ICML Workshop on Identifying and Understanding Deep Learning Phenomena (2019)

Function changes in the Backpropagation equation is equivalent to an implicit learning rate

Abstract

The backpropagation algorithm is the de-facto standard for credit assignment in artificial neural networks due to its empirical results. Since its conception, variants of the backpropagation algorithm have emerged. More specifically, variants that leverage function changes in the backpropagation equations to satisfy their specific requirements. Feedback Alignment is one such example, which replaces the weight transpose matrix in the backpropagation equations with a random matrix in search of a more biologically plausible credit assignment algorithm. In this work, we show that function changes in the backpropagation procedure is equivalent to adding an implicit learning rate to an artificial neural network. Furthermore, we learn activation function derivatives in the backpropagation equations to demonstrate early convergence in these artificial neural networks. Our work reports competitive performances with early convergence on MNIST and CIFAR10 on sufficiently large deep neural network architectures.

1. Introduction

Credit assignment (Schmidhuber, 2015) is the task of identifying neurons and weights that are responsible for a desired prediction. Currently, the backpropagation (BP) algorithm (Rumelhart et al., 1986) is the de-facto standard for credit assignment in artificial neural networks. The backpropagation algorithm assigns credit by computing partial derivatives for weights and neurons with respect to the networks cost function.

Variants of the backpropagation procedure have emerged since its conception. More specifically variants that exploit function changes in the backpropagation procedure. Feedback Alignment (Lillicrap et al., 2016) is considered a biologically plausible alternative to vanilla backpropagation. Feedback alignment is a variant of the backpropagation algorithm that uses a random weight matrix instead of the weight transpose matrix in the backpropagation equation. Despite not scaling to the ImageNet dataset (Bartunov et al., 2018), the algorithm relaxes BP weight symmetry requirements and demonstrate comparable learning capabilities to that of BP on small datasets

Similarly, Alber et al. (2018) produced unique backpropagation equations to train an artificial neural network by learning parts of the backpropagation equations. Alber et al. (2018) report early convergence on unique backpropagation equations for CIFAR10. In this work, we demonstrate that function changes in the backpropagation equations particularly activation function derivatives is equivalent to adding an implicit learning rate in stochastic gradient descent.

2. Backpropagation

The backpropagation algorithm iteratively computes gradients for each layer, from output to input, in a neural network using the derivative chain rule. Furthermore, we can interpret the backpropagation algorithm as simply a product of functions. For convenience sake, we call the partial derivative functions in the backpropagation equation, b-functions. These b-functions are the partial derivative terms $[\nabla_a C, \sigma^{'}(z^L), w^{l+1}, \delta^{l+1}, \sigma^{'}(z^l)]$ in the backpropagation equation as illustrated in equations 1 and 2.

$$\delta^L = \nabla_a C \odot \sigma^{'}(z^L) \qquad \delta^l = (((w^{l+1})^T \delta^{l+1})) \odot \sigma^{'}(z^l) \tag{1}$$

$$\frac{\partial C}{\partial b^l_j} = \delta^l_j \qquad \frac{\partial C}{\partial w^l_{jk}} = a^{l-1}_k \delta^l_j \tag{2}$$

Lillicrap et al. (2014) probe the backpropagation equations to produce a more biologically plausible credit assignment algorithm for artificial neural networks by replacing the b-function $(w^l)^T$ with a random matrix B. Similarly, we probe the backpropagation equation by replacing the activation function derivative $\sigma^{'}(z^l)$ with a b-function $g(z^l)$, as was also explored by Alber et al. (2018). Mathematically, this is equivalent to multiplying equation 1 with $\frac{g(z^l)}{\sigma^{'}(z^l)}$ when $\sigma^{'}(z^l) \neq 0$, this is illustrated in equations 3.

$$\delta^l = \delta^l \odot \frac{g(z^l)}{\sigma^{'}(z^l)},\ \sigma^{'}(z^l) \neq 0 \tag{3}$$

