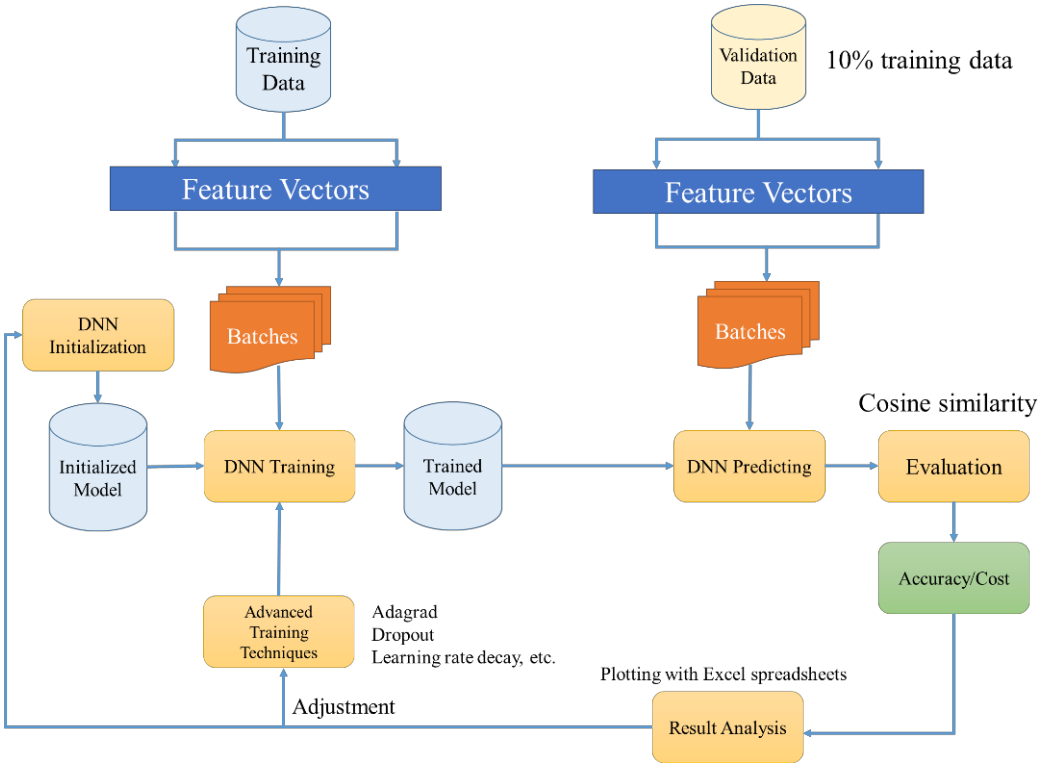
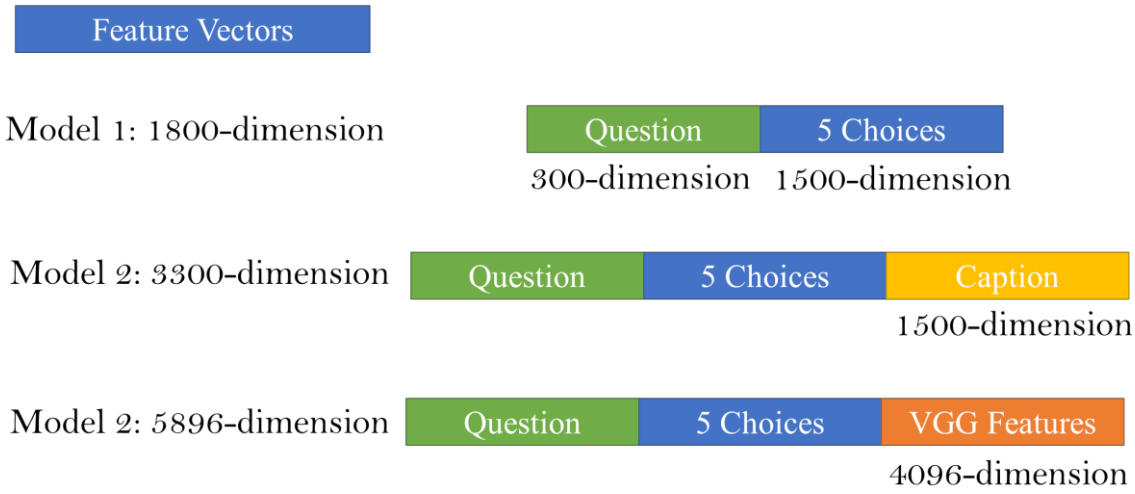
Final Project
Visual Question Answering
Report

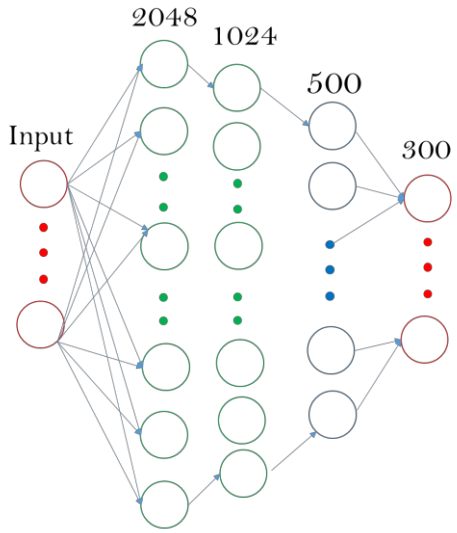
I. Group Information

組別名稱: Long Live Livers 第 15 組		
Name	**Department/ Student ID**	**Contribution**
蔡卓忻	電機工程學系四年級 b01901101	Input data processing (word2vector) Algorithm design Result analysis Report writing
李明安	資訊管理學系四年級 b01705039	Algorithm design Input data processing (word2vector) Code debugging Result analysis
盧勁瑋	電機工程學系四年級 b01502002	Input data processing (sentence2vector) Compare different models Report writing
余建遠	電信工程學研究所二年級 r03942125	Input data processing (caption) Compare different models Result analysis

II. Algorithm Design







Model:
Input × 2048 × 1024 × 500 × 300

Input data:
Depends on the above models

Hidden layer:
3 hidden layers

Output data:
300-dimension vectors

III. Data Preprocessing

1. Word Embedding(questions, 5 choices and captions):

Questions 和 Choices 使用 word2vector 和 sentence2vector 分別轉為 300 維的向量，而 captions 則從 reference C 中取得，每張圖片有 5 句描述的話

i. **word2vector:** obtained from the following reference A.

ii. **sentence2vector:**

We use the **First Billion Characters from Wikipedia** (which is approximately 0.2B words considering that the average length of English words is roughly 5) for model training.

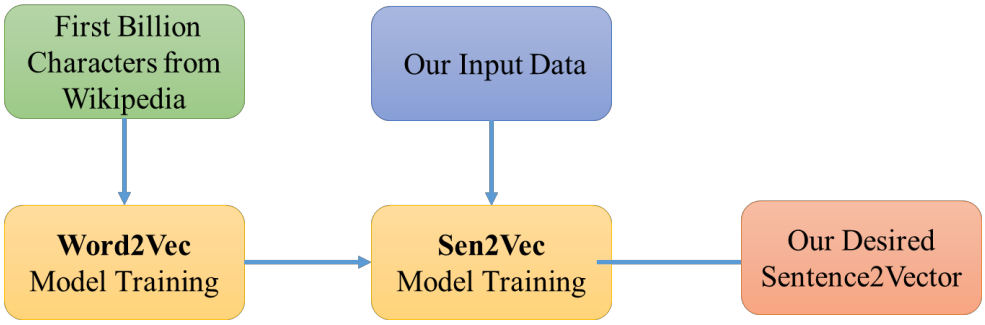
The model from the following reference B provides function **Word2Vec** to train and save a model on a specific set of text data. In function **Sent2Vec** we could take the pre-trained word2vec model as a parameter to realize the sentence-vector transformation.

