CSC 311 Winter 2024
Final Project

Pranjal Agrawal (1009526292), Rishit Dagli (100884068),
Law Kieu (1006663914), Shivesh Prakash (100869379)

April, 2024

Contents

1 Part A

1.1 Collaborative filtering with k-Nearest Neighbor

(a) The plot the validation accuracy as a function of k is given below,

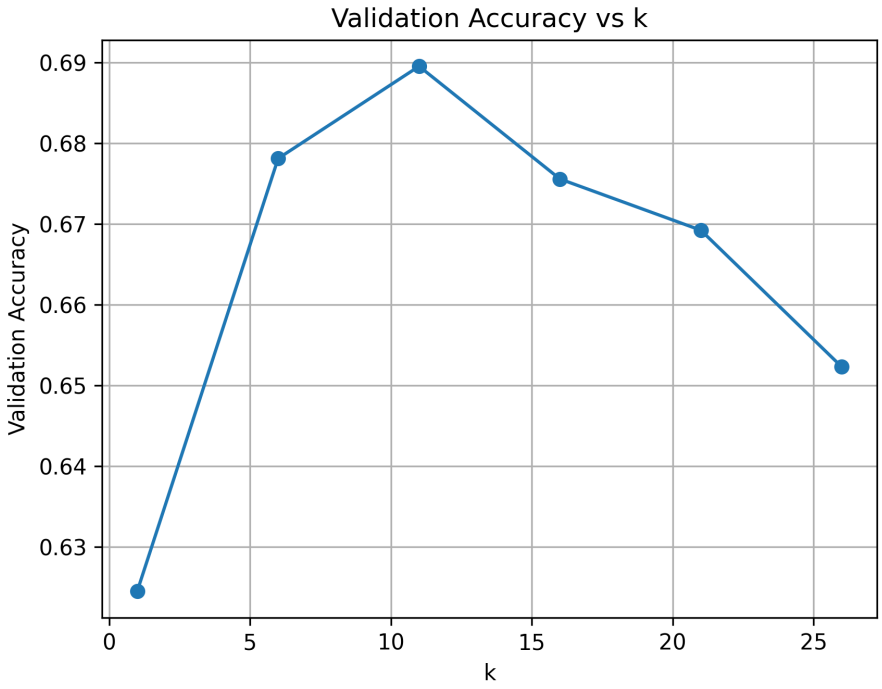

Figure 1: Plot the validation accuracy as a function of k for user-based collaborative filtering.

We choose $k = 11$ and the test accuracy of the final model is 0.6841659610499576.

(b) The underlying assumption is that a student will answer question A similarly to how the student answers the closest question B. In a way, it assumes that questions close to each other are similar and the student will repeat the same mistakes or achieve the same success.

(c) The plot of the validation accuracy as a function of k is given below,

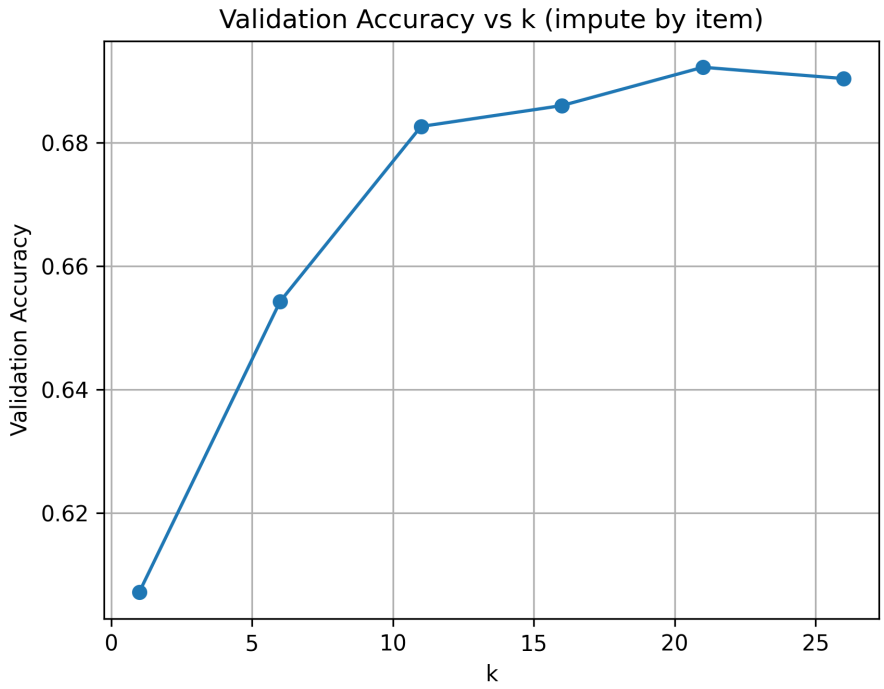

Figure 2: Plot the validation accuracy as a function of k for item-based collaborative filtering.

