CSC311 Final Project Report

Sohee Goo, Rumaisa Chowdhury, Julia Bulat

August 16th, 2023

Part A

1. **K-Nearest Neighbours (KNN)**

   a)

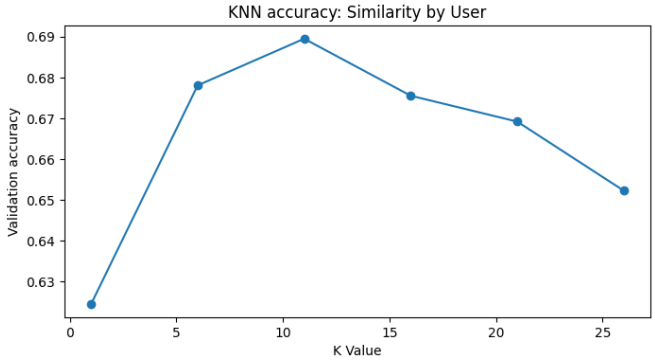


Figure 1: Knn Accuracy Impute by User

1. **b)** Optimal K Value: 11

   Highest Accuracy on Validation Set: 0.6895286480383855

   Final Test Accuracy: 0.6841659610499576

1. **c)**

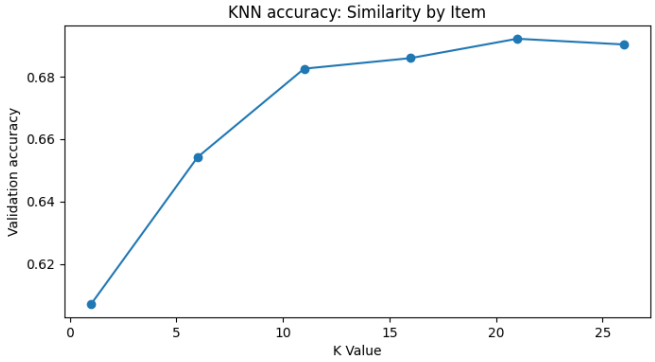


Figure 2: Knn Accuracy Impute by Item

   Optimal K Value: 21

   Highest Accuracy on Validation Set: 0.6922099915325995

   Final Test Accuracy: 0.6816257408975445

   With Item-based collaborative filtering, the underlying assumption is that if certain students answer Q1 the same way as Q2, then whether Q1 is answered correctly by specific students matches that of Q2.

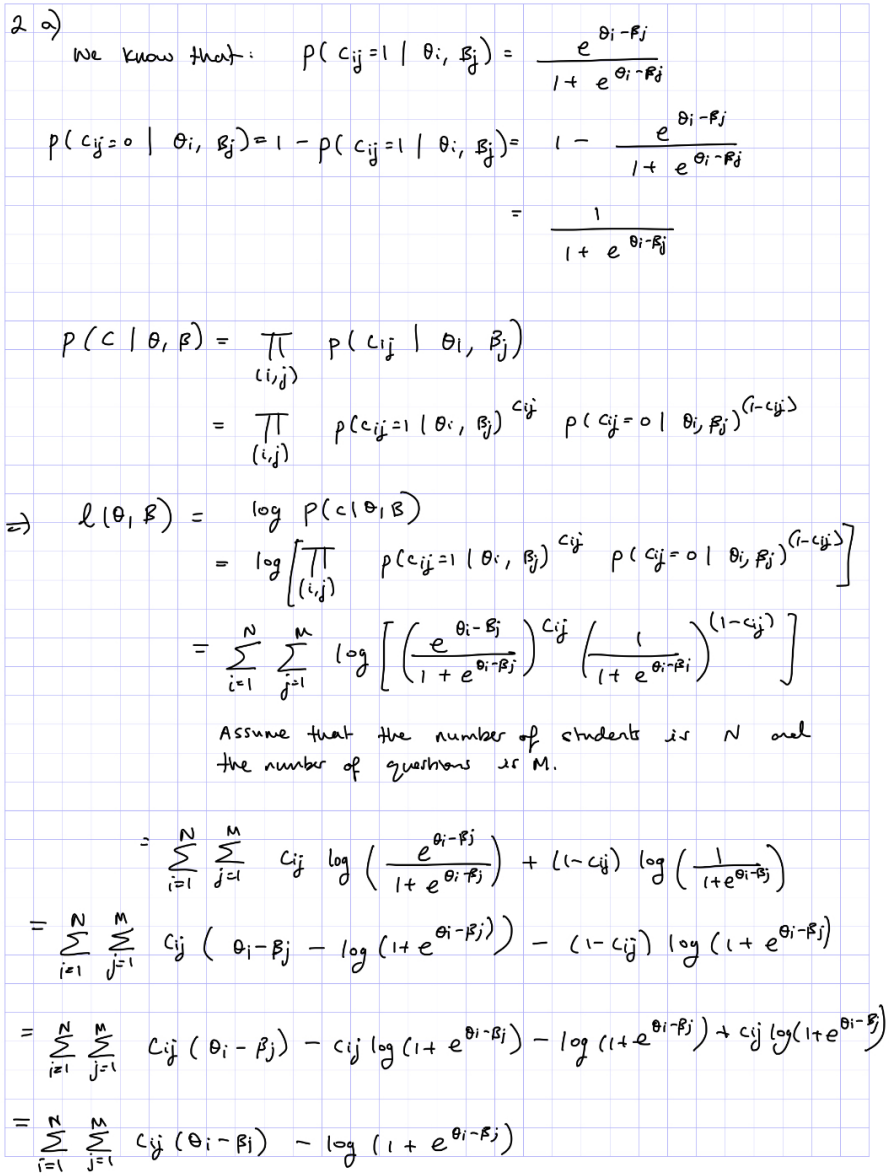
1. **d)** The test performance of the user-based collaborative filtering is slightly higher than of the item-based collaborative filtering, showing that the imputing by user method has a better performance.

1. **e)**

   - Curse of Dimensionality - As the dimension of our inputs is large, a limitation could be that the distances between most points are the same, impacting the overall performance of our model.

- High Cost - Since we are dealing with a non-parametric model, storing a large dataset can be very expensive. Additionally, due to this large dataset, it is computationally expensive at test time because we must go through the entire dataset each time.

