ASSIGNMENT

ON

Forward Forward Algorithm By Geoffrey Hinton

April 28th, 2023

Course No. : **BITS F464**

Course Title : **Machine Learning**

Submitted to:

Prof. Pratik Narang

Dept. of Computer Science And Information Systems

BITS PILANI

Submitted By:

Ritvik Mittal - 2021A8PS2545P - f20212545@pilani.bits-pilani.ac.in

INTRODUCTION:

In this report, I have compared the forward forward algorithm by Geoffrey Hinton against backpropagation algorithm. Backpropagation is a supervised learning algorithm used for training artificial neural networks. The backpropagation algorithm computes the gradients of the weights and biases in a neural network using the chain rule of calculus. A forward pass through the network is used to determine the output, and then a backward pass through the network is used to determine the error and update the weights.

Each layer of the network receives input during the forward pass, and each layer's output is computed using its own weights and biases. The last layer's output is compared to what was anticipated, and the error is determined using a loss function. Using the chain rule of calculus, the error is transmitted backward via the network's layers during the backward pass. Based on the error and output of the preceding layer, the weights' and biases' gradients are computed. Using an optimisation approach like stochastic gradient descent, these gradients are then utilized to update the weights and biases.

There are certain drawbacks to backpropagation, such as the vanishing or expanding gradient problem. The network may converge slowly or not at all when this happens because the gradients grow too small or huge as they move through the layers of the network. The fact that backpropagation requires precise knowledge of the calculation done in the forward pass in order to compute the right derivatives is a severe drawback. Backpropagation becomes impossible if a black box is added to the forward pass unless a differentiable model of the black box is learned.

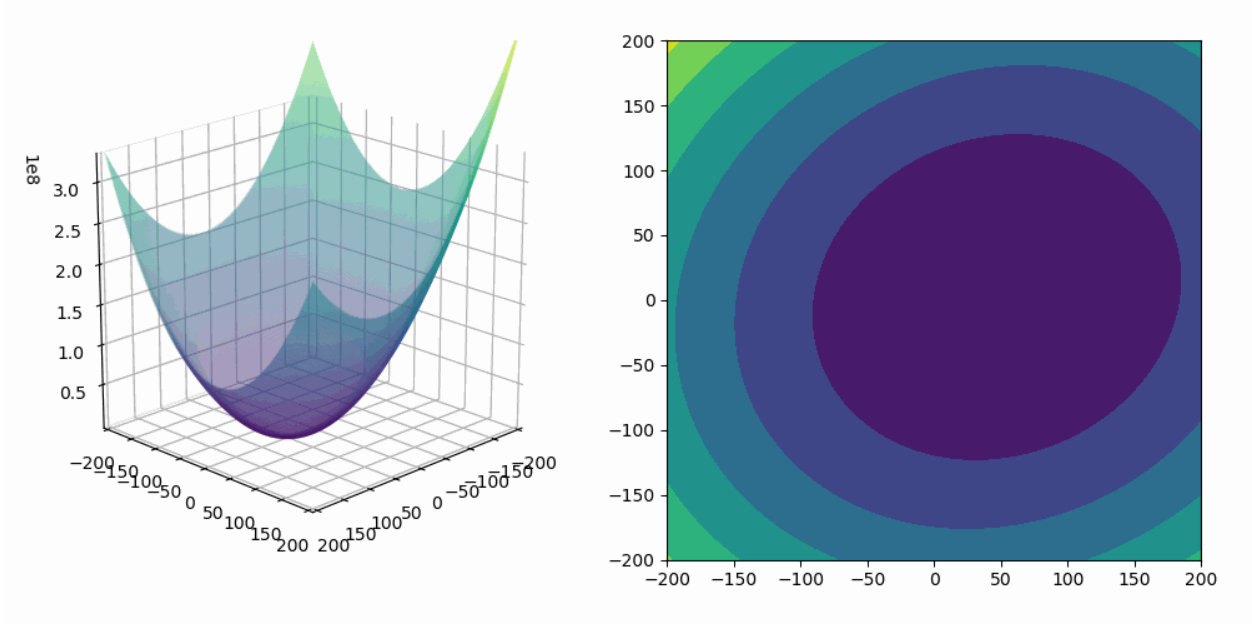
We shall see that the absence of the requirement to backpropagate through the black box has no impact whatsoever on the learning process for the **forward-forward algorithm**.

