CSC311 Project Report

Ewan Jordan, Kevin Chen, Marc Grigoriu, Xinyue Xu

April 2024

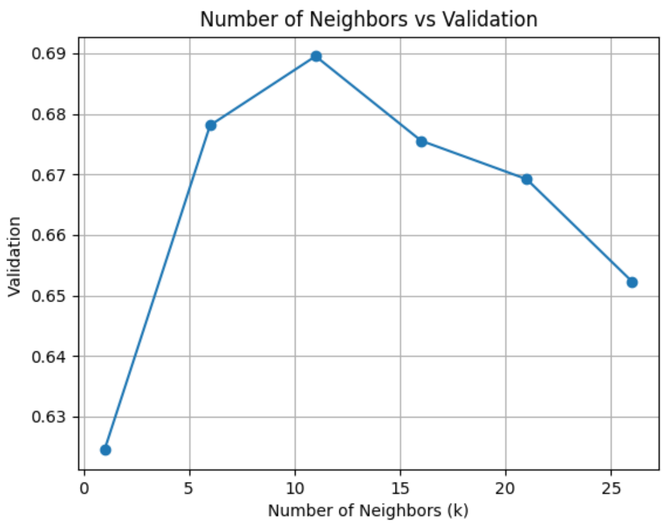
Contributions :
Part A questions 1 and 2 were worked on by Kevin. Part A question 3 was worked on by Marc. Everyone helped come up with model modifications for part B, however Ewan and Caroline took on most of the work with the modifications and question responses.

1 Part A

1.1 Collaborative filtering with k-Nearest Neighbor

1.1.1 (a)

Test Accuracy: 0.6841659610499576
k: 11

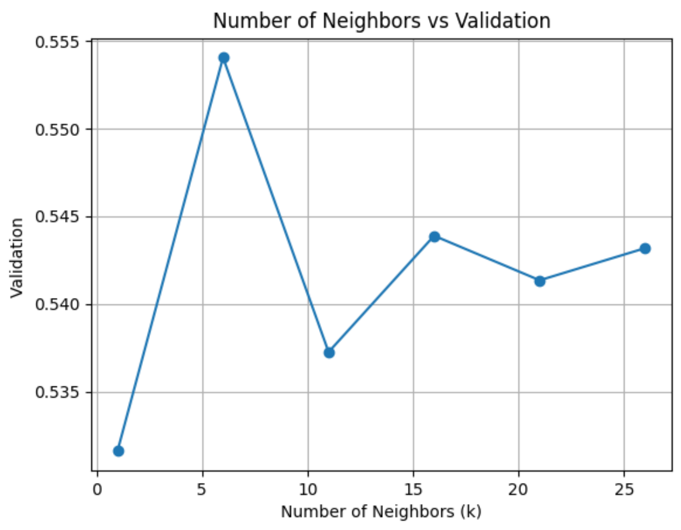


1.1.2 (b)

The underlying assumption is that if other students consistently assign the same scores to both Question A and Question B, then both questions are considered equally correct or incorrect for those particular students.

1.1.3 (c)

Test Accuracy: 0.545582839401637
k: 6



1.1.4 (d)

Item Based Filtering seems better for our use case.

Since user-based filtering is dependent on a students history, we need a student to answer a number of questions before we have reliable data, while item-based filtering just relies on other items.

A student may become better overtime, in which case, user-based filtering fails to consider a students improvement when training. On the other hand, questions don't change in difficulty overtime. We want to recommend harder questions if a student gets better and not penalize them for past incorrectness.

1.1.5 (e)

The effectiveness of KNN diminishes notably when dealing with sparse data. This is particularly evident in scenarios where the training set contains numerous missing values that require imputation. The presence of extensive empty spaces leads to limited overlap between data points, causing the nearest neighbors used for prediction to potentially be distant from the target point. For example,in the user-based filtering model, given we are trying to predict an answer for a student, if there are no students with closely matching diagnostic answers, the nearest neighbors may not accurately capture the local data structure.

KNN can be slow due to the need to compute distances between each and every data point, especially in this scenario, since we expect to have sparse matrices and a large number of missing values. Since we want to scale our application to get data from more students and increase the number of questions students can do, the large data set will increase computational cost.

