DIRECT FEEDBACK ALIGNMENT PROVIDES LEARNING IN DEEP NEURAL NETWORKS

Marco Lupia and Francesca Fraioli

ABSTRACT

In this paper we reported four Deep Neural Network training methods, referencing the work of [Lillicrap (2014)] and [Nøkland (2016)]. We explain how they work in detail, what are the advantages and disadvantages and why they were introduced. We then show our implementation of these methods in python on both DNN and CNN and we examine the results.

KEYWORDS Neural Networks; Nokland; Back-Propagation; Direct Feedback Alignment

Introduction

A deep neural network (DNN) is an artificial neural network, with multiple layers between the input and output layers, where a neural network is a technology built to simulate the activity of the human brain specifically, pattern recognition and the passage of input through various layers of simulated neural connections. Deep learning is the core of the machine learning and it has been utilized for many applications such as machine translation, speech recognition and object classification.
One of the DNN training methods mainly used for the last 30 years is the back-propagation algorithm. This method is intuitive and easy to implement in both software and hardware. It computes, layer by layer, a gradient that is needed in the calculation of the weights to be used in the network. It is commonly used by the gradient descent based training method which follows the steepest gradient to find the optimum weights. The back-propagation algorithm is not biologically plausible because it requires symmetric weights and separate phases for inference and learning. Learning signals are not local, but have to be propagated backward, layer-by-layer, from the output units. Hence the error derivative has to be transported as a second signal through the network, to do that the derivative of the non-linearities have to be known. So BP based DNN training suffers from the overfitting problem and it is too slow to be converged and shows low accuracy. For these reasons, we explore the possibility of learning networks without back-propagation.
In recent years, many new methods have been developed, like feedback-alignment (FA) and direct-feedback-alignment (DFA), that replaced the traditional back-propagation update with a random feedback update.
The FA method [Lillicrap (2014)], was proposed based on the *gradient descent* methodology but without following the steepest gradient, it shows that the weights used for propagating the error backward don't have to be symmetric with the weights used for propagation the activation forward. Thanks to the the network that learns how to make the feedback useful, the feedback weights work well.
The FA's principle is used for training hidden layers independently from the rest of the network, and from a zero initial condition [Nøkland (2016)].
This method demonstrated that one can swap out the weight matrices used in a backward pass with fixed random matrices and still achieve comparable performance to standard back-propagation. It also proves that for a linear network with a single hidden layer trained with FA, the feedforward weight matrix will approach the pseudo-inverse of the fixed random weight matrix. So the gradient updates will become correlated with the correct gradient update from back-propagation.[Justin Gilmer (2016)]
The feedback weights used to back-propagate the gradient do not have to be symmetric with the feed-forward weights.
The error is propagated through fixed random feedback connections directly from the output layer to each hidden layer. In order to reduce the error the network learns how to use fixed random feedback wights.
The FA is also known to converge slowly, and shows worse performance compared with the previous BP.
Another method designed to replace BP is direct-feedback-alignment (DFA), proposed in [Nøkland (2016)] and developed by getting the idea from the FA. Although the FA method propagates errors from the last layer back to the first layer step-by-step, the DFA propagates errors from the last layer directly to each layer. This approach allows the parallel processing. Since the errors of every layer are generated independently in DFA, it's possible to calculate immediately each layer's gradient if the DNN inference is finished.
Being the number of neurons in the last layer fewer than the

prior layer, the size of the feedback weight becomes smaller than the BP, so the number of required computation is reduced in the DFA. At the same time DFA suffers from accuracy degradation problem.
In order to lead to more efficient training algorithm, DFA allows for updates to internal matrices to be computed in parallel.
Experiments show that the test performance on MNIST dataset is almost as good as those obtained with back-propagation for fully connected networks.

