BEYOND GANS: TRANSFORMING WITHOUT A TARGET DISTRIBUTION

Anonymous authors
Paper under double-blind review

ABSTRACT

While generative neural networks can learn to transform a specific input dataset into a specific target dataset, they require having just such a paired set of input/output datasets. For instance, to fool the discriminator, a generative adversarial network (GAN) exclusively trained to transform images of black-haired *men* to blond-haired *men* would need to change gender-related characteristics as well as hair color when given images of black-haired *women* as input. This is problematic, as often it is possible to obtain *a* pair of (source, target) distributions but then have a second source distribution where the target distribution is unknown. The computational challenge is that generative models are good at generation within the manifold of the data that they are trained on. However, generating new samples outside of the manifold or extrapolating "out-of-sample" is a much harder problem that has been less well studied. To address this, we introduce a technique called *neuron editing* that learns how neurons encode an edit for a particular transformation in a latent space. We use an autoencoder to decompose the variation within the dataset into activations of different neurons and generate transformed data by defining an editing transformation on those neurons. By performing the transformation in a latent trained space, we encode fairly complex and non-linear transformations to the data with much simpler distribution shifts to the neuron's activations. Our technique is general and works on a wide variety of data domains and applications. We first demonstrate it on image transformations and then move to our two main biological applications: removal of batch artifacts representing unwanted noise and modeling the effect of drug treatments to predict synergy between drugs.

1 INTRODUCTION

Many experiments in biology are conducted to study the effect of a treatment or a condition on a set of samples. For example, the samples can be groups of cells and the treatment can be the administration of a drug. However, experiments and clinical trials are often performed on only a small subset of samples from the entire population. Usually, it is assumed that the effects generalize to all of the samples in a context-independent manner, potentially conflating sample-specific variation with treatment-induced variation. Mathematically modeling both sources of variation and isolating the treatment-induced parts provides an opportunity to improve generalization beyond the samples measured.

We propose a neural network-based method for learning a general edit function corresponding to treatment in the biological setting. Popular neural network architectures like GANs pose the problem as one of learning to *output* data in the region of the space occupied by the target distribution, no matter where the input data is coming from. To fool the discriminator, the generator's output must end up in the same part of the space as the target distribution. The discriminator does not take into account the input points into the generator in any way.

Instead, we reframe the problem as learning a transformation *towards* the target distribution that is more sensitive to where the input data starts. Thus, we could learn an edit between pre- and post-treatment samples on one patient and apply it to pre-treatment samples of another patient without also modeling patient-specific characteristics that were present both pre- and post-treatment.

We propose to learn such an edit, which we term *neuron editing*, in the latent space of an autoencoder neural network with non-linear activations. First we train an autoencoder on the entire population of data which we are interested in transforming. This includes all of the pre-treatment samples and the post-treatment samples from the subset of the data on which we have post-treatment measurements. Neuron editing then involves extracting differences between the observed pre-and post-treatment activation distributions for neurons in this layer and then applying them to pre-treatment data from the rest of the population to synthetically generate post-treatment data. Thus performing the edit node-by-node in this space actually encodes complex multivariate edits in the ambient space, performed on denoised and meaningful features, owing to the fact that these features themselves are complex non-linear combinations of the input features.

Neuron editing is a general technique that could be applied to the latent space of any neural network, even GANs themselves. We focus exclusively on the autoencoder in this work, however, to leverage its denoising ability, robustness to mode dropping, and superior training stability as compared to GANs. We demonstrate that neuron editing can work on a variety of architectures, while offering the advantages of introducing no new hyperparameters to tune and being stable across multiple runs.

While latent space manipulation has been explored in previous work, ours differs in several ways. For example, Radford et al. (2015) represents a transformation between two distributions as a single constant shift in latent space. In addition to assuming the latent transformation is the same for all points in the distribution, Upchurch et al. (2017) also uses an off-the-shelf pre-trained Imagenet classifier network. Our work, on the other hand, does not require a richly supervised pre-trained model; also, we model the shift between two distributions as a complex, non-constant function that learns different shifts for different parts of the space. We compare to this "constant-shift" approach and demonstrate empirically why it is necessary to model the transformation more complexly.

By performing the edit to the neural network internal layer, we allow for the modeling of some context dependence. Some neurons are less heavily edited but still influence the output jointly with edited neurons due to their integration in the decoding layers, propagating their effect into the output space.

