before and after adjustment (see Figure 10) shows that Confounded's output is often blurry, as is common with the output of variational autoencoders [54]. With MNIST, Confounded's accuracy with Random Forests is still much higher than baseline (84.8% vs. 11.4%), but with TCGA, the accuracy decreases below baseline (66.5% vs. 69.8%) while the other adjusters' accuracies remain above baseline. However, the particular set of parameters that we used in Confounded are likely not optimal for every dataset. Additional tuning may improve the performance metrics.

4 Discussion

Why should I use a neural network for batch adjustment? The process of measuring data typically leaves confounding effects. This is particularly problematic in expression data, where each of the 20,000 transcript levels may influence or be influenced by other transcript levels. These cascading network-like effects are extremely likely to have nonlinear components. Our

results (i.e. tools like ComBat do not fool some classifiers on the batch classification task) indicate that these nonlinear effects do indeed exist in gene expression data, thus rendering linear batch correction methods insufficient. Therefore, some form of nonlinear adjustment must be used in order to correct for real-world confounding effects. Rather than individually model each of infinitely many possible nonlinear interactions to see which represent confounders, we can use a neural network such as Confounded to both approximate and remove the confounders, since neural networks are proven to be universal function approximators [55].

How can I tell how well batch adjustment worked? Although the metrics and figures that have been used in the past to validate batch adjustment (such as PCA, MSE, and MMD) represent how well linear effects have been removed, they cannot completely display whether two batches are distinguishable from one another. Machine learning algorithms are designed specifically to tease out patterns in data that may distinguish one group from another. Our results show that in some cases where PCA, MSE, and MMD indicate that ComBat removes confounding effects better than Confounded, the effects are still identifiable by machine learning algorithms after ComBat-adjustment, but not after adjustment by Confounded This indicates that classification accuracy measures the presence of confounding effects better than these traditional tools. We suggest to users of batch correction software that they use machine learning classification accuracy before and after correction in order to determine the degree of batch removal. We also suggest to researchers in the field of batch correction that classification accuracy be used as a metric in validating their software. Specifically, the Random Forests algorithm [51] seems to work very well and runs relatively quickly on gene expression data.

Which batch adjuster should I use? In our testing, ComBat did very well with small ($n < 100$) datasets, even with removing any identifiable nonlinear effects. However, Confounded outperformed ComBat on the larger datasets according to the batch classification accuracy metric. In addition to dataset size, researchers selecting a batch adjustment algorithm should consider how important it is for them to accurately replicate their input data. Such researchers

can adjust Confounded's λ parameter in order to balance the tradeoff of removing batch and matching the inputs.

What limitations does Confounded have? (a) Confounded uses a variational autoencoder, which are known for often outputting a blurry version of the input data (as can be seen in Figure 10). However, recent work has identified modifications that may be made to the basic VAE structure to make output images sharper and more realistic [54]. Similar research with variational autoencoders and gene expression data may yield improved reconstruction losses and decrease the blurring effect. (b) Confounded takes a long time to run in comparison with ComBat and other linear adjusters. Although we acknowledge this as a limitation of many types of machine learning and of neural network in particular, we believe that 30-60 minutes is a reasonable amount of time for a step that will be run only once per pipeline and that can greatly improve data quality. (c) It can be difficult to identify the optimal network structure and parameter set for a neural network. Though this is the case for many applications of neural networks, we feel that Confounded's default structure worked well in our testing and that it will suffice for most batch correction applications. (d) Neural networks usually perform better when given large amounts of data, and traditional batch datasets typically have very few samples. Because of this, we were concerned that a neural-network-based adjuster may not work well on traditional datasets. However, although ComBat outperformed Confounded on the smaller, more traditional datasets, Confounded did perform reasonably well with the smaller datasets and appeared to avoid overfitting. In cases where ComBat is unable to completely remove confounding effects in a small dataset, Confounded may be a viable replacement method.

What else might Confounded be used for? At its root, batch correction is a data integration problem: data from multiple batches must have batch-specific confounding effects removed in order to be treated as one dataset. Confounded shows promise in removing traditional batch effects from microarray expression data in the Bladderbatch and GSE37199 datasets. It also effectively decreased artificial batch effects in image data and cancer-type-specific confounding

effects in RNA-Seq data. Confounded may be effective in other data integration problems, such as combining microarray with RNA-Seq datasets, or merging several large datasets measured under different conditions.

Confounded, and adversarial autoencoders in general, show promise as a valuable way to remove confounding biases from expression datasets. Such methods will enable researchers access to larger datasets, therefore increasing the scope of analyses and furthering science as a whole.

References

[1] Freytag S, Gagnon-Bartsch J, Speed TP, Bahlo M. Systematic Noise Degrades Gene Co-Expression Signals but Can Be Corrected. BMC Bioinformatics. 2015 Sep;16(1):309.

[2] Stuart JM, Segal E, Koller D, Kim SK. A Gene-Coexpression Network for Global Discovery of Conserved Genetic Modules. Science. 2003 Oct;302(5643):249–255.

[3] Henriksen PA, Kotelevtsev Y. Application of Gene Expression Profiling to Cardiovascular Disease. Cardiovascular Research. 2002 Apr;54(1):16–24.

[4] van 't Veer LJ, Dai H, van de Vijver MJ, He YD, Hart AAM, Mao M, et al. Gene Expression Profiling Predicts Clinical Outcome of Breast Cancer. Nature. 2002 Jan;415(6871):530.

[5] Sirota M, Dudley JT, Kim J, Chiang AP, Morgan AA, Sweet-Cordero A, et al. Discovery and Preclinical Validation of Drug Indications Using Compendia of Public Gene Expression Data. Science Translational Medicine. 2011 Aug;3(96):96ra77–96ra77.

[6] Leek JT, Scharpf RB, Bravo HC, Simcha D, Langmead B, Johnson WE, et al. Tackling the Widespread and Critical Impact of Batch Effects in High-Throughput Data. Nature Reviews Genetics. 2010 Oct;11(10):733–739.

[7] Yue F, Cheng Y, Breschi A, Vierstra J, Wu W, Ryba T, et al. A Comparative Encyclopedia of DNA Elements in the Mouse Genome. Nature. 2014 Nov;515(7527):355–364.

[8] Gilad Y, Mizrahi-Man O. A Reanalysis of Mouse ENCODE Comparative Gene Expression Data. F1000Research. 2015 May;4.

[9] Dayton JB, Piccolo SR. Classifying Cancer Genome Aberrations by Their Mutually Exclusive Effects on Transcription. BMC Medical Genomics. 2017 Dec;10(Suppl 4).

[10] Lazar C, Meganck S, Taminau J, Steenhoff D, Coletta A, Molter C, et al. Batch Effect Removal Methods for Microarray Gene Expression Data Integration: A Survey. Briefings in Bioinformatics. 2013 Jul;14(4):469–490.