Back-propagation

Back-propagation (BP, Rumelhart (1986)) was introduced as a new learning procedure for networks of neurone-like units. The procedure repeatedly adjusts the weights of the connections in the network so as to minimize a measure of the difference between the actual output vector of the net and the desired output vector. As a result of the weight adjustments, internal 'hidden' units which are not part of the input or output come to represent important features of the task domain, and the regularities in the task are captured by the interactions of these units.

The BP is the gradient descent based training method which follows the steepest gradient to find the optimum weights.

It's calculated starting from the error of the output and according to its partial differential equation, ie the error of each layer is strictly the weight of that layer and the next layer it depends on the error in (output direction).

Even though the BP shows the outstanding performance in deep neural network (DNN) training, BP based training suffers from the overfitting problem. Overfitting refers to a model that models the "training data" too well. Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data. Since the BP easily sinks into a local minimum, we need a large scale of the dataset to avoid the overfitting. In addition, we use various data augmentation techniques such as flipping and cropping. If the DNN training is done in the limited resources and dataset, BP based DNN training is too slow to be converged and shows low accuracy.

According to [Nøkland (2016)], the BP algorithm is not biologically plausible for several reasons. First, it requires symmetric weights. Second, it requires separate phases for inference and learning. Third, the learning signals are not local, but have to be propagated backward, layer-by-layer, from the output units. This requires that the error derivative has to be transported as a second signal through the network. To transport this signal, the derivative of the non-linearities have to be known.

Feedback Alignment

Feedback Alignment (FA, Lillicrap (2014)), was proposed based on the gradient descent methodology, but without following the steepest gradient.

It is known to converge slowly, and shows worse performance compared with the previous BP.

It demonstrated that one can swap out the weight matrices used in a backward pass with fixed random matrices and still achieve comparable performance to standard backpropagation. Moreover, for a linear network with a single hidden layer trained with FA, the feedforward weight matrix will approach the pseudo-inverse of the fixed random weight matrix. Consequently, the gradient updates will become correlated with the correct gradient update from backpropagation. [Justin Gilmer (2016)]

The FA pre-defines a feedback weight before starting the training, and the weight is determined by the random values.

The feedback weights used to back-propagate the gradient do not have to be symmetric with the feed-forward weights. The network learns how to use fixed random feedback weights in order to reduce the error. Essentially, the network learns how to learn.[Nøkland (2016)])

Layer weights are not used when calculating the error. Normally, weights are initialized at random, but learning stability is achieved by reducing their dependence.

Direct Feedback Alignment

Direct feedback alignment (DFA, Nøkland (2016)) was developed by getting the idea from the FA. Although the FA propagates errors from the last layer back to the first layer step-by-step, the DFA propagates errors from the last layer directly to each layer. This approach gives the opportunity of the parallel processing. Since the errors of every layer are generated independently in DFA, we can immediately calculate each layer's gradient if the DNN inference is finished. Moreover, the number of required computation is reduced in the DFA. This is because the number of neurons in the last layer is usually fewer than the prior layer, so the size of the feedback weight becomes smaller than the BP. In spite of these advantages, the DFA suffers from accuracy degradation problem. This method differs from feedback-alignment in that the error signal from the output is directly propagated to each internal layer via a random matrix. [Justin Gilmer (2016)] It uses the error of the final layer instead of the error of the next layer, in addition to not using the weight of the layer as FA.

Network architecture

We will take a look at four methods: Back-propagation (BP), Feedback Alignment (FA), and two methods proposed by Nøkland himself: Direct Feedback Alignment (DFA) and Indirect Feedback Alignment (IFA).

The Neural Network presented in the original paper [Nøkland (2016)] is just standard 3 layer feed forward network with `Tanh()` activation function.

$$\begin{aligned} a_1 &= W_1 x + b_1, h_1 = f(a_1) \\ a_2 &= W_2 h_1 + b_2, h_2 = f(a_2) \\ a_y &= W_3 h_2 + b_3, \hat{y} = f_y(a_y) \end{aligned}$$

$f()$ $\rightarrow$ Tanh() activation function.
W $\rightarrow$ Weights for each individual layers.
b $\rightarrow$ Bias Variables, not included for simplicity.

