EE 510: Deep Learning Theory and Practice **Winter 2020**

Course Project: Gradient Descent Optimization
Group Work, Due: March 15, 11:59 PM PT

Student Names: Ali Al-Bayaty, Matthew Bui, Sharanya Durgadasa

1 Comparison of Approaches/Models/Algorithms/Theories

- Interpretability. Adam[1] optimizer can be considered as a single and simpler approach to understand and to implement programmatically.
- Theoretical guarantees: Adam is considered as a more straightforward theoretical and it has the strongest theoretical guarantees for non-sequential data. Adaptive Gradient Algorithm(Adagrad)[2] for Online Learning was theoretically proved better than Non-adaptive gradient algorithms like SGD. As for the Online Learning model, they should be able to incorporate the behavior of the input data structure into the network.
- Empirical guarantees: Adam optimizer has better empirical results than many optimizers due to its simplest and straightforward achievements by using three different models of logistics regression, MLP, and CNN. They use different datasets for different models due to model complexity and requirements. Dozat[3] shows that Nadam; a modification of Adam, even has a better empirical result than Adam in benchmark data. In the experiments of Adagrad, it's shown that the Adaptive gradient algorithm is a better fit than a non-adaptive gradient algorithm in case of Online learning model[2].
- Computational complexity and training/evaluation time: In all optimizers, the most expensive computation is calculating ∇f which greater than linear time and the additional calculating can be done in linear time.

  $$\nabla f \in O(m), additional \in O(n), m >> n \Rightarrow \nabla f + additional \in O(m)$$

  As a result, the computational complexity of different optimizers is not significantly different. Since time complexity does not prevent us from choosing one optimizer over the others. We should pick them based on the dataset and learning model. Adam and Nadamare have been showed better if the data is dense[1][3], and Adagrad is better in case of sparse data[2]. Three models in KingmaBa2014[1] used different optimizers with their specific hyperparameters, most of the experiments use a fixed learning rate in order to test the adaptively generated learning rates and watch the convergence on the optimizers' convergence and computational training costs.

2 Implementation & Testing

This experiment will show the converging performances of different optimizers in non-sequential and sequential data as well as training time-consuming in non-sequential data.

- Language: Python 3.
- Framework & Library: Pytorch, Python time for estimate time.
- Optimizer: SGD, Momentum[4], NAG[5], RMSprop[6], AdaGrad[2], Adam[1], and Nadam[3].

2.1 Results on Synthetic Data

2.1.1 Experiment 1: Compare SGD, Adam, Adagrad, and RMSProp in Non-sequential Data

Data: We inherit the data from Professor Ted Willke's tutorial file[7]. Figure 1 shows the dataset generated randomly ten classes.