[11] Johnson WE, Li C, Rabinovic A. Adjusting Batch Effects in Microarray Expression Data Using Empirical Bayes Methods. Biostatistics (Oxford, England). 2007 Jan;8(1):118–127.

[12] Leek JT, Storey JD. Capturing Heterogeneity in Gene Expression Studies by Surrogate Variable Analysis. PLOS Genetics. 2007 Sep;3(9):e161.

[13] Liu H, Li P, Zhu M, Wang X, Lu J, Yu T. Nonlinear Network Reconstruction from Gene Expression Data Using Marginal Dependencies Measured by DCOL. PLoS ONE. 2016 Jul;11(7).

[14] Lu Z, Pu H, Wang F, Hu Z, Wang L. The Expressive Power of Neural Networks: A View from the Width. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, et al., editors. Advances in Neural Information Processing Systems 30. Curran Associates, Inc.; 2017. p. 6231–6239.

[15] Louizos C, Swersky K, Li Y, Welling M, Zemel R. The Variational Fair Autoencoder. arXiv:151100830 [cs, stat]. 2015 Nov;.

[16] Shaham U, Stanton KP, Zhao J, Li H, Raddassi K, Montgomery R, et al. Removal of Batch Effects Using Distribution-Matching Residual Networks. Bioinformatics (Oxford, England). 2017 Aug;33(16):2539–2546.

[17] Shaham U. Batch Effect Removal via Batch-Free Encoding. bioRxiv. 2018 Jul;.

[18] Upadhyay U, Jain A. Removal of Batch Effects Using Generative Adversarial Networks. arXiv:190106654 [cs, stat]. 2019 Jan;.

[19] Leek JT. Bladderbatch. Bioconductor; 2017.

[20] Leek JT, Johnson WE, Parker HS, Fertig EJ, Jaffe AE, Storey JD, et al.. Sva. Bioconductor; 2017.

[21] Schmidhuber J. Deep Learning in Neural Networks: An Overview. Neural Networks. 2015 Jan;61:85–117.

[22] Danaee P, Ghaeini R, Hendrix DA. A Deep Learning Approach for Cancer Detection and Relevant Gene Identification. In: Biocomputing 2017. WORLD SCIENTIFIC; 2016. p. 219–229.

[23] Chen Y, Li Y, Narayan R, Subramanian A, Xie X. Gene Expression Inference with Deep Learning. Bioinformatics. 2016 Jun;32(12):1832–1839.

[24] Way GP, Greene CS. Extracting a Biologically Relevant Latent Space from Cancer Transcriptomes with Variational Autoencoders. bioRxiv. 2017 Oct;p. 174474.

[25] Lin C, Jain S, Kim H, Bar-Joseph Z. Using Neural Networks for Reducing the Dimensions of Single-Cell RNA-Seq Data. Nucleic Acids Research. 2017 Sep;45(17):e156–e156.

[26] Aliper A, Plis S, Artemov A, Ulloa A, Mamoshina P, Zhavoronkov A. Deep Learning Applications for Predicting Pharmacological Properties of Drugs and Drug Repurposing Using Transcriptomic Data. Molecular Pharmaceutics. 2016 Jul;13(7):2524–2530.

[27] Beaulieu-Jones BK, Wu ZS, Williams C, Byrd JB, Greene CS. Privacy-Preserving Generative Deep Neural Networks Support Clinical Data Sharing. bioRxiv. 2017 Nov;p. 159756.

[28] Hinton GE, Salakhutdinov RR. Reducing the Dimensionality of Data with Neural Networks. Science. 2006 Jul;313(5786):504–507.

[29] Ganin Y, Ustinova E, Ajakan H, Germain P, Larochelle H, Laviolette F, et al. Domain-Adversarial Training of Neural Networks. arXiv:150507818 [cs, stat]. 2015 May;.

[30] Louizos C, Shalit U, Mooij J, Sontag D, Zemel R, Welling M. Causal Effect Inference with Deep Latent-Variable Models. arXiv:170508821 [cs, stat]. 2017 May;.

[31] Tzeng E, Hoffman J, Zhang N, Saenko K, Darrell T. Deep Domain Confusion: Maximizing for Domain Invariance. arXiv:14123474 [cs]. 2014 Dec;.

[32] Foundation PS. The Python Language Reference — Python 3.6.8 Documentation; 2019. https://docs.python.org/3.6/reference/index.html.

[33] Agarap AF. Deep Learning Using Rectified Linear Units (ReLU). arXiv:180308375 [cs, stat]. 2018 Mar;.

[34] Géron A. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 1st ed. O'Reilly Media, Inc.; 2017.

[35] Clevert DA, Unterthiner T, Hochreiter S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). arXiv:151107289 [cs]. 2015 Nov;.

[36] Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. arXiv:14126980 [cs]. 2014 Dec;.

[37] Kullback S, Leibler RA. On Information and Sufficiency. The Annals of Mathematical Statistics. 1951 Mar;22(1):79–86.

[38] Srivastava N, Hinton G, Krizhevsky A, Sutskever I, Salakhutdinov R. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research. 2014;15:1929–1958.

[39] Ioffe S, Szegedy C. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv:150203167 [cs]. 2015 Feb;.

[40] Santurkar S, Tsipras D, Ilyas A, Madry A. How Does Batch Normalization Help Optimization? arXiv:180511604 [cs, stat]. 2018 May;.

[41] LECUN Y. THE MNIST DATABASE of Handwritten Digits. http://yannlecuncom/exdb/mnist/;.

[42] Dyrskjøt L, Kruhøffer M, Thykjaer T, Marcussen N, Jensen JL, Møller K, et al. Gene Expression in the Urinary Bladder: A Common Carcinoma in Situ Gene Expression Signature Exists Disregarding Histopathological Classification. Cancer Research. 2004 Jun;64(11):4040–4048.

[43] Olmos D, Brewer D, Clark J, Danila DC, Parker C, Attard G, et al. Prognostic Value of Blood mRNA Expression Signatures in Castration-Resistant Prostate Cancer: A Prospective, Two-Stage Study. The Lancet Oncology. 2012 Nov;13(11):1114–1124.

[44] Golightly NP, Bell A, Bischoff AI, Hollingsworth PD, Piccolo SR. Curated Compendium of Human Transcriptional Biomarker Data. Scientific Data. 2018 Apr;5:180066.

[45] The Cancer Genome Atlas Research Network, Weinstein JN, Collisson EA, Mills GB, Shaw KRM, Ozenberger BA, et al. The Cancer Genome Atlas Pan-Cancer Analysis Project. Nature Genetics. 2013 Sep;45:1113–1120.

[46] R Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria: R Foundation for Statistical Computing; 2014.

[47] RStudio Team. RStudio: Integrated Development Environment for R. Boston, MA: RStudio, Inc.; 2018.

