University of Toronto

Final Project

Manit Gosalia

CSC311: Introduction to Machine Learning

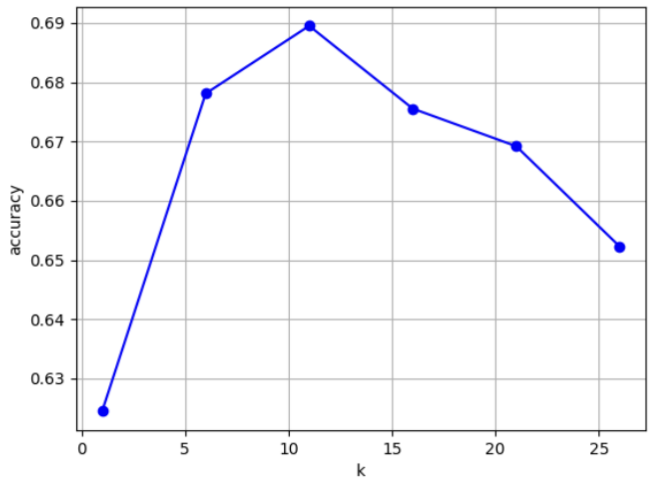
Sayyed Nezhadi

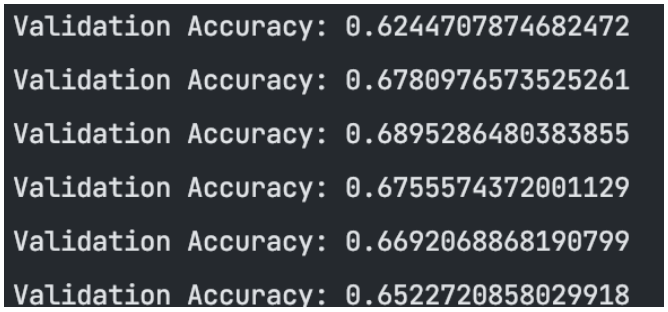
17th August 2023

Part A

1. k-Nearest Neighbour

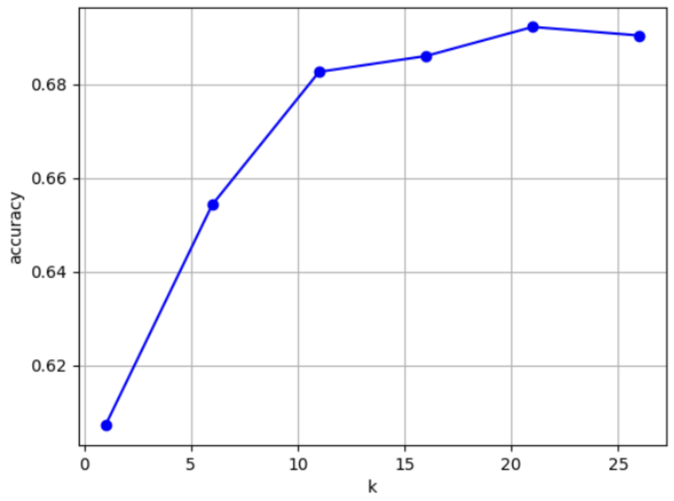
a. As you can see from the graph the highest validation accuracy is for k = 11.

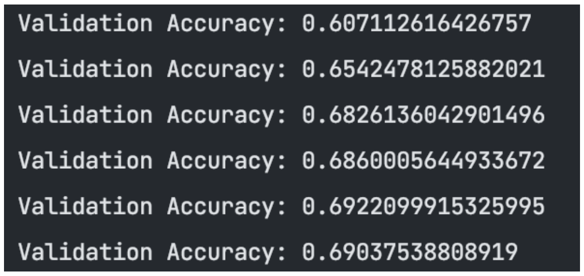



```
Validation Accuracy: 0.6244707874682472
Validation Accuracy: 0.6780976573525261
Validation Accuracy: 0.6895286480383855
Validation Accuracy: 0.6755574372001129
Validation Accuracy: 0.6692068868190799
Validation Accuracy: 0.6522720858029918
```

b. k* = 11 and the final test accuracy on k* is 68.417%

c. The underlying assumption is that if the users (in this case students) have agreed in the past on certain items (questions), they will likely agree again in the future. **k* = 21 and the final test accuracy on k* is 68.163%.**



