Deep Learning Assignment 1

Abstract

This paper explores the implementation and evaluation of Perceptron and Multilayer Perceptron (MLP) models, employing both Batch Gradient Descent and Stochastic Gradient Descent techniques.

1.perceptron

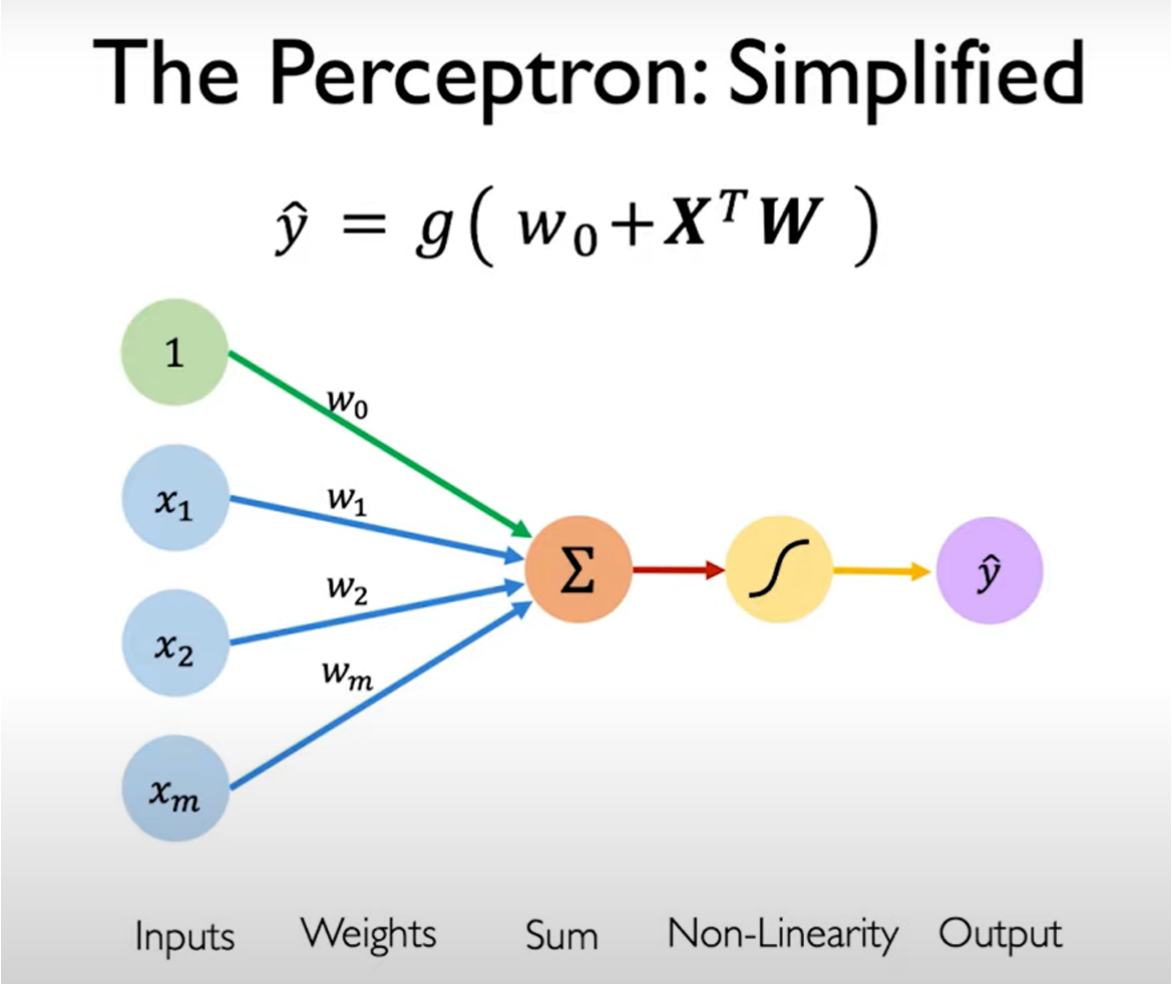
Theoretical analysis of perceptron

Perceptron, also known as "artificial neuron" or "naive perceptron," was proposed by Frank Rosenblatt in 1957. As an early algorithm in the realm of neural networks, delving into its learning process can assist us in gaining a better understanding of certain workings within neural networks.

