Linear Backprop in non-linear networks

Mehrdad Yazdani
Qualcomm Institute
University of California San Diego
La Jolla, CA 92093
myazdani@gmail.com

Abstract

Backprop is the primary learning algorithm used in many machine learning algorithms. In practice, however, Backprop in deep neural networks is a highly sensitive learning algorithm and successful learning depends on numerous conditions and constraints. One set of constraints is to avoid weights that lead to saturated units. The motivation for avoiding unit saturation is that gradients vanish and as a result learning comes to a halt. Careful weight initialization and re-scaling schemes such as batch normalization ensure that input activity to the neuron is within the linear regime where gradients are not vanished and can flow. Here we investigate backpropagating error terms only linearly. That is, we ignore the saturation that arise by ensuring gradients always flow. We refer to this learning rule as Linear Backprop since in the backward pass the network appears to be linear. In addition to ensuring persistent gradient flow, Linear Backprop is also favorable when computation is expensive since gradients are never computed. Our early results suggest that learning with Linear Backprop is competitive with Backprop and saves expensive gradient computations.

1 Overview

It is has been long known that deep neural networks with non-polynomial and non-linear units are universal function approximators [Leshno *et al.*, 1993]. In the early days of neural network research, however, it was not clear what learning algorithms would find the optimal set of synaptic weights for effective learning. Rosenblatt's pioneering work essentially only learned the weights at the output layer of the Multi-Layer Perceptron and keeping the input-layer weights fixed at random while Fukushima used Hebbian learning. Backprop as introduced into neural network research [Werbos, 1974; Rumelhart *et al.*, 1986] has been enormously successful at learning diverse sets of tasks by various deep neural architectures and as a result is by far the most used learning algorithm.

Although enormously successful, Backprop is a highly sensitive learning algorithm and numerous tricks have been collected to make it work in practice [Baydin *et al.*, 2016; Bottou, 2012]. Some of these issues are: dead or saturated units, appropriate learning rates, batch sizes, number of epochs in addition to many other issues. In particular, considerable effort has been placed into avoiding saturated units. The primary problem with saturated neurons is that gradients vanish in these regions and hence learning comes to a halt. As a result, considerable effort has been placed into ensuring that the input activity to neurons are in the linear region. Some of these efforts are the introduction of regularization such as l_2 penalty (also referred to as weight decay) [Krogh and Hertz, 1992; Srivastava *et al.*, 2014], batch normalization [Ioffe and Szegedy, 2015], and careful weight initialization schemes [Glorot and Bengio, 2010; Mishkin and Matas, 2015]. Other solutions is to consider activation functions that have limited saturating regions to ensure gradient flows [Nair and Hinton, 2010; Klambauer *et al.*, 2017]. [Gulcehre *et al.*, 2016] extensively study activation functions with non-saturating regions to provide gradients.

Since gradient flow is essential for learning, we investigate learning algorithms that ensure linear gradient flow. The Linear Backprop algorithm (see **Algorithm 2** below) ensures gradients flows for all regions and can be used as an alternative for learning. Compared with Backprop as shown in **Algorithm 1** [Goodfellow *et al.*, 2016], the forward pass in Linear Backprop is identical. The network architecture is still highly non-linear with non-linear activation functions, but when we compute the loss function we only consider linearly backpropagating errors. Since in Linear Backprop the derivatives of the activation functions are not computed (highlighted in red), the savings in computation in Linear Backprop compared to Backprop is $\mathcal{O}(ml)$.

Another way to think of our proposed learning rule is that we introduce a regularization term such that when gradients are computed, all the non-linear components are cancelled and only linear gradients persist. In other words, during inference we use a deep non-linear network however during training the loss function essentially reduces to a deep linear neural network. The forward pass is computed as usual but the backward pass only uses linear feedback terms.

Several recent investigations have considered variants to Backprop for biological plausibility [Baldi and Sadowski, 2016; Lillicrap *et al.*, 2016; Bartunov *et al.*, 2018] . In particular, [Lillicrap *et al.*, 2016] showed that learning is also possible with **random** weights. [Baldi and Sadowski, 2016] exhaustively considers many Hebbian and error back propagation learning algorithms. Linear Backprop also shares many similarity with the Straight Through Estimator [Bengio *et al.*, 2013], however we propose applying the estimator to any activation function.

Instead we suggest that Linear Backprop saves considerable computational costs as gradients are never computed. While further research is needed to understand random and alternative learning rules, the Linear Backprop learning rule that we consider here is especially favorable for cases with limited computing resources. Our empirical results suggest that Linear Backprop in certain conditions can be competitive with Backprop.

