January 2018

Deep Learning for Education

Jingwei Shen
Worcester Polytechnic Institute

Deep Learning

Project Report:

Dasheng Gu
Jingwei Shen
Xinyuan Wang

January 24th 2018
Instructor: Joseph E. Beck

Abstract

The main topic of our project is: Deep Learning on Education. Deep Learning is a method of Machine Learning, it can be supervised, semi-supervised or unsupervised. The problem we are trying to solve is to tweak and improve several neural networks for prediction of student correctness on questions. The approach we adopt to solve the problem is to compare different features (drop rate, number of hidden layers, hidden nodes, with/without autoencoder, optimize function, batch size, fully connected or not) in the neural network so that we can improve AUC (Area under Curve) to get the better prediction results. The results obtained in this research include all the AUC data from all the different neural networks. Our obtained results are great basics of following working on the neural network and succeeding studies in the field of education. The highest accuracy of our single output network reached 87%. We also found the optimized parameters having efficient running time without losing performance. In conclusion, we think that the network is efficient but still need improvement in performance.

Key words

Deep learning, Neural networks, Autoencoder, Competence, Affects, TensorFlow.

Introduction

Deep learning is one of the hottest topics in computer science field. It is an algorithm simulating animal neural networks structure to discover the relationship between data distributions. Neural networks are mathematical models consists of multiple layers of nodes. When data is passed through those nodes, each of them is multiplied by a weight and added by a bias. At last, the data are tuned and outputted.

In our experiment, the process of data flowing through the network is represented by student's information going through the network and finally outputting the competence of the student or the affects of the student. From the previous approach, we have several factors that showing the competence of students: wheelspin, retention, and next problem correctness (NPC). Besides, the factors that might influence students' performance are 4 affective states: confusion, boredom, concentration and frustration.

In this project, one of the accessory library source we applied is TensorFlow. It is an open source math library published by Google, which is named by its working principles. Tensor means multidimensional array and flow means tensor is computed from one end of the dataflow graph to the other one. TensorFlow can be used in aspect like image or sound identification or other deep learning fields. Which is used in our first project for classifying pictures of alphabet. Our team did not construct the network from basic level. We used Network Builder which is a wrapped-up package written by Anthony Botelho and Seth Adjei, so we can build the customized network model with only few lines of codes.

Goals

Deep Learning (DL) has a rapid development in recent years. More and more industries are trying to use DL as their tool to improve pertinence for different user communities. Education is the foundation underlying all our efforts to build the country. For many colleges, admission is a crucial part where educators need to assess a great deal of applicants in overall. Giving an objective and accurate evaluation to these students is always a problem. Certainly, we can apply artificial intelligence to education so that educators can offer more customizable plans directly to different kind of students. Starting from this point, we built a deep learning neural network for educators to predict students' Next Problem Correctness (NPC) and other features representing their performances. In this project, our focus is trying to optimize the performance of this network on predicting students' future performance and have an overall display of their competence as outcome. To achieve the goal, we need make comparison between outcomes with modifications on the factors and other values (e.g., the numbers of hidden layer) to find a more optimal solution to have a closer prediction.

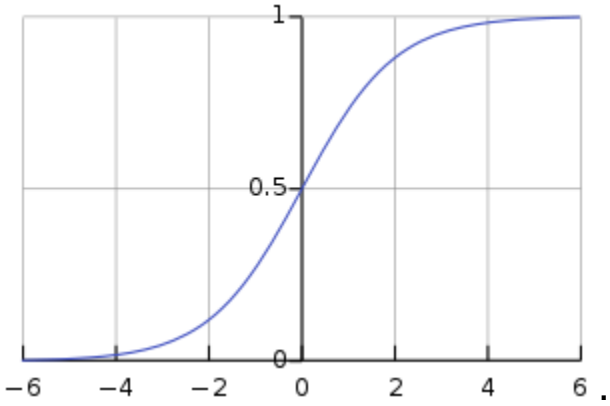
First experiment: Recognizing alphabet graph

Logistic regression

This is an experiment our team start with as a warm up to machine learning. The code implements logistic regression model from sklearn. Sklearn is a library for python already implemented several machine learning models including logistic regression model. Logistic regression can be considered as a branch of linear regression. Usually the outcome of logistic regression is binomial distribution. To transfer the independent variables to zeros and ones, we use sigmoid function as shown below:

$$f(x) = \frac{1}{1+e^{-x}}$$

And its graph is:



From the equation and its graph, we can see that it can convert any real number independent variable x to a value f(x) which is within the range of 0 to 1.