Theoretical Analysis of the Learning Process of the Perceptron

The perceptron, a fundamental concept in neural network theory, was introduced by Frank Rosenblatt in 1957. It serves as a foundational model for understanding neural network learning processes.

The perceptron is a binary linear classifier that maps input vectors to a single binary output. It consists of a set of input nodes, each associated with a weight, and an activation function that determines the output based on the weighted sum of inputs.



Learning Process

The learning process of the perceptron involves adjusting the weights of its connections to minimize classification errors. This is typically achieved through a form of supervised learning, where the perceptron is presented with training examples, and its weights are updated iteratively based on the errors made in classifying these examples.

forward component

The perceptron decision is based on these formulas:

$$f(x) = sign(w \cdot x + b)$$

$$\text{sign}(x) = \begin{cases} -1 & \text{if } x < 0 \\ +1 & \text{if } x >= 0 \end{cases}$$

Inputs (x): The features or data points provided to the perceptron.

Weights (w): Coefficients determining the importance of each input.

Bias (b): An extra input (always 1) with its own weight, allowing the activation function to shiff, fitting the data better.

Activation Function(sign function): Decides whether the neuron should activate, typically a step function for basic perceptrons.

learning way

For epoch = 1 ... T:

Compute the predictions of Perceptron of the whole training set. then compute the gradient of the loss function with respect to w and b. Update w ← w − learningrate ∗ gradient respect to w and Update b ← b − learningrate ∗ gradient respect to b.

Loss part

Since we want to classify all points correctly, a natural idea is to directly use the total number of misclassified points as a loss function , so the perceptron loss function is defined as

$$L(\mathbf{w}) = -\sum_{i=1}^{N} y_i(\mathbf{w}\cdot\mathbf{x}_i + b)(where(\mathbf{w}\cdot\mathbf{x}_i + b) < 0)$$

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b) = -\sum_{where(\mathbf{w}\cdot\mathbf{x}_i+b)<0}^{N} y_i x_i$$

$$\nabla_{\mathbf{w}} L(\mathbf{w}, b) = -\sum_{where(\mathbf{w}\cdot\mathbf{x}_i+b)<0}^{N} y_i$$

Detail of train

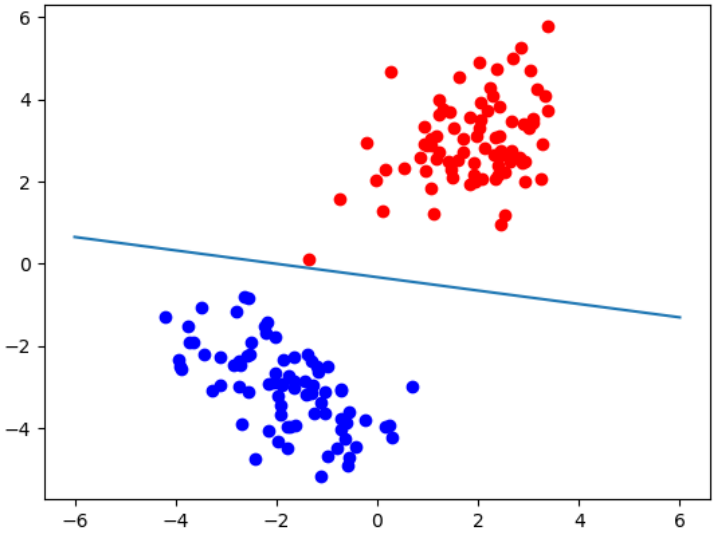
max_epochs	**1000**	
learning_rate	0.001	
initial bias	0	
initial weight	[0,0]	