1.2 The Item Response Theory Model

1.2.1 (a)

We have,

$$p(C_{ij} = 0|\theta_i, \beta_j) = 1 - \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}$$

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

$$p(C_{ij}|\theta_i, \beta_j) = p(C_{ij} = 1|\theta_i, \beta_j)^{C_{ij}} p(C_{ij} = 0|\theta_i, \beta_j)^{1-C_{ij}}$$

Then we derive the likelihood function.

$$p(C|\theta, \beta) = \prod_{i=0}\prod_{j=0} p(C_{ij}|\theta_i, \beta_j)$$

Now, we can get the log likelihood.

$$log(p(C|\theta, \beta)) = \sum_{i=0}\sum_{j=0} log(p(C_{ij}|\theta_i, \beta_j))$$

$$= \sum_{i=0}\sum_{j=0} C_{ij} log(\frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)}) + (1 - C_{ij}) log(1 - \frac{exp(\theta_i - \beta_j)}{1 + exp(\theta_i - \beta_j)})$$

$$= \sum_{i=0}\sum_{j=0} C_{ij}(\theta_i - \beta_j) - log(1 + exp(\theta_i - \beta_j))$$

Next, we derive the derivative of log likelihood with respect to θ_i and β_j.

$$\frac{\partial p(C|\theta_i, \beta)}{\partial \theta_{i'}} = \frac{\partial \sum_{i=0}\sum_{j=0} C_{ij}(\theta_i - \beta_j) - log(1 + exp(\theta_i - \beta_j))}{\partial \theta_i}$$

$$= \frac{\partial \sum_{j=0} C_{i'j}(\theta_{i'} - \beta_j) - log(1 + exp(\theta_{i'} - \beta_j))}{\partial \theta_{i'}}$$

$$= \sum_{j=0} C_{i'j} - \frac{exp(\theta_{i'} - \beta_j)}{1 + exp(\theta_{i'} - \beta_j)}$$

$$\frac{\partial p(C|\theta, \beta_j)}{\partial \beta_{j'}} = \frac{\partial \sum_{i=0}\sum_{j=0} C_{ij}(\theta_i - \beta_j) - log(1 + exp(\theta_i - \beta_j))}{\partial \theta_i}$$

$$= \frac{\partial \sum_{i=0} C_{ij'}(\theta_i - \beta_{j'}) - log(1 + exp(\theta_i - \beta_{j'}))}{\partial \beta_{j'}}$$

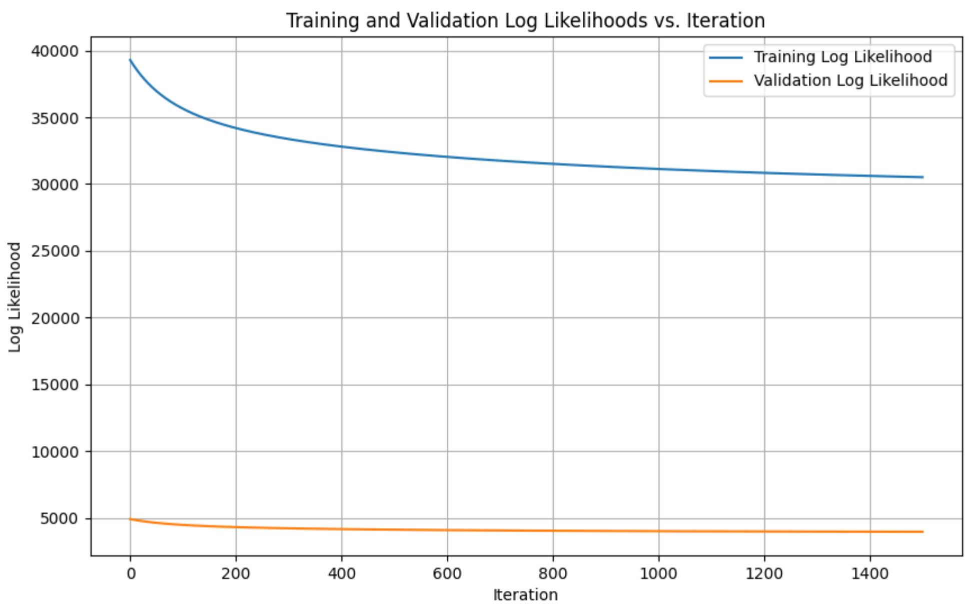
$$= \sum_{i=0} \frac{exp(\theta_i - \beta_{j'})}{1 + exp(\theta_i - \beta_{j'})} - C_{ij'}$$

