CSC311H1 F - Project Option 1

Ricky Pramanick - 1009320940

Gursimar Singh - 1009012685

Gerald Wang - 1008787252

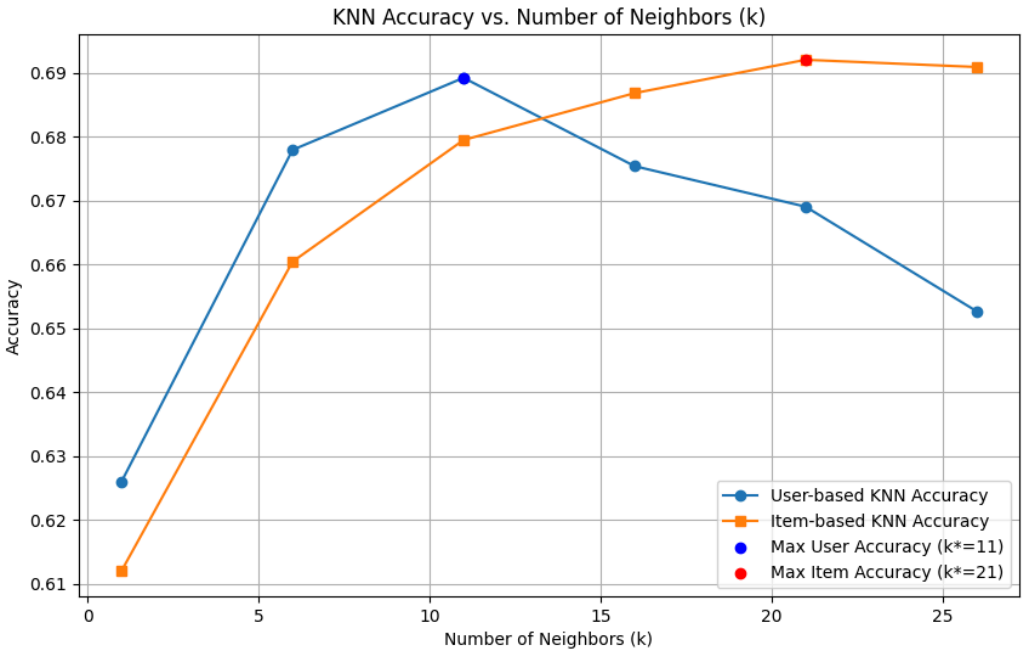
Contributions

Member of Team	**Contributions**
Ricky	Part A - Q3, Part B - Q1, Q3
Gursimar	Part A - Q2, Part B - Coding, Q2
Gerald	Part A - Q1, Q4, Part B - Q4

Part A

Q1)

a)




```
Validation Accuracy (user), k=1: 0.6260231442280553
Validation Accuracy (user), k=6: 0.6779565340107254
Validation Accuracy (user), k=11: 0.6892464013547841
Validation Accuracy (user), k=16: 0.6754163138583121
Validation Accuracy (user), k=21: 0.6690657634772792
Validation Accuracy (user), k=26: 0.652695455828394
```

```
Validation Accuracy (item), k=1: 0.6120519333897827
Validation Accuracy (item), k=6: 0.6604572396274344
Validation Accuracy (item), k=11: 0.6795088907705334
Validation Accuracy (item), k=16: 0.6868473045441716
Validation Accuracy (item), k=21: 0.6920688681907987
Validation Accuracy (item), k=26: 0.6909398814563928
```

b)

```
User KNN test accuracy for k*=11: 0.68219023426474 73
Item KNN test accuracy for k*=21: 0.6790855207451313
```

c) Implementation and accuracies for item-based included in answers to a) and b). The assumption that item-based collaborative filtering makes is that items responded to in similar ways by users are similar, and a user's interaction with one item can predict their interaction with a similar item. Additionally, it assumes that if a user performs well on one question, they

will perform similarly on a similar question, meaning that their answers are stable across questions.

d) User-based collaborative filtering has higher test accuracy.

e) Limitations:

   i) As the number of questions and users increases, scalability of the KNN algorithm becomes an issue as it becomes increasingly computationally expensive.

   ii) With more questions and users, it's not guaranteed that users will come close to answering all of the questions. Instead, they are likely to only answer a small subset of questions. This results in data sparsity, which could lead to unreliable predictions.