2. **Item Response Theory**



2 a)

We know that: $p(c_{ij}=1 \mid \theta_i, \beta_j) = \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}$

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

$= \frac{1}{1+e^{\theta_i-\beta_j}}$

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

$= \prod_{(i,j)} p(c_{ij}=1 \mid \theta_i, \beta_j)^{c_{ij}} \; p(c_{ij}=0 \mid \theta_i, \beta_j)^{(1-c_{ij})}$

$\Rightarrow \ell(\theta, \beta) = \log p(C \mid \theta, \beta)$

$= \log\left[\prod_{(i,j)} p(c_{ij}=1 \mid \theta_i, \beta_j)^{c_{ij}} \; p(c_{ij}=0 \mid \theta_i, \beta_j)^{(1-c_{ij})}\right]$

$= \sum_{i=1}^{N}\sum_{j=1}^{M} \log\left[\left(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\right)^{c_{ij}} \left(\frac{1}{1+e^{\theta_i-\beta_j}}\right)^{(1-c_{ij})}\right]$

Assume that the number of students is N and the number of questions is M.

$= \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij} \log\left(\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}\right) + (1-c_{ij}) \log\left(\frac{1}{1+e^{\theta_i-\beta_j}}\right)$

$= \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}\left(\theta_i-\beta_j - \log(1+e^{\theta_i-\beta_j})\right) - (1-c_{ij}) \log(1+e^{\theta_i-\beta_j})$

$= \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}(\theta_i-\beta_j) - c_{ij}\log(1+e^{\theta_i-\beta_j}) - \log(1+e^{\theta_i-\beta_j}) + c_{ij}\log(1+e^{\theta_i-\beta_j})$

$= \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}(\theta_i-\beta_j) - \log(1+e^{\theta_i-\beta_j})$

$$\frac{d}{d\theta_i}\left[\log\left(1+e^{\theta_i-\beta_j}\right)\right] = \frac{1}{1+e^{\theta_i-\beta_j}} \cdot \frac{d}{d\theta_i}\left(1+e^{\theta_i-\beta_j}\right)$$

$$= \frac{1}{1+e^{\theta_i-\beta_j}} \cdot e^{\theta_i-\beta_j}$$

$$= \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}$$

$$\frac{d}{d\beta_j}\left[\log\left(1+e^{\theta_i-\beta_j}\right)\right] = \frac{1}{1+e^{\theta_i-\beta_j}} \cdot \frac{d}{d\beta_j}\left(1+e^{\theta_i-\beta_j}\right)$$

$$= \frac{1}{1+e^{\theta_i-\beta_j}} \cdot -e^{\theta_i-\beta_j}$$

$$= -\frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}$$

Therefore plugging in these equations below we get:

$$\frac{d\,l(\theta,\beta)}{d\,\theta_i} = \frac{d}{d\theta_i}\left[\sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}(\theta_i-\beta_j) - \log\left(1+e^{\theta_i-\beta_j}\right)\right]$$

$$= \frac{d}{d\theta_i}\left[\sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}\theta_i - c_{ij}\beta_j - \log\left(1+e^{\theta_i-\beta_j}\right)\right]$$

$$= \sum_{j=1}^{M} c_{ij} - \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}$$

$$\frac{d\,l(\theta,\beta)}{d\,\beta_j} = \frac{d}{d\beta_j}\left[\sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}(\theta_i-\beta_j) - \log\left(1+e^{\theta_i-\beta_j}\right)\right]$$

$$= \frac{d}{d\beta_j}\left[\sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}\theta_i - c_{ij}\beta_j - \log\left(1+e^{\theta_i-\beta_j}\right)\right]$$

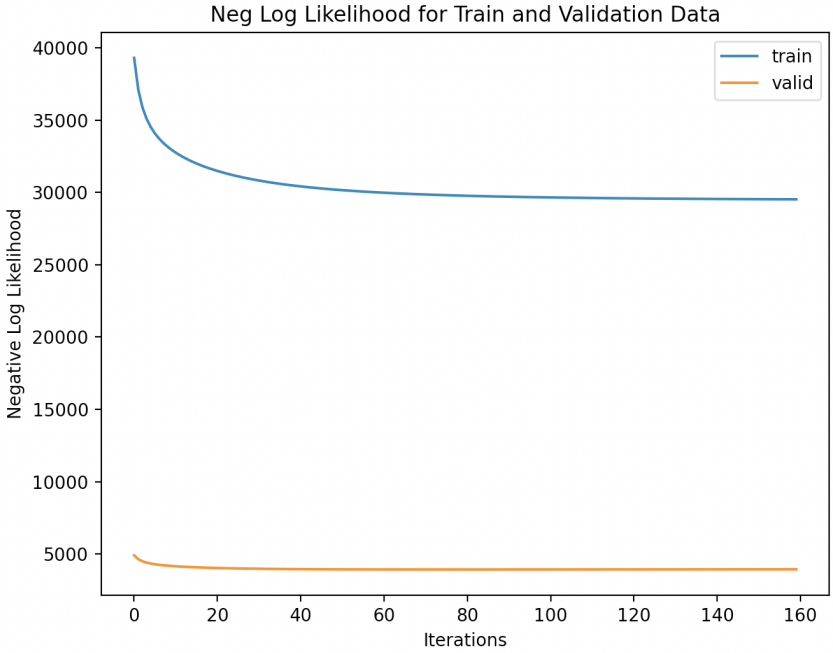
$$= \sum_{i=1}^{N} -c_{ij} + \frac{e^{\theta_i-\beta_j}}{1+e^{\theta_i-\beta_j}}$$

2. **a)**

   For this question, we will assume that given a student i, the joint probability of the question correctness is independent of all other students, conditioning on each θ_i and the question difficulties. Additionally, for a fixed student i, the probability of the i-th student getting the j-th question correct is independent of all other questions, conditioning on β_j.