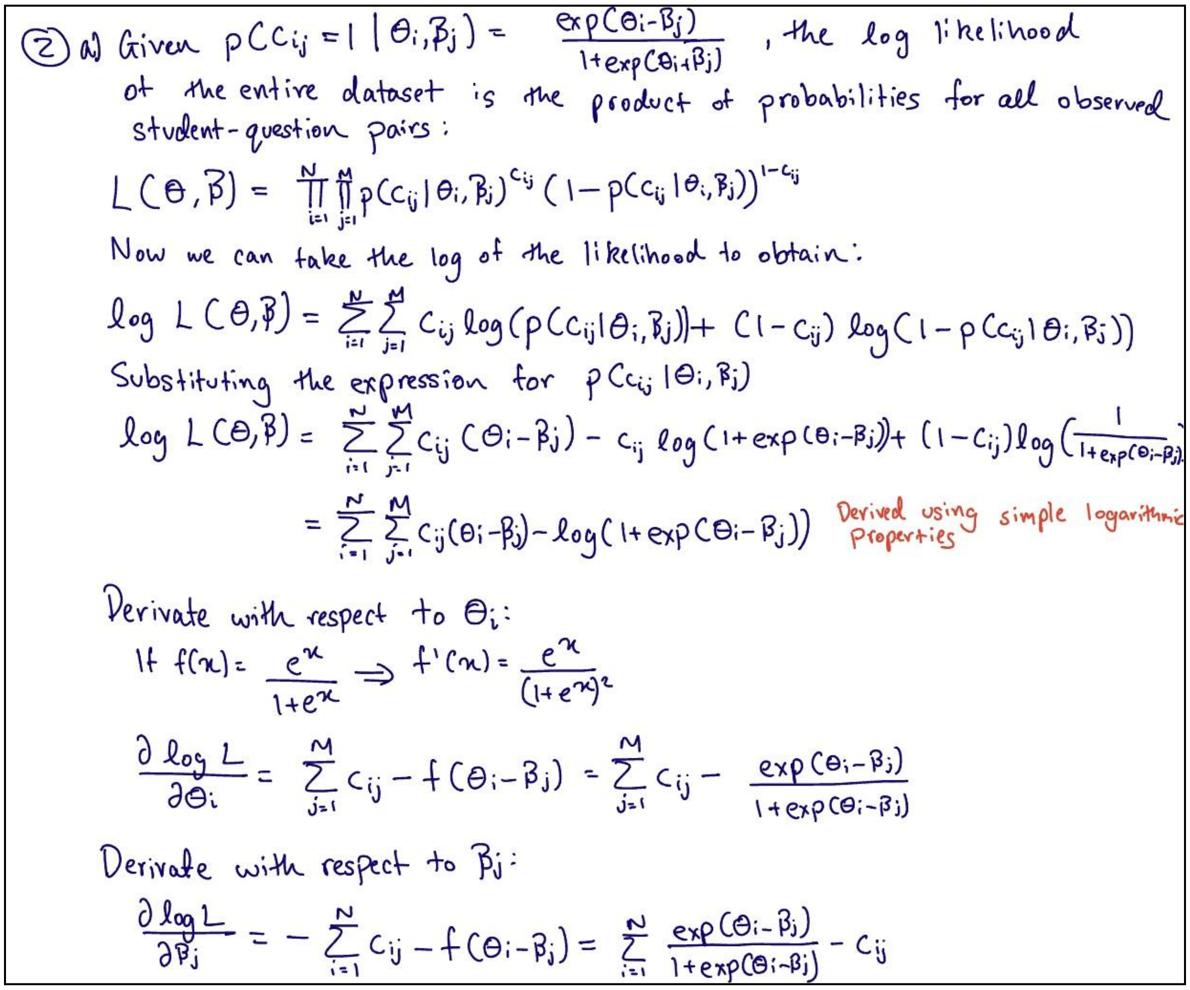

```
Validation Accuracy: 0.607112616426757
Validation Accuracy: 0.6542478125882021
Validation Accuracy: 0.6826136042901496
Validation Accuracy: 0.6860005644933672
Validation Accuracy: 0.6922099915325995
Validation Accuracy: 0.69037538808919
```

d. The test accuracy for user-based collaborative filtering is slightly better as seen from their optimal k's (k*) test accuracy, hence it performs better.

e. Limitations:
   i. kNN requires calculating the distance between the target user and all other users in the dataset, which can be <u>computationally expensive</u>
   ii. With high dimensional data approximately all points are equidistant due to the curse of dimensionality

2. Item Response Theory

a. The solution can be seen in the screenshot below:



② a) Given $p(c_{ij}=1 \mid \theta_i, \beta_j) = \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i+\beta_j)}$, the log likelihood of the entire dataset is the product of probabilities for all observed student-question pairs:

$$L(\theta,\beta) = \prod_{i=1}^{N}\prod_{j=1}^{M} p(c_{ij}\mid\theta_i,\beta_j)^{c_{ij}}\left(1-p(c_{ij}\mid\theta_i,\beta_j)\right)^{1-c_{ij}}$$

Now we can take the log of the likelihood to obtain:

$$\log L(\theta,\beta) = \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}\log(p(c_{ij}\mid\theta_i,\beta_j)) + (1-c_{ij})\log(1-p(c_{ij}\mid\theta_i,\beta_j))$$

Substituting the expression for $p(c_{ij}\mid\theta_i,\beta_j)$

$$\log L(\theta,\beta) = \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}(\theta_i-\beta_j) - c_{ij}\log(1+\exp(\theta_i-\beta_j)) + (1-c_{ij})\log\left(\frac{1}{1+\exp(\theta_i-\beta_j)}\right.$$

$$= \sum_{i=1}^{N}\sum_{j=1}^{M} c_{ij}(\theta_i-\beta_j) - \log(1+\exp(\theta_i-\beta_j))$$ Derived using simple logarithmi properties

Derivate with respect to θ_i:

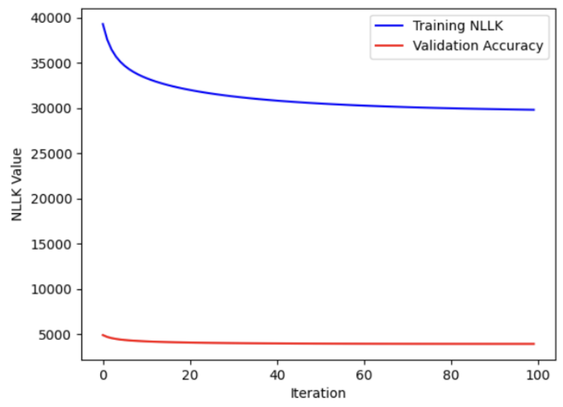
If $f(x) = \frac{e^x}{1+e^x} \Rightarrow f'(x) = \frac{e^x}{(1+e^x)^2}$

$$\frac{\partial \log L}{\partial \theta_i} = \sum_{j=1}^{M} c_{ij} - f(\theta_i-\beta_j) = \sum_{j=1}^{M} c_{ij} - \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)}$$

Derivate with respect to β_j:

$$\frac{\partial \log L}{\partial \beta_j} = -\sum_{i=1}^{N} c_{ij} - f(\theta_i-\beta_j) = \sum_{i=1}^{N} \frac{\exp(\theta_i-\beta_j)}{1+\exp(\theta_i-\beta_j)} - c_{ij}$$

b. The hyperparameters I selected: <u>Learning Rate=0.003</u> and <u>Iterations=100</u>

