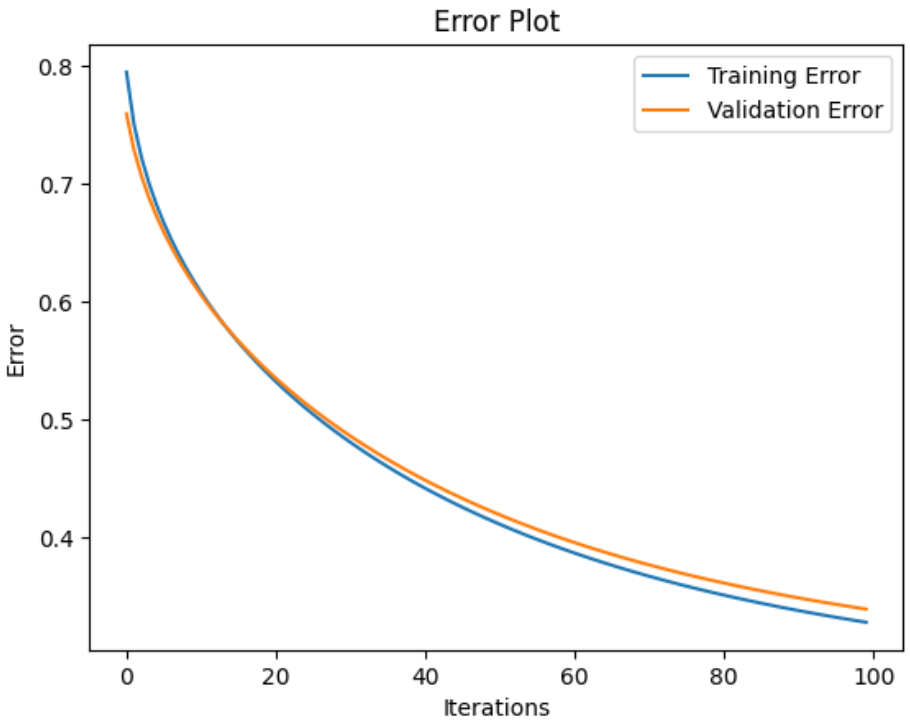
1- Completing the parts of the neural network

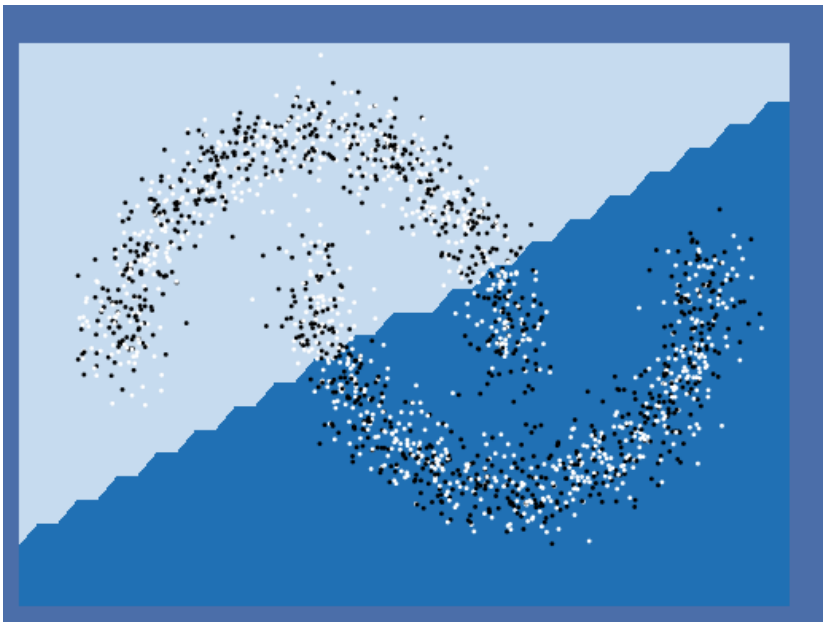
Model Architecture:

```
+-----+
| MLP |
+-----+
Input Shape: 2
+------------------------+------------+--------------+
| Layer Type             | Parameters | Output Shape |
+------------------------+------------+--------------+
| Flatten                | 0          | (2,)         |
| MyLinear               | 6          | (2,)         |
| Activation (MySigmoid) | 0          | (2,)         |
+------------------------+------------+--------------+
Total Parameters: 6
```