We choose $k = 21$ and the test accuracy of the final model is 0.6816257408975445.

(d) The best performing user-based collaborative filtering achieves a training accuracy of 0.6841659610499576, while the best performing item-based collaborative filtering achieves a training accuracy of 0.6816257408975445. Thus the user-based collaborative filtering performs better.

(e) Some limitations of the kNN task are listed here,

- For the user-based filtering, kNN assumes that similar performing students will continue to perform similarly across all questions. However, some students may have exceptional strengths or weaknesses in specific concepts, leading to discrepancies in performance for certain questions.
- For the item-based filtering, kNN relies on the similarity of student responses to make predictions, it may overlook the variability in question topics and concepts. Even questions with close question IDs may test different aspects of a subject, making them less suitable for comparison.
- The kNN algorithm requires calculating distances between the target point and all points in the training set. As the size of the dataset increases, the computational cost of finding nearest neighbors grows significantly.
- In high-dimensional spaces, such as those encountered in this datasets with numerous questions and students, the notion of distance becomes less meaningful.

- kNN faces challenges when dealing with new students or questions not present in the training dataset, known as the cold start problem. Without sufficient historical data to establish similarities, kNN may struggle to provide accurate predictions for these unseen instances.

1.2 The Item Response Theory Model

(a) Given the probability that student i answers question j correctly is:

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

The log-likelihood for one student-question pair is:

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

To express this for all students and questions, we sum over all i and j:

$$\log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta}) = \sum_{i,j} c_{ij} \log \left(\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right) + (1 - c_{ij}) \log \left(1 - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right)$$

Let $y = p(c_{ij} = 1|\theta_i, \beta_j) = \frac{\exp(z)}{1+\exp(z)}$, and $z = \theta_i - \beta_j$:

$$\begin{aligned}
\frac{\partial}{\partial \theta_i}(\log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})) &= \frac{\partial}{\partial \theta_i}(\log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})) \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial \theta_i} \\
&= \frac{\partial}{\partial \theta_i} \sum_{i,j} (c_{ij} \log y + (1 - c_{ij}) \log(1 - y)) \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial \theta_i} \\
&= \sum_j \left(\frac{c_{ij} - y}{y(1 - y)} \right) \cdot y(1 - y) \cdot 1 \\
&= \sum_j (c_{ij} - y) \\
&= \sum_j \left(c_{ij} - \frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} \right)
\end{aligned}$$

Similarly,

$$\begin{aligned}\frac{\partial}{\partial \beta_j}(\log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})) &= \frac{\partial}{\partial \beta_j}(\log p(\mathbf{C}|\boldsymbol{\theta}, \boldsymbol{\beta})) \cdot \frac{\partial y}{\partial z} \cdot \frac{\partial z}{\partial \beta_j} \\ &= \sum_i \left(\frac{c_{ij} - y}{y(1-y)}\right) \cdot y(1-y) \cdot (-1) \\ &= \sum_i (y - c_{ij}) \\ &= \sum_i \left(\frac{\exp(\theta_i - \beta_j)}{1 + \exp(\theta_i - \beta_j)} - c_{ij}\right)\end{aligned}$$

(b) The completed code for missing functions can be found in `part_a/item_response.py`.

(c) We run a grid search across the following hyperparameters:

- learning rate $(lr) \in [0.01, 0.05, 0.1]$
- iteration $(i) \in [5, 15, 25, 35, 45, 55]$

Through this we find the best performing hyperparameters listed in Table 1 with training/validation/test accuracy in Table 1 and Figure 3 as follow:

lr	i	Validation accuracy	Test accuracy
0.01	35	0.7076	0.7090

Table 1: Best learning rate and iterations with the respective Validation and Test Accuracy