We note that neuron editing makes the assumption that the internal neurons have semantic consistency across the data, i.e., the same neurons encode the same types of features for every data manifold. We demonstrate that this holds in our setting because we choose to let the autoencoder learn a joint manifold of all of the given data, including pre- and post-treatment samples of the experimental subpopulation and pre-treatment samples from the rest of the population. Recent results show that neural networks prefer to learn patterns over memorizing inputs even when they have the capacity to do so (Zhang et al., 2016).

We demonstrate that neuron editing extrapolates better than generative models on two important criteria. First, as to the original goal, the predicted change on extrapolated data more closely resembles the predicted change on interpolated data. Second, the editing process produces more complex variation, since it simply preserves the existing variation in the data rather than needing a generator to learn to create it. We compare to standard GAN approaches, dedicated parametric statistical methods used by computational biologists, and alternative autoencoder frameworks. In each case, we see that they stumble on one or more of several hurdles: out-of-sample input, desired output that differs from the target of the training data, and data with complex variation.

In the following section, we detail the neuron editing method. Then, we motivate the extrapolation problem by trying to perform natural image domain transfer on the canonical CelebA dataset (Liu et al., 2018). We then move to two biological applications where extrapolation is essential: correcting the artificial variability introduced by measuring instruments (batch effects), and predicting the combined effects of multiple drug treatments (combinatorial drug effects) (Anchang et al., 2018).

2 MODEL

Let $S, T, X \subseteq \mathbb{R}^d$ be sampled sets representing d-dimensional source, target, and extrapolation distributions, respectively. We seek a transformation such that: 1. when applied to S it produces a distribution equivalent to T 2. when applied to T it is the identity function and 3. when applied to X it does not necessarily produce T if S is different from X. While GANs learn a transformation with the first two properties, they fail at the third property due to the fact that T is the only target data

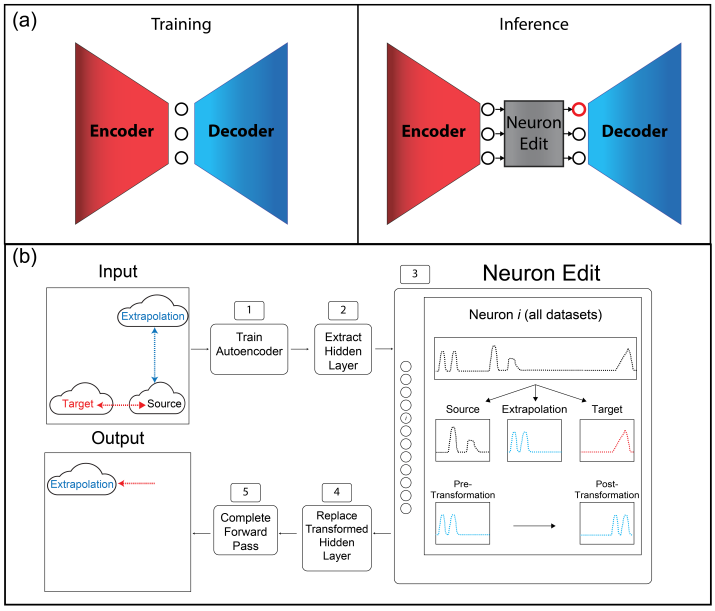

Figure 1: (a) Neuron editing interrupts the standard feedforward process, editing the neurons of a trained encoder/decoder to include the source-to-target variation, and letting the trained decoder cascade the resulting transformation back into the original data space. (b) The neuron editing process. The transformation is learned on the distribution of neuron activations for the source and applied to the distribution of neuron activations for the extrapolation data.

we have for training, and thus the generator only learns to output data like T. Therefore, instead of learning such a transformation parameterized by a neural network, we learn a simpler transformation on a *space learned by a neural network* (summarized in Figure 1).

