Testing the Limits of Biologically-Plausible Backpropagation

Nishith Khandwala
Department of Computer Science
Stanford University
nishith@cs.stanford.edu

Rishi Bedi
Department of Computer Science
Stanford University
rbedi@cs.stanford.edu

Abstract

Backpropagation is the core algorithm powering the training of (artificial) deep neural networks for tasks like image classification, natural language processing, drug discovery, and deep reinforcement learning. It is unclear, however, that any such process can realistically take place in the brain. Here we investigate Feedback Alignment, a way of circumventing the weight transport problem, first presented by Lillicrap et al. in 2014. We further perturb feedback alignment in various ways to probe the conditions under which it is successful, and empirically test it in a variety of settings besides MNIST classification. We also consider Dale's Law and potential ways to simulate neural networks which constrain the identities of excitatory and inhibitory neurons.

1 Introduction

Throughout this paper, we distinguish between biological neural networks and artificial neural networks by referring to the latter as "ANN"s. First we introduce backpropagation in the context of ANNs, and then move towards its biological roadblocks, and finally, some recently suggested avenues to circumventing said roadblocks. We experiment with feedback alignment, first reproducing Lillicrap et al.'s results and then extending them to convolutional networks, recurrent networks, and autoencoders. Finally, we experiment with another biological constraint not usually imposed on ANNs - we try to understand what it means for ANNs to obey Dale's Law by separating excitatory and inhibitory neurons and their synapses. In order to experiment with these ideas inspired by biological limitations, we implement them in ANNs are show our results empirically. Further theoretical work must be done to understand their mechanisms of action.

To summarize, the following are the major experiments we carry out in this paper:

1. We demonstrate feedback alignment's success on deep fully-connected networks, recurrent networks (RNNs) and convolutional networks (CNNs).

2. We attempt to train autoencoders using feedback alignment without success and suggest potential intuition as to why.

3. As suggested by [Liao et al., 2015], we demonstrate that batch normalization is sufficient (and sometimes necessary) to enable the training of deep networks with feedback alignment, and that its learnable parameters gamma and beta are negligibly useful.

4. We experiment with fixed sign weights to capture the essence of Dale's Law in ANNs but ultimately propose a new activation function, Mixed-ReLU, to do this.

1.1 Backpropagation

Backpropagation is an algorithm used to train ANNs. Consider an ANN designed to perform classification. Given an input, the network does a forward pass which includes a linear transformation followed by a non-linearity (often these "layers" are stacked upon each other to obtain more robust models) and then assigns scores to each class. The class with the highest score is considered to be the network's prediction. Initially, the network assigns arbitrary scores to each class. It still needs to "learn" what class this input represents.

This is where backpropagation comes in. The algorithm compares the predicted class with the true class and accordingly, it tries to push the network's weights (think beliefs) in a direction such that it predicts the right class next time. If the network predicted the wrong class, backprop will shift weights such that the score for that incorrect class is lower for the same input in the future. If the network predicted the right class, the algorithm will encourage the network to assign higher scores to that class, making the network more confident about its prediction.

Backpropagation achieves this optimization by "credit assignment" - penalizing neuron units in proportion to their contribution in the prediction. Mathematically, it does so by chain rule differentiation. Given a weight, w and error signal (defined by some loss between the prediction and label), y, the consequent change in w is given by dy/dx. For a more detailed explanation of backpropagation, please refer to Andrej Karpathy's notes on ANNs here: http://cs231n.github.io/optimization-2/.

Backpropagation can also be seen as an automatic "knob" adjustment mechanism for perceptrons. In the recent years, this algorithm has received a lot of attention, particularly due to its role in the success in deep learning. In this project, we attempt to provide correspondences between this algorithm and learning in the brain.

1.2 Biological Challenges to Backprop

There are numerous strong biological arguments suggesting backpropagation is implausible in the brain. The neurobiologists' concern has been articulated as early as 1989, in a paper published by David Stork (of Stanford) entitled "Is backpropagation biologically plausible?" Stork and others stop short of declaring its implausibility, but instead suggest a number of features which should be experimentally observed in cortex to either support or falsify a variety of proposed biological backprop implementations. [Stork, 1989]

In a seminar presented at Stanford this year, Geoffrey Hinton summarized four of the most damning problems neuroscientists continue to raise as evidence as to why backpropagation and neural circuitry are incompatible. [Hinton, 2016] We summarize them, and Hinton's responses here:

1. There is no obvious source of a supervision signal. Backpropagation in ANNs for supervised learning tasks require a "correct label" to be injected into the output of the ANN in order to compute the error on the current training example, which is then backpropagated through the network. In the brain, there is no clear external supervision signal for most tasks.

