Autonomous Lab 3: HPC and Deep Learning

Xavier Samos I Casañe

June 5, 2024

Contents

1. Introduction

This study focuses on optimizing deep learning models using different algorithms on the MNIST dataset. The performance of Gradient Descent, Momentum Optimizer, and Adam Optimizer was evaluated to identify the best optimizer and optimal learning rates. The goal was to enhance model accuracy and convergence rates by selecting appropriate hyperparameters for each optimizer.

2. Exercise 1

- Try GradientDescentOptimizer with different learning rates.
- Check out other descent methods (look for options online).
- Represent these extra experiments plus the Gradient Descent with 0.01 learning rate.

2.1 Gradient descent

Various learning rates were tested for the proposed linear model using Gradient Descent. The experiments highlighted two different scenarios: learning rates below 0.05 and above 0.05. The Figures 1 and 2 show the results of both experiments.

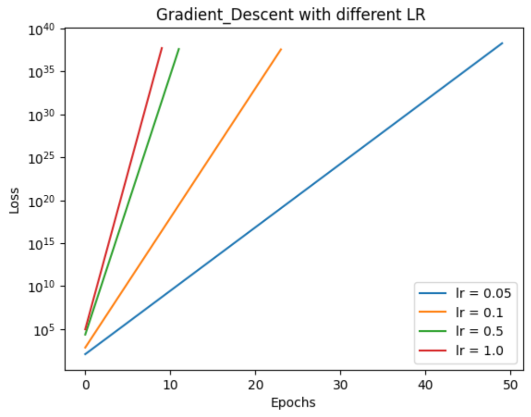


Figure 1. Gradient descent with LR above 0.05.

1. Learning Rates Below 0.05:
   - Convergence to the optimal value is achieved.
   - Faster convergence is observed with higher learning rates within this range.
   - Optimal performance was noted at a learning rate of 0.01, resulting in the fastest decrease in loss.
2. Learning Rates Above 0.05:
   - If the learning rate is too high, the algorithm overshoots the global optimum.
   - This results in the loss increasing towards infinity instead of decreasing.

2.2 Momentum optimizer

The Momentum optimizer is an improved version of the Stochastic Gradient Descent (SGD) algorithm. While SGD updates model parameters iteratively to minimize the loss function, it can sometimes be slow and get stuck in local minima. The Momentum optimizer helps to speed up this process and smooth out the path to the optimal solution.

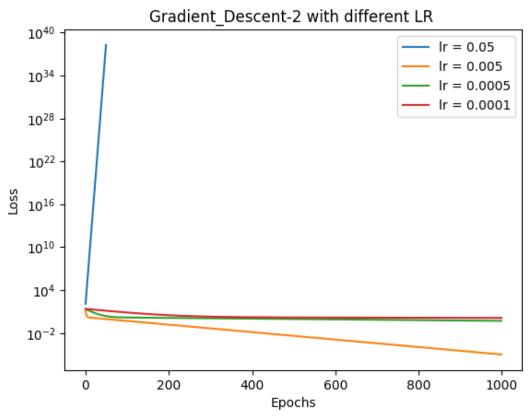


Figure 2. Gradient descent with LR below 0.05.

Momentum can be thought of as a push that keeps the optimizer moving in the same direction. It accumulates information from previous steps to maintain the movement towards the minimum, even if there are minor fluctuations in the gradient. This accumulated information helps to smooth out the updates and makes the optimization process more stable and faster.

- **Acceleration:** Momentum allows the optimizer to take larger steps towards the minimum, speeding up the convergence.
- **Smoothing Oscillations**: It reduces the zigzagging effect, leading to a smoother and more direct path to the optimal value.
- **Escaping Local Minima**: By maintaining a consistent direction, momentum helps the optimizer avoid getting stuck in small dips and continue towards the global minimum.

1. Lower Momentum (0.2) - (Fig. 3):
   - The convergence is slower because the optimizer doesn't carry much information from previous steps.
   - This makes the updates smaller and the progress more gradual.
2. Moderate Momentum (0.4 and 0.6) - (Fig. 4 and 5):
   - The optimizer starts to take larger steps and converges faster than with lower momentum.
   - There's a good balance between smoothing out oscillations and accelerating the learning process.

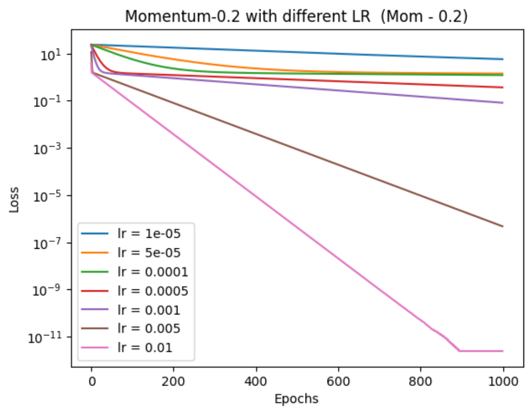


Figure 3. Momentum with different LR (Mom-0.2)

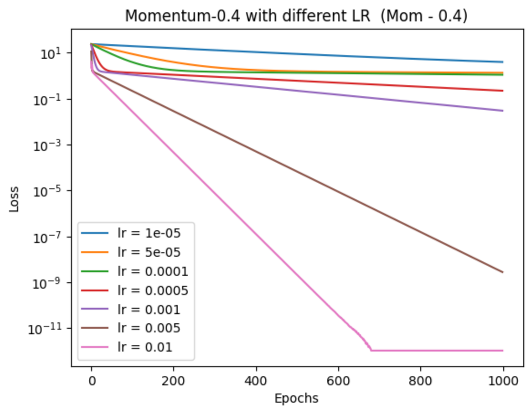


Figure 4. Momentum with different LR (Mom-0.4).

3. Higher Momentum (0.8)- (Fig. 6):
   - Convergence is even faster because the optimizer maintains a strong push in the direction of the overall gradient.
   - This helps in quickly reducing the loss, but if too high, it might overshoot the optimal value.

Thus, the fastest decrease is observed with the higher momentum experiment (Fig. 6), specifically the one with momentum equal to 0.8 and learning rate 0.01 (pink curve).

2.3 Adam optimizer

The Adam optimizer combines the advantages of two other optimization methods, AdaGrad and Momentum, to provide adaptive learning rates and smooth updates. Adam adjusts the learning rate for each parameter based on its individual gradient history, ensuring that parameters with larger gradients receive smaller updates and those with smaller gradients receive larger updates. This

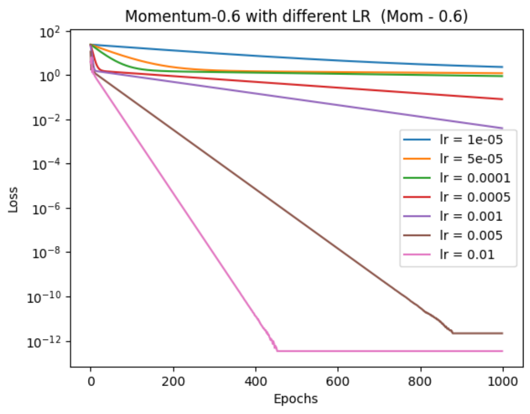


Figure 5. Momentum with different LR (Mom-0.6).

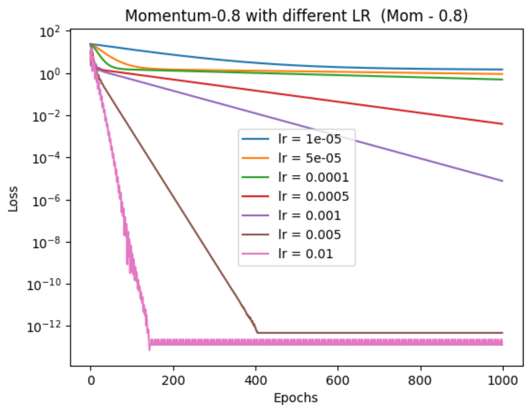


Figure 6. Momentum with different LR (Mom-0.8).