Implementation

The implementation of the model is from Udacity notMNIST tutorial. The code was provided by the tutorial mainly contains 5 steps. It first downloads data through the internet, then extract, process and store it. At last trains the network and print the result. This sklearn package has already implemented the model of logistic regression, so, we did not need to worry about that.

Conclusion

This experiment is a perfect warm up for machine learning beginners. By successfully going through this experiment, we understand that machine learning has several basic steps: read inputs, build model, train network and predict outcomes.

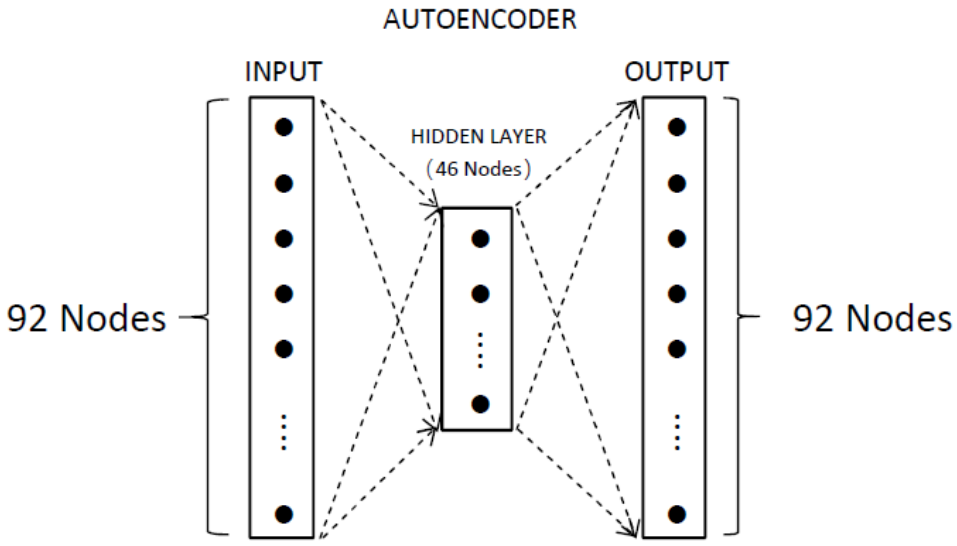
Second experiment: Predicting and optimizing single output network

In this experiment, instead of sklearn, we used TensorFlow which was mentioned above, for building models and training the network. What's more, the models are constructed by us, so we could try different models and test with different parameters. Basically, the process of training the network has 4 parts, processing inputs, building models, training the network and output predictions.

Processing inputs

The data we used is read from a csv file. They are results of actual students' survey and test answers. The data file contains more than 500000 rows of data. Each row has student's id, question number and 92 features. They are independent variables that will be fed into the network. Besides that, each row also has problem correctness, wheel spinning, first action, retention and affects. In this experiment, we use one of the five variables as output variable at a time. In our experiment, we offset problem correctness, so it represents the correctness of the next problem (NPC). Affects represent 4 columns of data, they are confusion, concentration, boredom and frustration. A student can and only be one of those 4 affective states.

Before feeding those inputs into the network, we first feed them into autoencoder. An autoencoder is a symmetric neural network which has same inputs and outputs. It is used to compress the input and reduce noise. Through our experiment, we found out that the autoencoder can increase the performance of the network by 5% to 10% and reduce the time to train a network by about 50%.