We first train an encoder/decoder pair E/D to map the data into an abstract neuron space decomposed into high-level features such that it can also decode from that space, i.e., the standard autoencoder objective L:

$$L(S,T,X) = \text{MSE}\left[(S,T,X), D(E(S,T,X))\right]$$

where MSE is the mean-squared error. The autoencoder is trained on all three data distributions S, T, and X and thus learns to model their joint manifold. Then, without further training, we separately extract the activations of an n-dimensional internal layer of the network for inputs from S and from T, denoted by $a_S : S \to \mathbb{R}^n, a_T : T \to \mathbb{R}^n$. We define a piecewise linear transformation, called $NeuronEdit$, which we apply to these distributions of activations:

$$NeuronEdit(a) = \begin{cases} \left(\frac{a-p_0^S}{p_1^S-p_0^S} \cdot (p_1^T - p_0^T)\right) + p_0^T & a < p_1^S \\ \left(\frac{a-p_j^S}{p_{j+1}^S-p_j^S} \cdot (p_{j+1}^T - p_j^T)\right) + p_j^T & a \in [p_j^S, p_{j+1}^S) \\ \left(\frac{a-p_{99}^S}{p_{100}^S-p_{99}^S} \cdot (p_{100}^T - p_{99}^T)\right) + p_{99}^T & a \geq p_{99}^S \end{cases} \tag{1}$$

where $a \in \mathbb{R}^n$ consists of n activations for a single network input, $p_j^S, p_j^T \in \mathbb{R}^n$ consist of the j^{th} percentiles of activations (i.e., for each of the n neurons) over the distributions of a_S, a_T correspondingly, and all operations are taken pointwise, i.e., independently on each of the n neurons in the layer. Then, we define $NeuronEdit(a_S) : S \to \mathbb{R}^n$ given by $x \mapsto NeuronEdit(a_S(x))$, and equivalently for a_T and any other distribution (or collection) of activations over a set of network inputs. Therefore, the $NeuronEdit$ function operates on distributions, represented via activations over network input samples, and transforms the input activation distribution based on the difference between the source and target distributions (considered via their percentile disctretization).

We note that the $NeuronEdit$ function has the three properties we stated above: 1. $NeuronEdit(a_S) \approx a_T$ (in terms of the represented n-dimensional distributions) 2. $NeuronEdit(a_T) = a_T$ 3. $NeuronEdit(a_X) = NeuronEdit(a_S) \implies a_X = a_S$. This last property is crucial since learning to generate distributions like T, with a GAN for example, would produce a discriminator who encourages the output to be funneled as close to T as posssible no matter where in the support we start from.

To apply the learned transformation to X, we first extract the activations of the internal layer computed by the encoder, a_X. Then, we cascade the transformations applied to the neuron activations through the decoder without any further training. Thus, the transformed output $\hat{X}$ is obtained by:

$$\hat{X} = D(NeuronEdit(E(X)))$$

We emphasize that at this point, since we do no further training of the encoder and decoder, and since the neuron editing transformation has no weights to learn, there is no further objective term to minimize at this point and the transformation is fully defined.

Crucially, the nomenclature of an *autoencoder* no longer strictly applies. If we allowed the encoder or decoder to train with the transformed neuron activations, the network could learn to undo these transformations and still produce the identity function. However, since we freeze training and apply these transformations exclusively on inference, we turn an autoencoder into a generative model that need not be close to the identity.

Training a GAN in this setting could exclusively utilize the data in S and T, since we have no real examples of the output for X to feed to the discriminator. Neuron editing, on the other hand, is able to model the variation intrinsic to X in an unsupervised manner despite not having real post-transformation data for X. Since we know *a priori* that X will differ substantially from S, this provides significantly more information.

Furthermore, GANs are notoriously tricky to train (Salimans et al., 2016; Gulrajani et al., 2017; Wei et al., 2018). Adversarial discriminators suffer from oscillating optimization dynamics (Li et al., 2017), uninterpretable losses (Barratt & Sharma, 2018; Arjovsky et al., 2017), and most debilitatingly, mode collapse (Srivastava et al., 2017; Kim et al., 2017; Nagarajan & Kolter, 2017). Mode collapse refers to the discriminator being unable to detect differences in variabillity between real and fake examples. In other words, the generator learns to generate a point that is very realistic, but produces that same point for most (or even all) input, no matter how different the input is.

Neuron editing avoids all of these traps by learning an unsupervised model of the data space with the easier-to-train autoencoder. The essential step that facilitates generation is the isolation of the variation in the neuron activations that characterizes the difference between source and target distributions.

There is a relationship between neuron editing and the well-known word2vec embeddings in natural language processing (Goldberg & Levy, 2014). There, words are embedded in a latent space where a meaningful transformation such as changing the gender of a word is a constant vector in this space. This vector can be learned on one example, like transforming *man* to *woman*, and then extrapolated to another example, like *king*, to predict the location in the space of *queen*. Neuron editing is an extension in complexity of word2vec's vector arithmetic, because instead of transforming a single point into another single point, it transforms an entire distribution into another distribution.

