CSC311 Project

Dharuniga Antony, Laiba Khan, Yiyun Zhang

August 8, 2025

Part A

1 k-Nearest Neighbors

1.1 Algorithm Overview

The k-Nearest Neighbors (KNN) algorithm performs collaborative filtering to predict student responses on diagnostic questions. We implemented two variants: user-based and item-based collaborative filtering.

User-based Collaborative Filtering: For a given student-question pair (i, j), the algorithm identifies the k most similar students based on their response patterns across all questions. The prediction is made by imputing missing values using the responses of these similar students.

Item-based Collaborative Filtering: For a given student-question pair (i, j), the algorithm identifies the k most similar questions based on response patterns across all students. The prediction uses the student's responses to these similar questions.

The underlying assumption for user-based filtering is that students with similar abilities and backgrounds will perform similarly across diagnostic questions. For item-based filtering, the assumption is that questions with similar response patterns across students share comparable difficulty levels or cover related concepts.

1.2 Implementation Details

Both methods use the KNNImputer from scikit-learn with NaN-Euclidean distance. For item-based collaborative filtering, we transpose the user-question matrix so that questions become rows and users become columns, apply KNN imputation, then transpose back to the original orientation.

1.3 Hyperparameter Tuning

We evaluated both methods across $k \in \{1, 6, 11, 16, 21, 26\}$ using validation accuracy as the selection criterion.

User-based Results:

- Best k^*: 11
- Validation accuracy: 0.6897
- Test accuracy: 0.6847

Item-based Results:

- Best k^*: 21
- Validation accuracy: 0.6916
- Test accuracy: 0.6828

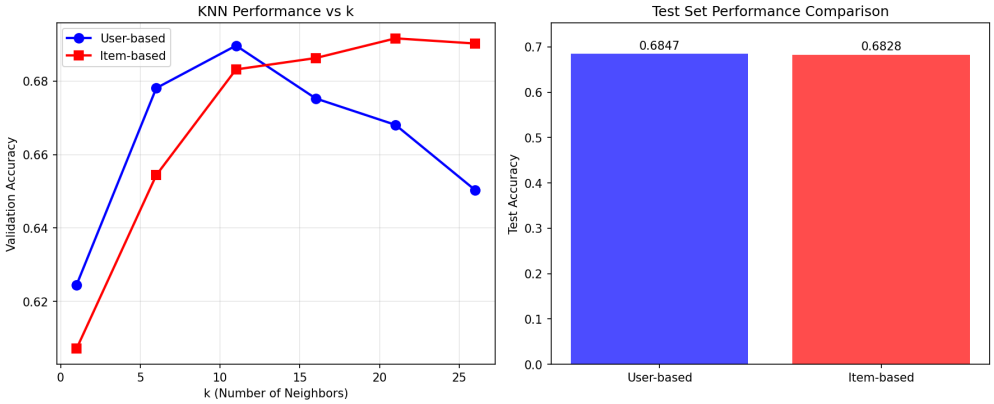

Figure 1: Left: Validation accuracy vs. number of neighbors k for both user-based and item-based collaborative filtering. Right: Test set performance comparison between the two methods using their respective optimal k^* values.

1.4 Performance Analysis

User-based collaborative filtering achieved slightly superior test performance (68.47% vs 68.28%), outperforming item-based filtering by 0.20 percentage points. However, both methods showed comparable performance overall.

1.5 Limitations

Several limitations affect KNN performance in educational recommendation systems:
Computational Complexity: KNN requires computing distances to all training samples for each prediction, becoming computationally expensive with large datasets. No model is learned during training—all computation occurs at prediction time.
Sparsity Issues: Educational data typically exhibits high sparsity, as most students have not answered most questions. This sparsity makes finding meaningful nearest neighbors challenging and reduces the reliability of distance calculations.
Cold Start Problem: The method cannot make predictions for new students with few or no responses (user-based) or new questions with limited response data (item-based).
Scalability: Memory and computational requirements grow quadratically with the number of users and questions, limiting applicability to larger educational platforms.
Distance Metric Sensitivity: KNN uses Euclidean distance, which may not be optimal for binary educational data. Alternative metrics such as cosine similarity or Jaccard similarity might be more appropriate for this domain.

2 Item Response Theory

2.1 Log-Likelihood and Derivatives

In the one-parameter logistic Item Response Theory (IRT) model, the probability that student i correctly answers question j is given by:

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

where β_j is the difficulty of question j, θ_i is the i-th student's ability, and $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

Let $C = \{c_{ij}\}$ be the set of observed correctness values, with $c_{ij} \in \{0, 1\}$.