[48] Espín-Pérez A, Portier C, Chadeau-Hyam M, van Veldhoven K, Kleinjans JCS, de Kok TMCM. Comparison of Statistical Methods and the Use of Quality Control Samples for Batch Effect Correction in Human Transcriptome Data. PLOS ONE. 2018 Aug;13(8):e0202947.

[49] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-Learn:

Machine Learning in Python. Journal of Machine Learning Research. 2011;12(Oct):2825–2830.

[50] Maron ME. Automatic Indexing: An Experimental Inquiry. J ACM. 1961 Jul;8(3):404–417.

[51] Tin Kam Ho. Random Decision Forests. In: Proceedings of 3rd International Conference on Document Analysis and Recognition. vol. 1; 1995. p. 278–282 vol.1.

[52] Fix E, Hodges JL Jr. Discriminatory Analysis, Nonparametric Discrimination: Consistency Properties. Randolph Field, Texas: USAF School of Aviation Medicine; 1951. 4.

[53] Cortes C, Vapnik V. Support-Vector Networks. Machine Learning. 1995 Sep;20(3):273–297.

[54] Hou X, Shen L, Sun K, Qiu G. Deep Feature Consistent Variational Autoencoder. arXiv:161000291 [cs]. 2016 Oct;.

[55] Csáji BC, Eikelder HT. Approximation with Artificial Neural Networks; 2001.

Tables

Table 1: **Dataset information** for each dataset used.

Dataset	Dimensions	Number of Batches	Batch Label	True Class Label	Data Type
Bladder Batch	$57 \times 22,283$	5	Batch	Cancer status	Microarray
GSE37199	$93 \times 20,024$	2	Plate	Cancer stage	Microarray
MNIST	$10,000 \times 784$	2	Artificial batch	Digit	Grayscale images
TCGA Pan-Cancer	$9,366 \times 325$	25	Cancer Type	TP53 mutation presence	RNA-Seq

Table 2: **Mean squared error (MSE)** of the unadjusted input data compared to the data output by the given adjusters. Lower MSE indicates that the output has changed less from the input.

Dataset	Unadjusted	Scale	ComBat	Confounded
Bladder Batch	0	0.247	0.0424	0.0698
GSE37199	0	0.003	0.00066	0.00168
MNIST	0	0.00312	0.00183	0.0187
TCGA	0	9.12e+05	1.17e+05	1.39e+05

Table 3: **Maximum mean discrepancy (MMD)** comparing the distributions of the batches to each other after a given adjustment. Lower MMD indicates that the distributions of the different batches are more similar. In cases with more than two batches, MMD is computed pairwise between each batch and averaged.

Dataset	Unadjusted	Scale	ComBat	Confounded
Bladder Batch	0.48	0.463	0.183	0.258
GSE37199	0.0941	0.0906	0.0255	0.0376
MNIST	0.113	0.0653	0.00665	0.0117
TCGA	0.0942	0.0942	0.0942	0.0942

Table 4: **Batch classification accuracy** for several datasets and adjusters. The ideal batch adjuster would completely remove all signal due to batch and would therefore *decrease* batch classification accuracy to around the baseline for all classifiers.

Dataset	Adjustment	Baseline	GaussianNB	KNeighbors	RandomForest	SVC
Bladder Batch	Unadjusted	0.333	0.626	0.764	0.661	0.578
	Scale		0.315	0.492	0.514	0.472
	ComBat		0.000	0.158	0.183	0.159
	Confounded		0.180	0.067	0.224	0.275
GSE37199	Unadjusted	0.538	0.803	0.705	0.873	0.535
	Scale		0.930	0.830	1.000	0.535
	ComBat		0.238	0.534	0.494	0.535
	Confounded		0.409	0.408	0.409	0.535
MNIST	Unadjusted	0.500	1.000	0.899	1.000	1.000
	Scale		0.992	0.760	1.000	1.000
	ComBat		0.466	0.499	0.999	0.528
	Confounded		0.688	0.519	0.637	0.561
TCGA	Unadjusted	0.117	0.832	0.758	0.876	0.117
	Scale		0.846	0.779	0.963	0.117
	ComBat		0.293	0.358	0.971	0.117
	Confounded		0.078	0.086	0.088	0.117

Table 5: **True class classification accuracy** for several datasets and adjusters. After adjustment by the ideal batch adjuster, all true class signal should be preserved, and all classifiers should therefore have the same accuracy in predicting true class before and after adjustment.

Dataset	Adjustment	Baseline	GaussianNB	KNeighbors	RandomForest	SVC
Bladder Batch	Unadjusted	0.702	0.908	0.905	0.884	0.906
	Scale		0.649	0.719	0.695	0.743
	ComBat		0.697	0.578	0.743	0.722
	Confounded		0.673	0.676	0.721	0.698
GSE37199	Unadjusted	0.667	0.661	0.689	0.690	0.662
	Scale		0.632	0.690	0.704	0.662
	ComBat		0.647	0.676	0.604	0.662
	Confounded		0.646	0.661	0.690	0.662
MNIST	Unadjusted	0.114	0.824	0.939	0.880	0.913
	Scale		0.816	0.946	0.874	0.915
	ComBat		0.815	0.948	0.876	0.914
	Confounded		0.794	0.891	0.848	0.853
TCGA	Unadjusted	0.698	0.625	0.738	0.768	0.698
	Scale		0.603	0.723	0.760	0.698
	ComBat		0.659	0.695	0.744	0.698
	Confounded		0.461	0.639	0.665	0.698

