IDP Report

September 27, 2016

1 Description of What We did

HingeLoss We managed to use hinge loss as the loss function. Firstly, we divided the mnist dataset into two classses, which are even and odd, the result wasn't good. So we still used 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 weights had a big impact. Meanwhile, if we kept β and γ as same instead of updating them every iteration, the result will be better.

Softmax The problem of softmax when we use it as the loss function is for updating the z in last layer, we can't solve the minimizing problem with a linear system. So we tried to use gradient descent and proximal gradient to solve the subproblem. As same as the Mean square, keeping β and γ same can achieve a higher predict accuracy.

2 Comparison of different parameters

2.1 Comparison for 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: 0.1
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 acheive 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: 0.1
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 for proximal gradient

ϵ : 1e-4
Loss function: Softmax + proximal gradient
Neural Network: 1 hidden layer with 300 units
Step size of proximal gradient: 0.1
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 for proximal gradient

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 General conclusion

Generally, our network with ADMM can only achieve accuracy around 87%. Our guess is that if we increase the training dataset(by image distortion and rotation), the result may be better.
According to the mnist dataset description, the first 5000 testing image are cleaner. But in our testing, the last one can acheive 90% accuracy while first only has 82% accuracy.
The different loss functions don't have a big impact on accuracy in this test. But for hingeloss and mean square, we can update the z_L by linear system. Softmax requires more complicated calculation. We tried to figure out the conjugate of it (we googled the convexity of softmax and someone has proved it's convex), but failed.

In conclusion, ADMM can work for the neural network. The performance depends on the hyperparameters. We haven't considered the speed yet. The first idea is that we need to decide the loss function first.