1.2.2 (b)

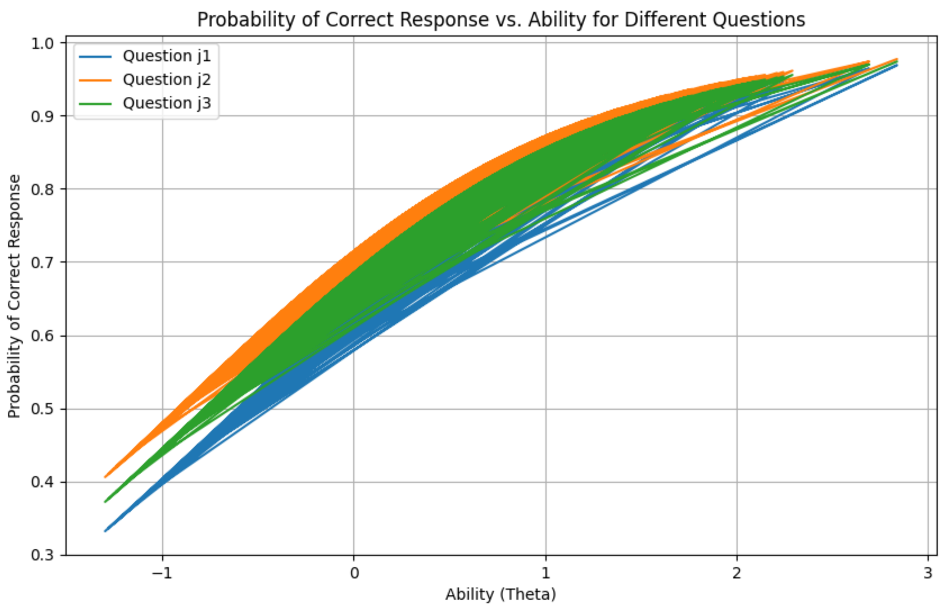
Code is included in zip.

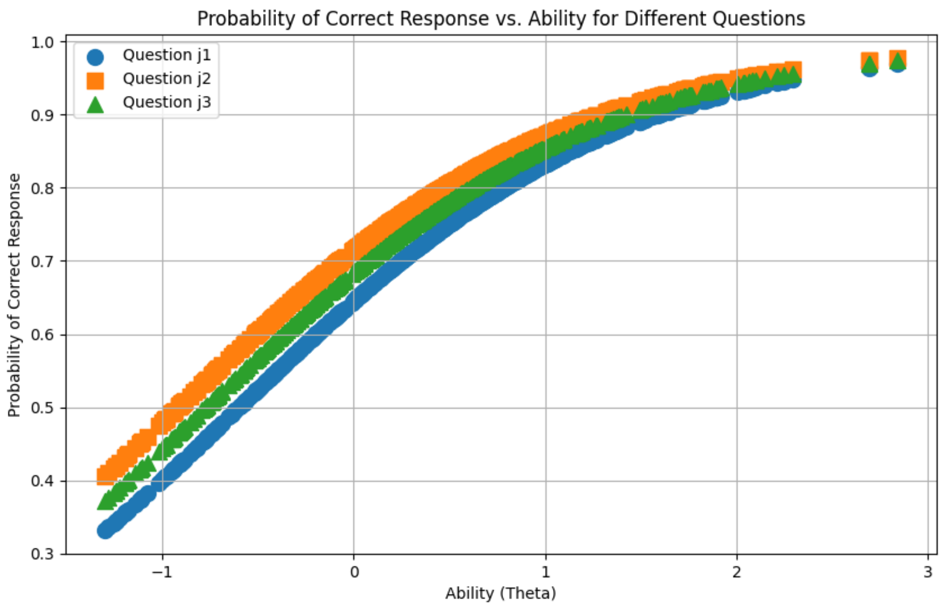
1.2.3 (c)