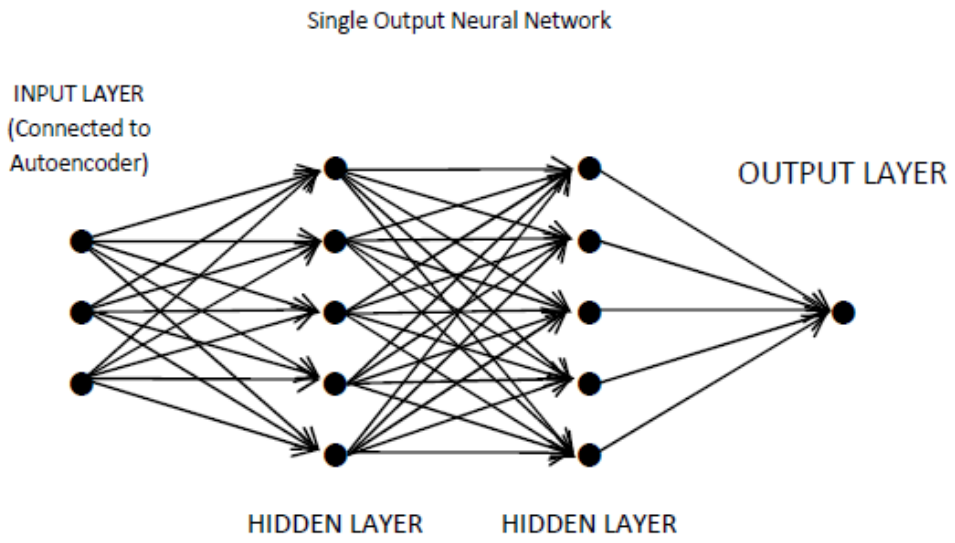


In this experiment, the autoencoder has one hidden layer with 46 nodes. The 92 features of data are fed into the input and output layer of the network. So, when the autoencoder is trained, the hidden layer in theory contains all information the inputs have with less nodes. After the autoencoder is trained, we keep the weights, bias and nodes of the autoencoder and connect the hidden layer to the main network below as inputs. This pre-training process reduces the size of inputs of the main network resulting in higher efficiency and better performance because it reduces calculation the main network need and the noise the original inputs have.

Building and training models

In this experiment, the models we build used code which handles basic model construction written by Anthony Botelho and Seth Adjei, so that we only need few lines of code to construct a fully customized network model. This allow us to focus on improving the performance of the network by matching different models with different parameters.

Neural network is an algorithm simulating animal neural system. It has nodes (hidden units) and connections corresponding to neurons and axons as shown in the figure below. The nodes are places to temporarily store data(number) and the connections are weights and biases. When the inputs go through the connections as a flow, they are multiplied by the weights and added by the biases, and the sum of those result is added and stored in the next node. When those data flows to the output layer, they are compared with the actual result. Then, the difference or so-called cost entropy is calculated, and the network will try to adjust those weights and biases to reduce the cost. When the cost is small enough, the network is well trained.



Prediction

After the network is trained, we can test the network by predicting students' competence given features as input.

Improving performance

To achieve the goal of improving the performance of the network, we mainly focused on reducing program running time and increasing the accuracy of the prediction.

We first tested the relationship between layer number with the performance. We found out that the running time of the 2-layer network is doubled compared to 1-layer network but the accuracy barely increased. 3-layer network took even longer, and we give up finishing running the network.

Second, we tried running network with different hidden units. We found out that more hidden units increase running time without increasing performance. We tested network with 50, 100 and 200 hidden units, so we thought that 50 hidden units is the optimized network among the three.

Then we compared network with different cost functions. They are adam and adagrad. We did not find any difference in their running time but the performance of adam overwhelm adagrad.