Since each data point is independent, the likelihood of all the data is:

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

Taking the log of this expression, you get the log-likelihood:

$$\begin{aligned}
\log \mathcal{L}(\theta, \beta) &= \log \left(\prod_{i=1}^{N} \prod_{j=1}^{M} \sigma(\theta_i - \beta_j)^{c_{ij}} \cdot (1 - \sigma(\theta_i - \beta_j))^{1-c_{ij}} \right) \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} \left[\log \left(\sigma(\theta_i - \beta_j)^{c_{ij}} \right) + \log \left((1 - \sigma(\theta_i - \beta_j))^{1-c_{ij}} \right) \right] \\
&= \sum_{i=1}^{N} \sum_{j=1}^{M} \left[c_{ij} \log \sigma(\theta_i - \beta_j) + (1 - c_{ij}) \log(1 - \sigma(\theta_i - \beta_j)) \right]
\end{aligned}$$

Now compute the gradient of the log-likelihood with respect to θ_i and β_j.

Identities:

$$\frac{d}{dz}\sigma(z) = \sigma(z)(1 - \sigma(z)) \tag{1}$$

$$\frac{d}{dz} \log \sigma(z) = \frac{1 - \sigma(z)}{\sigma(z)} \cdot \sigma(z)(1 - \sigma(z)) = 1 - \sigma(z) \tag{2}$$

$$\frac{d}{dz} \log(1 - \sigma(z)) = -\frac{\sigma(z)}{1 - \sigma(z)} \cdot \sigma(z)(1 - \sigma(z)) = -\sigma(z) \tag{3}$$

Gradient with respect to θ_i:

$$\begin{aligned}
\frac{\partial}{\partial \theta_i} \log \mathcal{L}(\theta, \beta) &= \sum_{j=1}^{M} \frac{\partial}{\partial \theta_i} \left[c_{ij} \log \sigma(\theta_i - \beta_j) + (1 - c_{ij}) \log(1 - \sigma(\theta_i - \beta_j)) \right] \\
&= \sum_{j=1}^{M} \left[c_{ij} \cdot \frac{d}{d\theta_i} \log \sigma(\theta_i - \beta_j) + (1 - c_{ij}) \cdot \frac{d}{d\theta_i} \log(1 - \sigma(\theta_i - \beta_j)) \right] \\
&= \sum_{j=1}^{M} \left[c_{ij} \cdot (1 - \sigma(\theta_i - \beta_j)) - (1 - c_{ij}) \cdot \sigma(\theta_i - \beta_j) \right] \quad \text{(by identities 2 and 3)} \\
&= \sum_{j=1}^{M} \left[c_{ij} - \sigma(\theta_i - \beta_j) \right]
\end{aligned}$$

Gradient with respect to β_j:

$$\begin{aligned}
\frac{\partial}{\partial \beta_j} \log \mathcal{L}(\theta, \beta) &= \sum_{i=1}^{N} \frac{\partial}{\partial \beta_j} \left[c_{ij} \log \sigma(\theta_i - \beta_j) + (1 - c_{ij}) \log(1 - \sigma(\theta_i - \beta_j)) \right] \\
&= \sum_{i=1}^{N} \left[c_{ij} \cdot \frac{d}{d\beta_j} \log \sigma(\theta_i - \beta_j) + (1 - c_{ij}) \cdot \frac{d}{d\beta_j} \log(1 - \sigma(\theta_i - \beta_j)) \right] \\
&= \sum_{i=1}^{N} \left[-c_{ij} \cdot (1 - \sigma(\theta_i - \beta_j)) + (1 - c_{ij}) \cdot \sigma(\theta_i - \beta_j) \right] \quad \text{(chain rule + identities)} \\
&= \sum_{i=1}^{N} \left[\sigma(\theta_i - \beta_j) - c_{ij} \right]
\end{aligned}$$

Final Gradients:

$$\frac{\partial \log \mathcal{L}}{\partial \theta_i} = \sum_{j=1}^{M} (c_{ij} - \sigma(\theta_i - \beta_j))$$
$$\frac{\partial \log \mathcal{L}}{\partial \beta_j} = \sum_{i=1}^{N} (\sigma(\theta_i - \beta_j) - c_{ij})$$