Learning rate: 0.0001
Iterations: 1500
Validation accuracy: 0.7075924357888794
Test accuracy: 0.7025119954840531



1.2.4 (d)





Questions used:
j1: 1525
j2: 1574
j3: 1030

Theta represents the ability of the student. It should be noted that as the ability of a student increases, they become more capable of solving a given question. This relationship is depicted in the graph. The curve illustrates that as Theta increases, the probability of a student solving question j1, j2, or j3 also increases. It is worth noting the separation between the curves: the orange curve has a higher probability of being solved than the green and blue curves. Therefore, we can infer that orange questions are easier than green questions, and green questions are easier than blue questions.

1.3 Neural Networks

1.3.1 (a)

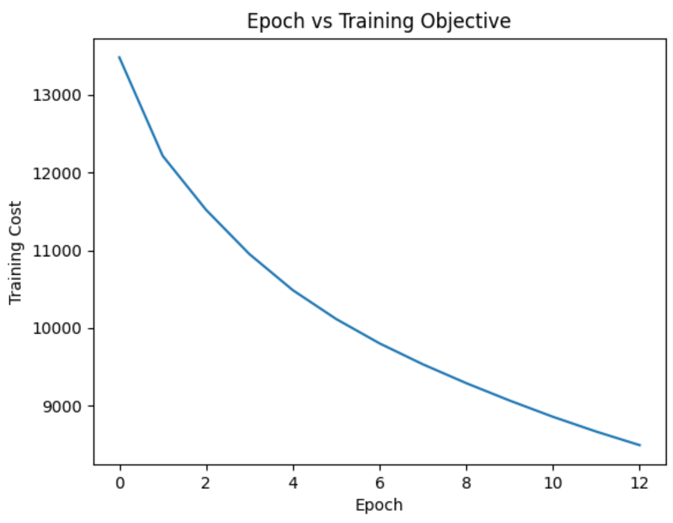
Ok!

1.3.2 (b)

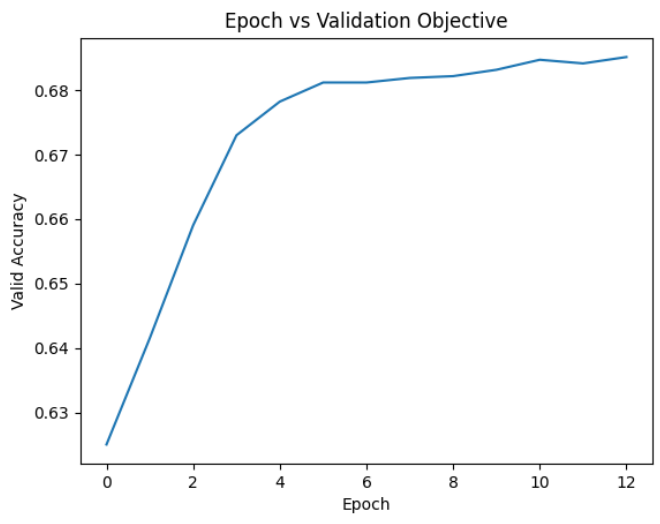
Final tuned hyperparameter values:
$k = 10$
$lr = 0.1$
$num\_epoch = 13$

1.3.3 (c)

Training objective as a function of epoch:



Validation objective as a function of epoch:



Test Accuracy: 0.6802145074795372

1.3.4 (d)

Final tuned lambda value:
$lamb = 0.001$

Validation accuracy for final model: 0.6841659610499576
Test accuracy for final model: 0.6852949477843635

Using a regularization penalty with our chosen λ, our model performs slightly better as the test accuracy increases by approximately 0.005.

2 Part B

2.1 Formal Description of Algorithm

Our team chose to modify the autoencoder used in Part A, question 3. We made several additions to the original, all which either centered around optimization improvements or attempts at reducing overfitting. We shall describe each modification in depth separately, and then the complete model visually.

