CMPE 597 Sp. Tp. Deep Learning Spring 2025 Assignment 1

Kutay Eroğlu, 2024700051 (kutay.eroglu@std.bogazici.edu.tr) Group **12**
Mücahit Erdoğan Ünlü, 2021400171 (mucahit.unlu@std.bogazici.edu.tr) 13 April 2025

1 Overview

Throughout this report, classification using cross-entropy will be referred to as Task 1, and classification using word embeddings as Task 2. All plots and metrics related to Task 1 are from NumPy-based implementation, unless stated otherwise. To establish a reliable baseline, Task 1.2 — Implementation with Deep Learning Libraries — was addressed first. The full implementation is available at the following GitHub repository and Google Drive Link

2 Classification with Cross-Entropy

Depth & Width of the Network and Activation Functions The network architecture was chosen considering the simplicity and dimensionality of the Quick Draw dataset, comprising small images (28×28) and five distinct categories. Two hidden layers were selected to sufficiently capture necessary feature hierarchies without overfitting. The number of neurons in these layers (128 and 64) were chosen based on common heuristics and typical values used for similar image-classification tasks. The Rectified Linear Unit (ReLU) activation function was used due to its computational efficiency, simplicity, and effectiveness in addressing vanishing gradients.

Momentum Value for Stochastic Gradient Descent with Momentum Momentum was set to the suggested value of 0.9. Since the model achieved satisfactory performance, this hyperparameter was not further adjusted.

Performance Metrics on Test Set Evaluation metrics on test set are presented in Figure 1; precision-recall curve in Figure 2; roc curve in Figure 3 For precision, the weighted averaging method was used to provide a more reliable assessment in the presence of class imbalance. Therefore, the accuracy score is equal to the recall, as defined in [1].

```
Test Metrics:
accuracy: 0.8322
precision: 0.8359
recall: 0.8322
roc_auc: 0.9684
pr_auc: 0.9114
```

Figure 1: Evaluation metrics on test set for Task 1

The test metrics, shown in Figure 1, indicate strong overall performance. The model achieves an accuracy of 83.22%, with balanced precision (83.59%) and recall (83.22%), suggesting consistent classification across categories. Additionally, the high ROC AUC of 0.9684 and PR AUC of 0.9114 highlight the model's ability to discriminate between classes and maintain reliable precision-recall trade-offs.

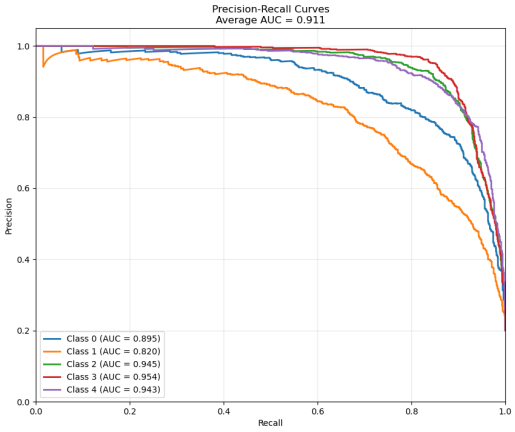

Figure 2: Precision-recall curve on test set for Task 1

Figure 2 presents the precision-recall curves for each class, with an average AUC of 0.911. While most classes (2, 3, and 4) achieve very high AUC values above 0.94, Class 1 notably lags behind with an AUC of 0.820, indicating that the model struggles more with that category. Class 0 performs reasonably well with an AUC of 0.895. Overall, the curves show strong precision across high recall ranges, suggesting reliable performance, though the variation between classes highlights potential room for improvement in class-specific representation or balance.

