# Deep Learning and Practice Report
# Lab2: Back Propagation

## 1 Introduction

Back propagation is an indispensable step when training deep neural networks. During back propagation, we perform gradient descent to update the model parameters of each layer so as to reduce the error term between the model outputs and ground truth labels. In this lab, we are asked to implement the forward pass and back propagation of a simple neural network with only two hidden layers. Deep learning frameworks such as Tensorflow and Pytorch are prohibited, we are only allowed to use Numpy and other python standard libraries.

## 2 Experiment Setups

### 2.1 Sigmoid Function

Sigmoid function is an activation function widely used in deep learning models. The function provides non-linearity, making the model able to fit data with more complexity. The curve of this function and its derivative is shown in figure 1. As we can see, this function has the property to map all numbers into the range between $0$ and $1$. Also, its derivative has small value throughout the whole number line, and the maximum is $0.25$ when $x$ is equal to $0$.

$$\sigma(x) = \frac{1}{1+e^x}$$
$$\sigma'(x) = \sigma(x)1 - \sigma(x)$$

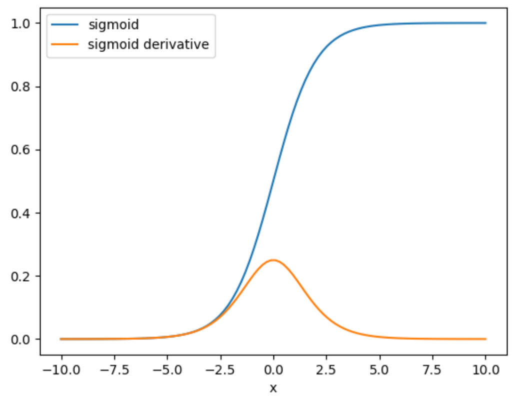


**Figure 1:** Curves of sigmoid function and its derivative.

## 2.2 Neural Network

In this lab, I build a neural network to classify data points into two classes. The model consists of one input layer, two hidden layers and one output layer. Each layer is followed by an activation function (Sigmoid, ReLU). The model has two inputs and a single output, where the output should be $0$ when the input belongs to the first class, and $1$ when the input belongs to the second class. In details, I set the model threshold to $0.5$. That is, the sample will be classified as the first class when the output is smaller than $0.5$, and the second class when the output is greater than $0.5$.

## 2.3 Back propagation

During back propagation, I first calculate the loss value between the model outputs and ground truth labels, and then perform gradient descent to update the model weights from the last layer to the front recursively. Specifically, I use **m**ean **s**quared **e**rror (MSE) as the loss function, and its objective is to minimize the error between the predicted value and the ground truth. The MSE is computed as follows,

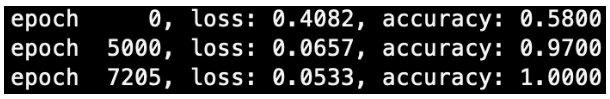
$$MSE = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

where $n$ is the number of training samples, $y_i$ and $\hat{y}_i$ denote the $i$-th ground truth and model output respectively.

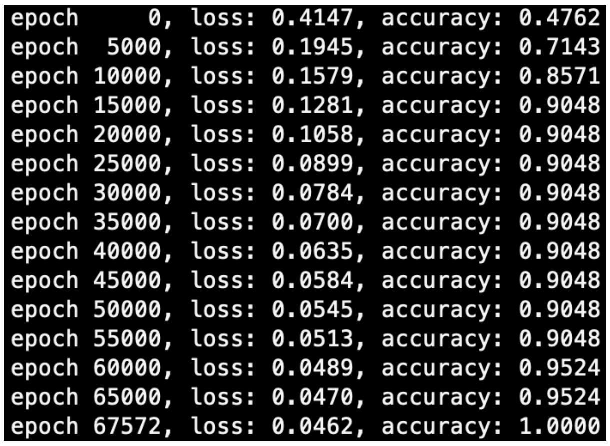
# 3 Results of Testing

The testing results include two datasets. The first one is linear, with 100 randomly generated data points. The second one is xor, with 21 data points. The data distribution is shown in figure 4.

