CSC 311 Fall 2023 Project

Wo Ming Boaz Cheung, Ivan Ye, Zexi Liu

December 4, 2023

Part A

1 k-Nearest Neighbor

(a) see table and figure below

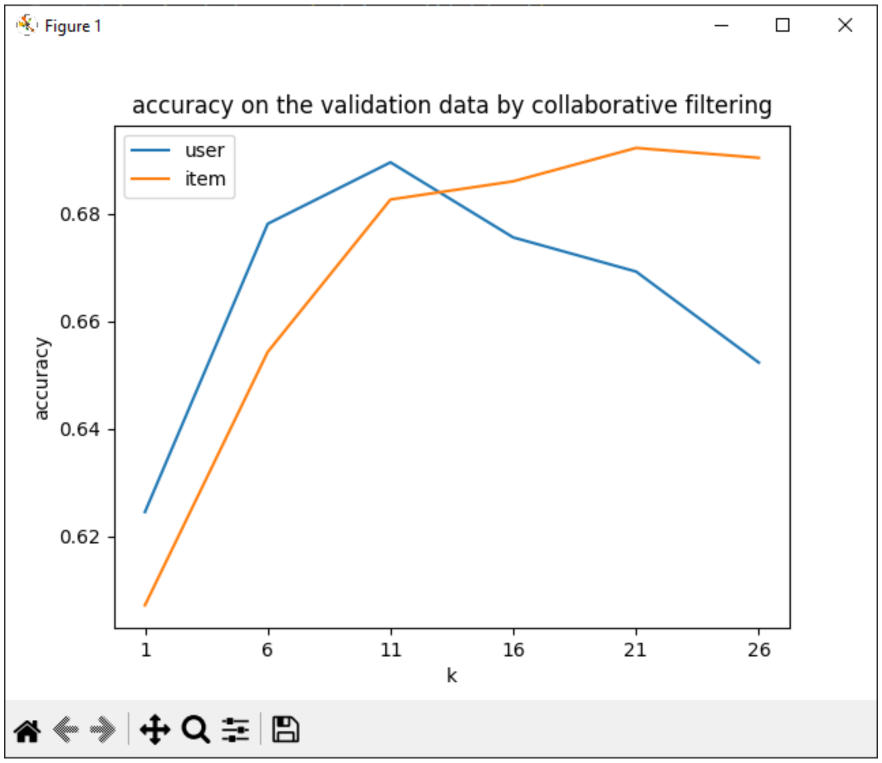

Figure 1: Accuracy of validation data as a function of k

k	knn_impute_by_user	knn_impute_by_item
1	0.6244707874682472	0.607112616426757
6	0.6780976573525261	0.6542478125882021
11	**0.6895286480383855**	0.6826136042901496
16	0.6755574372001129	0.6860005644933672
21	0.6692068868190799	**0.6922099915325995**
26	0.6522720858029918	0.69037538808919

(b) k* for user is 11 with final test accuracy of 0.6841659610499576

(c) see table and figure above
The underlying assumption on item-based collaborative filtering is that if question X is answered similarly by other students as question Y, X's difficulty for specific students matches that of question Y.
k* for item is 21 with final test accuracy of 0.6816257408975445

(d) knn_impute_by_user performs better than knn_impute_by_item by a slight amount based on the final test accuracies from the chosen k*s .

(e) A potential limitation of kNN for the task given is that the core underlying assumptions may not hold in the first place, where student A could have guessed and matched the same correct and incorrect answers as student B. Another potential limitations is the lack/sparsity of data. A quick glance at the train_data shows upwards of 1800 questions but about 600 students and 57000 entries, this limits the efficiency of kNN as 'close' neighbours could actually be far away.