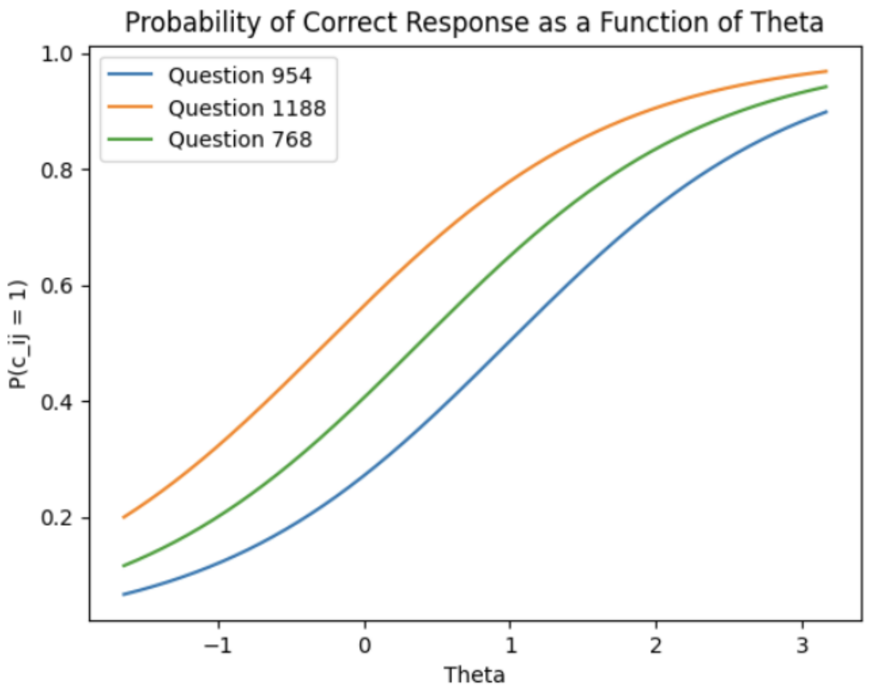


c. Validation Accuracy: 70.66%
   Test Accuracy: 70.53%

   All answers can be cross-checked by running the code in item_response.py

d. The shape of the curve represents a logistic function. The shape is S-shaped or sigmoidal. **As θ increases the probability of a correct answer increases**. All 3 curves show this trend between θ and probability and naturally so, there is a positive correlation.

   The questions are chosen randomly



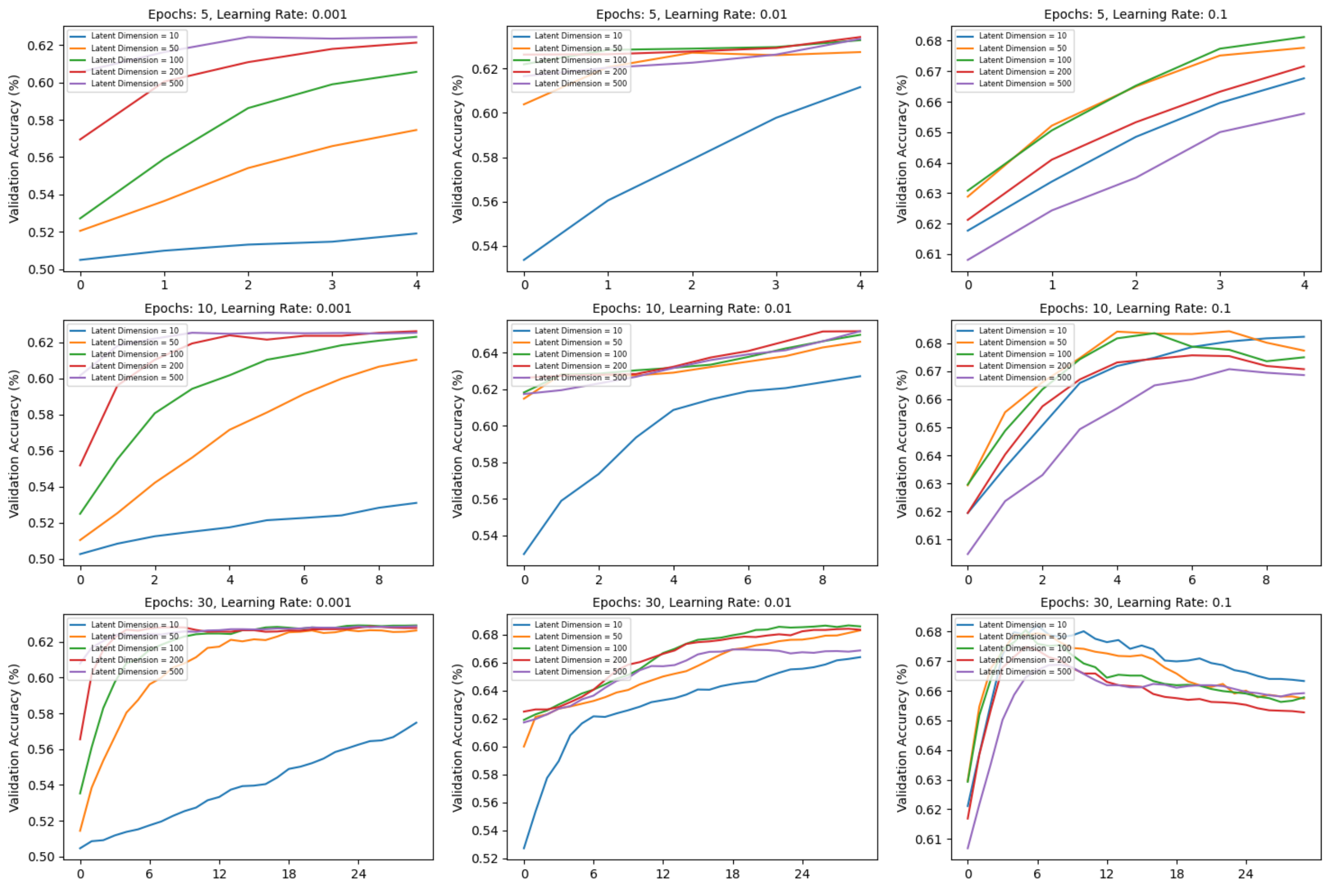
3. Neural Networks

a.