   Response: There are many examples of learning in artificial systems which don't rely on externally-derived supervision signals. Autoencoders, trained to reconstruct the input vector, are a good example. Alternatively, one could train a network to construct a generative model which assigns the real input a high probability. The wake-sleep algorithm developed in the 1990s was a good early approximation of this. Finally, recently, variational autoencoders (Welling and Kingma) and Generative Adversarial Networks have proved useful at successfully learning without the need to explicit label injection at train-time.

2. Neurons in the cortex do not send real values – instead, they send binary spikes. Most successful ANN backprop models use real-valued weights and signals, however – even work such as Courbariaux et al.'s BinaryConnect with binary weights use real values during the parameter update step. This requires real-valued signals to be passed even when the weights being stored and used during the forwards and backwards passes are binary.

   Response: Hinton makes the argument that converting from the real-valued signal regime to a binary spike regime is equivalent to adding Poisson noise to the real-valued signal regime. More precisely, if the task of a neuron is to communicate some real-valued signal to postsynaptic neurons, it first computes a rate of spiking that corresponds to that real value, and then sends Poisson-distributed spikes corresponding to that rate. In expectation,

this boils down to sending the correct value (encoded as a rate), plus Poisson noise, which effectively functions as a regularizer. In a regime where more parameters is generally favored in deep networks, provided good regularization exists, a biological model which implicitly regularizes is a good thing.

3. Neurons in ANNs need to be able to send two distinct types of information – the forward pass and the backwards pass. If a given post-synaptic neuron receives information from a given presynaptic neuron, how is the post-synaptic neuron supposed to disambiguate one from the other? The signals are different and one cannot be computed from the other.

   Response: The crux of Hinton's response here is that two signals can be encoded at once in a given connection: the value of the signal itself in the rate of the neuron spiking, and the error derivative with respect to its input can be encoded as the rate of change of the neuron's output. In other words, it's not necessary that the interpretation derivative of the rate of the neuron firing needs to be with respect to time – it's derivative with respect to time can actually represent an entirely different quantity (derivative with respect to the input).

4. To make problem 3 even harder: Individual pairs of neurons rarely share reciprocal connections. Entire regions of cortex certainly share forwards and backwards connections between each other, but that is on the region level, not the individual neuron level. More precisely, if there exists a connection from neuron A to neuron B, P(connection from neuron B to neuron A) is quite low. Alternatively, this issue can be characterized as the weight transfer problem: even if these reciprocal connections existed (which they usually don't), how could the reverse weights and the forward weights share the same values? This is typically thought to be necessary, since in ANNs, if W is the forward weight matrix for a given layer, W's transpose is used in backpropagating through that layer.

   Response: You can replace W's transpose in the backpropagation step through an affine layer with a random matrix and it works - it doesn't learn as quickly as backpropagation, but the network learns to use these fixed random weights to learn. This method was proposed by Lillicrap et al in 2014, which they termed "feedback alignment." Exploring feedback alignment is our first experimental objective in this paper.

1.3 Feedback Alignment

[Lillicrap et al., 2014] posit an alternative to backpropagation termed "feedback alignment," a powerfully simple and somewhat surprising concept. Backpropagation normally adjusts the weights of a hidden layer (W) proportionally to the gradient of the loss with respect to the weights. Given a simple network where W_0 is the weight matrix from the input (x) to the hidden layer, and W is the weight matrix from the hidden layer to the output layer, the change to W_0's weights is given by $\Delta W_0 \propto -(W^T e)x^T$, where e is the error signal being backpropagated through the network. In feedback alignment, however, we replace W^T with a fixed random matrix B - so our update is proportional to $-Bex^T$. This idea is readily extensible to multilayer fully connected affine layers (the standard chain rule from backpropagation is applied), but note that a different fixed random matrix is used for each affine layer's backwards pass. Furthermore, these random matrices do not change over the course of training.

