Neural Network Spam Classifier: Project Summary

Samudhyan Kahol

1. Introduction

In this project, I developed a feedforward neural network from scratch to classify emails as spam or not. I chose a neural network over other algorithms like Naive Bayes because the latter assumes that all features are conditionally independent. This assumption is not valid in the context of spam detection, where certain features such as suspicious sender addresses and specific keywords such as ’free’ are often correlated.

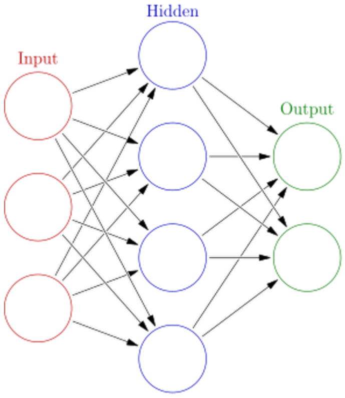


Figure 1: A simple feedforward neural network

2. Network Architecture and Methods

The model consists of an input layer with 54 features, one hidden layer containing 32 neurons employing the ReLU activation function, and an output layer with a single neuron using the sigmoid activation function to produce values between 0 and 1, so that can be interpreted as probability.

$$A_1 = \mathrm{ReLU}(XW_1 + B_1)$$

$$A_2 = \sigma\left(\mathrm{ReLU}(XW_1 + B_1)W_2 + B_2\right)$$

$$\begin{array}{ll} X \in \mathbf{M}^{m \times n} & \text{is the input feature matrix} \\ \mathrm{ReLU}(z) = \max(0, z) & \text{is the Rectified Linear Unit activation function} \\ \sigma(z) = \dfrac{1}{1 + e^{-z}} & \text{is the sigmoid activation function} \end{array}$$

Figure 2: A_1 is the hidden layer, A_2 is the output layer

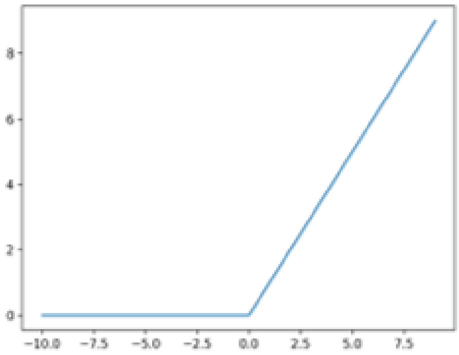


Figure 3: ReLU(z) function

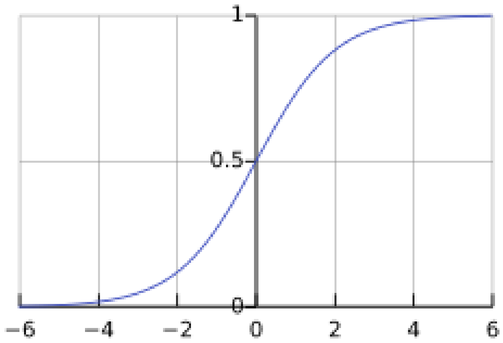


Figure 4: Sigmoid function

To initialise the weights, I used He initialisation, which is well-suited for networks with ReLU activations. This approach helps prevent vanishing gradients by maintaining an appropriate scale of weights during training.

$$\boxed{W \sim N(0, \frac{2}{\text{no. of inputs}})}$$

Figure 5: He initialisation

The training process employs forward propagation to pass inputs through the network, computing activations at each layer. The loss is calculated using binary cross-entropy, the loss measures the difference between predicted probabilities and true labels.

$$\text{BCE Loss} = -\frac{1}{m}\sum_{i=1}^{m}\left[y_i \cdot \log(\hat{y}_i) + (1 - y_i) \cdot \log(1 - \hat{y}_i)\right]$$

Where:

$$y_i \in \{0, 1\} \quad \text{(true label)}$$

$$\hat{y}_i \in (0, 1) \quad \text{(predicted probability from sigmoid)}$$

Figure 6: Binary Cross Entropy loss function

Backpropagation computes the gradients of the loss with respect to each weight and bias using the chain rule, which are then used to update the weights in the neural network via gradient descent.

$$\frac{\partial \text{Loss}}{\partial Z_2} = \hat{y} - y$$

$$\frac{\partial \text{Loss}}{\partial W_2} = A_1^T \cdot \frac{\partial \text{Loss}}{\partial Z_2}$$

$$\frac{\partial \text{Loss}}{\partial W_2} = A_1^T \cdot (\hat{y} - y)$$

Figure 7: Chain rule

$$W_{\text{new}} = W_{\text{old}} - \eta \cdot \frac{\partial \text{Loss}}{\partial W}$$

Where:

W_{new} is the updated weight

W_{old} is the current (old) weight

η is the learning rate

$\frac{\partial \text{Loss}}{\partial W}$ is the derivative of loss w.r.t weight

Figure 8: Updating the neural network weights

I implemented an exponentially decaying learning rate, which allows the network to learn quickly in the early epochs and fine-tune more gradually as training progresses. This helps the model converge smoothly and avoid overshooting the minimum loss.