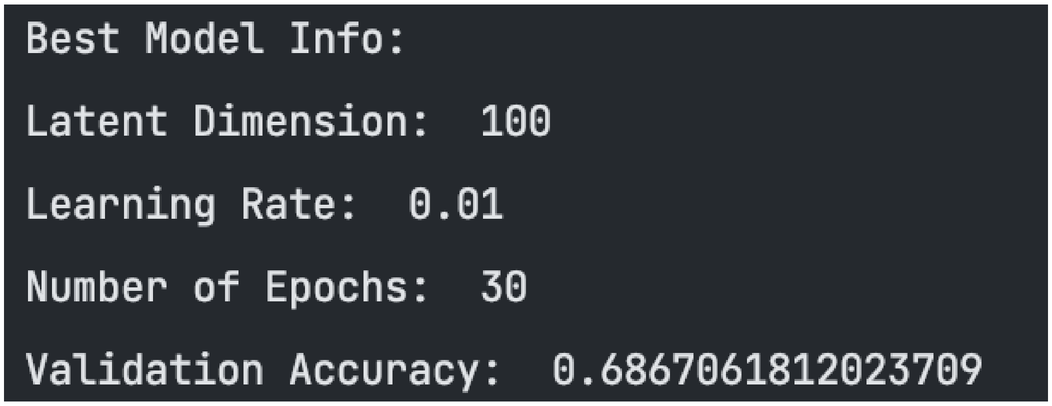
	ALS	**Neural Network**
Model Complexity and Flexibility	ALS is a specific optimization algorithm mainly used in matrix factorization problems, like collaborative filtering in recommendation systems. It alternately fixes the user and item factors to solve a least-squares problem. It assumes a linear structure between the latent factors.	NNs are more flexible and can model complex non-linear relationships. They consist of layers of nodes or "neurons" with activation functions that can capture deep patterns within the data. In the provided example, an autoencoder network is being used, with a non-linear sigmoid activation function, allowing the network to learn more intricate patterns.
Objective Function and Optimization	The objective of ALS is to minimize the mean squared error between the observed ratings and the predictions, given by matrix multiplication of the user and item latent factors. ALS does this by alternately fixing and optimizing the factors.	The objective function in the provided neural network example is the reconstruction error of an autoencoder, which aims to minimize the difference between the input and its reconstruction. The optimization is typically performed using stochastic gradient descent or other gradient-based optimization techniques.
Interpretability and Transparency	The model parameters in ALS (i.e., the latent factors for users and items) may be more interpretable, as they represent specific latent features within the data. This can make it easier to understand what the model has learned and how it is making predictions.	NNs are often considered "black-box" models, where the learned weights and biases might not have a clear or direct interpretation. This can make it more challenging to understand exactly what the network has learned about the data or to validate its behaviour.
Type of Learning	ALS, being a matrix factorization technique, is typically used in an unsupervised learning context. It works by discovering latent factors that explain the observed user-item interactions. The model does not need any explicit labels or target outcomes to learn these factors.	On the other hand, traditional neural networks are often used in supervised learning tasks. In these tasks, each instance in the training dataset has an associated target output value (or label), such as the category of an object in an image or the sentiment of a piece of text. The network is trained to predict these labels from the input data. However, it's also worth noting that there are variants of neural networks designed for unsupervised learning, such as autoencoders, and similarly, ALS could be used in a semi-supervised context if some form of user or item meta-data is available and used in the learning process. But the point about typical use cases remains valid.

b. Implemented in neural_network.py

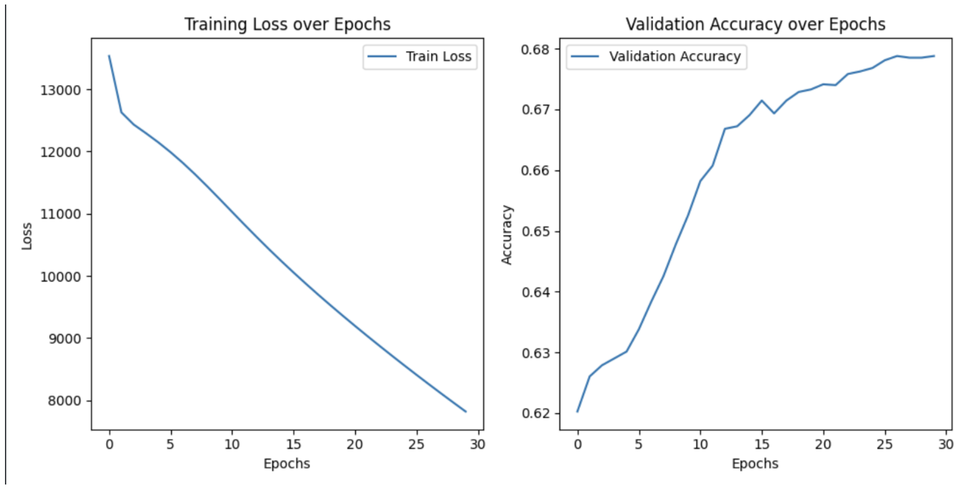
c. Below are plots showing the validation accuracy for latent dimension, $k \in \{10, 50, 100, 200, 500\}$ with hyperparameters $epochs \in \{5, 10, 30\}$ and $learning\ rate \in \{0.001, 0.01, 0.1\}$.

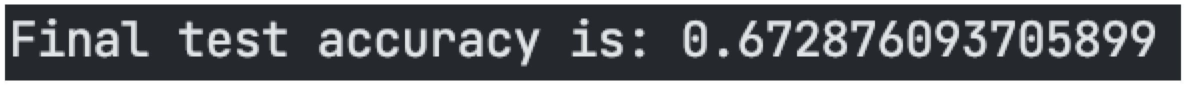


I obtained k* = 100 which has the highest validation accuracy ≈ 68.87% with learning rate = 0.01 and epochs = 30


```
Best Model Info:

Latent Dimension:  100

Learning Rate:  0.01

Number of Epochs:  30

Validation Accuracy:  0.6867061812023709
```

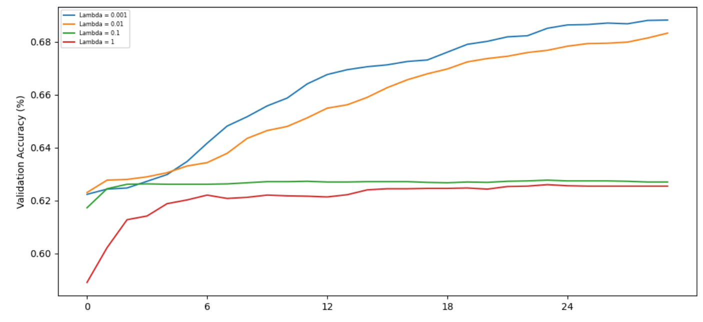
d. For my chosen k* = 100, as seen from the plots below, the validation accuracy seems to increase, however, they almost plateau after 25 epochs. The training loss is consistently decreasing as epochs increase.