The important takeaway from the feedback alignment algorithm is that no forward weight matrix is ever required for the backwards pass - the error derivative and the fixed random matrices (which need never be shared with the forward pass) are sufficient to compute the update on each element of each forward weight matrix W_i. Figure 1 shows a graphical representation of the feedback alignment mechanism. It is somewhat counterintuitive that this mechanism works - Lillicrap et al. shed some light on why it does through the lens of weight dynamics and by showing that the weight change induced by feedback alignment eventually begin to approximate the weight changes induced by backpropagation, but more theoretical investigation is required to understand why feedback alignment functions.

[Liao et al., 2015] also did interesting work that while not exactly feedback alignment, is spiritually similar: they primarily consider whether it is the signs or the magnitudes of the feedback weights which must be concordant with the actual feedback weights for "backprop" to succeed. They find that the signs are significantly more important than the magnitudes, and they demonstrate this on a variety of classification tasks using convolutional architectures.

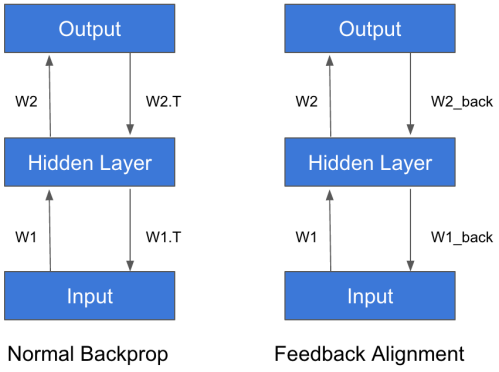


Figure 1: Feedback Alignment Mechanism

2 Our Work

2.1 Methods Summary

We are using custom Python ANN code we initially wrote as part of class assignments for Andrej Karpathy's CS231N course at Stanford (Winter 2016). The code we wrote in CS231N provides a robust framework for various experiments (it does not use any framework like TensorFlow, Theano, etc.), since we have complete control over the system and the behavior of weight initialization, and the forward and backwards passes of each layer (no auto-differentiation). We are not releasing the code online since it also constitutes solutions to most of the assignments in CS231N, but are happy to share the code upon request.

Experiments were run using the RGB CIFAR-10 dataset, usually trained on either 1,000 or 10,000 images, and validated on a held-out validation set of 1,000 images. In order to accelerate our rate of experimentation, we usually did not wait for training to converge – we are primarily concerned with finding strong evidence of successful (and firmly non-random) learning happening in each case... a good avenue for future work would be to perform a set of well-controlled experiments on each backpropagation variant with exactly the same hyperparameters and the full, 49,000 image CIFAR-10 dataset (much as Liao et al. performed for the experiments they ran).

2.2 Feedback Alignment on Fully Connected Layers

In conventional implementations of ANNs, both feedforward pass and backpropagation use symmetric weights. As discussed above, here we implement Feedback Alignment as it is described by [Lillicrap et al., 2014].

We first compared normal backpropagation to feedback alignment in fully-connected networks with 3 hidden layers of 512 ReLU units each. We found that feedback alignment was easily able to learn with no further modifications needed, but that it does learn slower than normal backpropagation. After 60 epochs of training on 10,000 images, backprop achieved 100% training accuracy, while after 60 epochs, feedback alignment only achieved 75% training accuracy. Both achieved comparable validation accuracy, however, with backprop achieving a maximum of 47.8% validation accuracy, and feedback alignment achieving 46.2%.

We then attempted to train an eight-layer net, again with each layer consisting of 512 ReLU units. Since the three-layer net trained, we expected the eight-layer net to also train with feedback alignment - but it did not. We experimented with various alterations to feedback alignment to induce training until stumbling upon batch normalization as an effective strategy to combating the increased network depth. With batch normalization, the eight-layer net trained without difficulty, on par with the three layer net that did not use batch normalization. The importance of batch normalization was suggested by [Liao et al., 2015] for convolutional networks - we affirm their finding for deep fully-connected networks as well. The three layer net with batch normalization performed the best out of the whole set.