2.1.1 Training based on Questions

The original model used as input the 584 student entities, each a tensor of length 1774 (each feature representing the students correctness on a question). For our modified model, we decided to reverse this and have our autoencoder take in the 1774 questions as input, each being a tensor of length 584 (each feature representing the correctness of a student's answer on this question). With this modification, the model's definition became, Given a question $q \in \mathbb{R}^N$, f is a reconstruction of the input q, where

$$f(q, \theta) = h(W^{(2)} g(W^{(1)} q + b^{(1)}) + b^{(2)}) \in \mathbb{R}^{N_{\text{input}}}$$

Here, $W^{(2)} \in \mathbb{R}^{N_{\text{users}} \times k}$ and $W^{(1)} \in \mathbb{R}^{k \times N_{\text{users}}}$. g and h are still sigmoid functions.

This change was motivated by the fact that we now have a higher number of data points, each with a lower number of features. We believe this could lead to a more accurate autoencoder, as there are less patterns to learn and more training data, which could theoretically strengthen the abilities of the network. We still tested out the rest of our modifications on both users and questions as entities. Proceeding, we use N_{input} to abstractly represent N_{users} and $N_{\text{questions}}$

2.1.2 Additional Layers

We have added additional layers in both the encoding and decoding process of the collaborative filtering. We have given these extra hidden layers a size of $2k$ after doing some testing, where k is the size of our latent dimension. Let σ denote the sigmoid function. This modification gives a new forward pass equation of,

$$L(x) = \sigma(H^{(1)}(\sigma(W^{(1)} x + b^{(1)}) + a^{(1)}) \in \mathbb{R}^k$$

$$f(x; \theta) = \sigma(W^{(2)} \sigma(H^{(2)} L(x) + a^{(2)}) + b^{(2)}) \in \mathbb{R}^N$$

Where x is our input, either $\in \mathbb{R}^{N_{\text{users}}}$ or $\in \mathbb{R}^{N_{\text{questions}}}$, $N = |x|$, and k is the size of the latent dimension. $W^{(2)} \in \mathbb{R}^{N_{\text{input}} \times 2k}$, $W^{(1)} \in \mathbb{R}^{2k \times N_{\text{input}}}$, $H^{(2)} \in \mathbb{R}^{2k \times k}$ and $H^{(2)} \in \mathbb{R}^{k \times 2k}$.

Deepening the network takes the same motivation as switching to question as entity; trying to improve the optimization. Adding layers in a neural network can improve performance because deeper neural networks can learn more complex, non-linear functions.

2.1.3 Dropout

Dropout is a regularization technique in neural networks aimed at preventing overfitting during training. It works by randomly removing neurons with a certain probability in each training iteration, forcing the network to learn more robust features independently and reducing dependency among neurons. This process effectively creates multiple sub-networks during training, leading to an ensemble effect that improves generalization. During evaluation, dropout is typically turned off, but the weights of connections are scaled to maintain consistency. This scaling is necessary because on a certain percentage of weights will be involved in each iteration of training, and thus will end up being much larger to account for the overall lack of latent input.

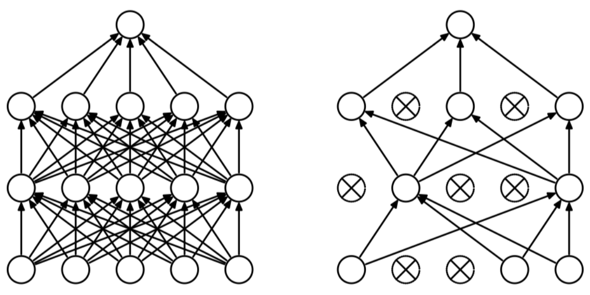


Figure 1: Dropout — Srivastava et al. (2014)

Figure 1 is what dropout looks like visually for a basic multilayer perceptron. We applied dropout to our autoencoder in each layer, in hopes to reduce overfitting and improve generalization.

2.1.4 Dense Re-feeding

One difficulty in training an autoencoder to perform collaborative filtering is the fact that training data is inherently sparse; Some data points will be missing values for certain features. However, we can take advantage of the fact that autoencoders output a vector of the same shape as the input, and use that output vector as a training data point of its own. As the model becomes more accurate these output vectors offer new opportunities for the model to consolidate its weights given vectors it has never seen before.