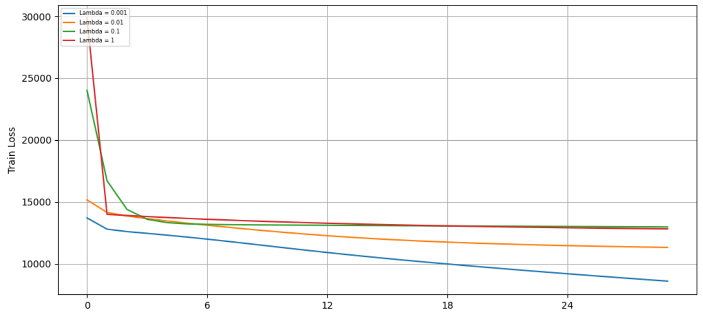




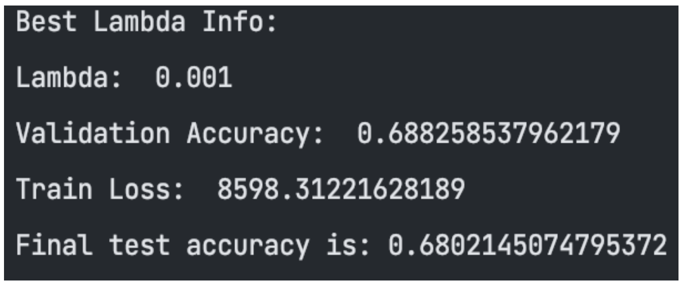
```
Final test accuracy is: 0.672876093705899
```

e. Below from the plots you can see the validation accuracy and training loss as a function of epochs for different values of the regularization penalty $\lambda \in \{0.001, 0.01, 0.1, 1\}$.
The chosen k is the same as part d) and the hyperparameters are the same as shown in the last image on the previous page showing "best model info".





Below I have reported the information for my chosen λ, which is optimal as seen by the above graphs.



```
Best Lambda Info:

Lambda:  0.001

Validation Accuracy:  0.688258537962179

Train Loss:  8598.31221628189

Final test accuracy is: 0.6802145074795372
```

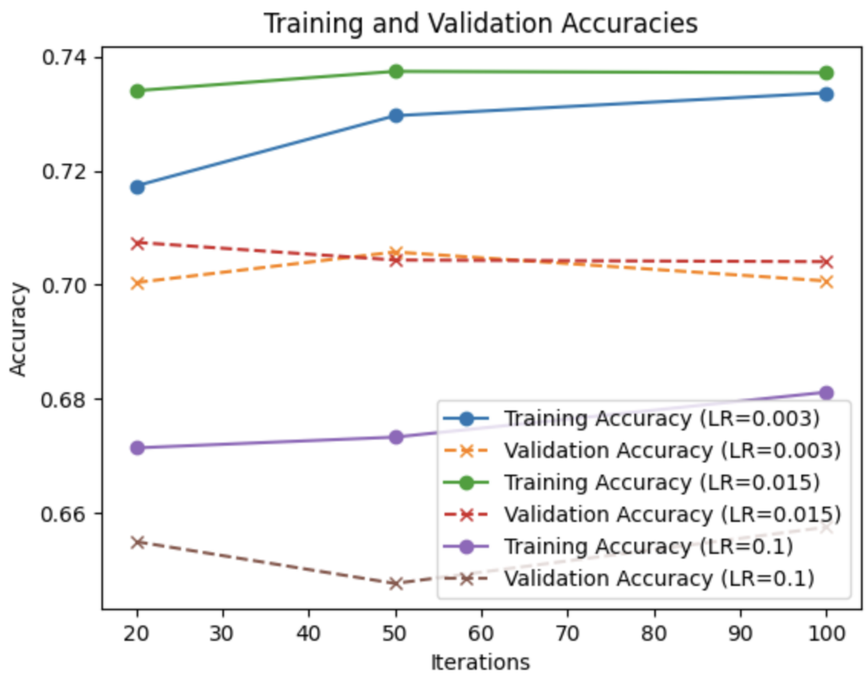
It can be seen for my optimal k* and the chosen hyperparameters, the model taking into account L2 regularization performs slightly better. This can be seen by the testing accuracy (68.02% > 67.29%). However, the training loss is higher when taking into account the model with the regularization penalty.

4. Ensemble

The Ensemble Process I Implemented:

- For the bagging ensemble I have chosen to use the Item Response Theory model from Part A Q2.
- **Bootstrap Sampling**: `by_student` is False; the method samples random individual data points, regardless of the student. It creates a dataset where some student-question pairs might be repeated while others are left out.
- **Training Multiple Models**: For a specified number of models (`num_models` = 3), the process creates a new bootstrap sample of the training data and trains an IRT model on this sample.
- **Averaging Predictions**: After all models have been trained:
    - For each data point in a dataset (training, validation, or test), the method calculates the predictions of all trained models using their respective theta and beta parameters
    - These predictions are averaged. Essentially, each model gets an equal vote in the final decision.
    - The averaged prediction is thresholded (using 0.5 as the threshold) to give a binary outcome – either the student answered the question correctly (1) or not (0)
- **Assessing the Ensemble**: The accuracy of the ensemble predictions is evaluated against the actual outcomes in the training, validation, and test datasets. The accuracy represents the fraction of predictions that match the true outcomes.