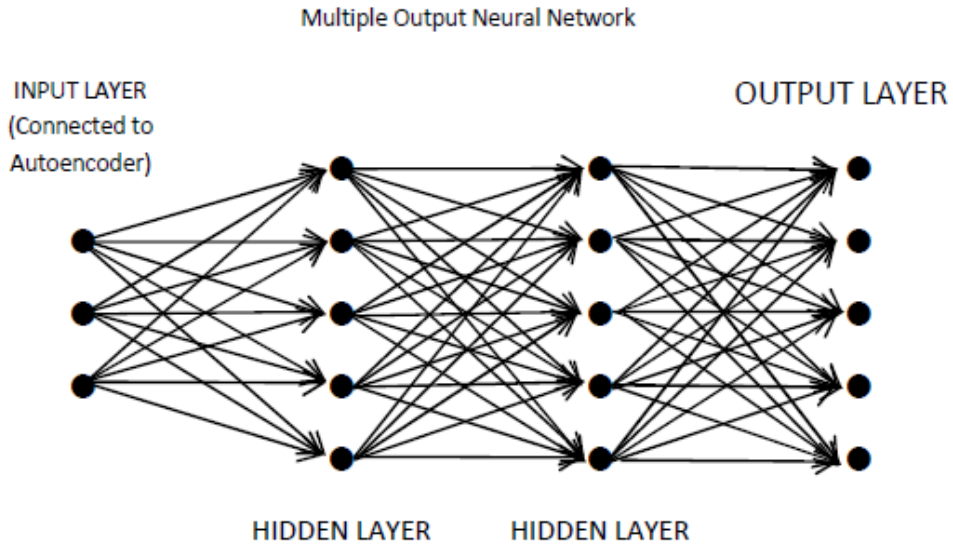
We also tested 2 different types of networks, LSTM and RNN networks. They have similar performance but RNN network cost 50% less time to train the network.

At last, we also found out that dropout rate does not affect the performance if the dropout rate is within 0.3 to 0.7. Dropout is a technique that randomly drop nodes and their connected weights and biases for neural network to reduce overfitting. An overfitted model correspond too closely to the specific data set, so the model is inaccurate when predicting with different data set.

In conclusion, the optimized model we found for this project is RNN network, 1 layer, with adam cost function.

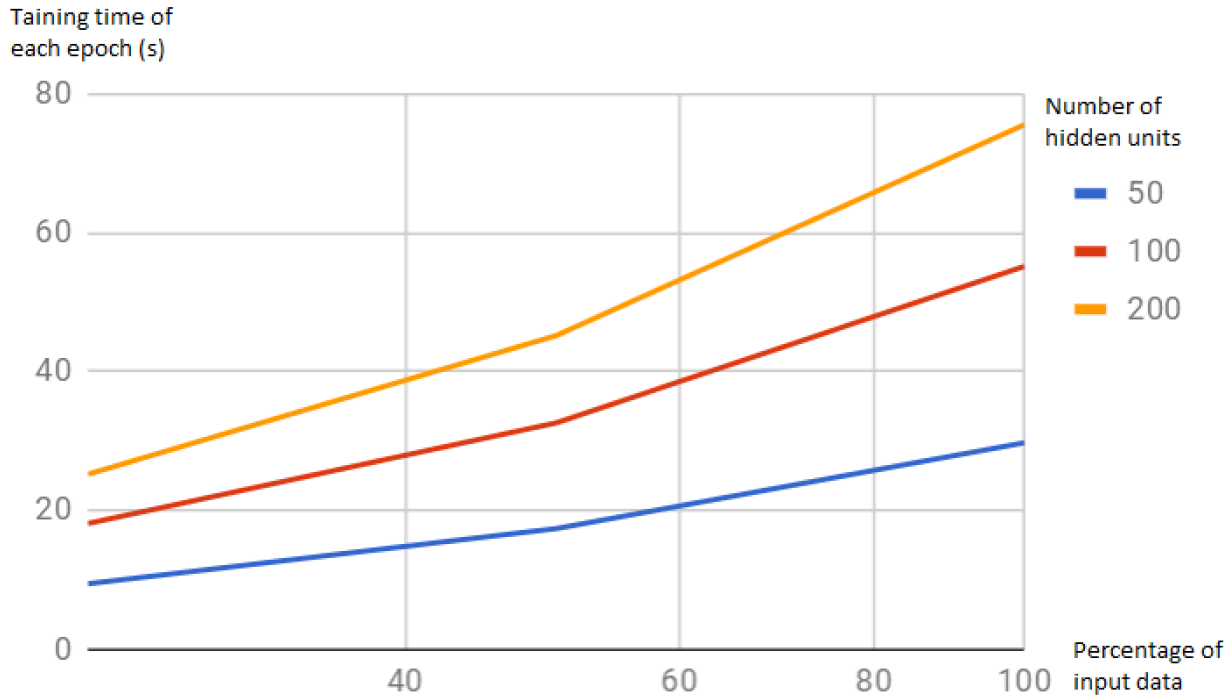
Third experiment: Predicting and optimizing multiple output network

