Matrix Factorization for Classification: A Neural Network Approach

Sami Taider
84401097
Rayan Hanader
84401096

November 30, 2024

1 Introduction

Matrix factorization is a technique commonly used in recommendation systems to predict missing values or classify items based on latent factors. In this project, a neural network-based approach was employed to classify ratings from a given matrix. The model uses embeddings for dimensionality reduction and a linear classifier to make predictions.

2 Model Architecture

The neural network consisted of:

- **Embedding Layers:** Used to represent rows (users) and columns (items) of the input matrix in a lower-dimensional latent space.
- **Linear Layer:** Combines the latent features and predicts classifications.
- **Classifier:** A fully connected layer with 6 output neurons corresponding to the possible classes.

The forward pass concatenates the embeddings of rows and columns and passes them through the classifier.

The architecture is defined as follows:

```
class MatrixFactorization (torch.nn.Module):
    def __init__(self, n, d, h=1, c=6):
        super().__init__()
        self.U = torch.nn.Embedding(n, h)
        self.V = torch.nn.Embedding(d, h)
        self.classifier = torch.nn.Linear(2*h, c)

    def forward(self, X_batch):
        U = self.U.weight[X_batch[:,0]]
        V = self.V.weight[X_batch[:,1]]
        out = torch.cat((U, V), 1)
        return self.classifier(out)
```

Listing 1: Model Architecture

3 Modifications and Choices

3.1 Loss Function

Initially, **Mean Squared Error (MSE) Loss** was used. However, it caused issues due to its unsuitability for classification tasks. MSE calculates the squared difference between predictions and true values, making it more appropriate for regression problems.

The switch to **CrossEntropyLoss** was necessary because it:

- Handles classification tasks by combining softmax and negative log-likelihood.
- Allows the model to work directly with raw logits, improving numerical stability.
- Produces probabilities for evaluation when applied to logits using softmax.

3.2 Handling Output Shapes

Initially, there was a mismatch between `y_batch` (1D tensor of true labels) and `y_pred` (2D tensor of logits). This was resolved by passing `y_batch`

directly as class indices to `CrossEntropyLoss`. This removed the need to convert `y_batch` to one-hot vectors.

4 Training Process

4.1 Parameters

- **Learning Rate:** 0.1
- **Epochs:** 1000
- **Optimizer:** Adam optimizer for adaptive learning.
- **Batch Size:** 16

4.2 Results

The model was trained on an 8x8 matrix with ratings ranging from 0 to 5. The training loss decreased over time, as shown in Figure 1.

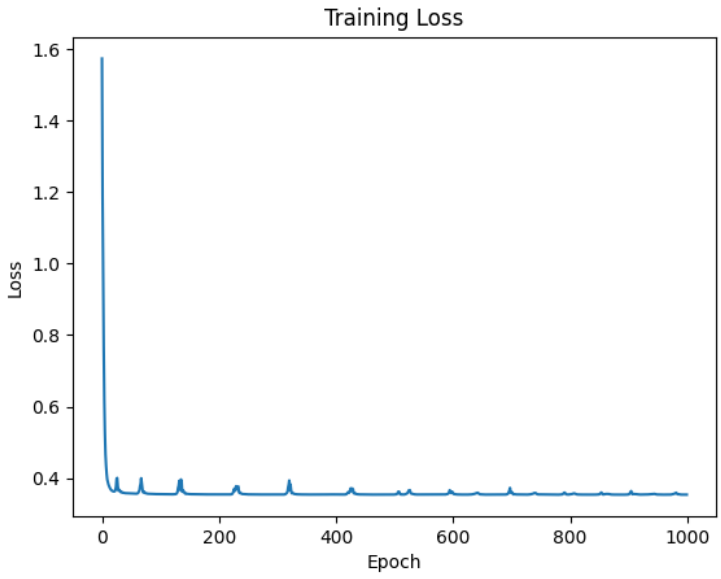

Figure 1: Training Loss Over Epochs

5 Evaluation

The model was tested on a new 8x8 matrix. Predictions were converted to probabilities using softmax and compared against true labels. The confusion matrix (Table 1) shows the distribution of correct and incorrect predictions.

True \Predicted	**0**	**1**	**2**	**3**	**4**	**5**
0	7	8	0	0	0	0
1	1	26	0	0	0	0
2	0	1	0	0	0	1
3	0	3	0	3	0	0
4	0	2	0	0	2	0
5	0	0	0	1	0	9

Table 1: Confusion Matrix

In addition to the confusion matrix, the following plot (Figure 2) compares the true ratings and predicted ratings for each sample in the test set. The x-axis represents the sample index, while the y-axis shows the rating. The lines connecting the points indicate the differences between the true and predicted ratings.

5.1 Metrics

- **Accuracy:** The model achieved an accuracy of 73

6 Conclusion

This project demonstrated how matrix factorization could be approached using ANNs for classification tasks. Key learnings include the importance of using appropriate loss functions and addressing output shape mismatches. The

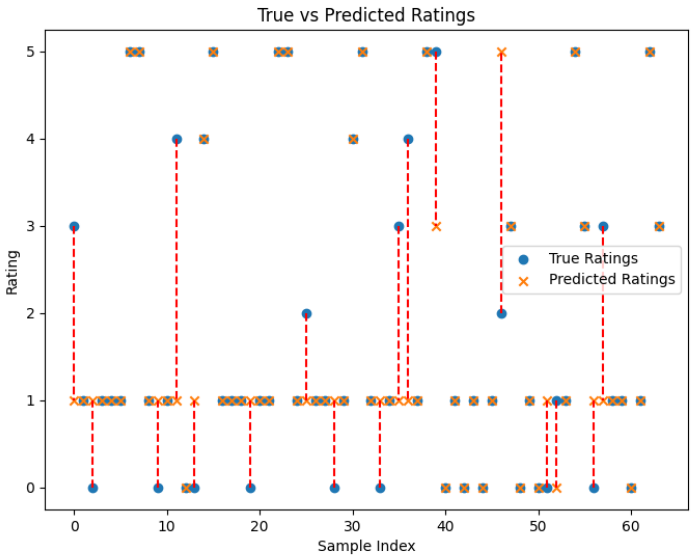

Figure 2: True vs Predicted Ratings.

final model showed strong performance, but there is room for improvement, such as tuning hyperparameters or experimenting with deeper architectures.