CSC 311 Final Project Report

Blair Yang, Murphy Tian, Tony Chen

Due: Mar 30, 2023 at 5:00pm

1 Part A

1.1 k-Nearest Neightbor

In this problem, using the provided code, you will experiment with k-Nearest Neighbor(kNN) algorithm.

The provided kNN code performs collaborative filtering that uses other students' answers to predict whether the specific student can correctly answer some diagnostic questions. In particular, the starter code implements user-based collaborative filtering: given a user, kNN finds the closest user that similarly answered other questions and predicts the correctness based on the closest students' correctness. The core underlying assumption is that if student A has the same correct and incorrect answers on other diagnostic questions as student B, A's correctness on specific diagnostic questions matches that of student B.

1.1.1 Question 1 (a)

Complete a function **main** located at **knn.py** that runs KNN for different values of $k \in \{1, 6, 11, 16, 21, 26\}$ Plot and report the accuracy on the validation data as a function of k.

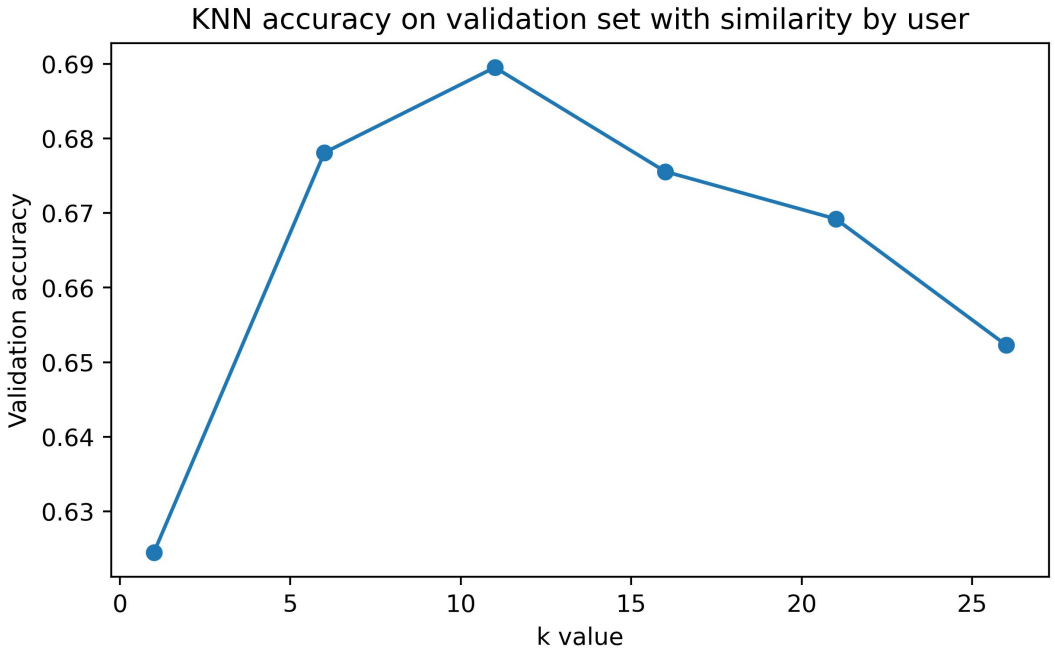

Figure 1.1: kNN accuracy on validation set with impute by user. Note that the accuracy peaked at $k^* = 11$.

1.1.2 Question 1 (b)

Choose $k^\star$ that has the highest performance on validation data. Report the chosen k^* and the final test accuracy.

Test accuracy of the kNN model with the maximum validation accuracy at $k^\star = 11$ being 0.6842.

1.1.3 Question 1(c)

Implement a function performs item-based collaborative filtering instead of user-based collaborative filtering. Given a question, kNN finds the closest question's correctness. State the underlying assumption on item-based collaborative filtering. Repeat part (a) and (b) with item-based collaborative filtering.