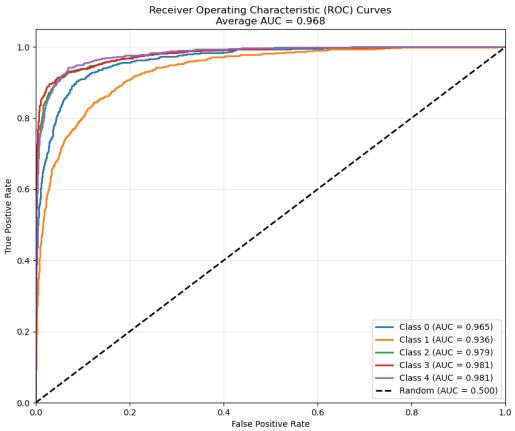

Figure 3: ROC curve on test set for Task 1

Figure 3 displays the ROC curves for each class, with an impressive average AUC of 0.968. All classes show strong separability, with Classes 3 and 4 achieving the highest AUC scores of 0.981. Even the lowest-performing class (Class 1) maintains a solid AUC of 0.936, indicating reliable discrimination between true and false positives across all categories. The tight clustering of most curves near the top-left corner highlights the model's overall effectiveness in distinguishing between classes, reinforcing its strong classification capability

Loss and Accuracy Plots for Training & Validation Figure 4 shows training and validation loss and accuracy over epochs. Training loss decreases steadily, indicating effective learning, while validation loss plateaus and slightly increases toward the end, suggesting early signs of overfitting. Similarly, training

accuracy improves consistently, whereas validation accuracy fluctuates and slightly declines after epoch 8. This indicates limited generalization and potential for improvement through regularization.

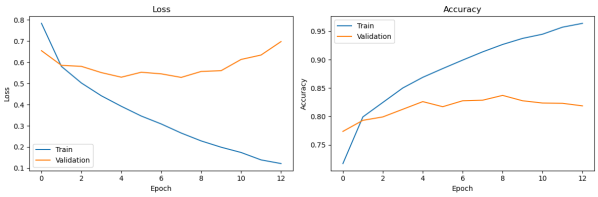

Figure 4: Loss and Accuracy Plots for Task 1

Choice of Hyperparameters Reasoning behind the depth and width of the network (hidden_dims in Figure 5) is discussed in Section 1.a; momentum in 1.b.

- **Batch Size:** A batch size of 64 is a common choice that offers a good trade-off between computational efficiency and the stability of gradient estimates. It helps maintain the stochastic nature of gradient descent, which can help in escaping local minima.
- **Validation Split:** Reserving 10% of the dataset for validation provides a sufficient subset to monitor the model's performance on unseen data during training. This helps in detecting overfitting and in making decisions about early stopping.
- **Learning Rate:** A learning rate of 0.01 is a standard starting point that allows for reasonably fast convergence without overshooting minima.
- **Epochs and Patience:** Setting the maximum number of epochs to 20, with an early stopping patience of 5, helps prevent overfitting. If the validation performance does not improve for 5 consecutive epochs, training stops early, saving computational resources and avoiding degradation in performance on unseen data.

```
batch_size: 64
val_split: 0.1
hidden_dims: [128, 64]
lr: 0.01
momentum: 0.9
epochs: 20
patience: 5
```

Figure 5: Hyperparameters for Task 1

Implementation with Deep Learning Libraries As specified in the assignment description, the model was trained using the same optimizer and hyperparameters as in Task 1. Compared to the NumPy implementation, the validation loss exhibits a more consistent and noticeable decrease. Similarly, validation accuracy increases more steadily. Both trends are illustrated in Figure 6.

Regarding the evaluation metrics, the PyTorch implementation demonstrates a slight advantage over the NumPy version, as shown in Figure 7. However, the difference is minimal.

