Adaptive Optimization Methods for Deep Learning

Mariko Makhmutova
School of Computer and Communication Sciences, EPFL

***Abstract*—Adaptive optimization methods are becoming more popular in the field of deep learning. It is becoming increasingly more difficult to decide which optimization algorithm to use when approaching a deep learning problem. In this paper, we test the performances of multiple adaptive methods with multi-layer perceptron models. In particular, we compare AdaDelta, Adam and AdaBound, all of which are recent and popular adaptive optimization algorithms. We use the MNIST digit classification dataset to test their performances. We conclude by evaluating which optimization algorithm may be preferable in the context of a deep learning problem.**

I. Introduction

Adaptive optimization algorithms adjust their parameters based on the data, which has shown to significantly improve the performance of neural network models [1,2]. The classic stochastic gradient descent algorithm scales the gradient uniformly in all directions, which can sometimes be unfavourable, depending on the dataset. The advantage of adaptive methods is that they scale the gradient individually for each parameter, so they often achieve a better performance than SGD.

As adaptive algorithms have been gaining popularity in the field of deep learning, it is becoming increasingly difficult to choose an optimization algorithm when approaching a new problem. In this work, we study how three different adaptive optimization methods perform in deep learning tasks.

In particular, we will be studying how multi-layer perceptrons (MLPs), the most common models that are encountered in deep learning, perform when using AdaDelta [1], Adam [2] and AdaBound [3] optimization algorithms. We will be testing the optimization algorithms on MLPs with varying numbers of hidden layers, in an attempt to establish which algorithms are preferable to use, or at least begin with, in the context of a multi-layer perceptron problem.

II. Models and methods

A. Algorithms

All three of the optimization algorithms we consider are stochastic gradient-based optimization algorithms. One of the first such algorithms published was AdaGrad, an algorithm that adapts the learning rate to the parameters. Lower learning rates are used for parameters that are associated with more frequent features and higher learning rates are used for parameters that are associated with less frequent features [4].

The AdaGrad update rule is quite similar to the SGD update rule, with learning rate η and objective function g, but with an additional factor. For every parameter θ_i at time step t, the update rule is as follows:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} \cdot g_{t,i}$$

In the update rule, $g_{t,i}$ is the partial derivative of g with respect to parameter θ_i at time step t and $G_t \in \mathbb{R}^{d \times d}$ is a diagonal matrix such that $G_{t,ii}$ is the sum of squares of gradients with respect to θ_i until time step t. The ϵ term is required to avoid divisions by zero in the implementation.

A major issue with AdaGrad is that the learning rate constantly decreases to a point where the algorithm stops learning. AdaDelta is an extension of AdaGrad, which aims solve this problem. It does so by considering only a fixed number of accumulated past gradients, effectively reducing the rate at which the learning rate decreases. The AdaDelta update rule replaces the exponentially decaying term with the root mean squared error of parameter updates as follows:

$$\mathbb{E}[\Delta\theta^2]_t = \rho\mathbb{E}[\Delta\theta^2]_{t-1} + (1-\rho)\Delta\theta_t^2$$
$$RMS[\Delta\theta]_t = \sqrt{\mathbb{E}[\Delta\theta^2]_t - \epsilon}$$
$$\theta_{t+1} = \theta_t - \frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t} \cdot g_t$$

Another benefit of AdaDelta is that it requires no manual tuning of the learning rate.

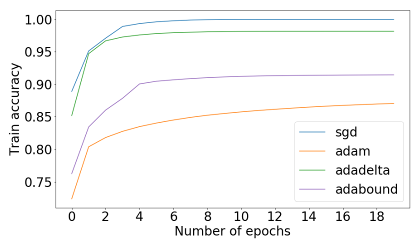
Adaptive Moment Estimation (Adam) is a more recently proposed optimization algorithm, that is described as a combination of RMSprop (similar to AdaDelta) and SGD with momentum. Adam uses bias-corrected estimators of first and second moments of the gradient to adapt the learning rates of the parameters. We refer the reader to the full update rule in Algorithm 1.