2.1.5 Meta-data Injection

We took advantage of the meta data included in the data folder, in hopes at improving optimization. We believe there is useful relationships between the metadata and the behaviour of students on the questions, which hopefully can be picked up on by the network. We applied different techniques for inclusion of metadata in the case of training with students as entities and questions as entities.
We decided to have a stacked encoding layer of the network process meta data, then concatenate the latent representation of meta data with the latent representation of the original input vector x (again, either $x \in \mathbb{R}^{N_{\text{input}}}$ or $x \in \mathbb{R}^{N_{\text{questions}}}$). This latent concatenation was then fed into the decoder to produce our output vector. Additionally, in the case of question meta data, we preprocess the data to be a onehot encoding of the question's subject list; $X_{meta} \in \mathbb{I}^{N_{\text{questions}} \times N_{subjects}}$. The mathematical representation for question as entity is as follows,

$$E_{meta} : \mathbb{I}^{N_{\text{subjects}}} \to \mathbb{R}^{k_{meta}}$$

$$E_{meta}(x_{meta}) = T^{(1)} x_{meta} + t^{(1)}$$

$T^{(1)} \in \mathbb{R}^{k_{\text{meta}} \times N_{\text{subjects}}}$

$$E_{\text{question}}(q) : \text{Latent representation of question}$$

$$f(q) = \text{Decode}(\text{Concatenate}(E_{meta}(x_{meta}), E_{\text{question}}(q)))$$

Where Decode is the decoder network, as defined within the Additional Layers section.

2.1.6 IRT Beta Injection

When performing IRT collaborative filtering, we learn latent beta values for each question, relating to the supposed difficulty of each question. If this pretrained value is fed to the network for each question, it could theoretically be used to gain more information about the expected correctness of students' answers on the question. The value is fed into the network in the latent layer in the same way at the metadata, except now there is no additional network to encode the value.

2.2 Model Diagram

The final autoencoder we came up with incorporates all the modifications listed above.

