CSC311 Final Report

Fall 2020

Chunjing Zhang, Peiqing Yu

University of Toronto

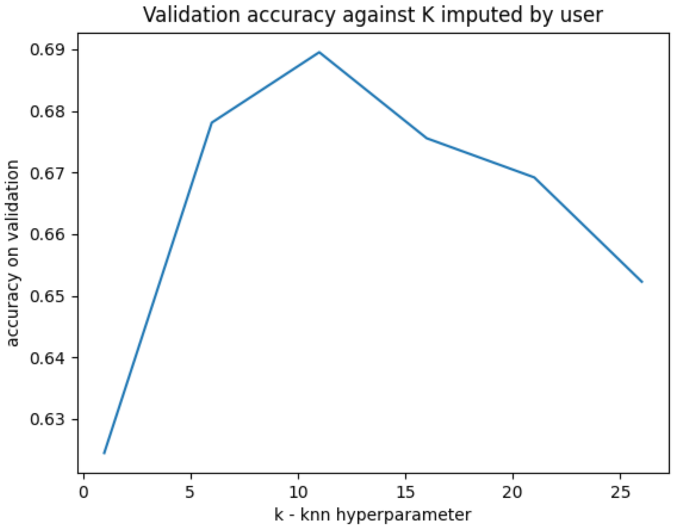
2020.12.15

Part A

Question 1

(a) For k = {1, 6, 11, 16, 21, 26}, the corresponding validation accuracy imputed by user are:

```
Knn imputed by user have accuracy
[0.6244707874682472, 0.6780976573525261, 0.6895286480383855, 0.6755574372001129, 0.6692068868190799, 0
.6522720858029918]
```



Please check the relevant code at knn.py

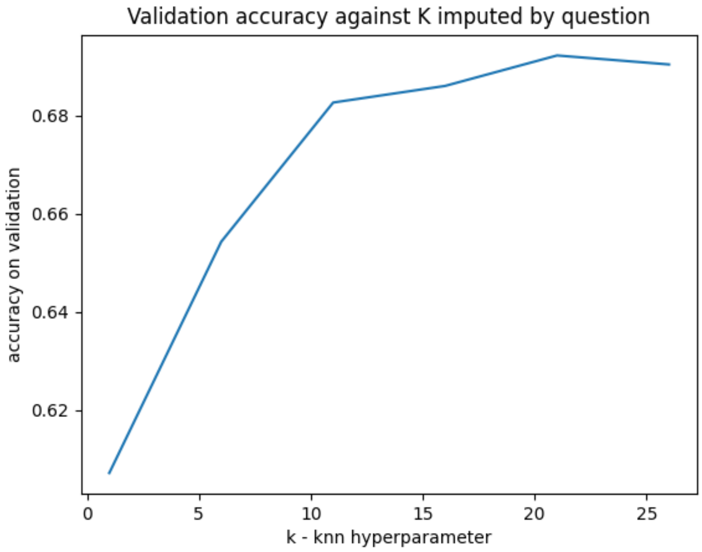
(b) We report the test accuracy with k* having the highest performance on validation data:

```
For knn imputed by users, k* = 11 has the highest performance on validation data.
The final test accuracy is 0.6841659610499576.
```

(c) For k = {1, 6, 11, 16, 21, 26}, the corresponding validation accuracy imputed by item are:

```
Knn imputed by question have accuracy
[0.607112616426757, 0.6542478125882021, 0.6826136042901496, 0.6860005644933672, 0.6922099915325995, 0
.69037538808919]
```

```
For knn imputed by questions, k* = 21 has the highest performance on validation data.
The final test accuracy is 0.6816257408975445.
```



(d) Comparing the test performance between user and item based collaborative filtering, we can see that user-based imputation has higher test accuracy of 68.42% and item-based imputation has slightly lower test accuracy of 68.16%. Thus user-based collaborative filtering method performs better.

(e) Limitation of KNN includes: slow computation with lots of memory occupied by training data

Relatively low prediction

Question 2

(a) We can first derive the probability that question j is wrongly answered by student i:

$$p(c_{ij}=0|\theta_i,\beta_j) = 1 - \frac{exp(\theta_i-\beta_j)}{1+exp(\theta_i-\beta_j)} = \frac{1}{1+e^{\theta_i-\beta_j}}$$

$$p(c_{ij}=1|\theta_i,\beta_j) = \frac{exp(\theta_i-\beta_j)}{1+exp(\theta_i-\beta_j)} = \frac{e^{\theta_i}}{e^{\theta_i}+e^{\beta_j}} = \frac{1}{1+e^{\beta_j-\theta_i}}$$

Thus the likelihood of the whole training data is:

$$\prod_{k=1}^{56688} \frac{1}{1+e^{\beta_{j(k)}-\theta_{i(k)}}}^{c_{ij(k)}} \frac{1}{1+e^{\theta_{i(k)}-\beta_{j(k)}}}^{1-c_{ij(k)}}$$

And we can derive the log-likelihood $log\ p(\mathbf{C}|\theta,\beta)$ for all students and questions:

$$\sum_{k=1}^{56688} c_{ij_{(k)}} log(1+e^{\beta_{j(k)}-\theta_{i(k)}}) + (1-c_{ij_{(k)}}) log(1+e^{\theta_{i(k)}-\beta_{j(k)}})$$

The derivative of the log-likelihood with respect to theta_i and beta_j are:

$$\frac{dl}{d\theta_i} = \sum_{k=1}^{56688} \frac{e^{\theta_{i(k)}}}{e^{\beta_{j(k)}}+e^{\theta_{i(k)}}} - c_{ij_{(k)}}$$

$$\frac{dl}{d\beta_j} = \sum_{k=1}^{56688} c_{ij_{(k)}} - \frac{e^{\theta_{i(k)}}}{e^{\beta_{j(k)}}+e^{\theta_{i(k)}}}$$

(b) Please see the relevant code at item_response.py

We chose learning rate = 0.15 and iteration = 50. The process of tuning is:

```
For learning rate = 0.1, iteration = 50, the validation accuracy is
[0.48701665255433246, 0.6905165114309907, 0.5819926615862263, 0.6655376799322608, 0.6577758961332204, 0
 .6364662715213096, 0.6549534292972058, 0.6566469093988145, 0.6299745977984759, 0.6747106971493085, 0
 .6481795088907706, 0.647897262207169, 0.6501552356759808, 0.6676545300592718, 0.6315269545582839, 0
 .663279706463449, 0.6543889359300028, 0.647897262207169, 0.6423934518769404, 0.6699125035280835, 0
 .6350550381033023, 0.6652554332486593, 0.6486028789161727, 0.6521309624611911, 0.6501552356759808, 0
 .6600338696020321, 0.6414055884843353, 0.65961049957663, 0.6572114027660175, 0.6453570420547559, 0
 .6446514253457521, 0.6718882303132938, 0.6322325712672876, 0.6622918430708439, 0.6541066892464014, 0
 .6517075924357889, 0.6477561388653683, 0.6689246401354784, 0.6343494213942986, 0.6622918430708439, 0
 .658058142816822, 0.642675698560542, 0.6512842224103866, 0.663279706463449, 0.6385831216483207, 0
 .6615862263618403, 0.6474738921817669, 0.6543889359300028, 0.6529777025119955, 0.6622918430708439]
```

We see that for learning rate = 0.1, the validation accuracy increases quickly and then oscillates a lot. Thus we try smaller learning rate.

```
For learning rate = 0.01, iteration = 50, the validation accuracy is
[0.5033869602032176, 0.635196161445103, 0.6610217329946373, 0.6742873271239063, 0.6817668642393452, 0
.6875529212531752, 0.6902342647473892, 0.694044594976009, 0.6979960485464296, 0.7003951453570421, 0
.7050522156364663, 0.7056167090036692, 0.7061812023708721, 0.7071690657634773, 0.7084391758396839, 0
.707451312447078 7, 0.7080158058142817, 0.707310189105278, 0.7064634490544736, 0.70575783234547, 0
.7053344623200677, 0.7050522156364663, 0.7044877222692634, 0.7040643522438611, 0.7050522156364663, 0
.7044877222692634, 0.7043465989274627, 0.7042054755856618, 0.7040643522438611, 0.7037821055602597, 0
.7039232289020604, 0.7042054755856618, 0.7037821055602597, 0.7047699689528648, 0.7058989556872707, 0
.7060400790290714, 0.7064634490544736, 0.7061812023708721, 0.7060400790290714, 0.7058989556872707, 0
.7060400790290714, 0.7061812023708721, 0.7060400790290714, 0.7056167090036692, 0.7054755856618685, 0
.7049110922946655, 0.7046288456110641, 0.7047699689528648, 0.7049110922946655, 0.7050522156364663]
```

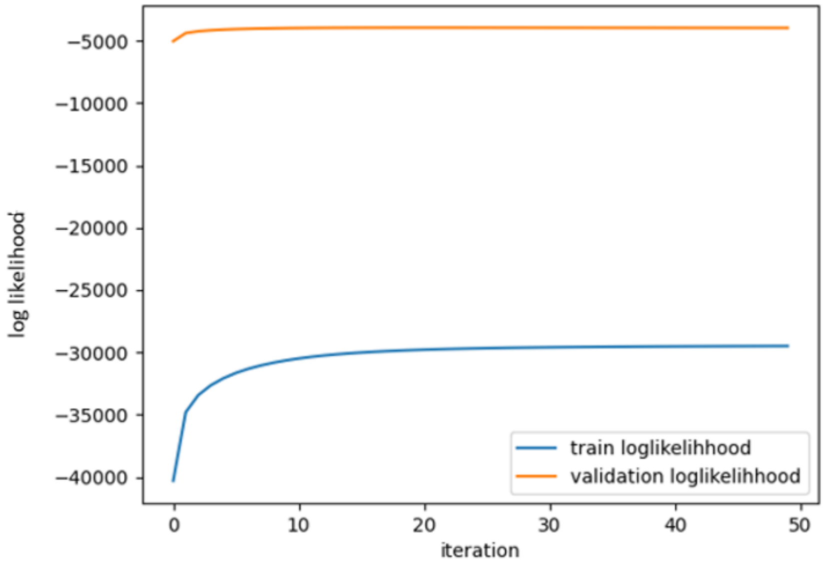
For learning rate = 0.1, we increase the iteration to check whether we miss the final convergence.

```
For learning rate = 0.01, iteration = 100, the validation accuracy is
[0.49562517640417725, 0.6450747953711544, 0.6707592435788879, 0.6819079875811459, 0.6902342647473892, 0
 .6960203217612193, 0.7001128986734406, 0.7020886254586508, 0.7036409822184589, 0.7047699689528648, 0
 .707874682472481, 0.7075924357888794, 0.7071690657634773, 0.7077335591306803, 0.7090036692068868, 0
 .7098504092576913, 0.7101326559412927, 0.7104149026248942, 0.7106971493084956, 0.7098504092576913, 0
 .7085802991814846, 0.707874682472481, 0.7074513124470787, 0.707310189105278, 0.7068868190798758, 0
 .7074513124470787, 0.707874682472481, 0.7081569291560824, 0.707874682472481, 0.7077335591306803, 0
 .7070279424216765, 0.7067456957380751, 0.7070279424216765, 0.7064634490544736, 0.7058989556872707, 0
 .7061812023708721, 0.7060400790290714, 0.7061812023708721, 0.7064634490544736, 0.7066045723962744, 0
 .7067456957380751, 0.7063223257126728, 0.7061812023708721, 0.7061812023708721, 0.7060400790290714, 0
 .7058989556872707, 0.7063223257126728, 0.7061812023708721, 0.7063223257126728, 0.7067456957380751, 0
 .7070279424216765, 0.7066045723962744, 0.7064634490544736, 0.7063223257126728, 0.7063223257126728, 0
 .70575783234547, 0.7061812023708721, 0.7058989556872707, 0.7058989556872707, 0.70575783234547, 0.7058989556872707,
  0.7063223257126728, 0.7066045723962744, 0.7064634490544736, 0.7064634490544736, 0.7066045723962744, 0
 .7067456957380751, 0.7063223257126728, 0.7064634490544736, 0.7063223257126728, 0.7063223257126728, 0
 .7063223257126728, 0.7060400790290714, 0.7058989556872707, 0.70575783234547, 0.70575783234547, 0.7054755856618685,
  0.7053344623200677, 0.7053344623200677, 0.7054755856618685, 0.7054755856618685, 0.7056167090036692, 0
 .7058989556872707, 0.7056167090036692, 0.70575783234547, 0.7058989556872707, 0.7058989556872707, 0
 .7056167090036692, 0.7056167090036692, 0.7056167090036692, 0.7056167090036692, 0.7056167090036692, 0
 .7053344623200677, 0.7056167090036692, 0.7054755856618685, 0.7056167090036692, 0.7056167090036692, 0
 .7056167090036692, 0.7054755856618685, 0.7058989556872707]
```

The results show that validation accuracy stables before the first 50 iterations. So, we set the iteration to be 50 and increase the learning rate at a small amount to 0.15.

```
For learning rate = 0.015, iteration = 50, the validation accuracy is
[0.49689528648038384, 0.6587637595258256, 0.6782387806943269, 0.6900931414055885, 0.698278295230031, 0
 .703076488851256, 0.7046288456110641, 0.7042054755856618, 0.7044877222692634, 0.7068868190798758, 0
 .7053344623200677, 0.7063223257126728, 0.7061812023708721, 0.7060400790290714, 0.7064634490544736, 0
 .70575783234547, 0.7056167090036692, 0.7058989556872707, 0.7061812023708721, 0.7061812023708721, 0
 .7058989556872707, 0.7061812023708721, 0.7058989556872707, 0.70575783234547, 0.7061812023708721, 0
 .7061812023708721, 0.7063223257126728, 0.7060400790290714, 0.70575783234547, 0.70575783234547, 0.7056167090036692,
  0.7066045723962744, 0.7066045723962744, 0.7067456957380751, 0.7068868190798758, 0.7067456957380751, 0
 .7064634490544736, 0.7064634490544736, 0.7063223257126728, 0.7064634490544736, 0.7058989556872707, 0
 .7060400790290714, 0.7058989556872707, 0.7064634490544736, 0.7060400790290714, 0.7060400790290714, 0
 .70575783234547, 0.70575783234547, 0.7054755856618685, 0.7053344623200677]
```

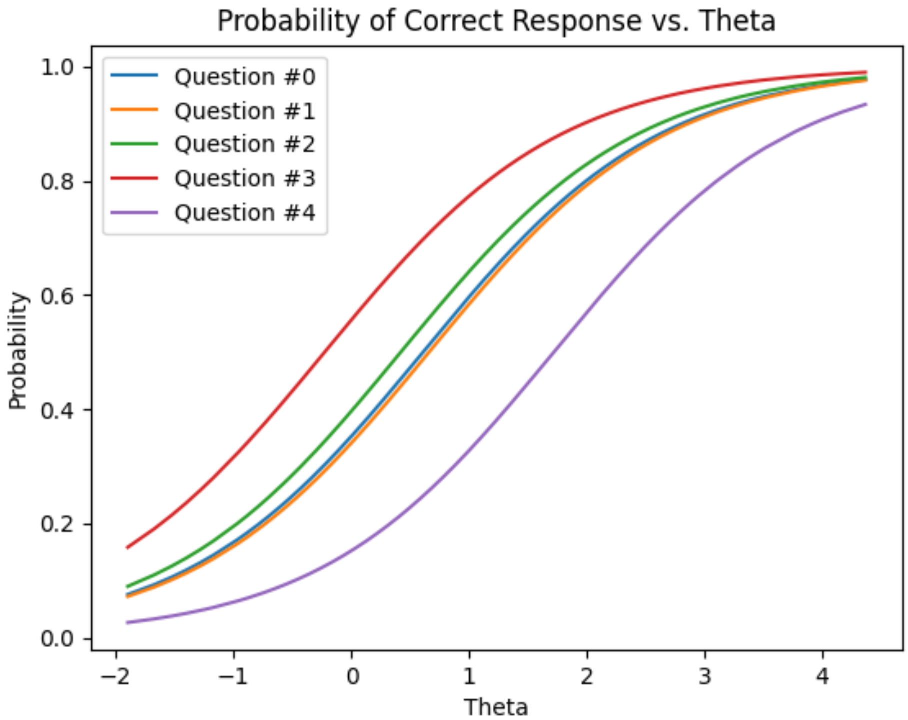
Here is the training curve that shows the training and validation log-likelihoods as a function of iteration with learning rate = 0.15, iteration = 50



(c) The final test and validation accuracy are:

```
With chosen learning rate = 0.015 and iteration = 50,
The final validation accuracy is 0.7053344623200677
The final test accuracy is 0.7064634490544736
```

(d) The probability of the correct response of question # 0, 1, 2, 3, 4 are:



We can see that the shape of the curves is sigmoid. This shape indicates that as the student ability increases, the probability of correctly answered the specific questions increases. This shape is consistent with all the randomly chosen question.

3

Option 2: Neural Networks.
While running nn, we got the UserWarning: nn.functional.sigmoid is deprecated. Use torch.sigmoid instead. However these two functions are identical, so we keep nn.functional.sigmoid, which is provided in the starter code.

(a)

ALS (Alternating Lesat Squares) is used in unsupervised learning, while neural networks is used for supervised learning.
ALS is a method used in a matrix multiplication to get to the final answer. While neural networks can be used more widely in other situations (universality).
ALS is updating one parameter at a time while fixing the other ones and alternating between the parameters, while neural networks update all the parameters at the same time using a matrix form.

(b)

Codes are in py file.

(c)

Test learning rate $\{0.01, 0.05, 0.1, 0.5\}$ and num_epoch under 30 (mainly around 10) for each k. The best validation accuracy with corresponding hyperparameters are below:

k	lr	num_epoch	accuracy
10	0.1	10	0.6862828
50	0.05	10	0.6868473
100	0.05	7	0.6803556
200	0.05	9	0.6765453
500	0.1	9	0.6732995

According to results above, $k^* = 50$.

$k = 10$:

```
Epoch: 0    Training Cost: 13509.171195  Valid Acc: 0.6193903471634208
Epoch: 1    Training Cost: 12262.740651  Valid Acc: 0.6395709850409258
Epoch: 2    Training Cost: 11545.053267  Valid Acc: 0.658058142816822
Epoch: 3    Training Cost: 10942.724810  Valid Acc: 0.6670900366920689
Epoch: 4    Training Cost: 10450.880671  Valid Acc: 0.6724527236804968
Epoch: 5    Training Cost: 10044.467480  Valid Acc: 0.68077900084674
Epoch: 6    Training Cost: 9703.641267   Valid Acc: 0.6817668642393452
Epoch: 7    Training Cost: 9410.230672   Valid Acc: 0.6814846175557437
Epoch: 8    Training Cost: 9142.778405   Valid Acc: 0.6838837143663562
Epoch: 9    Training Cost: 8909.321104   Valid Acc: 0.6862828111769687
```

$k = 50$:

```
Epoch: 0    Training Cost: 13340.221335  Valid Acc: 0.6271521309624611
Epoch: 1    Training Cost: 12292.556435  Valid Acc: 0.642111205193339
Epoch: 2    Training Cost: 11588.957260  Valid Acc: 0.6510019757267852
Epoch: 3    Training Cost: 10923.860900  Valid Acc: 0.665961049957663
Epoch: 4    Training Cost: 10311.696580  Valid Acc: 0.6752751905165114
Epoch: 5    Training Cost: 9753.755806   Valid Acc: 0.6804967541631386
Epoch: 6    Training Cost: 9235.304152   Valid Acc: 0.6867061812023709
Epoch: 7    Training Cost: 8743.937502   Valid Acc: 0.6874117979113745
Epoch: 8    Training Cost: 8274.097998   Valid Acc: 0.6869884278859724
Epoch: 9    Training Cost: 7824.574489   Valid Acc: 0.6868473045441716
```

$k = 100$:

```
Epoch: 0    Training Cost: 13454.517941  Valid Acc: 0.6255997742026531
Epoch: 1    Training Cost: 12410.300181  Valid Acc: 0.6416878351679368
Epoch: 2    Training Cost: 11614.630996  Valid Acc: 0.6589048828676263
Epoch: 3    Training Cost: 10856.562655  Valid Acc: 0.6661021732994638
Epoch: 4    Training Cost: 10151.541637  Valid Acc: 0.6756985605419137
Epoch: 5    Training Cost: 9480.503721   Valid Acc: 0.6776742873271239
Epoch: 6    Training Cost: 8818.394983   Valid Acc: 0.6803556308213379
```

$k = 200$:

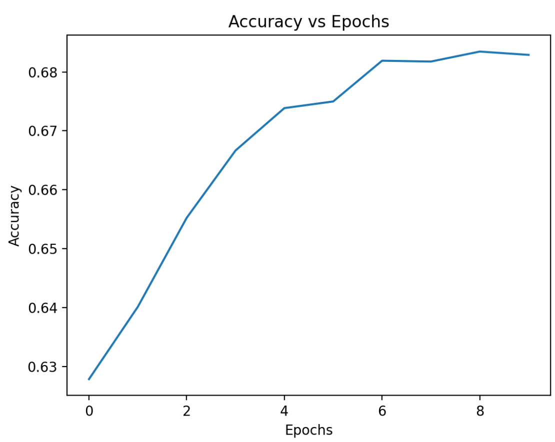
```
Epoch: 0    Training Cost: 13807.696970  Valid Acc: 0.6169912503528083
Epoch: 1    Training Cost: 12684.508000  Valid Acc: 0.6354784081287045
Epoch: 2    Training Cost: 11820.206228  Valid Acc: 0.6487440022579735
Epoch: 3    Training Cost: 11065.518775  Valid Acc: 0.6598927462602314
Epoch: 4    Training Cost: 10352.613084  Valid Acc: 0.6642675698560542
Epoch: 5    Training Cost: 9632.534934   Valid Acc: 0.6732994637313011
Epoch: 6    Training Cost: 8881.009156   Valid Acc: 0.6714648602878917
Epoch: 7    Training Cost: 8100.275627   Valid Acc: 0.6735817104149027
Epoch: 8    Training Cost: 7312.334899   Valid Acc: 0.676545300592718
```

$k = 500$:

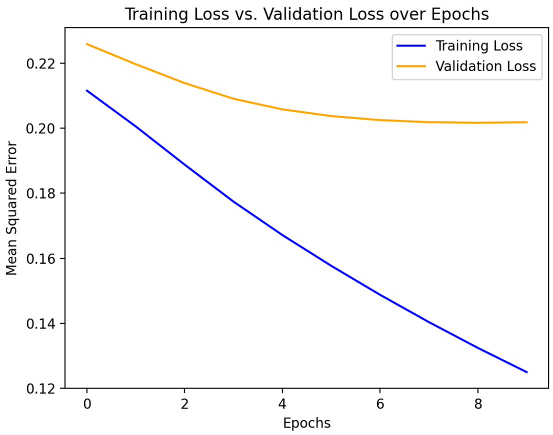
```
Epoch: 0    Training Cost: 15355.295413  Valid Acc: 0.6049957662997459
Epoch: 1    Training Cost: 13455.437402  Valid Acc: 0.6240474174428451
Epoch: 2    Training Cost: 12329.551403  Valid Acc: 0.6346316680779001
Epoch: 3    Training Cost: 11443.095316  Valid Acc: 0.6480383855489698
Epoch: 4    Training Cost: 10657.886461  Valid Acc: 0.6573525261078182
Epoch: 5    Training Cost: 9837.048679   Valid Acc: 0.6658199266158623
Epoch: 6    Training Cost: 8894.015684   Valid Acc: 0.672311600338696
Epoch: 7    Training Cost: 7817.088929   Valid Acc: 0.6732994637313011
Epoch: 8    Training Cost: 6677.067158   Valid Acc: 0.6732994637313011
```

(d)

Need to plot training loss, validation loss, and validation accuracy.
Validation accuracy and epoch:



Train & validation loss and epoch:



The validation accuracy increases as epoch increases. While the loss for both train and validation decrease as epoch increases. However, the MSE for validation decrease more and more slowly, nearly becomes flat at the end.

(e)

For $k^* = 50, lr = 0.05, num\_epoch = 10$, got this for each λ in list $\{0.001, 0.01, 0.1, 1\}$:

λ	validation acc	test acc
0.001	0.6879763	0.6838837
0.01	0.6860006	0.6771098
0.1	0.6907988	0.6827547
1	0.6241885	0.6226362

Here are results for each λ:

$\lambda = 0.001$:

```
Epoch: 0    Training Cost: 13329.817492  Valid Acc: 0.6308213378492803
Epoch: 1    Training Cost: 12269.545421  Valid Acc: 0.6406999717753317
Epoch: 2    Training Cost: 11546.206075  Valid Acc: 0.6584815128422241
Epoch: 3    Training Cost: 10883.661262  Valid Acc: 0.6679367767428732
Epoch: 4    Training Cost: 10292.029622  Valid Acc: 0.6752751905165114
Epoch: 5    Training Cost: 9758.018521   Valid Acc: 0.679226644086932
Epoch: 6    Training Cost: 9258.252411   Valid Acc: 0.682895850973751
Epoch: 7    Training Cost: 8780.376137   Valid Acc: 0.6855771944679651
Epoch: 8    Training Cost: 8319.096074   Valid Acc: 0.687694044594976
Epoch: 9    Training Cost: 7875.928446   Valid Acc: 0.6879762912785775
0.6838837143663562
```

$\lambda = 0.01$:

```
Epoch: 0    Training Cost: 13432.491016  Valid Acc: 0.6275755009878634
Epoch: 1    Training Cost: 12384.960312  Valid Acc: 0.6392887383573242
Epoch: 2    Training Cost: 11696.357103  Valid Acc: 0.6556590460062094
Epoch: 3    Training Cost: 11050.797025  Valid Acc: 0.6662432966412645
Epoch: 4    Training Cost: 10461.818962  Valid Acc: 0.6725938470222975
Epoch: 5    Training Cost: 9934.681867   Valid Acc: 0.6776742873271239
Epoch: 6    Training Cost: 9448.708539   Valid Acc: 0.6806378775049393
Epoch: 7    Training Cost: 8986.494424   Valid Acc: 0.6830369743155518
Epoch: 8    Training Cost: 8542.530875   Valid Acc: 0.6861416878351679
Epoch: 9    Training Cost: 8116.903748   Valid Acc: 0.6860005644933672
0.677109793959921
```

$\lambda = 0.1$:

```
Epoch: 0    Training Cost: 14133.327981  Valid Acc: 0.6257408975444538
Epoch: 1    Training Cost: 13232.630282  Valid Acc: 0.6394298616991251
Epoch: 2    Training Cost: 12706.057663  Valid Acc: 0.6521309624611911
Epoch: 3    Training Cost: 12237.017430  Valid Acc: 0.661727349703641
Epoch: 4    Training Cost: 11830.097083  Valid Acc: 0.6696302568444821
Epoch: 5    Training Cost: 11480.480174  Valid Acc: 0.6742873271239063
Epoch: 6    Training Cost: 11171.502251  Valid Acc: 0.6817668642393452
Epoch: 7    Training Cost: 10886.335584  Valid Acc: 0.6848715777589613
Epoch: 8    Training Cost: 10614.760466  Valid Acc: 0.6869884278859724
Epoch: 9    Training Cost: 10352.705603  Valid Acc: 0.6907987581145921
0.6827547276319503
```

$\lambda = 1$:

```
Epoch: 0    Training Cost: 18829.117218  Valid Acc: 0.6233418007338414
Epoch: 1    Training Cost: 15533.271581  Valid Acc: 0.6182613604290149
Epoch: 2    Training Cost: 14301.143335  Valid Acc: 0.6147332768839966
Epoch: 3    Training Cost: 14020.351263  Valid Acc: 0.6168501270110076
Epoch: 4    Training Cost: 13803.935659  Valid Acc: 0.619813717188823
Epoch: 5    Training Cost: 13602.622974  Valid Acc: 0.6213660739486311
Epoch: 6    Training Cost: 13421.036527  Valid Acc: 0.6212249506068304
Epoch: 7    Training Cost: 13263.540402  Valid Acc: 0.6232006773920407
Epoch: 8    Training Cost: 13131.815360  Valid Acc: 0.6230595540502399
Epoch: 9    Training Cost: 13023.678681  Valid Acc: 0.6241885407846458
0.6226361840248377
```

It seems like the model performs slightly better for $\lambda = 0.1$, which makes validation accuracy reach 0.69.

4

For this question, we used the neural networks as base model. First, we used np.random.randint() to randomly select n students with replacement for three times, where n is the total number of students. Then trained three base models corresponding to the three groups of selected data (selected rows from the data matrix where each row represents a selected students with all questionss). Then get prediction by the three models' average for evaluation.

```
Epoch: 0    Training Cost: 13571.452437  Valid Acc: 0.6079593564775614
Epoch: 1    Training Cost: 12145.219665  Valid Acc: 0.6038667795653401
Epoch: 2    Training Cost: 11077.611763  Valid Acc: 0.5970928591589049
Epoch: 3    Training Cost: 10169.012415  Valid Acc: 0.5906011854360711
Epoch: 4    Training Cost: 9452.807407   Valid Acc: 0.581145921535422
Epoch: 5    Training Cost: 8879.751910   Valid Acc: 0.581145921535422
Epoch: 6    Training Cost: 8407.603848   Valid Acc: 0.5817104149026249
Epoch: 7    Training Cost: 8010.175925   Valid Acc: 0.5798758114592154
Epoch: 8    Training Cost: 7670.466305   Valid Acc: 0.5786057013830087
Epoch: 9    Training Cost: 7376.891085   Valid Acc: 0.578464578041208
Epoch: 0    Training Cost: 13467.668539  Valid Acc: 0.6011854360711262
Epoch: 1    Training Cost: 12387.411854  Valid Acc: 0.6007620660457239
Epoch: 2    Training Cost: 11526.810686  Valid Acc: 0.596528365791702
Epoch: 3    Training Cost: 10690.442652  Valid Acc: 0.5930002822466836
Epoch: 4    Training Cost: 9994.157496   Valid Acc: 0.586931978549252
Epoch: 5    Training Cost: 9415.572773   Valid Acc: 0.5824160316116286
Epoch: 6    Training Cost: 8923.523271   Valid Acc: 0.5776178379904036
Epoch: 7    Training Cost: 8501.869490   Valid Acc: 0.5771944679650014
Epoch: 8    Training Cost: 8138.308293   Valid Acc: 0.5767710979395992
Epoch: 9    Training Cost: 7822.550102   Valid Acc: 0.574795371154389
Epoch: 0    Training Cost: 14249.315764  Valid Acc: 0.6079593564775614
Epoch: 1    Training Cost: 12882.573984  Valid Acc: 0.6104995766299746
Epoch: 2    Training Cost: 11897.499172  Valid Acc: 0.6064069997177534
Epoch: 3    Training Cost: 11055.499102  Valid Acc: 0.6034434095399379
Epoch: 4    Training Cost: 10356.602646  Valid Acc: 0.6013265594129269
Epoch: 5    Training Cost: 9763.409961   Valid Acc: 0.5992097092859159
Epoch: 6    Training Cost: 9253.768347   Valid Acc: 0.599633079311318
Epoch: 7    Training Cost: 8810.833066   Valid Acc: 0.5997742026531189
Epoch: 8    Training Cost: 8423.033213   Valid Acc: 0.5977984758679086
Epoch: 9    Training Cost: 8082.173427   Valid Acc: 0.5953993790572961
0.5935647756138865
0.5901778154106689
```

The final test accuracy is 0.5901778154106689 which is much lower than the single neural networks model. The final validation accuracy is 0.5935647756138865, also much lower than before.
Therefore, actually we didn't obtain better performance using the ensemble, and the running time for running ensemble once is quite long.
We think one main reason why accuracy decline could because when sampling with replacement, some data is not sampled into all of the three samples. Also, some model with wrong prediction may have higher weight for its wrong prediction which lead to a wrong results finally.

Part B

1 Formal Description

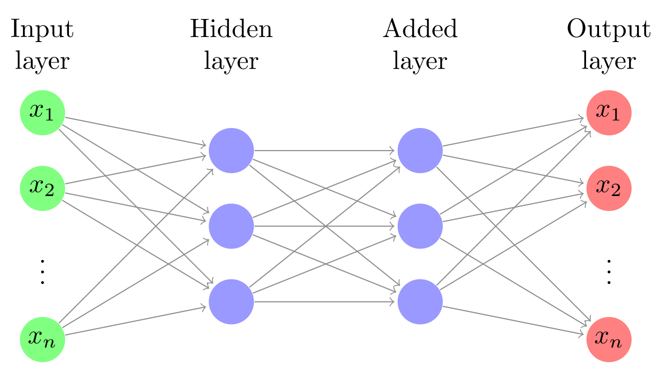
We have tried multiple ways to modify the implemented algorithm in part A to increase the test accuracy on students' answers to the diagnostic questions. Above all, we tried to consider the improvement on the neural networks algorithm. Since tuning hyperparameter is already done in Part A, we tried something else. For example, first we tried to add one hidden layer. Then we tried to use other activation functions like ReLU, Softmax, and LogSigmoid. Also, we tried different regularizer (ie. L_1 regularization). Finally we also thought about normalize the input data; however, since the input data is not in different scale we didn't modify the normalization.

After examining all the possible modifications on the neural network algorithm, we took a deeper look at the the student metadata, `student_meta.csv`. This dataset is composed of the user id, gender, date of birth and premium pupil. As the training accuracy of the baseline algorithms in part A is not too good, we hypothesized that there might exists a mixture of data where each group of data has its own density for the provided training, validation and test dataset. If we train the model with the combined data, the model may be weak to capture the overall existed pattern, and this may cause underfitting. The gender and age of users may both be the factor that affect the probability of answering questions correctly. This makes sense as younger students normally have lower accuracy, and as children get older they tend to have higher accuracy. Therefore, we separate the training, validation and test data into individual dictionaries based on gender and age, then train the item-response probability algorithm on each of them.

The separation of dataset based on age is at `age.py` and the separation of dataset based on gender is at `gender.py`. We expect the separate training may increase the train accuracy and reduce underfit, and optimally increase the test accuracy.

2 Figure or Diagram

Neural Network Modification



Add a hidden layer, also update the regularizer:

```
# Define linear functions.
self.g = nn.Linear(num_question, k)
self.s = nn.Linear(k, k)
self.h = nn.Linear(k, num_question)
```

```
g_w_norm = torch.norm(self.g.weight, 2)
s_w_norm = torch.norm(self.s.weight, 2)
h_w_norm = torch.norm(self.h.weight, 2)
```

Tried different activation functions with the added new layer:

```
activate_g = self.g(inputs)
output1 = F.sigmoid(activate_g)
# output = torch.sigmoid(activate_g)
# m = nn.ReLU()
# m = nn.LogSigmoid()
# m = nn.LogSoftmax()
# output = m(activate_g)
activate_s = self.s(output1)
output2 = F.sigmoid(activate_s)
activate_f = self.h(output2)
out = F.sigmoid(activate_f)
# out = torch.sigmoid(activate_f)
# out = m(activate_f)
```

Group separation using Student Metadata

This is the code snippets implemented the separerate training of each group data based on gender information provided in the student metadata.

```
# separate the train, validation, and test data by gender
boy, girl = separate_gender()
b_test, g_test = get_dic(boy, girl, test_data)
b_train, g_train = get_dic(boy, girl, train_data)
b_val, g_val = get_dic(boy, girl, val_data)

# train each male and female datasets separately
boy = irt(b_train, b_val, 0.015, 50)
girl = irt(g_train, g_val, 0.015, 50)
print(f"For learning rate = 0.015, iteration = 50, the train accuracy for boys is\n{boy[2]}")
print(f"For learning rate = 0.015, iteration = 50, the train accuracy for girls is\n{girl[2]}")
```

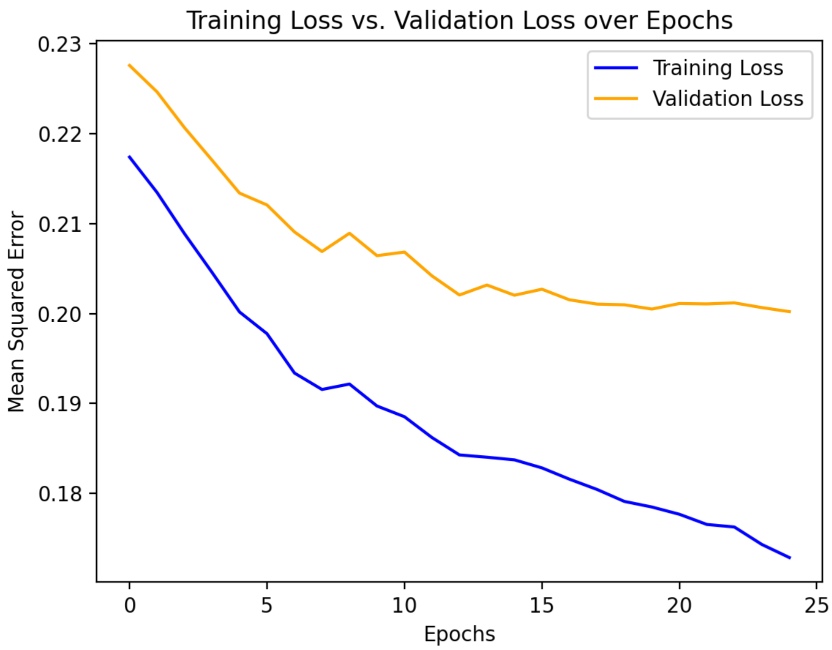
3 Comparison or Demonstration

Neural Network Modification

Here are results for using Sigmoid with an additional hidden layer:

```
Epoch: 0    Training Cost: 14173.529993  Valid Acc: 0.6277166243296641
Epoch: 1    Training Cost: 13453.212581  Valid Acc: 0.6305390911656789
Epoch: 2    Training Cost: 13106.100574  Valid Acc: 0.6349139147615016
Epoch: 3    Training Cost: 12803.304464  Valid Acc: 0.6412644651425345
Epoch: 4    Training Cost: 12560.058697  Valid Acc: 0.648461755574372
Epoch: 5    Training Cost: 12356.876511  Valid Acc: 0.6549534292972058
Epoch: 6    Training Cost: 12266.673736  Valid Acc: 0.6668077900084673
Epoch: 7    Training Cost: 12193.919032  Valid Acc: 0.668077900084674
Epoch: 8    Training Cost: 12119.076354  Valid Acc: 0.668077900084674
Epoch: 9    Training Cost: 12077.401896  Valid Acc: 0.6737228337567034
Epoch: 10   Training Cost: 12133.146861  Valid Acc: 0.6721704769968952
Epoch: 11   Training Cost: 12035.328566  Valid Acc: 0.6747106971493085
Epoch: 12   Training Cost: 11957.835894  Valid Acc: 0.6776742873271239
Epoch: 13   Training Cost: 11905.195812  Valid Acc: 0.677109793959921
Epoch: 14   Training Cost: 11822.732113  Valid Acc: 0.6806378775049393
Epoch: 15   Training Cost: 11807.365512  Valid Acc: 0.6779565340107254
Epoch: 16   Training Cost: 11770.325193  Valid Acc: 0.6827547276319503
Epoch: 17   Training Cost: 11752.178491  Valid Acc: 0.6816257408975445
Epoch: 18   Training Cost: 11741.590530  Valid Acc: 0.682895850973751
Epoch: 19   Training Cost: 11678.233868  Valid Acc: 0.6844482077335591
Epoch: 20   Training Cost: 11618.268810  Valid Acc: 0.6833192209991532
Epoch: 21   Training Cost: 11622.833282  Valid Acc: 0.6827547276319503
Epoch: 22   Training Cost: 11655.201063  Valid Acc: 0.6864239345187694
Epoch: 23   Training Cost: 11599.434348  Valid Acc: 0.6857183178097658
Epoch: 24   Training Cost: 11532.738863  Valid Acc: 0.6850127011007621
0.6782387806943269
0.6850127011007621
```

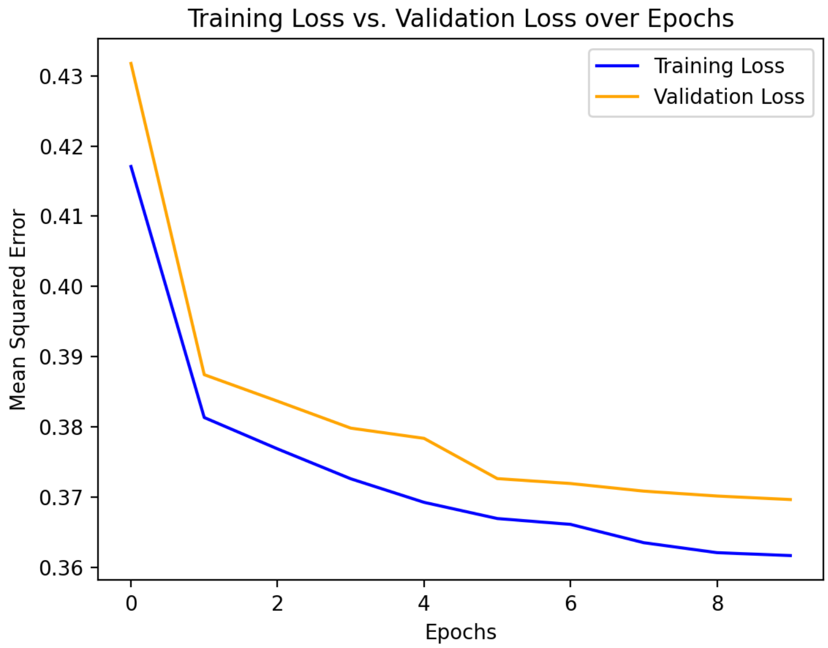
Where final validation accuracy is 0.6850127011007621 and test accuracy is 0.6782387806943269.



However, there's no obvious changing for using this.
Then by using ReLU as activation function, we got this:

```
Epoch: 0    Training Cost: 43074.088623  Valid Acc: 0.5193338978267006
Epoch: 1    Training Cost: 35417.084000  Valid Acc: 0.5397967823878069
Epoch: 2    Training Cost: 33547.767464  Valid Acc: 0.5438893593000282
Epoch: 3    Training Cost: 33337.407709  Valid Acc: 0.5467118261360429
Epoch: 4    Training Cost: 32972.840151  Valid Acc: 0.5496754163138583
Epoch: 5    Training Cost: 32852.308329  Valid Acc: 0.5491109229466554
Epoch: 6    Training Cost: 32645.523170  Valid Acc: 0.5517922664408693
Epoch: 7    Training Cost: 32485.007030  Valid Acc: 0.5534857465424782
Epoch: 8    Training Cost: 32290.693451  Valid Acc: 0.5532034998588766
Epoch: 9    Training Cost: 32172.030867  Valid Acc: 0.554614733276884
0.563646627152131
```

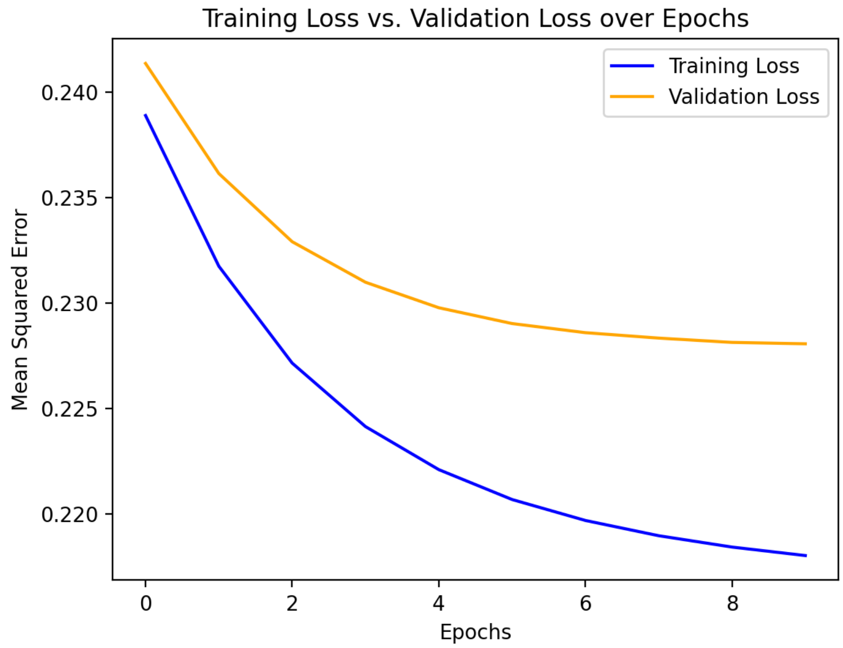
Validation accuracy is 0.554614733276884 and test accuracy is 0.563646627152131.



This also doesn't help, since ReLU is not a better activation function compared to Sigmoid. We then tried L_1 regularization:

```
Epoch: 0    Training Cost: 27082.602104  Valid Acc: 0.6123341800733841
Epoch: 1    Training Cost: 20425.982297  Valid Acc: 0.6233418007338414
Epoch: 2    Training Cost: 20050.000346  Valid Acc: 0.6253175275190517
Epoch: 3    Training Cost: 19795.116239  Valid Acc: 0.6270110076206604
Epoch: 4    Training Cost: 19618.048839  Valid Acc: 0.6270110076206604
Epoch: 5    Training Cost: 19489.839846  Valid Acc: 0.6275755009878634
Epoch: 6    Training Cost: 19395.148801  Valid Acc: 0.626728760937059
Epoch: 7    Training Cost: 19324.223396  Valid Acc: 0.6264465142534575
Epoch: 8    Training Cost: 19267.709228  Valid Acc: 0.6258820208862546
Epoch: 9    Training Cost: 19223.142982  Valid Acc: 0.6254586508608524
0.6223539373412362
```

Validation accuracy is 0.6254586508608524 and test accuracy is 0.6223539373412362.



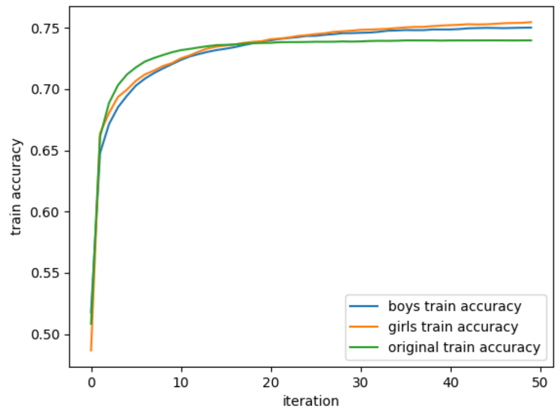
Similarly, this doesn't help to improve our results.

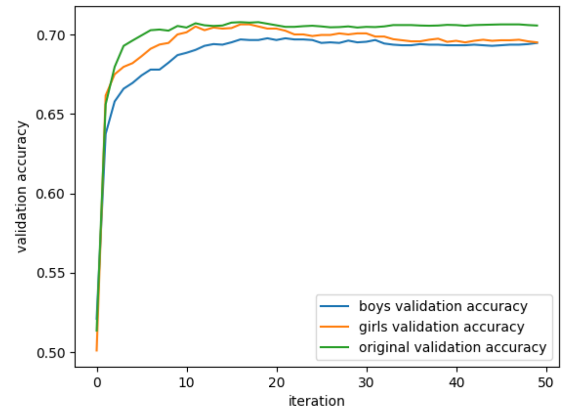
Group separation using Student Metadata

We can see that there is significant increase of training accuracy for all the gender groups. We can observe that the highest train accuracy for boy group has reached 75%. However, the validation accuracy have shown no improvement. The test accuracy of one group increases 2% and the other group stays the same.

```
For learning rate = 0.015, iteration = 50, the train accuracy for boys is
[0.5176057984568623, 0.6478372691138649, 0.6714051905541267, 0.6852466682253916, 0.6946457797521627, 0
 .7029693710544774, 0.7085807809212065, 0.7130699088145896, 0.7169043722235212, 0.7203647416413373, 0
 .723871872808043, 0.7267243394902969, 0.7286883329436521, 0.7305120411503391, 0.7319616553659107, 0
 .7330371755903671, 0.7343465045592705, 0.7359831657703998, 0.7375263034837503, 0.7389291559504325, 0
 .7398643909282208, 0.7409866729015665, 0.7417816226326864, 0.7424830488660276, 0.7434182838438158, 0
 .743558569090484, 0.7443535188216039, 0.7449146598082769, 0.7457096095393968, 0.7457563712882862, 0
 .7460369417816226, 0.7462239887771802, 0.7467851297638531, 0.7477203647416414, 0.747907411737199, 0
 .7482815057283142, 0.7481879822305354, 0.7481879822305354, 0.7487491232172083, 0.7487491232172083, 0
 .7487491232172083, 0.7490764554594341, 0.7496843581949965, 0.7498714051905542, 0.7500584521861118, 0
 .749964928688333, 0.7498246434416648, 0.7500116904372224, 0.7501519756838906, 0.7502922609305588]
```

```
With chosen learning rate = 0.015 and iteration = 50,
The final test accuracy for age 7-13 is 0.6238532110091743
The final test accuracy for age 13-16 is 0.7026178010471205
The final test accuracy for age 16-18 is 0.7263479145473042
```

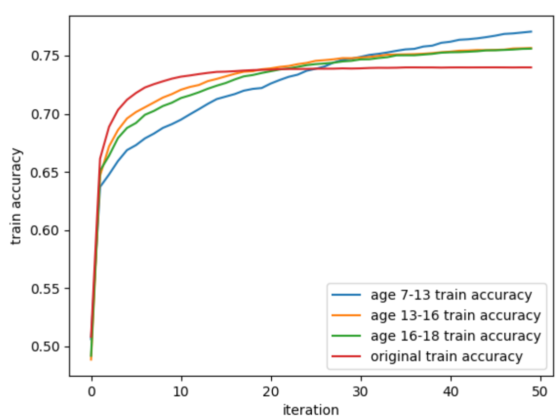


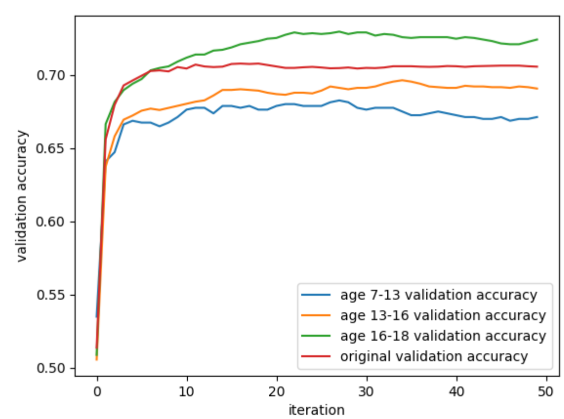


The train accuracy of all the age group separation increased to a higher extent, and we can observe that the highest train accuracy for age 7-13 has reached 76.8%. However, the validation accuracy and test accuracy have shown an overall no improvement.

```
For learning rate = 0.015, iteration = 50, the train accuracy for age 7-13 is
[0.4785867237687366, 0.637503823799327, 0.6494340776996024, 0.6563169164882227, 0.6648822269807281, 0
.671612113796268, 0.6751300091771184, 0.6789538085041297, 0.6838482716427042, 0.6882838788620373, 0
.6916488222698073, 0.6951667176506577, 0.6991434689507494, 0.7018966044661976, 0.7055674518201285, 0
.7106148669317834, 0.7129091465279902, 0.7184154175588865, 0.7213215050474151, 0.7245334964821046, 0
.7274395839706332, 0.7294279596206791, 0.7315692872438054, 0.7330988069746099, 0.7358519424900581, 0
.737840318140104, 0.7396757418170694, 0.7416641174671154, 0.7438054450902417, 0.7454879167941266, 0
.7470174365249311, 0.7486999082288162, 0.7503823799327012, 0.752370755582747, 0.7534414193943102, 0
.7543591312327929, 0.755429795044356, 0.7561945549097583, 0.7565004588559192, 0.7578770266136433, 0
.7601713062098501, 0.760477210156011, 0.7615478739675742, 0.7617008259406546, 0.7641480575099419, 0
.7649128173753441, 0.7661364331599878, 0.767207096971551, 0.7670541449984705, 0.7678189048638727]
```

```
With chosen learning rate = 0.015 and iteration = 50,
The final test accuracy for age 7-13 is 0.6238532110091743
The final test accuracy for age 13-16 is 0.7026178010471205
The final test accuracy for age 16-18 is 0.7263479145473042
```





4 Limitations

- Adding hidden layer requires tuning for hyperparameters again. Since the original k may lead to overfitting, while the original learning rate and number of iterations may lead to relatively longer running time. However, adding 1 layer seems to help less, doesn't contribute to both accuracy and loss in our experiment.
- One possible extension could be, trying to add more hidden layers, then tune hyperparameters to avoid overfitting. We think this may help to improve accuracy. Also, maybe mixing all methods (ie. k, number of hidden layers, activation function) we tried to use could also bring improvements. Since there could be some better combinations if try more.
- The data processing for converting given train dataset into spare matrix is time consuming and non-efficient.
- Even though we reduce the underfitting and increase train accuracy, the Group separation using student metadata is not modifying the algorithm, but focuses on data processing. The improved train accuracy would not apply to a new dataset.

Contribution

Group Member: Peiqing Yu

- Part A: `knn.py`
- Part A: `item_response.py`
- Part A: report question 1, question 2
- Part B: `irt_partB.py`
- Part B: `modified_nn.py`
- Part B: half of the report

Group Member: Chunjing Zhang

- Part A: `neural_network.py`
- Part A: `esemble.py`
- Part A: report question 3, question 4
- Part B: `age.py`
- Part B: `gender.py`
- Part B: half of the report