2.2 Hyperparamater Tuning

Chosen hyperparameters: Learning rate = 0.001, Iterations = 140
Final Train Accuracy: 0.7320
Final Validation Accuracy: 0.7083
Test Accuracy: 0.7034

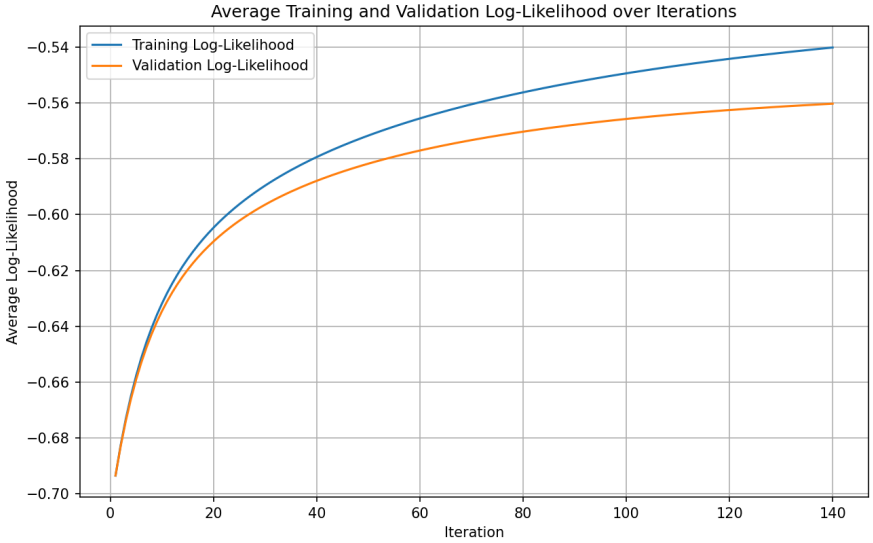

Figure 2: Training and validation log-likelihoods plotted against training iterations. The curve illustrates how model fit improves over time, with the training log-likelihood generally increasing (i.e., negative NLL decreasing). A large gap between training and validation curves may indicate overfitting.

2.3 Visualization of IRT Item Response Functions

The Item Characteristic Curves (ICCs) plotted for three selected questions (Figure 3) have a sigmoidal (S-shaped) form, consistent with the one-parameter logistic (1PL) IRT model. As the user ability parameter θ increases, the probability of a correct response $p(c_{ij} = 1)$ also increases. Each curve is centred around a different point on the θ-axis, reflecting the difficulty parameter β_j of the corresponding question. Questions with lower difficulty values shift the curve to the left, while higher difficulty values shift it to the right. The steepness of the curve indicates how sensitive the probability of a correct response is to changes in user ability near the difficulty level of the item. In the 1PL model, there is no item discrimination parameter a_j, so the slope at the inflection point is identical across items.

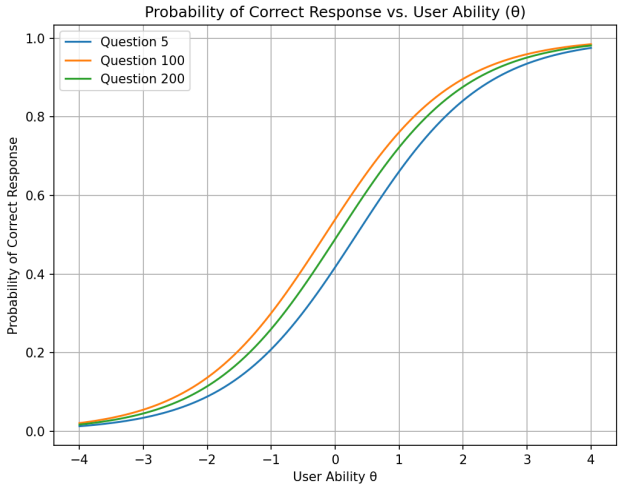

Figure 3: Item characteristic curves (ICCs) for three selected questions, showing the probability of a correct response as a function of user ability (θ). Each curve represents a different question and shifts based on its difficulty (β).

3 Neural Networks

3.1 ALS vs Neural Networks

The model's goal is to predict whether a student will answer a question correctly given the sparse matrix. Each entry in the matrix will be 1,0 or NaN depending on whether the student's answer is correct or withheld. The model should learn from this data and be able to predict how a student would perform on unseen questions. ALS (Alternating Least Squares) and Neural Networks attempt to achieve this using different training and modeling methods.