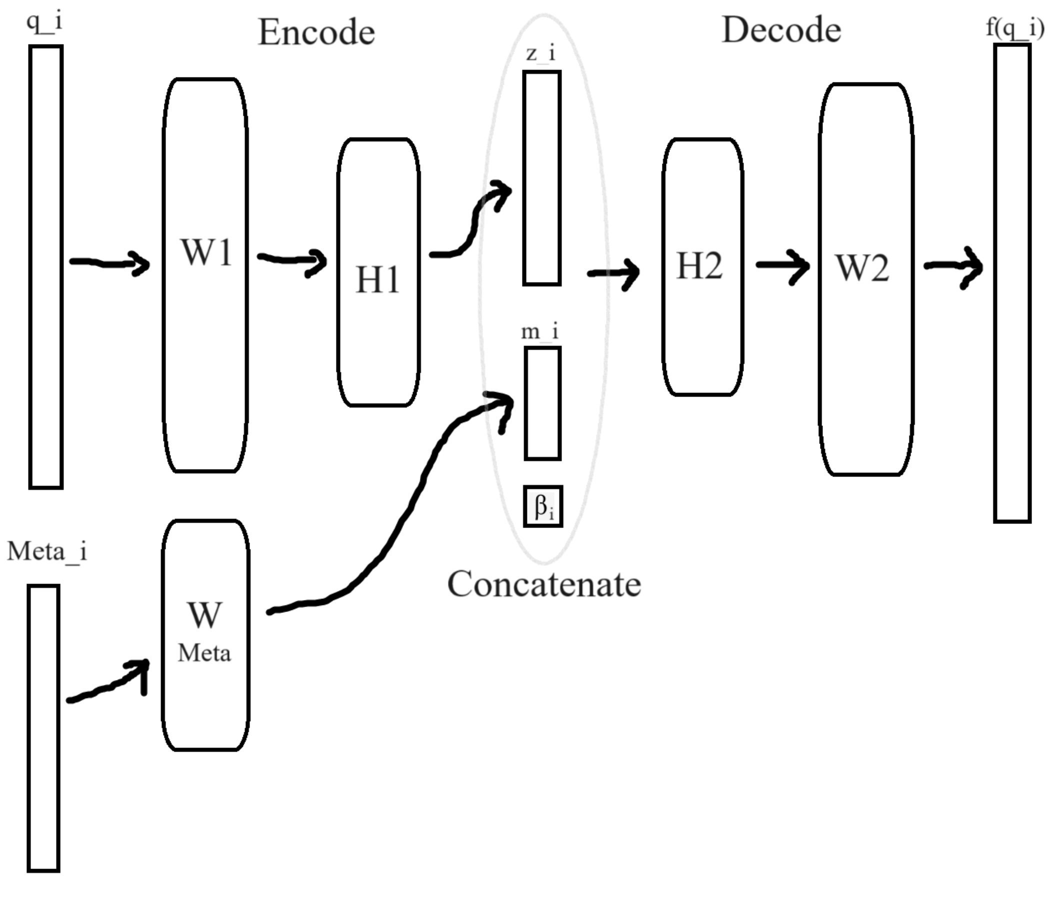


Figure 2: Diagram of Model during Evaluation/Inference

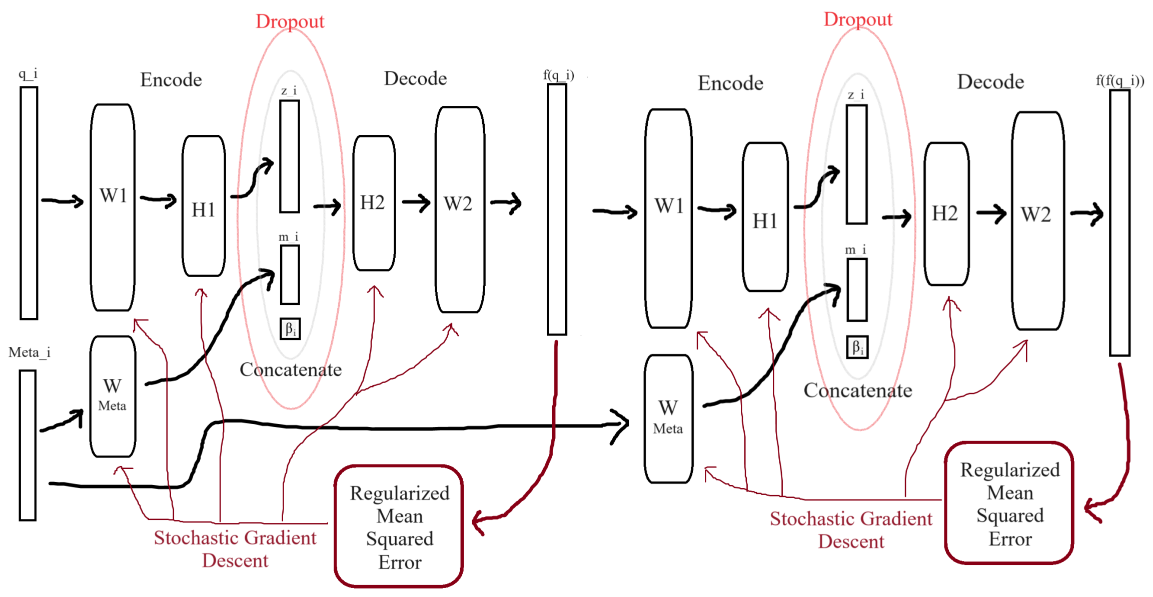


Figure 3: Diagram of Model during Training

2.3 Model vs Baseline Models

Model	Val. Acc.	Test Acc.	Hyperparameters
Part A Autoencoder	0.6841	0.6852	k=10, Epoch = 13, LR = 0.1 $\lambda = 0.001$
Metadata	0.6793	0.6815	k = 50, Epoch = 5, LR = 0.1 $\lambda = 0.001$
Theta Injection	0.6946	0.6943	k = 50, Epoch = 5, LR = 0.1 $\lambda = 0.001$
Question as Entity	0.69545	0.69432	k = 50, Epoch = 5, LR = 0.1 $\lambda = 0.001$
Question + Beta injection	0.7080	0.6973	k = 50, Epoch = 3, LR = 0.1 $\lambda = 0.001$
Question + Metadata	0.69602	0.6955	k = 50, Epoch = 3, LR = 0.1 $\lambda = 0.001$, $k_{meta} = 5$
Question + Dropout	0.6940	0.6901	k = 50, Epoch = 5, LR = 0.1 $\lambda = 0.001$
Question + Deeper	0.6934	0.6950	k = 25, Epoch = 15, LR = 0.1 $\lambda = 0.001$
Question + Deeper + Dropout	0.7132	0.7092	k = 25, Epoch = 100, LR = 0.1 $\lambda = 0.001$
Question + Deeper + Dropout + Refeeding	0.7116	0.7101	k = 25, Epoch = 50, LR = 0.1 $\lambda = 0.001$
Question + Deeper + Dropout + Refeeding + Metadata	0.7142	0.7151	k = 25, Epoch = 100, LR = 0.1 $\lambda = 0.001$, $k_{meta} = 5$
Question + Deeper + Dropout + Refeeding + Metadata + Beta Injection	0.7077	0.7061	k = 25, Epoch = 300, LR = 0.1 $\lambda = 0.001$, $k_{meta} = 5$

Note, each row of this table represents the performance of the modifications stated in the "Model" column. Each modification is has been added onto the baseline model from Part A, Q3.

2.4 Model Performance

Because our model is made up of many small modifications, the analyisis of performance can be broken down into smaller parts. Addition of just metadata was not beneficial as it likely just enhanced the model's overfitting tendencies. Additionally, the metadata available for user as entities is quite minimal, with gender and age likely being mostly irrelevant to quiz performance. Theta injection on its own did however make an improvement form the baseline. This is likely because we had no additional network being trained on the metadata itself, and it is only a single value. The model saw a major improvement when switching to a qsuestion as entity formulation, largely due to optimization enhancements.

Bringing metadata and beta injection in to play with question as entity also brought about performance improvements. The beta injection improvement was about the same as the theta injection's improvement on the baseline model. As well, the question meta data was far more nuanced and thus allowed for more complex patterns to be learned, thus enhancing optimization capabilities. The next modification to make was adding more layers to our neural net. By itself, this change actually hindered the models accuracy by a few hundredths of a percent, largely due to over fitting, although more layers allows for a model to learn more complex functions, without sufficient regularization techniques, the model will align itself almost completely to the training data.

