Task 1: Email Spam Classification using Naive Bayes

---

1. Introduction

Email spam detection is a classical text classification problem in machine learning. The objective of this task is to classify email messages into **Spam** or **Not Spam (Ham)** using a Naive Bayes classifier built from scratch.

Naive Bayes is a probabilistic classification algorithm based on Bayes Theorem. It assumes independence among features and is widely used in text classification tasks due to its simplicity and efficiency.

---

2. Dataset Description

The dataset consists of labeled email messages stored in a CSV file.

Classes:

- Spam
- Not Spam (Ham)

Dataset Split:

- Training Data: 80%
- Testing Data: 20%

---

3. Methodology

3.1 Text Preprocessing

The following preprocessing techniques were applied to extract meaningful features from email text:

1. Text Cleaning

- Converted all text to lowercase
- Removed punctuation
- Removed special characters

2. Tokenization

Emails were split into individual words (tokens).

3. Stopword Removal

Common words such as *the*, *is*, *and*, etc. were removed.

4. Feature Extraction

Text was converted into numerical representation using:

- Bag of Words (BoW)
- Term Frequency (TF)

These word-frequency vectors were used as input features.

3.2 Naive Bayes Model Implementation

The classifier was implemented from scratch using Python and NumPy.

Prior Probability:

Probability of each class occurring in the dataset.

Likelihood Estimation:

Multinomial distribution was used to calculate probability of words appearing in each class.

Posterior Probability:

Calculated using Bayes Theorem:

P(Class∣Words)=P(Words∣Class)P(Class)P(Words)P(Class | Words) = \frac{P(Words | Class) P(Class)}{P(Words)}P(Class∣Words)=P(Words)P(Words∣Class)P(Class)

The class with the highest posterior probability was selected.

3.3 Hyperparameter Experiments

Experiments were performed using:

- With Laplace Smoothing
- Without Laplace Smoothing
- Different vocabulary sizes
- Different preprocessing combinations

---

4. Performance Evaluation Metrics

The model was evaluated using:

- Confusion Matrix
- Accuracy
- Precision
- Recall
- F1-Score

---

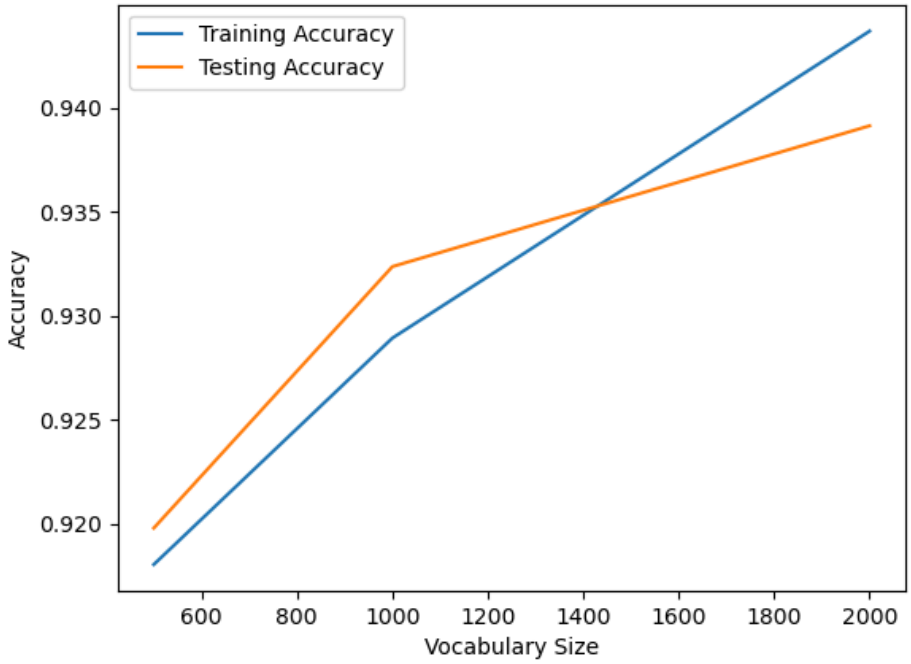
5. Results

```
Vocabulary Size: 500
Confusion Matrix:
      0    1
0  676   58
1   25  276
Accuracy: 0.9198067632850242
Precision: 0.8263473053867475
Recall: 0.916943521591638
F1 Score: 0.8692913380812897

Vocabulary Size: 1000
Confusion Matrix:
      0    1
0  683   51
1   19  282
Accuracy: 0.9323671497584541
Precision: 0.8468468468443038
Recall: 0.9368770764088477
F1 Score: 0.8895899048612436

Vocabulary Size: 2000
Confusion Matrix:
      0    1
0  689   45
1   18  283
Accuracy: 0.9391304347826087
Precision: 0.86280487804615
Recall: 0.9401993355450492
F1 Score: 0.8998410169861365
```