In the paper, four methods of training a neural network are introduced, as shown in Figure (1). The grey arrows is the forward propagation, while the black arrows is the backward propagation or the learning process.

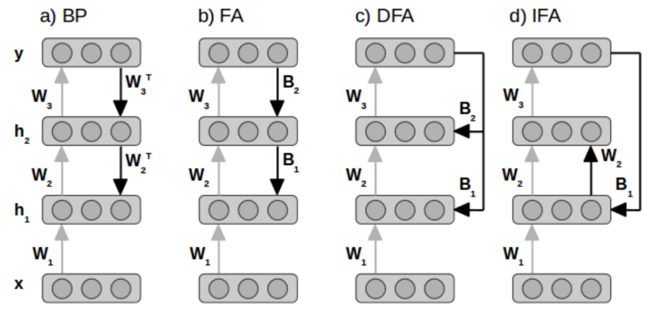
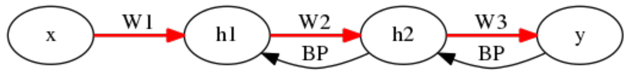


Figure 1 Comparison between Back Propagation (BP), Feedback Alignment (FA), Direct Feedback Alignment (DFA), and Indirect Feedback Alignment (IFA)

a) Standard Back-propagation



The gradient at the output layer is:

$$e = \delta a_y = \frac{\partial J}{\partial a_y} = \hat{y} - y$$

$e \rightarrow$ Derivative Respect to most outer layer

The gradients for hidden layers are:

$$\delta a_2 = \frac{\partial J}{\partial a_2} = \left(W_3^T e\right) \odot f'(a_2), \delta a_1 = \frac{\partial J}{\partial a_1} = \left(W_2^T \delta a_2\right) \odot f'(a_1)$$

$\delta a_2 \rightarrow$ Derivative Respect to Second Layer
$\delta a_1 \rightarrow$ Derivative Respect to First Layer

The above equations represents the standard back propagation equations. In order to get the derivative respect to the first layer we not only need W_3 but also W_2 to propagate the error backwards. $\odot$ is a binary operator that takes the element-by-element product of matrices or vectors of the same type.

b) Feedback Alignment

$$\delta a_2 = (B_2 e) \odot f' a_2, \ \delta a_1 = (B_1 \delta a_2) f' \odot a_1$$

$\delta a_2 \rightarrow$ Derivative Respect to Second Layer
$\delta a_1 \rightarrow$ Derivative Respect to First Layer
$B_i \rightarrow$ New variable B to replace w

There is one difference from standard back propagation: to calculate the derivative respect to the first layer, two the new variables B_1 and B_2 are introduced, instead of weights W_3 and W_2. According to [Nøkland (2016)] B_i is a fixed random weight matrix with appropriate dimension, while according to [Lillicrap (2014)] B_i was drawn from uniform distribution ranging from -0.5 to 0.5.

c) Direct Feedback Alignment

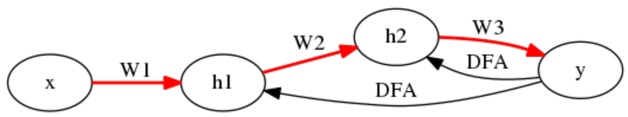


Figure 2 The Direct version affects the error of the output layer directly, not the error of the next layer

$$\delta a_2 = (B_2 e) \odot f' a_2, \ \delta a_1 = (B_1 e) f' \odot a_1$$

The symbol e stands for derivative of the cost function; in other words derivative respect to most outer layer. We use that derivative to update the weights rather than back propagating all the way through.

d) Indirect Feedback Alignment

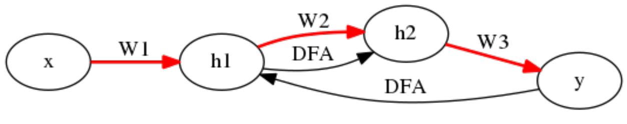


Figure 3 The Indirect version designs back propagation by making the error dependency freely by itself

$$\delta a_2 = (W_2 \delta a_1) \odot f' a_2, \ \delta a_1 = (B_1 e) f' \odot a_1$$

For the first layer we are going to use direct feedback alignment to update the weights and for the second layer we are going to use the derivative from the first layer to update the weights.

