Gradient Descent Optimization Algorithms in Neural Network

Yufeng Yang Yinghao Li YungAn Hsieh Xin Chen Xuan Guo

1 Project Summary

Optimization algorithm is an important part in all kinds of Neural Networks. A good choice of optimizer and learning rate not only increases the training speed, but also contributes to the ultimate training accuracy of the network. Although Gradient Descent (GD) is a simple but nonetheless reliable optimization method, it consumes a large amount of system resources but achieves limited convergence rate. Newton's method has impressive convergence rate, but the calculation of inverse Hessian matrix makes it not feasible in practical usage. In this project, we aim to explore the derivatives of gradient descent, and compare their performances on different kinds of networks.

Current papers concentrate on either the theoretical analysis and comparison of the algorithms or a specific application-level case. Instead of this, our project focuses on both the theoretical understanding of the algorithms and their practical working scenarios in real-world applications. There are many claims on gradient descent algorithms, such as "gradient descent converges to a local minimizer, almost surely with random initialization". We are curious on how these algorithms perform regarding on the claim and we believe this should be an interesting project.

In section 2, we investigated several popular optimization algorithms in detail according to evolution path as well as their features. Section 3 introduced our experiment platform, including the dataset we chose and network structures we tested on. The testing result is shown in section 4 along with discussion on the algorithm's behaviors.

2 Algorithm Introduction

In this section, we introduce algorithms we want to explore in our project, then compare and discuss the differences between different approaches.

2.1 SGD

In this report, stochastic gradient descent (SGD) stands for not only stochastic gradient descent but also mini-batch gradient descent, for these two algorithms share an identical concept. The formula to compute parameter-set θ at time $t+1$ is

$$\theta_{t+1} = \theta_t - \eta \cdot \nabla_{\theta t} \mathrm{J}\left(\theta_t, \mathbf{x}^{(i:i+n)}, \mathbf{y}^{(i:i+n)}\right) \tag{1}$$

where $\mathrm{J}(\cdot)$ is the loss function, n is batch size. For the stochastic gradient descent in a narrow sense, $n = 0$.

Compared with gradient descent, SGD performs a parameter update for each example or each batch of training examples, leading to a higher computing speed than gradient descent. SGD is guaranteed to converge into global or local minimum for convex or non-convex optimization except for some special initial starting points. However, SGD depends on hyper-parameter η and a proper value could be hard to find sometimes. In addition, all parameter updates depend on the same learning rate, which makes the network fitted better to the frequently appeared features.

2.2 SGDM

Stochastic gradient descent with momentum term (SGDM) [1] algorithm introduced the concept of momentum into SGD, targeting to solve the problem of the probable oscillates when one dimension has steeper slope than others. Suppose $g_t = \nabla_{\theta t} \mathrm{J}\left(\theta_t, \mathbf{x}^{(i:i+n)}, \mathbf{y}^{(i:i+n)}\right)$, the parameters are updated by

$$\begin{aligned} m_t &= \mu \cdot m_{t-1} + (1-\mu) g_t \\ \theta_{t+1} &= \theta_t - \eta \cdot m_t \end{aligned} \tag{2}$$

The momentum term increases for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions. As a result, faster convergence and reduced oscillation are achieved. In practical usage, μ is typically set to 0.9 or a similar value.

Nonetheless, similar to SGD, SGDM also suffers from the "consistent learning rate" problem.

2.3 Adagrad

Adaptive subgradient methods (Adagrad) [2] adapts the learning rate to parameters θ, adopting smaller learning rate for frequently occurring features and larger for rarely occurring ones. This decorrelation of input features makes it a more suitable method for large sparse data than SGD and SGDM. It's update function is

$$\begin{aligned} n_t &= n_{t-1} + g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{n_t + \varepsilon}} \cdot g_t \end{aligned} \tag{3}$$

where ε is a small number to prevent dividing by zero.

Although [3] states that most implementations use a default value of 0.01, the selection of learning rate η still affects the convergence rate and final accuracy. In our experiment, the default $\eta = 0.01$ gives a rather bad training performance. Besides, as n_t monotonically increases, the gradient term $\frac{\eta}{\sqrt{n_t+\varepsilon}}$ approaches zero when t becomes large.

2.4 RMSprop

To tackle with the disadvantage of Adagrad, a sliding-window-like approach is used to discard those gradients appears too long before. Its core concept is dividing the learning rate for a weight by a running average of the magnitudes of recent gradients for that weight, as stated in [4]. In each iteration only the gradient calculated in the iteration step is used for parameter updating instead of a set of previous gradients which reduces the consumption of memory and calculation complexity simultaneously.

$$\begin{aligned} \mathbb{E}\left[g^2\right]_t &= \gamma \mathbb{E}\left[g^2\right]_{t-1} + (1-\gamma) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\mathbb{E}\left[g^2\right]_t + \varepsilon}} \cdot g_t \end{aligned} \tag{4}$$

where $\gamma \in [0,1]$ and η is the learning rate. The suggestion for γ is 0.9 and η is 0.001.