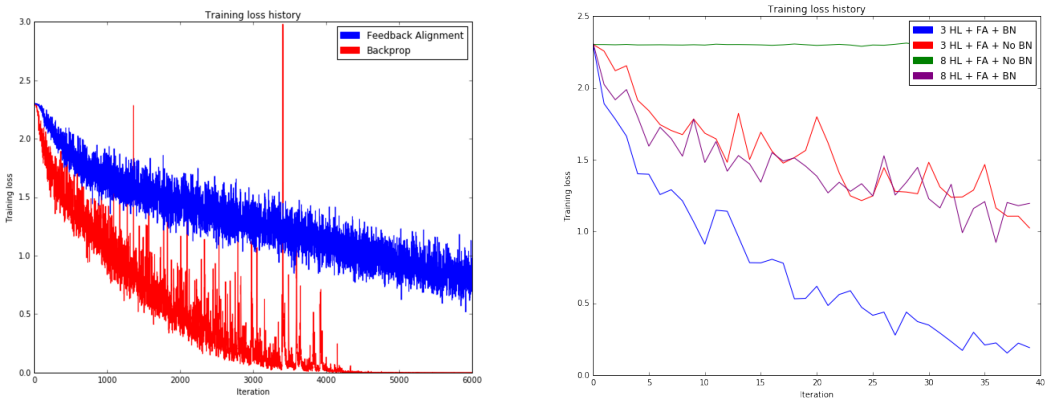


Figure 2: Left: Backpropagation outperforms feedback alignment, but feedback alignment still trains effectively. Right: An 8-layer net needs batch normalization to train. (HL = hidden layers, FA = feedback alignment, BN = Batch Normalization)

The eight layer network without batch normalization did not train at all, over a variety of attempted learning rates (only 1e-2 shown).

2.3 Batch Normalization

The success of batch normalization in getting deep networks to learn led us to a further investigation of batch normalization. Batch Normalization was introduced as a method of reducing internal covariate shift, a problem introduced by the fact that each layer's input distribution changes as the network learns [Ioffe and Szegedy, 2015]. The batch normalization transformation as presented by Ioffe and Szegedy consists of computing the mean and variance of the current minibatch, then normalizing the input of the current minibatch by the following formula:

$$\hat{x}_i = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \tag{1}$$

... where μ_B and σ_B^2 denote the mean and variance of the minibatch, respectively. This makes sure that the normalized outputs of the BN transform have mean 0 and variance 1, assuming the minibatch consists of data points sampled from the same distribution (a reasonable assumption). Finally, Ioffe and Szegedy propose the actual output of the BN transform to be:

$$y_i = \gamma \hat{x}_i + \beta \tag{2}$$

... where γ and β are learned parameters of the network.

First, we found that Batch Norm is very resistant to changes in batch size – we ran various sets of experiments on batch sizes ranging from 2 to 100 and had success across the board (data not shown). All of the experiments in this section were run with feedback alignment.

The original presentation of batch norm leverages two learnable parameters, γ and β, which they apply as a final affine transformation to the output of the forward pass of the batch norm layer. While batch normalization itself isn't out of the question for a biological network, having additional learnable parameters didn't sit well with us, so we wanted to experiment to see if γ and β actually played a crucial role. [Liao et al., 2015] allude to the fact that they weren't especially useful. Indeed, we find that eliminating γ and β from the forward pass of batch norm entirely does not affect performance at all. Batch Norm and Batch Norm without γ and β both successfully overfitted 1,000 training examples to 100% accuracy, and achieved validation accuracy on a hold-out set of 1,000 more examples of 36%.

We then investigated the effect of not using sample mean and variance at train time at all – what if we simply used the running mean and variance (accumulated over all previous batches not including

the current one) which are already used at test time? We found that replacing sample mean and sample variance with running mean and running variance is bad for all batch sizes (1...100) with all momentum values (0 - 0.9). This makes some sense because the intuition for why batch norm is useful is that different batches have significantly different means and variances, and batch norm... well... normalizes for those differences. Training based on the "wrong batch"'s values, is not helpful, if you use the running mean and running variance from only previous batches, before incorporating the current batch's sample mean and variance into the running averages. When we did incorporate the current batch's sample mean and variance into the running averages before using them, however, performance is as strong (or stronger) as using the sample mean and variance alone for normalization during training.