Multiple output network output multiple labels at once. Our goal is to find out if it has higher accuracy when the network output multiple labels at a time. That is because we believe there are relationships between those output. For example, the correctness of the problem is related to if the student is confused or concentrated.



Different from last experiment, the prototype of the code we used was written by a WPI graduate student, Liang Zhang, and then modified by Anthony Botelho. We tested it with similar steps to the above experiment.

First, different from the result of second experiment, we found that the running time of network with different hidden units shows that increasing the number of hidden units does not increase as much of running time. What's more, surprisingly, increasing the number of data does increase running time but not proportional to it (showing in figure below).



Then we also implemented the network by modifying Anthony's code. We output all 8 labels mentioned in the second experiment at once including next problem correctness, first action, retention wheel spinning and affects. The running time did not increase, However, due to a bug in the program, the result accuracy is not acceptable, our team is still working on figuring that our.

Conclusion of the project

By implementing deep learning neural network, we can predict students' competence in an efficient way. Dropout rate seems to not affect the performance of the network. However, autoencoder can increase the performance at the same time reduce running time. We can find the optimize parameter for the network resulting an efficient and time saving network. Adam cost function overwhelm adgrad cost function and a 1-layer RNN network is good enough for our purpose.

However, we have not finished trying 3-layer network or more, or even mixed type network. It is worth trying to have multiple layer network with different combination of number of hidden units and network type. Besides that, we have not done much on implementing and optimizing multiple output network which has lot of potential. In our project, our program always stops when reaching max epoch size. Finding the right cost size to stop can greatly reduce the running time.

Appendix:

Working Log:

Term 1st

(https://docs.google.com/spreadsheets/d/1ZYUxaOZeW_tAefsUuUAC3FgE3xLSD5y6kN6KwcrxeCI/edit?ts=58daf93f#gid=0) Learning Materials schedules

In the first term of the project, we were given materials about deep learning, which provided us with background information and knowledge necessary for the upcoming tasks. These include online deep learning courses and videos from Youtube.

To make an approach to deep learning, the first thing we need to do is to understand how the deep network is built for predicting things we interested about. The network used for deep learning is called neural network. As we learned, deep network is based on neural network which has multiple numbers of hidden layers. From input to output layer, there are weights between each two layers. To get the final output layer, between each layer there are complex calculation done. In learning process, there are concept of overfitting, cross validation, linear regression, logistic regression, and SoftMax regression. Linear, logistic and SoftMax regression are models we may apply when constructing our model for prediction. In the upcoming tasks, we are asked to do experiment on student competence with different models to find out the relationship between the student's competence and factors including frustration, confusion, etc with different models.

First Week:

- Learned basic knowledge of Neural Network (https://www.youtube.com/watch?v=bxe2T-V8XRs Start From here)
- Study deep learning and Neural Network from Udacity
- Learned Overfitting, cross validation, linear regression, logistic regression, SoftMax (https://www.youtube.com/watch?v=VZuKBKd4ck4 Start at Overfitting)
- Read the paper "Modeling student competence: a deep learning approach" and try to have some questions about the theories in the paper：

Second Week:

- Have Udacity deep learning assignment 1 finished which is the nonMNIST from Udacity.
- Watch videos on chapter 6: Autoencoder (Hugo (LaRochelle))
- Reviewed previous videos in the First Week
- Further questions about paper "Modeling student competence: a deep learning approach": questions about pictures in the paper.