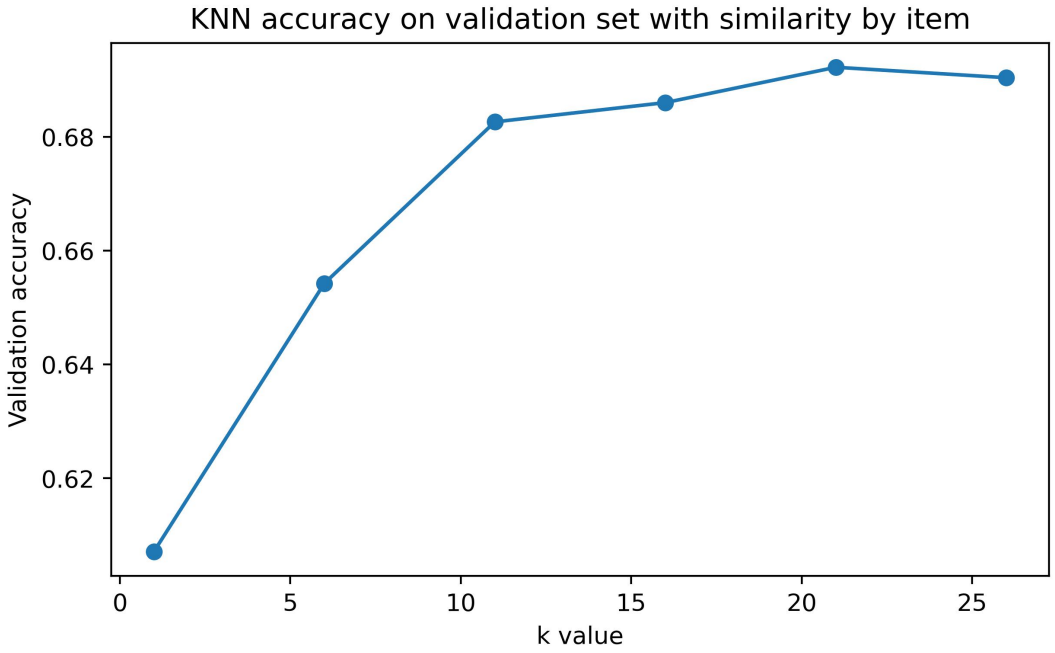

Figure 1.2: kNN accuracy on validation set with impute by item. Note that the accuracy peaked at $k^\star = 21$.

Test accuracy of the kNN model with the maximum validation accuracy at $k^\star = 21$ being 0.6816.

1.1.4 Question 1 (d)

Compare the test performance between user- and item- based collaborative filtering. State which method performs better.

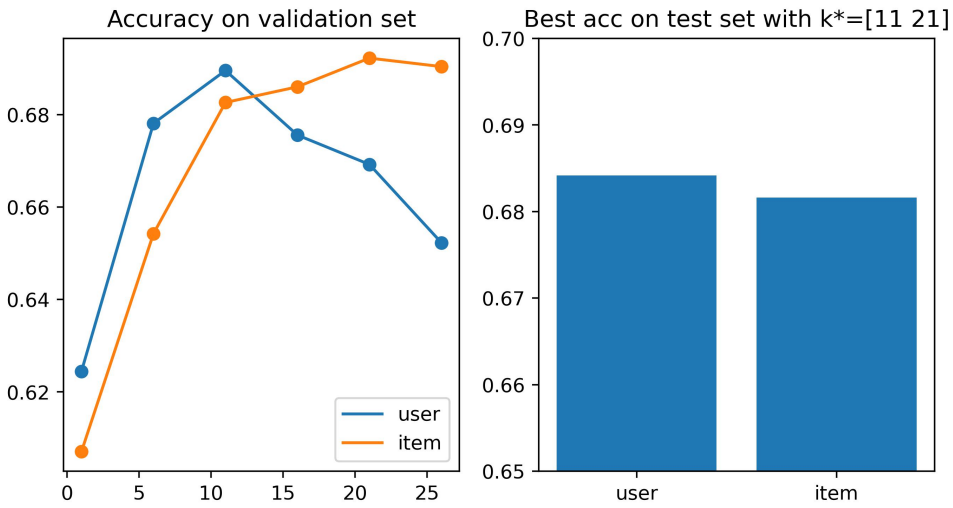

Figure 1.3

Based on the figure above, we claim them the user- collaborative filtering have better performance (slightly), with a improvement in test accuracy of 0.0026.

1.1.5 Question 1 (e)

List at least two potential limitations of kNN for the task you are given.

The curse of dimensionality: kNN models suffers from distant points and worsened computational time when dimension goes up.

Bad forward time: kNN is very slow when doing a forward pass, as it needs re-compute Euclidean distance with every point in the training set.

1.2 Item Response Theory

In this problem, you will implement an Item-Response Theory (IRT) model to predict students' correctness to diagnostic questions.