Training Method
ALS learns using matrix factorization. It approximates the entries of the student-question sparse matrix, S, using two smaller matrices A (number of students $\times$ latent factors) and B (number of questions $\times$ latent factors) such that:

$$S \approx AB^T$$

ALS alternates between fixing one matrix and solving for the other using least squares.

Whereas Neural Networks learn by minimizing the loss function. This model makes a prediction, compares it to the real answer, and updates the weights using backpropagation. Training utilizes stochastic gradient descent and updates all parameters simultaneously based on the computed gradients from the loss unlike ALS which does it in alternating steps.

Modeling Abilities
ALS models the interaction between students and questions using the linear dot product $A_i \cdot B_j$. Hence, the performance is determined by how similar the students' latent skill vector is to the question's concept vector. Therefore, the relationship is represented linearly and may limit the model's ability to represent complex dependencies between concepts. An example of this could be that an ALS model may learn that a student with a high algebra score with likely get an algebra question correct, but is not able to determine the probability the student does well if they're only good at a specific concept in math.

However, Neural Networks can model non-linear relationships using activation functions like sigmoid or ReLU. Thus, it is possible to capture how much a student's success is dependent on other concepts or how questions combine multiple skills. An example is a neural network can learn that a student only starts doing

well on word problems after they do better on questions related to reading comprehension.

Training Speed
ALS tends to be faster to train, specifically on sparse datasets such as the student-question matrix we have. This is because ALS relies on solving least square problems, which can be computed efficiently using linear algebra.

Neural Networks often require more computations and time to train due to training being iterative and the usage of stochastic gradient descent. This model is also more sensitive to hyperparameters such as the learning rate and can require more extensive tuning to achieve good performance.

3.2 Implementation Details of Class AutoEncoder

Given inputs, the function forward first compress the vector into a k-dimensional vector also known as the student's latent representation. Then applies sigmoid to keep the values between 0 and 1. Afterwards, the k-dimensional vector is expanded back to the full number of questions. Once again, sigmoid is apples so that the output predictions are between 0 and 1, and can be interpreted as the probability that the student will get each question right.

3.3 Hyperparameter Tuning

We evaluated across $k \in \{10, 50, 100, 200, 500\}$ using validation accuracy as the selection criterion. Note that lambda is set to 0 currently as regularization is not included here.

- Best k^*: 10
- Validation accuracy: 0.6904
- Test accuracy: 0.6771

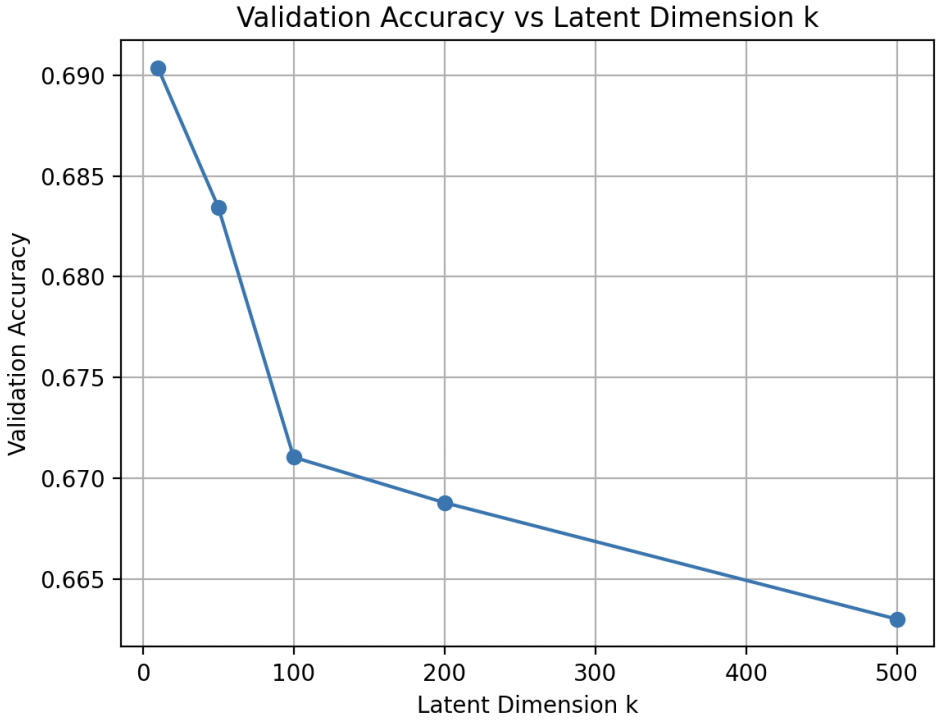

Figure 4: Validation Accuracy as k grows.