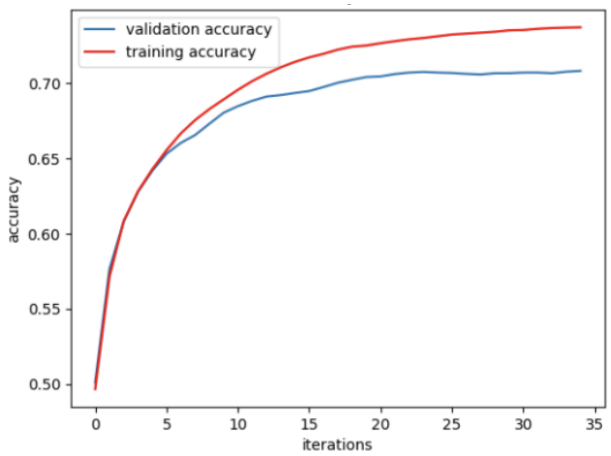

Figure 3: Training and Validation Accuracy over iterations

(continued)

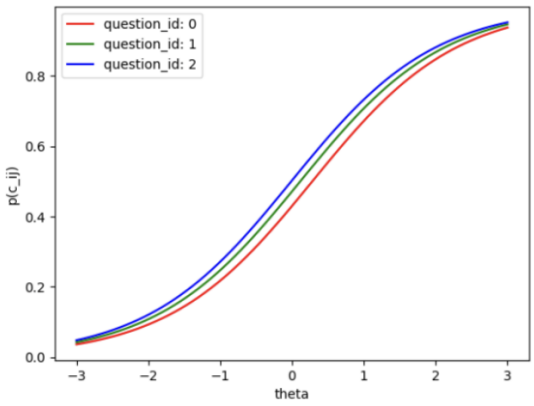

Figure 4: The probability of the correct response ($p(c_{ij} = 1)$) across the ability theta ($\boldsymbol{\theta}$)

(d) We select three questions in the dataset of $id = 1, 2, 3$ to plot the probability of the correct response ($p(c_{ij} = 1)$) across the ability $\boldsymbol{\theta}$ in Figure 7

We observe the shapes of the curves to resemble the Sigmoid function closely. The curves represent the IRT method of determining the probability of a student with ability θ to answer correctly questions. The questions' various difficulties (β) can be attributed to each curve's centre at $p(c_{ij} = 1) = 50\%$, where a lower theta suggest easier question, and vice versa.

1.3 Neural Networks

(a) The completed code for `AutoEncoder` class can be found in `part_a/neural_network.py`.

(b) We run a grid search across the following hyperparameters:

- $k \in [10, 50, 100, 200, 500]$
- $\alpha \in [0.01, 0.03, 0.05]$
- number of epochs $\in [10, 20]$

Through this we find the best performing hyperparameters listed in Table 2 which get to validation accuracy of 0.6905165114309907.

k	α	epochs
10	0.03	20

Table 2: Best Hyperparameters without regularization found through grid search.

(c) With the chosen hyperparameters, we show the average training and validation loss curves in Figure 5. These hyperparameters get us to a Test Accuracy of 0.6785210273779283.

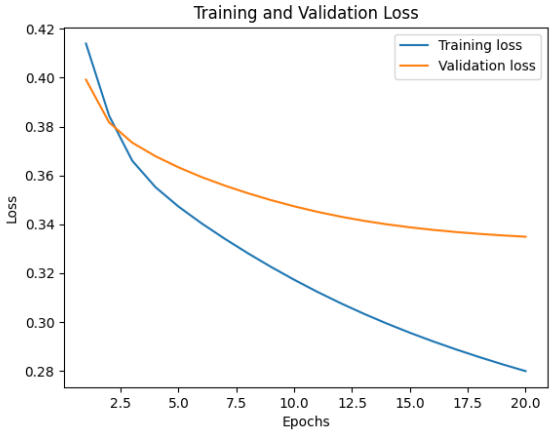

Figure 5: Training and Validation Loss for best hyperparameters without regularization.

(d) We perform a grid search using the values in Table 2 and $\lambda \in [0.001, 0.01, 0.1, 1]$.

We choose $\lambda = 0.001$, thus we have all the hyperparameters given in Table 3.

k	α	epochs	λ
10	0.03	20	0.001

Table 3: Best Hyperparameters with regularization found through grid search.

For this setting, we observe that the validation accuracy is 0.6255997742026531 and the test accuracy is 0.625176404177251.

We particularly observe that the model with regularization does not perform better than the model without regularization penalty.