2 Item Response Theory

$$\text{let } z = p(c_{ij} = 1 | \theta_i, \beta_j)$$

(a)

$$\begin{aligned}\ell &= \log\, p(C|\theta, \beta)\\ &= \log(z^{c_{ij}}(1-z)^{1-c_{ij}})\\ &= c_{ij}\log(z) + (1-c_{ij})\log(1-z)\end{aligned}$$

$$\begin{aligned}\frac{\partial \ell}{\partial \theta_i} &= \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial \theta_i}\\ &= (\frac{c_{ij}}{z} - \frac{1-cij}{1-z}) \cdot z(1-z)\\ &= c_{ij}(1-z) - z(1-c_{ij})\\ &= c_{ij} - c_{ij}z - z + c_{ij}z\\ &= c_{ij} - z\end{aligned}$$

$$\begin{aligned}\frac{\partial \ell}{\partial \beta_i} &= \frac{\partial \ell}{\partial z}\frac{\partial z}{\partial \beta_i}\\ &= (\frac{c_{ij}}{z} - \frac{1-cij}{1-z}) \cdot -z(1-z)\\ &= -c_{ij}(1-z) + z(1-c_{ij})\\ &= -c_{ij} + c_{ij}z + z - c_{ij}z\\ &= -c_{ij} + z\end{aligned}$$

(b) see figure below
hyperparameters: learning rate = 0.001, iterations = 50

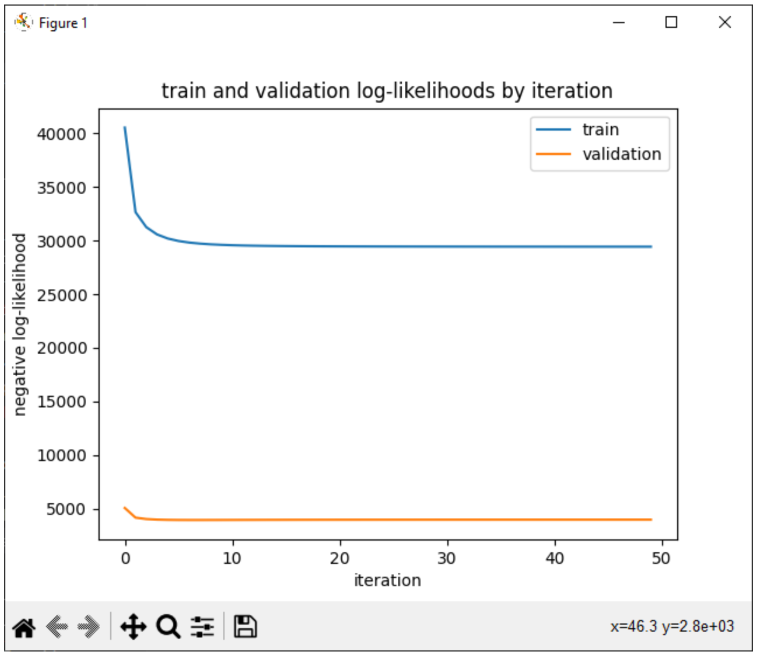

Figure 2: Training and validation log-likelihoods as a function of iteration

(c) Final validation accuracy: 0.7067456957380751
Test accuracy: 0.7084391758396839

(d) see figure below
The shape of the curves are in the form of the sigmoid function, as that is the probability of a student answering correctly, offset by the randomly chosen weights for θ and β. These curves represent the predicted probability that a student i with ability θ_i can answer questions j_1, j_2, j_3 correctly.