3 EXPERIMENTS

We compare the predictions from neuron editing to those of several generation-based approaches: a traditional GAN, a GAN implemented with residual blocks (ResnetGAN) to show generating residuals is not the same as editing (Szegedy et al., 2017), and a CycleGAN (Zhu et al., 2017). While in other applications, like natural images, GANs have shown an impressive ability to generate plausible individual points, we illustrate that they struggle with these two criteria. We also motivate why neuron editing is performed on inference by comparing against a regularized autoencoder that performs the internal layer transformations during training, but the decoder learns to undo the transformation and reconstruct the input unchanged (Amodio et al., 2018). Lastly, we motivate why the more complex neuron editing transformation is necessary by comparing against a naive "latent vector arithmetic" approach. We find the constant vector between the mean of the source and the mean

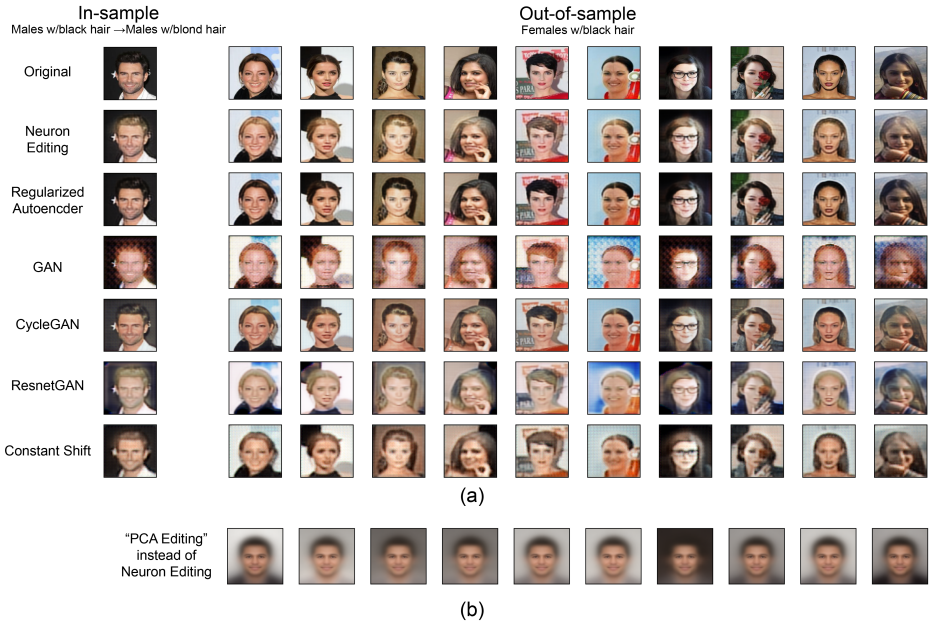

Figure 2: Data from CelebA where the source data consists of males with black hair and the target data consists of males with blond hair. The extrapolation is then applied to females with black hair. (a) A comparison of neuron editing against other models. Only neuron editing successfully applies the blond hair transformation. (b) An illustration that neuron editing must be applied to the neurons of a deep network, as opposed to principle components.

CelebA	Neuron Editing	GAN	CycleGAN	ResnetGAN	RegAE	Constant Shift
FID	**121.63 +/- 2.12**	282.26 +/- 13.32	153.03 +/- 6.55	184.31 +/- 9.71	272.12 +/- 1.10	320.97 +/- 1.04

Table 1: FID scores on the CelebA extrapolation task.

of the target in the internal layer of our pre-trained autoencoder, and apply this single shift to all neurons in the target (Constant Shift).

For the regularized autoencoder, the regularization penalized differences in the distributions of the source and target in a latent layer using maximal mean discrepancy (Amodio et al., 2018; Dziugaite et al., 2015). The image experiment used convolutional layers with stride-two filters of size four, with 64-128-256-128-64 filters in the layers. All other models used fully connected layers of size 500-250-50-250-500. Leaky ReLU activation was used with 0.2 leak. Training was done with minibatches of size 100, with the Adam optimizer (Kingma & Ba, 2014), and learning rate 0.001.