2 Part B

2.1 Method

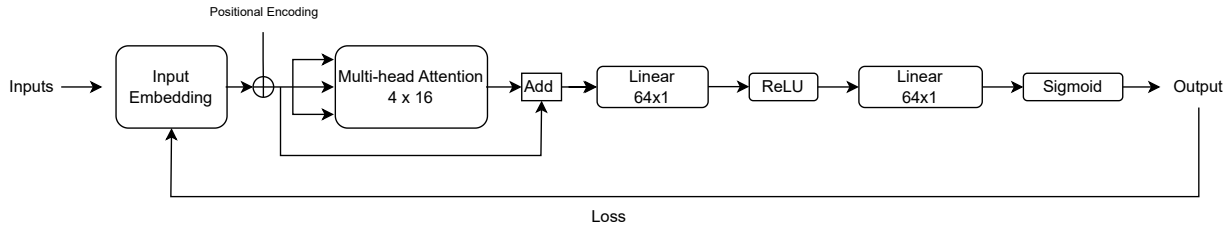

Figure 6: **Model overview.** The model uses Multi-head Self-Attention (4 heads with an embedding dimension of 16) to learn the features in the embedding space. The output is then processed by two fully connected layers, each with 64 nodes [1].

Architecturally, We modify the Auto-encoder model to use Multi-headed Self-Attention [1]. For our problem, the Auto-encoder model learns to compress the performance matrix and pick out the patterns necessary to reconstruct the matrix, including the originally unknown student performance on a question, with the minimal loss. The problem is sequential since each prediction involves patterns from student's past performance on different questions or others' performance of questions. The sequential nature of the problem, though, underscores Auto-encoder's weakness in only capturing and using the highly salient features of the training data. Whereas, Multi-headed Attention is designed to learn both the possible patterns and the sequential model of the data by tending to only the useful features [1].

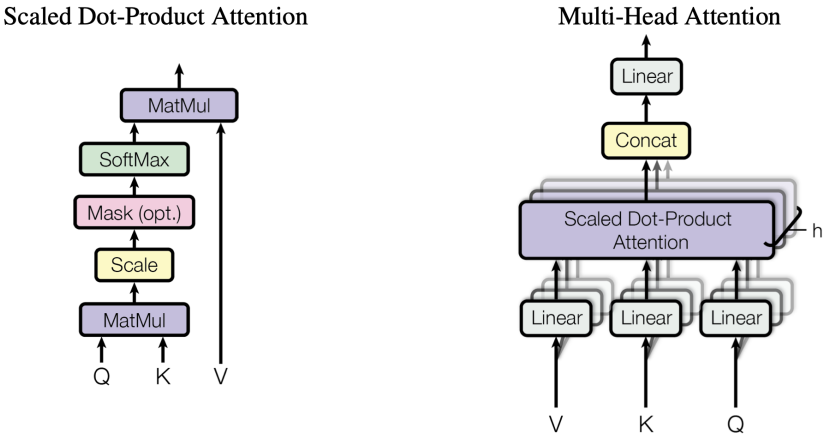

Figure 7: (left) Scaled Dot-Product Attention. (right) Multi-Head Attention consists of several attention layers running in parallel [1].

Through experimenting different optimization methods, we also found that utilizing the AdamW optimizer with a scheduling tweak instead of SGD, ReLU activation function in place of Sigmoid, and a custom loss function modifying MSE noticeably reduce overfitting and improve the model's convergence.

- During training, instead of SGD's one global static learning rate, AdamW's adaptive learning rates help the model variate the local learning rate for inputs with constrasting nature, like student's ability and question's difficulty. AdamW also steers away from processing batches independently, instead incorporating momentum by also processing a fraction of the previous gradients and accelerating the gradient in the right direction, which leads to a faster convergence. [2].
- If the validation accuracy plateaus, we also schedule the model to reduce the learning rate to avoid being stuck at a local minima. If the training loss plateaus for at least 4 epochs, we decreases the learning rate by 10%.
- We chose ReLU instead of Sigmoid as the common practice for Deep Learning models like Multi-Headed Attention [3].
- While MSE can proportionately penalize inaccurate weights, we recognize that MSE may be under-compensating for outliers. For a regularizing effect, we also see that L^2 method with quadratic loss may over-penalize the outliers, while L^1 method with absolute loss may under-penalize them. Thus, we aim for a balance between the two methods when penalizing large inaccuracy. Formally,

$$\mathcal{L} = \begin{cases} \frac{(t-y)^2}{2} & \text{if } t-y < \Delta \\ \Delta|t-y| - \frac{\Delta^2}{2} & \text{if } t-y \geq \Delta \end{cases} \tag{1}$$

  Here, t is the binary target, y is the model's prediction and Δ is a hyperparameter.