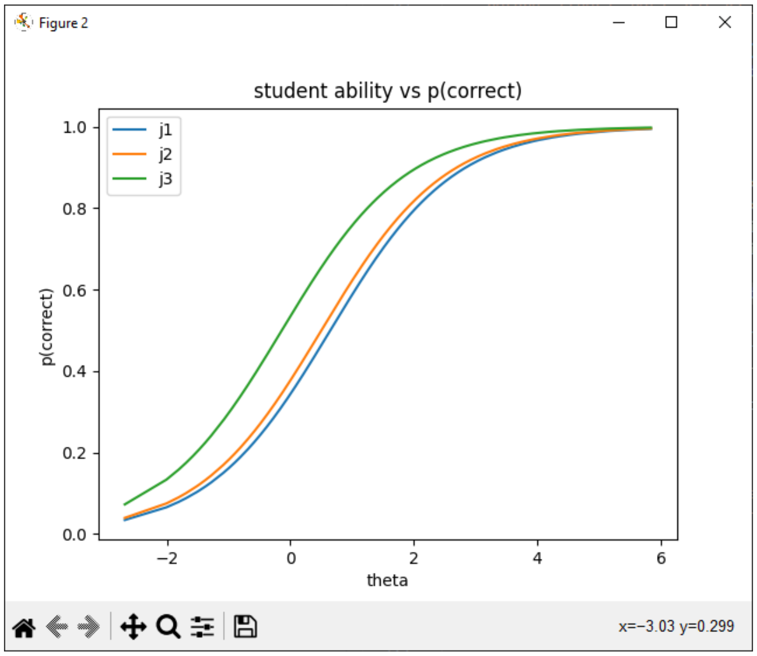

Figure 3: Training and validation log-likelihoods as a function of iteration

3 Neural Networks

(a) A key difference in Alternating Least Squares compared to neural networks is that there are no explicit labels since it's part of a type of principal component analysis which instead minimizes reconstruction error to find the pattern between data. Instead of minimizing error to the label.

Secondly, ALS tend to model simpler patterns with higher computational efficiency compared to neural networks which model complex patterns with multilayered neuron preceptors does depending on the scenario needed.

Building on the previous points, the final difference is that neural networks take in data that can have many different features to enable complex relationships in the model while ALS tend to only use the feature of interaction in the dimensionality reduction.

(b) Implemented

(c) $k^* = 100$
Best k value after tuning the optimization parameters to
num epoch = 25
learning rate = .01
Resulting in a validation accuracy of 68.44%