2. **b)** The hyperparameters I chose are:

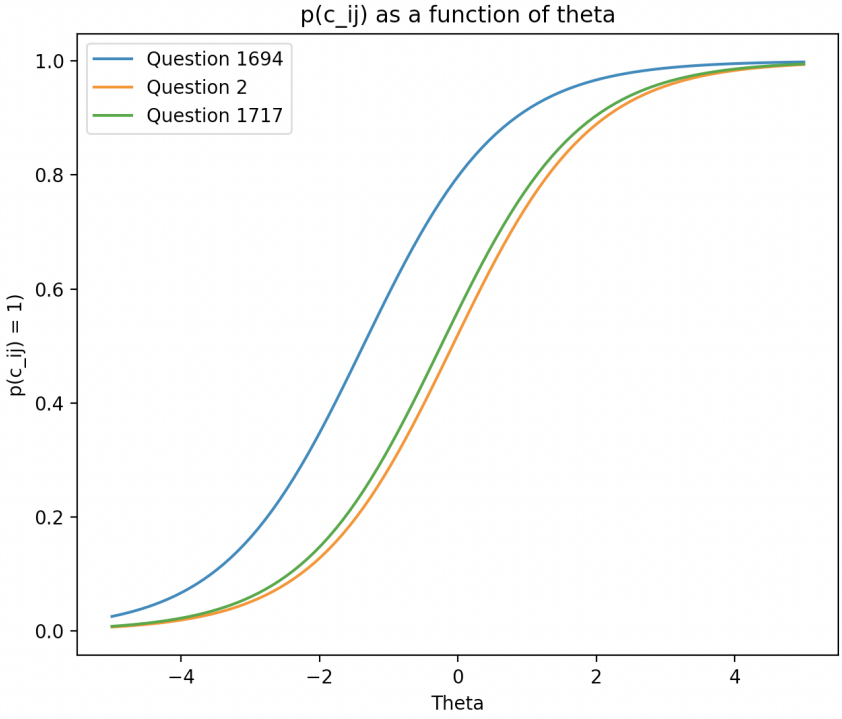
   - Number of Iterations: 160
   - Learning Rate: 0.004
   - Initialize theta values to all zeros
   - Initialize beta values to all zeros



2. **c)** The final validation and test set accuracies are:

```
Validation Accuracy:  0.7064634490544736
Test Accuracy:  0.7070279424216765
```

2. **d)** I selected three questions at random and plotted p(c_ij = 1) as a function of theta.



These curves represent how likely the selected question will be answered correctly as the student's ability changes.The x-axis represents the student's ability and the y-axis shows the probability p(c_ij). We see that the shape of these three curves looks similar to the sigmoid function because p(c_ij) = sigmoid(theta_i - beta_j) and so it is a transformed sigmoid function. Since theta represents the student's ability, we see that for all of the curves, as the student's ability (theta) increases then the probability of getting the question gets closer to 1. When the student's ability decreases towards negative infinity, we see the probability of getting the question right decreases and converges towards zero. This is natural since the student's ability should be correlated with getting the question correct. Also, for a fixed value of theta on the x-axis, we can see that the curve with the higher value on the y-axis has a higher probability that the student can answer it correctly. Each question has a beta value which represents the difficulty of the question. Therefore, the curves which are higher have a smaller beta value, since there's a higher probability of getting it right so it is easier, and the curves which are lower have a larger beta value.

3. **Neural Networks** We decided to do Option 2, Neural Networks.

3. **a)** The 3 key differences between ALS and Neural Networks are:

1. ALS relies on linear models which usually use two matrices multiplied together to predict missing values of another matrix. Neural networks use both linear and non-linear transformations through their layered architecture and activation functions. This allows neural networks to better capture complex patterns which can't be effectively represented by linear models alone.

2. They use different optimization techniques. For ALS, matrices U and Z are optimized iteratively solving linear equations. Each iteration solves for the optimal values of U or Z directly. Neural networks use gradient descent to adjust the weights since they are usually nonlinear models.

3. ALS is more interpretable in comparison to neural networks. For instance, in ALS with k latent dimensions, each dimension may be a distinct latent feature. For example, the two matrices U and Z provide a clear representation of Users and Questions in the latent dimension.

   In contrast, neural networks, such as autoencoders often resemble black boxes. Despite obtaining trained model components, like weights and biases, understanding their meaning, especially in the first latent dimension, remains challenging due to complex transformations.

3. **b)** Solution in `neural_networks.py`

3. **c)** After testing different hyperparameters we decided:

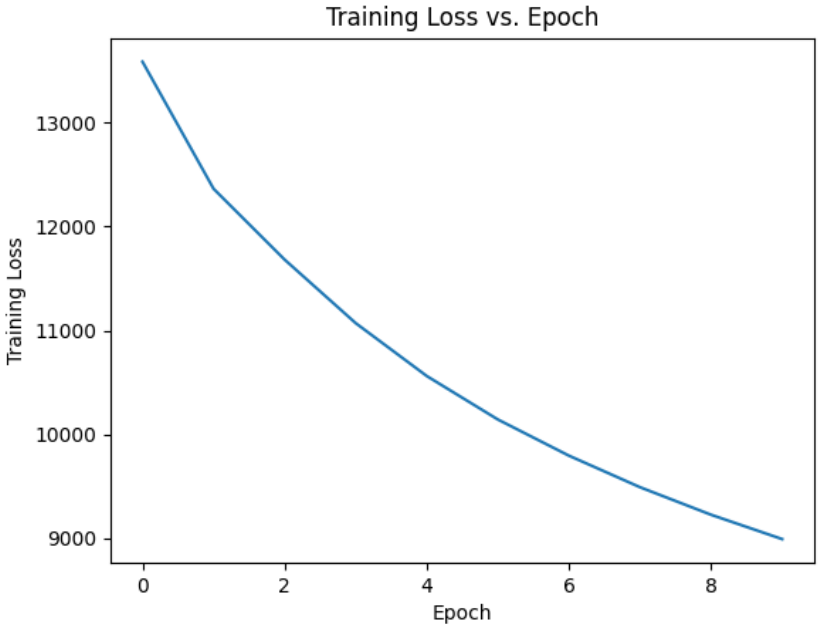
   - $k^* = 10$
   - Learning Rate: 0.1
   - Number of Epochs = 10