---

6. Plot Analysis

Training Accuracy vs Testing Accuracy



---

7. Observations

- Laplace smoothing improved model stability and prevented zero probability problems.
- Larger vocabulary improved representation but increased computation cost.
- Stopword removal reduced noise and improved classification performance.
- Bag of Words and Term Frequency both produced effective feature representations.

---

8. Conclusion

The Naive Bayes classifier proved effective for spam detection. Proper preprocessing and smoothing techniques significantly improved performance. The model demonstrated strong ability to classify email messages using probabilistic learning.

Task 2: Neural Network Training on MNIST Dataset

---

1. Introduction

Handwritten digit recognition is a common benchmark problem used to evaluate classification algorithms. The purpose of this task is to study how different activation functions and weight initialization techniques influence neural network performance.

---

2. Dataset Description

The MNIST dataset contains grayscale images of handwritten digits.

- Image Size: 28 × 28 pixels
- Number of Classes: 10
- Training Samples: 60,000
- Testing Samples: 10,000

Each image was flattened into a 784-dimensional vector.

---

3. Methodology

3.1 Data Preprocessing

- Pixel values were normalized
- Images were flattened into feature vectors
- Labels were converted into categorical format

3.2 Model Architecture

A fully connected neural network was implemented from scratch.

- Input Layer: 784 neurons
- Hidden Layer: Variable activation functions
- Output Layer: 10 neurons

3.3 Activation Functions Tested

- Sigmoid
- Tanh
- ReLU
- Leaky ReLU

3.4 Weight Initialization Techniques

- Zero Initialization
- Random Initialization
- Xavier Initialization

---

4. Experimental Results

Final Test Accuracy

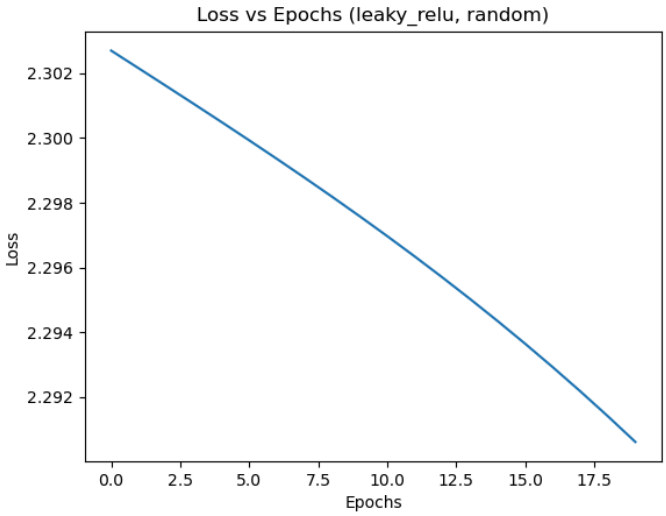
Activation Function	Initialization	Test Accuracy
Sigmoid	Zero	0.1135
Sigmoid	Random	0.1135
Sigmoid	Xavier	0.3435
Tanh	Zero	0.1135
Tanh	Random	0.5659
Tanh	Xavier	0.7198
ReLU	Zero	0.1135
ReLU	Random	0.4692
ReLU	Xavier	0.6463

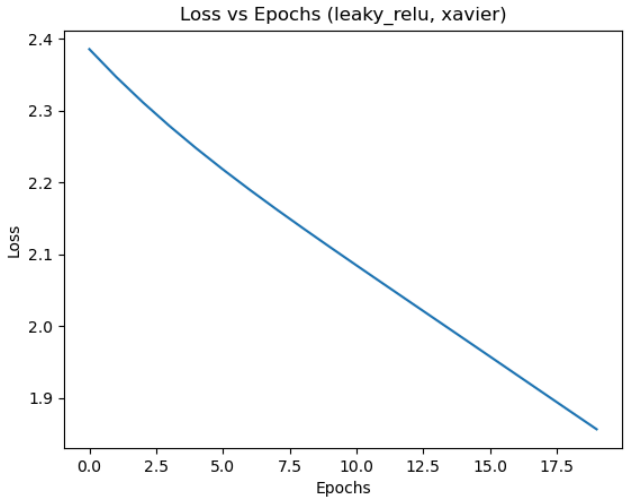
Leaky ReLU	Zero	0.1135
Leaky ReLU	Random	0.3723
Leaky ReLU	Xavier	0.6495