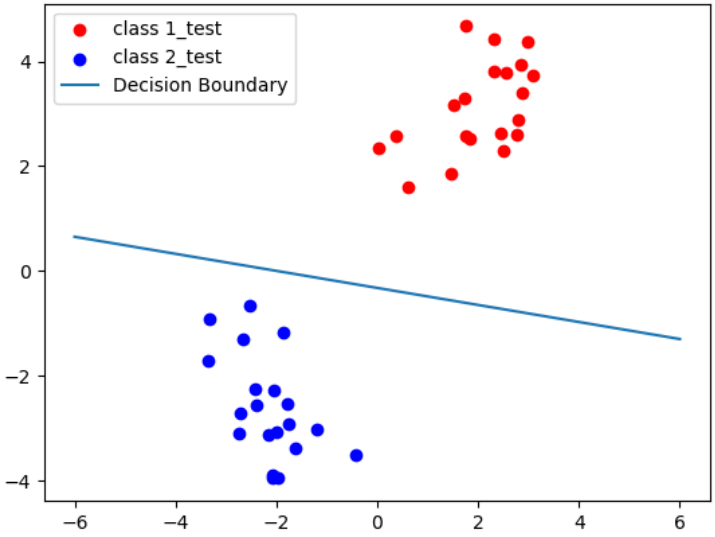
We generate a dataset of points in R^2 by deffning two Gaussian distributions and sampling 100 points from each. This results in a total of 200 points, with 100 points from each distribution. We then split the dataset, allocating 80 points per distribution for training (160 in total) and reserving 20 points for testing (40 in total).

```
mean1 = [2, 3]
covariance1 = [[1, 0.5], [0.5, 1]]
mean2 = [-2, -3]
covariance2 = [[1, -0.5], [-0.5, 1]]
```

This is the parameter of the initial data set.

Result



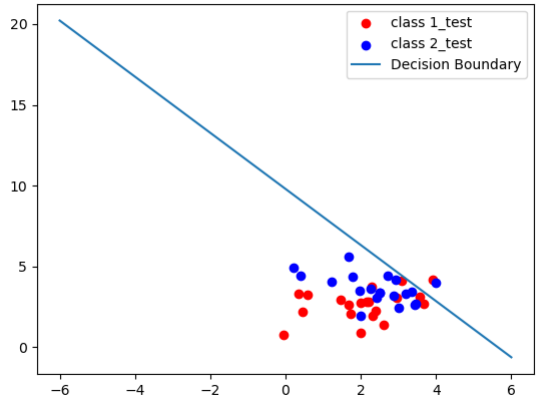


The classifcation accuracy on the test set is 1 .However, after many tests, there may be one or two points of error (that is, accuracy is not 1).

Mean nearly

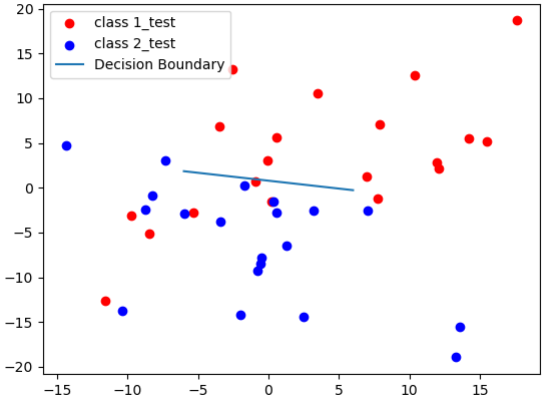
```
mean1 = [2, 3]
covariance1 = [[1, 0.5], [0.5, 1]]
mean2 = [2.5, 3.5]  # Modified mean closer to mean1
covariance2 = [[1, -0.5], [-0.5, 1]]
```

The classification is particularly poor because the points are so close together .The classifcation accuracy on the test set is 0.52-0.65



Increase variance

```
Increase variance
    accurary: 0.925 for [[4, 2], [2, 4]] and [[4, -2], [-2, 4]]
    accurary: 0.9 for [[10, 5], [5, 10]] and  [[10, -5], [-5, 10]]
    accurary: 0.225 for [[50, 25], [25, 50]] and [[50, -25], [-25, 50]]
```



At the beginning, the variance is not large, and the points of the same class are not far apart. Later, the variance is large, and the points of different classes are still close, and then the accuracy of the classification is significantly decreased

Limitations

Despite its significance, the perceptron has limitations. It can only learn linearly separable patterns and may not converge if the data is not linearly separable(In two of these cases, the variance is too large or the mean is close). Additionally, it cannot learn complex patterns or perform nonlinear transformations without additional layers or techniques such as multilayer perceptrons or kernel methods.