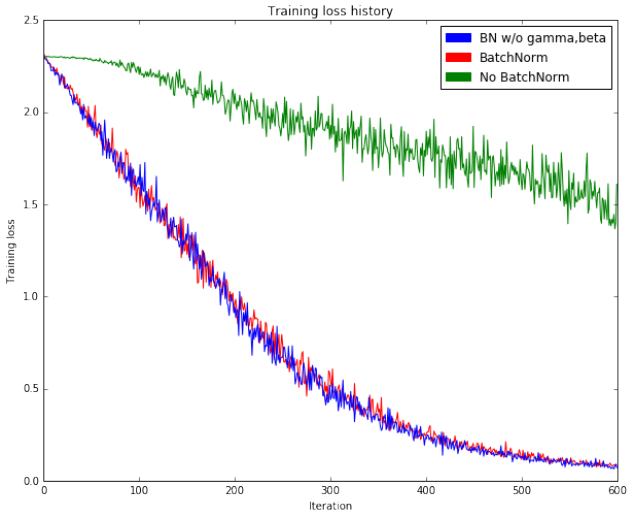


Figure 3: γ and β learned parameters do not measurably improve batch norm performance.

2.4 Convolutional Networks

Convolutional Networks or ConvNets have become increasingly popular in the computer vision community. Given an input image, the ConvNet recognizes spatial patterns and develops hierarchical representations to then perform tasks such as object classification. More specifically, the network applies a filter that makes a strided pass across the image. At each stride, the ConvNet develops local understanding of the image. The collective local information is then used to feed higher order concepts. For instance, given an image of a face, it first identifies edges, then creates a notion of an eyes and then recognizes the input as a face.

ConvNet architecture is on some level inspired by the organization of the visual cortex. Realizing the relevance of developing hierarchical representations of inputs in the brain, we attempt to apply feedback alignment to ConvNets. This is also interesting from an empirical standpoint since few previous works [Liao et al., 2015] have studied the biological plausibility of ConvNets. In a conventional ConvNet, both inference and backpropagation use weight symmetry i.e. use the same weights (transposed for one task).

When an input image is fed into the network, the filters create hidden representations and use them for the task the ConvNet was designed for. The loss, given by comparing the output and ground truth, is backpropagated through all the layers. It is important to note here that, for a given layer, the filter shares weights across all strides. In our feedback alignment implementation of the ConvNet, the weights on the backward pass are replaced by random matrices (which are initialized with the same dimensions as the backward pass weights). These random weights are never updated. The error signal, after interacting with these random weights, is used to then update the feedforward weights.

From a machine learning view, the error signal in the case of feedback alignment pushes the inputs belonging to different classes apart (in the case of classification), but just not in the direction a conventional ConvNet would have. But, it still accomplishes the task. This was also observed by our

implementation. The feedback-aligned ConvNet learned more slowly than the backprop ConvNet (with and without Batch Norm) but was clearly able to learn. These models were all trained and validated on subsets of 1000 images from CIFAR-10 - reproducing with a larger subset is an important next step that we didn't take due to the speed of our hand-written convolutional code.

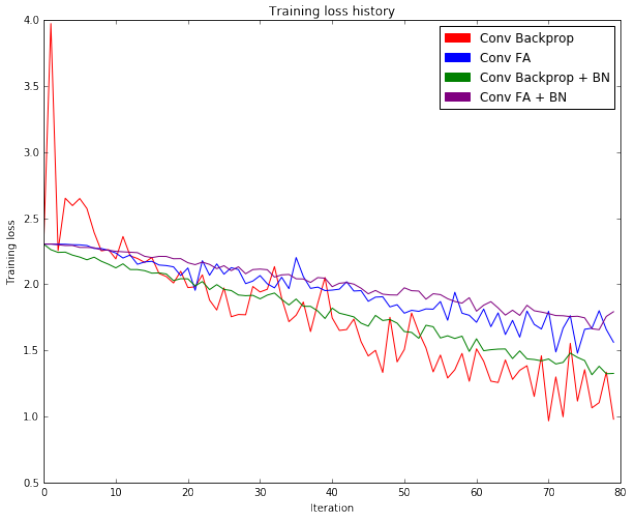


Figure 4: Feedback alignment is an effective substitute for backpropagation in CNNs.

2.5 Recurrent Networks

Recurrent networks (RNN) have been shown to work great with sequences. A RNN accepts a series of inputs, say words and predict the next word or words. It starts with a random hidden state, accepts an input, updates its hidden state, and repeats this on the remaining time-steps. The final hidden state is expected to be a high level representation of the entire input sequence. For instance, in the field of natural language processing, RNNs are used to build paragraph vectors - low dimensional embeddings that capture the semantic meaning of the constituent words individually and the paragraph as a whole. This sequence-to-sequence learning architecture has been also used to model time series processes in the brain. Given its relevance to neuroscience, we thought that it would be interesting to probe RNNs with feedback alignment and analyze its learning capacity.

