A
REPORT
ON

Forward-Forward Propogation

By

Aarav Goel **2020A1PS1696P**

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,
PILANI (Rajasthan)

(March, 2023)

Introduction

Deep Learning is a burgeoning discipline that has attracted a great deal of interest in recent years due to its success in solving complex problems such as image and speech recognition, natural language processing, and game playing. Deep Learning is a form of machine learning that employs artificial neural networks with multiple hidden layers for data-driven learning. The feedforward neural network, which consists of numerous layers of neurons connected to the next layer, is one of the most popular neural network architectures.

The feedforward neural network is trained using the backpropagation technique, which computes the gradient of the loss function with respect to the network's weights and biases and then modifies the network's weights using a gradient descent algorithm. Nevertheless, the backpropagation algorithm can be computationally expensive and memory-intensive, particularly when dealing with large datasets.

Backpropagation is an alternative training algorithm for feedforward neural networks that is less computationally expensive and requires less memory than the Forward-Forward algorithm. The algorithm was first described by Simon Haykin in his book "Neural Networks and Learning Machines" and has since been effectively implemented in a variety of applications, including speech recognition, image classification, and signal processing.

In this report, we will describe in detail the Forward-Forward algorithm and implement it in R for the classification task on the MNIST and CIFAR-10 datasets. We will compare the efficacy of the Forward-Forward algorithm and the backpropagation algorithm and discuss the pros and cons of both.

Algorithm Description

Forward-Forward Algorithm:

The Forward-Forward algorithm is a neural network algorithm that performs classification tasks. It is a variant of the traditional backpropagation algorithm, where the forward pass and backward pass are performed independently.

1. Initialization: Initialize the weights and biases of the network randomly.
2. Forward pass: Perform the forward pass to compute the output of the network given the input data. For each layer l, calculate the weighted input z(l) and activation a(l) as follows:

$z^{(l)} = W^{(l)} * a^{(l-1)} + b^{(l)}$ $a^{(l)} = g^{(l)}(z^{(l)})$
where $W^{(l)}$ is the weight matrix of layer l, $b^{(l)}$ is the bias vector of layer l, $a^{(l-1)}$ is the activation output of the previous layer, $g^{(l)}$ is the activation function of layer l, $z^{(l)}$ is the weighted input of layer l, and $a^{(l)}$ is the activation output of layer l.

3. Loss function: Compute the loss function, which is a measure of how well the network is performing. The loss function is given by:
   $J(\theta) = -(1/m) * \sum_{i=1}^{m}[\sum_{j=1}^{k}[y_j^{(i)} * \log(y\_hat_j^{(i)})]]$
   where m is the number of samples, k is the number of classes, $y_j^{(i)}$ is the true label of the i-th sample for class j, and $y\_hat_j^{(i)}$ is the predicted probability of the i-th sample for class j.
4. Gradient computation: Compute the gradients of the loss function with respect to the parameters of the network. For layer l, the gradient of the loss function with respect to the weighted input $z^{(l)}$ is given by:
   $\partial J / \partial z^{(l)} = (1/m) * (y\_hat - y) \odot g^{(l)'}(z^{(l)})$
   where y_hat is the predicted probability matrix, y is the true label matrix, $\odot$ is the element-wise product, and $g^{(l)'}$ is the derivative of the activation function $g^{(l)}$.
   The gradients of the loss function with respect to the weight matrix $W^{(l)}$ and bias vector $b^{(l)}$ of layer l are given by:
   $\partial J / \partial W^{(l)} = \partial J / \partial z^{(l)} * a^{(l-1)T}$ $\partial J / \partial b^{(l)} = (1/m) * \sum_{i=1}^{m}[\partial J / \partial z^{(l)}_i]$
   where $a^{(l-1)T}$ is the transpose of the activation output of the previous layer.
5. Update parameters: Update the weights and biases of the network using the gradients computed in step 4 and a learning rate η. The update rule is given by:
   $W^{(l)} = W^{(l)} - \eta * \partial J / \partial W^{(l)}$ $b^{(l)} = b^{(l)} - \eta * \partial J / \partial b^{(l)}$
6. Repeat steps 2-5 until convergence.

Backpropagation Algorithm:

The backpropagation algorithm is a neural network algorithm that performs classification tasks. It is a widely used algorithm for training deep neural networks. The algorithm is based on the chain rule of differentiation and performs both the forward pass and backward pass to update the weights and biases of the network.

1. Initialization: Initialize the weights and biases of the network randomly.
2. Forward pass: Perform the forward pass to compute the output of the network given the input data. For each layer l, calculate the weighted input z(l) and activation a(l) as follows:

$z^{(l)} = W^{(l)} * a^{(l-1)} + b^{(l)}$ $a^{(l)} = g^{(l)}(z^{(l)})$
where $W^{(l)}$ is the weight matrix of layer l, $b^{(l)}$ is the bias vector of layer l, $a^{(l-1)}$ is the activation output of the previous layer, $g^{(l)}$ is the activation function of layer l, $z^{(l)}$ is the weighted input of layer l, and $a^{(l)}$ is the activation output of layer l.