Implementation

Development enviroment

The technical specifications of our desktop machine are:

- OS: Ubuntu 16.04 LTS x64
- Processor: Intel Core i7-8700K CPU
- Graphics: Nvidia GeForce GTX 1080
- RAM: 8 GB
- Disk size: 500 GB

Direct Feedback Alignment VS Back-propagation

Our Python implementation of Direct Feedback Alignment shows how a neural network trained with DFA achieves very similar results to one trained with backpropagation. We used MNIST handwritten digit database [Yann LeCun (2018)] that has a training set of 60.000 examples, and a test set of 10.000 examples. For brevity, only the important parts are shown in this paper, the full implementation is available on Pastebin [1].

```
from multiprocessing import freeze_support
import matplotlib.pyplot as plt
import numpy as np
import scipy.ndimage.filters
import scipy.interpolate
import dataset.mnist_dataset
from network import *

if __name__ == '__main__':
    freeze_support()
    colors = [('blue', 'blue'), ('red', 'red'), (
        'green', 'green'), ('orange', 'orange'),
        ('black', 'black')]
    depths = [10, 15, 20, 50, 100]
    iterations = ([4] * 5)
    data = dataset.mnist_dataset.load('dataset/
        mnist')
    statistics = []

    for depth, num_passes in zip(depths,
        iterations):
```

[1] https://pastebin.com/dkd9VKt4

```
        layers = [ConvToFullyConnected()] + \
                 [FullyConnected(size=240,
                     activation=activation.tanh)
                     for _ in range(depth)] + \
                 [FullyConnected(size=10,
                     activation=None, last_layer=
                     True)]

        """ DFA """
        model = Model(
            layers=layers,
            num_classes=10,
            optimizer=GDMomentumOptimizer(lr=1e
                -3, mu=0.9),
        )
        stats_dfa = model.train(data_set=data,
            method='dfa', num_passes=num_passes,
            batch_size=64)
        loss, accuracy = model.cost(*data.
            test_set())

        """ BP """
        model = Model(
            layers=layers,
            num_classes=10,
            optimizer=GDMomentumOptimizer(lr=1e
                -2, mu=0.9)
        )
        stats_bp = model.train(data_set=data,
            method='bp', num_passes=num_passes,
            batch_size=64)
        loss, accuracy = model.cost(*data.
            test_set())

        statistics.append((stats_dfa, stats_bp))

    #draw plot
```

Results

Figures (6) and (7) show the Average Cost for Training Images and Average Accuracy for Training Images.