How does LSTM increase the performance?
- Suggested use TensorFlow to develop the neural network and further experiments and we decided to use TensorFlow.

Third Week:
- Went through videos of lecture 7: Dropout, Motivation
- Train basic logistic regression model and the link of data is built and listed below in seventh week.
    - Through the data we collected, we found that as the number of sample increases, the accuracy rises gradually, when the sample numbers reaches a high level, the rate of increase became less obvious.
- Try use autoencoder to reduce the complexity of data and field because we used too much time on setting TensorFlow.
- Reread the paper and have the questions about Long term short term memory.
- Redo all the parts above using different sample numbers

Fourth Week:
- Try training with SoftMax Regression model
- Test using model above with different sample numbers
    - From the collected data, the relationship between sample numbers and accuracy is proportional
- Try adding autoencoder to change the complexity of MNIST data
    - But failed
- Test to find the result under new condition to see the effect of autoencoder on dataset
- Reset the number of sample to see the impact on accuracy

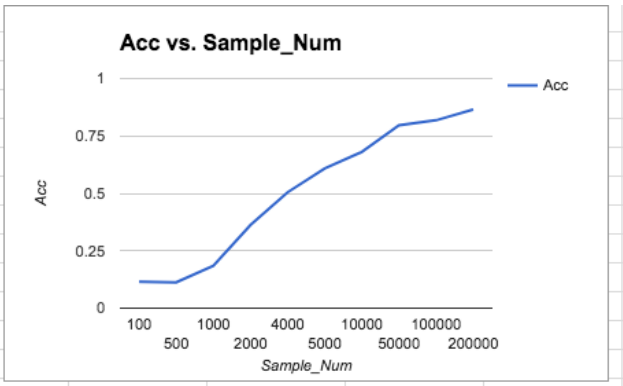
Fifth Week:
- Misunderstood the Neural Network and go through the TensorFlow tutorial
    - We actually built 2 layers of convolutional neural network
- Moved the training module from CPU version of TensorFlow to GPU Version
    - Failed because of out of memory of graphic card
- Trained the MNIST by convolutional neural network with CPU mode
    - With testing using 1,2, and 3 hidden layers, we collected the data and found that:
        - Compared to the result of using 1 hidden layer, the accuracy of using 2 hidden layers has a significant increase
        - Compared to the one using 2 hidden layers, using 3 hidden layers caused the accuracy to drop a little bit
    - Sample number is from 0 to 200000, test sample and validation are all 10000
    - Following the pattern, we found in the previous test, as the sample numbers increase, the accuracy increase in a linear regression
- The goal for next week is to build our own neural network
- Tried to get Autoencoder from Anthony but they still didn't have the answer yet

Sixth Week:

- Train MNIST using Neural Network
    - We built the Neural Network on our own
- Implement the neural network and test the dropout rate (0.6, 0.5, 0.4)
    - We can hardly find the difference using different dropout rate
    - The sample number is from 0 to 200000
- Need to make some graphs of the accuracy of the module
- Tried to implement Autoencoder
- Fit by using different hidden layer numbers for the test on the result to how number of hidden layer affects the result

Seventh Week:

Sample_Num	Acc	Time(s)	Drop Rate 0.5
100	0.1153	1.2311	
500	0.113	0.3227	
1000	0.1847	1.5505	
2000	0.3624	2.275	
4000	0.5043	2.5914	
5000	0.6087	2.9524	
10000	0.6804	4.6742	
50000	0.7967	19.1461	
100000	0.8183	37.5617	
200000	0.8648	70.4569	