Term $\sqrt{\mathbb{E}\left[g^2\right]_t + \varepsilon}$ in (4) is usually written as $\mathrm{RMS}[g]_t$, in which RMS indicates root mean square error. This is how RMSprop got it name in the first place.

2.5 Adadelta

Developed from Adagrad at aound the same time as RMSprop independently, Adadelta [5] completely eliminates the need for global learning rate selection. It is similar to RMSprop, except for substituting η with the exponentially decaying average of parameters θ. Adadelta can be regarded as an approximation of Newton's Method with first moment gradients.

$$\begin{aligned} \mathbb{E}\left[\Delta\theta^2\right]_t &= \gamma \mathbb{E}\left[\Delta\theta^2\right]_{t-1} + (1-\gamma)\Delta\theta_t^2 \\ \mathbb{E}\left[g^2\right]_t &= \gamma \mathbb{E}\left[g^2\right]_{t-1} + (1-\gamma) g_t^2 \\ \theta_{t+1} &= \theta_t - \frac{\sqrt{\mathbb{E}\left[\Delta\theta^2\right]_{t-1} + \varepsilon}}{\sqrt{\mathbb{E}\left[g^2\right]_t + \varepsilon}} \cdot g_t \end{aligned} \tag{5}$$

Where $\Delta\theta_t = -\frac{\sqrt{\mathbb{E}[\Delta\theta^2]_{t-1}+\varepsilon}}{\sqrt{\mathbb{E}[g^2]_t+\varepsilon}} \cdot g_t$.

Equation (5) can be written in a concise form as RMSprop:

$$\theta_{t+1} = \theta_t - \frac{\mathrm{RMS}[\Delta\theta]_{t-1}}{\mathrm{RMS}[g]_t} \cdot g_t \tag{6}$$

The only hyperparameter in (6) is the initial "learning rate" $\mathrm{RMS}[\Delta\theta]_0$, which is usually set to 1.

2.6 Adam

Adaptive Moment Estimation (Adam) [6] is another method that computes adaptive learning rates for each parameter. Adam keeps an exponentially decaying average of past gradients, similar to momentum, in addition to their squares. Its update rule is:

$$\begin{aligned} m_t &= \mu m_{t-1} + (1-\mu) g_t \\ n_t &= \nu n_{t-1} + (1-\nu) g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\mu^t} \\ \hat{n}_t &= \frac{n_t}{1-\nu^t} \\ \theta_{t+1} &= \theta_t - \frac{\eta}{\sqrt{\hat{n}_t + \varepsilon}} \cdot \hat{m}_t \end{aligned} \tag{7}$$

m_t and n_t can be respectively regarded as the mean and variance of gradients. As they are initialized to 0, i.e. $m_0 = n_0 = 0$, it is likely that in the first steps the calculated m_t and n_t are smaller than their true value. In this case, a bias-correction step is added to the calculation, as shown in the second and third equations in (7).

In practical usage, learning rate η is often selected according to the context. μ and ν are set to 0.9 and 0.999 as default value.

2.7 Potential Challenges

Although RMSprop, Adadelta and Adam achieves higher convergence speed, chances are they may not converge to a valley but oscillate around somewhere [7]. The reason for this is that unlike Adagrad and SGDM in practical usage, in which researchers manually apply a monotonically decreasing learning rate to the original algorithm, the learning rate for RMSprop, Adadelta or Adam varies according to the input data batch and cannot be predicted. There are other occasions that their learning rate becomes too small to learn anything from data. In some researches, Adam is substituted by SGD or SGDM with relatively large learning rate at later learning epochs to avoid the above case.

3 Dataset and Network Structures

3.1 Dataset

MNIST handwritten digit database is used in this project. This is a small dataset wildly used for algorithm performance evaluation in researches. It contains 60000 labeled handwritten digit samples for training and 10000 for testing. Digit samples are stored as 28×28 pixels black and white pictures.

3.2 Network Structures

In this project we intend to test these optimization algorithms on different networks, including Neural Network (NN), Convolutional Neural Network (CNN) and Long Short-term Memory (LSTM) Network.

NN The NN we use is very simple, containing only an input layer with 784 nodes, a fully connected layer with 500 nodes and an output layer with 10 nodes representing output classes. ReLU is used as activation function and logarithmic softmax is used to do classification.