Though it is slower than backpropagation, the **Forward-Forward algorithm (FF)** offers the benefit that it can be applied even when the specifics of the forward calculation are unknown. Additionally, it has the benefit of enabling learning as sequential input is being pipelined through a neural network, without ever storing neural processes or pausing to propagate error derivatives.

Forward Forward algorithm can be used in place of back-propagation instead of resorting to reinforcement learning.

ALGORITHM DESCRIPTION:

Backpropagation includes two passes - a forward pass and a backward pass. Initially weights and biases are randomly assigned for each layer and cost is calculated. Then in backward pass, the weights and biases are adjusted using functions like gradient descent or stochastic gradient descent to minimize the cost. The learning rate determines the step size of the weight and bias updates and should be chosen carefully to ensure the algorithm converges to a good solution. The algorithm calculates how much each weight and bias contributed to the error using partial derivatives. These contributions are then used to update the weights and biases, thereby improving the performance of the network.

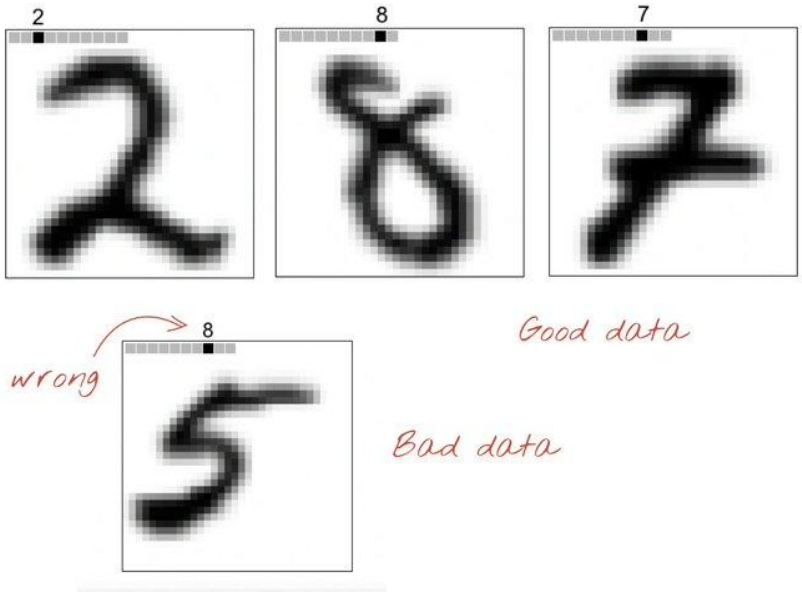


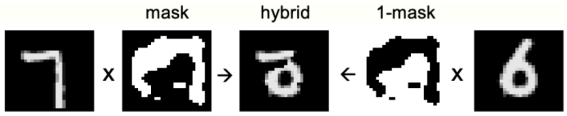
Red- Normal Gradient Descent, Blue - Stochastic Gradient Descent

In the **forward forward algorithm**, the forward and backward passes of backpropagation are swapped out for two forward passes that operate identically but on different sets of data and with different goals. The positive pass uses actual data and modifies the weights to improve each hidden layer's quality. The negative pass modifies the weights to reduce the goodness in each

hidden layer while operating on "negative data". This report focuses on two different measures of goodness: the sum of the squared neural activities and the negative sum of the squared activities.