3.1 CelebA Hair Color Transformation

We first consider a motivational experiment on the canonical image dataset of CelebA (Liu et al., 2018). If we want to learn a transformation that turns a given image of a person with black hair to that same person except with blond hair, a natural approach would be to collect two sets of images, one with all black haired people and another with all blond haired people, and teach a generative model to map between them. The problem with this approach is that the learned model may perform worse on input images that differ from those it trained on. This has troubling consequences for the growing concern of socially unbiased neural networks, as we would want model performance to go unchanged for these different populations (Tatman, 2017).

This is illustrated in Figure 2a, where we collect images that have the attribute male and the attribute black hair and try to map to the set of images with the attribute male and the attribute blond hair. Then, after training on this data, we extrapolate and apply the transformation to females with black hair, which had not been seen during training. The GAN models are less successful at modeling

this transformation on out-of-sample data. In the parts of the image that should stay the same (everything but the hair color), they do not always generate a recreation of the input. In the hair color, only sometimes is the color changed. The regular GAN model especially has copious artifacts that are a result of the difficulty in training these models. This provides further evidence of the benefits of avoiding these complications when possible, for example by using the stable training of an autoencoder and editing it as we do in neuron editing.

We quantify the success of neuron editing by using the common metric of Frechet Inception Distance (FID) that measures how well the generated distribution matches the distribution targeted for extrapolation. These scores are reported in Table 1, where we see neuron editing achieve the best result on an average of three runs. Notably, due to the autoencoder's more stable training, the standard deviation across multiple runs is also lower than the GAN-based methods.

In Figure 2b, we motivate why we need to perform the $NeuronEdit$ transformation on the internal layer of a neural network, as opposed to applying it on some other latent space like PCA. Only in the neuron space has this complex and abstract transformation of changing the hair color (and only the hair color) been decomposed into a relatively simple and piecewise linear shift.

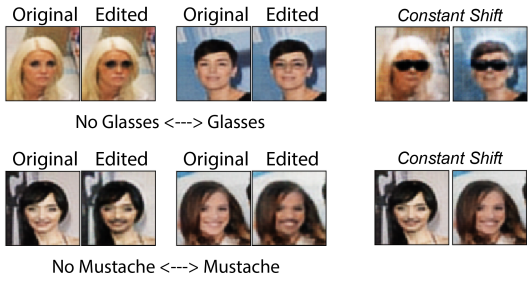

Figure 3: Additional CelebA transformations.

Beyond hair color transformation, neuron editing is able to learn general transformations on CelebA males and apply them to females. In Figure 3, we learn to transform between having/not having the mustache attribute and having/not having the glasses attribute. The latter transformation on glasses demonstrates the importance of learning a non-constant transformation. The glasses attribute is bimodal, with both examples of sunglasses and reading glasses in the dataset. With neuron editing, we are able to learn to map to each of these different parts of the latent space, as opposed to the constant shift which adds dark sunglasses to the entire distribution.

3.2 Batch correction by out-of-sample extension from spike-in samples

We next demonstrate another application of neuron editing's ability to learn to transform a distribution based on a separate source/target pair: biologicl batch correction. Batch effects are differences in the observed data that are caused by technical artifacts of the measurement process. In other words, we can measure the same sample twice and get two rather different datasets back. When we measure different samples, batch effects get confounded with the true difference between the samples. Batch effects are a ubiquitous problem in biological experimental data that (in the best case) prevent combining measurements or (in the worst case) lead to wrong conclusions. Addressing batch effects is a goal of many new models (Finck et al., 2013; Tung et al., 2017; Butler & Satija, 2017; Haghverdi et al., 2018), including some deep learning methods (Shaham et al., 2017; Amodio et al., 2018).

One method for grappling with this issue is to repeatedly measure an identical control (spike-in) set of cells with each sample, and correct that sample based on the variation in its version of the control (Bacher & Kendziorski, 2016). In our terminology of generation, we choose our source/target pair to be Control1/Control2, and then extrapolate to Sample1. Our transformed Sample1 can then be compared to raw Sample2 cells, rid of any variation induced by the measurement process. We would expect this to be a natural application of neuron editing as the data distributions are complex and the control population is unlikely to be similar to any (much less all) of the samples.

The dataset we investigate in this section comes from a mass cytometry (Bandura et al., 2009) experiment which measures the amount of particular proteins in each cell in two different individuals infected with dengue virus (Amodio et al., 2018). The data consists of 35 dimensions, where Control1, Control2, Sample1, and Sample2 have 18919, 22802, 94556, and 55594 observations, respectively. The two samples were measured in separate runs, so in addition to the true difference in biology creating variation between them, there are also technical artifacts creating variation between them. From the controls, we can see one such batch effect characterized by artificially low readings in the amount of the protein IFNg in Control1 (the x-axis in Figure 4a).