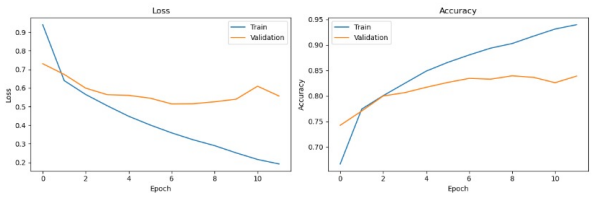

Figure 6: Loss and Accuracy Plots for Task 1: PyTorch Implementation

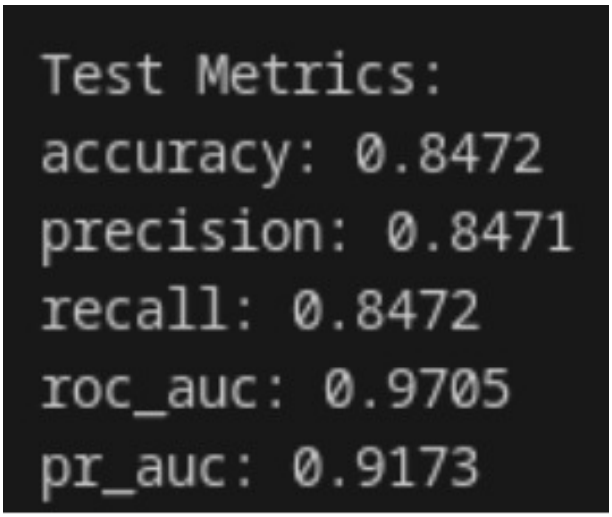

Figure 7: Evaluation metrics on test set for Task 1: PyTorch Implementation

3 Classification with Word Embeddings

Evaluation Metrics The performance of the model in Task 2, which employs embedding-based classification instead of cross-entropy loss, is summarized in Figure 8 and Figure 9. The reported test metrics indicate a strong performance: the model achieves an accuracy of 84.42%, precision of 84.33%, and recall of 84.42%. Additionally, the high ROC AUC score of 0.9562 and PR AUC of 0.9050 suggest that the model is effective at distinguishing between the five classes.

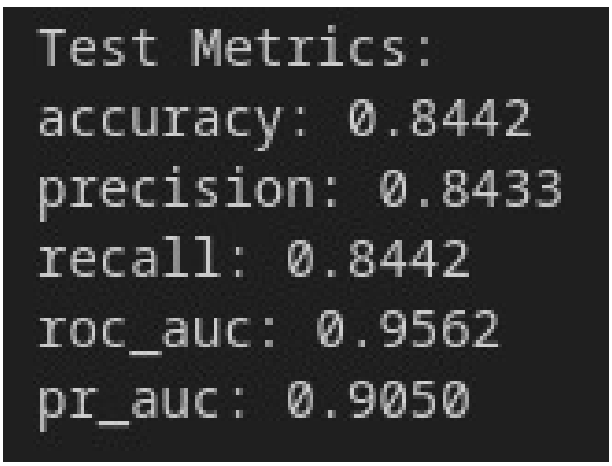

Figure 8: Evaluation metrics on test set for Task 2: PyTorch Implementation

Loss and Accuracy Plots In the training curves (Figure 9), the training loss steadily decreases, suggesting consistent learning throughout the epochs. However, the validation loss and accuracy display noticeable fluctuations, indicating some instability in generalization across epochs. These "zig-zag" patterns suggest that while the model learns useful representations, it may be sensitive to the validation

subset or require additional regularization to stabilize performance. Despite this, the overall trend in validation accuracy is positive, and the final performance metrics confirm the effectiveness of the shared embedding space in handling classification via similarity ranking.

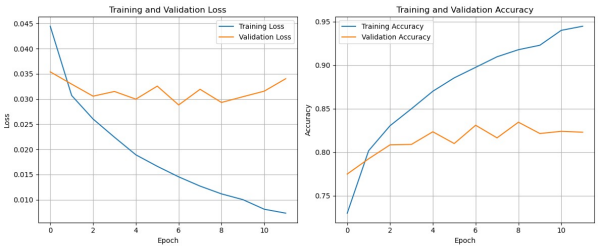

Figure 9: Loss and Accuracy Plots for Task 2: PyTorch Implementation

4 How to Run the Code

1. Clone the source code from the GitHub repository linked in Section 1.
2. Create a conda environment using the provided `environment.yml` file.
3. Download and place the Quick Draw dataset inside the `data` directory.
4. Run `main.py` for the NumPy-based implementation, or `torch_main.py` for the PyTorch-based implementation.

References

[1] Scikit-learn. *sklearn.metrics.recall_score*. Available at: `https://scikit-learn.org/stable/modules/generated/sklearn.metrics.recall_score.html`. Accessed: April 13, 2025.