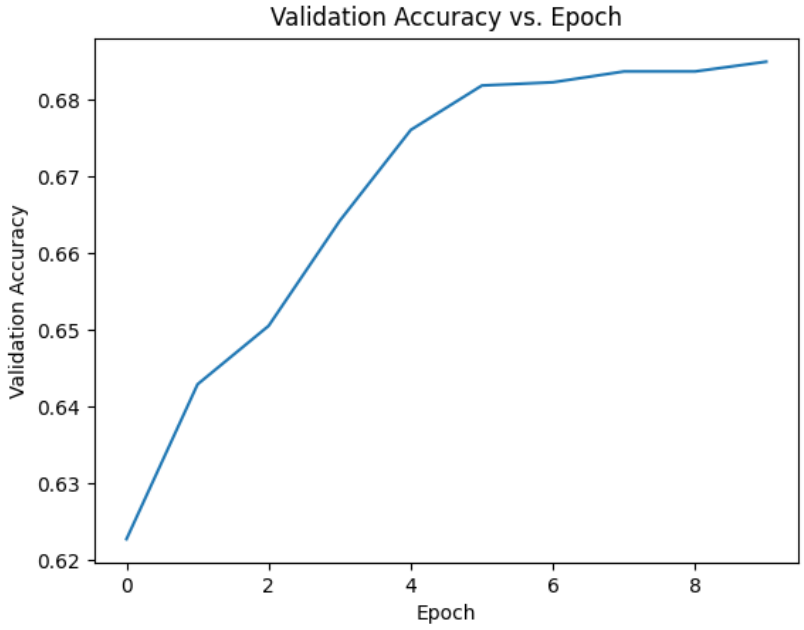
   Validation Accuracy: 0.68501

3. **d)**

   Test Accuracy: 0.67739

   Figures:





3. **e)**

Lambda	**Validation Accuracy**	**Test Accuracy**
0.001	0.6860005644933672	0.6864239345187694
0.01	0.6644086931978549	0.6663844199830652
0.1	0.6237651707592435	0.6260231442280553
1	0.6237651707592435	0.6254586508608524

Table 1: Validation and Test Accuracy for Different Lambda Values

The best final validation and test accuracy happen with $\lambda = 0.001$. The regularized validation accuracy is 0.00099 greater. The regularized test accuracy is 0.00903 greater. The regularizer doesn't improve our model

that much and only slightly increased our accuracies. We think this is because our model is not over-fitting so adding a regularizer doesn't do much.

4. **Ensemble**

   For this question, we decided to ensemble all three models from the previous questions: Knn, Item-response Theory, and Neural Nets.

   Final Validation Accuracy: 0.6880
   Final Test Accuracy: 0.6878

   Process:

   1. Bootstrap the original training set 3 times (sample (student, question) pairs randomly with replacement to create new datasets with the same size as the original) - once for each chosen model.
   2. Separately train each model (KNN, IRT, Auto Encoder) on it's own bootstrapped dataset using optimal hyperparameters and make unique predictions.
   3. Combine predictions and find the average over all three models to form the final prediction.
   4. Evaluate the accuracy of the combined predictions and compare with models' individual performances.

   Hyperparameters:

   1. KNN
      k = 11
   2. IRT
      learning rate = 0.004
      iterations = 160
   3. Neural Nets
      k = 10
      learning rate = 0.1
      lambda = 0.001
      epoch = 10

   Comparisons:

Knn	IRT	NN	Ensemble
0.6895	0.7065	0.6860	0.6880

Table 2: Validation Accuracies Across Models

Knn	IRT	NN	Ensemble
0.6842	0.7070	0.6864	0.6878

Table 3: Test Accuracies Across Models

Upon comparing the validation accuracies across all models, we can see that the ensembled model had a lower validation accuracy than both the KNN model and the IRT model, but a higher validation accuracy than the NN model.

Looking at the test accuracies, we can see that the ensemble model outperforms both the KNN and NN models, but still has a lower test accuracy than the IRT model.

One possible reason for the IRT model performing better than the ensemble model could be due to the nature of the bootstrapping method. Since the method involves resampling the data, it can introduce more noise into the training sets. Additionally, perhaps since all of the models use the same datasets (questions, students, correctness) and might capture the same features, the models produce similar predictions and averaging them out gives us similar final predictions.

Overall, the ensemble model did not report higher accuracy than some previously tested models.

Part B

1. **Formal Description**

From Part A, we noticed that the auto encoder was under fitting because the validation and test accuracy were not very high and similar to each other (0.688258537962179 and 0.6796500141123342 respectively). The neural network only had one hidden layer and limited input data which made us believe that the model was too simple to capture the underlying patterns in the data. Therefore, we tried different methods to increase the model's complexity. First we started off by changing the neural network to be based on questions instead of students and then modified our model in various steps.

1.1 Student Based Model

Unlike the original auto encoder which used student vectors as the input vectors, we changed the model to use question vectors. Therefore, the input vectors would contain all the students' answers to one question. This way, the model can use the data of how all the other students correctly or incorrectly answered one question to predict how well another student would answer it. This seemed like a more natural way of presenting the data because we can see the general performance of students across one question. The previous design seemed less intuitive as a student's performance on one question might not determine their ability to perform well on another question in a completely different subject. We also noticed that the metadata for questions is more complete than for users so modifying the question based model will be easier. Additionally, there are less students than questions so now our input vectors will have smaller dimension which takes less computational power from our model. For these reasons, we believe that using this model will help us find solutions to improve underfitting and use better optimization techniques. Therefore, our baseline model will now be a question based auto encoder which uses the same 3 layer architecture as given in Part A. We will be doing all of our modifications to this new model.

Formally, given a question $v \in \mathbb{R}^{N_{students}}$ from a set of questions Q, one pass through the neural network would give us:

$$f(v;\theta) = h(W^{(2)}g(W^{(1)}v + b^{(1)}) + b^{(2)}) \in \mathbb{R}^{N_{students}}$$

Where $f(v;\theta)$ is the reconstruction of the input v for some activation functions h and g. We will be using the sigmoid function in our baseline model. $W^{(1)} \in \mathbb{R}^{k \times N_{students}}$ and $W^{(2)} \in \mathbb{R}^{N_{students} \times k}$ where $k \in \mathbb{N}$ is the latent dimension. Just like Part A, the objective with the regularizer will be:

$$min_\theta \sum_{\mathbf{v} \in \mathbb{Q}} \|\mathbf{v} - f(\mathbf{v};\theta)\|_2^2 + \lambda/2 \left(\left\| \mathbf{W}^{(1)} \right\|_F^2 + \left\| \mathbf{W}^{(2)} \right\|_F^2 \right)$$

1.2 Mini-Batch Gradient Descent

Our baseline neural network uses stochastic gradient descent during optimization. This means that it updates the parameter values for each input vector. This might produce noisy steps as it is biased towards that specific input. To produce more stable steps and also use parallel calculations for a faster training time, we decided to use mini batch gradient descent. In mini batch gradient descent, since we sum up the gradients across more training examples before updating the parameters, each step is more representative of the data and will be less noisy. Therefore, we expect that changing this optimization technique will improve our accuracy and we will find better parameter values.