3.4 Function of Epoch

In Figure 2 it is seen that as the number of epochs increases, the training loss decreases, and the validation accuracy increases. The final accuracy is 0.6935.

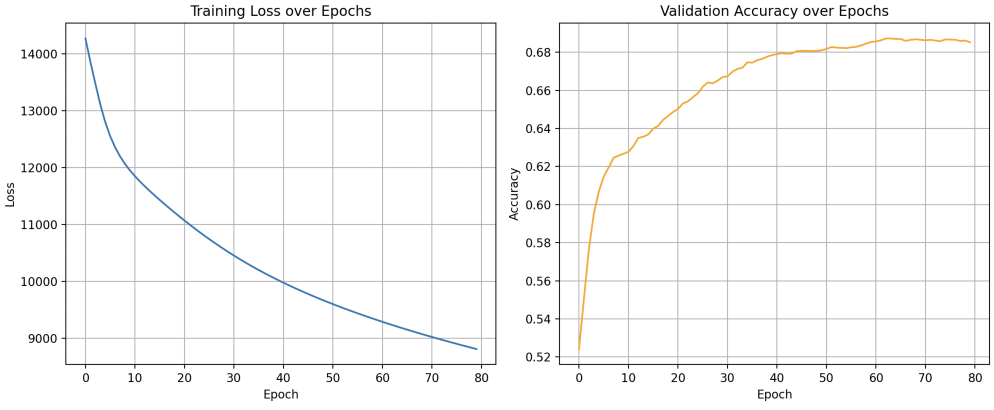

Figure 5: Training and validation objectives as a function of epoch.

3.5 Lambda

- Best $\lambda*$: 0.001
- Validation accuracy: 0.6857
- Test accuracy: 0.6915

Figure 3 shows that the model performs better when $\lambda*$ is not 0, so with regularization.

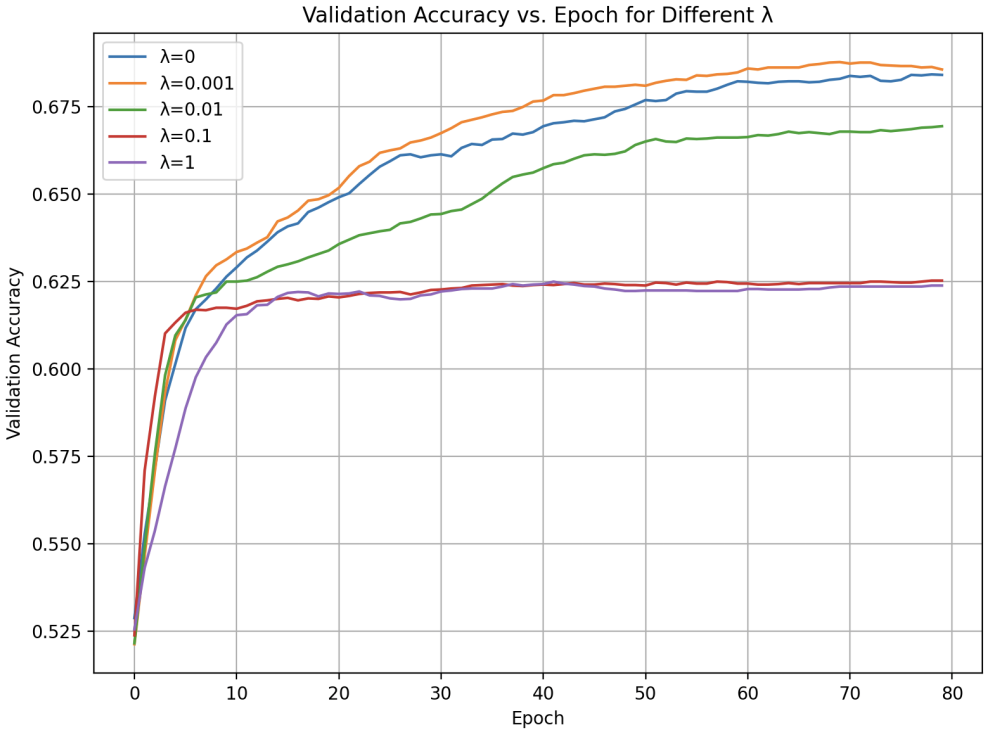

Figure 6: Validation Accuracy for different lambda values

4 Ensemble

To improve prediction stability and accuracy, we implemented a bagging ensemble using our three base models: K-Nearest Neighbors (KNN), Item Response Theory (IRT), and a Neural Network AutoEncoder (NN).