The IRT assigns each student an ability value and each question a difficulty value to formulate a probability distribution¿ In the one-parameter IRT model, β_j represents the difficulty of the question j, and θ_i that represents the i-th students ability. Then, the probability that the question j is correctly answered by student i is formulated as:

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

1.2.1 Question 2 (a)

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

From the given equation, we know that $c_{ij} = 1$ if the student answers correctly, and $c_{ij} = 0$ if the student answers incorrectly.
Hence, we claim the probability that

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

Assume the responses from students being independently and identically distributed, thus, the joint probability is the product of the individual probabilities:

$$\begin{aligned}\log_p(C|\theta, \beta) &= \sum_{i=1}^{N_{students}} \sum_{j=1}^{N_{questions}} c_{ij} \log \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)} + (1 - c_{ij}) \log \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)} \\ &= \sum_{i=1}^{N_{students}} \sum_{j=1}^{N_{questions}} c_{ij}(\theta_i - \beta_j) - \log(1 + exp(\theta_i - \beta_j))\end{aligned}$$

Then, taking the derivative of the logistic model with respect to θ_i, β_j, we have

$$\frac{\partial \log p(C|\theta, \beta)}{\partial \theta_i} = \sum_{j=1}^{N_{questions}} c_{ij} - \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}$$

$$\frac{\partial \log p(C|\theta, \beta)}{\partial \beta_j} = - \sum_{i=1}^{N_{students}} c_{ij} + \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}$$

1.2.2 Question 2 (b)

Implement missing functions that performs alternating gradient descent on θ and β to maximize the log-likelihood. Report the hyperparameters you selected. With your chosen hyperparameters, report the training curve that shows the training and validation log-likelihoods as a function of iteration.

We used hyperparameter with **learning rate** = 0.01, and **epochs** = 20

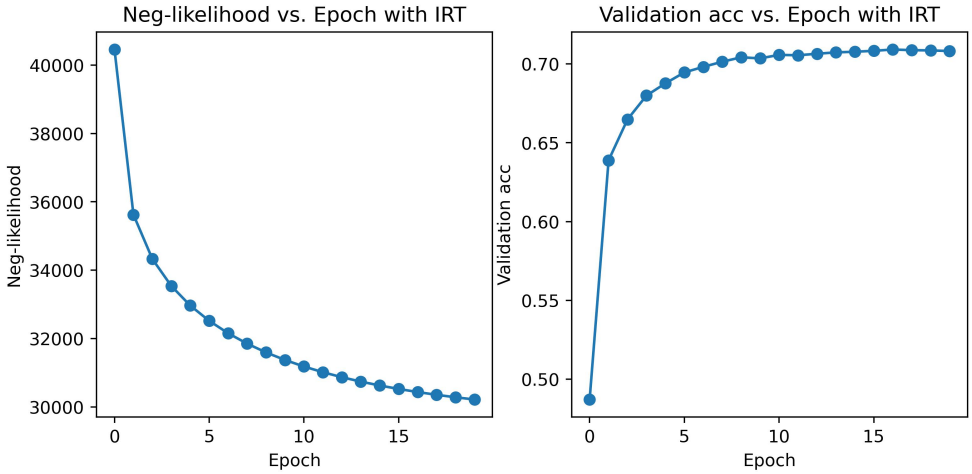

Figure 1.4 Negative log likelihood and validation accuracy in IRT.

1.2.3 Question 2 (c)

With the implemented code, report the final validation and test accuracies.

final validation = 0.7100
final test accuracy = 0.7062
Detailed implementation in `item_response.py` file

1.2.4 Question 2 (d)

Select three questions j_1, j_2, and j_3. Using the trained θ and β, plot three curves on the same plot that shows the probability of the correct response $p(c_{ij} = 1)$ as a function of θ given a question j. Comment on the shape of the curves and briefly describe what these curves represent.

Taking questions $j_1 = 1, j_2 = 100, j_3 = 1000$.

We find that the shape of the curve is similar with a sigmoid function.These curves suggests that as θ increases, the probability that the questions being answered correctly increases as well (bounded by $[0, 1]$).

The curve on the plot shows the probability of the correct response as a function of θ given a question j.