Within the context of stochastic gradient optimization, replacing the activation function derivative $\sigma^{'}(z^l)$ with the b-function $g(z^l)$ could be interpreted as having a learning rate $\beta(z^l) = \alpha \frac{g(z^l)}{\sigma^{'}(z^l)}$ when $\sigma^{'}(z^l) \neq 0$ as described in equation 4.

$$w^l_{i+1} = w^l_i - \frac{\partial C}{\partial w^l_i} * \alpha \frac{g(z^l)}{\sigma^{'}(z^l)},\ \sigma^{'}(z^l) \neq 0 \tag{4}$$

$\beta(z^l)$ is parameterized by z^l which infers that the learning rate $\beta(z^l)$ is an adaptive learning rate with respect to the pre-activations z^l. Similarly, as δ^l propagates backwards into $\delta^{(l-1)}$ there will be noise $\epsilon = \frac{g(z^l)}{\sigma^{'}(z^l)}$ propagated backwards into earlier layers. However, in many cases gradient noise ϵ can be beneficial for gradient optimization as described by Neelakantan et al. (2015).

When $\sigma^{'}(z^l) = 0$, this replaces the zero gradient with a custom credit value. It's ideal to have $g(z^l) = 0$ when $\sigma^{'}(z^l) = 0$ else this can be harmful to the gradient optimization step. In conclusion, the same argument applies to feedback alignment as demonstrated in equation 5, where B is the random matrix.

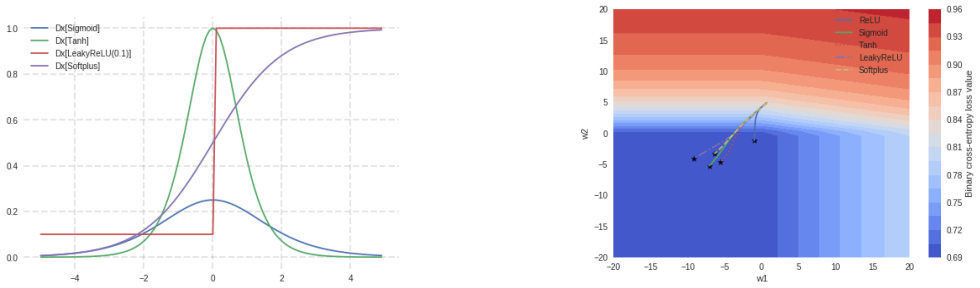
$$\delta^l = \delta^l \odot \frac{B\delta^{l+1}}{(w^{l+1})^T \delta^{l+1}},\ (w^{l+1})^T \delta^{l+1} \neq 0 \tag{5}$$

3. Experiments

3.1 Binary Classification

In this experiment, we compare various b-functions changes in a neural network on a simple binary classification task to demonstrate evidence of gradient information despite the b-function changes. The data-set for this task is generated from two Gaussian distributions, with mean 1 and -1 respectively and standard deviation of 1. Each Gaussian represents a classification class.

We replace the ReLU partial derivative in the backpropagation pass with derivatives from other existing activation functions. The neural network architecture employed for this experiment : Input ->**FC1** ->ReLU ->**FC2** ->ReLU ->**FC3** ->Softmax.



(a) (b)

Figure 1: Figure 1a are the partial derivatives to replace the ReLU derivative function in **FC2**. Figure 1b compares the gradient trajectories of each b-function in Figure 1a against vaniila backpropagation. Figure 1b shows evidence of true gradient information for b-function change.

We replaced the ReLU derivative in **FC2** and froze all but **FC2** weights to visualize the loss landscape against the rate of change of **FC2**. Our fully connected components have no bias. The network was trained on batch gradient descent on Adam (Kingma and Ba, 2014) with a learning rate of 0.1 for 100 epochs. The loss function is the cross entropy function.

Figure 1b shows evidence of the true gradient being propagated through the network despite changes to the ReLU derivative. Despite the noise that accompanies b-function changes as described in Section 2, the b-function changes still manage to navigate to local minima as shown in Figure 1b.

3.2 CIFAR10

In this experiment, we find an optimal b-function in place of the ReLU derivative in the backpropagation equation and compare it to vanilla backpropagation with a standard learning rate on an arbitrary gradient optimizer. The aim is demonstrate that learning an optimal b-function is analogous to learning an optimal learning rate.