4.1 Ensemble Process.

We first generated bootstrap variations of the training data:

- **KNN:** Performed user-based KNN imputation after randomly masking 20% of observed responses.
- **IRT:** Trained the IRT model using a bootstrapped sample of user-question-response triplets.
- **NN:** Trained a denoising autoencoder with dropout-style masking on the user-question matrix.

Each model output predicted probabilities $\hat{y}_{\text{KNN}}$, $\hat{y}_{\text{IRT}}$, and $\hat{y}_{\text{NN}}$. We computed the final ensemble prediction using a weighted average:

$$\text{Ensemble Prediction} = w_{\text{KNN}} \cdot \hat{y}_{\text{KNN}} + w_{\text{IRT}} \cdot \hat{y}_{\text{IRT}} + w_{\text{NN}} \cdot \hat{y}_{\text{NN}}$$

where the weights were normalized based on each model's validation accuracy:

$$w_{\text{KNN}} = 0.3601, \quad w_{\text{IRT}} = 0.3256, \quad w_{\text{NN}} = 0.3143$$

Table 1 shows the validation and test accuracies for each model and the final ensemble.

Table 1: Validation and Test Accuracy of Individual Models and Ensemble

Model	**Validation Accuracy**	**Test Accuracy**
KNN	0.6897	0.6847
IRT	0.7083	0.7034
Neural Network	0.6857	0.6915
Ensemble	**0.7211**	**0.7206**

4.2 Analysis.

The ensemble model achieves a higher validation and test accuracy than any individual model. This improvement is due to the ensemble's ability to combine diverse predictive strengths while reducing the variance associated with any single model. The use of weighted averaging based on validation performance ensures that more accurate models have greater influence, further enhancing generalization.

Part B

1 Formal Description

Baseline Model
The baseline model is a shallow autoencoder with one hidden layer. It has an input and an output layer, where the size is the number of questions. and a hidden layer of k units. This model is trained to reconstruct the user's full response vector, even if some answers are unknown. The goal is to fill in these blanks based on other patterns.

Modified Model
The modified model is deep denoising autoencoder architecture with additional optimization strategies. The main modification is the newly implemented deep architecture that uses two encoding and two decoding layers with ReLU activation and Batch Normalization. However, the model also uses denoising techniques and contains other strategies learned in the course such as:

1. Adam Optimizer: Replaces SGD for better gradient handling as it adapts learning rates per parameter.
2. Learning Rate Scheduler: Reduces learning rate when validation accuracy plateaus.
3. Early Stopping: Stops training if no improvement is seen in validation accuracy after 5 epochs to prevent overfitting.
4. Mini-batching: Trains using small batches of size 16 to improve convergence and generalization.

There are a few potential benefits, the largest benefit we hope to see is with the deep architecture. In the modified model, the two encoding layer compress the input which is a vector of size 1774. The first layer compresses it to size 100, then this output is normalized for faster and more stable training. ReLU activation is also applied to add non-linearity, allowing the model to learn more complex functions. The second encoding layer further compresses the vector to size 50, forcing the model to learn more efficient representation.

The decoding layer reverses the steps and uses sigmoid to turn the output into probabilities of the correct answer. This structure is beneficial as multiple layers allow the model to learn hierarchical features. For example, the first layer may detect "math vs reading" while the second layer detects "algebra vs geometry" or "vocabulary vs comprehension".

The benefit of this model could be the ability to learn more complex pattern, abstract the most essential information which is useful here, as we have very sparse and high-dimensional data.

Besides the deep architecture, the model applies noise to the input, randomly hiding about 10% of known responses during training. The reason we expect this to aid with performance is becasue the base model copies its input directly which could cause it to memorize patterns and over fit. By scrambling it and asking the model to reconstruct the correct version, we force it to learn a meaningful structure in the data. The model should improve in predicting missing or uncertain values, even when the data is incomplete. Denoising acts as a kind of regularization and prevents the model from overfitting. Additionally, helping the model capture general patterns instead of memorizing specific ones also helps with generalization.
Finally, our modified model also has some other optimization enhancements. Overall we can expect:

- **Denoising** to increase robustness to noise, leading to better generalization.
- **Deeper layers** to enable the model to learn more expressive features.
- **Batch normalization** and **ReLU activations** to accelerate and stabilize training.
- **Adam optimizer** to provide adaptive learning rates for smoother convergence.
- **Learning rate scheduling** and **early stopping** to help prevent overfitting.
- **Mini-batching** to reduce variance of gradients and speed up computation.