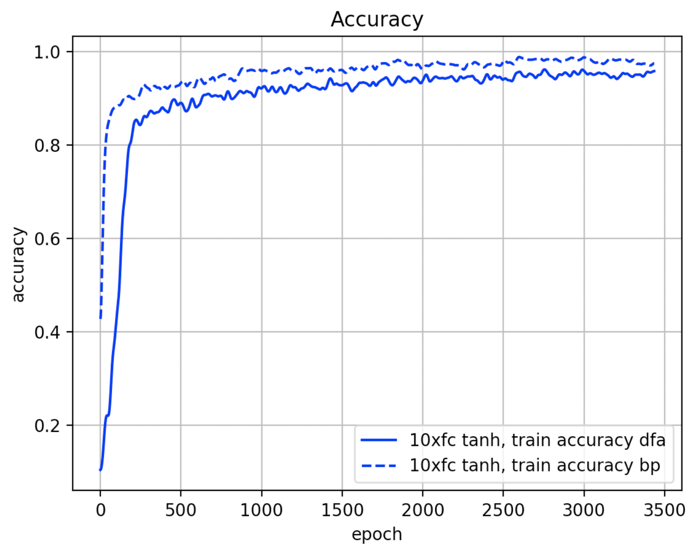


Figure 4 Accuracy

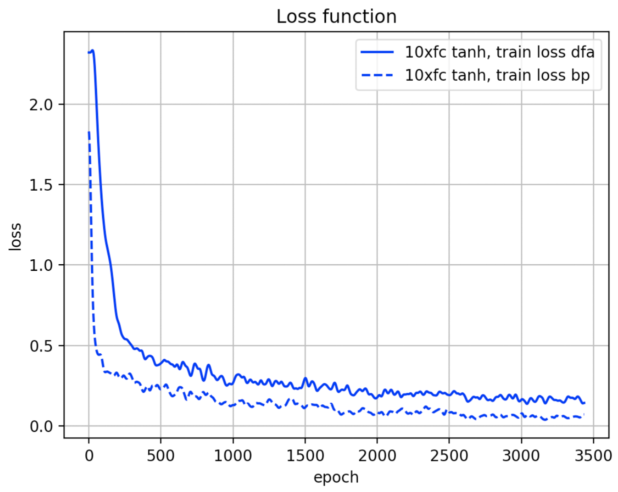


Figure 5 Loss Function

Listing 1 Training results

```
Result:
------------------------------------
loss on test set: 0.1299619719312314
accuracy on test set: 0.9623
```

From Figures (6) and (7) it is clear that the training accuracy and the computational efficiency look almost similar, but the results of DFA were slightly worse than BP.
With DFA, you can just use the gradient from the last layer to train all layers in Neural Networks. Each of your layers do not need to depends on gradient from the layers behind of them. So, the training process doesn't have to progress layer by layer anymore.

Discussion

The positive aspects of Feedback Alignment are:

- FA is a thorough exploration using asymmetric connections
- FA guaranteed convergence under some assumption of the network
- Shown in experiments that we can train very deep 100-layer network with DFA, whereas BP and FA suffers from gradient vanishing
- DFA replaces the reciprocal feedback assumption with a single feedback layer
- The relaxed version of DFA, IFA, can be viewed as skip connections on the feedback path, which opens up more freedom on the actual form of feedback connections, compared to the original FA
- One of the first exploration of error-driven learning using directly connected feedback path
- Can be used to send error signals skipping non-differentiable layers

On the other hand, negative aspects are:

- DFA assumes that there is a global feedback path, which may be biologically implausible since the single feedback layer need to travel long physical distance
- Both FA and DFA leverages the principle of feedback alignment to drive the error signal. Due to the alignment stage,

a layer cannot learn before its upper layers are roughly "aligned". This could also be biologically implausible
- FA and DFA are presented as less powerful optimization methods. A more impactful yet biologically plausible direction could be replace BP with a learning algorithm with better generalization performance
- FA and DFA rely on synchronous updates to update the weights at a layer, we need to fix the activation of the layer below
- Theoretical results on the negative descending direction is weak

Convolutional Neural Networks

In deep learning a convolutional neural network (CNN) is a class of deep neural networks (DNN) most commonly applied to analyze visual imagery, but also for many other tasks, such as speech recognition, natural language processing, and more.
The CNNs are pretty deep and have dozens of layers at least.
It consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of convolutional layers and fully-connected layers (FC).

The two different kinds of layers have different roles. For example in object classification, convolutional layers are considered as the feature extractor by using the characteristics of the convolution computation. In contrast, FC layers receive the result of the feature extractor and judge what the object is. However, in the initial iterations of the BP based CNN training, training the convolutional layers can be ambiguous because the FC layers cannot be considered as a good object classifier. During some iterations of BP, the weight of the FC layers will be changed and the convolutional layers should be adaptive to the changed FC layers. In other words, convolutional layers can be confused if the propagated error's domain is changed for every iteration.