Q2)

a)

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

We provide the starter code in `item_response.py`.

(a) Derive the log-likelihood $\log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})$ for all students and questions. Here $\mathbf{C}$ is the sparse matrix. Also, show the derivative of the log-likelihood with respect to θ_i and β_j (Hint: recall the derivative of the logistic model with respect to the parameters).

Assume we have N students and K questions.

$p(c_{ij} | \theta_i, \beta_j)$ can be concisely written as

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

So over the entire dataset, we get.

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

Therefore, our log-likelihood is;

$$\ell(\theta, \beta) = \log p(C | \theta, \beta) = \sum_{i=1}^{N} \sum_{j=1}^{K} \log p(c_{ij} | \theta_i, \beta_j)$$

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

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

$$= \sum_{i=1}^{N} \sum_{j=1}^{K} c_{ij} \theta_i - c_{ij} \beta_j - \cancelto{0}{c_{ij} \log(1 + \exp(\theta_i - \beta_j))} - \log(1 + \exp(\theta_i - \beta_j)) + \cancelto{0}{c_{ij} \log(1 + \exp(\theta_i - \beta_j))}$$

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

For the derivatives, we have

$$\frac{\partial \ell}{\partial \theta_i} = \frac{\partial}{\partial \theta_i} \left[\sum_{i'=1}^{N} \sum_{j=1}^{K} c_{i'j} \theta_{i'} - c_{i'j} \beta_j - \log(1 + \exp(\theta_{i'} - \beta_j)) \right] \quad [i \in \{1, \ldots, N\}]$$

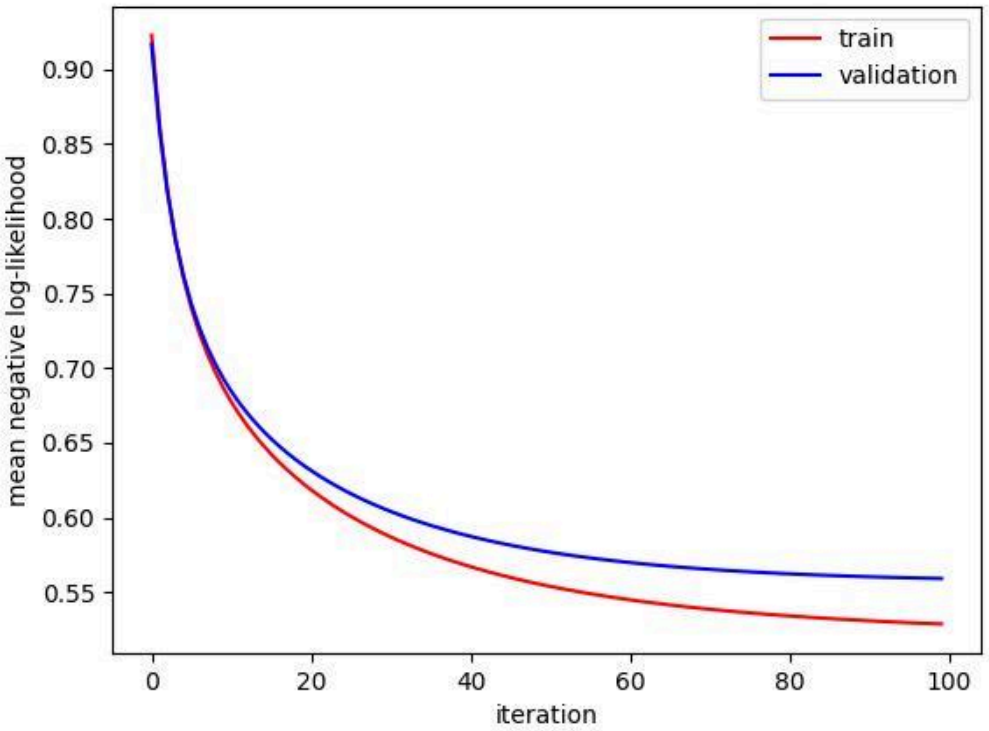
$$= \sum_{j=1}^{K} \frac{\partial}{\partial \theta_i} \left[\sum_{i'=1}^{N} c_{i'j} \theta_{i'} - c_{i'j} \beta_j - \log(1 + \exp(\theta_{i'} - \beta_j)) \right]$$

$$= \sum_{j=1}^{K} \left[c_{ij} - \frac{\partial}{\partial \theta_i} \log(1 + \exp(\theta_i - \beta_j)) \right]$$

$$= \sum_{j=1}^{K} \left[c_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right]$$