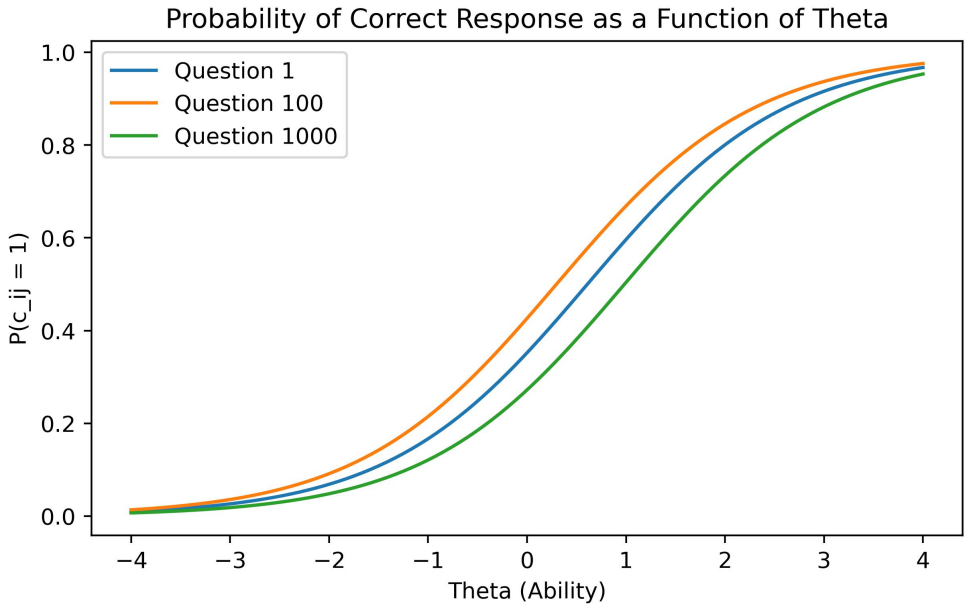

Figure 1.5 IRT probability with varying theta in IRT with fixed beta.

1.3 Neural Networks

In this problem, we implemented neural networks to predict students' correctness on a diagnostic question. Specifically, we designed an autoencoder model. Given a user $\mathbf{v} \in \mathbb{R}^{N_{questions}}$ from a set of users $\mathcal{S}$, our objective is

$$\min_{\boldsymbol{\theta}} \sum_{\mathbf{v} \in \mathcal{S}} ||\mathbf{v} - f(\mathbf{v}; \boldsymbol{\theta})||_2^2$$

where f is the reconstruction of the input $\mathbf{v}$. The network computes the following:

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

for some activation function h and g. In this question, we used sigmoid activation for both. $\mathbf{W}^{(1)} \in \mathbb{R}^{k \times N_{questions}}$ and $\mathbf{W}^{(2)} \in \mathbb{R}^{N_{questions} \times k}$, and $k \in \mathbb{N}$ is the latent dimension.

1.3.1 Question 3 (a)

Describe at least three differences between ALS and neural networks.

For the problem domains these two are used, ALS is an optimization algorithm often used in matrix factorization in scenarios like recommend system. Meanwhile, neural network is a computational model that is more expressive and can be applied various domains, for both regression and classification.

For the interpretability of results by these two, we have that ALS gives concrete weights in the form of matrices which can be understood and possibly reveal the underlying pattern, however, a neural network model can have an internal structure that is impossible to be comprehended due to its multi-layer.

For complexity when training, the computation required for ALS is usually less than that of neural network where for ALS, usually due to exponentially fewer parameters and that neural network could have different settings for number of layers and neurons leading to many more parameters to be considered.

1.3.2 Question 3 (b)

Implement a class AutoEncoder that performs a forward pass of the autoencoder following the instructions in the docstring.

See code `neural_network.py`.

1.3.3 Question 3 (c)

Train the autoencoder using latent dimensions of $k \in \{10, 50, 100, 200, 500\}$. Also, tune optimization hyperparameters such as learning rate and number of iterations. Select $k^\star$ that has the highest validation accuracy.

We found $k^\star = 10$ has the highest validation accuracy 0.6871. We set the hyperparameters to be the following, for optimal performance:

Parameter	Value
Learning rate	0.1
Emebdding dim ($k^\star$)	10
λ	0
Epochs	10

Table 1: Hyperparameters of the neurl net model without regularization.

1.3.4 Question 3 (d)

With $k^\star = 10$, we obtain training and validation objectives as two functions of epoch.