For this report, I have calculated negative data by merging the images in the training set of the datasets with incorrect labels. We can also create negative data by creating a mask containing fairly large regions of ones and zeros. We then create hybrid images for the negative data by adding together one digit image times the mask and a different digit image times the reverse of the mask. Masks like this can be created by starting with a random bit image and then repeatedly blurring the image with a filter of the form [1/4, 1/2, 1/4] in both the horizontal and vertical directions. After repeated blurring, the image is then thresholded at 0.5.





We can also use "video" input, for instance a static MNIST image that is replayed for each time-frame. The pixel picture is on the bottom layer, while the digit class is represented by a one-of-N representation on the top layer.

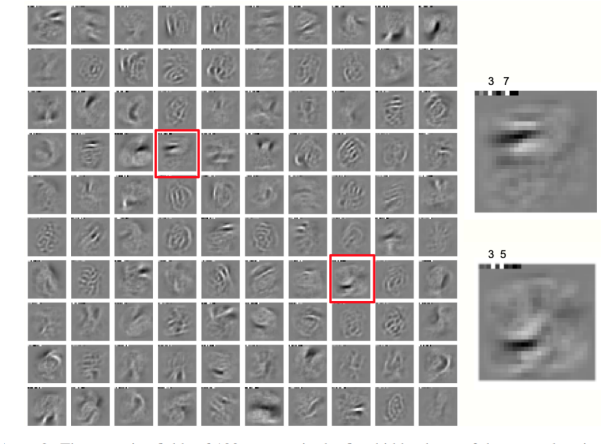


Fig a

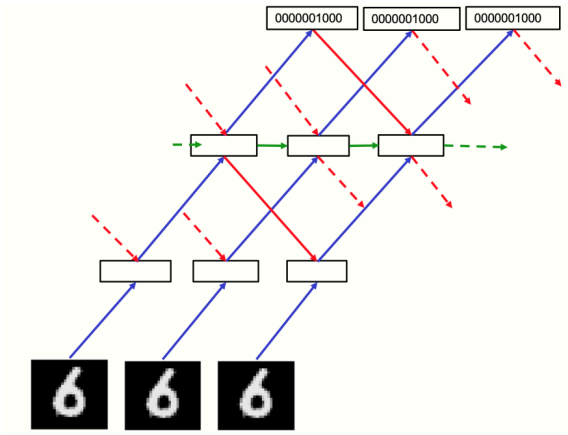
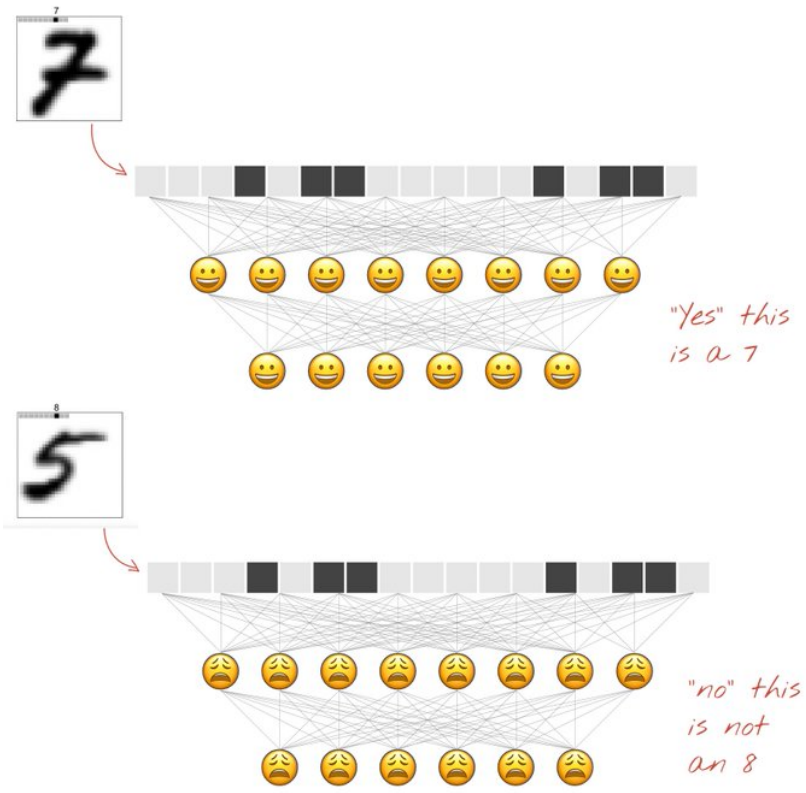