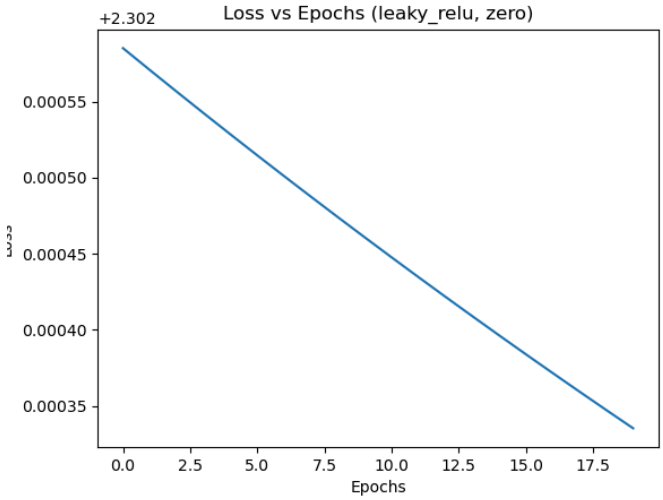
---

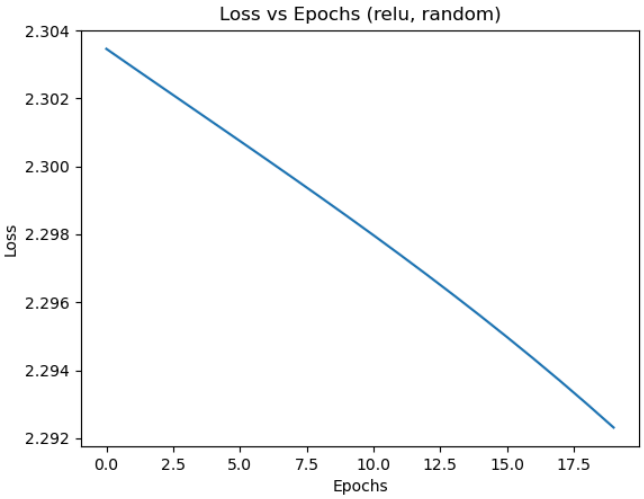
5. Plot Analysis

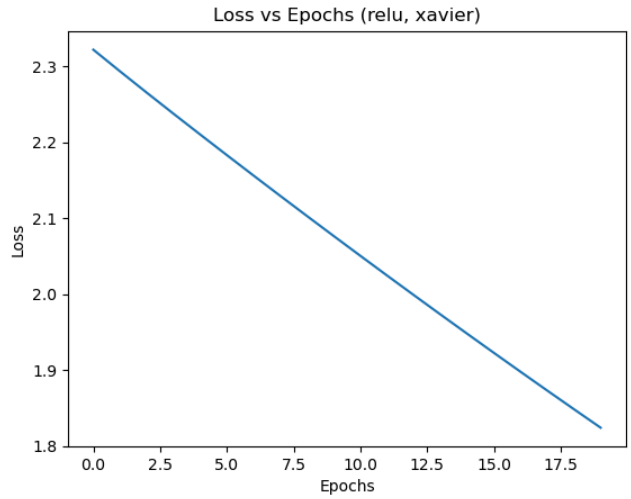
Training Loss vs Epochs

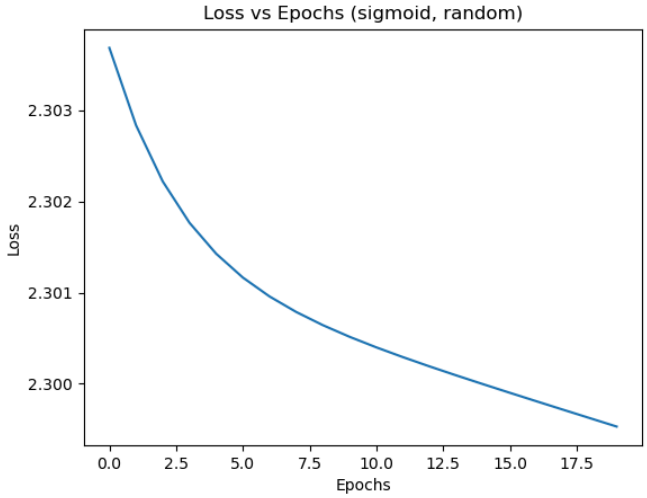


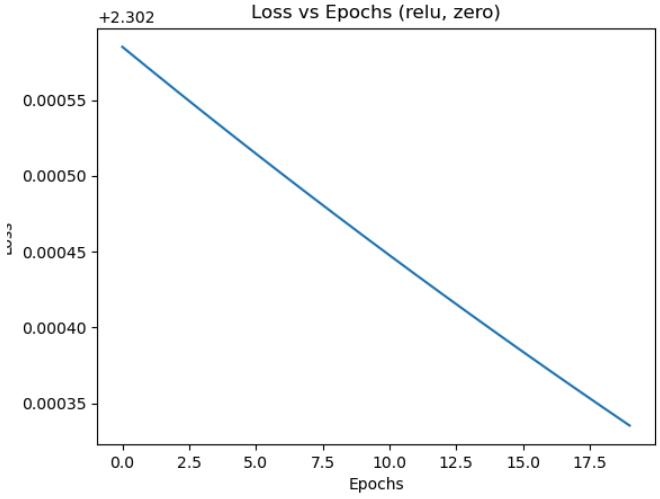

Loss vs Epochs (leaky_relu, xavier)
Loss
Epochs
2.4
2.3
2.2
2.1
2.0
1.9
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

