Evaluation of Optimization Algorithms on LeNet5 for MNIST Data

Roya Abdulrazeq
Electrical & Computer Engineering
Rutgers University
New Brunswick, NJ, US
rfa54@scarletmail.rutgers.edu

ABSTRACT

Optimization Algorithms play a crucial role in improving a deep learning model's performance. They affect the accuracy and training time of the deep learning model. Optimizers are used to adjust the models' parameters such as weights to reduce the overall loss and improve accuracy. In this project, I evaluate the performance of three optimizers, Stochastic Gradient Descent (SGD), Adaptive Gradient Optimizer (Adagrad), and Root Mean Squared Propagation (RMSprop), on a LeNet-5 network using MNIST data.

1. Introduction

Neural Networks consist of input, output, and hidden layers that contain neurons to process information and learn to recognize patterns in data. The raw data enters the neural network from the input layer. Input nodes process the data and pass it on to the next hidden layer. A hidden layer takes the outputs of the previous layer i.e., the input layer or other hidden layers as an input and processes it further and pass it into the next layer. The output layer then outputs the final result. After that, the loss is calculated by comparing the predicted output against the actual output. Once the loss is defined, an optimizer is used to adjust the model's parameters to minimize the loss. In this project, I evaluated the performance of three optimizers, SGD, Adagrad, and RMSprop, on a LeNet_5 network using the MNIST dataset.

This paper is organized into 7 sections. Section 2 presents related work. Section 3 presents a description of the MNSIT dataset, Section 4 presents method description. Section 5 presents the model description. And section 6 presents the experimental procedure and results and finally a conclusion in the last section.

2. Related Work

In [1], They did a comparative analysis of several optimizers including SGD, Adagrad, and RMSprop. Their obtained results show that RMSprop results in the lowest mean absolute error (MAE) of 0.3788 in 210 iterations. Adagrad results in a MAE of 0.3858 in 53 iterations. And RMSprop results in the MAE of 0.7221 in 85 iterations. The performance of an optimizer is affected by several factors such as learning rate, setting the learning rate too high can cause the optimizer to skip the optimal solution while setting it too low can cause the training to progress too slowly. The performance is also affected by the model architecture and the size of the dataset and its complexity. In this project, I will explore the performance of these optimizers on the LeNet-5 network using the MNIST dataset.

3. Data Description

In this project, I used the MNIST database to train the LeNet-5 network to recognize handwritten digits. The MNIST contains 70,000 grayscale images of handwritten digits from 0 to 9. The database splits into 60,000 training images and 10,000 test images. Each image is 28x28 pixels. Each image is associated with a label indicating the digit it represents. Fig. 1 shows some samples from the MNIST database. [2]

Fig. 1 Sample Images from the MNIST Database with the Corresponding Labels

4. Method Description

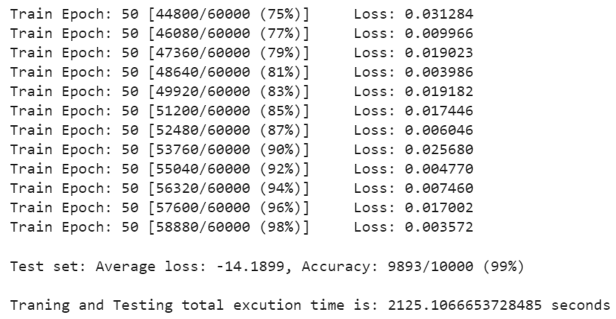
I used the MNIST database to explore the performance of different optimizers on the LeNet-5 network. In SGD, a few samples from the data are randomly selected to iteratively update the model's parameters, unlike the typical Gradient Descent where the batch is the whole dataset which becomes computationally expensive to perform when our dataset is huge. This problem is solved by the SGD since we are only using a small portion of the dataset for each iteration. In Adagrad, each weight has a different learning rate. The reason behind the need of making the learning rate adaptive for each weight is that in the real world, some input features are sparse where most of the values are zeros and some features are dense where they have values for most of their dimensions. Therefore, in this algorithm, each parameter has an adaptive learning rate. In RMSprop, the key feature of this algorithm is that it uses a moving average of the squared gradients to scale the learning rate for each parameter which helps stabilize the training process and improve the convergence speed [3]. I tested the performance of these optimizers on the LeNet-5 network.