Fig b

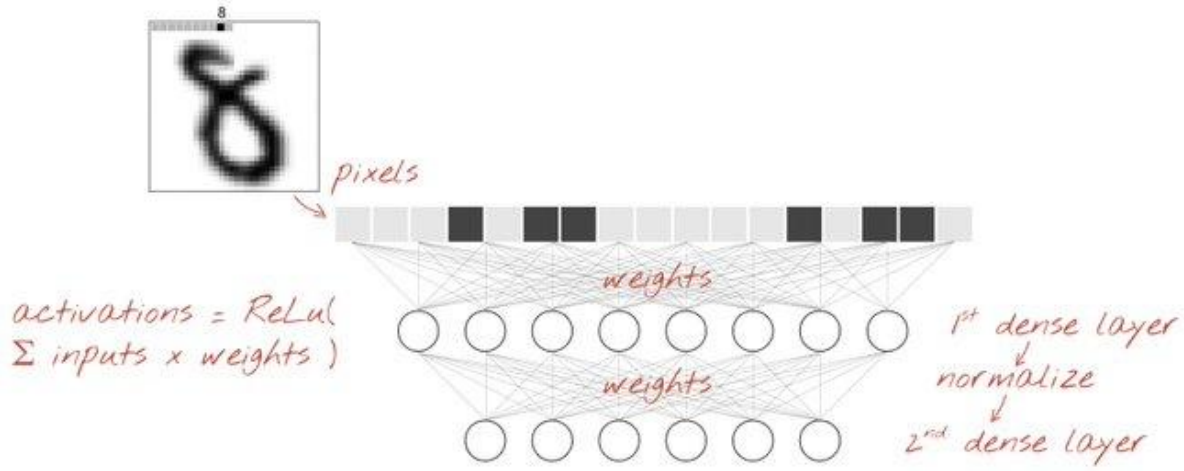
Fig a: The receptive fields of 100 neurons in the first hidden layer of the network trained on jittered MNIST. The class labels are represented in the first 10 pixels of each image.

Fig b: The recurrent network used to process video.

The sum of the squares of the activity of the rectified linear neurons inside a layer is the goodness function for that layer. The goal of learning is to raise goodness much above a predetermined threshold for actual data and to significantly lower it for actual data that is negative. Firstly we have to correctly classify input vectors as positive data when Probability(positive) = σ ((Sum over j (y(j)^2)) − θ) is greater than 0, where y(j) is the activity of hidden unit j before layer normalization and θ is the threshold, else to classify it as negative data and then we maximize the sum of squared RELu activations in a layer (goodness of this algorithm) on the positive data while minimizing it on negative data within one layer. Cost function which I have used for each layer(l) = ln(1 + exp((θ - (weight of lth layer)*(sum of squared RELu activations for positive points) +(weight of lth layer)*(sum of squared RELu activations for negative points)-θ)).



If the activities of the first hidden layer are then used as input to the second hidden layer, it is trivial to distinguish positive from negative data by simply using the length of the activity vector in the first hidden layer. There is no need to learn any new features. To prevent this, FF normalizes the length of the hidden vector before using it as input to the next layer.



DATA-SETS DESCRIPTION:

MNIST:

The MNIST dataset is a popular benchmark dataset in the field of machine learning and computer vision. It stands for "Modified National Institute of Standards and Technology" and is made up of 70,000 handwritten pictures of digits, each of which is 28 pixels by 28 pixels. There are two parts to the dataset: a training set with 60,000 pictures and a test set with 10,000 images.