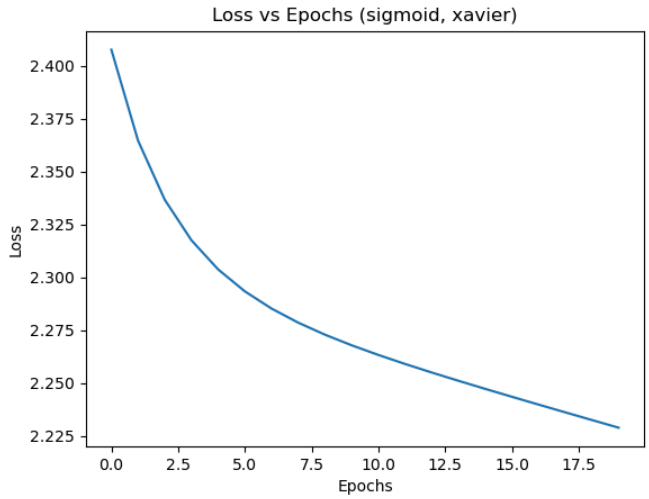

Loss vs Epochs (leaky_relu, zero)
+2.302
Epochs
0.00055
0.00050
0.00045
0.00040
0.00035
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

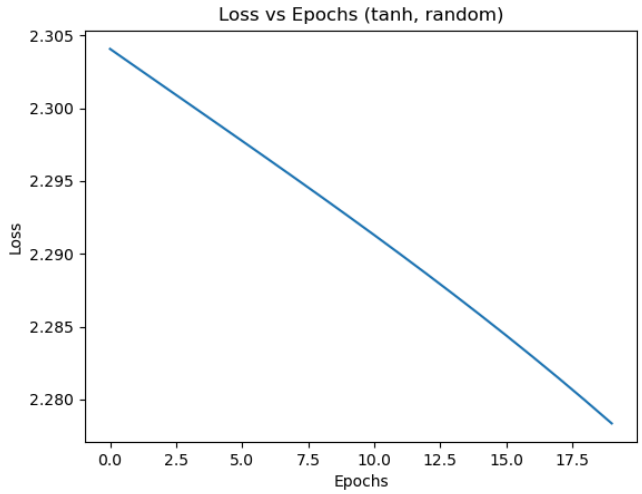

Loss vs Epochs (relu, random)
Loss
Epochs
2.304
2.302
2.300
2.298
2.296
2.294
2.292
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