Algorithm 1: Adam update rule

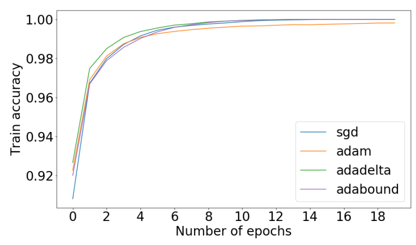
```
Requires: Learning rate $\eta$
Requires: Decay rates $\beta_1, \beta_2 \in [0,1)$
Requires: Objective function $f(\theta)$, initial parameter $\theta_0$
1 Initialize $m_0 = 0, v_0 = 0$
2 for $t = 1, ..., T-1$ do
3     $g_t \leftarrow \Delta f(\theta_t)$
4     $m_{t+1} \leftarrow \beta_1 m_t + (1-\beta_1) \cdot g_t$
5     $v_{t+1} \leftarrow \beta_2 v_t + (1-\beta_2) \cdot g_t^2$
6     $\widehat{m}_{t+1} \leftarrow m_{t+1}/(1-\beta_1^{t+1})$
7     $\widehat{v}_{t+1} \leftarrow v_{t+1}/(1-\beta_2^{t+1})$
8     $\theta_{t+1} \leftarrow \theta_t - \eta \cdot \widehat{m}_{t+1}/(\widehat{v}_{t+1} + \epsilon)$
9 end
```

The third optimization algorithm considered, AdaBound, is a recently developed variant of Adam.[1] AdaBound aims to avoid unstable and extreme learning rates often obtained by adaptive methods by applying dynamic bounds on learning rates to achieve a gradual transition from Adam to SGD.

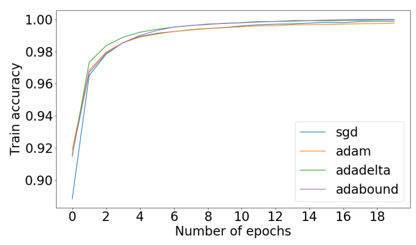
[1]Presented at ICLR 2019.



(a) Two-layer MLP

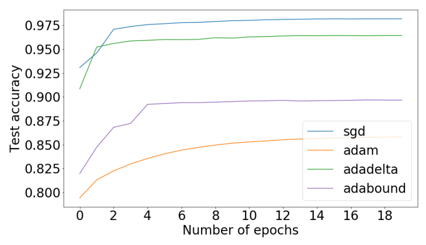


(b) Three-layer MLP

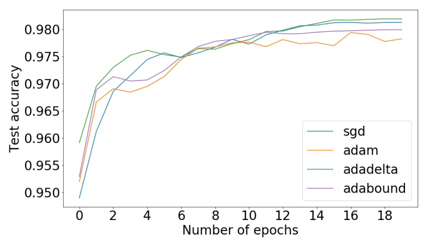


(c) Four-layer MLP

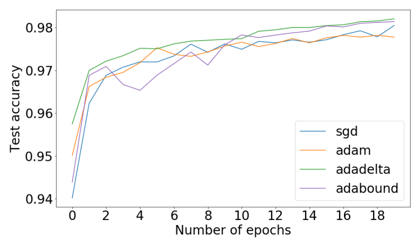
Fig. 1: MNIST training accuracy for different multi-layer perceptron models



(a) Two-layer MLP



(b) Three-layer MLP



(c) Four-layer MLP

Fig. 2: MNIST test accuracy for different multi-layer perceptron models

B. Models

The dataset we use to compare the different optimization methods is the MNIST digit classification dataset. We have 60000 samples in the training set and 10000 samples in the test set. We standardize the dataset to potentially improve convergence.