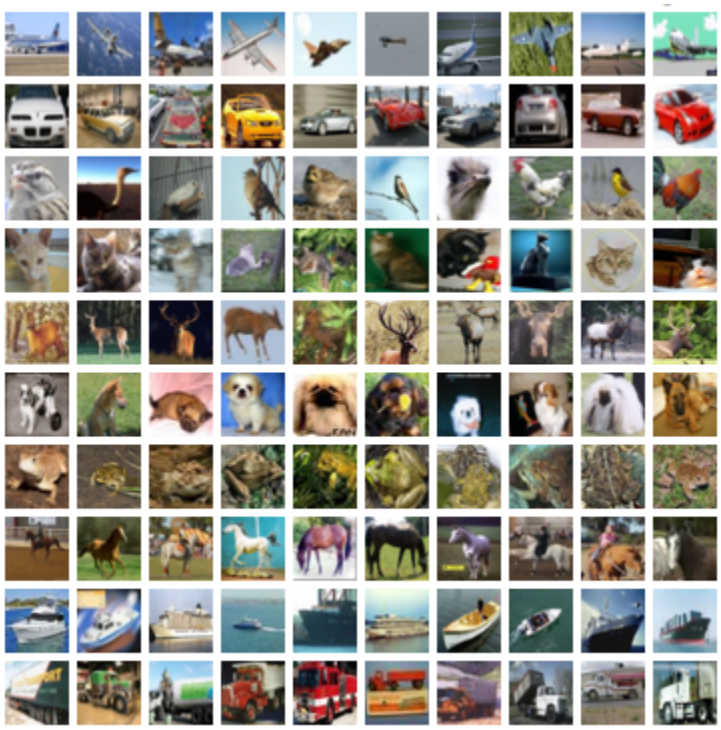
The MNIST dataset has been used in a lot of study to do things like classifying images, recognising objects, and reading handwriting. It is now the usual way to measure how well new machine learning algorithms work in these areas. In particular, it has been used to measure how well deep learning methods, such as convolutional neural networks (CNNs), work.

CIFAR-10:

CIFAR-10 is a frequently used benchmark dataset in machine learning and computer vision for image classification applications. The acronym "CIFAR" refers to the "Canadian Institute for Advanced Research," which funded the dataset's production. It comprises 60,000 32x32 color images divided into ten groups, each containing 6,000 images. There are 50,000 training photos and 10,000 test images in the dataset.

The CIFAR-10 dataset has ten classes: airplanes, autos, birds, cats, deer, dogs, frogs, horses, ships, and trucks. Because the photos in the dataset have been labeled with the appropriate class, it is ideal for supervised learning tasks.

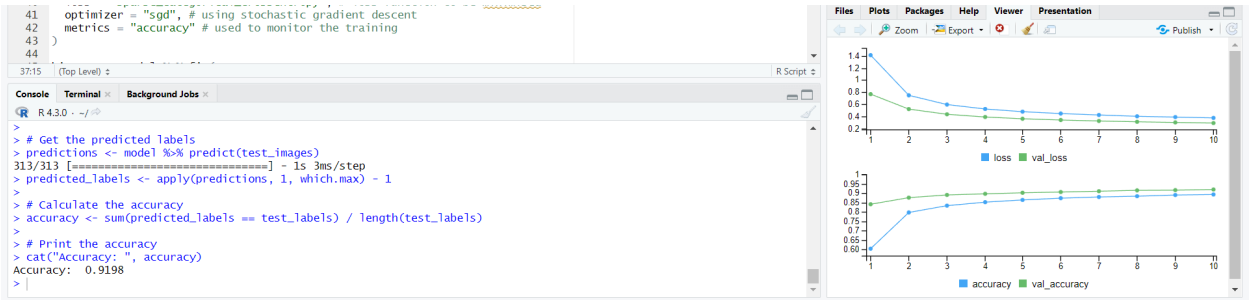
Because the images in the CIFAR-10 dataset are in color and the objects are more complex and varied, it is more difficult than the MNIST dataset. The dataset has become a standard for assessing the performance of image categorization systems, including deep learning approaches like convolutional neural networks (CNNs).