Train and Validation Curve:



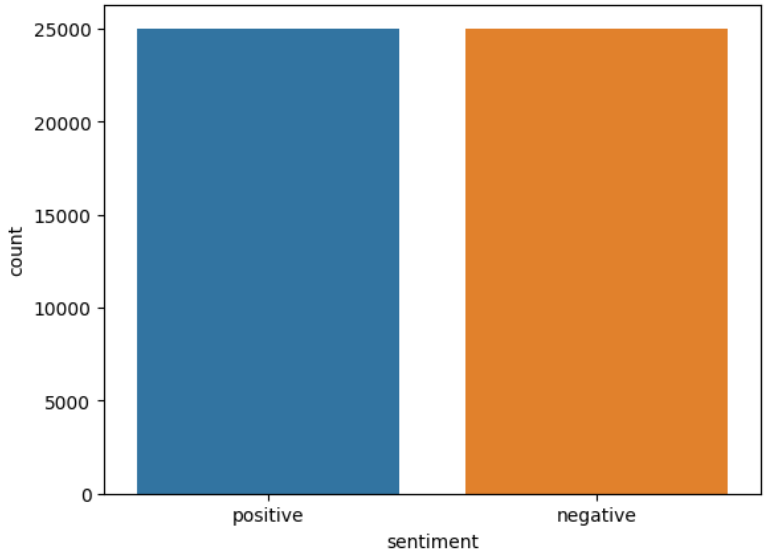
Decision Boundary Plot



2 - Dataset Preparation

Dataset:

IMDB dataset having 50K movie reviews for natural language processing or Text analytics.This is a dataset for binary sentiment classification containing substantially more data than previous benchmark datasets.



Methods:

Stemming and lemmatization are two common techniques used in natural language processing (NLP) to reduce words to their base or root form, which can help to improve text analysis and information retrieval.

Stemming involves removing the suffixes from words to obtain their stem or root form. For example, the stem of the word "running" would be "run". Stemming is a simple and fast method that can be used to reduce the number of unique words in a text, but it can also result in some inaccuracies, as the stem may not always be a valid word.

Lemmatization, on the other hand, involves reducing words to their base form or lemma, which is a valid word that can be found in a dictionary. For example, the lemma of the word "running" would be "run", but the lemma of the word "is" would be "be". Lemmatization is a more accurate method than stemming, but it can also be slower and more computationally expensive.

Preprocessing steps:

First, stemming and stop words removal is applied to help to reduce the dimensionality of the feature space by eliminating variations of the same word and common words that do not carry much meaning. This can help to improve the efficiency of the feature extraction process and reduce the risk of overfitting.

Second, TfidfVectorizer applied to help to weigh the importance of each term in the document based on its frequency and inverse document frequency (IDF). This can help to identify the most relevant and distinctive terms that are characteristic of a particular document or class of documents.

Finally, using these techniques together can help to improve the performance of text classification and information retrieval systems by reducing noise and increasing the signal-to-noise ratio in the feature space.

Data Normalization, Why?

If we don't normalize your data between 0 and 1, it can lead to several issues in machine learning models.

- normalization helps to ensure that all features contribute equally to the model, regardless of their scale or units. If some features have a much larger range of values than others, they may dominate the model and make it difficult for other features to have an impact.

- normalization can help to improve the convergence of optimization algorithms, such as gradient descent. If the features are not normalized, the optimization process may take longer to converge or may not converge at all.

- normalization can help to prevent numerical instability and overflow issues that can occur when working with very large or very small values.