(d) Figure of graph with given k^*

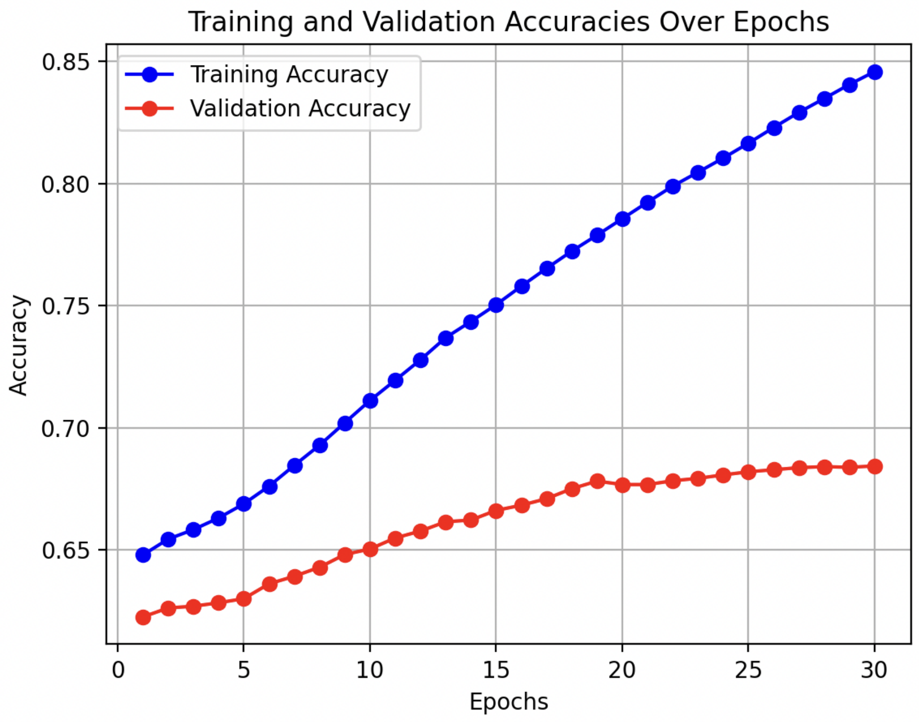

Figure 4: Accuracy of the training and validation data as function of epoch

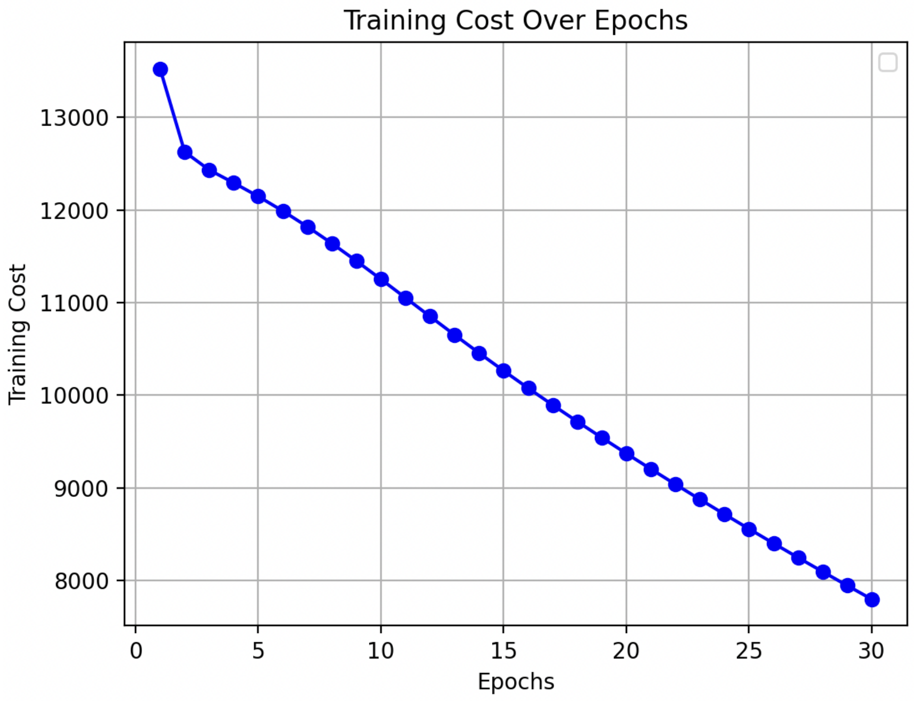

Figure 5: Training cost as function of epoch

Final test accuracy = 68.25%

(e) After trying all the regularization lambda = {.001, .01, .1, .1}
The best lambda is $\lambda = .001$
Resulting in the final validation = 68.71% and final test = 68.27%
After applying the regularization penalty the model preforms slightly better with the validation accuracy, and stays the same with the test accuracy. Thus overall there seems to be a slight benefit with using the regularization penalty.

4 Ensemble

Final validation accuracy: 0.7039232289020604
Test accuracy: 0.7109793959920971

The ensemble process that was implemented was done through first creating 3 datasets from the given train_data, each the same size as the original with replacement. The base model used was IRT, so there were 3 IRT models, each one trained on one of the bootstrapped training sets. Each IRT model then outputs its own θ and β which were each used to predict the correctness, using the given evaluate function, the average of which was taken.
Comparing it to the regular IRT model, the validation accuracy was slightly lower while the test accuracy was slightly higher. There are a few reasons as to why this was the outcome. For the validation accuracy, while not optimal, the 3 IRT models where trained using only 25 iterations as opposed to 50 iterations for the base model. Since validation accuracy usually increases as the number of iterations do, this was most likely what caused the lowered validation accuracy. As for the test accuracy however, the increase in accuracy can be credited to the random nature of sampling, where the training datasets could have been randomly skewed towards the test data.

Part B

Formal Description

The goal of this extension of the algorithm is to create a more complex model to better capture the results, since from question 3 about neural networks, it is seen that with the optimal hyper-parameters the benefit of regularization is minimal which could be a result from the model being under-fitted. Thus, to capture the complex relationships in the data set with more accuracy, the model is modified to have a deep neural network. Between the newly added hidden layers, the Leaky ReLU activation function is used for a simple non-linear activation function. The optimizer is also changed to the Adaptive Moment Estimation optimizer for a more robust optimizer that has an adaptive learning rate and momentum which enables the model to better deal with noisy gradients present in the data set. To compensate for the more computation power needed for this complex model, the modified model will also improve the optimization of the training process in the neural network by using the CUDA model to take advantage of the parallel computing architecture in a GPU.