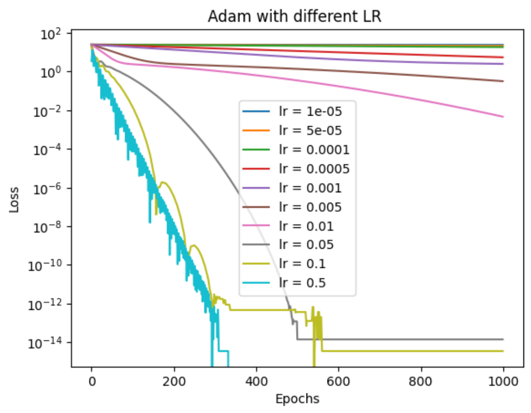


Figure 7. Adam with different LR.

adaptation helps to optimize the learning process for each parameter effectively.

Additionally, Adam incorporates momentum, which accumulates past gradient information to smooth out updates and prevent oscillations. By dynamically adjusting the learning rates and leveraging momentum.

In this case (Fig. 7), Adam converges even for learning rates above 0.05. Indeed, convergence appears much faster with an initial learning rate of 0.5. This rapid convergence is likely due to Adam's ability to automatically adapt the learning rate as it approaches the optimum. However, this automatic adaptation might also explain the erratic behavior observed for learning rates above 0.05, as the initial large updates can cause instability before the algorithm fine-tunes the learning rates.

2.4 Three approaches with LR - 0.01

Based on previous observations, the learning rate was set to 0.01 for both Momentum and Gradient Descent, and 0.05 for Adam. When comparing these three optimizers (Figure 8, Adam performs the best in terms of error minimization. The Momentum optimizer almost matches Adam in reaching the minimum but gets stuck at a slightly higher value. The Gradient Descent optimizer, however, converges much slower than the others and does not reach the plateau phase even after 1000 epochs. This comparison highlights the advantages of using more sophisticated optimization techniques, such as Adam, for gradient descent optimization.

3. Exercise 2

MNIST Dataset

- Plot convergence rates in a similar way as the previous exercise.

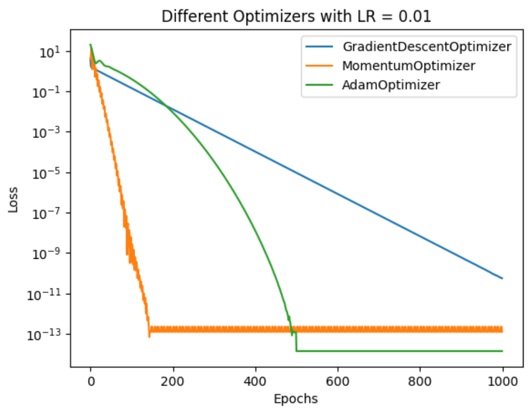


Figure 8. Comparison of the algorithms with LR 0.01

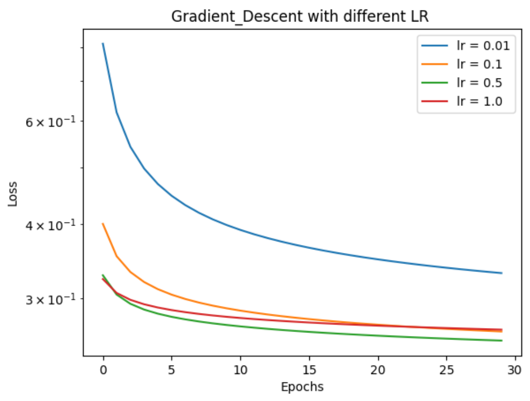


Figure 9. Gradient descent for different learning rates using MNIST dataset.

- Consider different optimizers and learning rates.

3.1 Gradient descent

This problem is more complex than the previous one because reaching the global optimum is no longer guaranteed. The goal is to choose the best optimizer and tune its hyperparameters to find the optimal local minima. In this scenario, there is a significant difference in convergence rates for each learning rate. For the number of epochs chosen, the best performing learning rate is 0.5. The results can be seen in Figure 9.