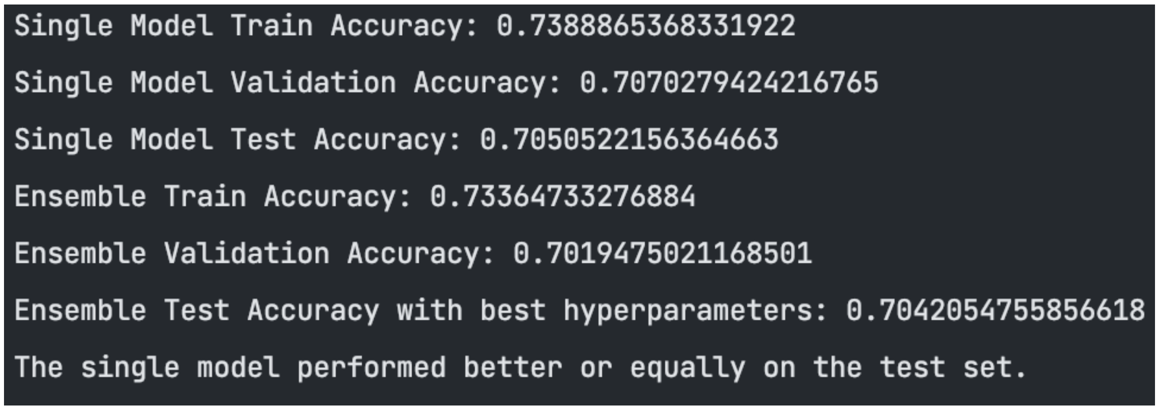
HyperParameters Tuning Results



HyperParameters Chosen:

Learning Rate: 0.015

Number of Iterations: 20



```
Single Model Train Accuracy: 0.7388865368331922
Single Model Validation Accuracy: 0.7070279424216765
Single Model Test Accuracy: 0.7050522156364663
Ensemble Train Accuracy: 0.73364733276884
Ensemble Validation Accuracy: 0.7019475021168501
Ensemble Test Accuracy with best hyperparameters: 0.7042054755856618
The single model performed better or equally on the test set.
```

Analysing and Reporting the performance

As seen from the above image the final validation accuracy is approximately **70.19%** and the final test accuracy is approximately **70.42%.**

It can also be seen that the ensemble model performed slightly worse than a single base model.

It's important to note that the validation and test accuracy are awfully similar to the training accuracy for both the ensemble and single model.

Why is the ensemble model performing worse or almost the same?

- **Consistent Data Distribution:** The performance metrics across training, validation, and test datasets were relatively close. This consistency indicates that our original model was already generalizing well, which means there might be limited room for improvement through ensembling.
- **Nature of Base Model**: The base model using Item-Response Theory (IRT) might inherently not be prone to overfitting. Given that one of the main advantages of ensembling is to counteract overfitting, the benefits of an ensemble become less pronounced when the base model already generalizes well.
- **Limitations of Ensemble Techniques**: It's important to understand that while ensemble methods can enhance performance in many cases, they are not a silver bullet. Their effectiveness relies on the premise that pooling together multiple "weak" or "diverse" models can lead to a stronger combined model. However, if individual models (like ours) are already performing near their capacity, the ensemble might not lead to substantial improvements.

Part B

Formal Description

Original Algorithm: Neural Network AutoEncoder Model (Part A Q3)

The original algorithm is based on a NN Autoencoder architecture. The Autoencoder comprises an encoder network and a decoder network. The encoder network takes the set of Students vector, **v**, as input and maps it to a lower-dimensional latent space representation. The decoder network reconstructs the input student vector, **v**, from the latent space. Both the encoder and decoder consist of linear transformations followed by **sigmoid activation functions**, g and h. The objective is to predict the students' correctness on a diagnostic question.

Limitation of Original Algorithm

While the original algorithm presents a promising approach, it has inherent limitations that hinder its predictive performance:

- Overfitting: The model's capacity, as determined by the depth and complexity of the architecture, can lead to overfitting. The Autoencoder may learn noise present in the training data, resulting in poor generalization to unseen examples. The training loss decreases at an extremely quick rate, while the validation accuracy plateaus. This is a strong indication of overfitting.

Proposed Modification

To address the limitations of the original algorithm, I propose a multifaceted modification that aims to improve generalization, optimization, and provide a measure of uncertainty in predictions. The following enhancements are suggested:

- **Regularization Using Dropout**: I introduce dropout regularization to both the encoder and decoder networks. During training, dropout stochastically deactivates neurons, preventing the network from relying excessively on specific neurons and thereby reducing overfitting.
- **Leaky ReLU Activation**: I replace the sigmoid activation functions with leaky rectified linear unit activations. Leaky ReLU helps in learning non-linear representations leading to more stable and faster optimization.
- **Increased Depth**: I extend the depth of both the encoder and decoder networks by adding extra layers. The increased depth allows the network to capture more intricate relationships in the data and potentially learn richer representations.
- **AdaGrad Optimatization**: Instead of using Stochastic Gradient Descent, I adopt the AdaGrad optimization algorithm. AdaGrad adapts the learning rate for each parameter based on its historical gradient information. This adaptive learning rate can help dealing with optimization challenges more effectively, especially when dealing with sparse data.
- **Monte Carlo Dropout**: In addition to dropout during training, I apply Monte Carlo Dropout during prediction. By performing multiple forward passes with dropout activated, we obtain a distribution of predictions. This distribution provides an estimate of prediction uncertainty, enabling a better understanding of model confidence and reliability.

Expected Performance Improvements

I anticipate that the combined effect of these modifications will result in the following benefits:

- **Regularization**: Dropout and Monte Carlo Dropout will mitigate overfitting, yielding improved generalization to unseen data. The uncertainty estimates from Monte Carlo Dropout will enable us to identify instances where the model's prediction is uncertain.
- **Optimization Enhancement**: The use of AdaGrad optimization, in conjunction with Leaky ReLU activations, will help mitigate optimization difficulties, leading to faster convergence and more stable training.
- **Increased Representational Capacity**: Additional layers and improved optimization will enhance the model's ability to capture intricate patterns in the data, leading to more expressive and accurate representations.
- **Uncertainty Quantification:** Monte Carlo Dropout will provide insights into the model's prediction uncertainty, which is valuable information for decision-making and identifying cases where further assessment is necessary.

Original AutoEncoder Reconstruction Function

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

Here, h and g are sigmoid activation functions with $b^{(1)}$ and $b^{(2)}$ being the bias terms. $\mathbf{W}^{(2)}$ and $\mathbf{W}^{(1)}$ are the weight matrix. Finally $\mathbf{v}$ is the input user vector.