3 - MLP in torch

Part 1 Model Architecture:

Layer (type)	Output Shape	Param #
MyLinear-1	[-1, 1024, 8000]	24,008,000
MyReLU-2	[-1, 1024, 8000]	0
MyLinear-3	[-1, 1024, 5000]	40,005,000
MyReLU-4	[-1, 1024, 5000]	0
MyLinear-5	[-1, 1024, 500]	2,500,500
MyReLU-6	[-1, 1024, 500]	0
MyLinear-7	[-1, 1024, 1]	501
Sigmoid-8	[-1, 1024, 1]	0

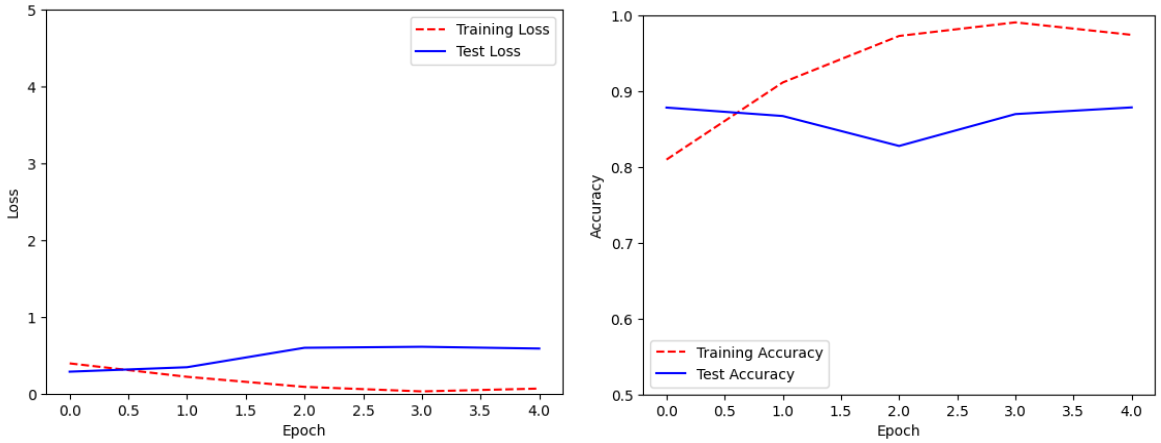
Total params: 66,514,001
Trainable params: 66,514,001
Non-trainable params: 0

Input size (MB): 11.72
Forward/backward pass size (MB): 210.95
Params size (MB): 253.73
Estimated Total Size (MB): 476.40

Results:

Classification Report:

	precision	recall	f1-score	support
0.0	0.87	0.89	0.88	6291
1.0	0.89	0.86	0.88	6209
accuracy			0.88	12500
macro avg	0.88	0.88	0.88	12500
weighted avg	0.88	0.88	0.88	12500



Part 2 Weights:

If we initialize all the weights of a neural network to 0, the network will not be able to learn effectively. This is because all the neurons in the network will have the same output, and the gradients during backpropagation will be the same for all the weights. As a result, the weights will not be updated during training, and the network will not be able to learn any useful features from the input data.

To avoid this problem, it is common practice to initialize the weights of a neural network randomly, using techniques such as Xavier initialization or He initialization. These methods ensure that the weights are initialized with small, non-zero values that allow the network to learn effectively.

Part 3 Learning Rate:

By testing different learning rates, we can find the optimal learning rate that allows the network to converge quickly and achieve good performance. This can be done by training the network with different learning rates and evaluating the performance on a validation set. We can then choose the learning rate that gives the best performance.

Learning Rate = 0.01
Results

:

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	6291
1.0	0.50	1.00	0.66	6209
accuracy			0.50	12500
macro avg	0.25	0.50	0.33	12500
weighted avg	0.25	0.50	0.33	12500

According to the results If the learning rate is too high, the weight updates can be too large and the network may fail to converge or even diverge. This can lead to unstable training and poor performance.