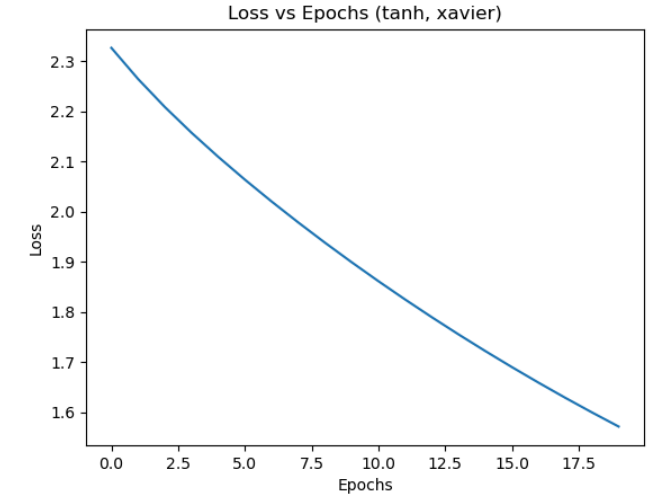

Loss vs Epochs (relu, xavier)
Loss
Epochs
2.3
2.2
2.1
2.0
1.9
1.8
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

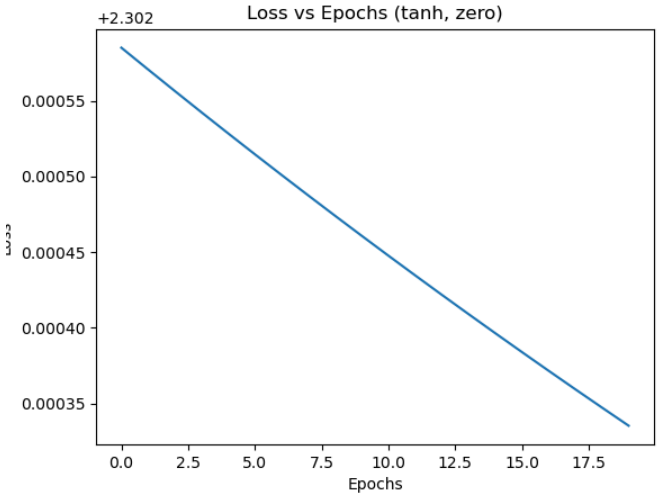

Loss vs Epochs (sigmoid, random)
Loss
Epochs
2.303
2.302
2.301
2.300
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

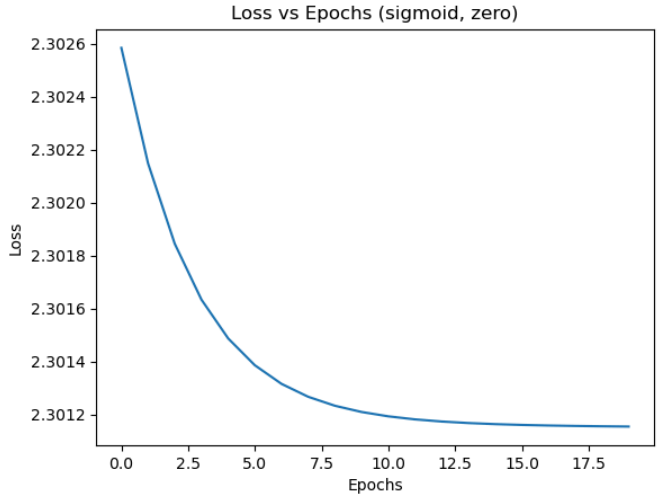

Loss vs Epochs (relu, zero)
+2.302
Epochs
0.00055
0.00050
0.00045
0.00040
0.00035
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Loss vs Epochs (sigmoid, xavier)
Loss
Epochs
2.400
2.375
2.350
2.325
2.300
2.275
2.250
2.225
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Loss vs Epochs (tanh, random)
Loss
Epochs
2.305
2.300
2.295
2.290
2.285
2.280
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Loss vs Epochs (tanh, xavier)
Loss
Epochs
2.3
2.2
2.1
2.0
1.9
1.8
1.7
1.6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5





---

6. Observations

Zero Initialization

Produced extremely poor results because all neurons learned identical weights.

Random Initialization

Improved performance but training was slower and less stable.

Xavier Initialization

Provided best gradient flow and improved convergence.

Activation Function Comparison

- Sigmoid suffered from vanishing gradients.
- Tanh performed significantly better.
- ReLU showed faster convergence.
- Leaky ReLU reduced dead neuron problem.

---

7. Best Performing Configuration

- Activation Function: **Tanh**
- Initialization: **Xavier**
- Accuracy: **71.98%**

---

8. Conclusion

The experiment shows that weight initialization and activation functions significantly impact neural network performance. Xavier initialization consistently improved learning. Tanh activation function achieved the highest classification accuracy.