## 3.1 Screenshot and Comparison Figure



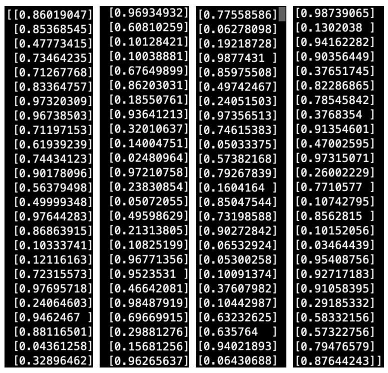
```
epoch     0, loss: 0.4082, accuracy: 0.5800
epoch  5000, loss: 0.0657, accuracy: 0.9700
epoch  7205, loss: 0.0533, accuracy: 1.0000
```



```
epoch     0, loss: 0.4147, accuracy: 0.4762
epoch  5000, loss: 0.1945, accuracy: 0.7143
epoch 10000, loss: 0.1579, accuracy: 0.8571
epoch 15000, loss: 0.1281, accuracy: 0.9048
epoch 20000, loss: 0.1058, accuracy: 0.9048
epoch 25000, loss: 0.0899, accuracy: 0.9048
epoch 30000, loss: 0.0784, accuracy: 0.9048
epoch 35000, loss: 0.0700, accuracy: 0.9048
epoch 40000, loss: 0.0635, accuracy: 0.9048
epoch 45000, loss: 0.0584, accuracy: 0.9048
epoch 50000, loss: 0.0545, accuracy: 0.9048
epoch 55000, loss: 0.0513, accuracy: 0.9048
epoch 60000, loss: 0.0489, accuracy: 0.9524
epoch 65000, loss: 0.0470, accuracy: 0.9524
epoch 67572, loss: 0.0462, accuracy: 1.0000
```

**Figure 2:** Loss values and accuracy during training process.

Figure 2 shows the loss values during the training process. The left one is for linear data and the right one is for xor data. As we can see, the loss value decreases as the number of epoch increases, indicating the model indeed learns to classify the data points into correct classes. Figure 3 shows the model predictions of each data point.



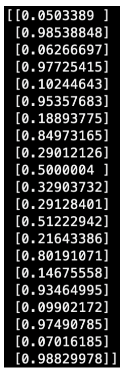
```
[[0.86019047]
 [0.85368545]
 [0.47773415]
 [0.73464235]
 [0.71267768]
 [0.83364757]
 [0.97320309]
 [0.96738503]
 [0.71197153]
 [0.61939239]
 [0.74434123]
 [0.90178096]
 [0.56379498]
 [0.49999348]
 [0.97644283]
 [0.86863915]
 [0.10333741]
 [0.12116163]
 [0.72315573]
 [0.97695718]
 [0.24064603]
 [0.9462467 ]
 [0.88116501]
 [0.04361258]
 [0.32896462]
```

```
 [0.96934932]
 [0.60810259]
 [0.10128421]
 [0.10038881]
 [0.67649899]
 [0.86203031]
 [0.18550761]
 [0.93641213]
 [0.32010637]
 [0.14004751]
 [0.02480964]
 [0.97210758]
 [0.23830854]
 [0.05072055]
 [0.49598629]
 [0.21313805]
 [0.10825199]
 [0.96771356]
 [0.9523531 ]
 [0.46642081]
 [0.98487919]
 [0.69669915]
 [0.29881276]
 [0.15681256]
 [0.96265637]
```

```
 [0.77558586]
 [0.06278098]
 [0.19218728]
 [0.9877431 ]
 [0.85975508]
 [0.49742467]
 [0.24051503]
 [0.97356513]
 [0.74615383]
 [0.05033375]
 [0.57382168]
 [0.79267839]
 [0.1604164 ]
 [0.85047544]
 [0.73198588]
 [0.90272842]
 [0.06532924]
 [0.05300258]
 [0.10091374]
 [0.37607982]
 [0.10442987]
 [0.63232625]
 [0.635764  ]
 [0.94021893]
 [0.06430688]
```

```
 [0.98739065]
 [0.1302038 ]
 [0.94162282]
 [0.90356449]
 [0.37651745]
 [0.82286865]
 [0.78545842]
 [0.3768354 ]
 [0.91354601]
 [0.47002595]
 [0.97315071]
 [0.26002229]
 [0.7710577 ]
 [0.10742795]
 [0.8562815 ]
 [0.10152056]
 [0.03464439]
 [0.95408756]
 [0.92717183]
 [0.91058395]
 [0.29185332]
 [0.58332156]
 [0.57322756]
 [0.79476579]
 [0.87644243]]
```

**(a)** Linear dataset



```
[[0.0503389 ]
 [0.98538848]
 [0.06266697]
 [0.97725415]
 [0.10244643]
 [0.95357683]
 [0.18893775]
 [0.84973165]
 [0.29012126]
 [0.5000004 ]
 [0.32903732]
 [0.29128401]
 [0.51222942]
 [0.21643386]
 [0.80191071]
 [0.14675558]
 [0.93464995]
 [0.09902172]
 [0.97490785]
 [0.07016185]
 [0.98829978]]
```

**(b)** XOR dataset

**Figure 3:** Model predictions for xor dataset

Figure 4 displays the comparison between ground truth and model predicted result. The two figures on the left side and right side respectively represent the result of linear data and xor data. It is obvious that the model correctly classifies all samples on both datasets.