Similarly, for β_j, we have

$$\frac{\partial \ell}{\partial \beta_j} = \sum_{i=1}^{N} \left[\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} - c_{ij} \right]$$



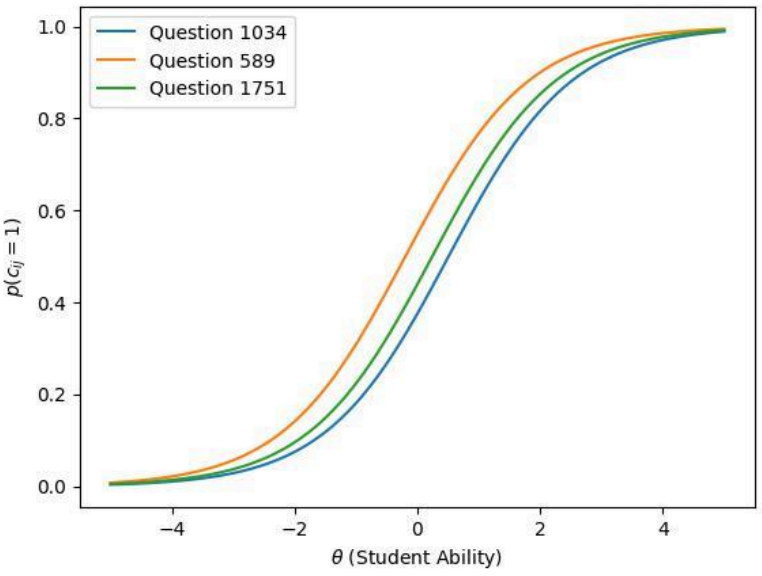
b)

c) seed was fixed with np.random.seed(311)

final validation accuracy: 0.7015241320914479

final test accuracy: 0.7033587355348575

d)



This graph essentially shows us how the probability of answering the question correctly changes with student ability. When putting 3 questions together on the same graph, we are able to compare the difficulty of each equation. For harder questions like Question 1034, the graph is shifted to the right and for easier questions like Question 589, the graph shifts to the left. For example, if we look at the cross-section where theta = 0, we see that a student with the same ability 0 across all questions is more likely to get Question 589 correct than Question 1034.

Q3) Option 2: Neural Networks

a) The first difference between the two approaches is the method of optimisation. In the Neural Network Autoencoder method, we utilise gradient based learning through the use of Backpropagation (with the use of a Stochastic Gradient Descent optimiser which updates the parameter but doesn't compute the gradients - that is dealt by backpropagation). On the other hand, in the Alternating Least Squares (ALS) we are manually computing the gradients and using Stochastic Gradient Descent to perform updates to u_n and z_m.

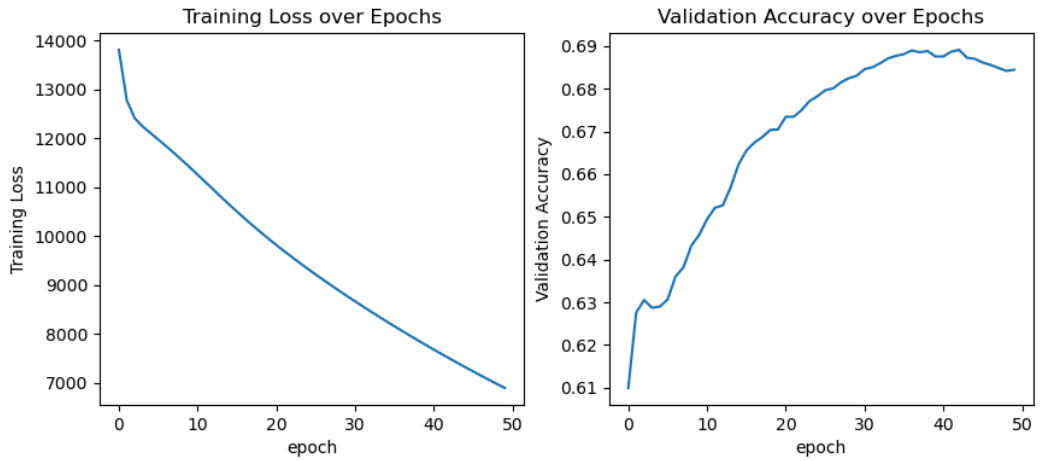
Another difference between the two approaches is the expressivity of the models. The ALS method is essentially assuming that the sparse matrix C can be decomposed into two matrices $U \in \mathbb{R}^{N \times k}$ and $Z \in \mathbb{R}^{M \times k}$ where N is the number of students and M is the number of questions. We are solving the matrix completion problem using a linear model, so performance of the ALS method may severely degrade if the data is highly non-linear. In the Neural Network Autoencoder method, we use a sigmoid activation function which expands the hypothesis space of our model through the use of non-linear activations, expanding the expressivity of our model.