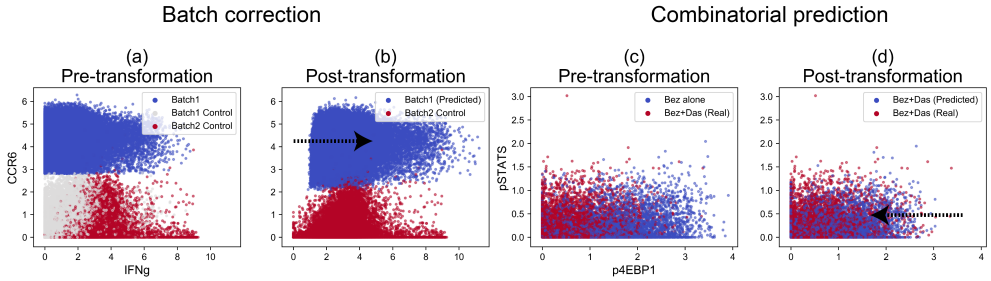

Figure 4: Two-dimensional slices of the data before and after neuron editing. In (a)-(b), we see neuron editing transform the sample (blue) to remove differences seen in the controls (red and white) and preserve differences not seen in the controls. This means it corrects the variation in IFNg while preserving the variation in CCR6. In (c)-(d), we see neuron editing's transformation correctly predicting the effect of combining two drugs.

Neuron Editing	GAN	CycleGAN	ResnetGAN	RegAE	Constant Shift	CCA	MNN	ComBat	Limma
0.96275	0.5108	0.4310	0.6268	-0.0508	0.88205	0.6034	0.5339	0.5569	0.5431

Table 2: Correlation between observed change in spike-ins and applied change to samples. Neuron editing most accurately applies just the transformation observed as batch effect and not true biological variation.

We would like our model to identify this source of variation and compensate for the lower values of IFNg without losing other true biological variation in Sample1. For example, Sample1 also has higher values of the protein CCR6, and as the controls show, this is a true biological difference, not a batch effect (the y-axis in Figure 4a).

We quantify the performance of the models at this goal by measuring the correlation between the change in median marker values observed in the spike-in with the change applied to the sample. If this correlation is high, we know the transformation applied to the samples only removes the variation where we have evidence, coming from the spike-ins, that it is a technical artifact. This data is presented in Table 2, where we compare to not only the deep generative models we have already introduced, but also dedicated batch correction methods commonly used by practitioners (Johnson et al., 2007; Haghverdi et al., 2018; Butler & Satija, 2017). We see that neuron editing outperforms all of the alternatives at extrapolating from the spike-ins to the samples. This is unsurprising, as the GAN methods are only trained to produce data like Control2, and thus will not preserve much of the variation in the sample. The traditional batch correction methods make specific parametric distributional assumptions on the data that are not held in practice, and thus also perform poorly. The regularized autoencoder, since the transformation is performed during training rather than after training like neuron editing, just reproduces its input unchanged.

In Figure 5a, a PCA embedding of the data space is visualized for Control1 (light blue), Control2 (light red), Sample1 (dark blue), and post-transformation Sample1 (dark red). The transformation from Control1 to Control2 mirrors the transformation applied to Sample1. Notably, the other variation (intra-sample variation) is preserved. In Figure 5b, we see that for every dimension, the variation between the controls corresponds accurately to the variation introduced by neuron editing into the sample. These global assessments across the full data space offer additional corroboration that the transformations produced by neuron editing reasonably reflect the transformation as evidenced by the controls.

	Neuron Editing	GAN	CycleGAN	ResnetGAN	RegAE	Constant Shift
r	**0.99661/0.97232**	0.87014/0.58130	0.92687/0.85380	0.94529/0.93032	0.91965/0.96053	0.96680/0.96925

Table 3: Correlation between real and predicted means/variances on the combinatorial drug prediction data. The GANs generate data that is less accurate (means are off) and less diverse (variances are smaller) than the real data, while neuron editing best models the true distribution.