If the error is propagated from the last FC layer to convolutional layer through the constant domain shifting, the training of the convolutional layer can be more stable than conventional BP. From this motivation, we use DFA instead of BP in the FC layers. As shown in Figure 2, the network maintains the BP in the convolutional layers but adopts the DFA for the FC layers. Since there is no data dependency among the FC layers, the DFA can directly propagate the error to the FC1. Since the DFA uses fixed feedback weight for error propagation, convolutional layers do not have to be adaptive to various errors which are derived from the different domain. In this method, the error propagation in the convolutional layers can be started even though the errors are not propagated for the remained FC layers. Therefore, the error propagation of both the convolutional layers and the FC layers can be computed in parallel right after DFA is done for the first FC layer.

In this paper we show that for the convolutional neural network, the difference between DFA and BP is clearest, as it happened for the deep neural network also in this case the training accuracy and the computational efficiency of DFA are worse than BP.

As we discussed above one of the disadvantages of DFA is the accuracy degradation problem, this one becomes more serious when DFA is applied to the CNN case.

In our Python implementation of Direct Feedback Alignment versus Back propagation in Convolutional Neural networks we show how a neural network trained with DFA achieves poor results compared to one trained with backpropagation.

We used CIFAR-10 dataset [Krizhevsky (2013)] which consists of 60000 32x32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.

For brevity, only the important parts are shown in this paper, the full implementation is available on Pastebin [2].

```
from multiprocessing import freeze_support
import matplotlib.pyplot as plt
import numpy as np
import scipy.interpolate
import scipy.ndimage.filters
import dataset.cifar10_dataset
from network import *

if __name__ == '__main__':
    freeze_support()
    num_iteration = 30
    data = dataset.cifar10_dataset.load()
    layers = [
        # MaxPool(size=2, stride=2),
        Convolution((8, 3, 4, 4), stride=2,
            padding=2, dropout_rate=0, activation
            =activation.tanh),
        #MaxPool(size=2, stride=2),
        Convolution((16, 8, 3, 3), stride=2,
            padding=1, dropout_rate=0, activation
            =activation.tanh),
        #MaxPool(size=2, stride=2),
        Convolution((32, 16, 3, 3), stride=2,
            padding=1, dropout_rate=0, activation
            =activation.tanh),
        #MaxPool(size=2, stride=2),
        ConvToFullyConnected(),
        FullyConnected(size=64, activation=
            activation.tanh),
        FullyConnected(size=10, activation=None,
            last_layer=True)
    ]

    #
        ---------------------------------------------------

    # Train with BP
    #
        ---------------------------------------------------

    model = Model(
        layers=layers,
        num_classes=10,
        optimizer=GDMomentumOptimizer(lr=1e-2, mu
            =0.9),
        lr_decay=0.5,
        lr_decay_interval=7
    )
    stats_bp = model.train(data_set=data, method=
        'bp', num_passes=num_iteration,
        batch_size=64)
    loss, accuracy = model.cost(*data.test_set())
    #
        ---------------------------------------------------

    # Train with DFA
    #
        ---------------------------------------------------

    model = Model(
        layers=layers,
        num_classes=10,
        optimizer=GDMomentumOptimizer(lr=1e-2, mu
            =0.9),
```

[2] https://pastebin.com/K8y6xLNW

```
        lr_decay=0.5,
        lr_decay_interval=7
    )
    stats_dfa = model.train(data_set=data, method
        ='dfa', num_passes=num_iteration,
        batch_size=64)
    loss, accuracy = model.cost(*data.test_set())

    # Draw plot
```

Results

The images above show the Average Cost for Training Images and Average Accuracy for Training Images.