The third difference between the ALS and Autoencoder method is their approaches to solve the matrix completion problem. In ALS, we are minimising the loss only on observed entries to get matrices U and Z such that the loss $(1/2) \sum_{(n,m) \in O} (C_{nm} - u_n^T z_m)^2$ is minimal. Then we infer the missing entries via the approximation $C \approx UZ^T$. The autoencoder method aims to reconstruct the sparse matrix by learning the underlying patterns (using the non-linear activation). It focuses on generalising over all data (even missing entries) by learning underlying relationships from observed entries.

b) Implemented in code.

c) The hyperparameters chosen are as such: $k^* = 50$, $num\_epoch = 50$ and $lr = 0.01$. The validation accuracy achieved using this hyperparameter configuration was 0.6878351679367768. However, this may vary as PyTorch neural networks are non-deterministic due to the initialised weights and biases being random.

d)



The final test accuracy was 0.6858594411515665.

e) After testing with different values for λ, I found that $\lambda = 0.001$ performed the best, reporting a validation accuracy of 0.6895286480383855 and a final test accuracy of 0.6768275472763196. From observing the data, it seems that the model does better without the regularisation penalty (as the test accuracy was higher for the model without the regularisation penalty), however, the reduction in accuracy may be due to the reduction in the magnitude of the weights. However, the more likely justification is to be that due to the non-deterministic nature of PyTorch Neural Networks (due to randomly initialised weight and bias matrices), there is some minor fluctuation in the accuracy scores.

Q4)

The ensemble process I implemented used 1 IRT model, 1 KNN model, and 1 NN model. The steps we used to ensemble are as follows:

1. Load the train, validation, and test data
2. Sample the train data 3 times with replacement, resulting in 3 different train datasets that are the same size as the original train data. Use each one to train a different model.
3. Train IRT, KNN, and NN models with optimal hyperparameters found in Q1 - Q3 on their respective sampled training datasets.
4. Evaluate each model for each data point in the test data, which results in 3 float predictions between 0 and 1, representing the confidence of the prediction from each model. Take the

average of those 3 float predictions to get a final float prediction, which is compared against the threshold of 0.5 to get a final classification for that data point (>=0.5 we will predict it is correct). Record whether our classification matched the true value.

5. Repeat step 4 for all data points in the test data, and report the final test accuracy.

We actually obtained worse performance using the ensemble than not using the ensemble. For each base model, our prediction accuracy was around 70%. However, with the ensemble, our accuracy was around 41%. This could be because neural networks generally require significantly more data to perform well, and the bootstrapping process reduces the effective size of the training data available to the NN. As a result, the NN struggled to generalise, achieving a much lower validation accuracy of around 40%, which dragged down the ensemble's overall performance.

Another issue could be that we weighted every model prediction equally. While IRT and KNN performed relatively well on their own, their contributions to the ensemble were diluted by the poor predictions from the NN. By assigning equal weights, we failed to account for the fact that some models (like IRT and KNN) were inherently more reliable in this low-data scenario. This lack of weighting amplified the negative impact of the NN's underperformance and prevented the ensemble from leveraging the strengths of the better-performing models.

Part B

Q1)

For part B, we decided to leverage Neural Networks from the autoencoder framework by integrating it with modified Item-Response Theory (IRT) concepts (ability and question characteristics) and utilising meta-data available.

Firstly, we create a student embedding matrix which is a $N \times P$ matrix where N is the number of students and P is the student embedding dimension (this is a hyperparameter). A student's embedding is a vector in latent space which numerically represents a student's characteristics (higher dimensional analogue of θ_i from the IRT model, reflecting the i^{th} student's ability). This can be thought of as building a profile for each student, encoding for ability in a range of question types in a higher dimensional space. The student embedding matrix is initialised with random values and is learnt during training through backpropagation. For each training batch, embeddings corresponding to the student IDs in that batch are selected.

