MLP Spam Detector

Report - Assignment 1

Daniele Gotti - 1079011

November 2025

1 Introduction

In this project, I developed a binary classifier to distinguish spam emails from non-spam emails. Given a training dataset with 57 numerical features, I implemented a MLP using the PyTorch framework. My development process focused on a structured hyperparameter tuning phase, leveraging an 80/20 training-validation split to evaluate different model configurations and prevent overfitting. In this report, I outline the design choices, experimental results, and the final model selection process.

2 Methodology

2.1 Data Preprocessing

The training dataset consists of 3680 samples. I split it into a training set (80%, 2944 samples) and a validation set (20%, 736 samples). To handle features with different scales, I employed the `StandardScaler` from scikit-learn. I fitted the scaler exclusively on the training data and then used it to transform both the training and validation sets, preventing any data leakage.

2.2 Model Architecture and Implementation

For the implementation, I closely followed the code presented in the `Practical1.ipynb` class exercise. I refactored this base code by encapsulating operations into more engineered classes and functions to facilitate the systematic testing of different parameters during the analysis phase.

I implemented a flexible MLP classifier class, `EmailClassifier`, in PyTorch. This class allows for easy customization of key architectural parameters:

- number and size of hidden layers;
- activation function (e.g., ReLU, Sigmoid);
- dropout regularization with a specified probability.

The model outputs raw logits for two classes. I used the Adam optimizer for weight updates.

3 Hyperparameter Tuning and Analysis

I conducted a systematic series of experiments to identify the optimal model configuration. The accompanying Jupyter notebook contains an in-depth, step-by-step documentation of this process. The analysis is structured into 13 distinct experiments, where I individually modified hyperparameters such as hidden layer architecture, regularization, batch size, and activation functions. Each experiment in the notebook is detailed with its *Purpose*, *Setup*, *Expectation*, and final *Analysis*. This phase was followed by three final tests where I combined the most promising configurations to determine the winning model. This report summarizes the key findings from that comprehensive analysis.

3.1 Baseline Model

I began the analysis by establishing a baseline for comparison.

- Configuration: 2 hidden layers ([64, 32]), ReLU, batch size 32, learning rate 0.001, 20 epochs.
- Key finding: the model showed clear signs of overfitting, with the validation loss increasing after epoch 9.

3.2 Impact of Model Complexity

To address overfitting, I varied the model's architectural complexity.

- Simpler model (1 layer, [32]): this configuration yielded good results, almost completely eliminating overfitting and achieving a stable validation accuracy of about 96%.
- Wider and deeper models: I found that both increasing the width and depth of the network worsened overfitting without providing any performance benefit.

This analysis led me to conclude that a simpler architecture provided better generalization.

3.3 Impact of Regularization

I tested dropout and L2 weight decay to combat overfitting.

- Dropout (p=0.2): I found that adding light dropout proved highly effective, significantly narrowing the gap between training and validation curves.
- Stronger regularization: I observed that more aggressive regularization (e.g., dropout p=0.5) led to underfitting, preventing the model from learning effectively.

Light dropout (p=0.2) emerged as the most effective regularization technique.

3.4 Impact of Other Hyperparameters

I investigated several other training parameters.

- Number of epochs and early stopping: I confirmed that the optimal point to stop training was around 10 - 15 epochs.
- Batch size and activation function: I concluded that varying the batch size had a negligible impact, while using Sigmoid resulted in a stable but less accurate model compared to ReLU.

4 Selection of the Optimal Model

I concluded my experimental analysis with three final tests, in which I combined the most effective strategies identified. The winning configuration, from "Final Test 3", was a model with a single, regularized hidden layer. It achieved the highest and most stable validation accuracy, peaking at **95.8%**.

I selected this model as it represents a good trade-off between capacity and generalization. Its final configuration is:

- **Architecture:** 1 hidden layer with 64 neurons.
- **Activation function:** ReLU.
- **Regularization:** dropout (p=0.2).
- **Optimizer:** Adam with learning rate 0.001.
- **Batch size:** 32.
- **Training epochs:** 15.

5 Conclusion

In this project, I successfully developed an effective spam detector by systematically tuning an MLP classifier. My analysis revealed that for this particular dataset, a simpler, well-regularized model outperforms more complex architectures.

I used the final model, trained on the full dataset with the optimized configuration, to generate predictions for the unseen test set. The resulting prediction vector, representing the classification for each sample in the test set, begins as follows: `[1 1 0 1 0 0 0 1 0 1 ...]`.

6 Contributions

I completed this project individually and was responsible for all aspects of the work.

7 AI Usage Acknowledgment

I acknowledge that no AI software was used for generating the implementation code or for writing the core content of this report. I used an AI-based tool solely for the purpose of refining English grammar and ensuring a scientific writing style.