Until this point, all our models had suffered from overfitting, which can be seen by the fact that their peak performance occured at very early epochs (they quickly drop off in accuracy due to overfitting). However, after adding dropout to our deeper, question as entity model, we saw a major improvement in performance. This was due to the regularizing effects of dropout, and its ability to dismantle co-dependence within the network.

From this point, our model's performance only increased slightly when adding dense re-feeding/meta injection and even decreased when all modifications were combined. We believe this came from complications with overfitting, due to have both the beta value and metadata injected, thus increase the size of our latent dimension significantly. I believe for these additional modifications to make a significant positive change to the performance, they would need to be added alongside other regularization techniques.

2.5 Model Limitation

One significant limitation of our proposed model lies in the hardware access. Since the only accessible machines to us are our laptops, the restriction in computation power arises naturally, making it hard for us to implement a network with more than two hidden layers. This constraint hinders our ability to develop deeper architectures and potentially deprives the model's capacity to represent data in scenarios where complicated patterns exist.

One other limitation exists due to the ineffectiveness of the dense re-feeding technique. Despite its theoretical potential to enhance the model's learning by leveraging output vectors as additional training data points, the validity of this approach was not as evident as expected due to the model's inability to achieve higher accuracy. With the model's performance plateauing at approximately 70% accuracy, a substantial proportion (30%) of results remains to be inaccurate predictions. Consequently, these inaccurate predictions could potentially mislead the gradient descent updates during training, diminishing the efficacy of the dense re-feeding strategy.

These limitations mark areas where further improvements and optimizations could be made to enhance the performance and efficacy of our model. One possible modification we can make to improve on this is to select alternative techniques in addition to our model. These techniques should have better, if not optimal, performance with shallower networks.

3 References

N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting" (2014)