5. Model Description

LeNet-5 is a convolutional neural network that was introduced in the research paper "Gradient-Based Learning Applied to Document Recognition" [4]. In this project, I used a modified model. The model contains 5 layers, three convolutional layers and two fully connected layers. The input to this model is a 32x32 grayscale image. Therefore, the number of channels is one. The first convolutional layer takes an input grayscale image and applies 6 filters with a kernel size of 5x5. This is followed by a ReLU activation function and a max pooling layer with a kernel size of 2x2 and stride of 2. The output then is passed into the second convolutional layer which applies 16 filters with a kernel size 5x5, and this is again followed by a ReLU and a max pooling layer. Then the output is passed into the third convolutional layer which applies 120 filters with a kernel size of 5x5 followed by a ReLU function. After the convolutional layers, the output is flattened into a vector and passed into the fully connected layers. The first fully connected layer has 120 input features and 84 output features followed by a ReLU function. The second fully connected layer has 84 input features and 10 output features corresponding to the 10 classes in the MNSIT dataset.

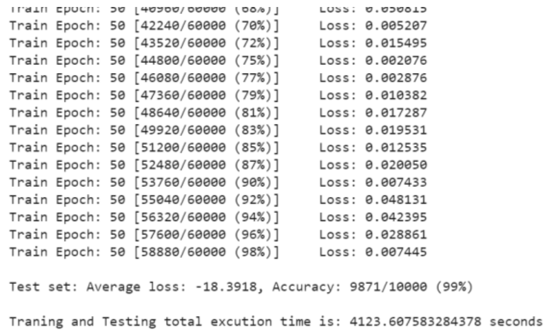
6. Experimental Procedure & Results

I set the learning rate to 0.05 and the number of epochs to 50 and evaluated the accuracy and the running time of the model using the different optimizers. Fig. 2 shows the result for the SGD optimizer, Fig. 3 shows the result for the Adagrad optimizer and Fig. 4 shows the result for the RMSprop optimizer.



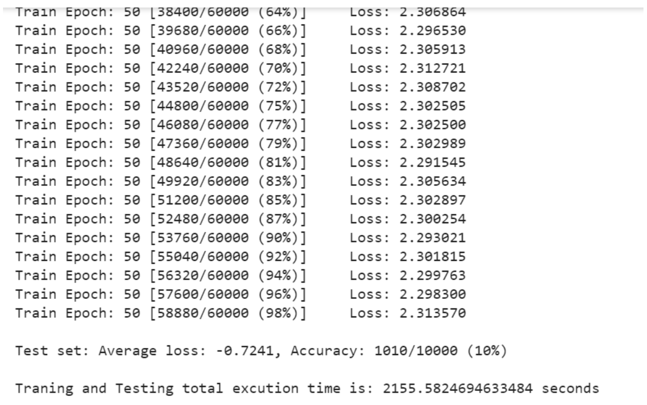
```
Train Epoch: 50 [44800/60000 (75%)]     Loss: 0.031284
Train Epoch: 50 [46080/60000 (77%)]     Loss: 0.009966
Train Epoch: 50 [47360/60000 (79%)]     Loss: 0.019023
Train Epoch: 50 [48640/60000 (81%)]     Loss: 0.003986
Train Epoch: 50 [49920/60000 (83%)]     Loss: 0.019182
Train Epoch: 50 [51200/60000 (85%)]     Loss: 0.017446
Train Epoch: 50 [52480/60000 (87%)]     Loss: 0.006046
Train Epoch: 50 [53760/60000 (90%)]     Loss: 0.025680
Train Epoch: 50 [55040/60000 (92%)]     Loss: 0.004770
Train Epoch: 50 [56320/60000 (94%)]     Loss: 0.007460
Train Epoch: 50 [57600/60000 (96%)]     Loss: 0.017002
Train Epoch: 50 [58880/60000 (98%)]     Loss: 0.003572

Test set: Average loss: -14.1899, Accuracy: 9893/10000 (99%)

Traning and Testing total excution time is: 2125.1066653728485 seconds
```