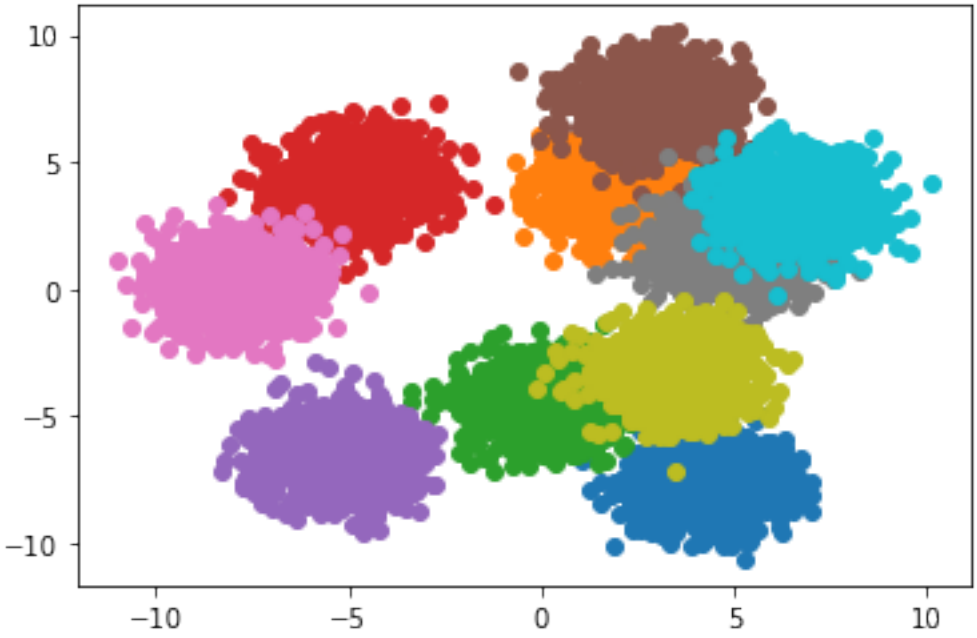

Figure 1: Non-sequential Synthetic Data

Model: The experiment uses the MLP model for the architecture of 3 hidden layers of 100x100x10 neurons each with a learning rate of 0.01, a batch size of 128, ReLU's, and 30 epochs.

Procedure: Training SGD, Adam, Adagrad, and RMSProp in Non-sequential synthetic data to get the loss performance and measure time consuming.

Result: Regarding the synthetic datasets (training/validation), Adam has a faster convergence than SGD, RMSProp, and AdaGrad, but Adam has fewer choppy than RMSProp as well as less smother than AdaGrad, as shown in the figure 3. The fastest computational training cost is SGD but with the lowest accuracy, while Adam and AdaGrad have the highest accuracy but Adam has the longest computational training cost followed by RMSProp. As an overall, Adam outperforms on synthetic datasets, while SGD performs badly on such datasets.

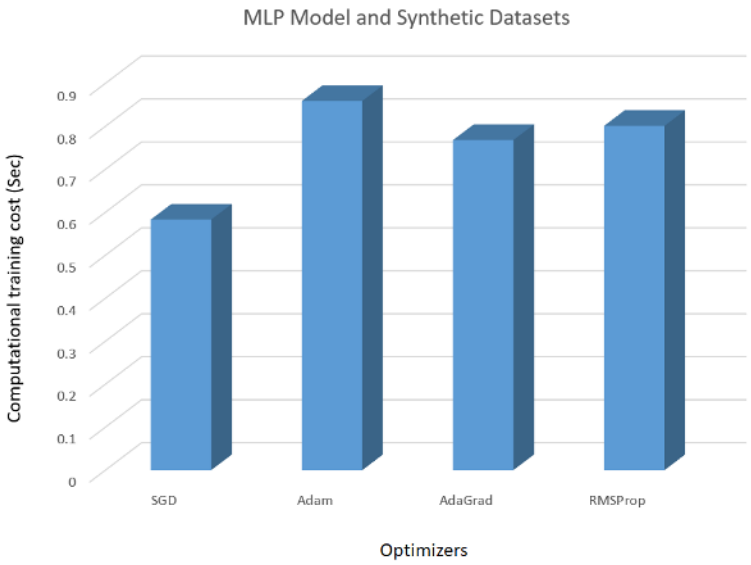

Figure 2: Computational training costs (in seconds) for different optimizers using synthetic datasets.

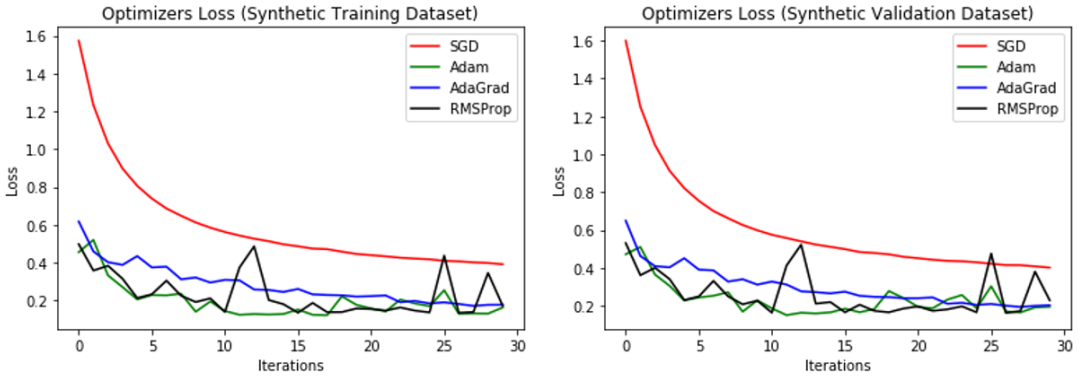

Figure 3: Loss performances for different optimizers using synthetic datasets.