Modified Model – DeepAutoEncoder

Encoder Modifications:

The encoder now includes a leaky ReLU activation function and dropout regularization. The dropout operation can be represented as a mask applied to the output of the activation function:

$$\mathbf{z} = LeakyReLU(\mathbf{W}^{(1)}\mathbf{v} + b^{(1)}) \odot DropoutMask$$

Where:

- z is the latent representation
- $\odot$represents element-wise multiplication
- $DropoutMask$ is the mask generated during dropout

Decoder Modifications:

Similar to the encoder, the decoder also incorporates leaky ReLU activation and dropout regularization:

$$\mathbf{v}' = LeakyReLU(\mathbf{W}^{(2)}\mathbf{z} + b^{(2)}) \odot DropoutMask'$$

Where:

- $\mathbf{v}'$ is the reconstructed input vector
- $DropoutMask'$ is the mask applied to the output of the decoder's activation

Monte Carlo Dropout and Uncertainty Estimation

Monte Carlo Dropout can be applied during both encoding and decoding to estimate model uncertainty. The output can be averaged over multiple dropout runs:

$$\mathbf{v}'_{MC} = \frac{1}{T}\sum_{t=1}^{T} \mathbf{v}'_{T}$$

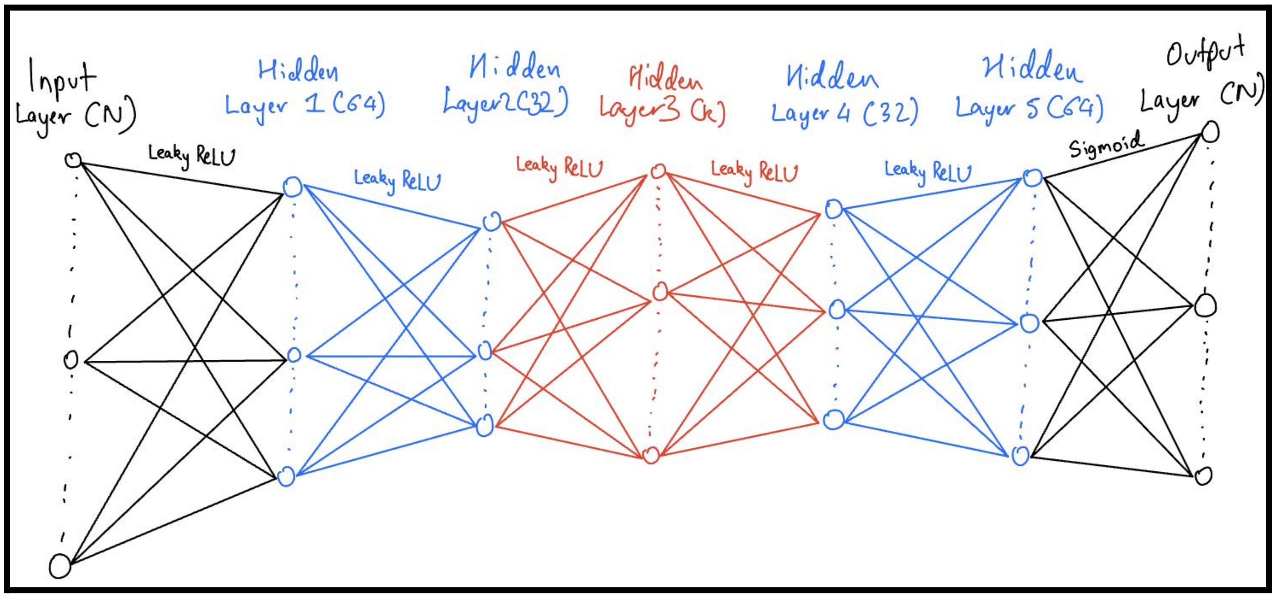
Where:

- $\mathbf{v}'_{MC}$ is the Monte Carlo Estimate for the reconstructed input vector

Note: Since my model has multiple hidden layers you need to iterate each encoder layer 3 times and each decoder 3 times making the function for the model something like this where the last activation function h is a sigmoid activation:

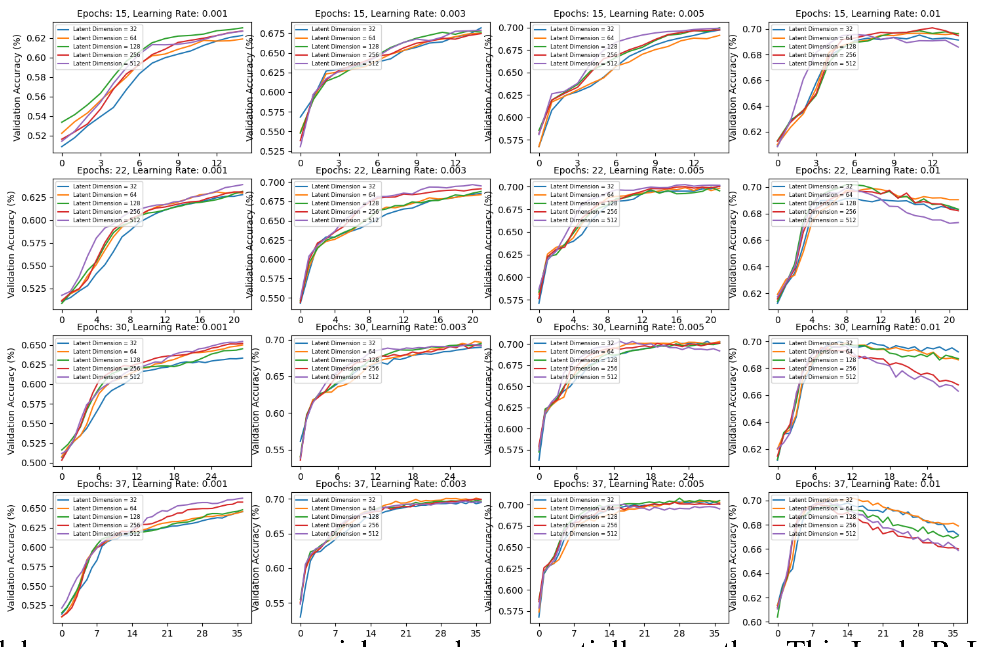
$$f(\mathbf{v};\boldsymbol{\theta}) = h(\boldsymbol{W}^{(6)} LeakyReLU(\boldsymbol{W}^{(5)} LeakyReLU(\boldsymbol{W}^{(4)} LeakyReLU(\ldots) + b^{(4)}) \odot DropoutMask'''' + b^{(5)}) \odot DropoutMask''''' + b^{(6)}) \odot DropoutMask''''''$$