2.the mutli-layer perceptron

Theoretical analysis of forward

In an MLP, the input data is passed through multiple layers of neurons, each of which applies a set of weights and a nonlinear activation function . The output of each neuron is then passed as input to the neurons in the next layer .

$$x\tilde{}(l) = W(l)x(l-1) + b(l)$$

This is the linear layer, multiplying the incoming with the weight, and then adding the bias, passing in the subsequent layer.

$$x(l) = max(0, x\tilde{}(l))$$

This is the ReLU unit, then Applies the ReLU activation function element-wise to the input.

$$x(N) = \frac{e^{x\tilde{}(N)}}{\sum_{j=1}^{k} e^{x\tilde{}(N)}}$$

This is the softmax layer Applies the softmax function to the input to obtain output probabilities. pay attention subtract x_max to reduce computer and Prevent overflow.

$$L = -sum(y_i * log(p_i + 1e - 10)),$$

where p is the softmax probability of the correct class y.

This is the compute the cross entropy loss L ,Adding a minimal constant to the probability prevents mathematical errors and does not affect the result.

Back propagation:

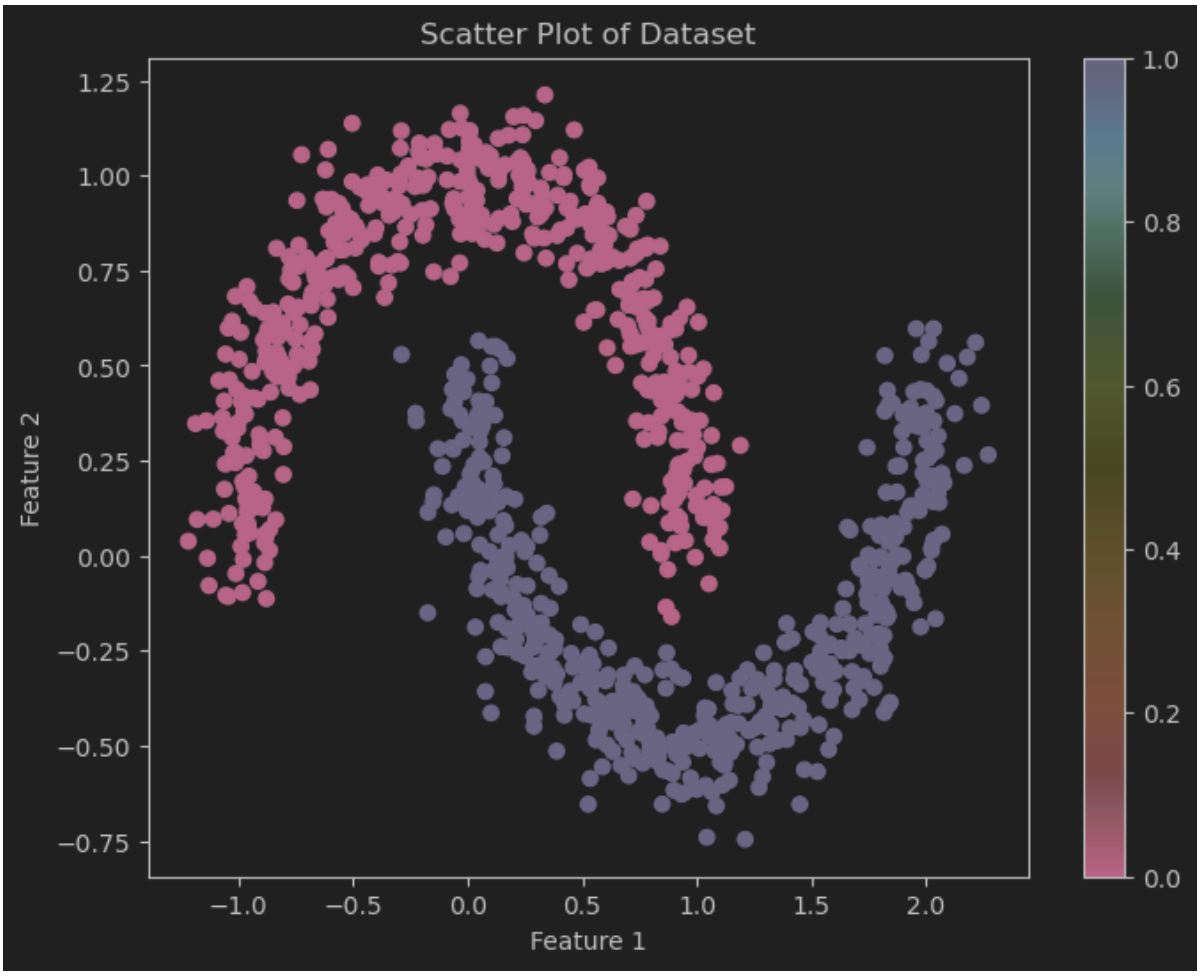
Starting at the output layer, calculate the gradient of each parameter to the loss function.
The gradient is propagated backwards to each layer using the chain rule to calculate the gradient of the parameters of the hidden and input layers.
Update parameters to reduce loss functions. This is usually done by gradient descent or a variant of it.

```
For softmax output followed by cross-entropy loss, the gradient simplifies to: p
- y.

For ReLU, the gradient is Gradient is 1 for x > 0, otherwise 0 ,which mean input
is significant

For linear, the gradient of weight is calculated by taking the dot product of the
transposed input matrix and Backward incoming parameter, the gradient of bias is
the mean of  Backward incoming parameter.
```