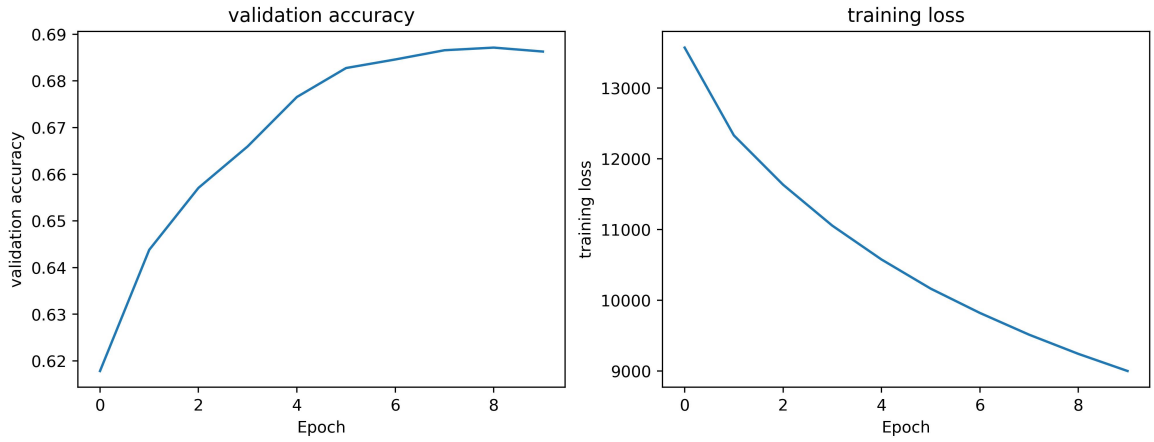

Figure 1.6: Validation accuracy and training loss of our model with fine-tuned hyperparams.

And the final test accuracy is 0.68304, obtained by hyperparameters addressed in **Table 1**.

1.3.5 Question 3 (e)

The final validation accuracy is 0.6923 and the final test accuracy is 0.6825, obtained by $k^\star = 10$, epoch $= 10$, learning rate $= 0.1$, regularization coefficient $\lambda = 0.001$. Our model

Parameter	Value
Learning rate	0.1
Emebdding dim ($k^\star$)	10
λ	0.001
Epochs	10

Table 2: Hyperparameters of the neurl net model with regularization.

performance worsens with increasing $\lambda > 0.01$.

Our current implementation of the model does not over fit; thus, higher values of λ does not improve our model, but rather reduces model performance.

1.4 Ensemble

In this problem, you will be implementing bagging ensemble to improve the stability and accuracy of your base models. Select and train 3 base models with bootstrapping the training set. You may use the same or different base models. Your implementation should be completed in ensemble.py. To predict the correctness. Report the final validation and test accuracy. Explain the ensemble process you implemented. Do you obtain better performance using the ensemble? Why or why not?

We implemented the ensemble model from three base models: kNN, Item Response Theory, and Neural Network models, with the following hyper parameters

Model	$k^\star$	**Learning Rate**	**Epochs**	λ
kNN	11	-	-	-
IRT	-	0.01	20	0
Neurl net	10	0.1	20	0

Table 3: Hyperparameters for the three models used in the study.

After running the model with the above parameters, we obtained the following accuracies, as given in **table 2**.

Model	**Validation accuracy**	**Test Accuracy**
kNN	0.6895	0.6842
IRT	0.7099	0.7061
Neurl net	0.6925	0.6825
Ensemble	0.7058	0.7019

Table 4: Validation and test accuracies for different models.

To implement ensemble, we process it as follows:

1. Load the data and split it into training, validation, and test sets.
2. Bootstrap the training set by generating three different bootstrapped of training data that based on each base model
3. Train each base model on its respective bootstrapped dataset. Use the trained base models to make predictions on the validation and test sets.
4. Combine the predictions, and then evaluate the ensemble model on the validation and test set.

We found that ensemble model achieved a better performance than kNN and Neural Network, but with similar (or worse) result when compared to IRT model. It does not achieve a significant improvement over the base models. This suggested the ensemble did not reduce the bias during estimation.

2 Part B

2.1 Formal description

We proposed a **Deep-attention model** to solve the problem, and achieved 73% accuracy (0.7302) on the validation set and 74% accuracy on test set (0.7367), which improved significantly over the baseline neural net model.