Like fully-connected and convolutional networks, A conventional RNN uses weight symmetry. The backward pass uses the transpose of the weights used in the forward pass. Given an input x, we obtain the hidden state at time t, $h(t) = \sigma(Hh_{t-1} + x(t))$. Here, h_{t-1} is the hidden state at time $(t-1)$, σ is the sigmoid non-linearity function and H is a weight matrix. Using this hidden state, we obtain the prediction, y at time t by: $y(t) = softmax(Uh(t))$. Here, U is another weight matrix. Note that these two weights are applied at all time-steps during the forward pass and their transposes are used during backpropagation. However, as we discuss in the previous sections, this is not biologically plausible. The human brain, owing to the circuitry, cannot have a two-way stream. Feedback alignment tries to circumvent around this problem.

In addition to H and U, we introduce H_{back} and U_{back} as weight matrices to be used during backpropagation. Both of these backprop weights are randomly initialized and are constrained to have the same dimensions as their forward counterparts. With feedback alignment in place, the RNN uses the forward weights, H and U to do inference and then learns using the backprop weights, H_{back} and U_{back}. The error signal is then used to update the forward pass weights. Note that the backprop weights are never updated. At its core, they are random matrices that are multiplied by the error signal.

Upon training, we observe that the RNN with feedback alignment (RNN-FA) required a lower learning rate than a conventional RNN to have stable learning. This meant that RNN-FA learned rather slowly. However, it is important to note that RNN-FA was able to achieve similar performance as the conventional RNN, albeit slowly. The plots below assert our claim.

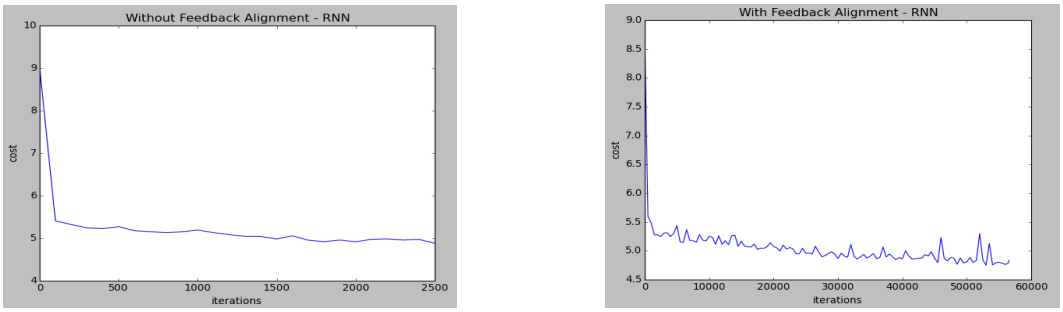


Figure 5: Left: A conventional RNN training curve Right: RNN-FA training curve

We speculate that the change in the learning rate was deemed necessary due to the complexities of backpropagation-through-time (bptt). Since the RNN uses a single parameter to interact with the hidden state at all time steps, there is compound adverse effects on the error signal. This might explain why a smaller learning rate allows the RNN-FA to train.

2.6 Autoencoders

Autoencoders form a category of unsupervised frameworks used to obtain high level representations. Given an input, the autoencoder tries to reconstruct it, optimizing to minimize the "difference" between the reconstruction and the original input. The input acts like a label, making the process unsupervised (self-supervised, to be precise). After training, the resulting hidden layer is expected to capture semantic meaning associated with the input. In case of images as input, the "hidden" characteristics such as composition, appearance and edges form are stored in this representation.

In terms of architecture, a standard autoencoder consists of multiple hidden layers stacked upon each other followed by a final layer whose output size is the same as the input size thereby allowing reconstruction. We attempted to train a variant of this architecture called the sparse autoencoder (sparse-ae) with two hidden layers to learn to a toy dataset. A sparse-ae is exactly like the standard autoencoder except it prefers larger hidden units and penalizes the cost function for dense activations.

Applying feedback alignment to autoencoders would provide an insight on the generalizability of feedback alignment to unsupervised learning. So, we removed weight symmetry from our architecture and introduce new weights for backpropagation. The autoencoder would receive an input, use its feedforward weights to reconstruct, compute the loss and backpropagate the error signal using the backward weights. Although this mechanism sounds similar to our previous experiments, there is a subtlety. The random weight that is multiplied by the error signal in the backward pass distorts the model's "view" of the input image. Since this model is a generative one, it is not able to learn what the input image and by extension, the reconstruction should look like.