Dataset

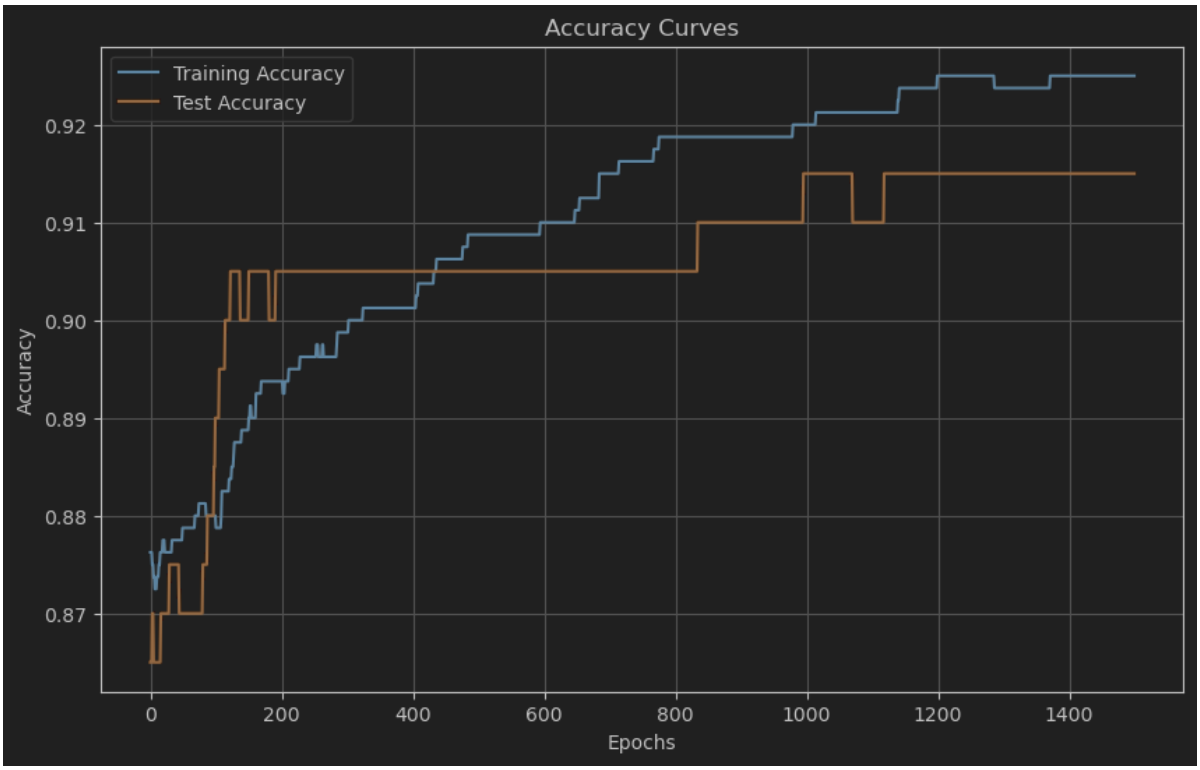
In this second part of Assignment I you're asked to implement a multi-layer perceptron using numpy. Using scikitlearn and the make moons method , create a dataset of 1, 000 two-dimensional points. And the unique thermal coding and sub-training test set are carried out.