3.2.1 Learning Framework

Learning b-functions in the backpropagation procedure using gradient optimization techniques is a complex problem. Hence, to learn these b-functions, we reduce the problem to a black-box optimization problem.

We use Bayesian optimization for black-box optimization. Following convention, we use Gaussian process (GP) (Rasmussen, 2004) priors over functions as our probabilistic prior model and Expected Improvement (EI) (Močkus, 1975) as our acquisition function for exploration.

Our learning framework consists of *three* components. A *target network* $f(\theta)$ which is the subject of the experiment, this could be a convolution neural network. A function approximator which we call the *meta-network*. The meta-network $g(\omega)$ is a neural network with weights ω and its role is to approximate a b-function. An *optimization component* to learn b-functions for the target network $f(\theta)$. Hence, learning b-functions in the backpropagation pass consists of learning weights ω such that $J(\theta)$, the cost function for $f(\theta)$, is minimized in training.

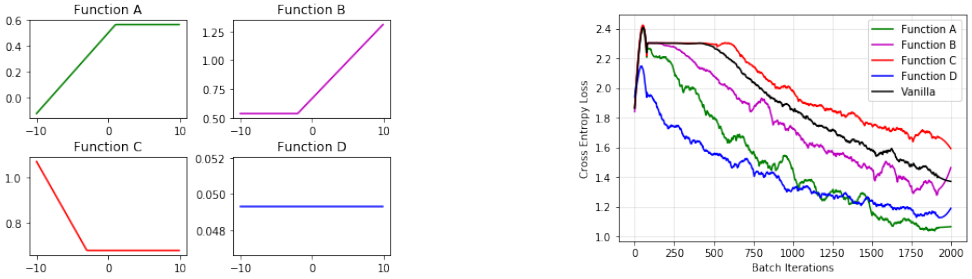
We formulate the black-box optimization problem as follows: the weights ω of the meta-network $g(\omega)$ serves as the input to the black-box. The *area under the loss curve* (AULC) when training the target network $f(\theta)$ serves as the evaluation of the black-box function.

This framework will allow us to learn an optimal b-function in the backpropagation pass to compare its effects to learning an optimal learning rate.

3.2.2 Evaluation on CIFAR 10

We evaluated our method on CIFAR-10 for a sufficiently large neural architecture, SimpleNet (Hasanpour et al., 2016). We ran SimpleNet for 5 epochs on CIFAR 10 on the Adadelta optimizer with a learning rate of 0.01. We replaced the ReLU activation function derivative for one layer in the convolution component. The learned b-functions are illustrated in Figure 2a and their performances in Figure 2b.

An implicit learning rate could relax the requirement for an explicit learning rate. A learning rate can be learned as a weight in a network via BP instead of as a hyperparameter.



(a) (b)

Figure 2: In Figure 2a, B-functions discovered for CIFAR10 on SimpleNet using our proposed method. Corresponding performances of the b-functions described in Figure 2a. Function A converges earlier than backpropagation. The meta-network architecture consisted of one input, hidden and output neuron.

References

Maximilian Alber, Irwan Bello, Barret Zoph, Pieter-Jan Kindermans, Prajit Ramachandran, and Quoc Le. Backprop evolution. *arXiv preprint arXiv:1808.02822*, 2018.

Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pages 9368–9378, 2018.

Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, and Mohammad Sabokrou. Lets keep it simple, using simple architectures to outperform deeper and more complex architectures. *arXiv preprint arXiv:1608.06037*, 2016.

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

Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

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.

J Močkus. On bayesian methods for seeking the extremum. In *Optimization Techniques IFIP Technical Conference*, pages 400–404. Springer, 1975.

Arvind Neelakantan, Luke Vilnis, Quoc V Le, Ilya Sutskever, Lukasz Kaiser, Karol Kurach, and James Martens. Adding gradient noise improves learning for very deep networks. *arXiv preprint arXiv:1511.06807*, 2015.

Carl Edward Rasmussen. Gaussian processes in machine learning. In *Advanced lectures on machine learning*, pages 63–71. Springer, 2004.

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

Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61: 85–117, 2015.