Overall, our model with Multi-head Attention was able to adapt its learning rate and loss over different inputs to achieve better results across different statistics.

2.2 Implementation Details

We performed hyperparameter tuning using random search instead of grid search. This allowed us to efficiently explore a wide range of hyperparameter combinations without exhaustively searching through all possible combinations.

The best hyperparameters we found are as follows:

- **Initial Model**:
  - Optimizer: AdamW
  - Learning Rate Scheduler: None
  - Learning Rate: 1e-1
  - Batch Size: 64
  - Epochs: 8

  - Random Seed: 3047
- **Multi-Head Self-Attention Model**:
  - Optimizer: AdamW
  - Learning Rate Scheduler: ReduceLROnPlateau
  - Patience: 15
  - Learning Rate: 1e-1
  - Batch Size: 32
  - Epochs: 250
  - Random Seed: 3047

2.3 Comparison with Baseline Models

The evaluation metrics used in the comparison represent different aspects of the model's performance in predicting whether a student will answer a diagnostic question correctly or not. Here is a brief description of what they represent:

- **Accuracy (Acc):** measures the proportion of correctly predicted answers out of all predictions made by the model.
- **True Positive Rate (TPR):** measures the proportion of actual positive instances that are correctly predicted by the model.
- **True Negative Rate (TNR):** measures the proportion of actual negative instances that are correctly predicted by the model.
- **False Positive Rate (FPR):** measures the proportion of actual negative instances that are incorrectly predicted as positive by the model.
- **False Negative Rate (FNR):** measures the proportion of actual positive instances that are incorrectly predicted as negative by the model.
- **F1 Score:** The harmonic mean of precision and recall, F1 score provides a balance between precision (the ability of the model to correctly identify positive instances) and recall (the ability of the model to find all positive instances).

In the context of student assessments, metrics like TPR and TNR are particularly important as they directly impact the ability to accurately identify correct and incorrect answers. Maximizing TPR and TNR while minimizing FPR and FNR is crucial for ensuring accurate assessments.

Let's compare our model's performance against the various baseline models,

- Our model achieves competitive accuracy compared to baseline models, indicating its ability to make correct predictions overall.
- The high TPR of our model suggests that it effectively identifies correct answers by students, leading to fewer missed detections of correct answers.

	Val Ac ↑	Test Ac ↑	TPR ↑	TNR ↑	FPR ↓	FNR ↓	F1-0 ↑	F1-1 ↑
Bayesian Reg	59.84	48.99	33.25	72.21	27.79	66.75	0.53	0.44
Linear Reg	59.84	49.03	33.25	72.28	27.72	66.75	0.53	0.44
Random Forest	60.82	59.78	67.12	48.95	51.05	32.88	0.50	0.67
XGBoost	65.57	64.52	80.81	40.50	59.50	19.19	0.48	0.73
KNN-user (1.1.a)	68.95	68.42	82.19	48.11	51.89	17.81	0.55	0.76
KNN-item (1.1.c)	69.22	68.16	78.64	52.72	47.28	21.36	0.57	0.75
ITR (1.1.c)	70.76	70.90	78.49	58.45	41.55	21.51	0.61	0.76
NN (1.3.c)	69.05	67.85	77.40	54.89	45.11	22.60	0.58	0.74
NN-reg (1.3.d)	62.56	62.52	78.11	39.04	60.96	21.89	0.46	0.71
Ours	69.60	70.62	83.14	52.16	47.84	16.86	0.59	0.77

Table 4: Comparing our models with the baseline models in part (a) and some other common models

- The relatively high TNR indicates that our model also performs well in identifying incorrect answers, leading to fewer false alarms.
- Our model demonstrates balanced performance with one of the lowest FPR and FNR, indicating a good trade-off between minimizing false alarms and missed detections.
- The competitive F1 scores for both classes further confirm the model's ability to maintain a balance between precision and recall, ensuring robust performance across different prediction scenarios.

2.4 The Model's Performance

The model implemented in Part B uses the Reduce Learning Rate on Plateau scheduler to dynamically change the learning rate. This scheduler adjusts the learning rate if the validation loss plateaus. A plateau could imply that the learning rate is too high, causing the weights to oscillate. Reducing the learning rate in such a situation helps the model converge to a minimum. Changing learning may also help adapt to changes in the dataset, such as if the students in the batches have significantly different performances than the previous students.