$$\text{lr}_{\text{epoch}} = \text{lr}_{\text{initial}}\ e^{-k\ \text{epoch}}$$

$$\text{k is the decay constant, k} = 0.003$$

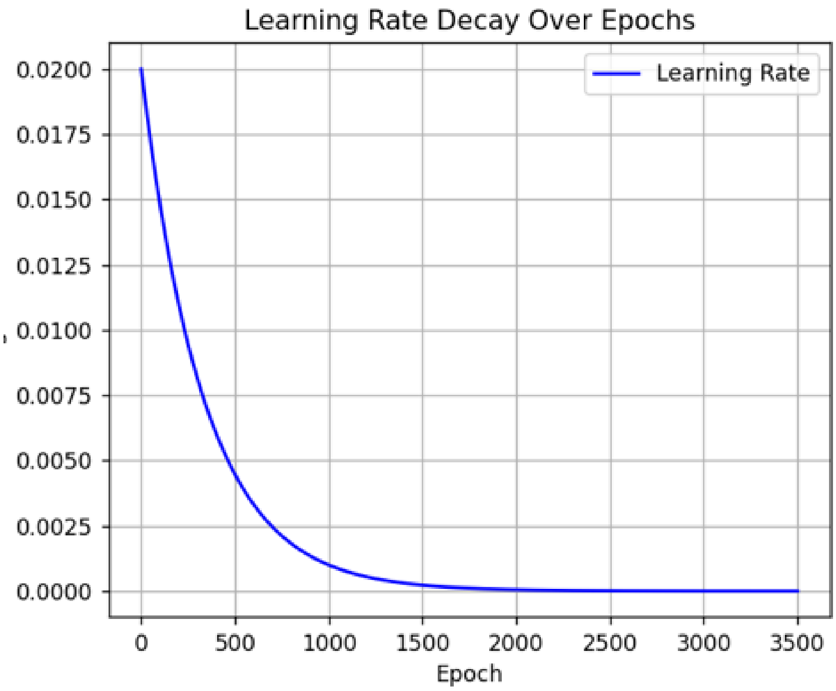


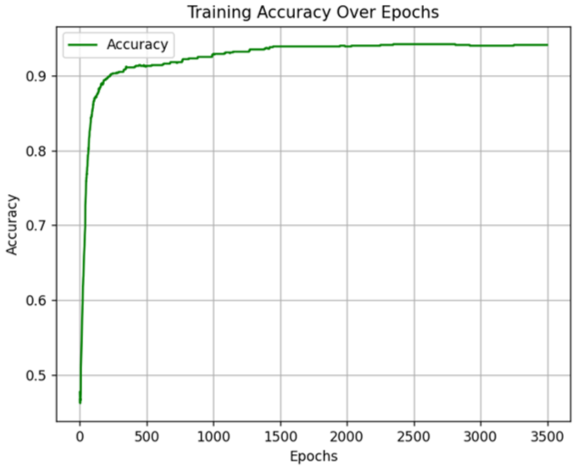
Figure 9: Learning Rate Decay

Predictions are made by classifying outputs with values above 0.5 as spam (1) and below as not spam (0).

3. Training Performance

After training for 3500 epochs with an initial learning rate of 0.02, the network achieved a training accuracy of 94.1% and a test accuracy of 92.8%. The loss decreased steadily from approximately 0.8 to 0.17 during the training period, indicating effective learning.

The relatively small gap between training and test accuracies suggests that the model generalises well to unseen data without significant overfitting.



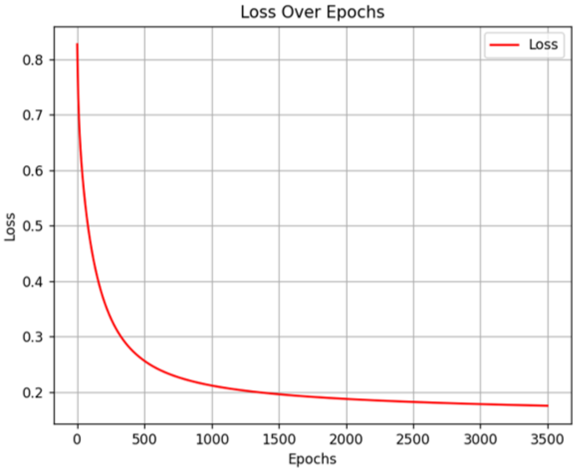


Figure 10: How training accuracy increased over iterations/epochs

Figure 11: How loss decreased over iterations/epochs

4. Strengths

- The use of ReLU activation and He initialisation helped mitigate vanishing gradient problems and improved training speed.
- Exponentially decaying learning rate enabled stable convergence.
- The network captured nonlinear dependencies between features, resulting in high classification accuracy.
- The training process and hyperparameters were carefully chosen to avoid overfitting.

5. Limitations

- The model uses only one hidden layer; deeper architectures might better capture complex patterns.
- No regularisation methods such as dropout or L2 were applied, which might affect robustness on noisier datasets.
- Hyperparameters were selected manually without systematic tuning such as grid search.

6. Future Work

Possible improvements include:

- Adding more hidden layers and experimenting with different architectures.

- Incorporating regularisation techniques to prevent overfitting.
- Implementing momentum-based optimisation algorithms.
- Conducting hyperparameter tuning via grid search or random search.
- Exploring alternative activation functions such as Leaky ReLU or tanh.

7. Conclusion

Building this neural network from scratch deepened my understanding of machine learning fundamentals, including forward and backward propagation, gradient descent optimisation, and activation functions. The project demonstrated the effectiveness of neural networks in spam classification and highlighted important considerations for training and optimising such models.