Team 得意得意的佑佑

MLDS HW1 Report

Group Information

Group Name: 得意得意的佑佑

StudentID	Name	Contribution
B01902015	陳柏佑	(25%) Survey, part of report.
B01902028	周書禾	(25%) part of report, data preprocessing, smoothing, basic functions implementation.
B01902032	江東峻	(25%) Optimization(large improvement in performance), beautify code.
B01902078	陳瀚浩	(25%) Most of the coding job, theano implementation.

What have we done?

Data Structure and algorithm design

Data Structure:

- numpy
  we use theano as the library to implement hw1. All matrix and vectors are in numpy form with dtype="float32" in order to make it runnable on GPUs.
- Three-layer DNN
  Using fbank dataset with 69 features, we build a three-layer DNN with 256 neurons.

Algorithms:

- Activation function
  sigmoid, hyperbolic tangent, ReLu.
- Cost Functions
  softmax, MSE
- Optimization
  Adagrad
  Momentum
- Smoothing
  We actually do the "post-processing" of the predicted result.
  The intuition that the consecutive frame tend to have the same label.
  If the frame's former three and latter three frames have more than 4 same label, I assume that this frame is also the same.

The comparison results of all those algorithms will be discussed below.

Team 得意得意的佑佑

Data Preprocessing

We noticed that the order of the train data and its labels are not the same. To deal with the situation, we sort both the train data and the label data with key as the sentence ID. We also make some subsets of the dataset to let testing easier. There are

Validation_data_50000, Validation_label_50000:
50000 data and labels extracted randomly.

ValidationData, ValidationLabel:
225603 data and labels extracted randomly.

TrainData, TrainLabel:
All data except for the ones in ValidationData and ValidationLabel.

Standardizing the data: Not working though, almost the same compared to the raw data.

Implementation tips and obstacles

Adagrad implementation
As the adagrad function goes, learning rate = initial learning rate / sqrt(epochs + 1), we need the information of epoch to determine the new learning rate, but it's not that easy to pass the parameter into the update function, so the tip here is to pass it from input.

Momentum implementation
Momentum parameters should be changed before the data is trained in every epochs; namely, the momentum function can not be contained in train function. If they are contained in the same function, the order will be different and thus create wrong results.

Theano mechanism
Understanding how theano really works is the hardest part. The nature that theano pack the functions together and not letting us to use condition judging like **if** and printing message inside make it hard for us to debug. It's also tough to realize the correspondence between the functions of theano.

Bugs and how to solve them

NaN problem
There are several possible cause of this problem, one of them is caused by ReLu with cross entropy, in the situation of something like 0 * log(0), which can often happen. We tried to modify the ReLu function to T.switch(z < 0.0001, 0.0001, z), but the problem is not

solved. After several trials we found out that the NaN problem will disappear if we set the batch size below some 60. This is maybe not the best way to solve the problem, but we don't have any clue how the bug was generated, and thus we have to do this trial and error instead.

Experiment Settings and Results

Experiment settings

As mentioned above, there are several data subsets for the original some one million data, and they are used to easier decide which activation functions, cost function to be used, as well as other parameters like initial learning rate and epochs to be run. It sure takes a lot of time if we use all data to train, however we can't use too few data to train as it's useless. So I decided to cut out about 1/5 of the whole data, which is about 220,000 data, to be training data when we're deciding which configuration should be applied. And the Validation_data_50000 will be treated as the validation set of these 220,000 data.

Initialization of each parameters:
Weight matrix: Gaussian variance with "magic number" sqrt(6 / dim_next + dim_now) / 3.
bias vector: 0

Other parameters with testing(without learning rate decay and momentum):

Activation Function	Hidden Layer/ Neurons	Learning Rate	Iterations	E_in (out of 255603)	E_val (out of 50000)
ReLu	**1/128**	0.1	100	50.3%	55.0%
Sigmoid	**1/128**	0.1	100	53.9%	57.2%
Hyperbolic Tangent	**1/128**	0.1	100	51.8%	55.3%
ReLu	**1/128**	0.05	100	50.9%	55.0%
ReLu	**1/128**	0.1	200	48.9%	54.0%
ReLu	**1/128**	0.15	100	49.8%	55.7%
ReLu	**1/128/ Standardized**	0.1	100	50.3%	55.5%
ReLu	**2/256**	0.1	100	42%	49%

All the testing are with cost function softmax, and the results of cost function MSE is not listed in the table because the performance were far from comparable with those with softmax. After testing, we can see that ReLu performs significantly better than sigmoid and hyperbolic tangent, and thus the testing afterwards are all changed to ReLu. We can also see that after we increase one hidden layer and add neurons to 256, the performance boost a lot. Based on this result, we decide to try to add more hidden layers and neurons rather than tuning other parameters.

Team 得意得意的佑佑

Comparisons

Further we apply Adagrad and Momentum to improve accuracy

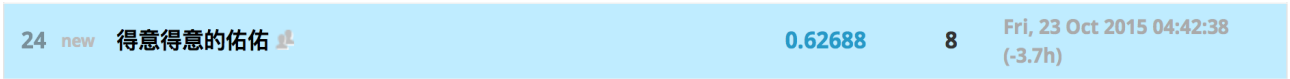
Hidden Layer/ Neurons	Adagrad	Momentum	Smoothing	Iterations	E_in(out of 1124823)
2/256	no	no	no	100	45.4%
2/256	yes	no	no	100	42.7%
2/256	yes	yes	no	100	40.7%
3/256	yes	yes	no	100	39.9%
3/256	yes	no	yes	100	39.5%

In the last trial, I forgot to use momentum but used the smoothing method and the results still improved, so smoothing did work in this situation.

Other Methods Used

Voting
Use three output files to vote for each and every answer. It's not effective though, the possible reason is that they all learn the same (or similar) way, so the voting doesn't have any influence.

Final Result



We use momentum with three-layer neurons 256 nodes in each layer
Settings:

#Iteration 100
#lambda for momentum 0.7
#Use Adagrad to decay learning rate
#smooth the final result