Experiment 2: Compare Momentum, NAG, Adam, and Nadam in Non-sequential Data

Data: Same as experiment 1, we inherit the data from Professor Ted Willke's tutorial file[7].

Model: The experiment uses the MLP model for the architecture of 3 hidden layers of 100x100x10 neurons each with a learning rate of 0.01, a batch size of 128, ReLU's, and 30 epochs.

Procedure: Training the optimizers in Non-sequential synthetic data to get the loss performance and measure time-consuming.

Result: In figure 5, NAG, Adam, and momentum optimizers' loss reports confirm the theory in the papers. However, Nadam works unexpectedly when it has a worse loss than Adam. This may be because the Adam optimizer is written by Pytorch, so they have some auto tune function helping Adam have better performances. In the meanwhile, Nadam is implemented straight-forwardly. Although Nadam can not beat Adam, it still has better performance than momentum and NAG. In figure 4 shows Adam is the most expensive and the momentum is the cheapest. This is also unexpected for Nadam because it is supposed to take longer than Adam. The unexpected training cost of Adam may come from the framework and the

implementation. They are using different libraries, so the time-consuming is different However, the differences in the training cost are not big, so time complexity does not bother us from picking the optimizers.

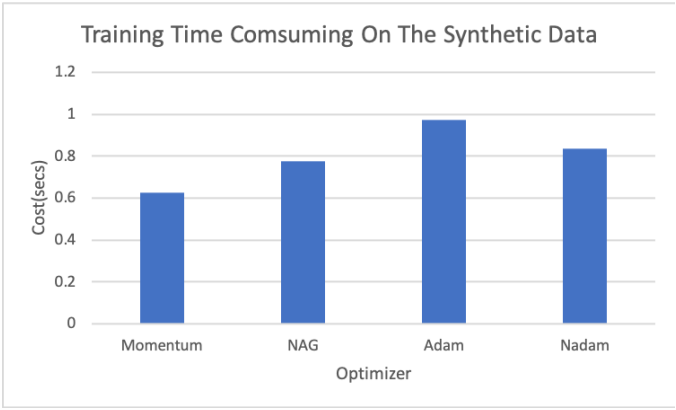

Figure 4: Experiment 2 Cost Comparasion.

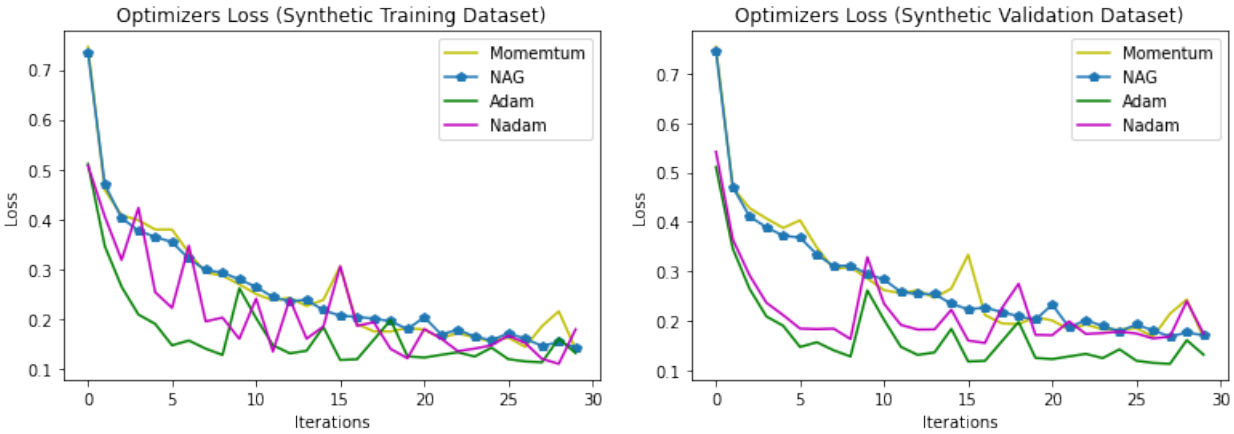

Figure 5: Experiment 2 Loss Comparasion.