CNN CNN we use contains two 5×5 convolution layer with depth 20 and 50 respectively. Two 2×2 max pooling layers and a fully connected layer with 500 nodes are included. The rest is the same as NN.

LSTM Our LSTM only contains a bi-directional LSTM layer with 64 hidden nodes. The rest is the same as NN.

4 Results and Analysis

In this section, we compare the performance of different optimization algorithms on NN, CNN and LSTM respectively. In each comparison identical learning rate is used for each algorithm, which is set to a value such that the differences between algorithms can be clearly observed. Other hyperparameters are set to suggested values indicated in section 2. Fig.s 1 – 5 shows our experimental results.

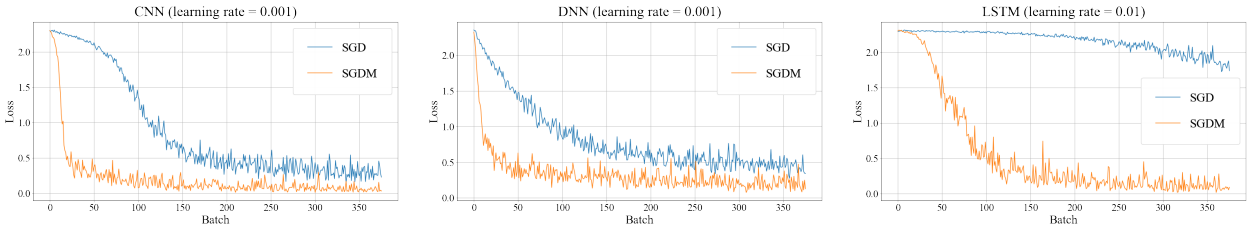

Figure 1: SGD – SGDM

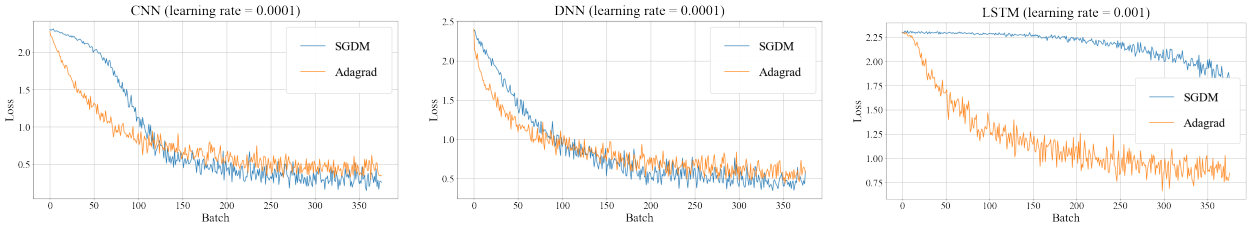

Figure 2: SGDM – Adagrad

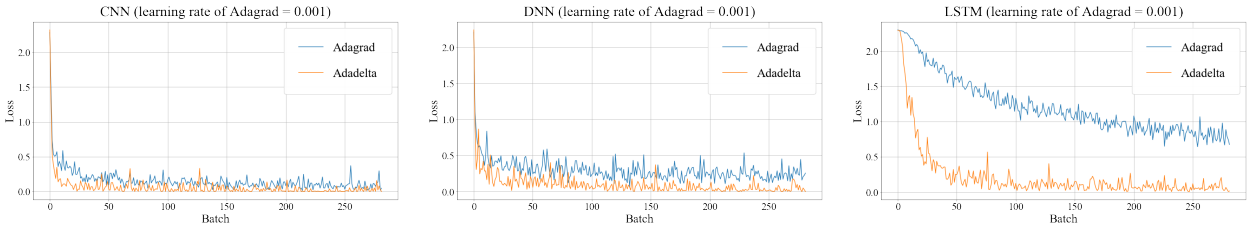

Figure 3: Adagrad – Adadelta

From these figures we can observe that optimization algorithms have relatively consistent behaviors in the context of difference network structures, which proves that network structure is not a primary factor that affects the performance of optimization algorithms.

As expected, Fig. 1 shows that the momentum term contributes significantly to the overall convergence rate. Although it is unwise to discuss about the ultimate prediction accuracy on training set since SGD has not achieved a stable stage in Fig.1, we can safely draw the conclusion that at least in the case of MNIST SGDM is a better optimization algorithm than SGD.