The model also makes use of the AdamW [2] optimiser. This algorithm determines different learning rates for each parameter. As a result, it can adapt better to the characteristics of the dataset. It is also less sensitive to the initial values of the learning rate.

We have also changed the loss function to a custom loss function. This is less sensitive to outliers since it uses Mean Absolute Error Loss for large error values. On the other hand, the neural network implemented in Part A always uses Mean Squared Error Loss.

Multi-headed Self Attention [1] excels at capturing sequential patterns. Thus, it is effective in a problem such as this, where a student's answer's correctness depends on the correctness of the student's other answers and the correctness of the answers of other

students to the same question. On the other hand, an autoencoder may not preserve such sequential patterns.

For these reasons, this model performed better than the neural network implemented in Part A

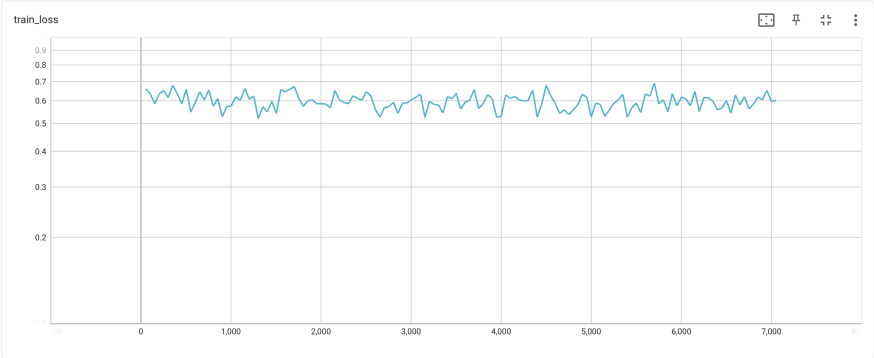

Figure 8: Training Loss

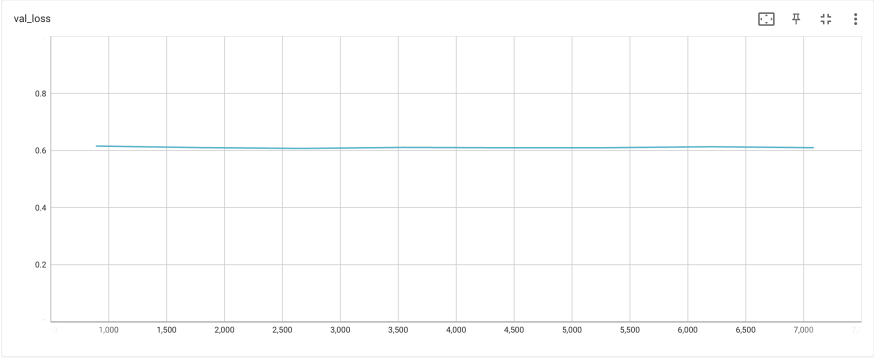

Figure 9: Validation Loss

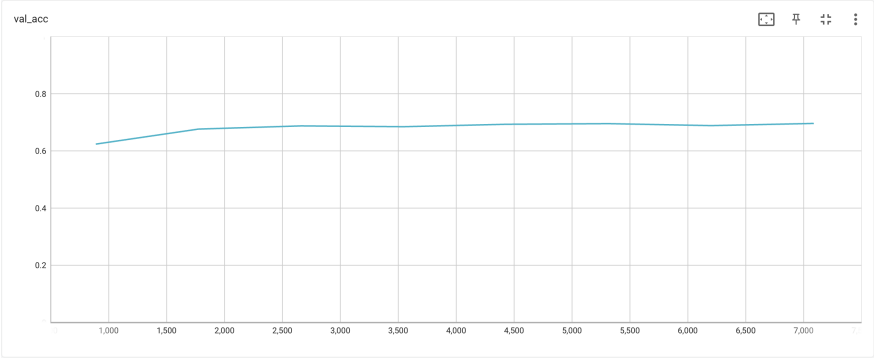

Figure 10: Validation Accuracy

2.5 Limitations

With this work, we perform some experiments and carefully tune our model training code to get to ≈ 0.71 test accuracy on the Education dataset to predict if a student gets an answer correct given the 'question id' and 'user id'. With our experimentation we demonstrate better results than all baselines using multiple modern training techniques. We observe this increase in test accuracy due to multiple changes to the starter code, primarily due to using: a well-crafted model for this task, a correct batch size, using AdamW [2] as the optimizer, and using a better learning rate schedule.