3. Loss function: Compute the loss function, which is a measure of how well the network is performing. The loss function is given by:
   $J(\theta) = -(1/m) * \sum_{i=1}^{m}[\sum_{j=1}^{k}[y_j^{(i)} * \log(\hat{y}_j^{(i)})]]$
   where m is the number of samples, k is the number of classes, $y_j^{(i)}$ is the true label of the i-th sample for class j, and $\hat{y}_j^{(i)}$ is the predicted probability of the i-th sample for class j.
4. Backward pass: Compute the gradients of the loss function with respect to the parameters of the network using the chain rule of differentiation. For layer l, the gradient of the loss function with respect to the weighted input $z^{(l)}$ is given by:
   $\partial J / \partial z^{(l)} = (\partial J / \partial a^{(l)}) \odot g^{(l)'}(z^{(l)})$
   where $\odot$ is the element-wise product, $g^{(l)'}$ is the derivative of the activation function $g^{(l)}$, and $\partial J / \partial a^{(l)}$ is the gradient of the loss function with respect to the activation output $a^{(l)}$ of layer l.
   The gradients of the loss function with respect to the weight matrix $W^{(l)}$ and bias vector $b^{(l)}$ of layer l are given by:
   $\partial J / \partial W^{(l)} = \partial J / \partial z^{(l)} * a^{(l-1)T}$ $\partial J / \partial b^{(l)} = \partial J / \partial z^{(l)}$
   where $a^{(l-1)T}$ is the transpose of the activation output of the previous layer.
5. Update parameters: Update the weights and biases of the network using the gradients computed in step 4 and a learning rate η. The update rule is given by:
6. $W^{(l)} = W^{(l)} - \eta * \partial J / \partial W^{(l)}$ $b^{(l)} = b^{(l)} - \eta * \partial J / \partial b^{(l)}$
7. Repeat steps 2-5 until convergence.

Dataset Description

MNIST Dataset:

The MNIST dataset is a collection of handwritten numerals used to train and evaluate machine learning algorithms. The dataset consists of 60,000 training images and 10,000 testing images, each 28 x 28 pixels in size. Each grayscale image is labelled with the number it represents, spanning from 0 to 9.

The MNIST dataset is widely utilised as a benchmark dataset for image classification tasks and has been utilised to evaluate the performance of a broad variety of machine learning algorithms, including neural networks. The dataset's simplicity and compact size make it an ideal option for testing and comparing machine learning algorithms.

CIFAR-10 Dataset:

The CIFAR-10 dataset is comprised of sixty thousand 32 x 32 colour images in ten classes, with six thousand images per class. There are 50,000 training images and 10,000 test images in the dataset. The 10 classes represent ubiquitous objects including aeroplanes, automobiles, birds, cats, deer, dogs, frogs, horses, and ships.

The CIFAR-10 dataset is widely used to evaluate and compare image classification algorithms, particularly for small-scale object recognition tasks. It is an ideal dataset for evaluating and comparing the performance of various algorithms, including neural networks, due to the tiny size of the images and the limited number of classes. However, the small size and simplicity of the dataset make it unsuitable for more complex tasks, such as fine-grained object recognition and image segmentation.

Results

We implemented the Forward-Forward algorithm in R for the classification task on the MNIST and CIFAR-10 datasets and compared its performance with backpropagation. We used the same architecture for both algorithms: a feedforward neural network with two hidden layers, each containing 128 neurons, and the ReLU activation function. We trained the networks using the same hyperparameters: a learning rate of 0.001 and a batch size of 32, for 50 epochs.
Table 1 shows the classification accuracy of the Forward-Forward algorithm and backpropagation on the MNIST and CIFAR-10 datasets.

Algorithm	Dataset	Accuracy
Forward-Forward	MNIST	97.86%
Backpropagation	MNIST	94.60%
Forward-Forward	CIFAR-10	5.75%
Backpropagation	CIFAR-10	40.99%

Table 1: Classification accuracy of the Forward-Forward algorithm and backpropagation on the MNIST and CIFAR-10 datasets

As can be seen from Table 1, backpropagation outperforms the Forward-Forward algorithm on both datasets. On the MNIST dataset, backpropagation achieves an accuracy of 94.60%, while the Forward-Forward algorithm achieves an accuracy of 97.86%. On the CIFAR-10 dataset,

backpropagation achieves an accuracy of 40.99%, while the Forward-Forward algorithm achieves an accuracy of 5.75%.

Conclusion

The results indicate that backpropagation outperforms the Forward-Forward algorithm on more complex datasets such as the CIFAR-10 datasets. This is not remarkable given that backpropagation is a well-established and effective algorithm for training neural networks. In contrast, the Forward-Forward algorithm is a relatively new algorithm that has not been extensively investigated.

A absence of information propagation between layers could be a contributing factor to the Forward-Forward algorithm's poor performance. Backpropagation involves the propagation of information from the output layer to the input layer, enabling the algorithm to modify the weights and biases in all layers. Information is only propagated from the input layer to the output layer in the Forward-Forward algorithm, making it more difficult to modify the weights and biases in the hidden layers.

The use of the ReLU activation function is an additional possible explanation. ReLU is a popular choice for neural networks due to its simplicity and efficacy, but it is not optimal for all applications. On occasion, other activation functions, such as sigmoid and tanh, may perform better.

Conclusion

In this report, we describe the Forward-Forward algorithm for training feedforward neural networks and compare its efficacy on the MNIST and CIFAR-10 datasets to that of backpropagation. The results indicate that backpropagation outperforms the Forward-Forward algorithm on complex datasets like CIFAR-10 while for MNIST dataset, forward propagation gives better results. Future research could examine the use of alternative activation functions or modifications to the Forward-Forward algorithm in an effort to enhance its performance.