Function of leaky ReLU define α to be a small value to capture behavior of negative values

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ \alpha x, & \text{otherwise} \end{cases}$$

Formula for Adaptive Moment Estimation optimizer

$$\begin{aligned}
t &: \text{Time step} \\
\theta_t &: \text{Parameters being optimized (model weights).} \\
g_t &: \text{Gradient of the loss with respect to the parameters at time step } t. \\
m_t &: \text{First moment estimate (mean) of the gradients.} \\
v_t &: \text{Second moment estimate (uncentered variance) of the gradients.} \\
\hat{m}_t &: \text{Bias-corrected first moment estimate.} \\
\hat{v}_t &: \text{Bias-corrected second moment estimate.} \\
\alpha &: \text{Learning rate, determining the size of the step taken during optimization.} \\
\beta_1 &: \text{Exponential decay rate for the first moment estimate } (m_t) \\
\beta_2 &: \text{Exponential decay rate for the second moment estimate} \\
\epsilon &: \text{Small constant added to the denominator to avoid division by zero.}
\end{aligned}$$

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \tag{1}$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \tag{2}$$

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{3}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{4}$$

$$\theta_{t+1} = \theta_t - \alpha \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{5}$$

Figure/Diagram

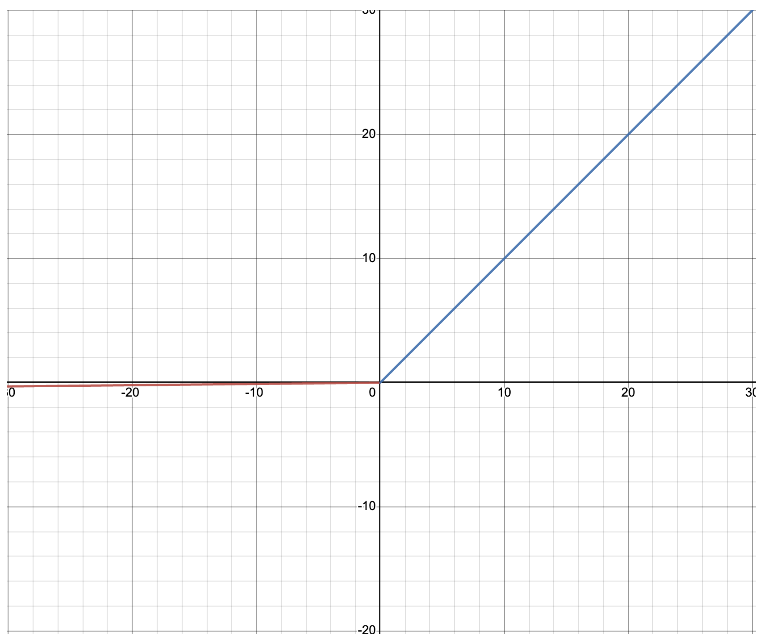

Figure 6: function graph of leaky ReLU

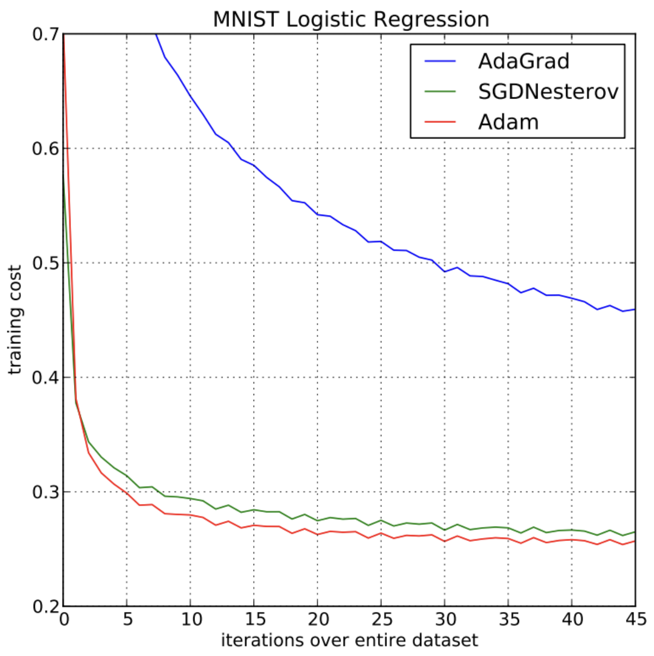

Figure 7: Training cost with Adam

This is a figure that shows the benefit of optimizing with Adam with the MNiST data set.