Similarly, we create a question embedding matrix which is a $M \times Q$ matrix where M is the number of questions and Q is the question embedding dimension (this is a hyperparameter). A question's embedding is a vector in latent space which numerically represents a question's difficulty (higher dimensional analogue of β_j from the IRT model, reflecting the j^{th} question's difficulty). The question embedding matrix is initialised and optimised in the same manner as the student embedding matrix.

We also decided to use the student metadata available. In particular, we decided to use the gender and premium pupil status. We decided to not use the data corresponding to a student's date of birth as it seems to not be useful for the scope of this problem. We think that knowing the age at which the students answered a question would've been useful information as then we could discover some underlying structure from that, but we don't know when the questions were answered and thus couldn't extrapolate this information from the dates of birth.

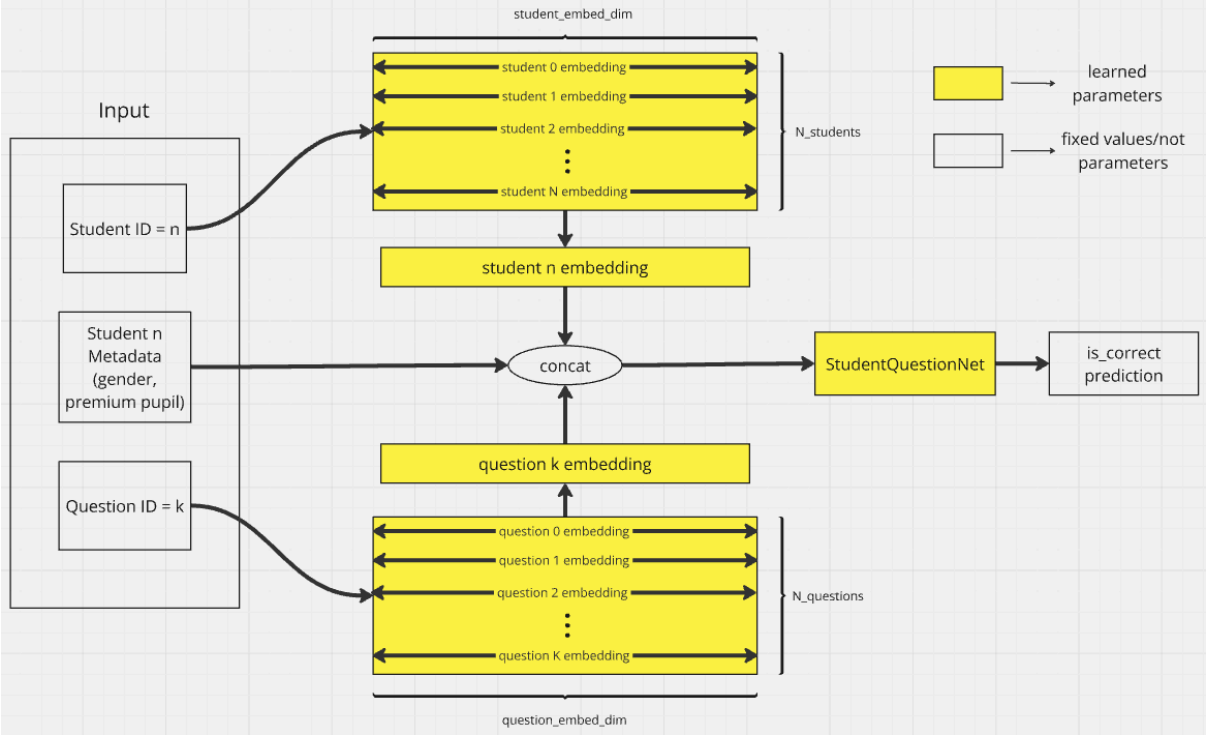
The embedding matrices are concatenated with the selected metadata, before being passed through the neural network. During backpropagation, the gradients of the loss function with respect to the embeddings are calculated. The Adam optimiser updates the embedding vectors in a direction that reduces the prediction error.