Experiment 3: Compare all available optimizers' performances in Sequential Data

Data: We inherit the data from Professor Ted Willke's sequential example tutorial file[7]. Figure 6 shows sine wave generated sequentally by the code.

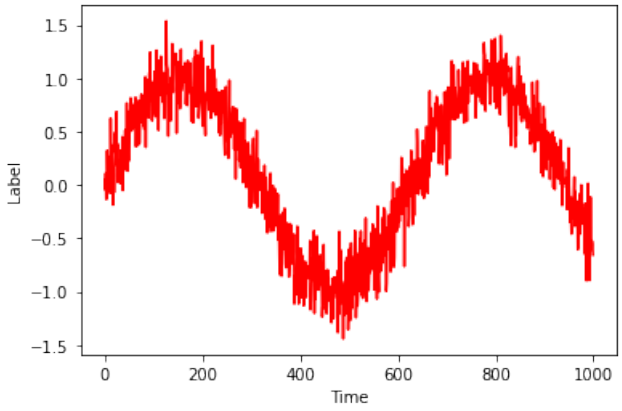

Figure 6: Sequential Synthetic Data

Model: The experiment uses the MLP model for the architecture of 1 hidden layers of 10 neurons each with a learning rate of 0.01, a batch size of 128, ReLU's, the history length = 4, number of time steps = 1000, number of training = 100, and 30 epochs.

Procedure: Training the optimizers in sequential synthetic data to get predicting function.

Result: They all perform expectedly. Adagrad have excellent output function. Adam, Nadam, NAG, and Momemtum have the same shapes which are not the sine wave. Although SGD is better other optimizers, it is worse than Adagrad. This confirms theory in Duhasi2011[2].

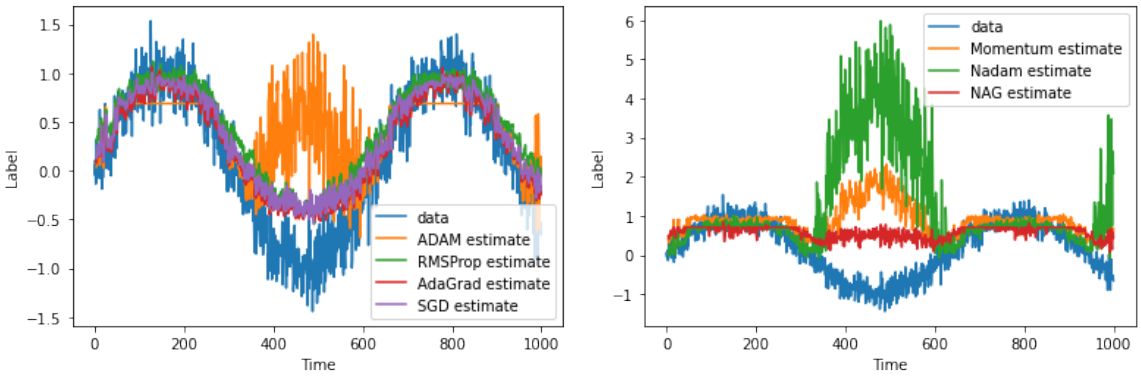

Figure 7: Experiment 3 Results in Synthetic Data

2.2 Results on Benchmark Data

2.2.1 Experiment 4: Compare SGD, Adam, Adagrad, and RMSProp in Non-sequential Data

Data: We inherit the handwritten digits image data from MNIST[8].

Model: The experiment uses the MLP model for the architecture of 3 hidden layers of 100x100x10 neurons each with a learning rate of 0.01, a batch size of 128, ReLU's, and 30 epochs.

Procedure: Training the optimizers in Non-sequential benchmark data to get the loss performance and measure time-consuming.

