Deep Learning Project 3

Group 2
Hong Kong University of Science and Technology

December 15, 2017

Abstract

In this project we have implemented ADMM optimization methods for the convolution layers. For the convolution layers, we considered converting a convoluted matrix into a dense toeplitz matrix. The weights updates of kernels are done by solving a system of equations by sampling equations obtained from Toeplitz matrix. We found that our method worked faster than author's implementation since they only considered the pseudoinverse of the matrix. We used MNIST digit dataset (odd-even classification) and found that our method gives 89.6% over traditional backprop method 79.6%.

1 Introduction

ADMM (Alternating Direction Method of Multipliers) is the divide and conquer approach to solve a convex optimization problem [Taylor et al., 2016]. For this project, we have implemented ADMM for Neural Networks, with our added tweak on sampling full rank toeplitz matrix for convolutions.

Notations for this report are: W_l is the weight filter for layer l, z_l is the outputs for each layer, a_l are activations. Rest are the hyperparameters. We will focus on details of solving of sub-problems updates of the minimization problem, in the following sections.

2 Weight Update

For each layer l, the optimal solution would minimize $||z_l - W_l a_{l-1}||^2$. The authors approached this problem by taking the pseudoinverse of a_{l-1}, which

is a^*_{l-1}. In our opinion, taking pseudoinverse of a very tall matrix (in this case Toeplitz matrix [Gray et al., 2006] of a_{l-1}) is an overkill and inaccurate. Hence, we approach this problem in a different manner.

Firstly, we write the convolution-kernel operation in the form of equation matrix, let's say $a_{l-1}W_l = z_l$. where W_l is the unravelled kernel and a_{l-1} is the Toeplitz matrix of the convolution layer. Here if $k = rank(W_l)$, then we only choose $k * k$ rows from the augmented $a_{l-1}W_l|z_l$ block and solve them using standard linear equation solvers. We found this to be a lot faster than taking traditional pseudoinverse.

3 Activation Update

Here we tweaked the original formulation which could work well for the ReLU function.

$$a_l = \begin{cases} (\beta_{l+1}W_{l+1}^T W_{l+1} + \gamma_l I)^{-1}(\beta_{l+1}W_{l+1}^T z_{l+1} + \gamma_l z_l), & \text{for } z > 0 \\ (\beta_{l+1}W_{l+1}^T W_{l+1} + \gamma_l I)^{-1}(\beta_{l+1}W_{l+1}^T z_{l+1}), & \text{otherwise} \end{cases}$$

4 Output Update

For this step, we need to minimize,

$$\min_z \gamma_l ||a_l - h_l(z)||^2 + \beta_l ||z - W_l a_{l-1}||^2$$

For the ReLU function, this function would be piecewise linear. Just putting the gradients to zero, we get,

$$\frac{\partial}{\partial z}\left(\gamma_l ||a_l - h_l(z)||^2 + \beta_l ||z - W_l a_{l-1}||^2\right) = 0$$

which gives,

$$z_l = \begin{cases} \frac{\gamma_l a_l + \beta_l w_l a_{l-1}}{\gamma_l + \beta_l}, & \text{for } z > 0 \\ w_l a_{l-1}, & \text{otherwise} \end{cases}$$

5 Experimental Setup

We used MNIST dataset for classification. However, due to time constraints we resorted to binary classification, rather than multiclass classification. Hence, we only classified even against odd numbers classification. Although we found that this was harder than numerical classification, since set of pixels were trying to generalize the image, making it even hard to classify.

For vanilla backprop, we did 2-layer implementation of the neural nets.

For the ADMM, we also implemented 2-layer with same hyperparameters as vanilla backprop. We used hinge loss as reported in the original paper, $\beta = 10$ and $\gamma = 1$. We used no fully connected layers for this purpose. We just used 3 convolutional stages, followed by max-pooling then ReLU. Here, convolution stands for Toeplitz matrix - kernel matrix multiplication. Thus, we avoid the complications of im2col here. We also took help from existing implementations of this paper [Dongzhuoyao, 2017].

6 Final Remarks

	Testing Accuracy
Vanilla Backprop	79.6 %
ADMM	**89.6** %

Table 1: Tesing Accuracy for the experiments

The results of our experiments are reported in Table 1. Clearly, ADMM outperforms vanilla backprop method. We also feel that our implementation of ADMM is faster than the original solution. However, our implementation of ADMM was buggy since we have to re-run it couple of times so that it does not converges on local minima (last layer is all zeros). Due to time constraints, we cannot solve that bug otherwise we could empirically show that our method is better.

We feel that ADMM is not appropriate for complex architecture since inverting a large matrix is unreasonable and random initializations are unreliable. However, in our implementation we show how this problem could be partially alleviated through sampling into a full-rank matrix.

References

[Dongzhuoyao, 2017] Dongzhuoyao (2017). Github implementation.

[Gray et al., 2006] Gray, R. M. et al. (2006). Toeplitz and circulant matrices: A review. *Foundations and Trends® in Communications and Information Theory*, 2(3):155–239.

[Taylor et al., 2016] Taylor, G., Burmeister, R., Xu, Z., Singh, B., Patel, A., and Goldstein, T. (2016). Training neural networks without gradients: A scalable admm approach. In *International Conference on Machine Learning*, pages 2722–2731.