1.3 Maximum Likelihood Estimations

In Part A and our baseline model, we replace all the missing entry (NaN) values in our training data matrix with 0s. This isn't logical because 0s correspond to answering the questions incorrectly but the student not having answered that question doesn't mean they got it wrong. They could've possibly gotten it correct had they done that question. Therefore, the data is misleading because the neural network doesn't know the difference. It views missing entries and incorrect entries as the same during the encoding process. Therefore, we would like to fill the missing entries with a better value. For each question, we decided to use the maximum likelihood estimate of the Bernoulli distribution for the correct/incorrect random variable. Equivalently, it is the empirical mean of the number of students who got the correct answer for a question. So for each question, we calculate the mean value and replace the missing entries with it. This is assuming that students who didn't answer the question can answer at the average level of all the other students who answered the question. It calculates how much we think the student could answer the question correctly.

We expect that this modification will improve the model because it seems more probable that a student will have the ability of an average student who answered that question. It is much less misleading than simply assuming that they got the answer wrong. By improving the training data and making it less biased, we expect that the model will learn better parameter values because it can distinguish missing entries from incorrect entries.

1.4 Activation Functions

In Part A and our baseline model, we use the sigmoid activation function during the encoding and decoding process. In this section, we wanted to experiment with different activation functions and see how they affected the training and validation performance. The sigmoid activation function can have problems and might be causing our network to underfit. For example, the sigmoid function can squish the inputs to be very close to 0 or 1 which causes the gradients of the function to become too small so the weights are not updated as effectively. Other activation functions might provide benefits. For example, ReLU activation function can introduce more sparsity because the outputs of the activations can be zero which allows the network to focus more on relevant features. Therefore, we expect that changing the activation function might increase the model's ability to learn the weights better.

1.5 IRT Question Difficulty

From the IRT Model in part A, we derived the trained vector $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_M)$, where M is the total number of questions and β_j is the estimated difficulty for question j. Injecting these question difficulty parameters might enable the neural network to predict how well a student will answer by factoring in how hard the question is. We append the beta parameter to the compressed latent vector, treating it as an extra dimension of the question latent space. This is because the beta parameters are already learned features from IRT so we think it makes sense to concatenate it to the learned latent features of the neural network. Therefore, the model will learn to decode this concatenated latent vector. We expect that by giving the neural network more information about questions, it will be able to make better predictions.

Formally, given a question $v \in \mathbb{R}^{N_{students}}$ from a set of questions Q and $\beta \in \mathbb{R}^1$ is the beta value for the corresponding question, one pass through the neural network would give us:

$$f(v, \beta; \theta) = h(W^{(2)}(g(W^{(1)}v + b^{(1)}) + \beta) + b^{(2)}) \in \mathbb{R}^{N_{students}}$$

where $W^{(1)} \in \mathbb{R}^{k \times N_{students}}$ and $W^{(2)} \in \mathbb{R}^{N_{students} \times (k+1)}$ where $k \in \mathbb{N}$ is the latent dimensions and g,h are sigmoid functions.

1.6 Question Subject Metadata

Since we made the decision to change our model to use question vectors as input vectors, we wanted to experiment with the effects of implementing the provided question metadata on the model's overall performance. The question metadata consists of the question and subject ids, the use of which we believe will make it easier for the neural network to recognize the patterns in the data, as questions belonging to the same subjects could be treated in a similar fashion. To do this, we passed in the metadata into the neural network to get a latent representation of it. Then, we concatenated this latent representation of the subjects with our encoded input vector. The model will then decode this concatenated vector. We decided to encode the subject data so that the model could extract the most important patterns from it. Additionally, it reduced the size of the metadata input which is a vector with length of the number of subjects with 0 if a question is not in that subject and 1 if a question is in that subject. For example, a question with subject ids 1, 2 would look like [1,1,0,0,...,0]. We believe this approach will allow the model find complex patterns in the question subjects and make more accurate predictions.

Formally, given a question $v \in \mathbb{R}^{N_{students}}$ from a set of questions Q and $m \in \mathbb{R}^{N_{subjects}}$ is the metadata vector, one pass through the neural network would give us:

$$f(v, m; \theta) = h(W^{(2)}(g(W^{(1)}v + b^{(1)}) + g(W^{(3)}m + b^{(3)})) + b^{(2)}) \in \mathbb{R}^{N_{students}}$$

where $W^{(1)} \in \mathbb{R}^{k \times N_{students}}$, $W^{(3)} \in \mathbb{R}^{j \times N_{subjects}}$ and $W^{(2)} \in \mathbb{R}^{N_{students} \times (k+j)}$ where $k, j \in \mathbb{N}$ are the latent dimensions and g,h are sigmoid functions.

1.7 More Layers

Upon our investigation of the original model, we noticed that it was too simple, exhibiting behaviours that correlated with underfitting. To fix this issue, we decided to increase the depth of the model by adding 2 more

hidden layers. Therefore, we will now by using a 5-layer architecture instead of a 3-layer architecture. We expect that this approach will increase the model's complexity and thus help the model learn more complex patterns in the data.

To add hidden layers, we simply modified the Autoencoder class and 3 of its functions.