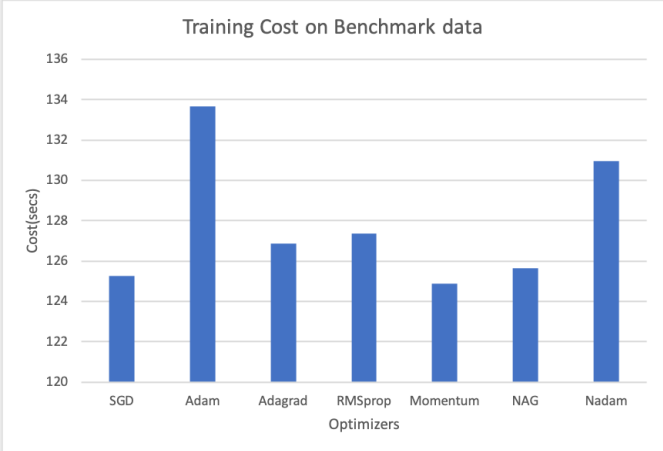

Figure 8: Training Cost Report of Experiment 4.

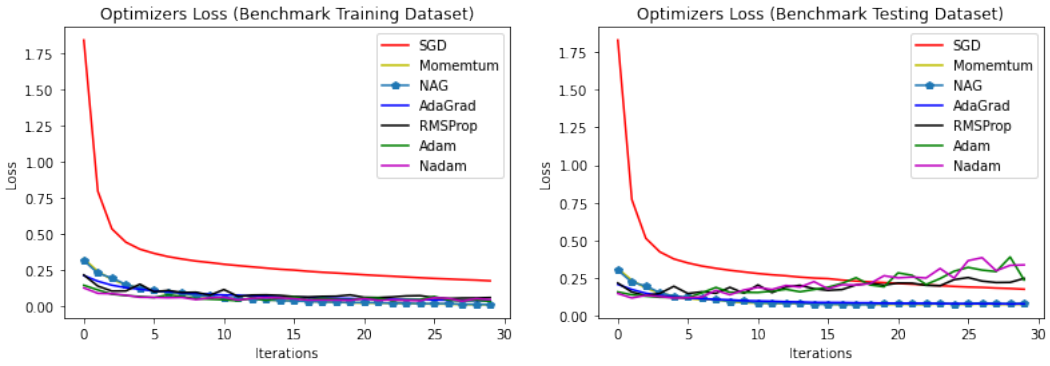

Figure 9: Loss Report of Experiment 4.

Result: The SGD has the lowest accuracy, while the momentum has the highest accuracy in training which is unexpected. NAG, Adam, and Adagrad have the same performances which go to the second place of 1 percent. Look at the test loss in figure 9, we can see overfitting happens in RMSprop, Nadam, and Adam. The training cost is not surprising comparing with experiments 1, 2. Adam is still the most expensive one and SGD is the cheapest one. Based on this experiment MLP model and synthetic/benchmark datasets, AdaGrad and NAG have the best values in the account of time and accuracy, and this is somewhat not comparable with KingmaBa2014[1] and Dozat2016[3]. The unexpected results may come from the set up of optimizers. Hyper-parameters are crucial to the optimizer and choosing them is challenging for deep learning researchers. It requires a lot of works such as weight initialization and cross-validation. Beside, the architecture is also important. The different used MLP architecture of the two fully connected hidden layers with 1000 neurons each and usage of the SFO (sum-of-functions) method that has shown good performance on optimization of MLP as proposed in KingmaBa2014[1].

References

[1] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[2] E. H. J. Duchi and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," 2011.

[3] T. Dozat, "Incorporating nesterov momentum into adam," 2016.

[4] B. T. Polyak, "Some methods of speeding up the convergence of iteration methods," *1964*.

[5] Y. Nesterov, "A method of solving a convex programming problem with convergence rate $o(1/k^2)$." 1983.

[6] G. Hinton, "Lecture 6a overview of mini-batch gradient descent." [Online]. Available: http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[7] T. Willke, "Ee 510: Deep learning theory and practice, winter 2020," 2020. [Online]. Available: http://web.cecs.pdx.edu/~willke/courses/EE510W20/

[8] C. J. B. Yann LeCun, Corinna Cortes, "The mnist database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/

A Appendix

Github repository: `https://github.com/mbui0529/GradientDescentOptimizer`