References:

A. Sen2Vec: https://github.com/klb3713/sentence2vec

B. Word2Vec training data: https://code.google.com/p/word2vec/

C. http://cs.stanford.edu/people/karpathy/deepimagesent/coco.zip



2. Image Embedding(image features extraction):

每張圖片含有 4096 維的向量。從 reference C 中取得。

IV. Experimental Results and Discussions

我們使用了三種不同的 feature vector 當作 input data(如上頁中的 Model 1~3)，分別使用 word2vector 和 sentence2vector 來做 word embedding，並針對不同的 DNN 參數做調整，實驗結果和分析如下個段落所示。

1. Best-performing model:

最佳 Kaggle 成績:0.54069

詳細資料:

Feature Vector: Model 2 (feature vectors 為 3300 維，包含 question, 5 choices 和 caption)

Word Embedding: word2vector

Optimizer: Adagrad

Initialization: glorot_normal

Dropout: 0.3

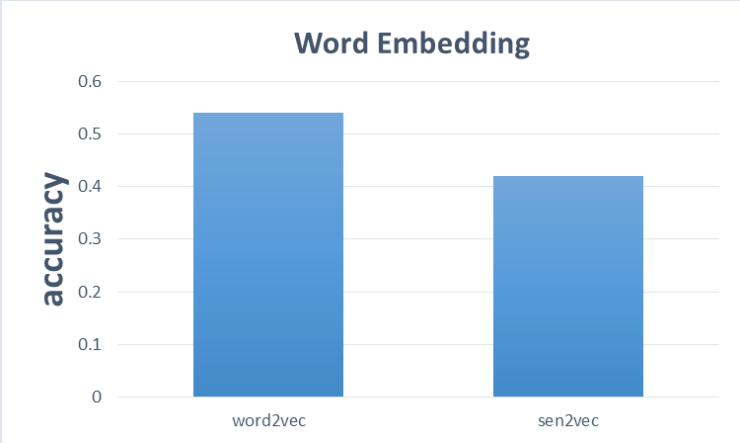
Batch size: 100

Activation function: PReLU

Cost function: mean squared error

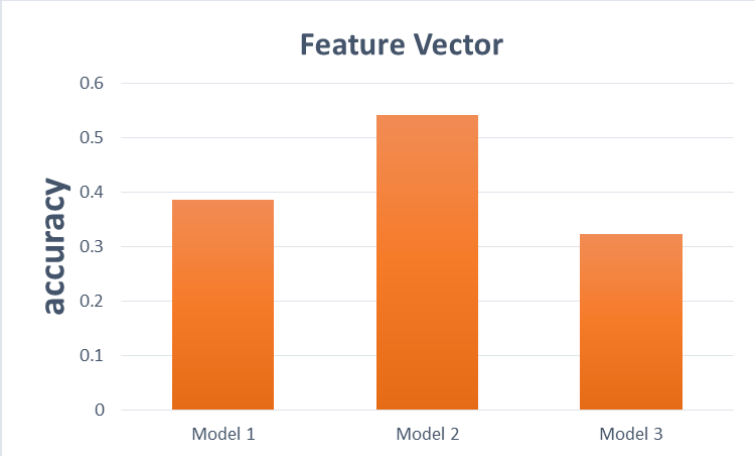
2. Comparisons:

i. word embedding: word2vector v.s. sentence2vector:



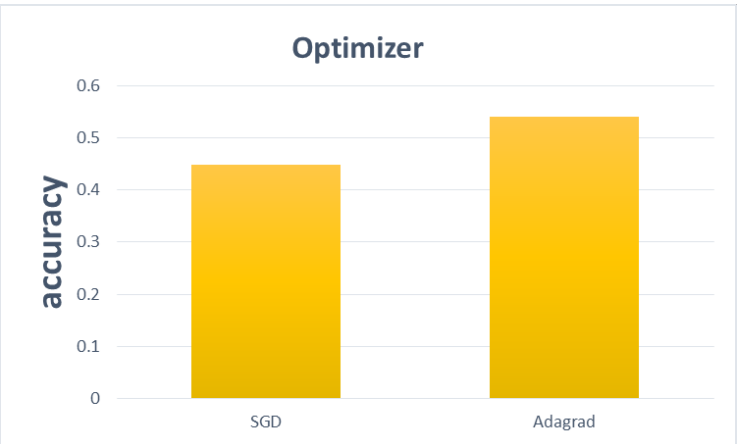
由上圖可看出，使用 sentence2vector 不如 word2vector。我們推測可能是由於我們 sentence2vector 是我們自己使用 Wikipedia 的 corpus 所 train 而成，因此才不如 Google 自己訓練的 word2vector。

ii. feature vectors: model 1~3:



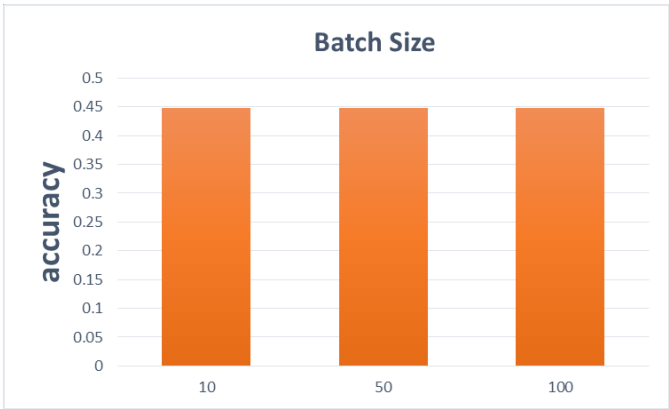
Model 2 (question + 5 choices + captions)表現最佳。Captions 較 4096 維的 VGG features 更精準的指出圖片的重要特徵，因此會勝過含有 VGG features 的 Model 3。也有可能是因為加入 VGG features 後，需要更龐大複雜的 DNN 結構，才能訓練成功，而我們使用的 DNN 模型不夠大。VGG Features 本是為了圖片分類而所產生，和此處我們的圖片問答不同，因此可能無法提供我們想要的資訊。

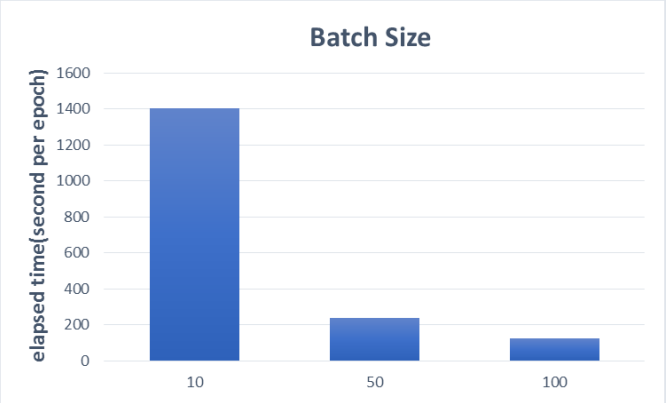
iii. optimizer for DNN model:



此比較乃是兩者跑完 20 個 epoch 之後的結果。Adagrad 較能 adaptively 調整 learning rate，因此收斂速度較 SGD 快。

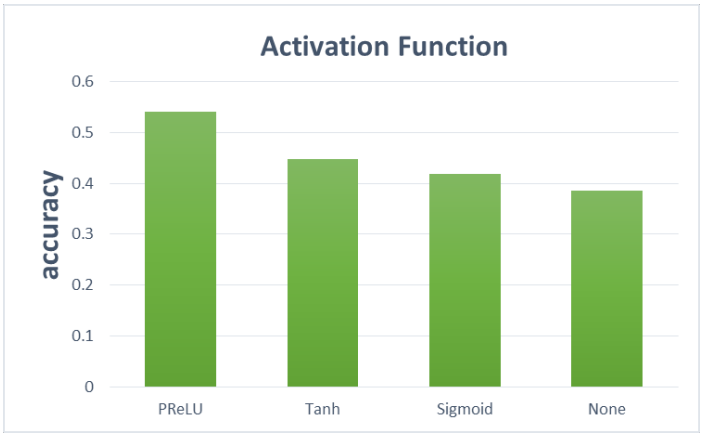
iv. batch size:





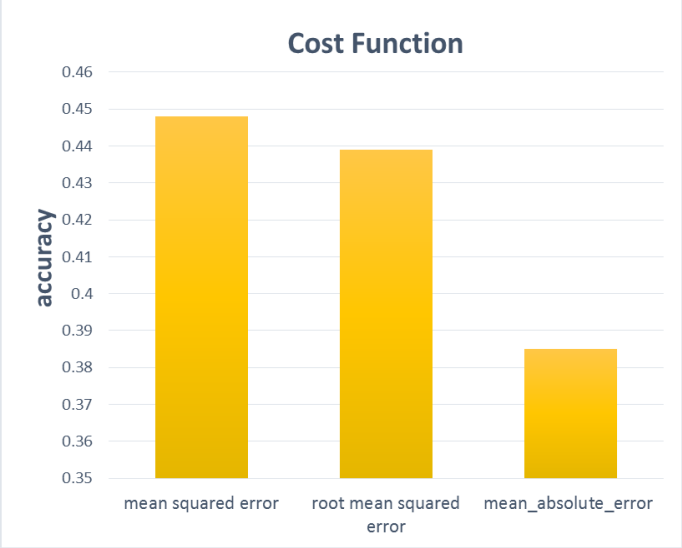
Batch Size 對於最後的 accuracy 並不會影響太多，但隨著 batch size 變大，跑完一個 epoch 所需要的時間則近似等比例地遞減(上左圖縱軸單位為秒)。

v. activation function:



PReLU 表現最佳。若不用任何 activation function，則表現最差。PReLU 對於橫軸小於 0 的部分能彈性調整，因此在 training 的時候其斜率會一起被調整。

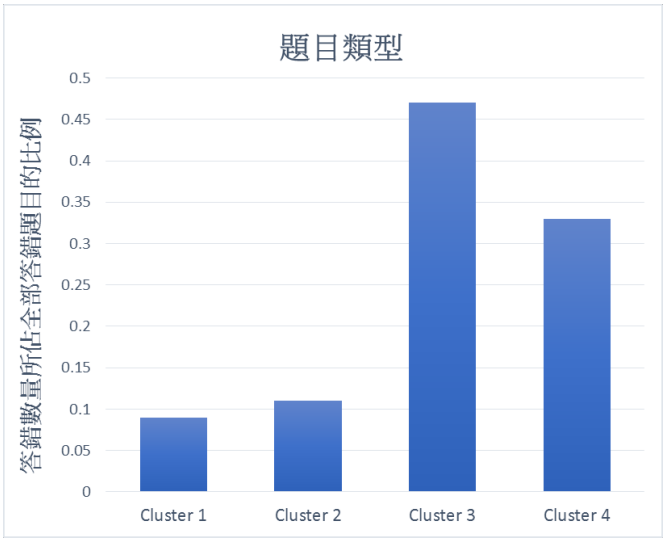
vi. cost function



Mean squared error 表現最好。可能是由於我們使用 cosine similarity 決定 testing data 該選什麼答案，所以 mean squared error 相差越小的兩個向量，其 cosine similarity 可能越大。

V. 分析答錯狀況

我們使用 10%的 training data 當成 validation data 中，並分成四組，觀察答錯的題目是何種類型。



Cluster 1 是較簡單的題目(如:問顏色，物品等等)。**Cluster 2** 是 Yes/No 的問題(如:問 could, can, is 等等)。**Cluster 3** 是數字題(如:問 What time, what is the number)。**Cluster 4** 是問較抽象的題目(如:Why, how 等等)

由左圖可看出，我們的模型較無法處理數字題(cluster 3)，因為 word2vector 無法將「8:30」之類的數字化為向量。而抽象題目(cluster 4)也較為困難，因為必須要對於文意有深一層的了解，才能正確答對，這就是為什麼我們模型表現較差的原因。

Homework 3
Structured Learning
Report

I. Team Contribution

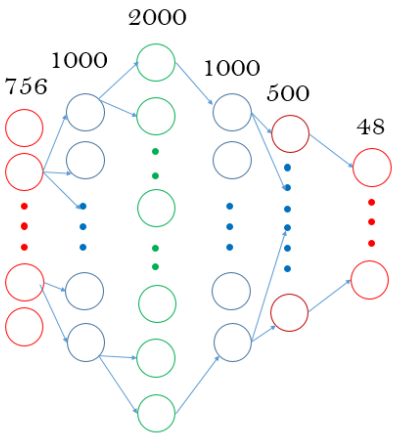
組別名稱:	**Long live livers**	第 **15** 組
Name	**Grade/ Student ID**	**Contribution**
蔡卓忻	電機工程學系四年級 b01901101	Algorithm design Code debugging Output data processing (smoothing) Report writing
李明安	資訊管理學系四年級 b01705039	Input data processing (DNN) Code debugging Result analysis
盧勁瑋	電機工程學系四年級 b01502002	Input data processing (RNN) HMM implementation Compare different models Report writing
余建遠	電信工程學研究所二年級 r03942125	Compare different models Output data processing (trimming) Result analysis

II. Data processing

1. Input data:

我們使用以 DNN 和 RNN 所輸出的 48 維 probability 當作 input data，以下是兩個模型的詳細資料

i. DNN:



DNN Model:
756 × 1000 × 2000 × 1000 × 500 × 48

Input data:
756-dimension data
756 = 7 × 108
7: previous 3 frames + center frame + subsequent 3 frames
108: 69 FBANK + 39 MFCC features

Activation function: ReLU

Output data:
48-dimension probability

AdaGrad and softmax function are utilized.

ii. RNN:

- Bi-directional RNN
- Input: 48-dimension DNN output probabilities
- Hidden layer: 4 hidden layers, each having 128 neurons
- Learning rate: 10^{-4}
- Activation function: ReLU
- Cost function: Cross entropy
- Wh: $0.01I$ (identity matrix)
- Softmax function is included at output layer.
- RMSProp is used to automatically adjust the learning rate at each epoch.
- Gradients are clipped to avoid exploding gradient problem.