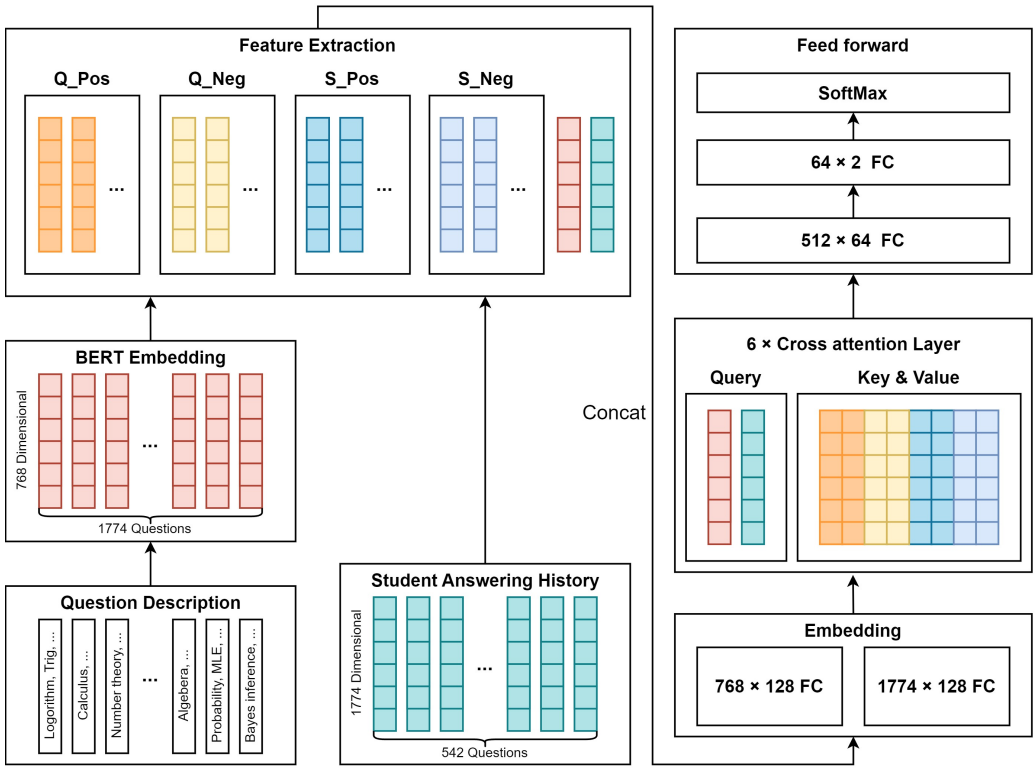

Figure 2.1: Model architecture.

We extended the model architecture based on the "Part A. 3 neural network", and used a modified version of it as the feed forward block that outputs the prediction. Our model was inspired by the self-attention architecture in popular tensor2tensor tasks like image classification (ViT), which uses multi-layer of multi-head self-attention mechanism to extract useful information from context.

We used four sequences of embedding indices (non-fixed length) and two query vector indices (fed into the net one at a time) as input to our model, and outputs a binary classification result, which is a probability vector on $\mathbb{R}^2$.

Most importantly, we used the following scalar product to improve our model:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^\top}{\sqrt{d_k}})V$$

Where Q, K, V are the query, key, and value matrices from input, d_k is the embedding dim.

We believed that this is the only important equation that we used in our model. All the other parts of the model are just simple linear layers with $y = Wx + b$.

We expect the model to out-perform the baselines by a significant amount due to the following reasons:

a.) **Richer Representations**: The model leverages BERT embeddings, which are pre-trained on a large corpus and can capture complex semantic information. This means that the model has access to more expressive features than a simple 2-layer FC network.

b.) **Attention Mechanisms**: The attention mechanisms allow the model to focus on the most relevant parts of the input sequences for making predictions. This enables the model to learn more meaningful relationships between students and questions, whereas a simple 2-layer FC network would not be able to perform this type of selective weighting.

c.) **Context-Aware Embeddings**: The model computes context vectors for students and questions, which capture the relationships between the input sequences. These context-aware embeddings allow the model to better understand the interactions between students and questions, leading to improved performance compared to a 2-layer FC network that lacks this level of contextual understanding.

2.2 Model parameters & introduction

Parameter	Value
Learning rate	0.0003
No. Attention Layers	6
No. heads	8
Embedding dim	128
Dropout	0.05
Loss	Cross Entropy

Table 5: Hyperparameters of the deep-attention model.