Figures

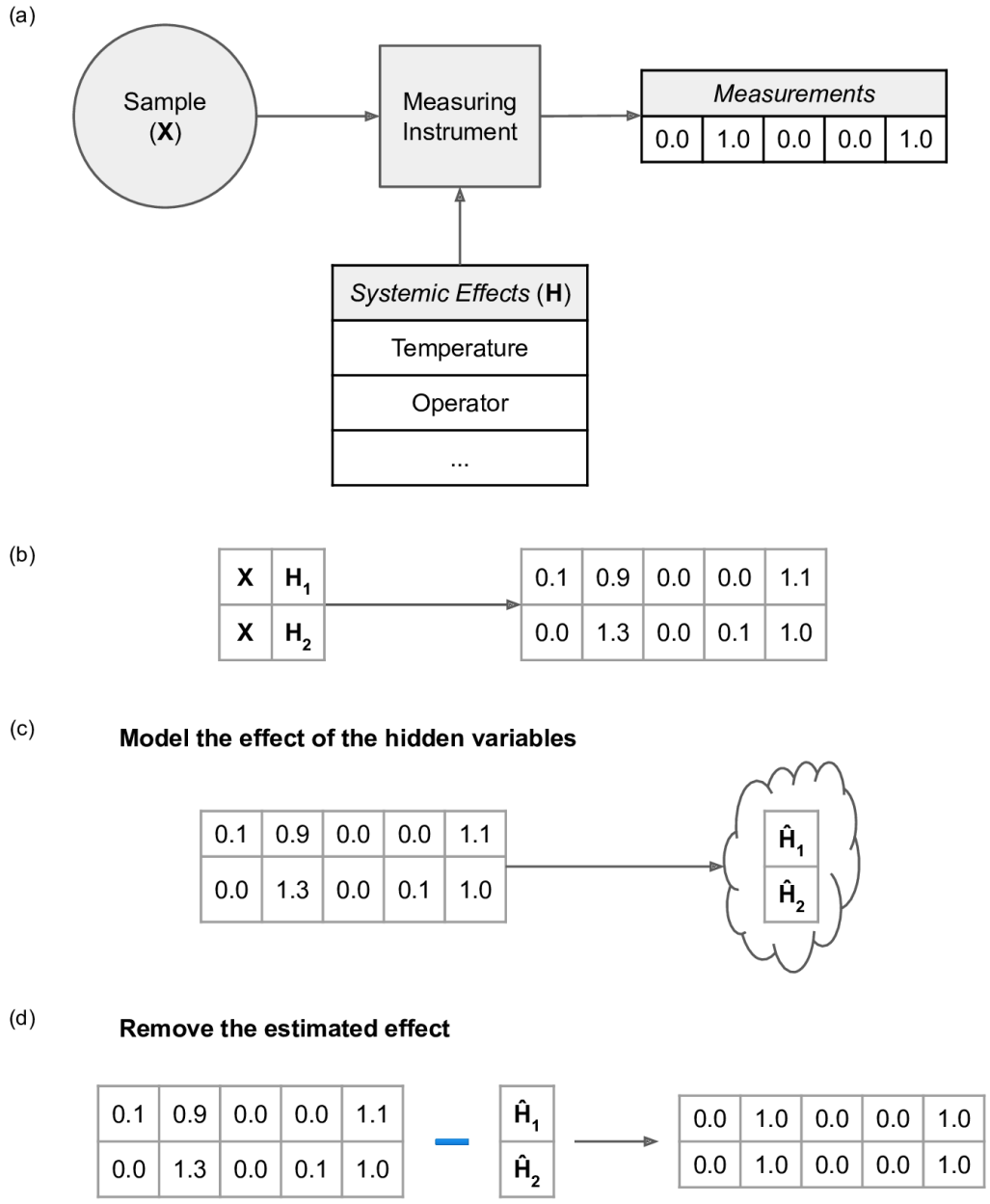

Figure 1: **Batch adjustment justification and steps.** (a) When measurements are collected from a sample (X), systemic effects (H) also affect the measurements. (b) If data from the same sample X is measured under two different conditions, H_1 and H_2, we may obtain slightly different measurements. (c) In order to normalize batches of data relative to one another, we first estimate the effect of the hidden variables based on differences in measurements between batches. (d) Second, we remove the estimated effects in order to normalize the batches relative to one another.

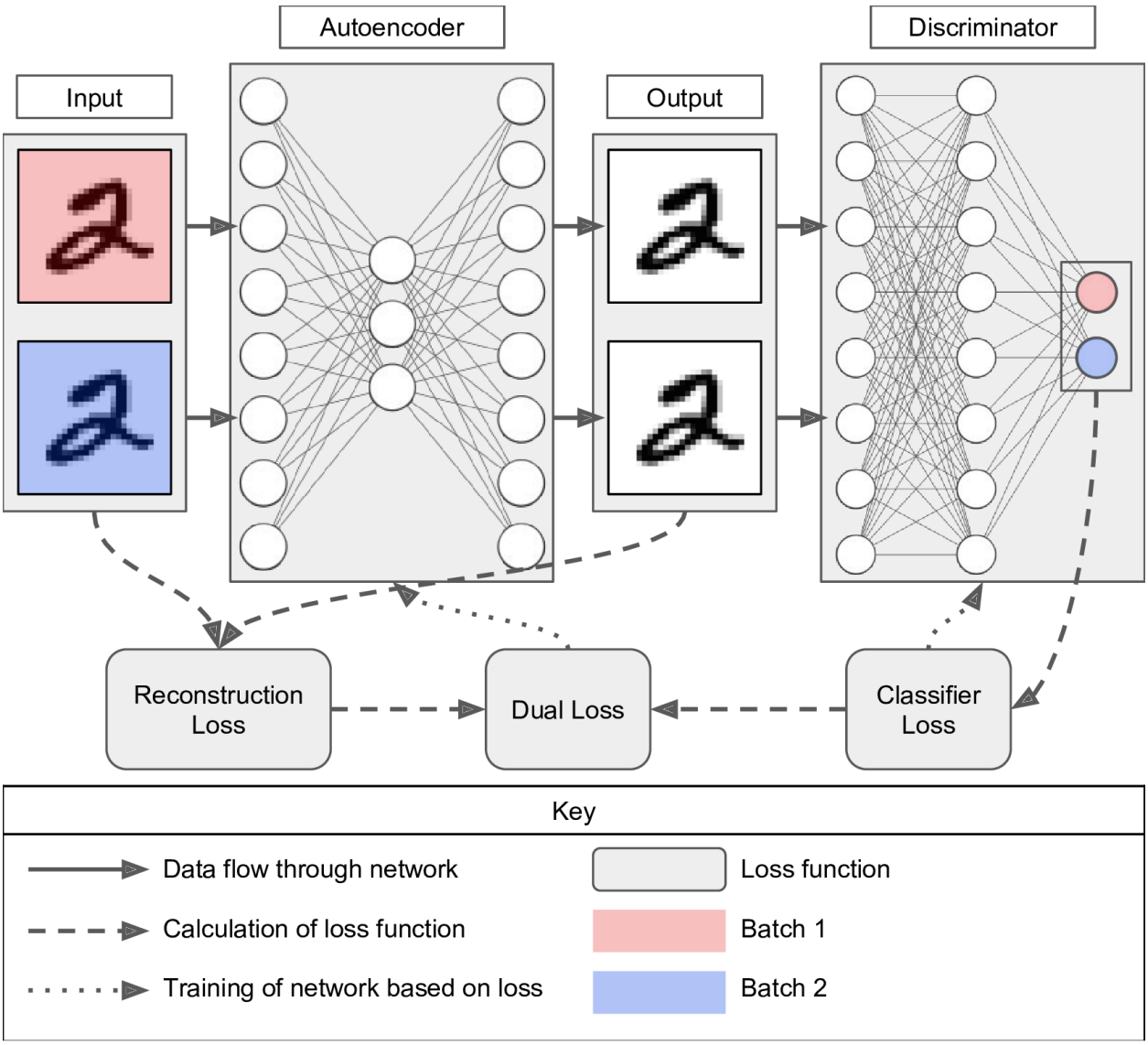

Figure 2: **Network architecture of Confounded.** Data with batch effects (represented by different colors) are input into an autoencoder. The output of the autoencoder is classified by a discriminator network based on batch. The autoencoder is then penalized based on the success of the discriminator. Over time, the autoencoder learns to output a faithful representation of the data without signal due to batch.