2. Smoothing:

針對輸出的 phone sequence，我們使用了幾個 smoothing 的規則，如下(window size = 5):

舊的 phone sequence	新的 phone sequence
aabaa	aaaaa
aabdd	aaddd
aabdc	aaccc
abaaa	aaaaa
aabdc	aaccc
abcaa	aaaaa
abbaa	aaaaa

III. Comparisons and Experimental Results

1. Best-performing model:

最佳 Kaggle 成績: 8.95608

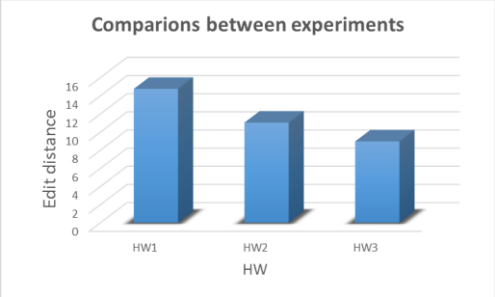
最佳 Model:

Structure learning algorithm: HMM

Input data: DNN 48-dimension probability

Smoothing: Implemented

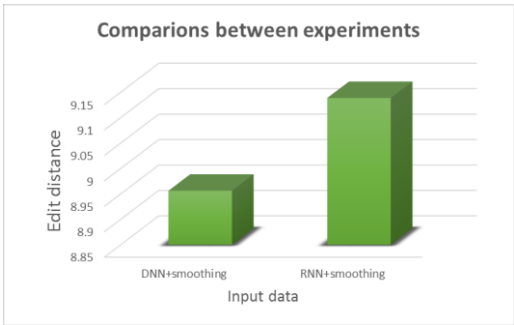
2. Comparisons between HW1 and HW2:



從上圖可看出 HW3 的表現優於 HW2 和 HW3。HW1 是 DNN 模型，單純考慮每個獨立的 phone，故表現不如考慮整個句子的 RNN 模型 (HW2)。而在 HW3，我們使用 structure learning 的 HMM 演算法，利用 HW1 和 HW2 的結果和 Viterbi algorithm 求出最有可能的 phone sequence，故結果最佳。

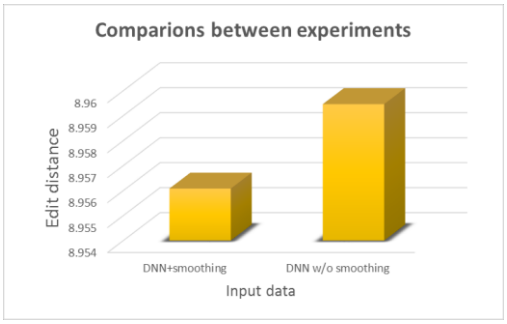
3. Other comparisons:

i. About input data:



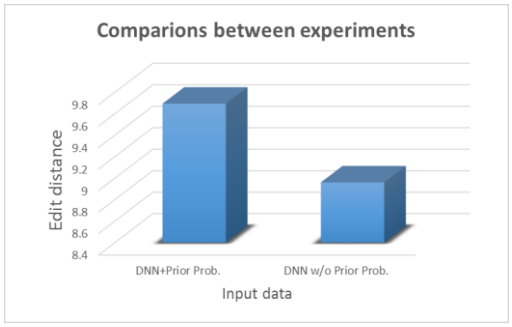
出乎意料的是，使用 RNN 當作 input data 表現反而比較差，推測可能是 RNN 考慮了整個句子，因此每個 phone 的 probability 會被左右鄰居影響，但在 HW3 的 HMM 當中，我們已考慮了 transition 和 observation probability(相等於考慮左右鄰居)，因此若使用 RNN 的 probability，反而可能因為功能重疊而造成不精確。

ii. About smoothing:



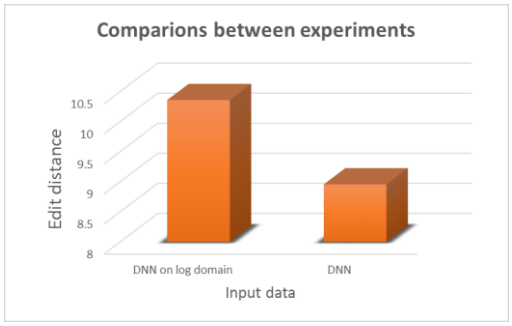
從上圖可以看出有 smoothing 之後的表現較佳，乃因 smoothing 之後讓每個 phone 至少連續出現 3 次，能修正一些誤判的 phone。

iii. About prior probability $\mathbf{P}(s)$:



從上圖可看出除掉 Prior probability $\mathbf{P}(s)$ 之後表現會變差，據李宏毅教授所說，這部分表現會不會變好可能是 corpus dependent。

iv. About addition and multiplication:



從上圖可看出，若在 Viberbi 演算法求 arg max 的時候，不相乘而在 log domain 相加，表現會稍許變差。

If TAs and Professor Lee have any question regarding our program, readme file or this report, please contact us for further detailed explanation.

Homework 2
Training Recurrent Neural Network
Report

I. Group Information

Group name: **Long live livers**		Group number: **15**
Name	**Grade/ Student ID**	**Contribution**
蔡卓忻	電機工程學系四年級 b01901101	Structure and algorithm design Vanilla RNN implementation Data testing Report writing
李明安	資訊管理學系四年級 b01705039	Data preprocessing Advanced techniques implementation (bi-directional, RMSProp, etc.) Data testing Result analysis
盧勁瑋	電機工程學系四年級 b01502002	Compare different models (learning rate, initialization, etc.) Data post-processing (smoothing) Result analysis
余建遠	電信工程學研究所二年級 r03942125	Compare different models (Epoch, activation function, etc.) Data post-processing (trimming) Result analysis

II. What Have We Done?

We here briefly present what we have considered and implemented in this homework, whereas the experimental results of respective work is expressed in section **III**.

1. Data structure and algorithm design

i. About best model:

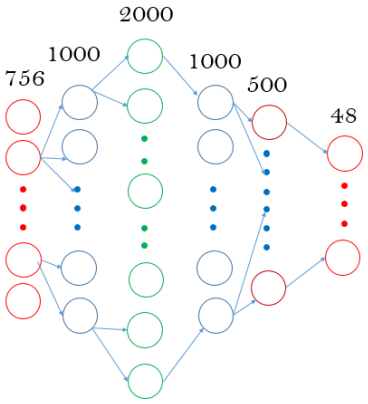
The program, named ***rnn_1.4.4_relu_4hid.py*** in the directory, represents our best model. TAs can adjust every single relevant parameter in our program (see readme file).Our best model has earned a score of **10.58** on HW-2 Kaggle. We submitted our result to HW-1 Kaggle and obtained an accuracy of **0.73496**. The parameter setting is as follows.

Best-performing RNN model:

- Bi-directional RNN
- Input: 48-dimension DNN output probabilities generated by ourselves
- Hidden layer: 4 hidden layers, each having 128 neurons
- Learning rate: 10^{-4}
- Activation function: ReLU
- Cost function: Cross entropy
- Wh: $0.01I$ (identity matrix)
- Softmax function is included at output layer.
- RMSProp is used to automatically adjust the learning rate at each epoch.
- Gradients are clipped to avoid exploding gradient problem.

2. Data preprocessing

i. **About DNN output probability:** To generate more useful data, rather than directly using the data provided by TAs, we utilized DNN output probabilities ourselves. The model successfully achieving a score of **0.72187** on HW-1 Kaggle is shown below.



DNN Model:
$756 \times 1000 \times 2000 \times 1000 \times 500 \times 48$

Input data:
756-dimension data
$756 = 7 \times 108$
7: previous 3 frames + center frame + subsequent 3 frames
108: 69 FBANK + 39 MFCC features

Activation function: ReLU

Output data:
48-dimension probability

AdaGrad and softmax function are utilized.