Each layer of the more complex model will use the following functions:

$$\mathbf{h}^{(1)} = f^{(1)}(\mathbf{v}) = \sigma(\mathbf{W}^{(1)}\mathbf{v} + \mathbf{b}^{(1)})$$

$$\mathbf{h}^{(2)} = f^{(2)}(\mathbf{h}^{(1)}) = \sigma(\mathbf{W}^{(2)}\mathbf{h}^{(1)} + \mathbf{b}^{(2)})$$

$$\mathbf{h}^{(3)} = f^{(3)}(\mathbf{h}^{(2)}) = \sigma(\mathbf{W}^{(3)}\mathbf{h}^{(2)} + \mathbf{b}^{(3)})$$

$$f(v;\theta) = f^{(4)}(\mathbf{h}^{(3)}) = \sigma(\mathbf{W}^{(4)}\mathbf{h}^{(3)} + \mathbf{b}^{(4)})$$

The Updated Neural Network function has several new variables. $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(3)}$ represent the newly added weight matrices, as well as their respective biases: $\mathbf{b}^{(2)}$ and $\mathbf{b}^{(3)}$. The final NN model can be rewritten as:

$$f(v;\theta) = m(\mathbf{W}^{(4)}h(\mathbf{W}^{(3)}f(\mathbf{W}^{(2)}g(\mathbf{W}^{(1)}\mathbf{v} + \mathbf{b}^{(1)}) + \mathbf{b}^{(2)}) + \mathbf{b}^{(3)}) + \mathbf{b}^{(4)})$$

where

- g, f, h, m are the sigmoid activation function.
- $v, f(v;\theta) \in \mathbb{R}^{N_{students}}$
- $W^{(1)} \in \mathbb{R}^{k \times N_{students}}$ where $k \in \mathbb{N}$ is the latent dimension of the first hidden layer
- $W^{(2)} \in \mathbb{R}^{j \times k}$ where $j \in \mathbb{N}$ is the latent dimension of the second hidden layer
- $W^{(3)} \in \mathbb{R}^{k \times j}$ where $k \in \mathbb{N}$ is the latent dimension of the third hidden layer again
- $W^{(4)} \in \mathbb{R}^{N_{students} \times k}$
- k and j are hyperparameters

1.8 Final Model

For our final model, we will combine all of our modifications from each of the sections above into one neural network. We expect this model to have the highest accuracy overall because it combines all of our ideas so it is the most complex model, which will help with underfitting.

2. Figures and Diagrams

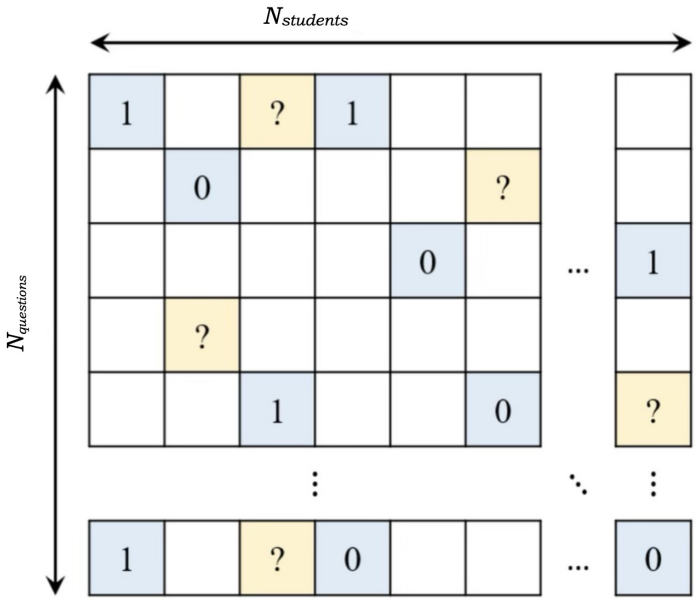


Figure 3: Transposed Sparse Matrix

- We used a transposed version of the original sparse matrix for the question based model, inputing question vectors when training the model.

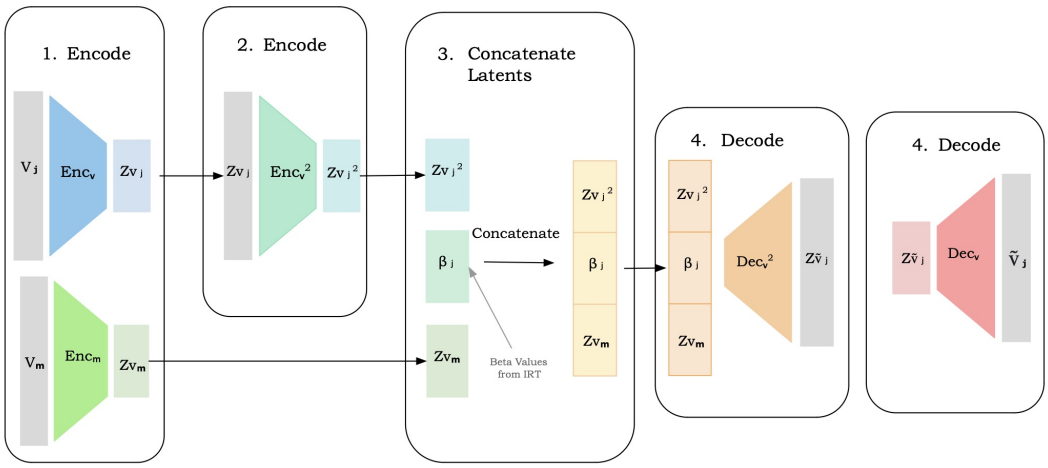


Figure 4: Autoencoder Architecture for Final Model

- We first encode the input question vector and meta data vector.
- Following the initial encoding, we encode the question vector one more.
- After encoding, we concatenate the resulting latents with the β parameter.
- Finally we decode the the concatenated latent twice more to reconstruct the original input vector.