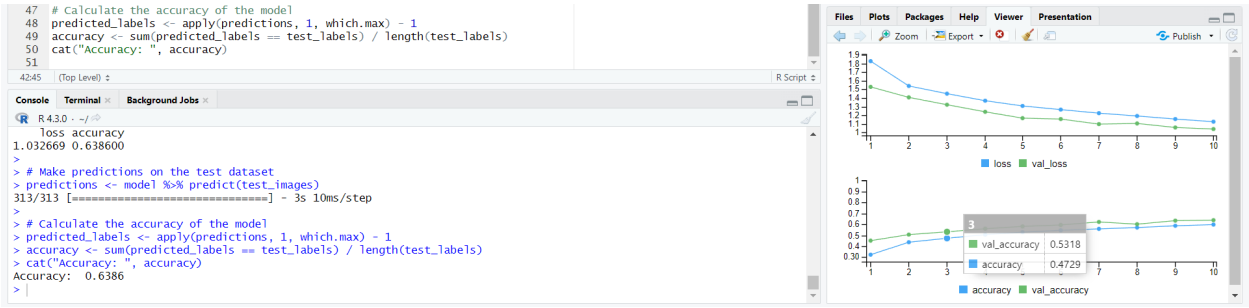
RESULTS:

BACKPROPAGATION:

By testing on MNIST data with one hidden layer of 128 neurons and RELu activation function and Stochastic gradient descent, I was getting 91.98% testing accuracy.



CIFAR-10 with two convolutional layers and two max pooling layers and two fully connected layers, I got 63.68% testing accuracy.



MNIST:

Training a network with four hidden layers, each with 2000 ReLUs, for 100 epochs resulted in a test error rate of 1.37% when using the normalized activity vectors of the last three hidden layers as input to a softmax for label prediction, according to Geoffrey Hinton's research. However, including the first hidden layer in the linear classifier's input reduced test performance. The performance was enhanced by employing local receptive fields (without weight-sharing) instead of fully connected layers. Only one architecture was tested, and it achieved a test error of 1.16% after 60 epochs of training. Furthermore, "peer normalization" of the hidden activity was used in the study to avoid any of the hidden units from becoming overly active or permanently switched

off.

It is quite simple to illustrate what the first hidden layer learns if we replace the first 10 pixels with a one of N representation of the label. After 60 epochs, a network with four hidden layers, each with 2000 ReLUs and full connection between them, had 1.36% test errors on MNIST. Backpropagation takes roughly 20 epochs to get comparable test performance. Doubling the learning rate of FF and training for 40 epochs rather than 60 results in a somewhat worse test error of 1.46% rather than 1.36%.

Using Video Input:

In a preliminary experiment, the recurrent net was run for ten time steps, with the even layers being updated based on the normalized activities in the odd layers, and the odd layers being updated based on the new normalized activities in the even layers.

CIFAR-10:

