IDP Report

September 27, 2016

1 Short sum up

Up till now, we focus our work on investigating and implementing the idea proposed by the paper.

1.1 Theoretical aspect

We have derived the formulas by ourselves. It turned out the algorithm is mathematically correct. The most ambiguous part should be the intuition of λ and the way they work with it.

1.2 Implementation aspect

The main obstacle we met with was that the Hinge loss function on the paper is not totally applicable for Mnist because it is a multi-class dataset. Focusing on the issue we tried several different approaches. Altogether we implemented the program with hinge, mean square as well as softmax as loss function. We finally also decided to stick to hot-one representations of image labels for two reasons:
1. It seems that it is the common method. According to the tutorial on tensor flow, they uses softmax and applied hot-one labels.
2. Besides, the method provided us with a reasonable result already.

Hinge Loss When using Hinge loss, we also tried to divide the mnist dataset into two classses, which are even and odd, the result wasn't good. So we still stick to the hot 1 and trained the neural network with hinge loss. Finally, we can achieve a reasonable result.

Mean Square When we used the mean square as the loss function, we found two interesting things. The λ didn't play an important role but the ϵ, which was used to initialize the weight matrix, had a big impact. Meanwhile, if we kept β and γ same instead of updating them every iteration, the result will be better.

Softmax The main problem of using softmax as the loss function is that we can't solve the minimization problem to update z in last layer with a linear

system. So we tried to use both gradient descent and proximal gradient to solve the subproblem. Both of them worked out and achieved reasonable results. As same as the Mean square, keeping β and γ same can achieve a higher predict accuracy.

1.3 Accuracy comparison between different loss functions with Mnist dataset

Training set: 60000
Testing set: 10000

Loss function	accuracy
Hinge	86.89%
Mean Square	87.07%
Softmax with proximal gradient	87.28%
Softmax with gradient descent	87.23%

2 Tuning Hyperparameters in the case of using softmax and proximal gradient

We picked softmax as our loss function, which minimizes z update with proximal gradient. We tried to tune its hyperparameters to see what we can get. The results and our guessing is demonstrated as follows.

2.1 Comparison of different ϵ

Loss function: Softmax + proximal gradient
Neural Network: 1 hidden layer with 300 units
Iteration times of proximal gradient: 25
Step size of proximal gradient: 0.01
Update β and γ: No

(training set, test set)	accuracy($\epsilon = 1e-1$)	accuracy($\epsilon = 1e-4$)
(600, 100)	14%	34%
(6000,100)	78%	87%
(10000,100)	83%	85%

Conclusion and Guess Smaller ϵ will acheive a better result. In Zhenzhang's test, sometimes if the ϵ is too big, we cannot achieve an overfitting with small same training and testing set.
For the z_L (the z in the last layer), updating it includes the λ item. We guess that there is a trade-off between these three items. If ϵ is too big, theλ item will have a higher weight. Then we have to spend some accuracy to achieve a smaller objective cost.

2.2 Comparison of different network structure

ϵ : 1e-4

Loss function: Softmax + proximal gradient
Iteration times of proximal gradient: 25
Step size of proximal gradient: 0.01
Update β and γ: No

training, testing	[300]	[300,150]	[150,300]
(6000,100)	87%	85%	85%
(10000,100)	86%	85%	86%
(20000,100)	88%	87%	88%

Conclusion and Guess 1 layer and 300 units network should be enough for the MNIST. It's highly possible if we add more trainning data and the 2-layer network will acheive a higher result. Unfortunately, due to the limitation of our computers, using 60000 training data is extremely slow.

2.3 Comparison of iteration times

ϵ : 1e-4
Loss function: Softmax + proximal gradient
Neural Network: 1 hidden layer with 300 units
Step size of proximal gradient: 0.01
Update β and γ: No

training, testing	25	35	50	150	10	60
(6000,100)	87%	87%	86%	86%	85%	
(10000,100)	85%	85%	88%			85%
(15000,100)	87%		87%			87%
*(15000,100)	86%		86%			

*: 2 hidden layers and [300,150]

2.4 Comparison of step size

ϵ : 1e-4
Loss function: Softmax + proximal gradient
Neural Network: 1 hidden layer with 300 units
Iteration times of proximal gradient: 25
Update β and γ: No

training, testing	1e-2	1e-3	1e-1	1	10	1e-4
(6000,100)	87%	85%	87%	86%	85%	85%
(10000,100)	87%	85%	87%			

Conclusion and Guess Choosing step size as 0.01 and iteration times as 25 seems enough. But it's not sufficient. Because the samller step size may need more tieration times. Generally speaking, the step size and the iteration time are not very important for the whole algorithm.

2.5 Comparison of updating β and γ

ϵ : 1e-4

Loss function: Softmax + proximal gradient
Neural Network: 1 hidden layer with 300 units
Iteration times of proximal gradient: 25
Step size of proximal gradient: 0.1

training, testing	*1	*1.05	*0.65	*0.45	*0.35
(6000,100)	87%	86%	86%	86%	
(10000,100)	87%	85%		86%	85%

Conclusion and Guess Keeping β and γ is better.

3 Conclusion

To conclude, the testing result showed the best accuracy our neural network with ADMM can achieve is around 87%, which is far away from the result on the Mnist website.

Based on the result, we have several guess on how to improve accuracy. One guess is that we might improve it by increasing the training dataset(such as image distortion and rotation as Thomas mentioned sometime).

According to the Mnist dataset description, the first 5000 testing images are cleaner. However, it turns out that the later 5000 images gives much better result. We trained the network with 60000 images and then tested it with first 5000 images which gave us 82% accuracy, while the later 5000 images led to 90% accuracy.

On the other hand, from our experiment, we found that different loss functions have only a little impact on the accuracy. However, with Hinge and Mean Square, we can update the z_L by linear system, while Softmax requires more complicated minimization subproblem. We tried to figure out the conjugate of it (we googled the convexity of softmax and someone has proved it's convex), but failed to get the result.

Anyway, we are aiming at examining whether ADMM works for the neural network. We believe that it works, only a satisfactory performance requires more investigation on the hyper parameters. We also neglect the performance such as speed or memory at present. To further tune the hyper parameters, we believe we need to decide the loss function first.