However, our model still has a few limitations. First, our model does not use any of the metadata for students or the subjects while being trained, currently, the model is being solely trained on 'question id' and 'user id'. Second, our model also does not use the subjects that these questions are from. These potentially have some useful information for predicting correctness as well since subjects are often related for example, computer science and math maybe be considered similar subjects when training a classifier. We believe our current approach could directly benefit from incorporating these subject-specific priors through a pre-trained text encoder. Moreover, with the current approach while we see an improvement throughout the metrics we care about, our model sacrifices on explainability over some of the baseline models. With or without the metadata, the data that we train on has very few features on which we train, since the dataset collects very few features, we expect that our model could be influenced by some confounding variables which are not captured in the dataset.

2.6 Running the Project

To run the project code for Part B, navigate to the `part_b` directory. Ensure that you have Python 3.9.x or above installed on your system. Then, execute the following command in your terminal to set up the environment:

```
pip install -r requirements.txt
```

Once the environment is set up, you can train the model using the `main.py` script with various options. For example:

```
python main.py --project education_model --model_type original --optimizer adamw \
--lr_scheduler none --lr 1e-1 --base_path ../data --epochs 8 --batch_size 64 \
--save_model model.pth --seed 3047
```

Replace the options with your desired configuration. For more details on available options and how to run specific configurations, refer to the README.md file in the project directory.

To view the training logs using TensorBoard, execute the following command:

```
tensorboard --logdir=logs
```

Access the TensorBoard server by visiting http://localhost:6006 in your browser.

To evaluate the trained models on the test set, you can use the following commands:

```
python main.py --model_type original --base_path ../../data --batch_size 3543 \
--epochs 0 --checkpoint_path model.pth

python main.py --model_type mhsa --base_path ../../data --batch_size 3543 \
--epochs 0 --checkpoint_path model.pth
```

Again, refer to the README.md file for more detailed instructions and additional options.

3 Member Contributions

Shivesh Prakash (Student No: 1008693790, UTorId: praka150) contributed to the following aspects of this project:

- Implemented, trained and reported the work on kNN models in Part A.
- Cleaned up the meta data to make it uniform, removing invalid entries, and interpolating empty data points as needed.
- Trained and tested multiple neural network-based models using the cleaned meta data, achieving a test accuracy of 62%. Note: This work was not submitted as part of the final submission, as only the best-performing model was included.
- Wrote the code and report for the 'Comparison with Baseline Models' section.
- Conducted rigorous testing of three models and their variants in part A, along with four commonly used ML models (Linear Regression, Bayesian Regression, Random Forest, XGBoost), using validation accuracy, test accuracy, TPR, TNR, FPR, FNR, and F1 scores. The code for these comparisons can be found in the `comparison` directory.

Law Kieu (Student No: 1006663914. UTorID: kieudat) contributed to the following aspects of this project:

- Implemented, trained and reported the work on Item Response Theory Model in Part A.
- Independent implementation and training of kNN and Neural Networks for cross-validation.
- Modified and tested the Autoencoder to cooperate the metadata.cThis wasn't submitted as part of the final submission since only the best-performing model was included.

- Wrote the Method section of Part B Report for our best modified model.
- Collaborated on finding the best hyperparameters for our reported model.

Rishit Dagli (Student No: 1008840685. UTorID: dagliris) contributed to the following aspects of this project:

- Implemented, trained, and reported the work on Neural Networks in Part A.
- Wrote all the training and testing code as well as built the architecture for the final models we report the performance of in this report.
- Wrote and ran code to perform hyperparameter tuning on the neural nets we built and reported in this report.
- Wrote the Limitations section of Part B Report.

Pranjal Rahul Agrawal (Student No: 1009526292. UTorID: agraw274) contributed to the following aspects of this project:

- Wrote code to cluster the questions based on the metadata using K-means. Trained and tested neural networks on this data. This wasn't submitted as part of the final submission since only the best-performing model was included.
- Wrote section 2.4 of the report (The Model's Performance).

References

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.

[2] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019.

[3] Chigozie Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. Activation functions: Comparison of trends in practice and research for deep learning, 2018.