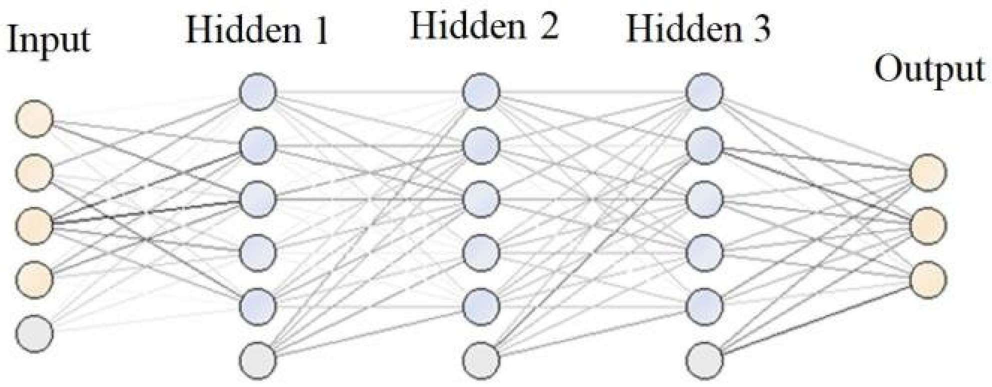

Figure 8: Neural network model with 3 hidden layers

Comparison/Demonstration

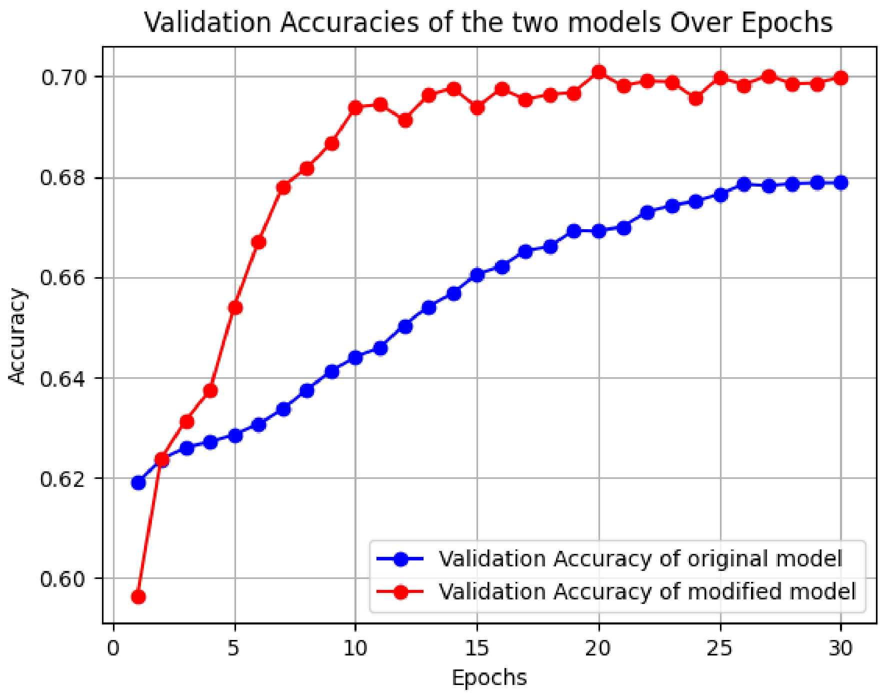

Figure 9: Validation accuracy comparison between the original neural network model and the modified one

Test accuracy of original model 67.51%
Test accuracy of modified model 70.25%

As seen from the increased model accuracy and faster accuracy grow with the new model, it can be concluded that the modified model is better at modeling and predicted the data.

Based on my argument of optimization, a way to test will be to analyse the time needed to run the two models with the same GPU and CPU without calculating validation.
This experiment is conducted on a GPU of a Nvidia RTX A2000 and CPU of a Intel xeon w-1390p @ 3.50ghz
Running time of original model : 13.82 seconds

Running time of modified model with CUDA : 23.16 seconds
Running time of modified model without CDUA : 28.50 seconds

To separate the benefits of adding depth to the neural network and using ADAM optimizer, each separate modification would be tested against the base model. The hypothesis is that each modification increases the validation accuracy of the model compared to the base model.