Fig. 2 The result of the model using the SGD optimizer with learning rate = 0.05



```
Train Epoch: 50 [40960/60000 (68%)]     Loss: 0.050815
Train Epoch: 50 [42240/60000 (70%)]     Loss: 0.005207
Train Epoch: 50 [43520/60000 (72%)]     Loss: 0.015495
Train Epoch: 50 [44800/60000 (75%)]     Loss: 0.002076
Train Epoch: 50 [46080/60000 (77%)]     Loss: 0.002876
Train Epoch: 50 [47360/60000 (79%)]     Loss: 0.010382
Train Epoch: 50 [48640/60000 (81%)]     Loss: 0.017287
Train Epoch: 50 [49920/60000 (83%)]     Loss: 0.019531
Train Epoch: 50 [51200/60000 (85%)]     Loss: 0.012535
Train Epoch: 50 [52480/60000 (87%)]     Loss: 0.020050
Train Epoch: 50 [53760/60000 (90%)]     Loss: 0.007433
Train Epoch: 50 [55040/60000 (92%)]     Loss: 0.048131
Train Epoch: 50 [56320/60000 (94%)]     Loss: 0.042395
Train Epoch: 50 [57600/60000 (96%)]     Loss: 0.028861
Train Epoch: 50 [58880/60000 (98%)]     Loss: 0.007445

Test set: Average loss: -18.3918, Accuracy: 9871/10000 (99%)

Traning and Testing total excution time is: 4123.607583284378 seconds
```

Fig. 3 The result of the model using the Adagrad optimizer with Learning Rate = 0.05

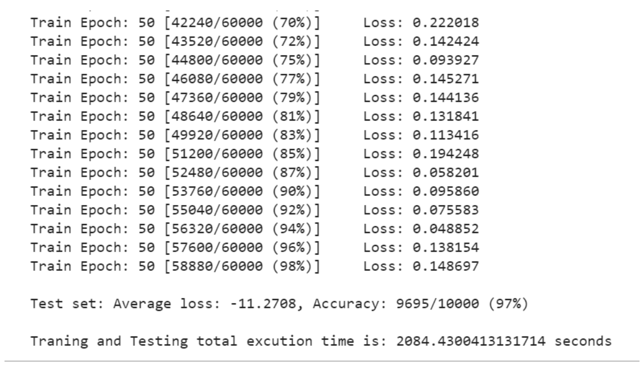


```
Train Epoch: 50 [38400/60000 (64%)]     Loss: 2.306864
Train Epoch: 50 [39680/60000 (66%)]     Loss: 2.296530
Train Epoch: 50 [40960/60000 (68%)]     Loss: 2.305913
Train Epoch: 50 [42240/60000 (70%)]     Loss: 2.312721
Train Epoch: 50 [43520/60000 (72%)]     Loss: 2.308702
Train Epoch: 50 [44800/60000 (75%)]     Loss: 2.302505
Train Epoch: 50 [46080/60000 (77%)]     Loss: 2.302500
Train Epoch: 50 [47360/60000 (79%)]     Loss: 2.302989
Train Epoch: 50 [48640/60000 (81%)]     Loss: 2.291545
Train Epoch: 50 [49920/60000 (83%)]     Loss: 2.305634
Train Epoch: 50 [51200/60000 (85%)]     Loss: 2.302897
Train Epoch: 50 [52480/60000 (87%)]     Loss: 2.300254
Train Epoch: 50 [53760/60000 (90%)]     Loss: 2.293021
Train Epoch: 50 [55040/60000 (92%)]     Loss: 2.301815
Train Epoch: 50 [56320/60000 (94%)]     Loss: 2.299763
Train Epoch: 50 [57600/60000 (96%)]     Loss: 2.298300
Train Epoch: 50 [58880/60000 (98%)]     Loss: 2.313570

Test set: Average loss: -0.7241, Accuracy: 1010/10000 (10%)

Traning and Testing total excution time is: 2155.5824694633484 seconds
```