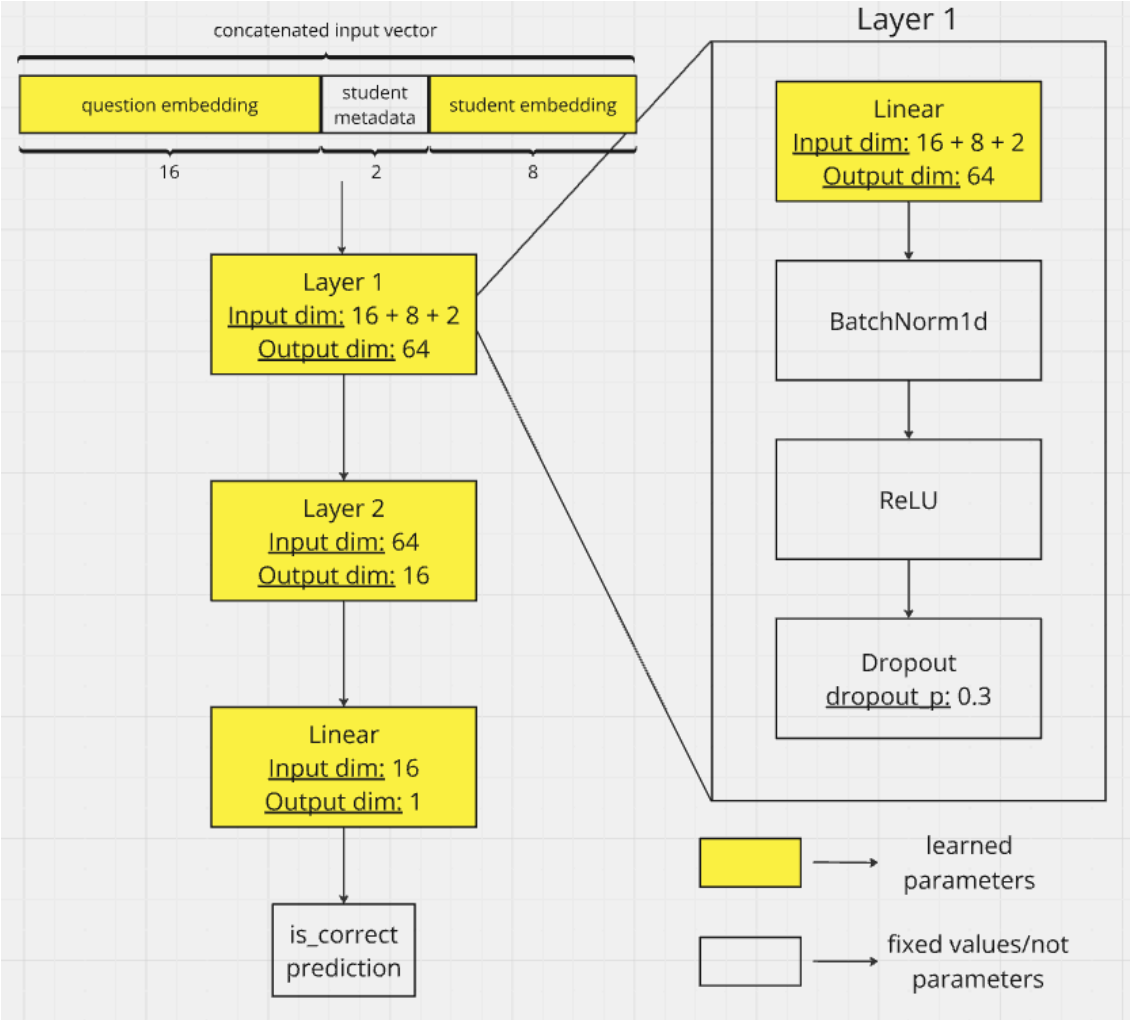
These combined features (student embeddings, question embeddings, and selected metadata) are passed through a fully connected neural network, which makes the predictions. The neural network consists of multiple hidden layers with ReLU activation functions, batch normalisation and dropout for regularisation, helping the model generalise better to unseen data and prevent overfitting. Dropout helps prevent overfitting by randomly deactivating a fraction of neurons during training. This is administered by the hyperparameter "dropout_p" which is the probability of deactivating a unit. Batch Normalisation normalises the inputs of each layer within a mini-batch which mitigates internal covariate shift, which is the resultant change in network activation's distribution from change in network parameters during training (Ioffe and Szegedy, 2015).