3.2 Momentum optimizer

In Exercise 1, momentum proved to be a crucial factor for the optimizer. However, in this case, momentum only slightly affects performance, and the difference doesn't appear to be significant. The best combination obtained here is a learning rate of 0.1 and a momentum of 0.8 (Red trace in Figure 13. The results can be seen in Figures 10, 11, 12 and 13.

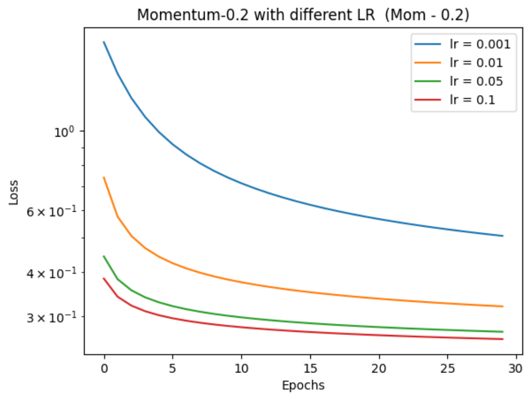


Figure 10. Momentum optimizer using different learning rates using MNIST dataset (Mom-0.2).

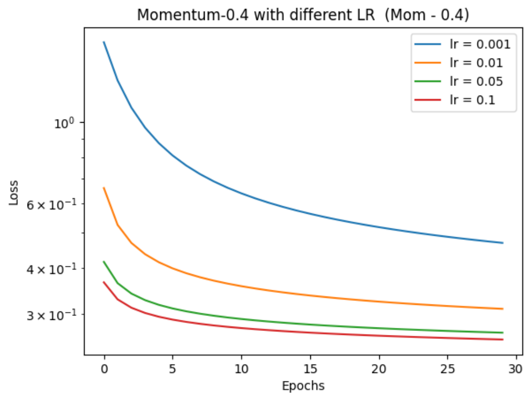


Figure 11. Momentum optimizer using different learning rates using MNIST dataset (Mom-0.4).

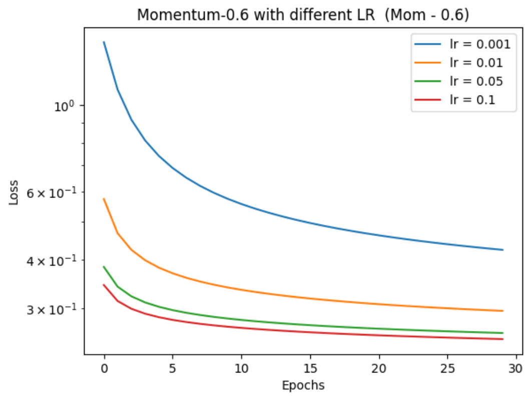


Figure 12. Momentum optimizer using different learning rates using MNIST dataset (Mom-0.6).

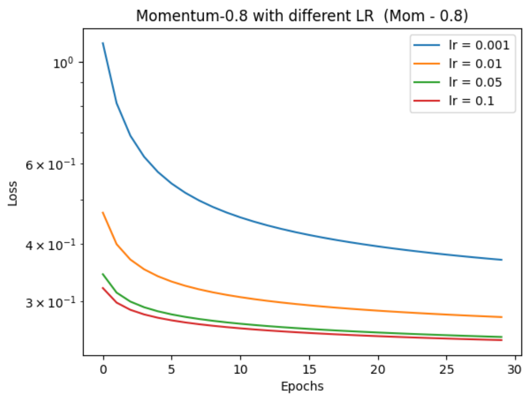


Figure 13. Momentum optimizer using different learning rates using MNIST dataset (Mom-0.8).

3.3 Adam optimizer

As previously explained, the Adam optimizer combines the advantages of both the Momentum and RMSprop optimizers, providing an adaptive learning rate and momentum. In this case, Adam shows robust performance, and the best learning rate is found to be 0.001. The results can be seen in Figure 14.