Fig.4 The Results of the Model Using the RMSprop Optimizer with Learning Rate = 0.05

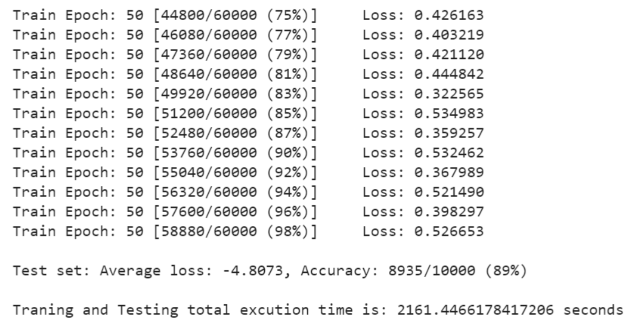
We can see that the SGD gives the best performance results with accuracy = 9893/10000 and the shortest execution time. Adagrad also performs well with accuracy = 9871/10000 but the execution time is much longer than the execution time in the experiment with the SGD (nearly double the execution time). Nevertheless, the RMSprop optimizer did not perform well. The accuracy is only 1010/10000.

Let's change the learning rate to 0.0001 and see how it will affect the performance of these optimizers. Fig. 5 shows the result of the model using the SGD optimizer with a learning rate = 0.0001. Fig. 6 shows the result of the model using the Adagrad optimizer with a learning rate = 0.0001. Fig. 7 shows the result of the model using the RMSprop optimizer with a learning rate = 0.0001



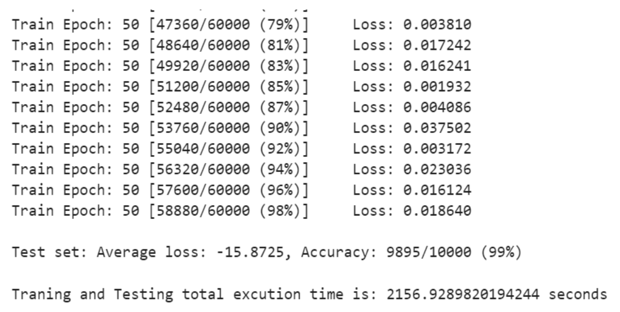
```
Train Epoch: 50 [42240/60000 (70%)]     Loss: 0.222018
Train Epoch: 50 [43520/60000 (72%)]     Loss: 0.142424
Train Epoch: 50 [44800/60000 (75%)]     Loss: 0.093927
Train Epoch: 50 [46080/60000 (77%)]     Loss: 0.145271
Train Epoch: 50 [47360/60000 (79%)]     Loss: 0.144136
Train Epoch: 50 [48640/60000 (81%)]     Loss: 0.131841
Train Epoch: 50 [49920/60000 (83%)]     Loss: 0.113416
Train Epoch: 50 [51200/60000 (85%)]     Loss: 0.194248
Train Epoch: 50 [52480/60000 (87%)]     Loss: 0.058201
Train Epoch: 50 [53760/60000 (90%)]     Loss: 0.095860
Train Epoch: 50 [55040/60000 (92%)]     Loss: 0.075583
Train Epoch: 50 [56320/60000 (94%)]     Loss: 0.048852
Train Epoch: 50 [57600/60000 (96%)]     Loss: 0.138154
Train Epoch: 50 [58880/60000 (98%)]     Loss: 0.148697

Test set: Average loss: -11.2708, Accuracy: 9695/10000 (97%)

Traning and Testing total excution time is: 2084.4300413131714 seconds
```