Algorithm 1: Backprop

```
Require Network depth $l$
Require Activation function $f(\cdot)$
Require Parameters $\boldsymbol{W}^{(i)}$ for $i \in \{1, \dots, l\}$
Require Inputs $\mathbf{x}$ and targets $\mathbf{y}$, loss function $L$
Compute $\hat{y}$, hidden activation's $a^{(i)}$ and $h^{(i)}$
(e.g. Algorithm 6.3 in Goodfellow et al. [2016])

$g \leftarrow \nabla_{\hat{y}} L(\hat{y}, y)$
for $k = l, l-1, \dots, 1$ do
    $g \leftarrow g \odot f'(a^{(k)})$
    $\nabla_{\boldsymbol{W}^{(k)}} L(\hat{y}, y) = g h^{(k-1)\top}$
    $g \leftarrow \boldsymbol{W}^{(k)\top} g$
```

Algorithm 2: Linear Backprop

```
Require Network depth $l$
Require Activation function $f(\cdot)$
Require Parameters $\boldsymbol{W}^{(i)}$ for $i \in \{1, \dots, l\}$
Require Inputs $\mathbf{x}$ and targets $\mathbf{y}$, loss function $L$
Compute $\hat{y}$, hidden activation's $a^{(i)}$ and $h^{(i)}$
(e.g. Algorithm 6.3 in Goodfellow et al. [2016])

$g \leftarrow \nabla_{\hat{y}} L(\hat{y}, y)$
for $k = l, l-1, \dots, 1$ do
    $g \leftarrow g \odot a^{(k)}$
    $\nabla_{\boldsymbol{W}^{(k)}} L(\hat{y}, y) = g h^{(k-1)\top}$
    $g \leftarrow \boldsymbol{W}^{(k)\top} g$
```

1.1 Synthetic data example

We begin by considering a synthetic data example to investigate the overfitting and generalization capabilities of Linear Backprop. There has been considerable recent interest in the generalization capabilities of deep neural networks [Zhang *et al.*, 2016; Poggio *et al.*, 2018]. The gist of the research is how networks that have far more parameters than training data (often by many orders of magnitude) generalize well to unseen data. Many experiments have shown that even when training loss is 0 (severe overfitting), there is still generalization [Zhang *et al.*, 2016; Wu *et al.*, 2017].

Here we reproduce the motivating example from [Wu *et al.*, 2017] to understand if Linear Backprop shares similar generalization properties. We consider learning the third-order polynomial $y = x^3 - 3x^2 - x + 1 + N(0, 0.1)$. In this experiment, the training set consists of only 5 points and the neural network is trained until the training error is small (for example, $\leq 1 \times 10^{-6}$). The neural network is a feed forward Multi-Layer Perceptron (MLP) with 4 hidden layers each with width of 50 ReLU units (over 7,000 parameters). For a problem with 5 points, this is a highly overparameterized network.

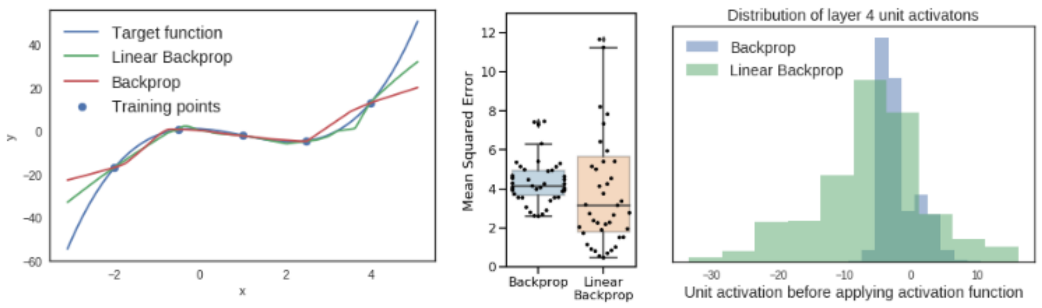

Figure 1: **(Left)** The same MLP architecture overfitted on 5 samples with Backprop vs Linear Backprop. As discussed in Wu *et al.* [2017], though the MLP is overparameterized for this simple problem, Backprop still learns solutions that generalize well. Our results suggest that Linear Backprop also generalizes. **(Middle)** We compute the MSE between the networks prediction and the target function for 50 different random initializations of weights in the MLP using the same 5 points for overfitting. **(Right)** The histogram of the unit activations after training with sigmoidal activation functions. Since Linear Backprop still manages to learn with saturated regions, unit activations are much higher. Similar results hold for different layers and ReLU activation functions.

We reproduce the result from [Wu *et al.*, 2017] that such an overparameterized network overfits the training data yet also generalizes gracefully to unseen data (see Figure 1). We also observe similar generalization capability with the same architecture when trained with Linear Backprop. The Linear Backprop also overfits on the training data and has similar generalization when trained with Backprop.