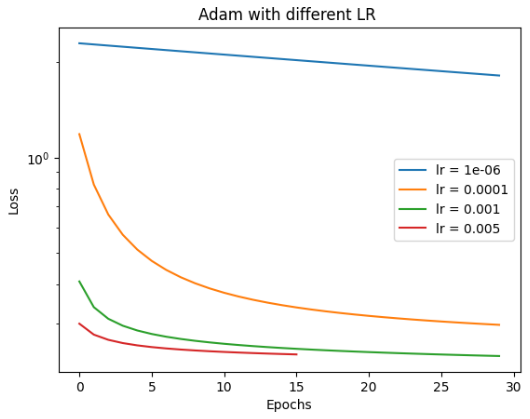


Figure 14. Adam optimizer using different learning rates using MNIST dataset.

3.4 Comparison of optimization algorithms

Finally, the best learning rates are chosen for each algorithm, and the number of epochs is increased. Once again, the Adam optimizer outperforms the others, with the Momentum optimizer in second place and the Gradient Descent optimizer in third. However, the performance gap is smaller compared to the first problem. This is mainly due to the use of different loss metrics (MSE vs. Categorical Cross Entropy), where the latter is not convex with respect to the parameters, making it more challenging to reach a global optimum. The results can be seen in Figure 15.

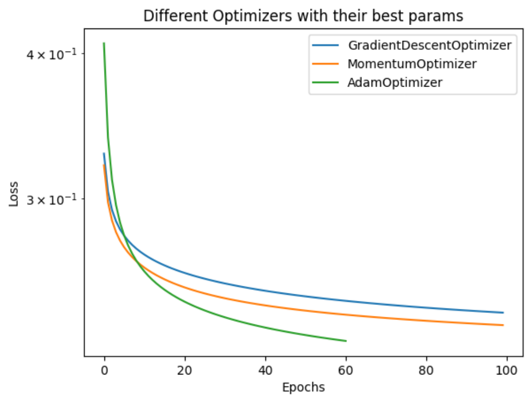


Figure 15. Comparison of the different algorithms for the MNIST dataset.

4. Exercise 3

The object is to increase the accuracy as much as possible using a deep neural network with multiple layers.

In the original code, only one epoch was being trained. Thus, the code has been modified as to train with more epochs and, the validation data is used to monitor the performance without incurring in overfitting. Different hyperparameters were tested, resulting in similar testing accuracies (0.992-0.993). The one provided yielded a test accuracy of 0.992 with the next configuration:

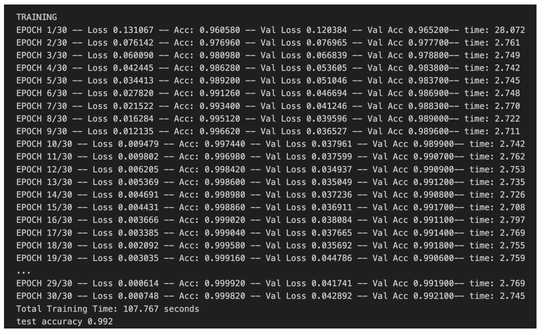
- **Learning rate**: 1e-4
- **Batch size**: 50
- **Epochs**: 30
- **Total training time**: 107.767s

The output of the execution can be seen in Figure 16. The loss and accuracy plots can be seen in Figure 17, respectively.

5. Conclusions

This study explored deep learning and optimization algorithms. Gradient Descent with varying learning rates, the Momentum Optimizer, and the Adam Optimizer were examined. Adam showed the best performance with its adaptive learning rates and effective convergence.