3. Comparison or Demonstration

3.1 Comparison of Different Models

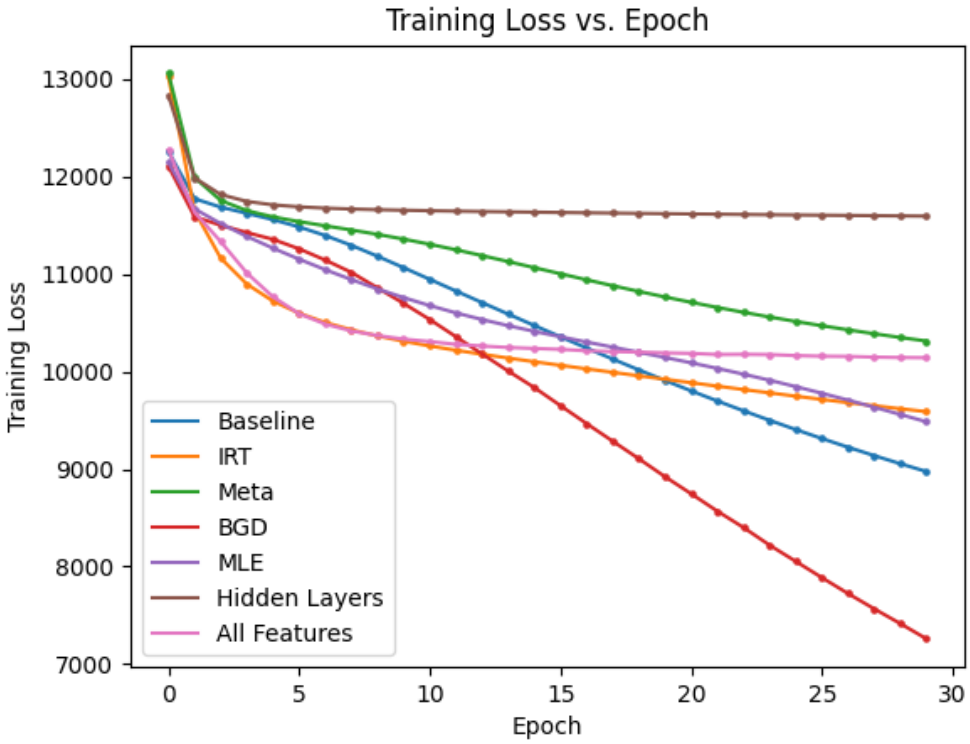


Figure 5: Training loss vs. Epoch for Question-based Model

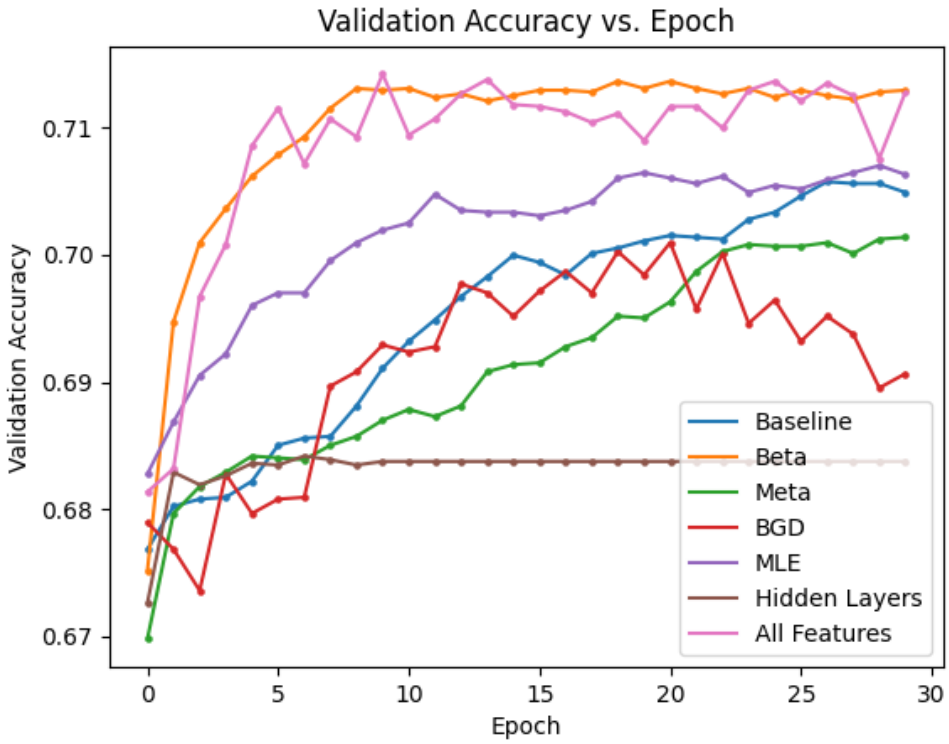


Figure 6: Accuracy vs. Epoch for Question-based Model

Model	Hyperparameters	Validation Accuracy	Test Accuracy
Student Based Neural Network	k: 10, Learning Rate: 0.1, Epoch: 10, Lambda: 0.001	0.688258537962179	0.6796500141123342
Question Based Neural Network (Baseline NN)	k: 100, Learning Rate: 0.01, Epoch: 10, Lambda: 0.001	0.6929156082416031	0.6841659610499576
Baseline NN + Mini-Batch Gradient Descent	k: 100, Learning Rate: 0.01, Epoch: 10, Lambda: 0.001 Batch size: 15	0.6941857183178097	0.6833192209991532
Baseline NN + MLE	k: 100, Learning Rate: 0.01, Epoch: 10, Lambda: 0.001	0.7012418854078465	0.6965848151284222
Baseline NN + Activation Functions	k: 100, Learning Rate: 0.01, Epoch: 10 Activation: sigmoid	0.6907987581145921	0.6827547276319503
Baseline NN + IRT	k: 10, Learning Rate: 0.01, Epoch: 10, Lambda: 0.001	0.7154953429297206	0.7081569291560824
Baseline NN + Question MetaData	k: 500, Learning Rate: 0.01, Epoch: 10, Lambda: 0.001 meta latent dim: 5	0.6885407846457804	0.6776742873271239
Baseline NN + More Layers	k: 50, Learning Rate: 0.005, Epoch: 10, Lambda: 0.001 j: 10	0.6845893310753599	0.6720293536550945
Final NN	k: 100, Learning Rate: 0.01 Epoch: 10, Lambda: 0.001 Batch size: 26, j: 5 meta latent dim: 5	0.7116881625740897	0.7096522156364663

3.2 Model Information

Model	Section of Document	Code File
Student Based Neural Network	Part A Question 3	neural_network.py
Question Based Neural Network (Baseline)	Section 1.1	nn_question_baseline.py
Baseline NN + Mini-Batch Gradient Descent	Section 1.2	nn_question_bgd.py
Baseline NN + MLE	Section 1.3	nn_question_mle.py
Baseline NN + Activation Functions	Section 1.4	nn_question_activation.py
Baseline NN + IRT	Section 1.5	nn_question_beta.py
Baseline NN + Question MetaData	Section 1.6	nn_question_meta.py
Baseline NN + More Layers	Section 1.7	nn_question_more_hidden.py
Final NN	Section 1.8	nn_question_final.py

* Final NN is the Question Based Neural Network with MetaData, IRT, Mini-Batch Gradient Descent, MLE, and more Layers

3.2 Model Analysis and Hypothesis Verification

For each of the models stated above in our table, we conducted experiments by designing a new model (located in the corresponding python file) and testing their performances. Since we tested our models extensions in a modular fashion, this allows us to easily see how much each modification improved our baseline question based model.