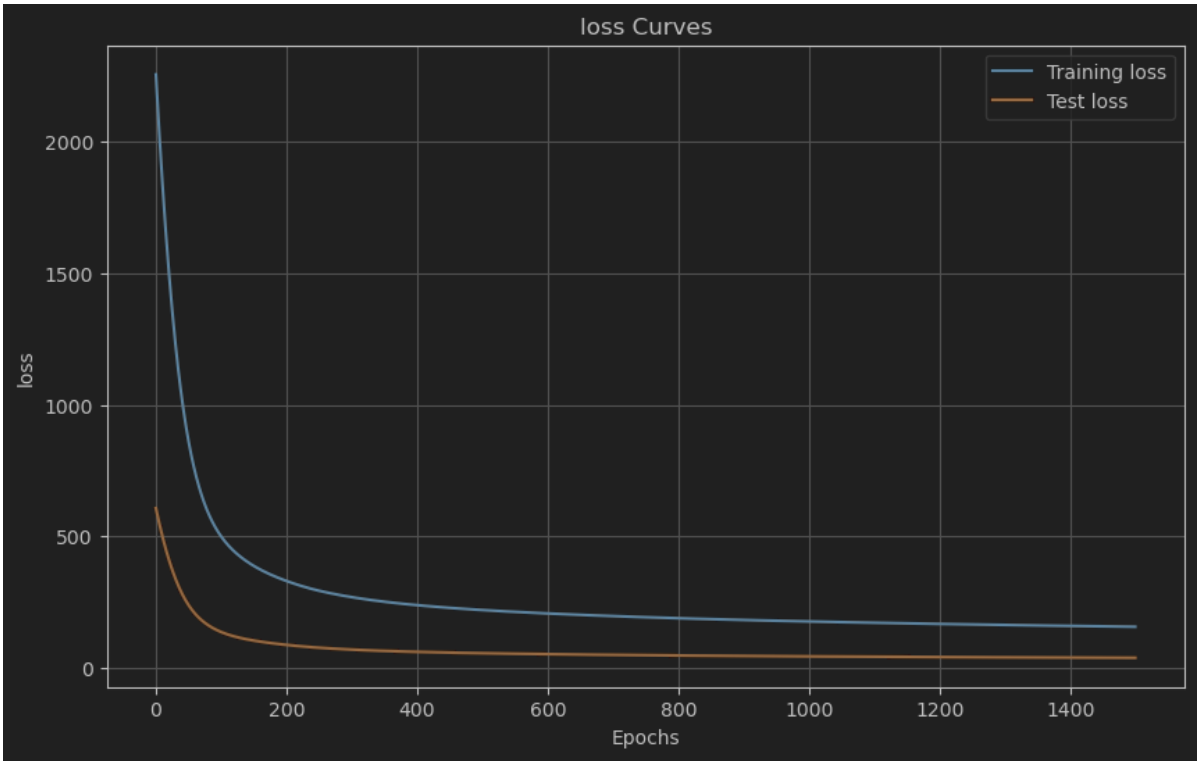
```
X, y = make_moons(n_samples=num_samples, noise=0.1, random_state=42)
one_hot_encoder = OneHotEncoder(categories='auto')
y_one_hot = one_hot_encoder.fit_transform(y.reshape(-1, 1)).toarray()
# Split data into training and testing sets
X_train, X_test, y_train, y_test = train_and_test_data_split(X, y_one_hot)
```



The default parameters are listed below:

```
DNN_HIDDEN_UNITS_DEFAULT = '20'
LEARNING_RATE_DEFAULT = 1e-2
MAX_EPOCHS_DEFAULT = 1500
EVAL_FREQ_DEFAULT = 10
num_samples = 1000
num_features = 10
num_classes = 2
```