Here is part of the table we built using google docs

- Build tables for test results based on MNIST
    - Logistic Regression (https://docs.google.com/spreadsheets/d/1dEDh4RQtxE4urFXJJl9W96Cm7anc7JiHaChR8PCROg0/edit#gid=0)
    - SoftMax(0 level) (https://docs.google.com/spreadsheets/d/1Nk2-aJf1EinWxK2IeLNsuPqYxeGK-eohLysyBUv5wgs/edit#gid=0)
    - Neural Network with 1 hidden layer (https://docs.google.com/spreadsheets/d/1VzkD4vr7wfSdPwGqJDKlafnyzrwg8kPuWyCbTc9BE0Q/edit#gid=0)
    - Neural Network with 2 hidden layers (https://docs.google.com/spreadsheets/d/1VzkD4vr7wfSdPwGqJDKlafnyzrwg8kPuWyCbTc9BE0Q/edit#gid=0)
    - Neural Network with 3 hidden layers (https://docs.google.com/spreadsheets/d/1FHAj1Q9ioA1tDiPWIp7-p5jkOGTlmD4u1F2aeOHuv8s/edit#gid=0)
- Includes: Using Logistic Regression, applying 0,1,2,3 hidden layers in testing with different dropout rate, here is our conclusion:
    - As sample numbers becomes larger, the accuracy gets higher
    - When the number of hidden layer is 2, the accuracy of outcomes is the highest
    - Dropout rate has tiny influence on the accuracy result
- Finish the progress report for later paper and convenience

- Finished the Autoencoder and test it
- Be prepared for the next term's task
- Extra stuffs in summer

8:
try to do this approach by education data
Before number
Now text

Week 5.20-5.26

- Test with different factors in input size, numbers of hidden units, CPU version and GPU version
- Build tables of the result including: time elapse of each epoch, total time on finishing the process, and average time elapse of every epoch

Term 2nd:

Week 1 8/28/2017-9/1/2017

- ●Decide the project direction,
- ●Separate the features of student at 12 years old, train them respectively
- ●Need the data to train for competition
- ●Download the code, modify and train

Week 2 9/1/2017-9/8/2017

- ●Code work
- ●Predict one outcome
- ●Build 4 separate models

Week3: 9/8-9/15

- ●Let the code run successfully
- ●Turn the data given into 3D-array
- ●Need to know the specific features results have

●The data structure

●array[student] [time step] [feature number] (just for make sure)

Week4: 9/15-9/22

●Built the model for training data

●Try several epics of running with autoencoder

●Failed on training 10 percent of data when input number is 10, 1000, 10000

Week5: 9/22-9/29

●Meet with Anthony, solving bugs

●The code can run at least one batch, but had some problem with offsets

●We decided to delete the one have problem in the dataset when building the model.

Week7: 10/6-10/13

●Experiment the data with different factor including: AUC, hidden layer, output type

●Collect the data and calculate the time cost and average epoch time

Term 3rd:

Week one 10/27-11/3:

- More layers perform worse

Data: ExperimentData.csv

Fri 11/10 2017

- We can apply some change to AUC (e.g. 0.6 vs. 0.7)
- RNN VS LSTM
- ADAGRAD VS ADAM
- Use stepsize as a factor
- Test with alpha {0.0.1, 10^ (-6)}, and keep rate
- Turn off the threshold for Anthony's code

Data: ExperimentData.csv
adam_vs_adagrad.csv
Step_Size.csv

Fri 11/17 2017

- Before last week we kept using ADAGRAD
- Predict each label individually and compare with the Theano Framework (Tensor Flow vs. Theano).
    - 1. Individual models: wheel spin, retention, first action
        - Harder: affects: boredom, frustration, confusion, concentration.
    - 1.5. Run npc on Xiaolu dataset. (The difference on different data set for same model)
    - 2. All above at once (8): multiply all the model at once, probably 20 times longer
- No tune for the parameters and keep going.

Data: adam_vs_adagrad.csv
SingleOutPut

Fri 11/24 2017

- Thanksgiving Break

Fri 12/1 2017

- Get experiment data for single output(npc,rc,fa,ws)

Fri 12/8 2017

- Working on fixing bugs on multiple output with Anthony and Seth.
- Working on One-Hot encoding (Boredom, Frustration, Confusion, Concentration)