We use a two-layer MLP, a three-layer MLP and a four-layer MLP as our testing models for the algorithms. The decision to use MLPs to study the difference in performance of each of the optimization algorithms is due to the popularity of MLPs in deep learning. The outcomes of this study can then be more beneficial when choosing the initial optimization algorithm when designing an MLP architecture.

The first MLP contains one hidden layer with 250 neurons. The second MLP has two hidden layers, with 250 and 100 neurons, respectively. The third MLP contains three hidden layers, with 250, 250 and 100 neurons, respectively. All three multi-layer perceptrons use ReLU activation functions and mini-batches of size 64.

The baseline algorithm used for comparison is the stochastic gradient descent algorithm. For each of the algorithms and neural networks in the comparison, we use the hyperparameters that give the lowest training loss, which can be referred to in Appendix A.

III. Results

In Fig. 1 and Fig. 2, we see the results of the training accuracy and test accuracy of each of the optimization methods for each of the models tested. The accuracy results are calculated for 20 epochs, and averaged over five models for a more accurate representation of the behaviour of the models with each optimization method.

We can see from Fig. 1a and Fig. 2a that SGD appears to be the best performing algorithm with a shallow two-layer network (one hidden layer).

However, we can see that the performances of the different algorithms significantly change with the addition of a single hidden layer, as we see in Fig. 1b and Fig. 2b. We can see that the best-performing algorithm appears to be AdaDelta here, with the fastest growing training accuracy and highest final test accuracy. We observe that Adam does not appear to be very robust, with many fluctuations and a final test score lower than the baseline SGD. Luo et al. [3] noticed the lack of robustness of Adam, developing from this algorithm. It appears that AdaBound is indeed performing slightly better and more smoothly than Adam, with consistent results over multiple models. This can also be said for AdaDelta.

In Fig. 1c and Fig. 2c, one can observe that Adam appears to be performing better, but it is still quite unstable. AdaBound has the second highest test accuracy, after AdaDelta, which supports that AdaBound might be a better suited optimization algorithm than Adam for more shallow multi-layer perceptrons.

We conclude that AdaDelta and AdaBound produced the most consistent and best training and test accuracy results, whereas Adam produced less stable results with lower training and test accuracy.

Algorithm	Two-layer MLP	Three-layer MLP	Four-layer MLP
SGD	2.476	2.662	3.071
AdaDelta	2.835	4.107	4.907
Adam	2.881	4.570	5.659
AdaBound	3.419	3.865	4.276

TABLE I: Average training time per epoch in seconds

We also consider the training time per epoch provided in Table I. We observe that AdaBound consistently has a lower training time than the other two algorithms, making it a potentially more favourable method to use in the case of time constraints.

IV. Discussion

From our experiments, we can conclude that AdaDelta and AdaBound are better suited adaptive methods for multi-layer perceptron models than Adam. Adam appears to suffer from instability in results and it has a larger computation time.

We refer the reader to Appendix B to study the evolution of the training loss for each of the algorithms for a further analysis on their performances.

One limitation of this work is the use of a single dataset to evaluate the performance of the different algorithms. This study could be extended to consider other datasets in order to observe the extent to which the data used affects the effectiveness of the optimization algorithm. Another further inquiry could be done into another recently developed adaptive method, YamAdam [5], which combines the Adam and AdaDelta algorithms. Given the effectiveness of AdaBound, an extension of Adam and SGD, this could be another optimization algorithm worth discussing.

References

[1] M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://arxiv.org/abs/1212.5701

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[3] L. Luo, Y. Xiong, Y. Liu, and X. Sun, "Adaptive gradient methods with dynamic bound of learning rate," in *Proceedings of the 7th International Conference on Learning Representations*, New Orleans, Louisiana, May 2019. [Online]. Available: https://arxiv.org/abs/1902.09843

[4] S. Ruder, "An overview of gradient descent optimization algorithms," Jan 2016. [Online]. Available: http://ruder.io/optimizing-gradient-descent/

[5] K. D. Yamada, "Yamadam: a hyperparameter-free gradient descent optimizer that incorporates unit correction and moment estimation," *bioRxiv*, 2018. [Online]. Available: https://www.biorxiv.org/content/early/2018/07/17/348557

APPENDIX A
OPTIMAL HYPERPARAMETER VALUES

We provide the hyperparameter values used to train the MLPs with each optimization algorithm. The hyperparameters used were those that gave the lowest training loss amongst those tested in a grid search.