Fig. 5 the result of the model using the SGD optimizer with learning rate = 0.0001



```
Train Epoch: 50 [44800/60000 (75%)]     Loss: 0.426163
Train Epoch: 50 [46080/60000 (77%)]     Loss: 0.403219
Train Epoch: 50 [47360/60000 (79%)]     Loss: 0.421120
Train Epoch: 50 [48640/60000 (81%)]     Loss: 0.444842
Train Epoch: 50 [49920/60000 (83%)]     Loss: 0.322565
Train Epoch: 50 [51200/60000 (85%)]     Loss: 0.534983
Train Epoch: 50 [52480/60000 (87%)]     Loss: 0.359257
Train Epoch: 50 [53760/60000 (90%)]     Loss: 0.532462
Train Epoch: 50 [55040/60000 (92%)]     Loss: 0.367989
Train Epoch: 50 [56320/60000 (94%)]     Loss: 0.521490
Train Epoch: 50 [57600/60000 (96%)]     Loss: 0.398297
Train Epoch: 50 [58880/60000 (98%)]     Loss: 0.526653

Test set: Average loss: -4.8073, Accuracy: 8935/10000 (89%)

Traning and Testing total excution time is: 2161.4466178417206 seconds
```

Fig. 6 The Result of the Model Using the Adagrad Optimizer with Learning Rate = 0.0001



```
Train Epoch: 50 [47360/60000 (79%)]     Loss: 0.003810
Train Epoch: 50 [48640/60000 (81%)]     Loss: 0.017242
Train Epoch: 50 [49920/60000 (83%)]     Loss: 0.016241
Train Epoch: 50 [51200/60000 (85%)]     Loss: 0.001932
Train Epoch: 50 [52480/60000 (87%)]     Loss: 0.004086
Train Epoch: 50 [53760/60000 (90%)]     Loss: 0.037502
Train Epoch: 50 [55040/60000 (92%)]     Loss: 0.003172
Train Epoch: 50 [56320/60000 (94%)]     Loss: 0.023036
Train Epoch: 50 [57600/60000 (96%)]     Loss: 0.016124
Train Epoch: 50 [58880/60000 (98%)]     Loss: 0.018640

Test set: Average loss: -15.8725, Accuracy: 9895/10000 (99%)

Traning and Testing total excution time is: 2156.9289820194244 seconds
```

Fig. 7 The Result of the Model Using the RMSprop Optimizer with Learning Rate = 0.0001

We can see how changing the learning rate affected the model performance of the different optimizers. Changing the learning rate to 0.0001 decreases the accuracy and execution time of the SGD slightly (accuracy = 9695/10000 and execution time = 2084 seconds). And the accuracy for the Adagrad decreases to 8935/10000. However, the execution time of the Adagrad experiment decreases to nearly half the execution time of the experiment with the learning rate = 0.05. RMSprop gives the highest accuracy value 9895/10000 with execution time = 2156 seconds. Indeed, changing the learning rate affected the model performance, and the model with the RMSprop optimizer gives the best accuracy value.

7. Conclusion

In this project, I have done a comparative evaluation of the optimizers: SGD, Adagrad, and RMSprop on the LeNet-5 network using the MNIST dataset. We can conclude that the choice of the optimizer affects the model performance significantly for constant hyperparameters. Changing the hyperparameters such as the learning rate affects the model performance as well. SGD represents the best accurate result for the learning 0.05 and 50 epochs. RMSprop represents the most accurate result for the learning 0.0001 and 50 epochs. Indeed, besides the choice of the optimizer, the model performance is also affected by the choice of hyperparameters such as the learning rate and the model architecture itself. Therefore, to get the best results for any task, we should do a lot of experiments with different optimizers and hyperparameters.

REFERENCES

[1] Aatila Mustapha *et al* 2021 *J. Phys.: Conf. Ser.* 1743 012002.

[2] Nutan, "PyTorch convolutional neural network with Mnist Dataset," Medium, https://medium.com/@nutanbhogendrasharma/pytorch-convolutional-neural-network-with-mnist-dataset-4e8a4265e118 (accessed May 10, 2023).

[3] A. Gupta, "A Comprehensive Guide on Optimizers in Deep Learning," *Analytics Vidhya*, Oct. 07, 2021. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-on-deep-learning-optimizers/#Summary (accessed May 12, 2023).

[4] Y. Lecun, L. Bottou, Y. Bengio and P. Haffner, "Gradient-based learning applied to document recognition," in Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, Nov. 1998, doi: 10.1109/5.726791.