- In Section 1.1, we expected that our question based model would have better accuracy than the student based auto encoder because of a more intuitive input vector. From our experiment, we see that our hypothesis is true because our validation and test accuracies are slightly higher. Therefore, this justifies using the question based model as our new baseline model for the rest of the modifications.
- From Section 1.2, we expected that changing from stochastic to mini batch gradient descent would make our steps less noisy leading to better parameter estimations. We see from our experiment that the

accuracy from using minibatch gradient is very similar to our baseline model. Therefore, from the point of optimization, making this change doesn't affect our model's ability.

- From Section 1.3, our hypothesis was that the MLE would be a better value to replace the Nan values in the data matrix. Given the improved accuracies, we learn that doing this data pre-processing step helps our model's underfitting problem a little bit.
- From Section 1.4, we wanted to test out different activation functions to see if the sigmoid function was limiting our model's ability to learn weights. However, after running this test we see from the accuracies that the best activation function is still the sigmoid function. The accuracies are also similar to our baseline model. So our hypothesis was proven false and changing the activation function will not help our model. Therefore, we will still be using the sigmoid activation function in the final model.
- From Section 1.5, we expected that injecting the beta values from IRT would give our model more information about questions to make better predictions. Since the validation and test accuracies reached over 70 percent, we see that this modification improves our model the most out of all the modifications.
- From Section 1.6, we expected that the injection of question meta data into our input vectors would allow our model to better understand the underlying subject relationships between different questions. However, upon testing the theory, our hypothesis was proven false. We saw that both the validation and test accuracy slightly decreased from our baseline question neural network, resembling the accuracies of the student based neural network.
- From Section 1.7, our hypothesis was that adding more hidden layers would help our model generalize better to unseen data, capturing complex patterns and discarding noise. After running our model with these additions, we found that the accuracy slightly decreased from our base model so our hypothesis is false.
- Finally, after putting all the modifications together into our final model, we get the highest test accuracy out of all the models. We see that combining all of our hypothesis together helps the model a little bit at overcoming underfitting.

4. **Limitations**

   There are several limitations that we discovered when we thought about the ways our approaches might fail or under perform:

   1. **Mini Batch Hyperparameters:** While our decision to use Mini-batch gradient descent has its benefits, it also introduces additional hyperparameters (ex. batch size), which requires careful tuning just like all our previous hyperparameters. A poor choice of batch size could lead to slower convergence and negatively affect the optimization process. We see in our model that it lowered our accuracies so perhaps our batch-size hyperparameters weren't good enough. Finding these hyperparameters also takes more computational power. The Mini-batch approach can also make the model more sensitive to the choice of learning rate.
   2. **Underfitting Turned Overfitting:** As mentioned before, increasing the depth of our model with the addition of two new hidden layers could lead to overfitting, even though the issue we were initially trying to solve is underfitting. The size of our dataset also comes into the equation since a dataset that is too small might not be able to support the increased complexity of extra hidden layers. Deeper networks are harder to train and require careful tuning. Additionally, since the dataset we are working with is simple, perhaps a deeper network is losing important low-level details and thus making our results worse.
   3. **Inaccurate Missing Values:** While we adopted the MLE approach with the expectation that it would yield more accurate predictions compared to our original approach, it's important to note that the MLE approach gives all missing entries for a question the same value. However, this assumption may not hold true for all scenarios or questions. For example, if only older students answered a difficult question and we took the mean value of their correct responses, we would fill the missing values with this mean. However, this is not a good prediction for how well another student of a younger age might answer this question. In this case, our model would perform poorly.
   4. **Subject Difficulties & Metadata** One limitation of applying the question metadata into our model is that we aren't taking into account the difficulties of the subjects. Due to this, the effect of the addition of metadata into our input vectors was negative as our model stayed around the same accuracy. The subject data doesn't tell us enough relevant information to help predictions. Perhaps we could have calculated the subject difficulties by seeing the ratio of how many students get questions of a certain subject correct and incorporating this into our model.

Some possible extensions are:

1. **Improving IRT model:** We saw that injecting the beta values from our IRT model gave us significant improvements to the accuracies in our hypothesis testing from section 3. A setting in which our model would perform poorly is if the IRT was trained poorly, these beta values were not useful, and noise was added to our autoencoder's latent vectors. Our neural network model's performance is dependent on how well the IRT model learned these beta values. A possible extension could be to improve the performance of the IRT model by increasing its performance with more relevant parameters. If we pretrain our IRT model to be better, it could give our neural network even better performance. Additionally, we could also try to incorporate the theta values from IRT which encode student's abilities.
2. **External Data:** One of the ways we extended the original model was by concatenating the input vectors with the question metadata. However both the question and student metadata are very limited and don't take into account the many factors that affect a student's question answers. By incorporating external data sources, such as socioeconomic indicators, student demographics, or curriculum information, it is possible to enrich the model's understanding of student performance factors and find more intricate patterns in our dataset.
3. **Missing Values:** The problem of what to fill in our missing values of the data matrix is an open problem. By filling in the entries with MLE values we only took into account a specific question. However, a possible extension could be to take into account specific user's information as well so each missing entry gets a customized value. Maybe we could calculate how many questions of that particular subject the user has gotten correct. We could combine a quantity representing a user's mastery of the subject as well as the average performance of that question across all students. Additionally, we could take into account the student's age as older students might have a better aptitude at answering harder questions.

5. **Contributions**

Part A

- Julia Bulat: KNN, Ensemble
- Sohee Goo: IRT
- Rumaisa Chowdhury: Neural Networks

Part B

- Julia Bulat: Metadata, More layers, Limitations section
- Sohee Goo: MiniBatch gradient descent, MLE, Activation function, Comparison section
- Rumaisa Chowdhury: Question Based Model, IRT injection, Figures section
- All members contributed to making the final model and looking over each other's parts.