To predict the correctness of i^{th} student doing j^{th} question, our model uses the following input indices:

a.) The set of questions that previously done correctly by student s_i, denoted as `Q_pos`

b.) The set of questions that previously done in-correctly by student s_i, denoted as `Q_neg`

c.) The set of students that previously done question q_j correctly, denoted as `S_pos`.

d.) The set of students that previously done question q_j in-correctly, denoted as `S_neg`.

Note that our model does not have a position-specific encoding, thus the order of the set does not matter when passed into the model.

Our model contains two rule-based feature-extraction blocks and three neural network blocks. The exact usage and purposes are given below:

2.3 Pre-processing & Feature extraction

2.3.1 Matrix processing

The purpose of these steps is to prepare the train data sparse matrix for input to the model.

a.) We modified the train data sparse matrix, with 1 representing a correct answer, -1 representing incorrect, 0 representing NaN

b.) We created a dictionary called PREPROCESSED_DATA_DICT that has keys (i, j) and maps these keys to the a tuple of the four sequences of indices of the questions and students described above, as `Q_pos_ids, Q_neg_ids, S_pos_ids, S_neg_ids`.

c.) Retrieve the question and student embedding for each non-zero entry (i, j) in the Training matrix, record as $\texttt{Q}_{i,j}$, $\texttt{S}_{i,j}$, respectively.

2.3.2 BERT embedding

We used BERT embeddings to represent questions based on their topics, as they provide a rich representation that captures semantic information.

a.) We paired each question with their subject id from `question_meta.csv`

b.) We used the subject ids as queries to retrieve the corresponding keywords from `subject_meta.csv`

c.) We passed the keywords to the **BERT** encoder, and obtain the topic embedding for each question ($49,152$ dimensional = 64×769)

d.) We summed the topic embedding by the last axis to get the question embedding for each question (768 dimensional)

2.4 Forward pass

This section describes how the input embeddings are processed by the model to produce the final binary classification output.

a.) Retrieve the student embedding using indices previously recorded as `S_pos_ids, S_neg_ids`, as `S_pos, S_neg`, multiply `S_neg` with -1
(each with dimension $No.Students_{pos/neg} \times 1774$)

b.) Retrieve the question embedding using indices previously recorded as `Q_pos_ids`, `Q_neg_ids`, as `Q_pos`, `Q_neg`, multiply `Q_neg` with -1
(each with dimension $No.questions_{pos/neg} \times 768$)

c.) Pass `S_pos`, `S_neg`, `Q_pos`, `Q_neg`, `Q`, `S` to the transformer model

d.) Pass `S_pos`, `S_neg`, `S` to `transformer.student_embed`, which is a 1774×128 matrix to map the student vectors.

e.) Pass `Q_pos`, `Q_neg`, `Q` to `transformer.question_embed`, which is a 768×128 matrix to map the question vectors.

f.) Use the modified six attention blocks to compute the attention (cross attention), use `S` as **query**, and `S_pos` $\oplus$ `S_neg` as **key** and **value**, to compute the question context vector `Qc`

g.) Use the modified six attention blocks to compute the attention (cross attention), use `Q` as **query**, and `Q_pos` $\oplus$ `Q_neg` as **key** and **value**, to compute the student context vector `Sc`

h.) Get the two context vectors, concatenate it with `Q`, `S` as `v = Sc` $\oplus$ `Qc` $\oplus$ `Q` $\oplus$ `S` (512 $= 4 \times 128$ dimensional) and send `v` to a two-layer FF network ending with softmax to give a 2-dimensional probability output.