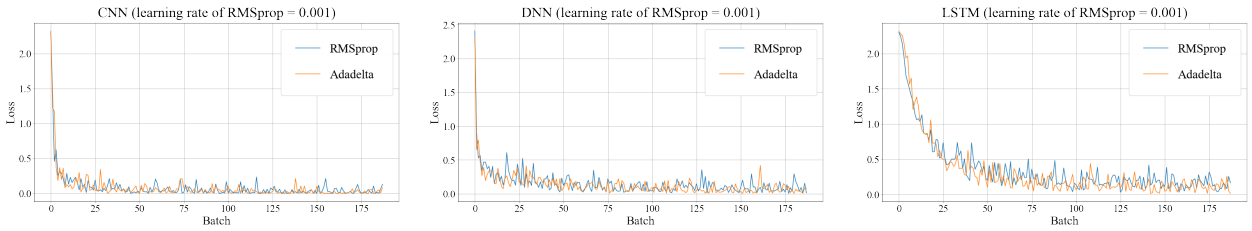

Figure 4: RMSprop – Adadelta

Figure 5: Adadelta – Adam

When the learning rate is set to a small value, the character of Adagrad becomes obvious. The adaptive learning rate gives Adagrad a higher convergence rate than SGDM. However, as iteration increases, Adagrad's learning rate becomes so small that it is not able to "learning anything new" from dataset, making its prediction accuracy eventually exceeded by SGDM, as shown in Fig. 2.

Adadelta is a milestone of the development of optimization algorithms. Fig. 3 clearly indicates that Adadelta not only achieves a higher convergence rate but also gives a better final training accuracy. In the case of LSTM, if the learning rate for Adagrad is carefully selected, Adagrad may yield better performance than the rightmost curve in Fig. 3, which highlights even more the Adadelta's merit of not depending on a fixed learning rate.

RMSprop and Adadelta are so similar that it is hard to notice any difference between them in Fig. 4, although Adadelta seems slightly better than RMSprop in LSTM, just by a tiny bit.

Adding back the momentum term, Adam does have a more stable convergence than adadelta, either in NN, in DNN or in LSTM, as shown in Fig. 5, making it currently the most popular optimization algorithm in the world.

In conclusion, give a new scenario where one is not sure which optimization algorithm to use, Adam should be the first one he tries. Although there is a potential risk dictated in section 2.7, one can easily tackle with it by switching optimization algorithm halfway through the training process.

5 Conclusions

In this project, we explore the state-of-the-art gradient descent algorithms and combine them with several neural networks. Based on the MNIST dataset, we apply NN, CNN and LSTM model to solve the problem. Here we enroll different gradient descent algorithms, from the development of SGD to SGDM, SGDM to Adagrad, Adagrad to RMSprop and Adadelta, Adadelta to Adam. Then we get the corresponding performance of different algorithms. In this way we get the sense of difference of gradient descent algorithms in different scenes from theoretical and experimental aspects.

6 Teamwork and Contributions

- Yufeng Yang: Analysis of algorithms, project organization, proposal and poster composing.
- Yinghao Li: Analysis of algorithms, programming, proposal, poster and final report composing.

- YungAn Hsieh: Programming, proposal, poster and final report composing.
- Xin Chen: Analysis of algorithms, proposal and poster composing.
- Xuan Guo: Poster composing.

Our schedule is in Fig. 6, and our project runs very well as our schedule in the proposal. In this project, we benefit a lot from the concepts of machine learning and neural network. We want to thank professor Bloch for giving us the great introduction to statistical machine learning and we believe what we learned from the ECE 6254 course will benefit us a lot in the future career and research.

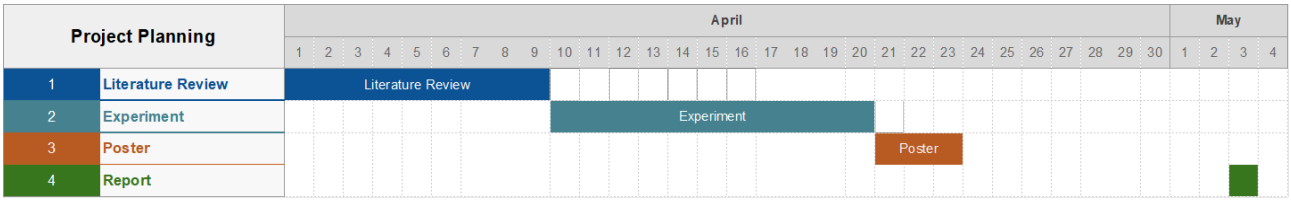

Figure 6: Project Planning

References

[1] N. Qian, "On the momentum term in gradient descent learning algorithms," *Neural networks*, vol. 12, no. 1, pp. 145–151, 1999.

[2] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *Journal of Machine Learning Research*, vol. 12, no. Jul, pp. 2121–2159, 2011.

[3] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[4] T. Tieleman and G. Hinton, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.

[5] M. D. Zeiler, "Adadelta: an adaptive learning rate method," *arXiv preprint arXiv:1212.5701*, 2012.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[7] S. J. Reddi, S. Kale, and S. Kumar, "On the convergence of adam and beyond," 2018.