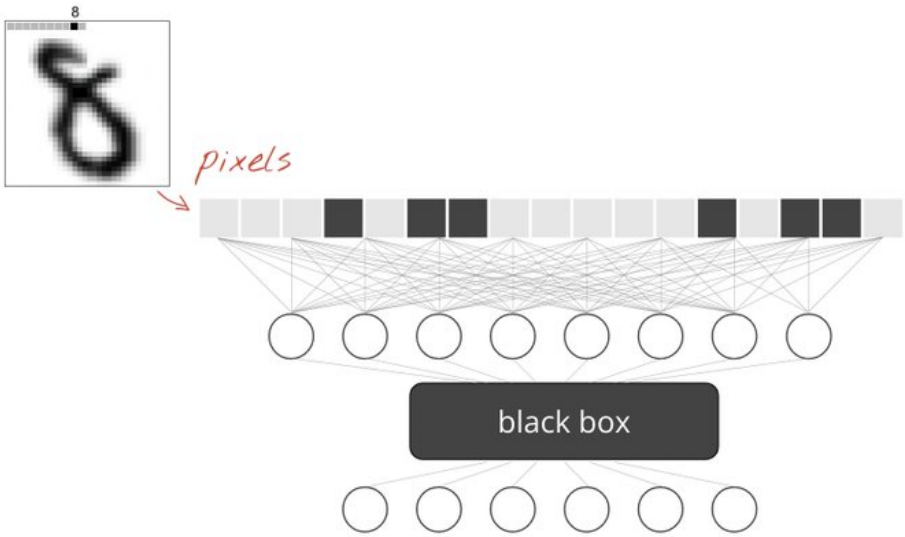
The networks contained two or three hidden layers of 3072 ReLUs each. Each hidden layer is a 32 x 32 topographic map with 3 hidden units at each location. Each hidden unit has an 11 x 11 receptive field in the layer below so it has 363 bottom-up inputs. For FF, hidden units in the last hidden layer have 10 top-down inputs and in other layers they have up to 363 top-down inputs from an 11 x 11 receptive field in the layer above.

learning procedure	testing procedure	number of hidden layers	training % error rate	test % error rate
BP		2	0	37
FF min ssq	compute goodness for every label	2	20	41
FF min ssq	one-pass softmax	2	31	45
FF max ssq	compute goodness for every label	2	25	44
FF max ssq	one-pass softmax	2	33	46
BP		3	2	39
FF min ssq	compute goodness for every label	3	24	41
FF min ssq	one-pass softmax	3	32	44
FF max ssq	compute goodness for every label	3	21	44
FF max ssq	one-pass softmax	3	31	46

DISCUSSION:

Weight updates in the Forward Forward algorithm do not change the layer normalized output for that input vector, allowing for simultaneous online weight updates in many different layers because weight updates in earlier layers do not change the activity vectors in later layers for that input vector.

It is possible to modify all of the weights at once so that each layer achieves the desired goodness of S* for the input vector x. Assuming that the input vector and all of the layer-normalized hidden vectors are all of length one, the learning rate that achieves this is given by: (S* / SL)^1/2 − 1, where SL is the current sum of squared activities of layer L prior to layer normalization. Unlike backpropagation, FF has no trouble learning if a black box is placed between the layers being trained by FF. The black box transforms the output of one layer using an unknown and possibly stochastic transformation and feeds this modified activity vector into the next layer. Making these black boxes into neural nets with a few hidden layers is an intriguing idea. Even though these inner layers learn slowly, the "outer loop" FF learning can swiftly adapt to new input if the black boxes are immobile. Slow learning in black boxes can thus be used to develop the system over a much longer timeline.



A slow reinforcement learning procedure, for example, could add small random noise vectors to the inputs to neurons inside the black box, then multiply these activity perturbation vectors by the change in the cost function used by the positive phase of FF to obtain a noisy but unbiased estimate of the derivative of the FF cost function with respect to the activities of neurons inside the black box.

CONCLUSION:

In summary, by lowering storage costs and enhancing model flexibility, the forward forward algorithm has cleared the path for significant advances in the field of deep learning. However, much more research is needed, especially in the fields of negative data production and activation function optimization. This algorithm's future potential could be much larger if we can devise a technique for the model to produce its own negative data, which would considerably improve its scalability. FF would be much easier to implement in a brain if the positive input was processed while awake and the negative data was created and processed by the network itself while sleeping. If the positive and negative phases can be separated, it would be fascinating to observe if removing the negative phase updates for a while results in effects that approximate the terrible effects of severe sleep deprivation. As a result, the forward forward algorithm continues to be an intriguing and promising study field for the future of deep learning.

REFERENCES:

https://www.cs.toronto.edu/~hinton/FFA13.pdf
https://twitter.com/martin_gorner/status/1599755684941557761
https://www.youtube.com/watch?v=NWqy_b1OvwQ
https://www.youtube.com/watch?v=F7wd4wQyPd8&t=375s