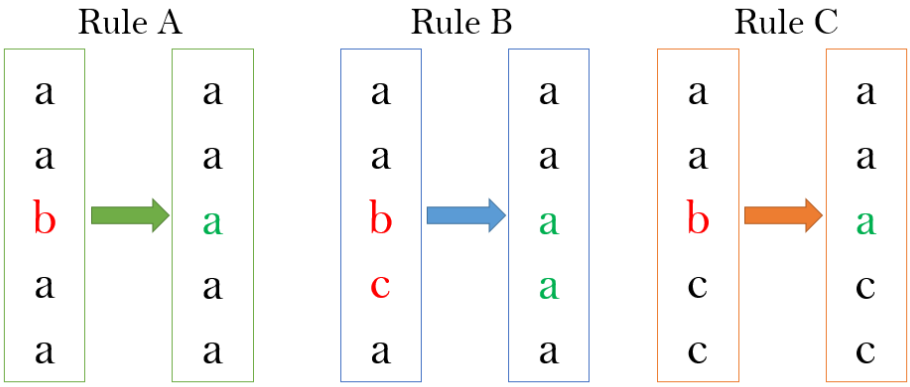
ii. **About validation set:** Since no validation set was offered, we therefore split the tremendous training data to obtain validation data. 1% of all training data was taken out to form the validation set *validation.ark*.

iii. **About shuffling and label lookup:** We shuffled the order of the sentence at each epoch. Furthermore, to look up the corresponding label as correct output phoneme in a more efficient way, we created *validation.result* and *train.result*, where the correct labels are listed. As a consequence, when dealing with each sentence, we do not need to open *label.ark* and search from top to bottom to find corresponding target output label.

3. Implementation tips

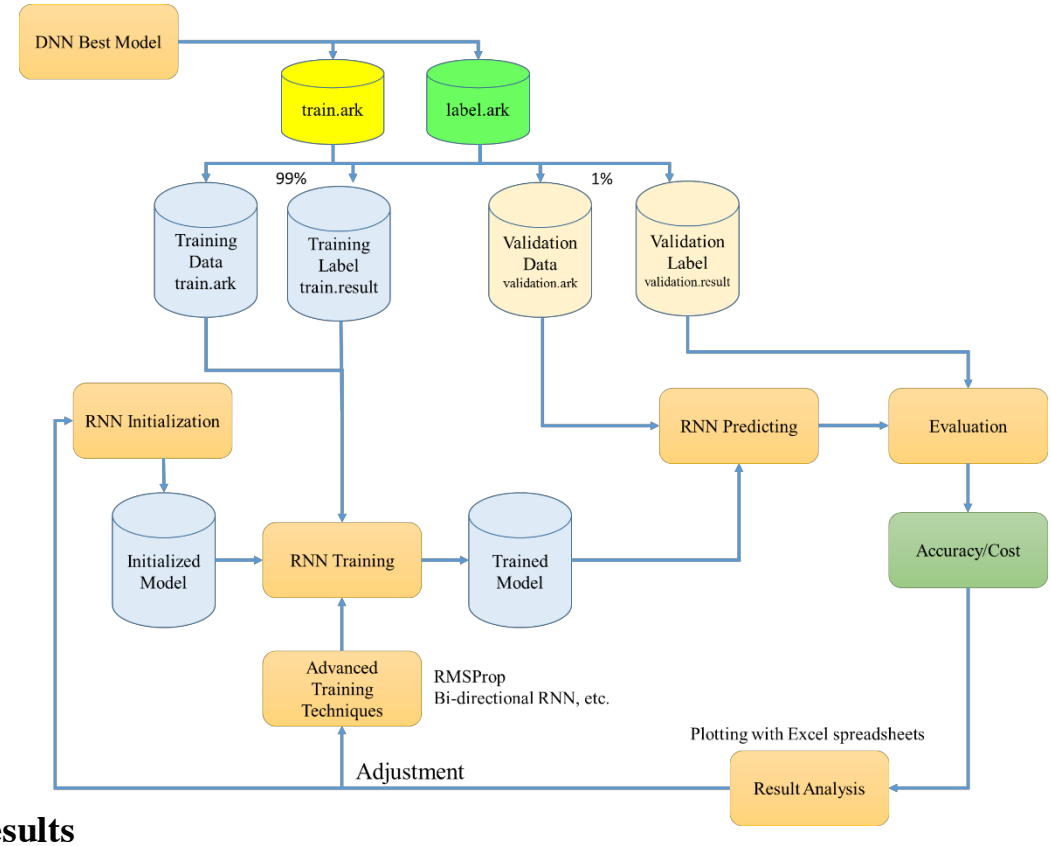
i. **About "relay" training:** If the validation error is small enough, we *save* those good-performing parameters as the initial values for the next training round, saving lots of time from "rectifying" random initial parameters again.

ii. **About smoothing:** There are a number of rules to help us correct the possibly wrong output phones. Below we only demonstrate a few examples. For more detail, please refer to readme file and the program ***smoothAndTrim.py***.



III. Experimental Setting and Results

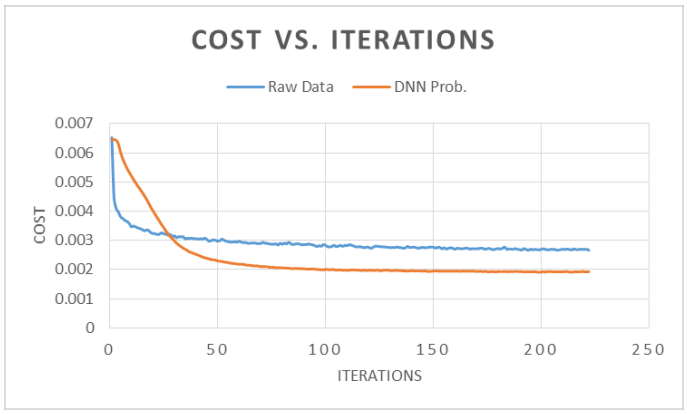
1. Experiment design

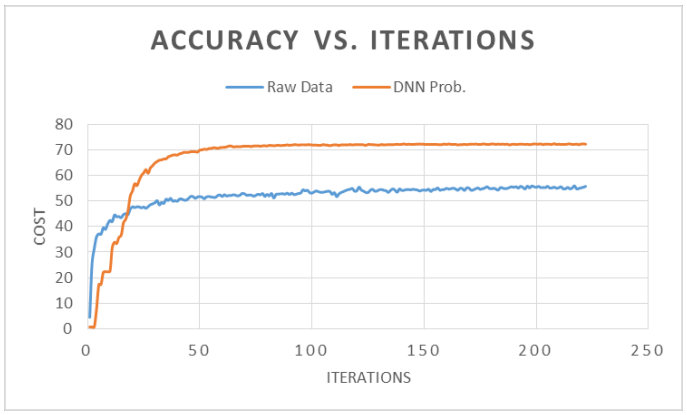


2. Results

i. **Kaggle competition:** Best entry earns a score of **10.58** for HW-2 Kaggle. When submitted to HW-1 Kaggle, the accuracy is **0.73496**

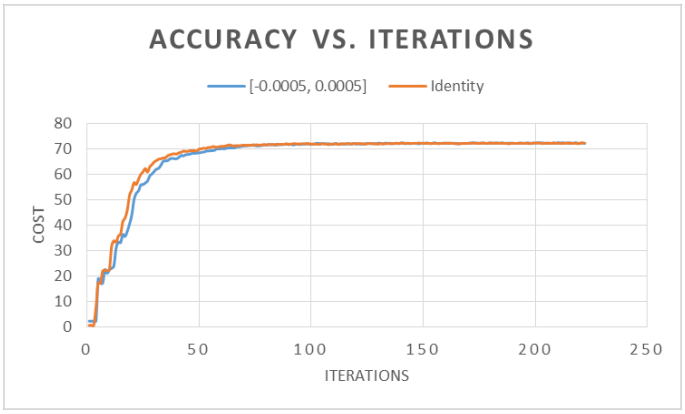
ii. About input data:





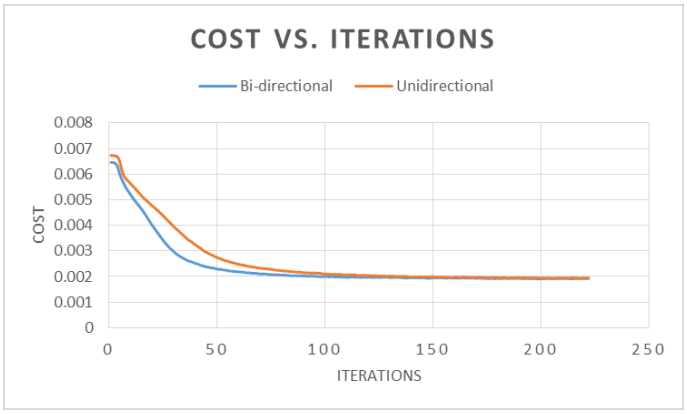
Raw data refers to 108-dimension data, containing 69 FBANK and 39 MFCC features. From above two figures, it is obvious that our DNN probabilities are better than raw data.

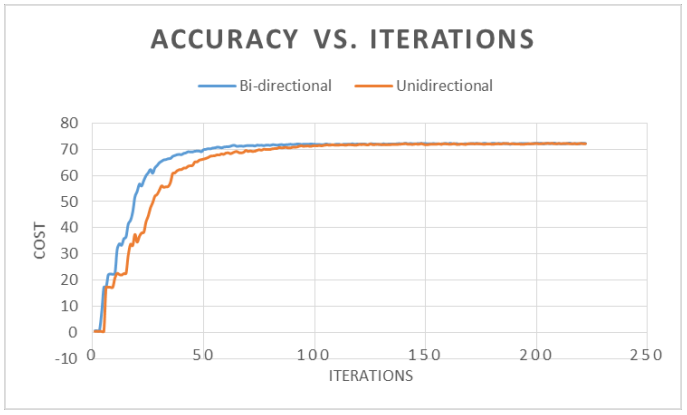
iii. About different initialization on the matrix Wh:



We find an identity matrix, which may be multiplied by a small scalar, outperforms the one generated by uniformly random numbers at a faster pace for convergence.

iv. About bi-directional RNN and unidirectional RNN:

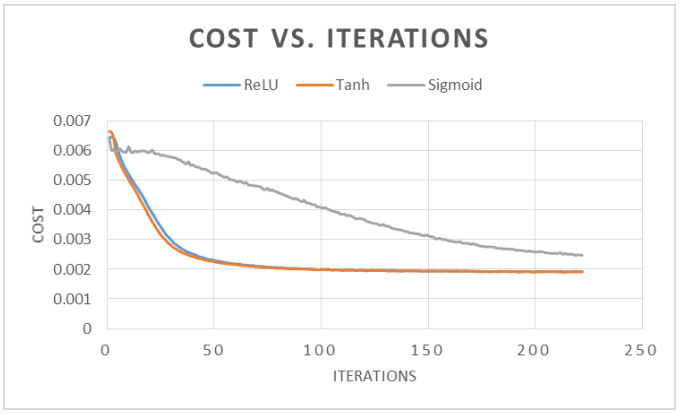


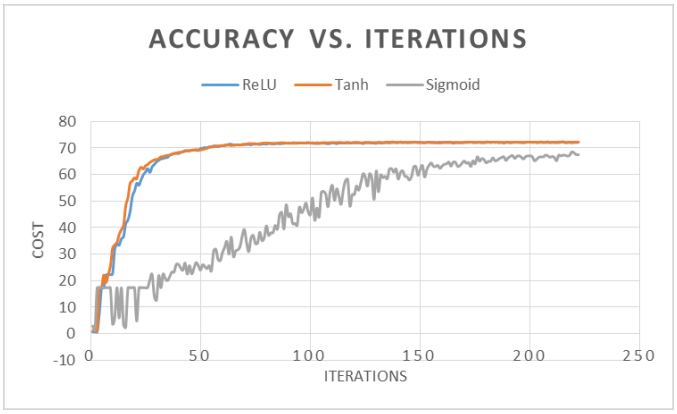


Unidirectional RNN takes more time to converge to local minimum.

v. About activation function:

Sigmoid function cannot be worse than ever. Accuracy ascends slowly and fluctuates seriously, totally beaten by the other two.





If TAs and Professor Lee have any question regarding our program, readme file or this report, please contact us for further detailed explanation.

Homework 1

Training Deep Neural Network

Report

I. Group Information

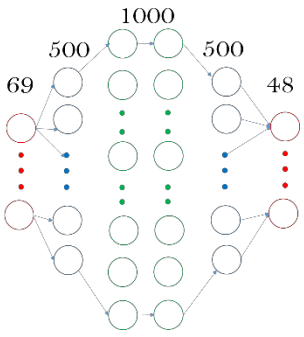
Group name: **Long_live_livers**		Group number: **15**
Name	**Grade/ Student ID**	**Contribution**
蔡卓忻	電機工程學系四年級 b01901101	Data preprocessing/postprocessing Structure and algorithm design/debugging Data testing Report writing
李明安	資訊管理學系四年級 b01705039	Advanced techniques implementation (Dropout, ReLU, etc.) Data preprocessing Algorithm design Data testing Result analysis
盧勁瑋	電機工程學系四年級 b01502002	Compare different models (Learning rate, batch size, training data, etc.) Basic techniques implementation (Feedforward) Result analysis
余建遠	電信工程學研究所二年級 r03942125	Compare different models (Initialization, epoch, activation function, etc.) Basic techniques implementation (Gradient descent)

II. What Have We Done?

We here briefly discuss what we considered and implemented in this homework, while the experimental data of respective work is presented in section **III**. Anyone can adjust every relevant parameter in our program (see readme file).

1. Data structure and algorithm design

i. About deep neural network model: As shown below, we implemented *2-hidden-layer* model as well as *4-input-layer* model. Output phonemes were further mapped into corresponding symbols in 39-phoneme system.

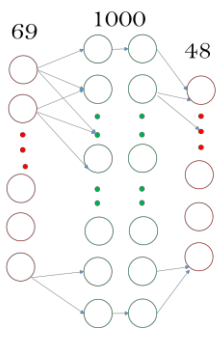


Model:
69 × 500 × 1000 × 1000 × 500 × 48

Input data:
69 FBANK features

Hidden layer:
4 hidden layers

Output data:
48 phonemes



Model:
69 × 1000 × 1000 × 48

Input data:
69 FBANK features

Hidden layer:
2 hidden layers

Output data:
48 phonemes

ii. **About initialization:** To enhance flexibility of our program, we parameterized respective weights and biases for each layer (*wMulti*, *wAdd*, *bMulti*, *bAdd* as shown below). Hence, we could initialize them as we wish. As can be seen from experimental results, initialization exerts a great influence. Parameters with identical values get stuck at the same place and cannot be updated. Typically, suitable parameters are within a certain range.

```
class Layer(object):
    def __init__(self, neuron_num, last_layer_neuron_num, wMulti, wAdd, bMulti, bAdd):
        self.neuron_num = neuron_num
        self.weights = theano.shared(np.asmatrix( wMulti * (np.random.randn(neuron_num, last_layer_neuron_num) +wAdd), dtype='float32'))
        self.bias = theano.shared(np.asarray( bMulti * (np.random.rand(neuron_num) + bAdd), dtype='float32'))
        self.dw = T.matrix(dtype='float32')
        self.db = T.vector(dtype='float32')
        self.last_dw = np.zeros(shape = (neuron_num, last_layer_neuron_num), dtype = 'float32')
        self.last_db = np.zeros(shape = (neuron_num, ), dtype = 'float32')
```

iii. **About activation function:** *Sigmoid function*, *ReLU*, *Tanh* were all taken into account. The *Softmax function* was combined with ReLU. Sigmoid performs worst of all.

iv. **About batch size:** In our experiment, we automatically *adjusted batch size* in each epoch. An unfixed batch size yields a surprising benefit—higher accuracy!

v. **About learning rate:** A *decaying* learning rate helps approach the local minimum, whereas a constant learning rate may possibly fail to reach the valley of gradient descent. We decreased the learning rate in each epoch so as to gradually shrink the step.

vi. **About size of training examples:** We selected different sizes of training examples and observed that too few training data could not yield decent performance even in many epochs.

vii. **About advanced techniques:** *Momentum* and *Dropout* were successfully included. It can be seen that Momentum stabilizes the cost; Dropout, on the contrary, renders cost wavering.

2. Data preprocessing

i. **About validation set:** Since no validation set was offered, we therefore split the tremendous training data to obtain validation data. One-tenth of the training data was taken out to form the validation set *validation.ark*.

ii. **About shuffling and label lookup:** The training set provided by the TAs is organized in an alphabetical order, and thus it could be malapropos for training. We solved this problem simply by shuffling the order of the data. Furthermore, to look up the corresponding label as correct phoneme output in a more efficient way, we created *validation.result* and *train.result*, where the correct labels are listed. As a consequence, we do not need to open *label.ark* and search from top to bottom to find each target output label.

3. Implementation tips

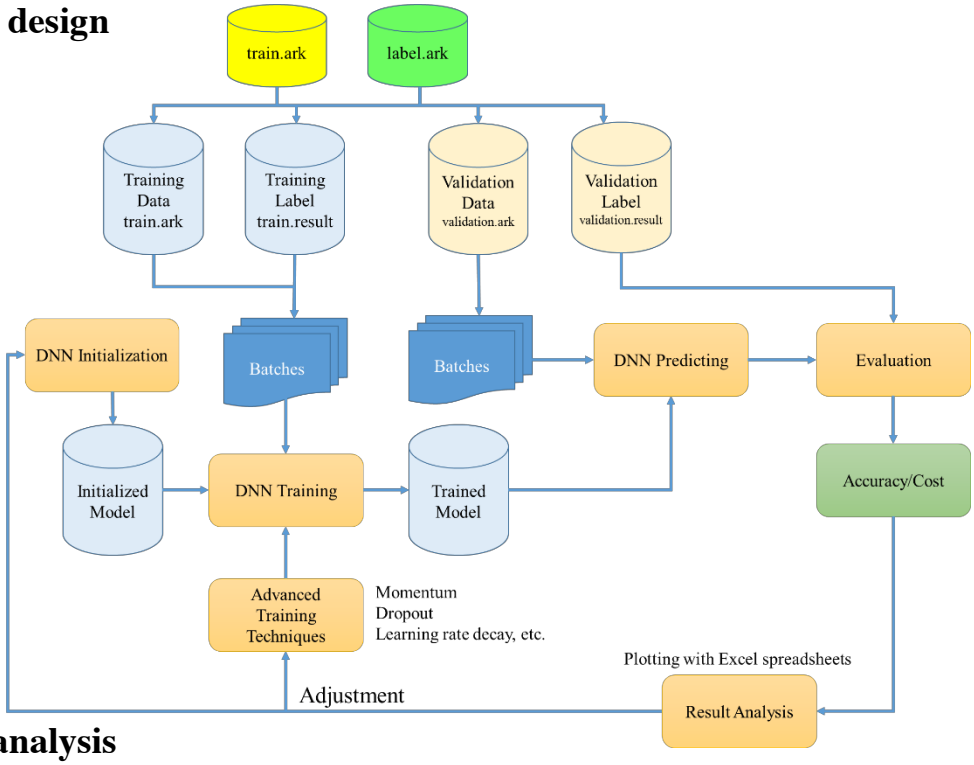
i. **About target output:** It is doubtless that the maximal output value of hyperbolic tangent function is 1, and therefore by intuition the target output $\hat{y}$ should be set to 1 at correct index and -1 for the rest. However, we deliberately set the target output $\hat{y}$ larger than 1, say, 8. What is beyond our expectation is the result does present a noticeably *faster* speed of ascent of accuracy, though it might then get stuck if the inaccurate "inflated" coefficient $\hat{y}$ would not drop to smaller values (and eventually, to its original value, 1).

ii. **About "relay" training:** If the validation error is small enough, we *preserve* "outstanding parameters" as initial values for next training round, saving lots of time from "rectifying" the random initial parameters.

iii. About result analysis: *Exponential smoothing* is especially incorporated when plotting raw data of cost and accuracy values as shown later. It serves as a low-pass filter to leave out unimportant fine-scale phenomena in our experimental curves, making it easier for us to capture significant patterns or tendencies.

III. Experimental Setting and Results

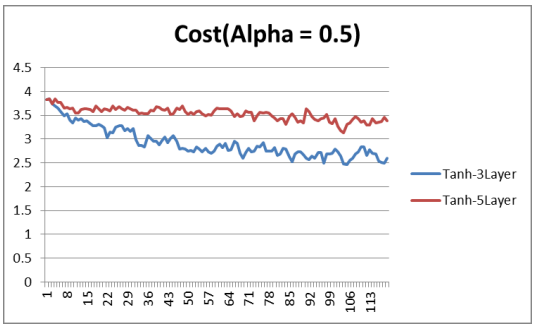
1. Experiment design

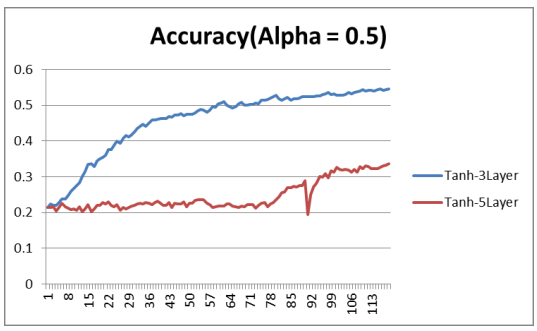


2. Result and analysis

i. Kaggle competition: Best entry earned an accuracy of 0.64529 without advanced training techniques such as momentum or dropout.

ii. About network model:

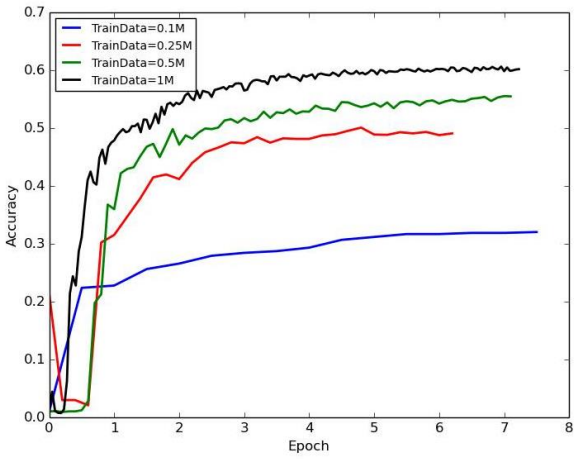




5-layer (4 hidden layers) model does not outperform 3-layer model, possibly due to the vanishing gradient problem and thus some parameters cannot be updated in each turn.

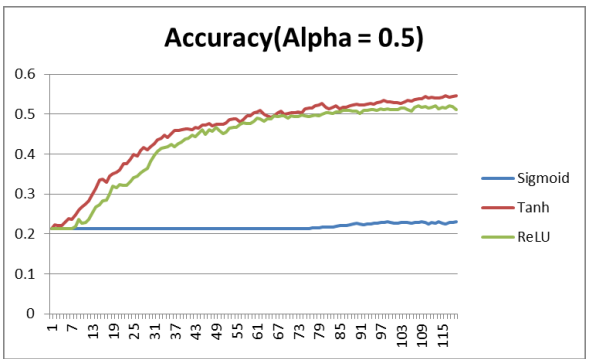
iii. About training data:

It does not take us by surprise that using fewer training data yields poorer performance. When 0.1M training data is applied (10% of the original amount), accuracy can hardly get higher than 0.3.



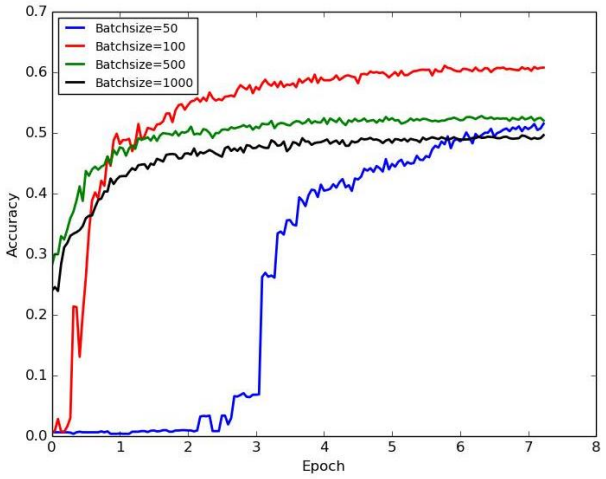
iv. About activation function:

Sigmoid function cannot be worse than ever. Accuracy ascends very slowly, totally outperformed by the other two. The *Alpha* value displayed on the right figure refers to the smoothing constant.



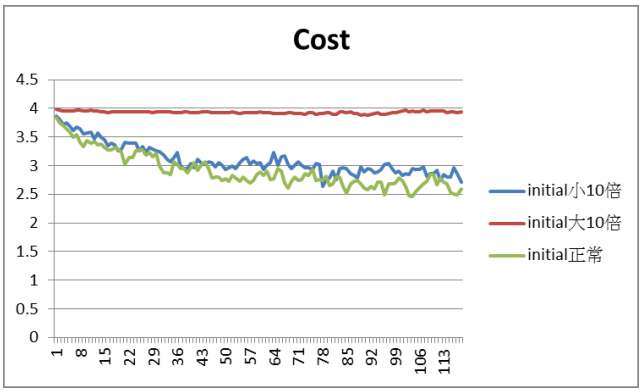
v. About batch size:

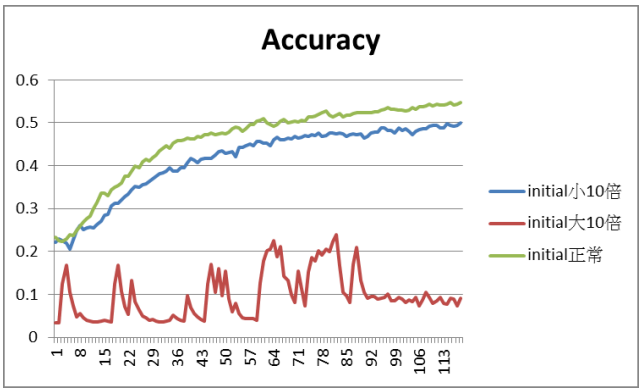
The batch size is influential to result in our experiment. If it is too large, say, 1000, corresponding to the black curve on the right figure, accuracy gets converged to a dissatisfactory value. By contrast, a very small batch size, say, 50, requires a large amount of time for accuracy to increase as shown by the blue curve.



vi. About initialization of weights and biases:

For one thing, in our experiments, we found different models (e.g., models with different activation function) have respective "appropriate parameters." Thus, the best parameters for one model do not necessarily ensure similarly excellent results to other models. For another, too large values could hardly improve cost and accuracy. Below shows our result. We picked a suitable set of parameters (denote by "initial 正常") and did multiplication (or division) to



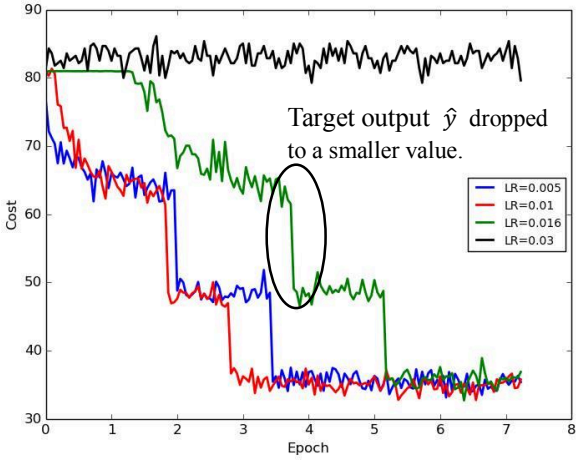


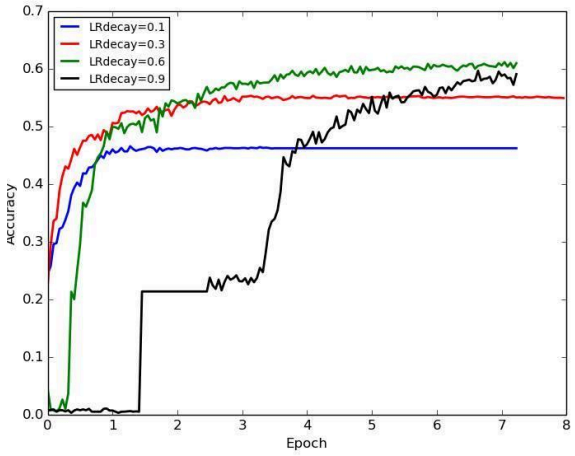
obtain ten times (or one-tenth) of the original weights and biases. One can easily observe the best parameters ("initial 正常") do not go to extreme values

vii. About learning rate and its decay:

As can be observed from the following two figures, similar to the behavior of the batch size, a huge learning rate does no good to our classification problem, whereas smaller learning rates, or larger *LRdecay* (*Learning rate*$^{t+1}$ = *Learning rate*^{t} × *LRdecay*) need more training

epochs to improve performance. It should be noted that on the lower left figure there are *abrupt declines* for those curves. Actually, this is where our tricks are! We adjusted the maximal target output values from 8 gradually down to 1 (see **Implementation tips** for more detail), which had brought a sudden enhancement in terms of cost and accuracy.





If TAs and Professor Lee have any question regarding our program, readme file or this report, please contact us for further detailed explanation.