Applying these algorithms to the MNIST dataset highlighted the importance of choosing the right learning rates for different optimizers. For instance, the best



```
TRAINING
EPOCH 1/30 -- Loss 0.131067 -- Acc: 0.960580 -- Val Loss 0.120384 -- Val Acc 0.965200-- time: 28.072
EPOCH 2/30 -- Loss 0.076142 -- Acc: 0.976960 -- Val Loss 0.076965 -- Val Acc 0.977700-- time: 2.761
EPOCH 3/30 -- Loss 0.060090 -- Acc: 0.980980 -- Val Loss 0.066839 -- Val Acc 0.978800-- time: 2.749
EPOCH 4/30 -- Loss 0.042445 -- Acc: 0.986280 -- Val Loss 0.053605 -- Val Acc 0.983800-- time: 2.742
EPOCH 5/30 -- Loss 0.034413 -- Acc: 0.989200 -- Val Loss 0.051046 -- Val Acc 0.983700-- time: 2.745
EPOCH 6/30 -- Loss 0.027820 -- Acc: 0.991260 -- Val Loss 0.046694 -- Val Acc 0.986900-- time: 2.748
EPOCH 7/30 -- Loss 0.021522 -- Acc: 0.993400 -- Val Loss 0.041246 -- Val Acc 0.988300-- time: 2.770
EPOCH 8/30 -- Loss 0.016284 -- Acc: 0.995120 -- Val Loss 0.039596 -- Val Acc 0.989000-- time: 2.722
EPOCH 9/30 -- Loss 0.012135 -- Acc: 0.996620 -- Val Loss 0.036527 -- Val Acc 0.989600-- time: 2.711
EPOCH 10/30 -- Loss 0.009479 -- Acc: 0.997440 -- Val Loss 0.037961 -- Val Acc 0.989900-- time: 2.742
EPOCH 11/30 -- Loss 0.009802 -- Acc: 0.996980 -- Val Loss 0.037599 -- Val Acc 0.990700-- time: 2.762
EPOCH 12/30 -- Loss 0.006205 -- Acc: 0.998420 -- Val Loss 0.034937 -- Val Acc 0.990900-- time: 2.753
EPOCH 13/30 -- Loss 0.005369 -- Acc: 0.998600 -- Val Loss 0.035049 -- Val Acc 0.991200-- time: 2.735
EPOCH 14/30 -- Loss 0.004691 -- Acc: 0.998980 -- Val Loss 0.037236 -- Val Acc 0.990800-- time: 2.726
EPOCH 15/30 -- Loss 0.004431 -- Acc: 0.998860 -- Val Loss 0.036911 -- Val Acc 0.991700-- time: 2.708
EPOCH 16/30 -- Loss 0.003666 -- Acc: 0.999020 -- Val Loss 0.038084 -- Val Acc 0.991100-- time: 2.797
EPOCH 17/30 -- Loss 0.003385 -- Acc: 0.999040 -- Val Loss 0.037665 -- Val Acc 0.991400-- time: 2.769
EPOCH 18/30 -- Loss 0.002092 -- Acc: 0.999580 -- Val Loss 0.035692 -- Val Acc 0.991800-- time: 2.755
EPOCH 19/30 -- Loss 0.003035 -- Acc: 0.999160 -- Val Loss 0.044786 -- Val Acc 0.990600-- time: 2.759
...
EPOCH 29/30 -- Loss 0.000614 -- Acc: 0.999920 -- Val Loss 0.041741 -- Val Acc 0.991900-- time: 2.769
EPOCH 30/30 -- Loss 0.000748 -- Acc: 0.999820 -- Val Loss 0.042892 -- Val Acc 0.992100-- time: 2.745
Total Training Time: 107.767 seconds
test accuracy 0.992
```

Figure 16. Multilayer network training using the MNIST dataset.

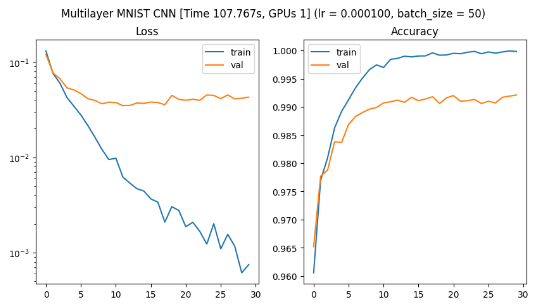


Figure 17. Multilayer network training using the MNIST dataset - Accuracy and Loss plots.

learning rate for Gradient Descent on MNIST was 0.5. Efforts were made to enhance performance by modifying the code for more epochs and using validation data. The achieved accuracy on the test set was 0.992, using a learning rate of 1e-4, a batch size of 50, and 30 epochs.

In conclusion, this research provides valuable insights into optimizing deep learning models. It emphasizes the importance of selecting the right algorithm and hyperparameters, demonstrating practical applications on the MNIST dataset.