Training Algorithm

Algorithm 1 Training with Denoising and Optimization Enhancements

for each epoch **do**
  for each batch of students **do**
    Apply masking noise if denoising enabled
    Replace NaNs in targets with model output
    Compute loss:

$$\text{Loss} = \|f_\theta(\mathbf{x}) - \mathbf{x}\|^2 + \frac{\lambda}{2} \sum_l \|W^{(l)}\|^2$$

    Backpropagate and update parameters
  end for
  Evaluate validation accuracy
  Reduce learning rate if validation accuracy plateaus
  if early stopping criteria met **then**
    Break
  end if
end for=0

2 Comparison or Demonstration

Model Comparison
We compared a baseline autoencoder model with an enhanced model that includes several regularization and optimization improvements.

Model	Test Accuracy
Baseline Autoencoder (SGD)	0.6825
Deeper + Denoising (Adam)	0.6142
Enhanced Model (All Combined)	**0.6669**

Table 2: Test accuracy across different model variants.

Interestingly, the baseline model trained with SGD outperformed all enhanced variants in terms of raw test accuracy. Although individual enhancements such as de-noising and implementing the deep architecture decreased the performance, implementing all the enhancements led to a test accuracy similar to the baseline but did not surpass it.

Analysis
The comparison shows that although the enhanced model achieved a much lower training loss early on, it did not translate into better validation or test accuracy. In fact, the baseline model outperformed the enhanced one in terms of validation performance.

- Training Loss: The enhanced model, aided by batch normalization, deeper layers, and the Adam optimizer, converged quickly to a lower loss. This indicates improved learning efficiency.
- Validation Accuracy: Despite faster convergence, the enhanced model's validation accuracy plateaued and fluctuated, peaking around 0.655. In contrast, the baseline model improved steadily, reaching a higher accuracy of approximately 0.688.
- Generalization: These results suggest that while the enhancements improved optimization, they did not necessarily improve generalization on unseen data.

Overall, the baseline autoencoder with SGD maintained a more consistent and generalizable learning trajectory, outperforming the deeper denoising model in this setting.

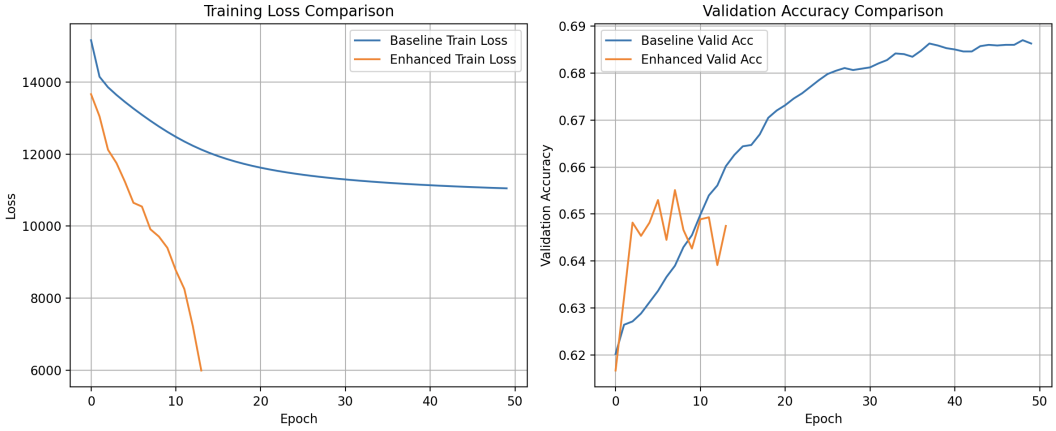

Figure 7: Training loss (left) and validation accuracy (right) for baseline vs. enhanced model.