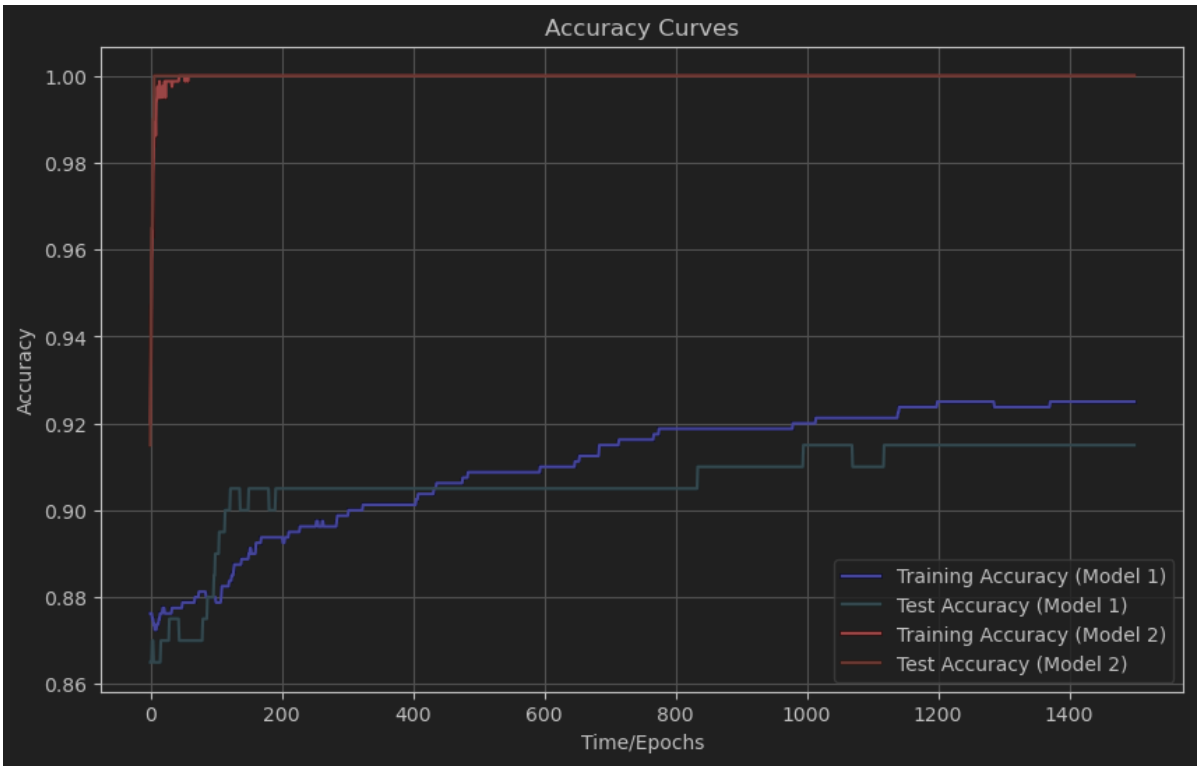


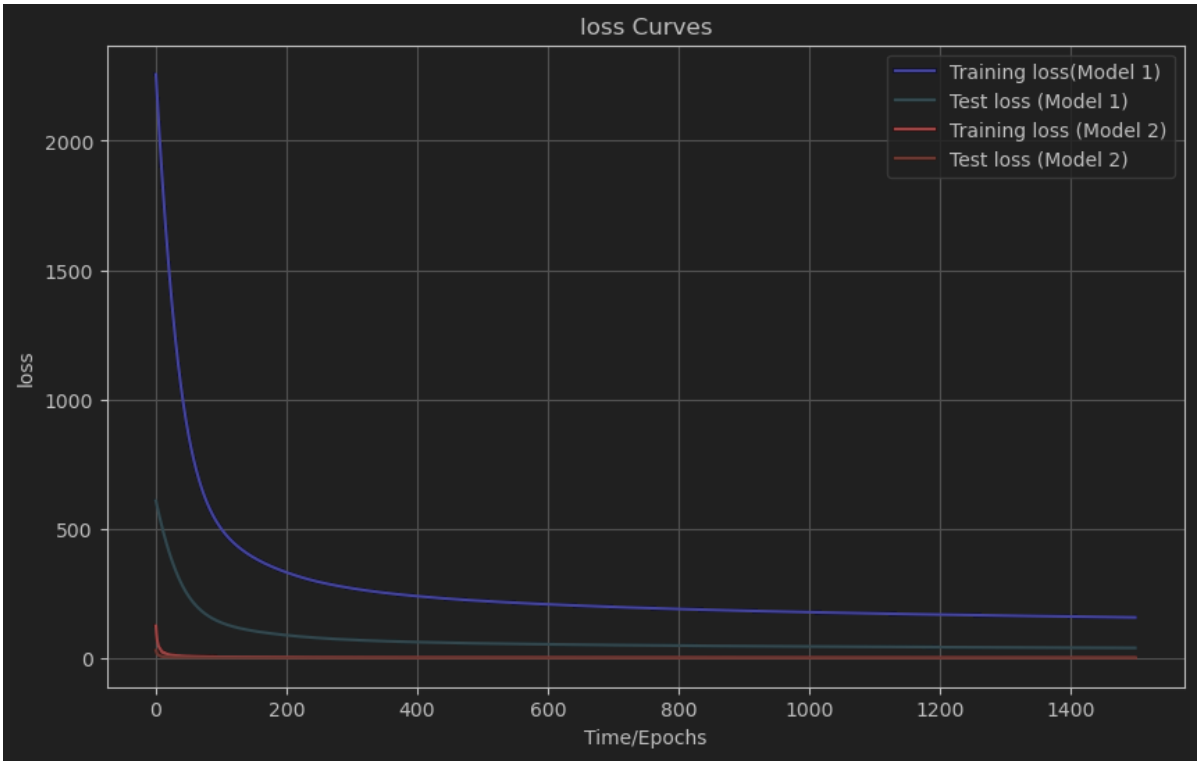
Accuracy: Step: 1120, Loss: 37.33615475219883, Accuracy: 0.915 In the following, the accuracy does not change, and the loss keeps decreasing. When the step is 1500, it becomes 34.7201833876942.If the step is adjusted to 7000 steps, there is a possibility of Accuracy becoming 1, which is related to the data generated by random seeds

3.stochastic gradient descent

A parameter of allows the user to use stochastic gradient descent is batch(DEFAULT is True) , true mean batch gradient descent ,false mean

stochastic gradient descent .When batch is false, another parameter batch_size(DEFAULT is 1) is using.





Model 1 is batch gradient descent

Model 2 is stochastic gradient descent (batch_size=1)

```
Step: 0, Loss: 39.60007698942094, Accuracy: 0.915
Step: 10, Loss: 6.73647457906465, Accuracy: 1.0
Step: 20, Loss: 3.2797773060103816, Accuracy: 1.0
```

Obviously, it can be seen that the stochastic gradient descent can find the optimal solution within a few steps, with the accurary being 1. By contrast, the stochastic gradient descent is slow in convergence and has less good improvement effect.

batch_size	cost_time	the step of accuracy become 1(One of the test results)
1	141.86 s	About 10 steps(19)
2	80.75 s	20 -30(22)
5	37.09 s	40 -50(39)
10	19.01 s	100-150 (114)
20	10.64 s	150 -200 (168)
50	4.88 s	500 -1000 (881)
100	3.23 s	1000 -1500 (no ,finally Accuracy: 0.995)
200	2.12 s	1000 -1500 (1106)
500	1.66 s	1500+(no ,finally Accuracy: 0.915)

batch gradient descent time cost is 1.30 s

It is obvious that the larger the batch_size, the shorter the training time, but the accuracy cannot be reduced to 1 in a shorter number of steps. One obvious reason is that the number of times the weights are modified according to the gradient after back propagation is reduced.

Some other findings

MLP doesn't necessarily work for different data sets, the quality of the data is also important, and if you just randomly generate a bunch of meaningless Xy, there is a good chance that the classification effect will be as good as random. At the very beginning of data generation, I encountered this problem, so I doubted life. Data noise also affects the highest accuracy