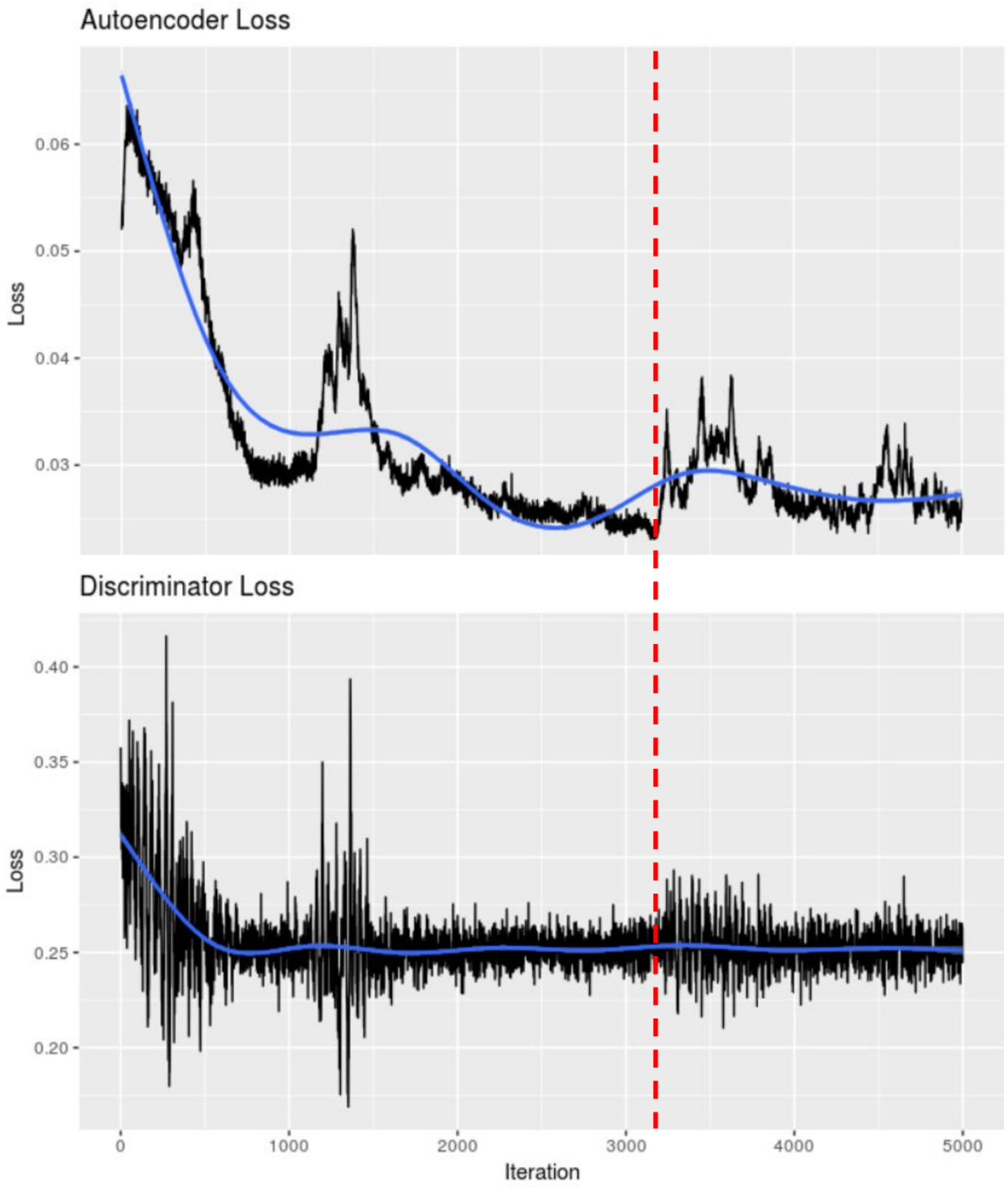

Figure 3: **Autoencoder and discriminator loss** over time for one run of Confounded on the MNIST dataset. Over the course of training, the autoencoder more faithfully replicates the input data. The autoencoder also seems to introduce noise (see the red dashed line around iteration 3100) in response to the discriminator's slight improvements.

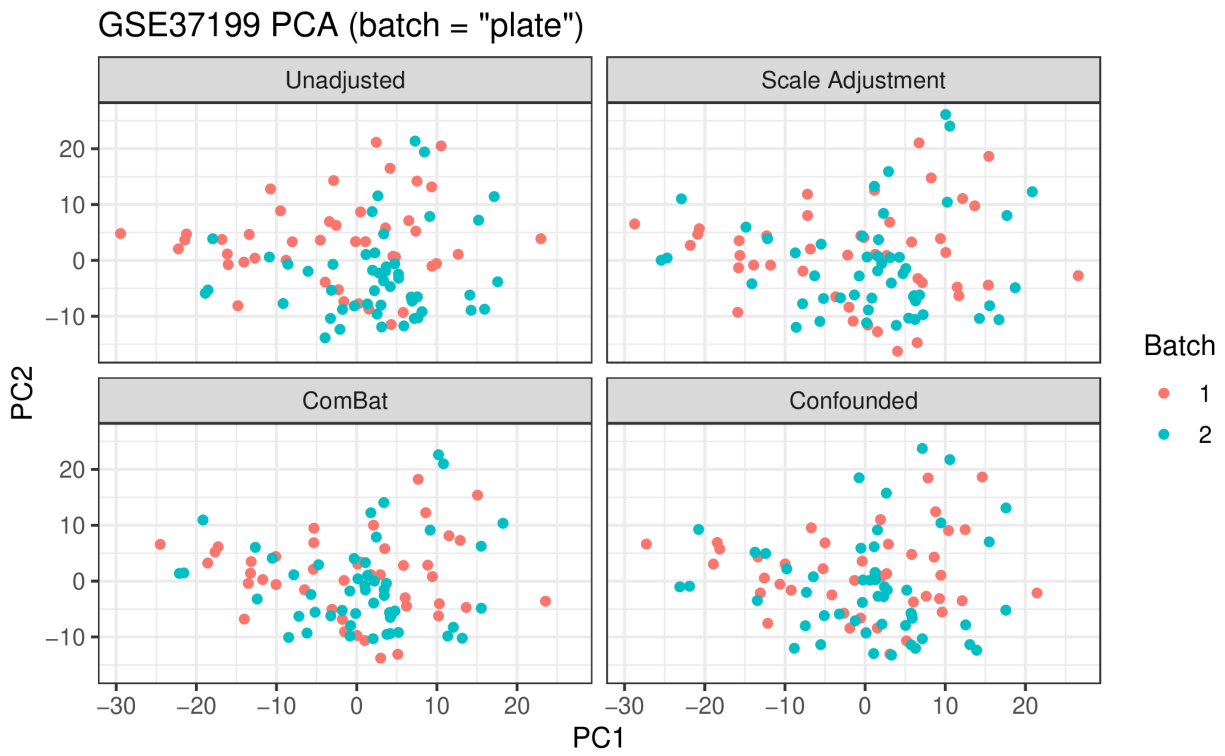

Figure 4: **Principal components analysis (PCA)** of the GSE37199 dataset before and after batch adjustment with various adjusters. None of the datasets appear to be linearly separable. Confounded appears to maintain the same distribution of data overall as the unadjusted data while perhaps aligning the batches' distributions.