3 Model Diagram

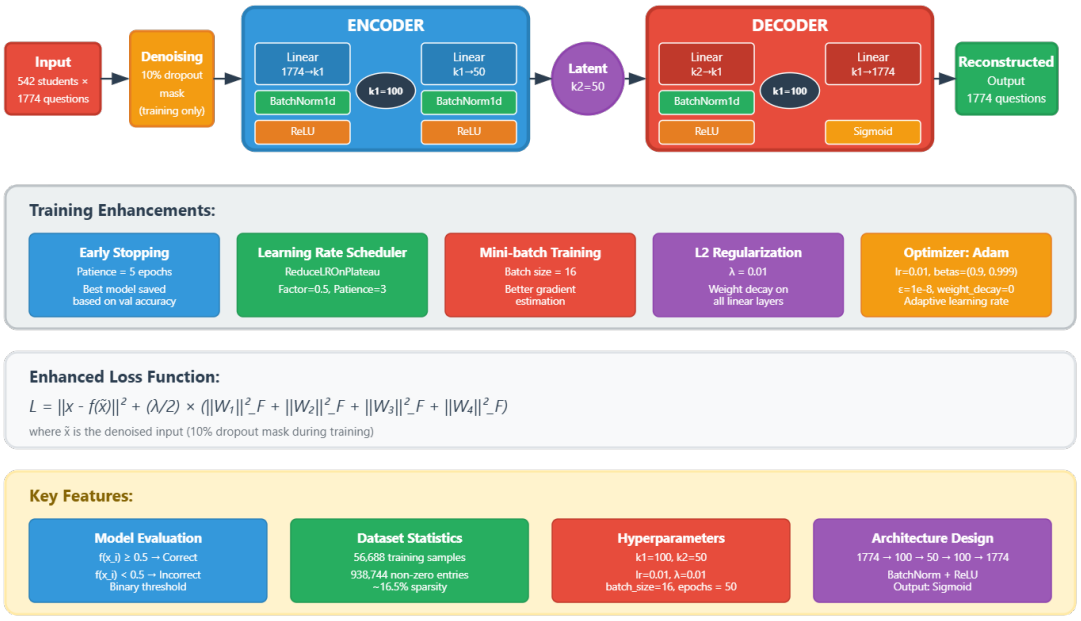

Figure 8: Diagram of full model concept including architecture and key features.

4 Limitations

Despite our efforts to improve the neural network with various enhancement techniques, our enhanced model actually performed worse than the baseline (test accuracy of 0.6497 vs 0.6825). This reveals several important limitations of our approach:

Over-Engineering and Complexity Trade-offs: Our enhanced model combines multiple techniques (deep architecture, batch normalization, denoising, early stopping) but achieves worse performance than the simple baseline. This suggests that adding complexity does not always lead to better results, especially with limited training data. The educational dataset may not be large or complex enough to benefit from deep architectures, and the additional parameters may lead to overfitting despite regularization attempts.

Inappropriate Denoising Strategy: Our denoising approach randomly masks 10% of known responses during training, but this may not match real-world patterns in educational data. Students don't randomly skip questions - they typically avoid questions that are too difficult or outside their knowledge area. By applying random noise instead of realistic missing data patterns, the model may learn incorrect assumptions about student behavior, leading to worse predictions.

Conflicting Regularization Effects: Our ablation study shows that individual improvements (like switching to Adam optimizer or adding batch normalization) sometimes help, but combining them all together makes performance worse. This suggests that different regularization techniques may interfere with each other. For example, batch normalization and denoising might both be trying to solve similar problems, and using them together could over-regularize the model.

Dataset Size Limitations: The educational dataset contains 542 students and 1774 questions, which may be too small to effectively train deep neural networks. Deep models typically need much larger datasets to learn meaningful representations. Our baseline shallow autoencoder might be more appropriate for this dataset size, explaining why it outperforms the deeper architecture.

Potential Improvements and Future Work

Our results demonstrate that adding complexity to machine learning models does not guarantee improved performance. The failure of our enhanced model highlights the importance of understanding the problem domain before applying advanced techniques. Educational data has unique characteristics, such a high sparsity, meaningful missing patterns, and relatively small dataset sizes, that may favor simpler, domain-specific approaches over general deep learning methods.

Future work should prioritize systematic experimentation over feature accumulation. Rather than combining multiple enhancement techniques simultaneously, researchers should conduct careful ablation studies to understand which individual improvements actually help. Additionally, denoising strategies should reflect realistic student behavior patterns rather than random noise assumptions.

The success of our ensemble method (0.7206 test accuracy) compared to individual neural networks suggests that combining different model types may be more effective than making a single model more complex. Future approaches should focus on leveraging educational metadata such as question topics and prerequisite relationships, which traditional methods like IRT can naturally incorporate. The key insight is that machine learning improvements should be guided by domain understanding rather than simply applying the latest techniques.