2.5 Comparison

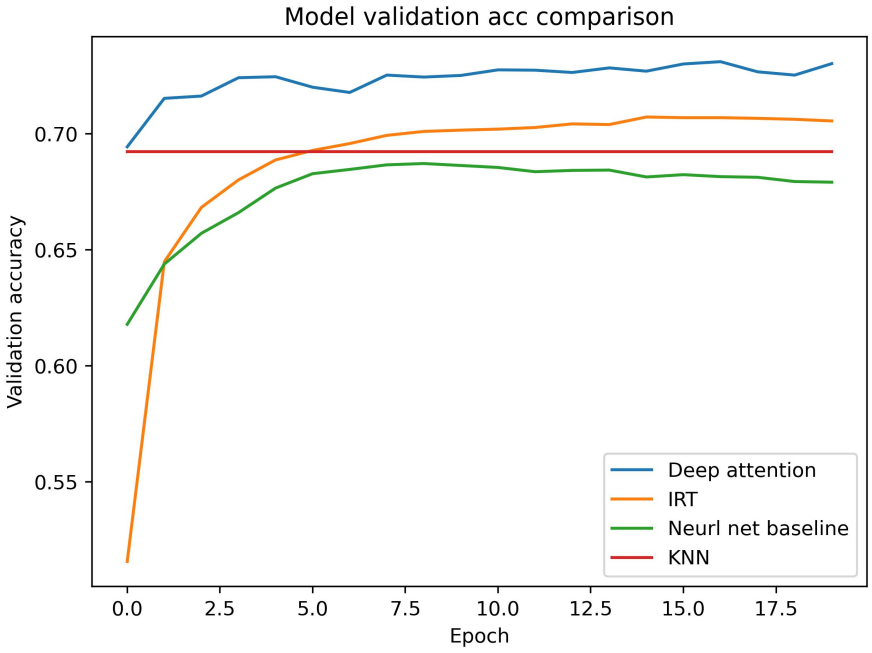

Figure 2.2: An overview of validation accuracy of our deep-attention model over other baselines, note kNN is a non-parametric model and has constant validation accuracy over

all epochs.

Model	Validation accuracy	Test Accuracy
kNN	0.6895	0.6842
IRT	0.7099	0.7061
Neurl net	0.6925	0.6825
Ensemble	0.7058	0.7019
Deep-attention	**0.7302**	**0.7367**

Table 6: Validation and test accuracies for different models.

Our model demonstrated superiority in performance in all baseline models, meeting our expectations. The reasoning why our model might perform better is given above, which we would not address here for ease of presentation.

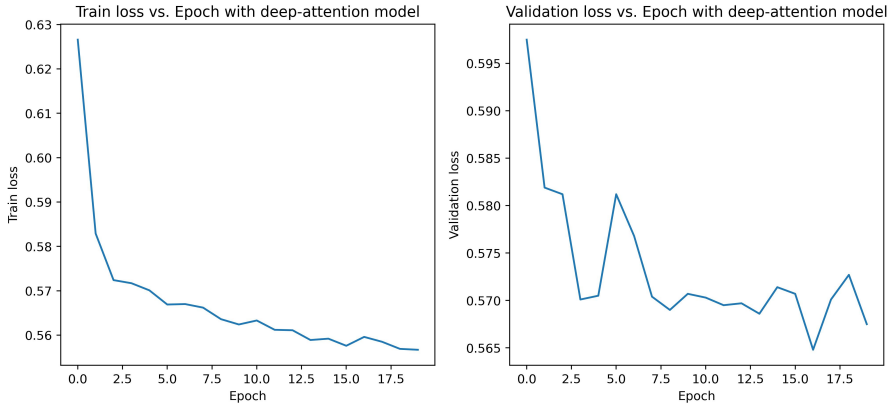

Figure 2.3 Training loss and validation loss over epochs during training.

2.6 Limitations

2.6.1 Potential performance drop and reasoning

If properly trained, we believed that our model are able to generalize our data pretty well, with one exception of having questions that has only been done by one specific question, this may cause the context vector to be zero and thus significantly alter its prediction.

Our model to subject to several limitations:

a.) **Lack of global data ingestion:** our model attends to filtered data, but not the global data, which might able to provide some useful information for classification.

b.) **Efficiency:** our model uses a deep attention network, which uses a significant amount of computational power during training. Training one epoch on a RTX 3080Ti takes approximately 2 minutes, which is significantly slower than the baselines.

c.) **Lack of layer norm:** Our model did not use the feed forward, residual connection, or layer norm for our attention block. We experimented such implementation but the results turned to be less accurate and significantly boosted the training time. We believed that this might because of lack of optimization and fine-tuning. Our model might be able to gain some performance after correct and elegant implementation of such structures.

2.6.2 Extension

Our model follows the implementation of the most state-of-the-art model vision transformer (ViT). Thus, our model cannot be extended significantly. We believed an extension can be introducing global attention (rather extracting feature by human). However, this would significantly increase training time.

3 Contribution

Blair Yang contributed to the coding in Part B and part A, and writing in Part B.

Murphy Tian contributed to the writing in Part A, B and coding in Part A.

Tony Chen contributed to the writing in Part A and coding in Part A.