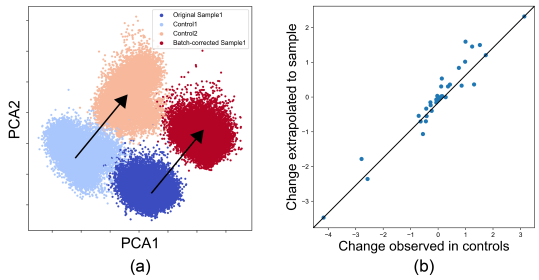

Figure 5: (a) The global shift in the two controls (light blue to red) is isolated and this variation is edited into the sample (dark blue to red), with all other variation preserved. (b) The median change in the sample in each dimension corresponds accurately with the evidence in each dimension in the controls.

3.3 Combinatorial drug treatment prediction on single-cell data

Finally, we consider biological data from a combinatorial drug experiment on cells from patients with acute lymphoblastic leukemia (Anchang et al., 2018). The dataset we analyze consists of cells under four treatments: no treatment (basal), BEZ-235 (Bez), Dasatinib (Das), and both Bez and Das (Bez+Das). These measurements also come from mass cytometry, this time on 41 dimensions, with the four datasets consisting of 19925, 20078, 19843, and 19764 observations, respectively. In this setting, we define the source to be the basal cells, the target to be the Das cells, and then extrapolate to the Bez cells. We hold out the true Bez+Das data and attempt to predict the effects of applying Das to cells that have already been treated with Bez.

We quantitatively evaluate whether neuron editing produces a meaningful transformation in Table 3, where we calculate the correlation between the real and generated means and variances of each dimension. Neuron editing more accurately predicts the principle direction and magnitude of transformation across all dimensions than any other model. Furthermore, neuron editing better preserves the variation in the real data. The GANs have trouble modeling the diversity in the data, as manifested by their generated data having significantly less variance than really exists.

We see an example of the learned transformation by looking at a characteristic effect of applying Das: a decrease in p4EBP1 (seen on the x-axis of Figure 4c). No change in another dimension, pSTATS, is associated with the treatment (the y-axis of Figure 4c). Neuron editing accurately models this change in p4EBP1, without introducing any change in pSTATS or losing variation within the extrapolation dataset (Figure 4d).

We note that since much of the variation in the target distribution already exists in the source distribution and the shift is a relatively small one, we might expect the ResnetGAN to be able to easily mimic the target. However, despite the residual connections, it still suffers from the same problems as the other models using the generating approach: namely, the GAN objective encourages all output to be like the target it trained on. This leaves it unable to produce the correct distribution if it differs from the target of the learned transformation, as we see in this case.

4 Discussion

We tackled a data-transformation problem inspired by biological experimental settings: that of generating transformed versions of data based on observed pre- and post-transformation versions of a small subset of the available data. This problem arises during clinical trials or in settings where effects of drug treatment (or other experimental conditions) are only measured in a subset of the population, but expected to generalize beyond that subset. Here we introduce a novel approach that we call *neuron editing*, for applying the treatment effect to the remainder of the dataset. Neuron editing makes use of the encoding learned by the latent layers of an autoencoder and extracts the changes in activation distribution between the observed pre-and post treatment measurements. Then, it applies these same edits to the internal layer encodings of other data to mimic the transformation. We show that performing the edit on neurons of an internal layer results in more realistic transformations of image data, and successfully predicts synergistic effects of drug treatments in biological data. Moreover, we note that it is feasible to learn complex data transformations in the non-linear dimensionality reduced space of a hidden layer rather than in ambient space where joint probability distributions are difficult to extract. Future work along these lines could include training parallel encoders with the same decoder, or training to generate conditionally.

REFERENCES

Matthew Amodio, David van Dijk, Krishnan Srinivasan, William S Chen, Hussein Mohsen, Kevin R Moon, Allison Campbell, Yujiao Zhao, Xiaomei Wang, Manjunatha Venkataswamy, et al. Exploring single-cell data with deep multitasking neural networks. *bioRxiv*, pp. 237065, 2018.

Benedict Anchang, Kara L Davis, Harris G Fienberg, Brian D Williamson, Sean C Bendall, Loukia G Karacosta, Robert Tibshirani, Garry P Nolan, and Sylvia K Plevritis. Drug-nem: Optimizing drug combinations using single-cell perturbation response to account for intratumoral heterogeneity. *Proceedings of the National Academy of Sciences*, 115(18):E4294–E4303, 2018.