Learning Rate = 0.001
Results

	precision	recall	f1-score	support
0.0	0.87	0.89	0.88	6291
1.0	0.89	0.86	0.88	6209
accuracy			0.88	12500
macro avg	0.88	0.88	0.88	12500
weighted avg	0.88	0.88	0.88	12500

Part 4 Activation Functions:

Sigmoid:

	precision	recall	f1-score	support
0.0	0.00	0.00	0.00	6291
1.0	0.50	1.00	0.66	6209
accuracy			0.50	12500
macro avg	0.25	0.50	0.33	12500
weighted avg	0.25	0.50	0.33	12500

Tanh:

	precision	recall	f1-score	support
0.0	0.79	0.92	0.85	6291
1.0	0.90	0.76	0.82	6209
accuracy			0.84	12500
macro avg	0.85	0.84	0.84	12500
weighted avg	0.85	0.84	0.84	12500

ReLU:

	precision	recall	f1-score	support
0.0	0.87	0.89	0.88	6291
1.0	0.89	0.86	0.88	6209
accuracy			0.88	12500
macro avg	0.88	0.88	0.88	12500
weighted avg	0.88	0.88	0.88	12500

Sigmoid and tanh activation functions were popular choices for hidden layers in neural networks in the past, but they have some limitations compared to other activation functions that have been developed more recently.

One limitation of sigmoid and tanh activation functions is that they can suffer from the vanishing gradient problem, which can make it difficult for the network to learn effectively. This occurs because the gradients of these functions become very small as the input values move away from zero, which can cause the gradients to vanish during backpropagation.

Another limitation of sigmoid and tanh activation functions is that they are not zero-centered, which can make it harder for the network to learn effectively. This is because the output of the activation function is always positive, which can cause the gradients to always be positive or always be negative, depending on the sign of the weights.

More recent activation functions such as ReLU (Rectified Linear Unit) and its variants (e.g. Leaky ReLU, ELU) have been developed to address these limitations. ReLU is zero-centered and does not suffer from the vanishing gradient problem, which makes it a good choice for hidden layers in neural networks

Leaky ReLU

	precision	recall	f1-score	support
0.0	0.87	0.89	0.88	6291
1.0	0.89	0.86	0.87	6209
accuracy			0.88	12500
macro avg	0.88	0.88	0.88	12500
weighted avg	0.88	0.88	0.88	12500

Leaky ReLU is a variant of the popular activation function ReLU (Rectified Linear Unit) that addresses one of its limitations. The main advantage of Leaky ReLU over ReLU is that it avoids the "dying ReLU" problem, which can occur when the ReLU function outputs zero for all negative input values. This can happen when the weights of the network are initialized in a way that causes the ReLU units to always output negative values, or when the learning rate is too high and causes the weights to update in a way that makes the ReLU units always output negative values.

Here we achieved 88% accuracy with ReLU and leaky Leaky ReLU.

Part 5 Batch Size:

It is important to note that the optimal batch size can depend on the specific problem and network architecture, so it may be necessary to experiment with different batch sizes for each new problem**.**

Batch size = 16

	precision	recall	f1-score	support
0.0	0.92	0.81	0.86	6291
1.0	0.82	0.93	0.87	6209
accuracy			0.87	12500
macro avg	0.87	0.87	0.87	12500
weighted avg	0.87	0.87	0.87	12500

Batch size = 256

	precision	recall	f1-score	support
0.0	0.87	0.90	0.88	6291
1.0	0.89	0.86	0.88	6209
accuracy			0.88	12500
macro avg	0.88	0.88	0.88	12500
weighted avg	0.88	0.88	0.88	12500

Advantages of big batch sizes:

• Faster convergence: With a larger batch size, the optimization algorithm can take larger steps towards the optimal weights, leading to faster convergence.
• Better hardware utilization: Larger batch sizes can make better use of hardware resources such as GPUs, as they can process more examples in parallel.
• More stable gradients: With a larger batch size, the gradient estimates are more stable and less noisy, which can lead to more stable training.

Advantages of small batch sizes:

• Generalization: With a smaller batch size, the network sees more diverse examples, which can help it generalize better to new data.
• Reduced overfitting: Smaller batch sizes can help prevent overfitting by introducing more randomness into the training process.

• Lower memory requirements: With a smaller batch size, less memory is required to store the examples and gradients, which can be important for large datasets or limited hardware resources.

Here we achieved better results with a bigger batch size.(256)