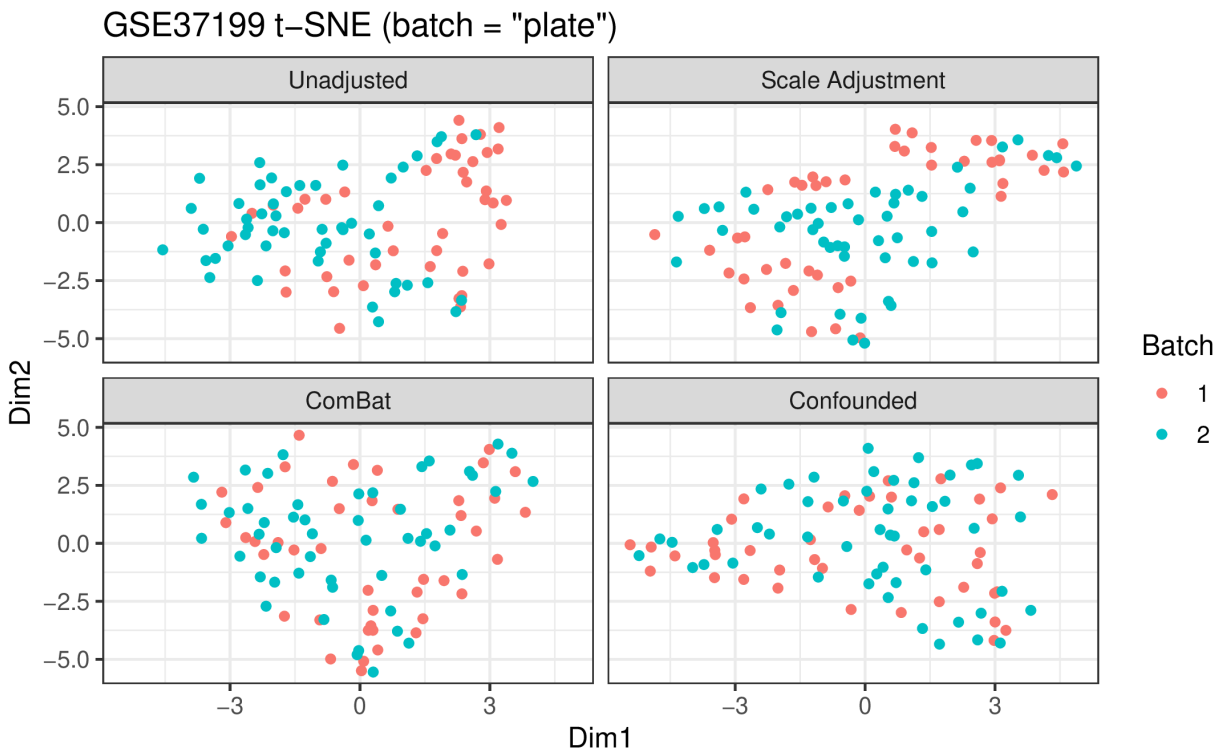

Figure 5: **T-distributed Stochastic Neighbor Embedding (t-SNE)** plot for the GSE37199 dataset before and after adjustment with several algorithms. The data seem to cluster less by batch for both Confounded and ComBat, indicating that both adjusters may be removing nonlinear effects in this dataset.

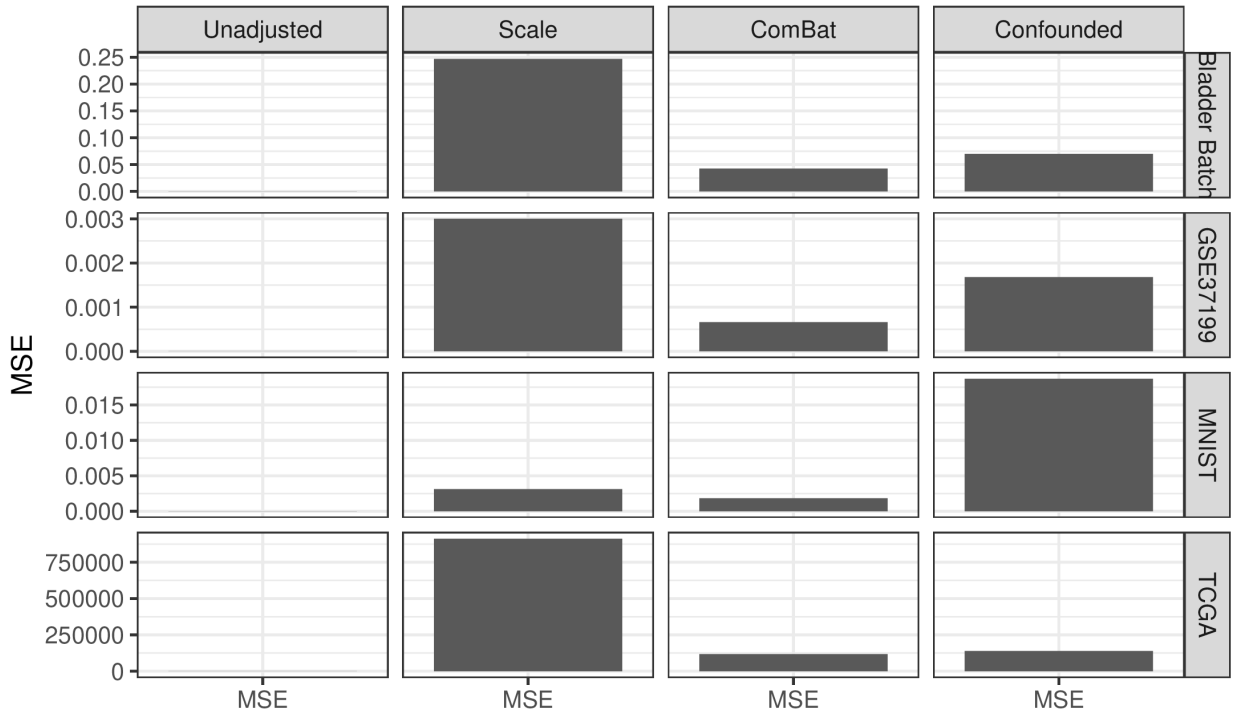

Figure 6: **Mean squared error (MSE)** between the data prior to and after adjustment with various algorithms. Lower MSE represents that the adjuster has more faithfully reproduced the input data. MSE for unadjusted data will always be 0 because the input data is identical to the output data. Confounded usually performs better than the scale adjuster and somewhat worse than ComBat when measuring MSE.