1.2 VGG and ResNet Binarized Neural Networks on CIFAR-10

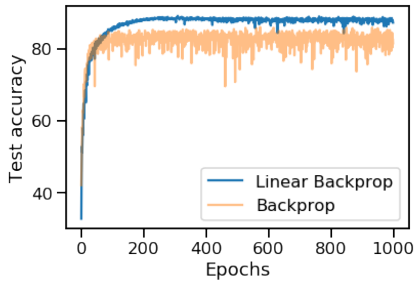

Figure 2: Learning curves for VGG19 on CIFAR-10

We compare Backprop and Linear Backprop on the CIFAR-10 data set using two different architecture: the 19 layer VGG architecture [Simonyan and Zisserman, 2014] and a Binarized Neural Network [Courbariaux *et al.*, 2016] using the ResNet architecture [Hubara, 2018]. For VGG architecture, we use tanh activation functions with vanilla SGD and sweep the learning rate in $\{10^{-3}, 10^{-4}, 10^{-5}\}$ and l_2 penalty in $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. In all experiments we use a batch size of 128 points, train with 100 epochs, and evaluate on the same test set and select the best leaning rate and l_2 for Backprop and Linear Backprop respectively. Figure 2 shows the learning curves on the test set for the best learning rate and l_2 penalty for each learning rule. The tanh activation function is in particular prone to vanishing gradients since it has many saturating regions. By using Linear Backprop, however, we are able to ensure that gradients flow to continue learning.

We also consider how learning changes when the architecture is changed. For the ResNet Binarized Neural Network, we use the hard tanh activation function and the same ResNet architecture in [Courbariaux *et al.*, 2016]. We similarly use only vanilla SGD with sweeping the learning learning rate in $\{10^{-3}, 10^{-4}, 10^{-5}\}$ and l_2 in $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$. Again, in all experiments we use a batch size of 128 points, train with 100 epochs, and evaluate on the same test set. We train this architecture with both Backprop and Linear Backprop. At 100 epochs, the best Precision@5 validation error we find using Backprop is 28.11% whereas using Linear Backprop the best best Precision@5 validation error is 18.43%. These networks are significantly slower to train and in our investigation are are interested in knowing which learning algorithm is more favorable at the start of training. Although these Precision levels are far from state-of-the-art, at only 100 epochs we find that Linear Backprop is highly competitive with traditional Backprop and that further fine-tuning (longer epochs and better batch size) will yield much better results.

Acknowledgments

This work was supported in part by NSF awards CNS-1730158, ACI-1540112, ACI-1541349, and the University of California San Diego's California Institute for Telecommunications and Information Technology. We also thank Larry Smarr, Tom DeFanti, John Graham, and Dmitry Mishin for the support and guidance provided.

References

Pierre Baldi and Peter Sadowski. A theory of local learning, the learning channel, and the optimality of backpropagation. *Neural Networks*, 83:51–74, 2016.

Sergey Bartunov, Adam Santoro, Blake A Richards, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. *arXiv preprint arXiv:1807.04587*, 2018.

Atılım Güneş Baydin, Barak A Pearlmutter, and Jeffrey Mark Siskind. Tricks from deep learning. *arXiv preprint arXiv:1611.03777*, 2016.

Yoshua Bengio, Nicholas Léonard, and Aaron Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv preprint arXiv:1308.3432*, 2013.

Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

Caglar Gulcehre, Marcin Moczulski, Misha Denil, and Yoshua Bengio. Noisy activation functions. In *International Conference on Machine Learning*, pages 3059–3068, 2016.

Itay. Hubara. Binarized neural network (bnn) for pytorch. `https://github.com/itayhubara/BinaryNet.pytorch`, 2018.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 971–980, 2017.

Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

Moshe Leshno, Vladimir Ya Lin, Allan Pinkus, and Shimon Schocken. Multilayer feedforward networks with a nonpolynomial activation function can approximate any function. *Neural networks*, 6(6):861–867, 1993.

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random synaptic feedback weights support error backpropagation for deep learning. *Nature communications*, 7:13276, 2016.

Dmytro Mishkin and Jiri Matas. All you need is a good init. *arXiv preprint arXiv:1511.06422*, 2015.

Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

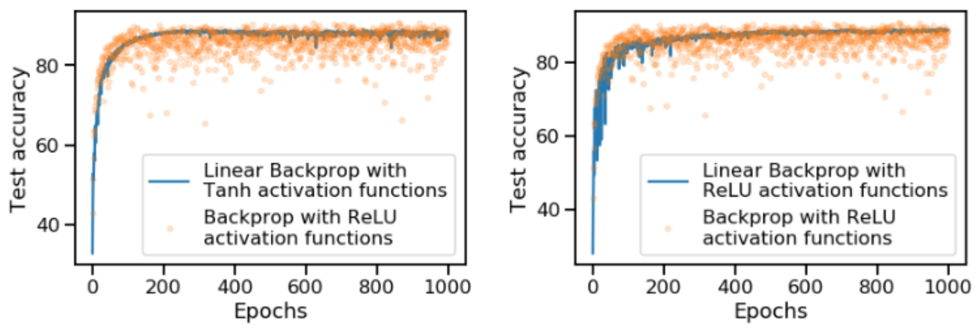

Figure 3: **(Left)** Comparing learning CIFAR-10 using the ReLU activation function with Backprop against using the tanh activation function using Linear Backprop. Learning ReLU with Backprop can yield similar results to tanh with Linear Backprop, although it appears the variance with Backprop learning is much higher.**(Right)** Similarly, we compare learning CIFAR-10 using the ReLU activation function with Backprop against using the tanh activation function using Linear Backprop. Learning ReLU with Backprop can yield similar results to tanh with Linear Backprop, although it appears the variance with Backprop learning is much higher.

Tomaso Poggio, Qianli Liao, Brando Miranda, Andrzej Banburski, Xavier Boix, and Jack Hidary. Theory iiib: Generalization in deep networks. *arXiv preprint arXiv:1806.11379*, 2018.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Paul Werbos. Beyond regression:" new tools for prediction and analysis in the behavioral sciences. *Ph. D. dissertation, Harvard University*, 1974.

Lei Wu, Zhanxing Zhu, et al. Towards understanding generalization of deep learning: Perspective of loss landscapes. *arXiv preprint arXiv:1706.10239*, 2017.

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

A Comparing Linear Backprop on ReLU activation functions on CIFAR-10 with VGG19

The ReLU activation function is designed to ensure that gradients flow maximally. For this reason we compare how learning with ReLU activation functions compare with Linear Backprop. We again sweep the learning rate in $\{10^{-3}, 10^{-4}, 10^{-5}\}$ and l_2 penalty in $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ for 100 epochs and select the best learning rate and $l2$ for a all activation function and learning algorithm combinations. Figure 3 (left) shows 1000 epochs of learning comparing Linear Backprop with tanh and Backprop with ReLU.

Unlike Backprop with tanh shown in Figure 2, Backprop with ReLU is able to match learning with tanh and Linear Backprop. However, the variance of Backprop learning with ReLU is significantly higher. We see similar results when we compare Linear Backprop learning with ReLU and Backprop learning with ReLU as shown in Figure 3 in the right.

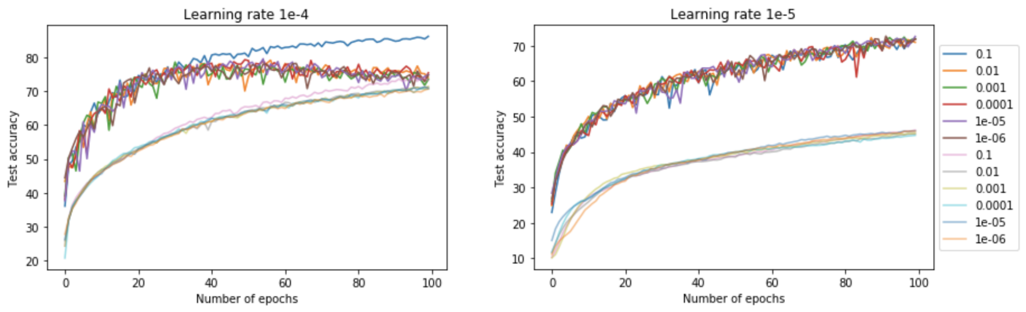

Figure 4: **(Left)** Comparing Backprop and Linear Backprop learning curves on CIFAR-10 test accuracy for fixed VGG architecture with varying learning rates and weight decay terms (higher is better). Light and dark color markings indicate Backprop and Linear Backprop respectively, different color indicate different weight decay rates. For large learning rate, to avoid unit saturation and stay in linear regime of the neuron, larger weight decay is required for Linear Backprop as we expect.

B Sweeping different learning rates and weight decays for VGG19 on CIFAR-10

In all our VGG19 experiments, we sweep the learning rate in $\{10^{-3}, 10^{-4}, 10^{-5}\}$ and l_2 penalty in $\{10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}\}$ for 100 epochs. It is instructive to observe how learning progresses for these different parameter settings. In Figure 4 we show the learning curves using the tanh activation function compared with Backprop (lighter color) and Linear Backprop (darker color). In the early stages of learning, Linear Backprop in particular performs favorably compared with Backprop learning.