Team 得意的佑佑

MLDS HW1 Report

Group Information

Group Name: 得意的佑佑

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

What have we done?

Data Structure and algorithm design

Data Structure:

- numpy
  we use theano as the library to implement hw2. All matrix and vectors are in numpy form with dtype="float32" in order to make it runnable on GPUs.
- input data
  Using output of hw1 DNN with 48 features.

Algorithms:

- Activation function
  sigmoid, hyperbolic tangent, ReLu.
- Cost Functions
  softmax, entropy,
- Optimization
  RMSprop
  NAG

Team 得意的佑佑

Data Preprocessing

we have tried several training data sets in this assignment. First, using hw1 DNN model's output(softmax features) with 48 dimensions for train and test data. Second, we use train and test data from Kaggle. we also use classmate's hw1 DNN model's output to test our model. For all these data sets, we segment each sequence into length of 100 to make a consistent training data

Implementation tips and obstacles

RMSprop implementation
We use additional shared variables to store the accumulated gradients. For each epochs, we first compute the gradient for current weights, add them in accumulated gradients variables, and then use those variables to divide original gradients. Finally, we use the new gradients to update our weights.

NAG 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. And for each training, we first update weights and bias by the momentum, and then compute the gradients of updated weights and update the weights by the gradients.

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. We use clip to bound the gradients, and it works in Train10Data, but it doesn't work in TrainData.

Train Data problem
We train our model by our hw1 data and Kaggle data. We can only fit small sub data of these two data sets(Ein 25%), the Ein of them both get bad results(Ein 80%~89%). As a result, we revise our RNN model many times. Finally, we use classmate's hw1 DNN model's output as our RNN input. This training data gets Ein 10.6% from our RNN model. The result shows that our training data form can't provide good feature for RNN model.

Experiment Settings and Results

Experiment settings

Because of the difficult of RNN training, we first try to overfit the training data. We use Ein rate (0/1 error) to evaluation our overfitting methods.
‘”

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 0.1.
Bias vector: Gaussian variance with 0.1.
Memory weight matrix: 0.01 * identity matrix
Memory bias vector: 0

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

Training Data	Activation Function	learning rate decay	Hidden Layer/ Neurons	Learning Rate	Iterations	E_in (out of 1124823)
our DNN feature all	**ReLu**	**RMSprop**	**1/256**	0.0005	50	89.4%
our DNN feature 1/10	**ReLu**	**RMSprop**	**1/256**	0.0005	50	85.2%
our DNN feature 1/100	**ReLu**	**RMSprop**	**1/256**	0.0005	50	25.01%
Kaggle data feature all	**ReLu**	**RMSprop**	**1/256**	0.0005	50	89%
Kaggle data feature 1/10	**ReLu**	**RMSprop**	**1/256**	0.0005	50	85.9%
Kaggle data feature 1/100	**ReLu**	**RMSprop**	**1/256**	0.0005	50	54%
Classmate’s DNN feature all	**ReLu**	**RMSprop**	**1/256**	0.0005	10	10.6%

This table shows that our RNN model can still get good result from finely preprocessed training data and smaller data set. The problem might be our training features are too smooth in each dimensions. Although by extracting the max value of index to be predict label can get a quite good performance in hw1, it is hard for RNN to learn parameters to fit these too smooth training data.

Comparisons

Further we apply RMSprop to improve accuracy

Hidden Layer/ Neurons	RMSprop	Smoothing	Iterations	E_in (feature 1/100)
1/256	yes	no	50	83.3%
1/256	no	no	50	25%

By adding RMSprop, we can train our Ein to 25% in our hw1 DNN model's feature 1/100.

Other Methods Used

- Smoothing

After frame-wise prediction, concatenate continuous labels and count each one of them, if the count is smaller than the threshold, say 5 in our implementation, and the label before this and after this are the same, then the label is changed to that one. Then we have the second run, deleting all labels with count smaller than another threshold.