Martin Arjovsky, Soumith Chintala, and Léon Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

Rhonda Bacher and Christina Kendziorski. Design and computational analysis of single-cell rna-sequencing experiments. *Genome biology*, 17(1):63, 2016.

Dmitry R Bandura, Vladimir I Baranov, Olga I Ornatsky, Alexei Antonov, Robert Kinach, Xudong Lou, Serguei Pavlov, Sergey Vorobiev, John E Dick, and Scott D Tanner. Mass cytometry: technique for real time single cell multitarget immunoassay based on inductively coupled plasma time-of-flight mass spectrometry. *Analytical chemistry*, 81(16):6813–6822, 2009.

Shane Barratt and Rishi Sharma. A note on the inception score. *arXiv preprint arXiv:1801.01973*, 2018.

Andrew Butler and Rahul Satija. Integrated analysis of single cell transcriptomic data across conditions, technologies, and species. *bioRxiv*, pp. 164889, 2017.

Gintare Karolina Dziugaite, Daniel M Roy, and Zoubin Ghahramani. Training generative neural networks via maximum mean discrepancy optimization. *arXiv preprint arXiv:1505.03906*, 2015.

Rachel Finck, Erin F Simonds, Astraea Jager, Smita Krishnaswamy, Karen Sachs, Wendy Fantl, Dana Pe'er, Garry P Nolan, and Sean C Bendall. Normalization of mass cytometry data with bead standards. *Cytometry Part A*, 83(5):483–494, 2013.

Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

Ishaan Gulrajani, Faruk Ahmed, Martin Arjovsky, Vincent Dumoulin, and Aaron C Courville. Improved training of wasserstein gans. In *Advances in Neural Information Processing Systems*, pp. 5767–5777, 2017.

Laleh Haghverdi, Aaron TL Lun, Michael D Morgan, and John C Marioni. Batch effects in single-cell rna-sequencing data are corrected by matching mutual nearest neighbors. *Nature biotechnology*, 36(5):421, 2018.

W Evan Johnson, Cheng Li, and Ariel Rabinovic. Adjusting batch effects in microarray expression data using empirical bayes methods. *Biostatistics*, 8(1):118–127, 2007.

Taeksoo Kim, Moonsu Cha, Hyunsoo Kim, Jung Kwon Lee, and Jiwon Kim. Learning to discover cross-domain relations with generative adversarial networks. *arXiv preprint arXiv:1703.05192*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Jerry Li, Aleksander Madry, John Peebles, and Ludwig Schmidt. Towards understanding the dynamics of generative adversarial networks. *arXiv preprint arXiv:1706.09884*, 2017.

Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Large-scale celebfaces attributes (celeba) dataset. *Retrieved August*, 15:2018, 2018.

Vaishnavh Nagarajan and J Zico Kolter. Gradient descent gan optimization is locally stable. In *Advances in Neural Information Processing Systems*, pp. 5585–5595, 2017.

Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pp. 2234–2242, 2016.

Uri Shaham, Kelly P Stanton, Jun Zhao, Huamin Li, Khadir Raddassi, Ruth Montgomery, and Yuval Kluger. Removal of batch effects using distribution-matching residual networks. *Bioinformatics*, 33(16):2539–2546, 2017.

Akash Srivastava, Lazar Valkoz, Chris Russell, Michael U Gutmann, and Charles Sutton. Veegan: Reducing mode collapse in gans using implicit variational learning. In *Advances in Neural Information Processing Systems*, pp. 3308–3318, 2017.

Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, pp. 12, 2017.

Rachael Tatman. Gender and dialect bias in youtubes automatic captions. In *Proceedings of the First ACL Workshop on Ethics in Natural Language Processing*, pp. 53–59, 2017.

Po-Yuan Tung, John D Blischak, Chiaowen Joyce Hsiao, David A Knowles, Jonathan E Burnett, Jonathan K Pritchard, and Yoav Gilad. Batch effects and the effective design of single-cell gene expression studies. *Scientific reports*, 7:39921, 2017.

Paul Upchurch, Jacob Gardner, Geoff Pleiss, Robert Pless, Noah Snavely, Kavita Bala, and Kilian Weinberger. Deep feature interpolation for image content changes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7064–7073, 2017.

Xiang Wei, Boqing Gong, Zixia Liu, Wei Lu, and Liqiang Wang. Improving the improved training of wasserstein gans: A consistency term and its dual effect. *arXiv preprint arXiv:1803.01541*, 2018.

Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *arXiv preprint*, 2017.