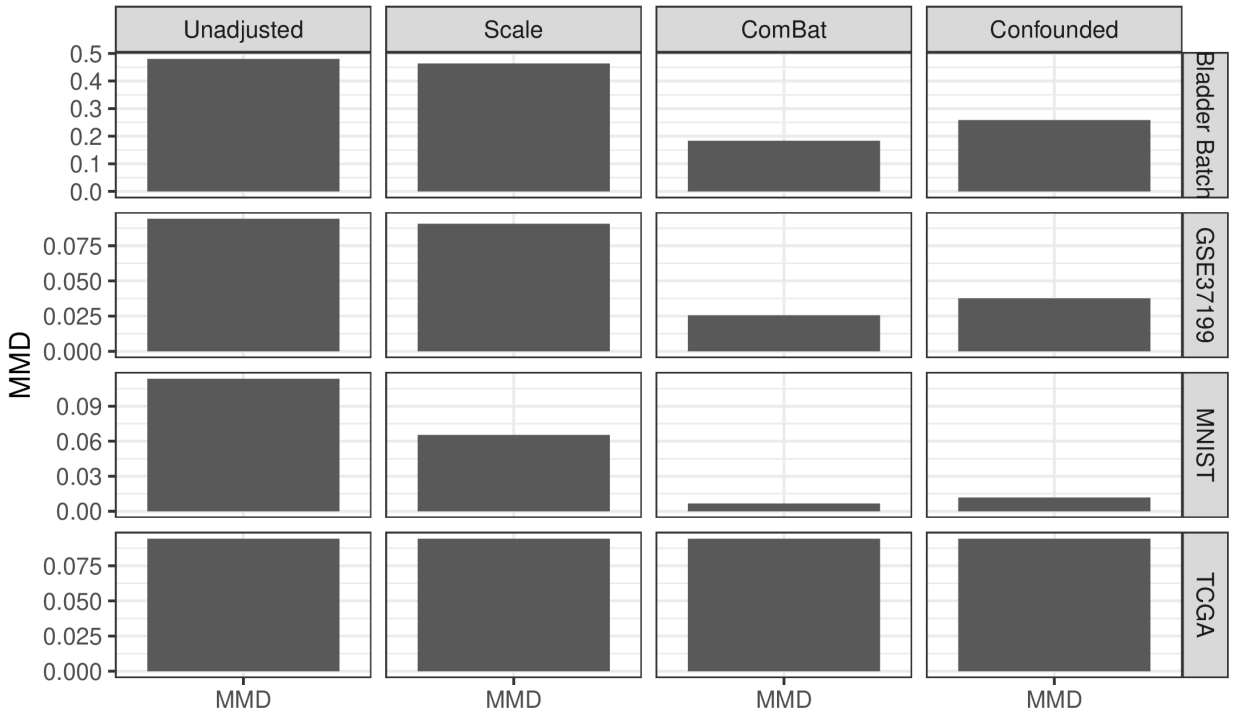

Figure 7: **Maximum mean discrepancy (MMD)** between different batches. Lower MMD indicates that the distributions of the different batches are more similar. In cases with more than two batches, MMD is computed pairwise between each batch and averaged. In each case, Confounded usually performs better than the scale adjuster and somewhat worse than ComBat when measuring MSE.

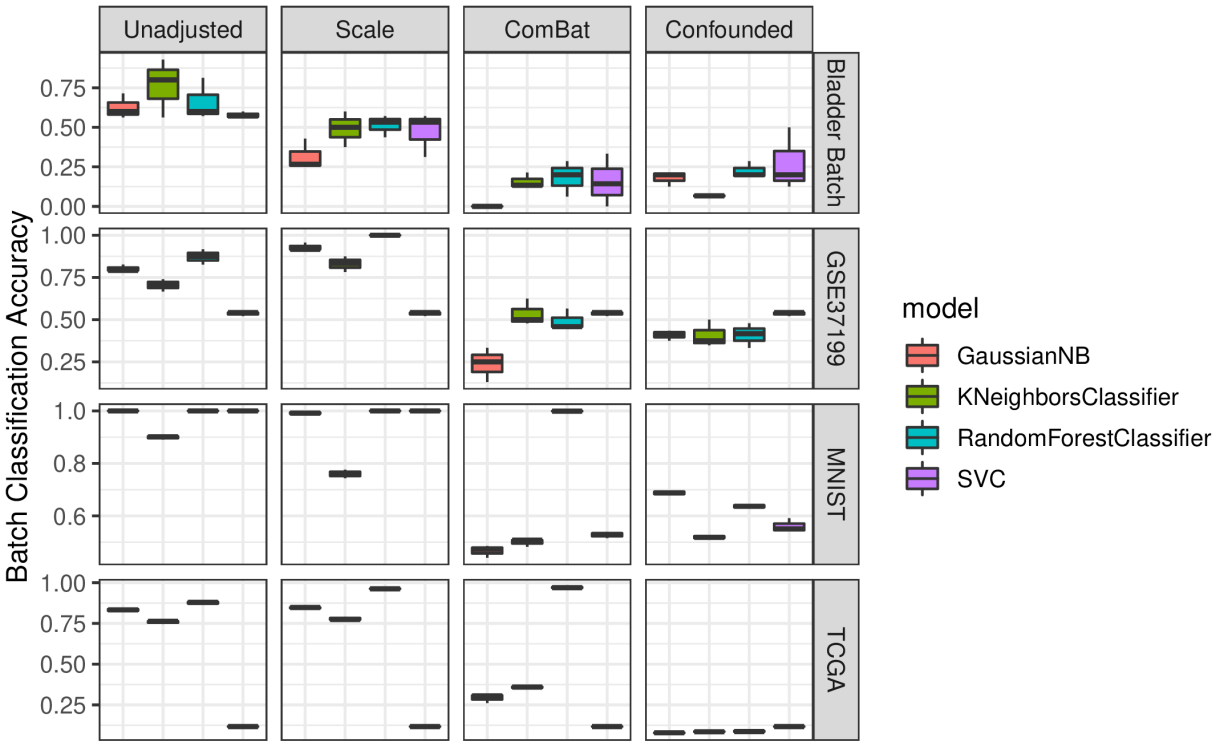

Figure 8: **Batch classification accuracy** from 4-fold cross-validation repeated 3 times for several classifiers. Lower batch accuracy indicates that more batch-related signal has been removed and therefore indicates better performance. Confounded's performance is similar to ComBat's for the smaller datasets and is improved for the larger datasets.

