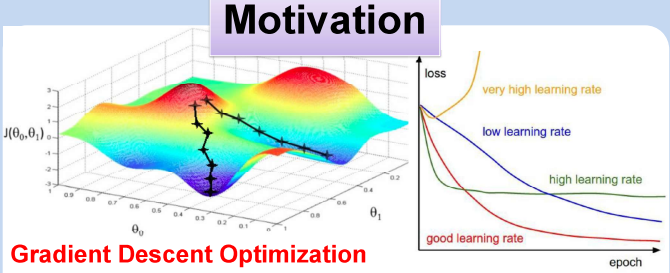
Experimental Study of Gradient Descent Optimization Algorithms in Deep Neural Network

Georgia Institute of Technology

Yufeng Yang, Yinghao Li, YungAn Hsieh, Xin Chen, Xuan Guo

Motivation



Gradient Descent Optimization

Theoretical analysis of various representative GD algorithms: how did they evolve? How to tune the learning rate?

Experimental studies of these algorithms on benchmarks: how do they perform? How are the designs reflected?

Neural Network

GD optimization algorithm plays a crucial role in neural networks. How various networks influence the performances?

Background and Overview

Optimization Problem: $\min_{\mathbf{x}\in\mathbb{R}^d} f(\mathbf{x})$ where $f:\mathbb{R}^d\to\mathbb{R}$.

Gradient Descent: $\mathbf{x}_{j+1}=\mathbf{x}_j-\eta\nabla f(\mathbf{x}_j)$, where $\eta>0$ is *stepsize*

Newton's method: $\mathbf{x}_{j+1}=\mathbf{x}_j-\left[\nabla^2 f(\mathbf{x}_j)\right]^{-1}\nabla f(\mathbf{x}_j)$, where $\nabla^2 f(\mathbf{x})$ is the Hessian. (This is infeasible for high-dimensional data).

Our Approach:

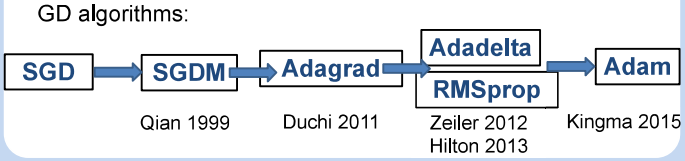
Dataset: MNIST Database

Network Structure: CNN (c1 5x5x20; p1 2x2; c2 5x5x50; p2 2x2; fc1 500; fc2 10)

Metrics: Training Loss, Training Accuracy

Test bed: Python with pytorch

GD algorithms:



Theoretical analysis

SGD: $\theta=\theta-\eta\cdot\nabla_\theta J(\theta;x^{(i)};y^{(i)})$ — Perform a parameter update for each x_i, y_i

SGDM: $v_t=\gamma v_{t-1}+\eta\nabla_\theta J(\theta)$, $\theta=\theta-v_t$ — The addition of momentum

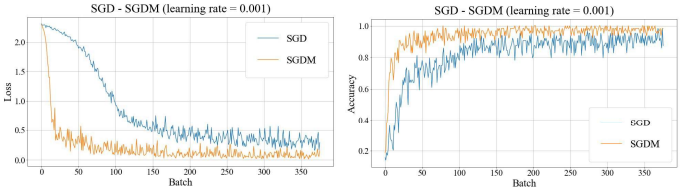
Adagrad $\theta_{t+1,i}=\theta_{t,i}-\frac{\eta}{\sqrt{G_{t,ii}+\epsilon}}\cdot g_{t,i}$ — Different learning rate for infrequent and frequent parameters

RMSprop: $E[g^2]_t=0.9E[g^2]_{t-1}+0.1g_t^2$, $\theta_{t+1}=\theta_t-\frac{\eta}{\sqrt{E[g^2]_t+\epsilon}}g_t$ — Exponentially decaying influence of past squared gradients

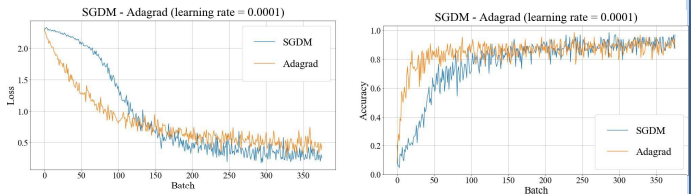
Adadelta: $E[\Delta\theta^2]_t=\gamma E[\Delta\theta^2]_{t-1}+(1-\gamma)\Delta\theta_t^2$, $\Delta\theta_t=-\frac{RMS[\Delta\theta]_{t-1}}{RMS[g]_t}g_t$ — Approximate with RMS of updates on previous step

Adam $m_t=\beta_1 m_{t-1}+(1-\beta_1)g_t$, $\hat{m}_t=\frac{m_t}{1-\beta_1^t}$, $\theta_{t+1}=\theta_t-\frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t$ — Keeps an decaying average of past gradients, similar to the momentum.

Experimental results 1

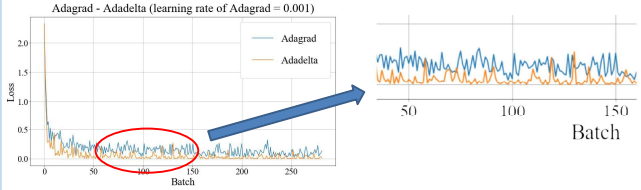


SGDM converges faster and achieves a lower loss and a higher accuracy than SGD, because of the momentum term.

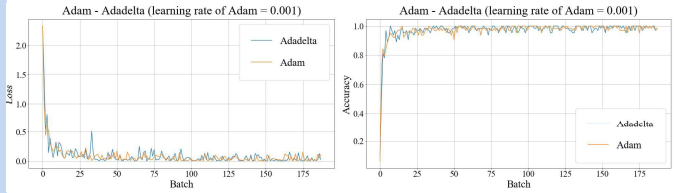


Adagrad converges faster than SGDM at the beginning. But its learning rate approaches 0 when iteration time becomes large.

Experimental results 2



Adadelta achieves a lower loss and a higher accuracy, because it decays the average of all past squared gradients.



Adadelta and Adam are similar algorithms -- although theoretically Adam is more "stable" than Adadelta, especially at the beginning, due to the existence of bias-correction procedure and momentum term.

Conclusions

	Advantage	Disadvantages
SGD	Higher training speed than Gradient Descent	Learning rate is fixed during the whole training process
SGDM	Reduces oscillation and accelerate convergence	Learning rate is fixed during the whole training process
Adagrad	Adaptive learning rate for each parameter	Learning rate approaches zero when iteration increases
RMSprop	Exponentially decay average of squared gradients	Ignores the momentum that may still cause oscillation
Adadelta	No need to tune learning rate, similar to Adagrad	Ignores the momentum that may still cause oscillation
Adam	Keeps an average of past gradients as momentum	May not converge

Contributions of team members:

Yufeng Yang: Analysis of algorithms and organizations;
Yinghao Li: Study of algorithms and part of programming;
YungAn Hsieh: Experimenting and part of programming;
Xin Chen:
Xuan Guo: