CSC311 Final Report

2021 Fall

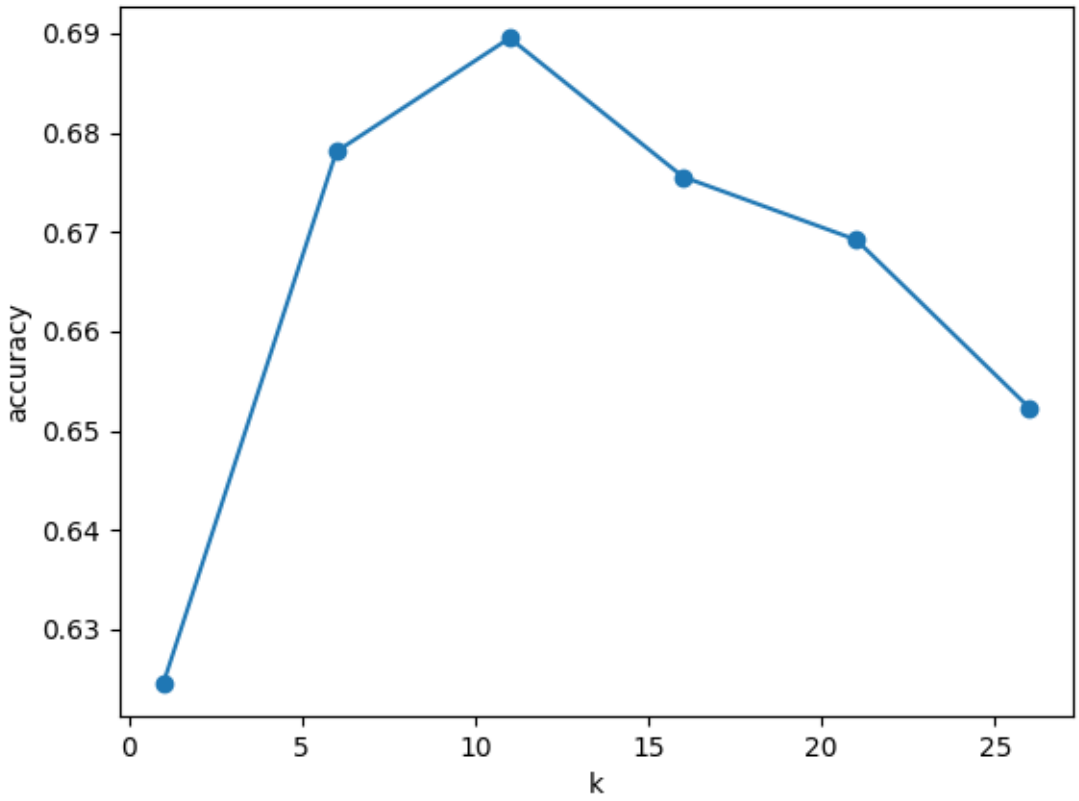
Contributions by team member:

- Part A
  - Q1
    - Xiaoxue Yang: Completed code and report
  - Q2
    - Luomeng Tan: Completed code and report
  - Q3
    - Jingyu Hu: Completed code and report
  - Q4
    - Xiaoxue Yang: Completed code and report
- Part B
  - Q1
    - Jingyu Hu: Completed report and code for model modification
  - Q2
    - Jingyu Hu: Drew the diagram for the model
  - Q3
    - Luomeng Tan: Completed report and code for plotting
  - Q4
    - Xiaoxue Yang: Completed report, implemented code for attempting to address the "cold start" problem as part of the experimentation process

Part A

1.

(a) For user-based collaborative filtering: the accuracy on the validation data as a function of k:

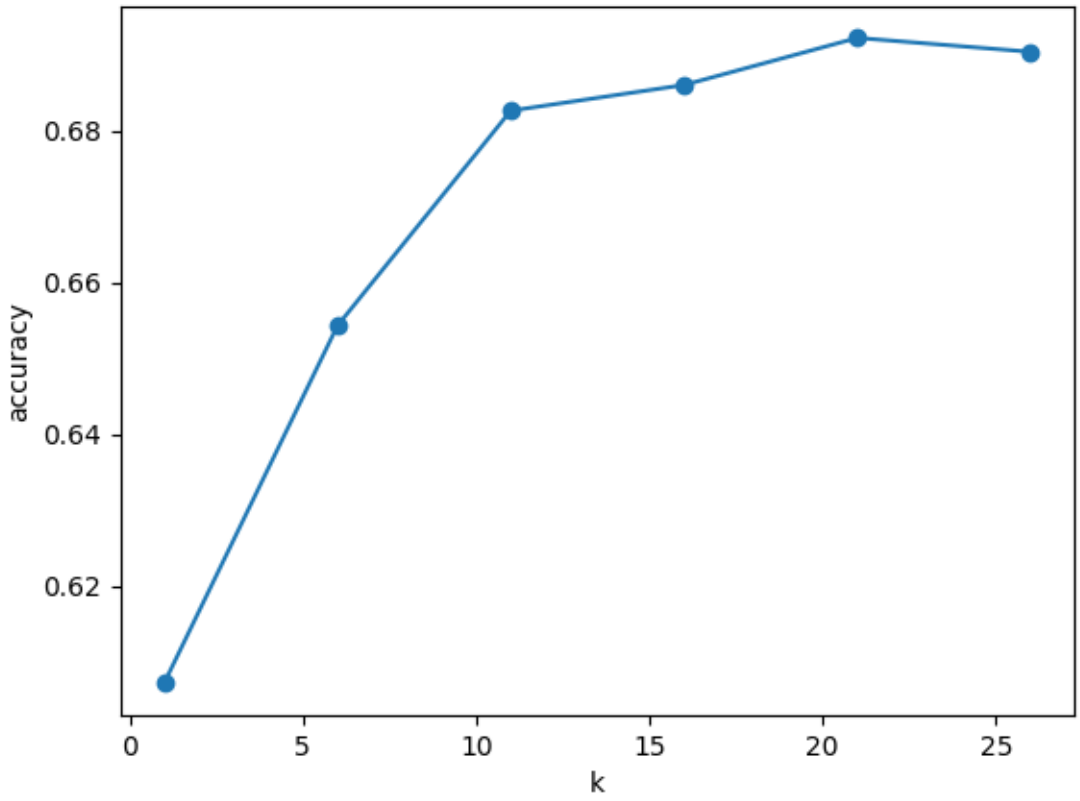


(b) $k^* = 11$. The final test accuracy is 0.6841659610499576.

(c) The underlying assumption for item-based collaborative filtering is that if students answer both question A and question B correctly or incorrectly, then whether question A is answered correctly by a new student

matches whether question B is answered correctly by that student.

For item-based collaborative filtering: the accuracy on the validation data as a function of k:



$k^* = 21$ has the highest performance on validation data. The final test accuracy is 0.6816257408975445.

(d) User-based collaborative filtering has a slightly better test performance (accuracy) than item-based collaborative filtering.

(e)
- If we were to run kNN on a larger dataset with more questions and more students, and run more frequently, the test time and memory cost would be huge.
- If some of the questions are only answered by a few students (item-based collaborative filtering), or if some of the students have only answered a few questions (user-based collaborative filtering), the patterns found by kNN could be very noisy and therefore the predictions may not be accurate and generalizable.

2.

(a) Let N be the number of students and D be the number of questions, so C is a $N \times D$ sparse matrix. Then we derive the log-likelihood:

$$p(C \mid \theta, \beta) = \prod_{i=1}^{N} \prod_{j=1}^{D} p(c_{ij} = 1 \mid \theta_i, \beta_j)^{c_{ij}} (1 - p(c_{ij} = 1 \mid \theta_i, \beta_j))^{1-c_{ij}}$$

$$\begin{aligned} \log(p(C \mid \theta, \beta)) &= \sum_{i=1}^{N} \sum_{j=1}^{D} c_{ij} \log(p(c_{ij} = 1 \mid \theta_i, \beta_j)) + (1 - c_{ij}) \log(1 - p(c_{ij} = 1 \mid \theta_i, \beta_j)) \\ &= \sum_{i=1}^{N} \sum_{j=1}^{D} c_{ij}(\theta_i - \beta_j) - \log(1 + \exp(\theta_i - \beta_j) \mathbb{1}(c_{ij} \in 0, 1) \end{aligned}$$

We add the indicator function at the last line because the sparse matrix C also has NaN entries, but we only want to consider the case where the user did the question, so c_{ij} can be either 0 or 1.

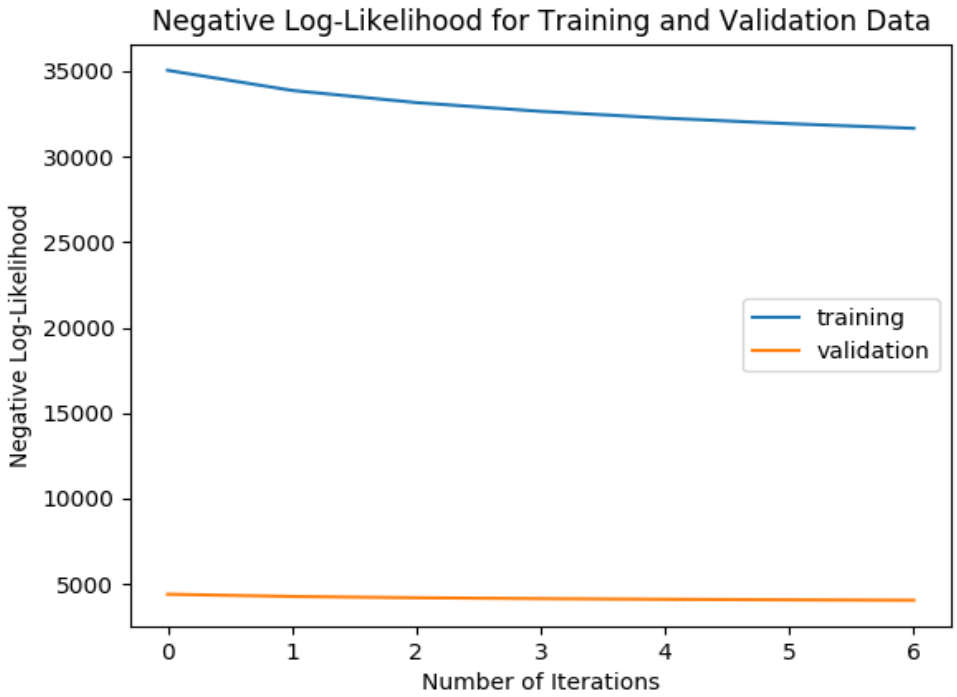
The derivatives are:

$$\frac{\partial \log(p(C \mid \theta, \beta))}{\partial \theta_i} = \sum_{j=1}^{D} (c_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}) \mathbb{1}(c_{ij} \in 0, 1)$$

$$\frac{\partial \log(p(C \mid \theta, \beta))}{\partial \beta_j} = \sum_{i=1}^{N} (-c_{ij} + \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)}) \mathbb{1}(c_{ij} \in 0, 1)$$

(b) In the part, we use the following hyperparameters:

- initialized theta and beta: all zeros
- learning rate: 0.01
- number of iterations: 7

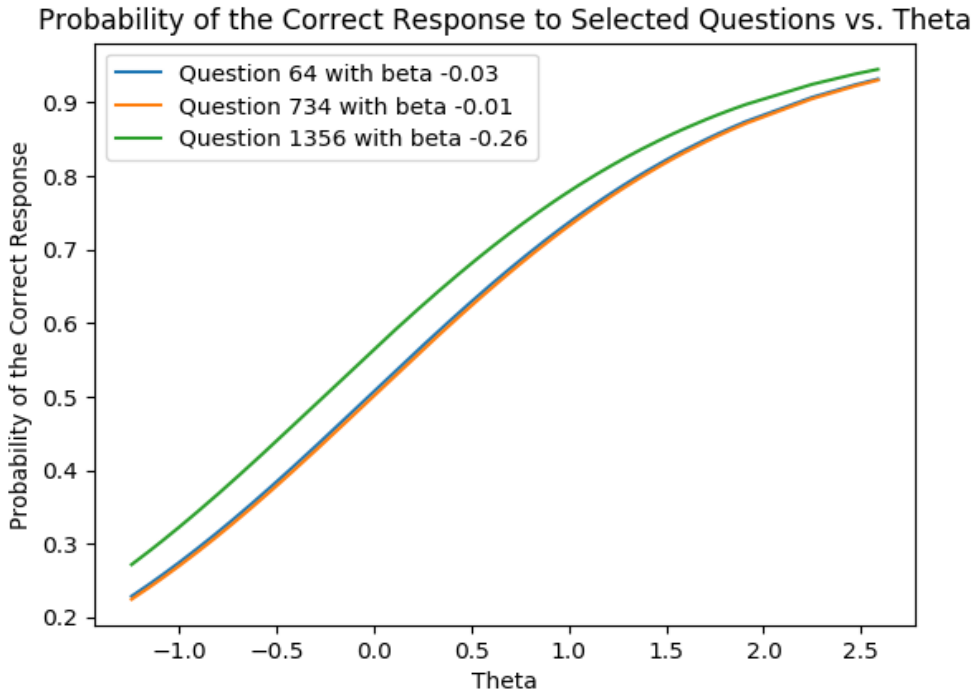


(c) validation accuracy: 0.7088625458650861

test accuracy: 0.6971493084956252

(d) The shape of the curves is similar to the shape of sigmoid function, which is as expected since we are using sigmoid function to calculate the probability. The x-axis is theta, which represents the ability of the student. The curves shows a positive correlation between the ability of students and the probability of the correct response. That is to say, for a given question, students with more ability have a higher chance of answering the question correctly, which is what the model expects. Also, at a given theta value, questions with higher beta value have a

lower probability of the correct response, which means more difficult questions are less likely to be answer correctly by a given student.



3.

a): Three differences between Alternating Least Squares and Neural Network:

1. Neural network updates all parameters during its backward pass. But updating ALT's parameters requires fixing one of z and u and update the other.
2. The training objectives of matrix factorization using ALS and neural network using gradient descent are different.
   - The training objective of matrix factorization using ALS is to minimize $\min_{\mathbf{U},\mathbf{Z}} \frac{1}{2} \sum_{(i,j)\in O} (R_{ij} - \mathbf{u_i}^\top \mathbf{z_j})^2$, where R is the sparse matrix and

O is the set of observed entries in R, $\mathbf{U}$ is the latent representation of users in K-dimensional space, and $\mathbf{Z}$ is the latent representation of questions in K-dimensional space. So there are 2 K-dimensional latent variables we need to optimize for, i.e., matrix factorization using ALS needs to get an optimal solution for both $\mathbf{U}$ and $\mathbf{Z}$.

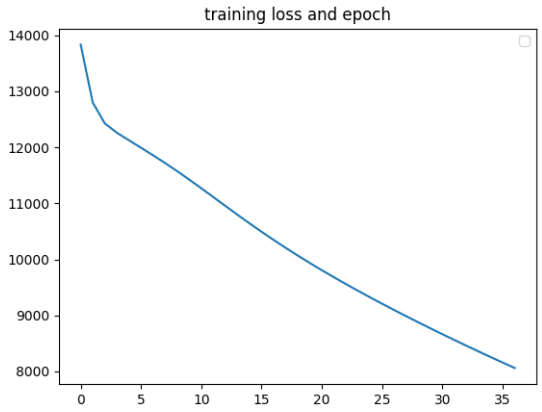
- The training objective of the neural network is $\min_{\boldsymbol{\theta}} \sum_{\mathbf{v} \in S} \|\mathbf{v} - f(\mathbf{v}; \boldsymbol{\theta})\|_2^2$, where $\mathbf{S}$ is the set of users, $\mathbf{v}$ is a user in the set of users, and $f(\mathbf{v}; \boldsymbol{\theta}) = h(\mathbf{W^{(2)}} g(\mathbf{W^{(1)}} \mathbf{v} + \mathbf{b^{(1)}}) + \mathbf{b^{(2)}})$ is the reconstruction of the input $\mathbf{v}$ in a K-dimensional subspace. So there is only one K-dimensional latent variable that we are trying to optimize for, which is the K-dimensional code vector for all users $\mathbf{v} \in S$.

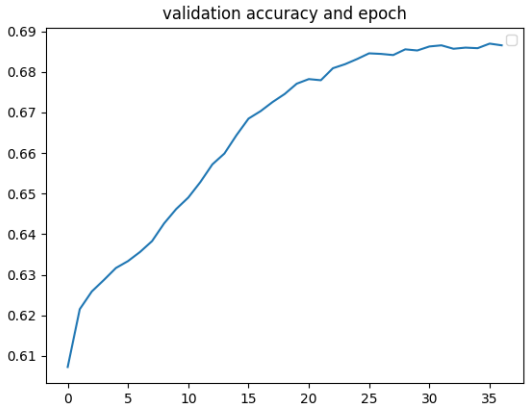
3. Matrix factorization using ALS will get an optimal K-dimensional latent factor for each student and each question, while neural network with gradient descent will only get one optimal K-dimensional latent factor for all student vectors (each student vector is a row in the sparse matrix).

4. Matrix factorization with ALS is a linear problem as it just factorizes the sparse matrix into 2 K-dimensional matrices, while neural network could learn non-linear latent representations as well.

c): After tuning through parameters, when $k = 50$ along with other hyperparameters, the validation accuracy achieves the highest, approximately 0.6875.

d):





The final test accuracy for chosen hyperparameters is 0.6824 (rounded).

e): By tuning the hyperparameter lambda, the best value is $\lambda = 0.001$. Final validation accuracy is: 0.6854 (rounded), and test accuracy is 0.6889 (rounded).
Based on final testing data, the new model trained with regularizer performs slightly better than the original model.

4.

We decided to build an ensemble of 3 autoencoder models.

First, we bootstrap the 2D sparse train matrix by user_id with replacement 3 times to generate 3 bootstrap samples of the same size as the original sparse train_matrix, one bootstrap sample for each of the models in the ensemble.

Then, with each bootstrap sample of the 2D sparse train matrix, we train an autoencoder model and evaluate the trained model on the original 2D sparse matrix (not bootstrapped) and validation data to get validation accuracy. We pick 3 sets of hyperparameters for each bootstrap sample, and choose the set of hyperparameters that generates the highest validation accuracy.

For each bootstrap sample, after we narrow down to 1 set of hyperparameters that results in the best validation accuracy, we train an autoencoder using this set of hyperparameters and evaluate on validation data to get the val-

idation predictions, and also evaluate on test data to get the test predictions.

To generate the ensemble predictions on validation data, we average the predictions generated by each of the individual autoencoder models on validation data, then compare the averaged predictions with the threshold (0.5), if an averaged prediction is ≥ 0.5, then that ensemble prediction will be 1, otherwise it will be 0. We repeat the same steps (but using predictions generated by each of the individual autoencoder models on test data) to generate the ensemble predictions on test data.

The final validation accuracy of the ensemble is 0.6725938470222975, and the final test accuracy of the ensemble is 0.6663844199830652. We obtained slightly worse performance using an ensemble of 3 autoencoder models compared to the performance of base autoencoder model from Q3 (trained on the entire training dataset without bootstrapping). The reason could be that, since we are only generating 3 bootstrap samples, the variation in the data from bootstrapping does not compensate for the loss of information in each bootstrap sample (since we are bootstrapping with replacement, some users might appear multiple times in the bootstrap sample while some users might not appear at all).

Part B

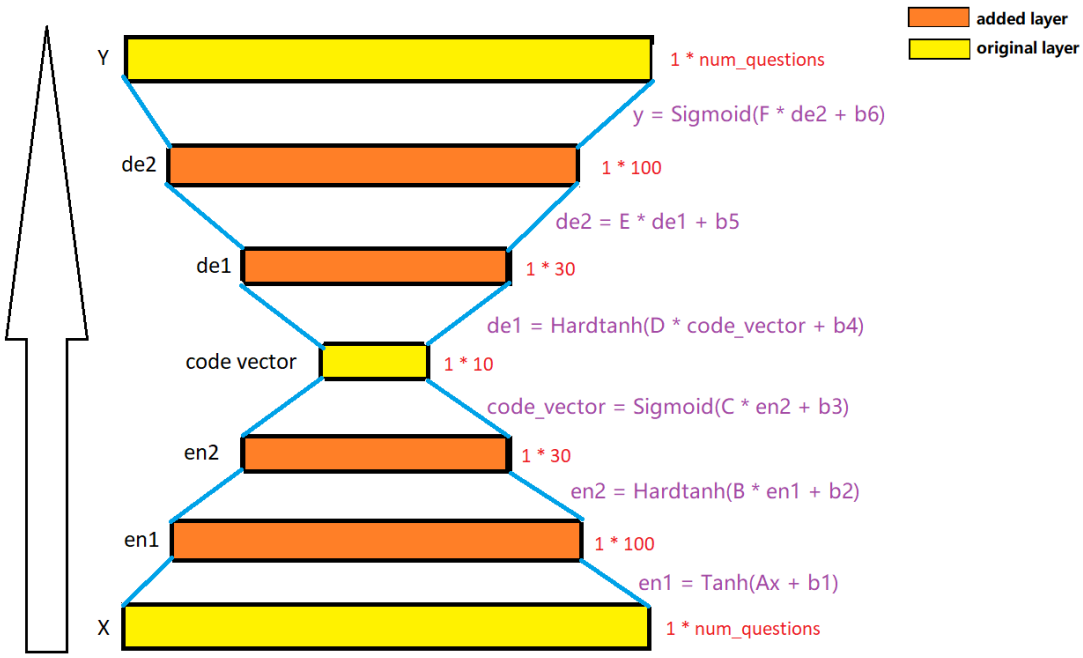
1.

To improve the accuracy of our autoencoder model, we decided to add extra hidden layers with activation functions, in an attempt to achieve a better compressed coding of features through multi-layer gradual encoding and decoding. The modified model can be found in the file part_b/NN_modified_more-layers.py.
The modified model will accept user's answer towards each question as input (same as original model), and has five hidden layers, where first two are intermediate encoding layers, last two are decoding layers with the third layer as coding vector.
The forward pass of modified model is as below: (x is input vector)

$$en_1 = Tanh(A * x + b_1) \tag{1}$$
$$en_2 = Hardtanh(B * en_1 + b_2) \tag{2}$$
$$code\_vector = Sigmoid(C * en_2 + b_3) \tag{3}$$
$$de_1 = Hardtanh(D * code\_vector + b_4) \tag{4}$$
$$de_2 = E * de_1 + b_5 \tag{5}$$
$$y = Sigmoid(F * de_2 + b_6) \tag{6}$$

In the above forward pass, en denotes layers for encoding, and de is the layer for decoding. Both en and de are vectors. A, B, C, D, E, F are all real matrices, act as linear function. "b" stands for bias.
Each hidden layer of the model has number of elements as follows: x and y have "number_of_question" many, $en1$ and $de1$ has 100 many, $en2$ and $de2$ has 30 many, $code\_vector$ has 10 many.
With extra hidden layers provided, the model is expected to have a higher accuracy than original one.

2.

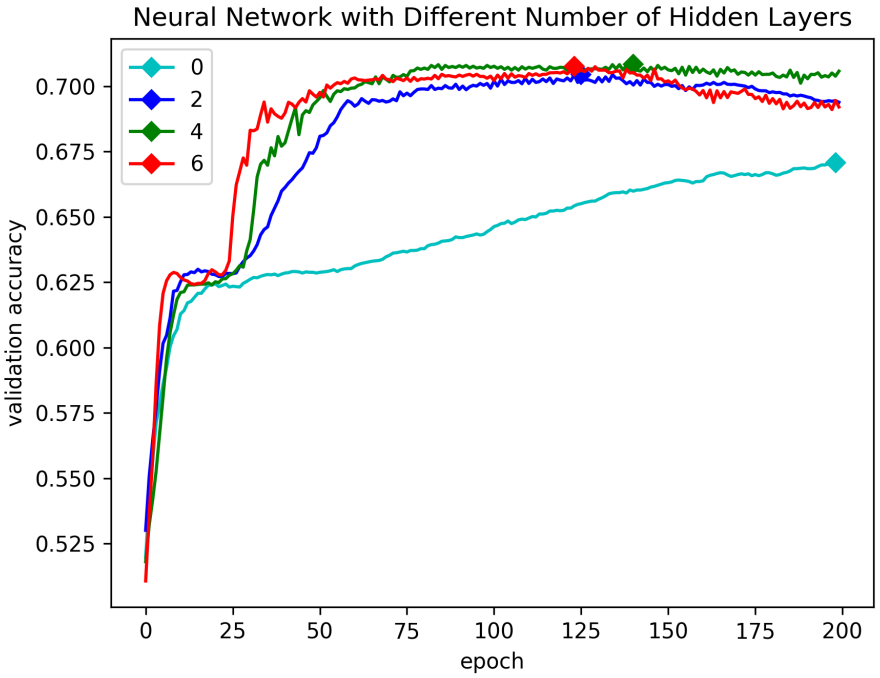


3.

Below is a table showing the validation accuracy and test accuracy for different models in part A and the modified model in part B. Among the baseline models (the ones in part A), the IRT model outperforms the other ones significantly. Our proposed neural network with four additional hidden layers performs slightly better than the IRT model. Comparing the two neural network models in part A and B, we can see a significant improvement in both validation and test accuracy.

Model	Validation Acc	Test Acc
KNN	0.6895	0.6842
IRT	0.7089	0.6971
Ensemble	0.6726	0.6664
Neural Network (Part A)	0.6854	0.6889
Neural Network (Part B)	0.7055	0.7042

Below we show a plot which comparing the validation accuracy of the neural network models with 0, 2, 4, and 6 hidden layers. (The code for each model can be found in part_a/neural_network.py, part_b/NN_2_layers.py, part_b/NN_modified_more-layers.py, part_b/NN_6_layers.py respectively, and the code for ploting is in part_b/NN_experiment.py) All of them use the same optimization hyperparameters in this experiment (lr = 0.001, epoch = 200, lamb = 0.0001). We propose that adding more layers can increase the flexibility of the model (improve the performance), but too many layers will lead to overfitting. In the plot below we can see that the model with 2 hidden layers outperforms that with no hidden layer, and the model with 4 hidden layers outperforms that with 2 hidden layers. However, when we add up to 6 hidden layers, the performance of the model is worse than that with 4 hidden layer, and the accuracy starts to decrease at about the 125th epoch, which indicates overfitting of the model. Besides, if we look at the maximum validation accuracy of the models (represented by the rectangles in the plot), the model with 4 hidden layers have the highest maximum validation accuracy. Thus, we choose to implement our neural network with 4 hidden layers to optimize the performance.



4.

(a) We would expect our approach to fail if the data is too sparse.
Since the autoencoder in our approach is a user-based autoencoder, where we try to learn a representation of the users, an extreme case of data being too sparse is that there is little data on what questions are answered correctly or incorrectly for many students (i.e., the entire row corresponding to that student would only contain nulls).

(b) Our guess for why our autoencoder performs poorly when the data is too sparse is that, if a student vector (1 row in the sparse matrix) has a large number of nulls, then the weights and biases of the model won't be changed much after training on that student.
Because when training using stochastic gradient descent with batch size = 1, the loss for each student will only be calculated based on the very few values in the input vector that are not null, and the partial derivative of the loss with respect to each output corresponding to a null input value will be 0 - so in the last layer, the gradient of the weights corresponding to this output will only be influenced by the regularization term, and the gradient of the bias corresponding to this output will be 0; and the magnitude of the gradient of the weights and biases of other layers will be smaller as well.
So if almost all of the values in the input vector are 0, then there will be very little update to the weights and biases after training on this student, in other words, the model will not be able to learn much from this student input vector.

(c) One possible extension to address this limitation caused by sparse data is to utilize side information (e.g., in this case, the student metadata). For this project, we were given metadata on gender, date of birth, and premium pupil for each student, although some of the values are null. So one way [1] to use this metadata is to treat the metadata for each student as a vector, so we will have a matrix of size $N \times 3$, where N is the number of students, and the columns are the student information given. Where there are null values, we can fill them with the average of the not-null values. After appropriate pre-processing of the metadata such as filling nulls, we can append the metadata vector for each student to the corresponding row in the sparse matrix, where the information on whether a student correctly answered a question is recorded.

We can then use this matrix (sparse matrix with student metadata appended) as our input matrix to the autoencoder, or even append this metadata to every layer inputs of the autoencoder [1]. This way, even if a student has no information on which question they answered correctly or incorrectly, the network will still have some information about the student themselves through the metadata.
We did try to improve our autoencoder using this approach (the code can be found in part_b/NN_modified_cold-start.py), but we got approximately the same validation and test accuracy as the base model. This could be due to the quality of the metadata, since about 20.8% of the gender data is null, 32.3% of the date of birth data is null, and 68.8% of the premium pupil data is null; it could also be due to whether *these* students answered *these* questions correctly or incorrectly are not that correlated with their gender, date of birth, or premium pupil status. Another reason could be that this approach still has limitations for capturing the students' ability to answer these questions correctly, as with other recent approaches that try to use side information in autoencoders [2].

References

[1] Florian Strub, Romaric Gaudel, Jérémie Mary. Hybrid Recommender System based on Autoencoders. The 1st Workshop on Deep Learning for Recommender Systems, Sep 2016, Boston, United States. pp.11 - 16, ff10.1145/2988450.2988456ff. ffhal-01336912v2

[2] Maksims Volkovs, Guangwei Yu, Tomi Poutanen. DropoutNet: Addressing Cold Start in Recommender Systems. 31st Conference on Neural Information Processing Systems (NIPS 2017), Long Beach, CA, USA. https://proceedings.neurips.cc/paper/2017/file/dbd22ba3bd0df8f385bdac3e9f8be207-Paper.pdf.