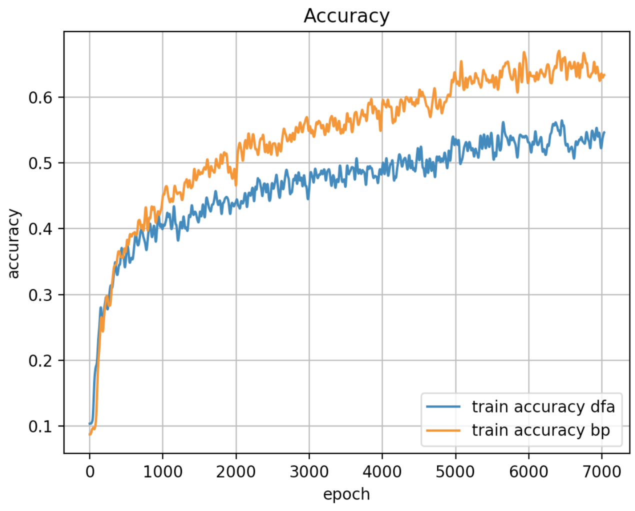


Figure 6 Accuracy

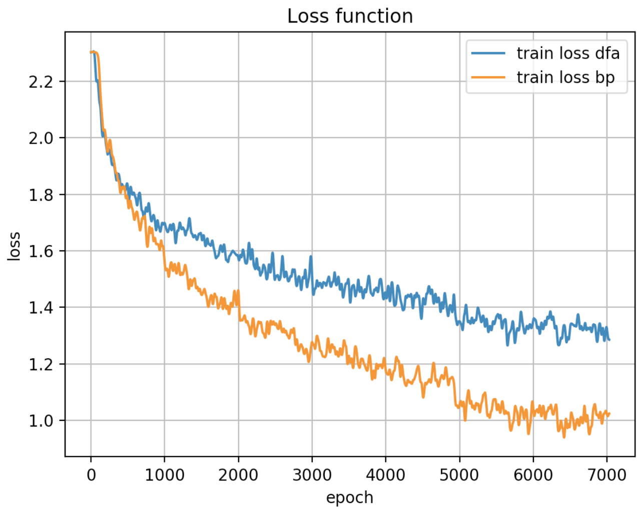


Figure 7 Loss Function

Listing 2 Training results

```
Result:
------------------------------------
loss on test set: 1.420182137961529
accuracy on test set: 0.5014
```

From the images above we can see that backpropagation still outperforms direct feedback alignment in convolutional neural networks.

Conclusion

Nøkland (2016) proved that we can achieve zero training error with Feedback Alignment under the following assumption:

- Network is linear with one hidden layer
- Input data have zero mean and unit variance
- The feedback weight matrix has Moore-Penrose pseudo-inverse
- The forward weights are initialized to zero
- The output layer weights are adapted

However, it is unclear how the training error can approach zero with several non-linear layers. This paper gives new theoretical insight with less assumption of the network topology, under the assumption of constant update direction. This paper generalizes previous FA results by considering two consecutive layers. For any layer k and $k+1$, δh_k will end up within 90 degrees of cosine angle with the back-propagated gradient c_k, and δh_{k+1} with c_{k+1}. Although we assume δh_k is constant for all data points, it can still a function of the parameters.

Compared with the BP, DFA showed similar training performance for the multi-layer perceptron (MLP) structure. Moreover, DFA requires fewer computations because the number of neurons in the last layer is generally smaller than the number of neurons in the intermediate layers. In summary, DFA dramatically degrades the accuracy when it applied in the CNN training compared with the accuracy applied in the DNN training.

Bibliography

Justin Gilmer, C. R., 2016 Explaining the learning dynamics of direct feedback alignment. https://openreview.net/forum?id=HkXKUTVFl.

Krizhevsky, A., 2013 Cifar dataset. https://www.cs.toronto.edu/~kriz/cifar.html.

Lillicrap, T., 2014 Random feedback weights support learning in deep neural networks. https://arxiv.org/abs/1411.0247.

Nøkland, A., 2016 Direct feedback alignment provides learning in deep neural networks. https://arxiv.org/abs/1609.01596.

Rumelhart, D. E., 1986 Learning representations by back-propagating errors. https://www.nature.com/articles/323533a0.

Yann LeCun, C. B., Corinna Cortes, 2018 Mnist handwritten digit database. http://yann.lecun.com/exdb/mnist/.