Figure or Diagram to better understand Model

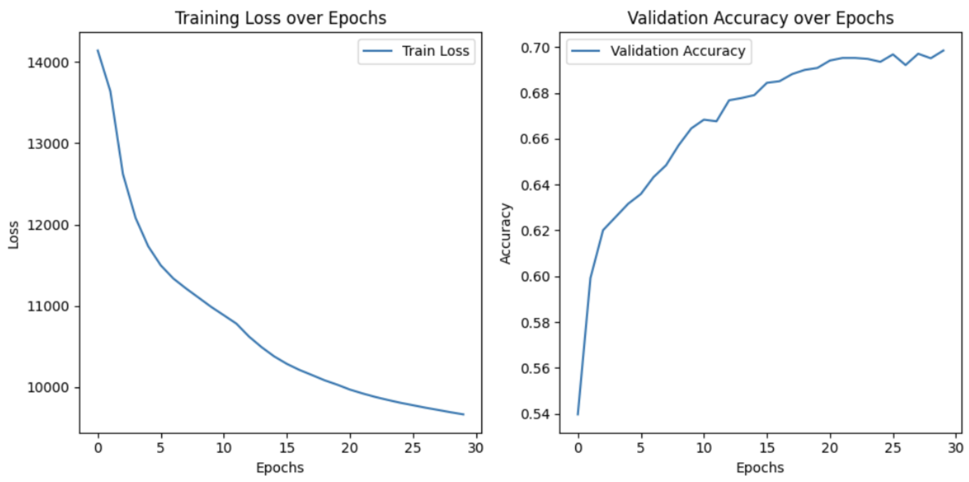


Comparison and Demonstration

First I tune the hyperparameters for my model. The results are seen in the plots below,similar to the plot used for Part A Q3c. As seen from the image below the most optimal hyper parameters I get are for k=128, learning rate of 0.003 and 30 epochs.



The model on average coverges quicker, and exponentially smoother. This LeakyReLU activation is causing a much more stable optimizations. This is one of the major improvements from the base model. Below we can see the plot for the training loss and validation accuracy using the optimal hyperparameters, using the new model.



Compared to the base model, the training loss decreases exponentially slower. While the validation accuracy is converging quicker, while its smoother and more stable. This shows a sign of reducing overfitting. By using Monte Carlo dropouts as well as dropout layers in the more complex model, I have tackled overfitting as the training loss doesn't decrease rapidly, while the validation accuracy is better. Below is the table showing the Test Accuracy comparison between both models using optimal hyper parameters.

	Test Accuracy	Test Accuracy (λ = 0.002)
Base Model (AutoEncoder)	67.288%	67.739%
New Model (DeepAutoEncoder)	69.038%	70.23%

As you can see from the table the New Model performs better on training data.

Experiment

I believe there was an overfitting problem with the original model, hence I hypothesize by increasing the probability of dropout layers and adding more samples in the Monte Carlo Dropout the performance if the model will increase.

I set k=128, learning rate=0.003 and epochs=30 and λ=0.0

I conduct an experiment where I vary Monte Carlo Num Samples and Dropot Probabilty. The table on the next page shows the testing accuracy for all variations.

Dropout Probability	0.0	0.25	0.5	0.75
Monte Carlo Samples				
5	69.03%	68.42%	69.18%	68.02%
10	69.04%	68.78%	**69.28%**	67.85%
20	69.26%	**69.91%**	**69.42%**	68.64%

As seen from the table the best training accuracies are for when we have 20 monte carlo samples. These help reduce overfitting as Monte Carlo Dropout introduces randomness into the network during training. This prevents the network from relying too heavily on specific neurons or features, making the model more adaptable and less likely to memorize the training data. Similarly a probability of around 0.5 for the dropout layer helps maintain overfiiting. Dropout serves as a form of regularization. By introducing noise and randomness, it implicitly adds a regularization term to the loss function. This discourages the network from fitting the training data too closely and encourages it to learn more generalized features.

With λ=0.002 here are the results:

Dropout Probability	0.0	0.25	0.5	0.75
Monte Carlo Samples				
5	69.26%	69.15%	69.26%	69.12%
10	69.55%	69.24%	69.52%	68.13%
20	**69.59%**	**70.15%**	**70.31%**	68.95%

As seen form the above table its made clear that high number of MC samples is allowing the model to reduce overfitting. However, there is no significant evidence regarding Dropout probability.

Limitations

- Limited Generalization to Unseen Patterns: Autoencoders tend to learn patterns present in the training data. If the data contains a narrow distribution of patterns, the model might struggle to generalize to unseen or unexpected patterns. This is a common occurrence in the training data as I don't use any meta data in the training model.
- Complexity vs. Interpretability Trade-off: Deep autoencoders with multiple layers can become complex, leading to reduced interpretability. As the number of hidden layers increases, it becomes harder to understand how the model makes predictions.
- Optimization Challenges: Deeper networks can be harder to optimize. The vanishing gradient problem might affect the training of deep autoencoders, leading to slow convergence or poor solutions.
- Limited Data: If the available data is extremely limited, the deep autoencoder might struggle to learn meaningful representations. Deep models require a significant amount of data to generalize well. In such cases, overfitting could be a major concern. This is quite a common occurrence for our model as for those categories where there isn't enough data, overfitting is a concern.

Some possible Extensions to the DeepAutoEncoder:

- Data Augmentation: Augmenting the training data can help mitigate the limitations of limited data. Techniques like rotation, scaling, and cropping can artificially increase the dataset size. Can also make use of meta data to mitigate for limited data.
- Hybrid Models: Combining the strengths of different models, such as using deep autoencoders for feature extraction and simpler models like IRT for prediction, can provide a balanced solution.