MLP with one hidden layer:

- SGD: `lr=0.1`, `momentum=0.7`
- AdaDelta: `lr=0.9`, `rho=0.99`
- Adam: `lr=0.0001`, `beta1=0.99`, `beta2=0.99`
- AdaBound: `lr=0.01`, `beta1=0.99`, `beta2=0.999`, `final_lr=0.1`

MLP with two hidden layers:

- SGD: `lr=0.05`, `momentum=0.7`
- AdaDelta: `lr=0.9`, `rho=0.9`
- Adam: `lr=0.001`, `beta1=0.9`, `beta2=0.99`
- AdaBound: `lr=0.001`, `beta1=0.9`, `beta2=0.99`, `final_lr=0.1`

MLP with three hidden layers:

- SGD: `lr=0.05`, `momentum=0.7`
- AdaDelta: `lr=0.9`, `rho=0.95`
- Adam: `lr=0.001`, `beta1=0.9`, `beta2=0.99`
- AdaBound: `lr=0.001`, `beta1=0.9`, `beta2=0.999`, `final_lr=0.1`

APPENDIX B
TRAINING LOSS PLOTS

We present the training loss plots for each of the optimization algorithms and MLPs tested. We can see that, as stated in the main section of the report, Adam appears to be more unstable than the other algorithms. We also observe that all of the algorithms appear to converge to zero for the three-layer and four-layer MLPs (with two and three hidden layers, respectively).

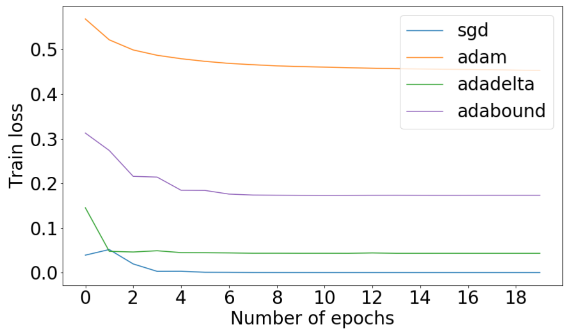


Fig. 3: Training loss for two-layer MLP

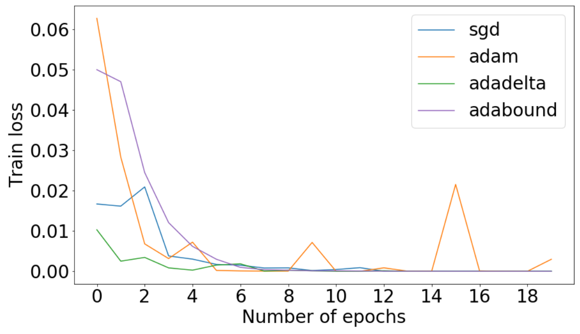


Fig. 4: Training loss for three-layer MLP

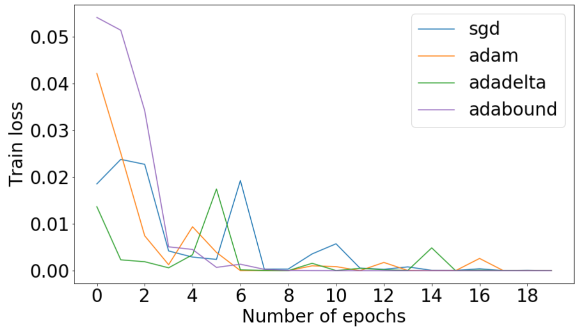


Fig. 5: Training loss for four-layer MLP