Figure 6: Left: The weights of a conventional autoencoder Right: The weights of a autoencoder trained with feedback alignment. The weights look random, indicating that the network failed to learn good representations

As a result, we did not obtain good reconstructions. We were unable to successfully train the autoencoder with feedback alignment. Even though the loss decreased, the reconstructions did not get

better. This means that the autoencoder was unable to learn good representations, While disappointing, the subtlety mentioned in the previous paragraph removes the surprise element. [Bengio et al., 2015] is a more successful attempt at building a biologically-plausible auto-encoder, though it is unrelated to the idea of feedback alignment, and more concerned with interpreting spike-timing dependent plasticity (STDP) as a form of gradient descent leading to a denoising autoencoder.

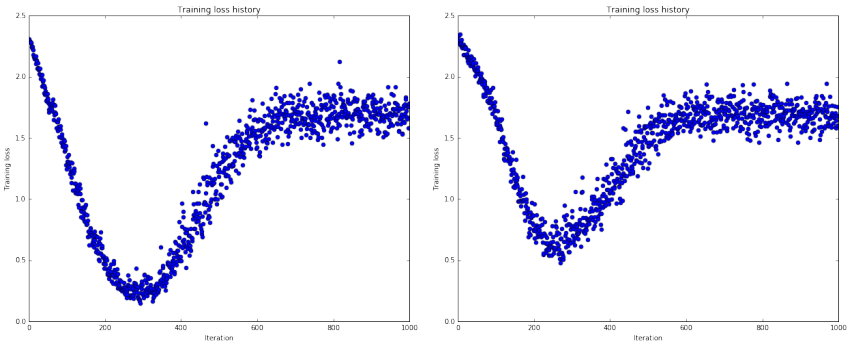
2.7 Fixed-Sign Weights

Another argument against the biological plausibility of backpropagation is its reliance on changing the signs of weights. In other words, if a neuron's weight in an ANN is randomly initialized to be positive, there is no guarantee it will stay positive. Indeed, in an unconstrained ANN setting on a standard classification task, weight sign reversals occur quite frequently.

Our goal here is to model Dale's Principle – the law that a given neuron performs the same chemical action at all of its synaptic connections to other neurons. It's not obvious what the correct ANN formulation of this constraint is - is it to constrain weights of individual synapses to be positive or negative across feedback alignment/backprop updates? Or is it to constrain the signs of entire rows of the weight matrix (i.e., all of a single neuron's outbound synapses)? We want to build an ANN model that is akin to the biological notion of neurons being either excitatory or inhibitory, without the ability to switch between the two, to test whether or not this limits learning potential.

We start by only constraining the sign of individual synapses, instead of entire neurons (i.e., we allow an individual neuron to emit both excitatory and inhibitory projections, but do not allow an individual weight to change sign once initialized – if a sign change is attempted by the feedback alignment learning process, we instead set the weight of the synapse to 0 – though it is allowed to change and become non-zero as learning continues). With random, equal probability selection between starting as excitatory (positive weight) or inhibitory (negative weight), we find that backpropagation is still effective at learning, at no observable deficiency compared to sign-unconstrained backpropagation, but that if training continues indefinitely, eventually feedback alignment (and backprop) attempt to change the signs on too many synapses, sending too many synapses to zero for the network to retain what it learned – the network eventually unlearns even what it had learned before.

This does not match the biological constraint directly, however. To actually echo Dale's principle, we need to constrain the entire row in the weight matrix corresponding to each neuron to be either positive or negative. In this case, too, we see effective learning until a certain inflection point after which too many of the weights have "attempted" a sign switch (pushing them to zero) and the network "unlearns".



(a) Fixed Synapses (elements of W) (b) Fixed Neurons (rows of W)

2.8 Mixed ReLU Activation Function

While the weight constraint idea is interesting to investigate, especially its eventual un-learning, it's not really a good approach to simulating Dale's Law in ANNs. Our extremely simplistic model which zeroes out weights when they attempt to cross from positive to negative or vice versa is certainly wrong and not founded on any strong biological basis. Moreover, it's not really the weights which should be constrained, but instead, the **outputs** from each layer. To achieve this, we design a new activation function to replace the ReLU activation function we have used thus far in all our models.