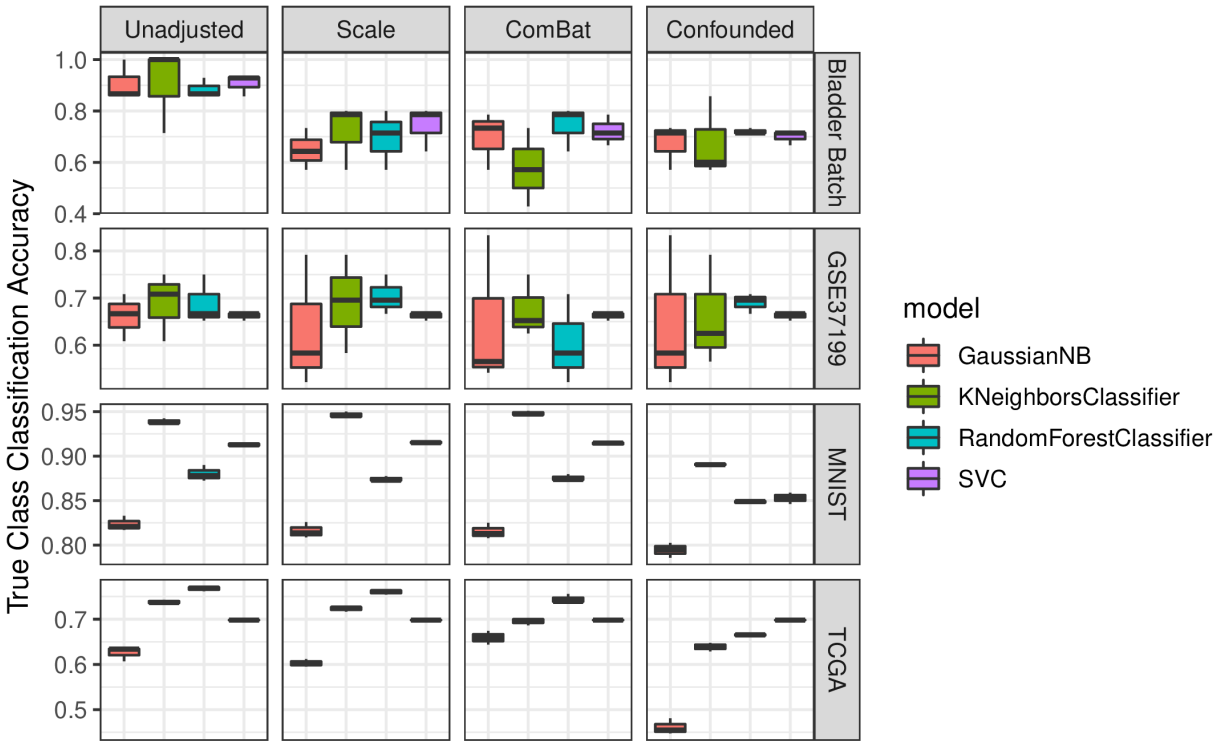

Figure 9: **True class classification accuracy** for several datasets and adjusters with 4-fold cross-validation repeated 3 times. A higher accuracy after adjustment is desired because it represents that the adjuster has not destroyed the true class signal.

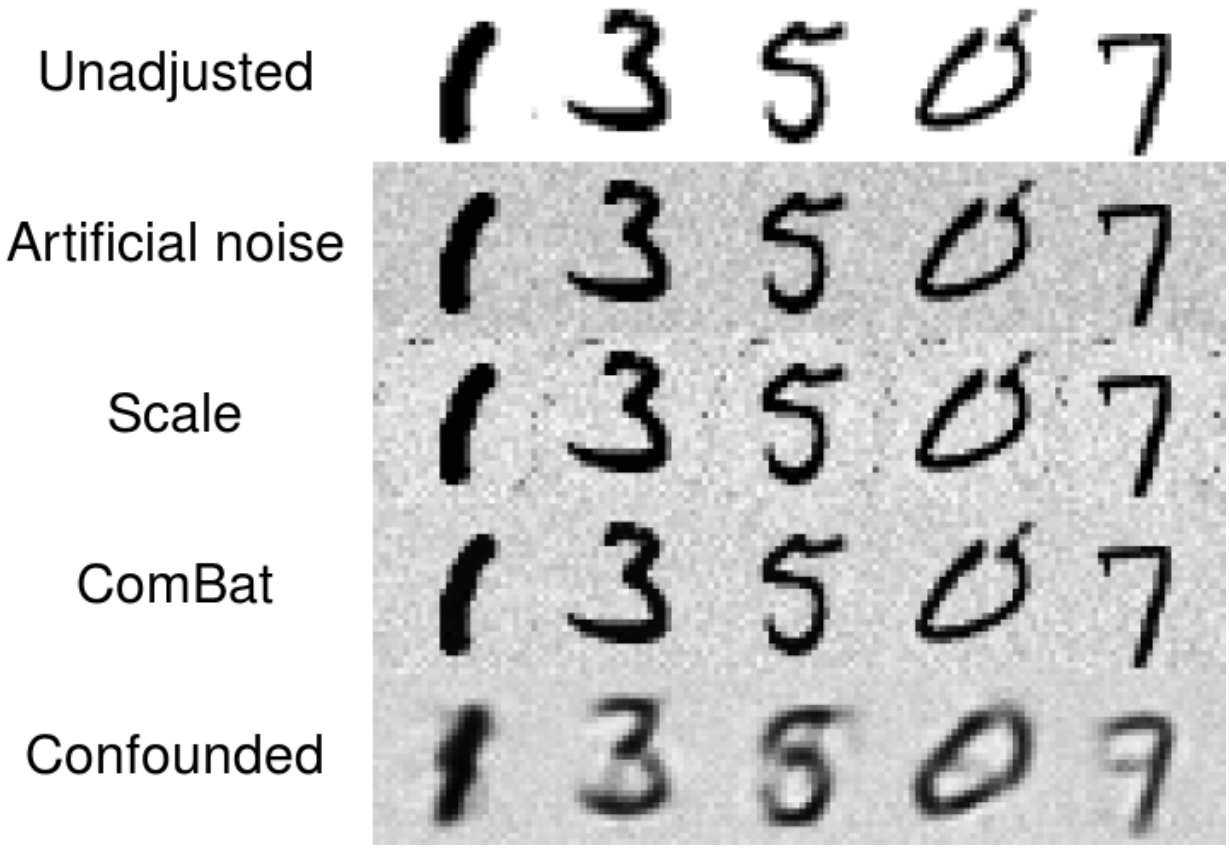

Figure 10: **MNIST handwritten digits** (a) before any adjustment, (b) with artificial noise added, (c) adjusted for noise by the scale adjuster, (d) adjusted for noise by ComBat, and (e) adjusted for noise by Confounded. Although Confounded seems to remove more noise from the background, it struggles in some cases to accurately replicate the input data.