The final output layer produces a logit value, which is transformed into a probabilistic interpretation using the sigmoid activation function. The number of hidden layers and the number of units in a given layer are dictated by the "hidden_layers" array hyperparameter passed to the "train_model" function. Note that this is a hyperparameter for our model. We have additional hyperparameters, namely batch size and learning rate.

We believe that this modified approach should give us a higher accuracy as the higher dimensional analogues from the IRT model, added with Neural Network's versatility in expanding the hypothesis space via using non-linear activations should allow us to produce more accurate results.

Q2)





Q3)

After validation testing on our modified model, the hyperparameters we chose were as such:

```
"student_embed_dim": 8,
"question_embed_dim": 16,
"hidden_layers": [
    64,
    16
],
"dropout_p": 0.3,
"batch_size": 32,
"learning_rate": 0.001,
```

Model (with best hyperparameters)	**Validation Accuracy**	**Test Accuracy**
kNN (User-based)	0.6892464013547841	0.6821902342647473
kNN (Item-based)	0.6920688681907987	0.6790855207451313

Item Response Theory	0.7015241320914479	0.7033587355348575
Neural Network Autoencoder	0.6895286480383855	0.6768275472763196
Modified IRT with Neural Network	0.7118261456489563	0.7047699689865112

Evidently, our model performed slightly better than the baseline models explored in Part A. There is a greater difference between performances of the modified IRT with Neural Network model when compared with the kNN approaches and the Neural Network Autoencoder model. This difference in performance highlights the effectiveness of the modified IRT with Neural Network model in capturing intricate student-question relationships and addressing the limitations of simpler models. The kNN approaches, while straightforward and interpretable, lack the ability to model complex, non-linear relationships within the data. Similarly, the Neural Network Autoencoder, although capable of capturing latent factors, may not fully leverage the student-question dynamics in the same way that the IRT-based model does, particularly in scenarios where a more nuanced understanding of question difficulty and student ability is needed. This may be the reason as to why the Modified IRT with Neural Network and baseline IRT models have similar performance, due to their ability to capture the intricate relationships underlying in the data, explored via their respective parameters addressing question difficulty and student ability.

Q4)

Some settings where we would expect our model to perform poorly or where all existing models fail would be if we had too little data, an uneven distribution of the number of questions solved per student (some students solved ~400, some solved <50), and sparse representations. Also, if all of the students tend to perform worse on the beginning questions and learn and then get better, the question difficulties would also be skewed, and it becomes more difficult to predict.

We think that the limitations might be the way they are because of the nature of the problem, as student-question interactions are inherently sparse, and learning progression in general introduces temporal dependencies that are not easily captured by static embeddings. Furthermore, the uneven distribution of data means the model may be biased toward the behavior of students with more interactions, struggling with generalization for underrepresented ones. This could be caused by differences in student background, motivation, and curriculum.

Some possible ways that we could address these issues would be to normalize the contributions of students and questions in the dataset, perhaps by weighting sparse users or rare questions more heavily during training. We could also explore dynamic embeddings that evolve over time to capture students' learning progress, enabling the model to reflect changes in ability. Another approach would be clustering students or questions based on similar patterns, so sparse representations can leverage shared embeddings from similar groups. Additionally, incorporating regularization techniques and data augmentation methods to balance the dataset could help mitigate issues of sparsity and skewed distributions. Finally, we could also increase the amount of metadata fields that are recorded for both questions and users to enable the model to capture richer patterns and make more accurate generalizations.

Overall, this was a very interesting project, and it left us with two open questions. First, most models assume that students have a single-faceted ability to learn, but in reality, there are many different factors that contribute to problem solving. For example, there is problem solving speed, recall ability, and even emotional state. This brings us to the question, how can we build accurate models that can capture these complex learning traits? Second, training a model on educational data may inadvertently pick up biases that have to do with underrepresented groups or socio-economic status, which could lead to unfair recommendations based on a user's perceived background rather than the user's actual capabilities. How can we ensure that our models identify and mitigate biases, guaranteeing fairness in predictions and recommendations? And how can we design models that support equal opportunities for all students, enabling them to reach their full potential regardless of their background?

Bibliography

1. Ioffe, S. and Szegedy, C. (2015) 'Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift'. arXiv. Available at: https://doi.org/10.48550/arXiv.1502.03167.