To refresh, normal ReLU does the following:

$$out = \max(in, 0) \tag{3}$$

Our Mixed-ReLU activation function instead takes each row of output of $Wx + b$ and compares the sign of each element to a mandated sign which was randomly selected for that neuron (corresponding to a row of the weight matrix) at initialization (i.e., excitatory (+) or inhibitory (-)). If an element in a given row of $Wx + b$ (i.e., output signal) is the wrong sign compared to what the neuron which produced it is allowed to produce, it is set to 0. This forces positively-initialized neurons to only produce positive signal, and negatively-initialized neurons to only produce negative signal over the entire course of training.

$$out_i = \begin{cases} \max(in, 0), & \text{if neuron i is excitatory} \\ \min(in, 0), & \text{if neuron i is inhibitory} \end{cases}$$

We believe this better captures the spirit of Dale's Law. We find that using Mixed-ReLU (in conjunction with feedback alignment) slows training down slightly but does not qualitatively affect the network's ability to learn. All these experiments were run with parameter-free batch normalization. This is reassuring because it suggests that allowing neurons to freely output both positive and negative signal (which by Dale's law we know does not happen) is not necessary for backpropagation (or feedback alignment-based) training.

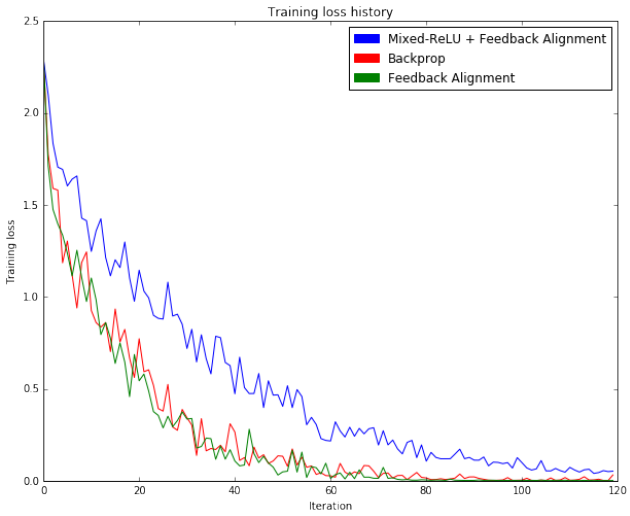


Figure 8: Mixed-ReLU slows training from feedback alignment with normal ReLU but is still successful.

3 Future Work

Our work, much like [Liao et al., 2015] is very empirical - much theoretical understanding is needed to support the conclusions drawn here. We also would like to further investigate the Dale's Law issue, and build a more robust model of neuron activation which properly captures the dynamics of excitatory/inhibitory networks instead of traditional ANNs whose neurons are essentially of uniform character.

Acknowledgments

We thank Surya Ganguli and Friedemann Zenke for their guidance, and Andrej Karpathy and the CS231N course staff for the assignments whose code we built off of.

References

Yoshua Bengio, Dong-Hyun Lee, Jorg Bornschein, and Zhouhan Lin. Towards Biologically Plausible Deep Learning. *arXiv preprint arxiv:1502.0415*, page 18, 2015. ISSN 0717-6163. doi: 10.1007/s13398-014-0173-7.2. URL http://arxiv.org/abs/1502.0415.

Geoffrey Hinton. Can the brain do backpropagation? *Stanford Seminar*, 2016.

Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167*, pages 1–11, 2015. ISSN 0717-6163. doi: 10.1007/s13398-014-0173-7.2. URL http://arxiv.org/abs/1502.03167.

Qianli Liao, Joel Z Leibo, and Tomaso Poggio. How important is weight symmetry in backpropagation? *arXiv preprint arXiv:1510.05067*, 2015.

Timothy P. Lillicrap, Daniel Cownden, Douglas B. Tweed, and Colin J. Akerman. Random feedback weights support learning in deep neural networks. *arXiv:1411.0247 [cs, q-bio]*, pages 1–27, 2014. URL http://arxiv.org/abs/1411.0247$\delimiter"026E30F$nhttp://www.arxiv.org/pdf/1411.0247.pdf.

D.G. Stork. Is backpropagation biologically plausible? *Neural Networks, 1989. IJCNN., International Joint Conference on*, 2:241–246, 1989. doi: 10.1109/IJCNN.1989.118705.