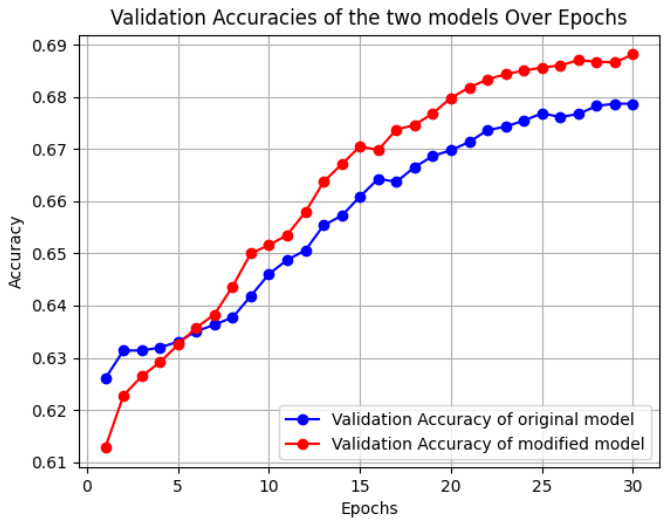

Figure 10: Validation accuracy comparison between the original neural network model and ADAM optimizer modified

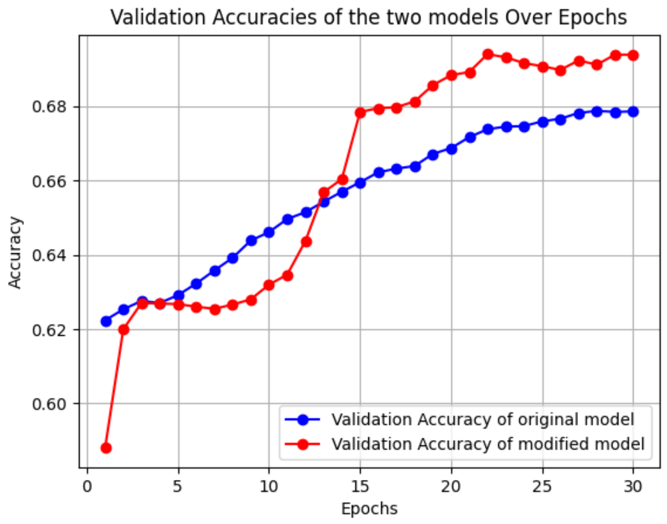

Figure 11: Validation accuracy comparison between the original neural network model and deeper neural network modifed

As seen from both graphs, each modification provides an increase to the accuracy of the model prediction. Thus, a conclusion could be drawn to the hypothesis that both modifications are better than the base model. While both are still lower then being combined together, which concludes the most optimal way is to implement both of the techniques of increased depth and ADAM optimizer.

Limitations

The main limitation of my model is a small sample size, since the model is trying to fit the data into a more complex data. This approach could be prone to over fitting.

A way to address this problem would either sample more data, or alternatively to employ methods of bagging to simulate more data with probabilistic taking data from the given subset to lower variance.

Another common method to address overfitting, that is already implemented, is a regularizer. With the one that is currently used being a L2 regularizer

Another limitation with my model is the numerous of hyper-parameters that needs to be set, including the alpha , k, learning rate, num epoch, and lambda. If they are not tuned appropriately, it would lead to a model being unable to converge or have a sub-optimal solution. While this issue is present in most models with hyper parameters, this implementation has especially many.

By increasing the depth of the neural network model, the model is highly sensitive to small changes to the data. Resulting in a vulnerability with adversity attacks with modifications of the data set. Especially for this use case where the data is information from student's past experience, which an attacker could pretend to be. Adversarial attacks will gain the gradient information for this neural network model and create examples that maximize the error resulting in the model making bad predictions. While all models could also be vulnerable to attack, this sensitive neural network is especially prone.

While this is an ongoing issue of research, currently a method to address this issue is to regularize gradients as well to prevent large changes in the loss function. Thus, a small subset in the data set wouldn't be able to shift the model in a significant amount. In addition, just by incorporating randomness into the training process would also make the adversary example harder to model and create.

References

Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980.

Individual Contribution

All members of the team have submitted the course evaluations

(i) **Wo Ming Boaz Cheung**
Part B: Formal Description, Figure/Diagram, Comparison/Demonstration

(ii) **Zexi Liu**
Part A: Neural Networks
Part B: Limitations

(iii) **Ivan Ye**
Part A: kNN, Item Response Theory, Ensemble