Accelerating Deep Learning Optimization in Mocha.jl

Student: Alexander Amini
Proffessors: Alan Edelman, David Sanders

***Abstract*— The process of iteratively minimizing a cost function, as in machine learning, is typically done through Stochastic Gradient Descent (SGD). Even though this technique has existed and been widely used for decades, recent advances in deep-learning and other large scale learning problems have brought about new challenges to feasibly using SGD for very large scale problems. While there have been advances in accelerating gradient descent, minimal work has been done to investigate their effectiveness in a deep or any non-convex setting. In this project, I will improve the state of the art Julia deep learning platform, Mocha.jl, to support various different types of optimization schemes. By having a modular programing approach the optimizers created as part of this project will be able to be "plugged into" existing deep learning code by changing only a single line of code, but still achieve accelerated training speedups.**

I. Mocha.jl

Mocha.jl [1] is a deep learning framework for Julia that enables quick prototyping of various deep architectures, such as Convolutional Neural Network (CNN), Recurrent Neural Networks (RNN), Autoencoders, and more. It was inspired by, and shares much of the same syntactical features of the popular C++ deep learning library, Caffe [2].

Deep Neural Networks are built in a sequential pattern, one layer at a time, stacked on top of and feeding into another layer. The collection of weights and parameters of the whole network are aggregated together and optimized to best fit the training data. Stochastic gradient descent (SGD) is widely used to optimize these parameters for a number of reasons, from having theoretical guaranteed convergence rates, to being relatively simple to implementation. However, despite these advantages a wide topic of machine learning in the past decade has been to develop accelerated gradient descent methods, mainly in the context of convex optimization problems.

In this project, I will extend the Mocha.jl framework to offer various accelerated optimization algorithms. These algorithms will be implemented in modular setting so that they can be inserted straight into existing deep Mocha architectures by changing only a single line of code and automatically achieve speedups

II. Previous Work

In machine learning, given a training set of m inputs, $x_1, x_2, ..., x_m$ and labels $y_1, y_2, ..., y_m$, we define a loss function $\psi_i(\theta)$ where θ is the vector of weights parameterizing a function that estimates the ith label given the ith sample as input. The empirical loss, Ψ, of this problem can now be expressed as the sum loss over all m data points.

Stochastic Gradient Descent (SGD) [4] is a method to minimize $\Psi(\theta)$ by identifying the optimal weight vector, θ. In this algorithm, we randomly pick a data sample, (x_i, y_i), from the training set and use the gradient at this sample to update the weight vector as follows:

$$\Psi(\theta) = \sum_{i=1}^{m} \psi_i(\theta) \tag{1}$$

$$\theta_{t+1} = \theta_t - \gamma \nabla \Psi(\theta) \tag{2}$$

where γ is the learning rate, representing the magnitude to step in the direction of the negative gradient.

Mocha.jl provides support to solving deep neural networks utilizing the SGD update equation presented above. Furthermore, they also provide a slightly modified extension using momentum, known as Nesterov's Gradient Descent algorithm [3]. These solvers can used to optimize the network parameters by initializing them as follows:

```
method = SGD(); # init SGD
method = Nesterov(); # init Momentum
```

Using Mocha.jl, I have already started to build networks, and datasets to test the performance of the different optimizers. First, to generate a dataset of linear data, perturbed with normally distributed Gaussian noise, then to build a single Perceptron model in order to estimate a line of best fit from the data θ = (slope,y-intercept). In addition I've created a `Recorder` type to record some of the internal Julia convergences of the model.

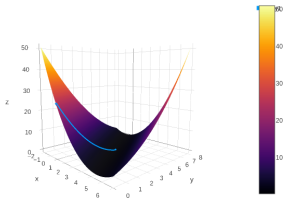

Fig. 1. Convergence of the SGD algorithm over time (blue line), descending into the global minimum over the topology of $\theta = (w, b)$ (slope and y-intercept).

III. Proposal

One of the main problems of SGD and Nesterov's Momentum algorithm is the issue of fixed learning rates (γ). As the optimizer descends, the learning rate should also be decreasing to ensure stable convergence patterns, of which I have already begun to see as part of my preliminary testing. In this project, I will implement several *adaptive* gradient descent algorithms (such as AdaGrad [5], and AdaDelta [6]). The code created will be modular such that a Mocha.jl user can test and evaluate their existing algorithms by simply changing a single line of code, as shown above). Since these algorithms adaptively modify learning rates for each dimension of the data, they have theoretically faster convergence rates than the vanilla SGD. Therefore, the aim of this project is for existing Mocha.jl users (as well as new users) will be able to achieve training speedups by using a wider array of faster accelerated optimization schemes.

REFERENCES

[1] Mocha.jl, pluskid. GitHub repository. https://github.com/pluskid/Mocha.jl

[2] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional Architecture for Fast Feature Embedding (2014). arXiv preprint arXiv:1408.5093

[3] Nesterov, Yurii. "A method of solving a convex programming problem with convergence rate O (1/k2)." Soviet Mathematics Doklady. Vol. 27. No. 2. 1983.

[4] Robbins, Herbert, and David Siegmund. "A convergence theorem for non negative almost supermartingales and some applications." Herbert Robbins Selected Papers. Springer New York, 1985. 111-135.

[5] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. Journal of Machine Learning Research, 12(Jul), 2121-2159.

[6] Zeiler, M. D. (2012). ADADELTA: an adaptive learning rate method. arXiv preprint arXiv:1212.5701.