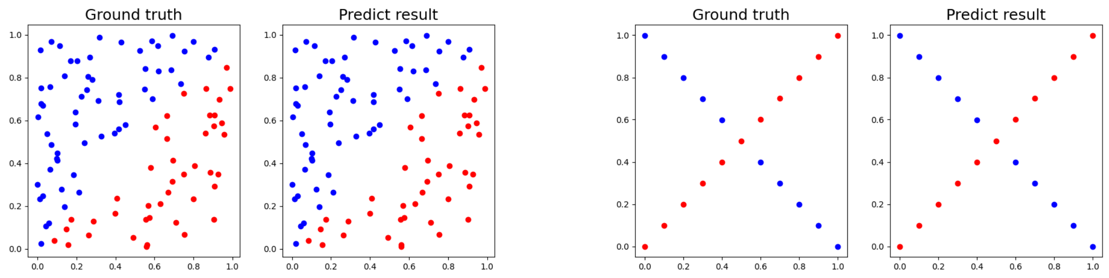


**Figure 4:** Comparison figures between ground truth and predicted result.

## 3.2 Accuracy of Prediction

The accuracy is also displayed in figure 2. We can see that the model has the ability to fit both linear and xor datasets perfectly. Also, as the loss value decreases, the accuracy increases.

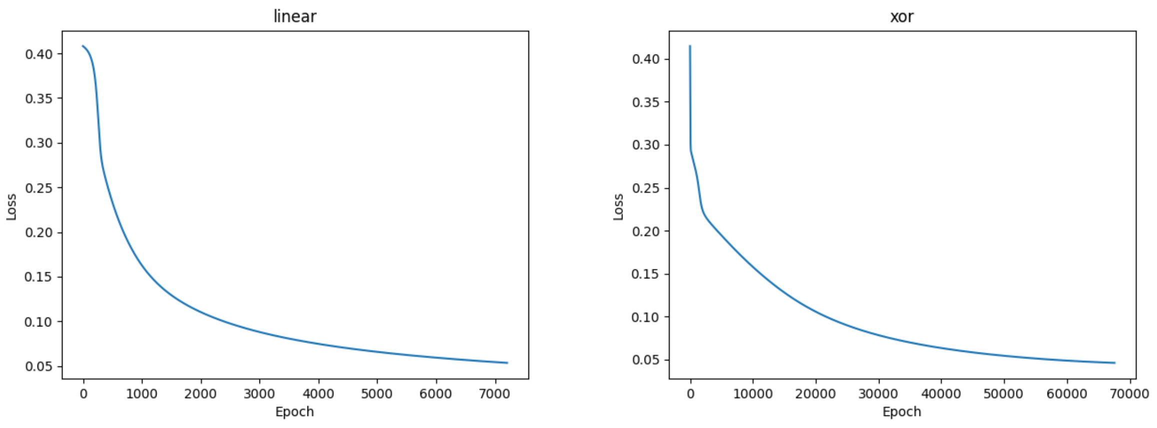


**Figure 5:** Learning curves for both linear and xor dataset.

| Learning Rate | Hidden Units | Activation | Linear / XOR | | | |
|---|---|---|---|---|---|---|
| | | | Stop Epoch | # Failure | Avg. Acc. | Best Acc. |
| $\mathbf{1 \times 10^{-1}}$ | 50 | Sigmoid | 10743 / 2649 | 1 / 0 | - / - | 1 / 1 |
| $\mathbf{5 \times 10^{-2}}$ | 50 | Sigmoid | 11485 / 5296 | 1 / 0 | - / - | 1 / 1 |
| $\mathbf{1 \times 10^{-2}}$ | 50 | Sigmoid | 17421 / 26475 | 1 / 0 | - / - | 1 / 1 |
| $\mathbf{5 \times 10^{-3}}$ | 50 | Sigmoid | 24841 / 52948 | 1 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **10** | Sigmoid | 1840 / 13935 | 0 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **20** | Sigmoid | 1316 / 6737 | 0 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **30** | Sigmoid | 963 / 4283 | 0 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **40** | Sigmoid | 775 / 3301 | 0 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **50** | Sigmoid | 10743 / 2649 | 1 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **60** | Sigmoid | 682 / 2039 | 0 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **70** | Sigmoid | 10533 / 3090 | 1 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **80** | Sigmoid | 635 / 11665 | 0 / 1 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **90** | Sigmoid | 3304 / 3709 | 0 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | **100** | Sigmoid | 419 / 11205.2 | 0 / 0 | - / - | 1 / 1 |
| $1 \times 10^{-1}$ | 50 | **W/O** | 2990 / 99999+ | 0 / 10 | 1 / 0.581 | 1 / 0.7619 |
| $1 \times 10^{-1}$ | 50 | **ReLU** | 20253 / 50154 | 2 / 5 | 0.884 / 0.7619 | 1 / 1 |

**Table 1:** Experiment results with different hyperparameters.

### 3.3 Learning Curve

Figure 5 shows the learning curve for both datasets, the left one is for linear data and the right one is for xor data. Unsurprisingly, the loss value decreases as the number of epoch increases, which is same as I mentioned in 3.1.

## 4 Discussion

In this section, some additional experiments with different hyperparameters are performed. The overall results is shown in table 1. The results can be roughly divided into 3 parts, the upper one has different learning rates, the middle one has different hidden units, and the lower one has different activation functions. Note that for each experiments displayed in each row, I perform a total of 10 trials and report the average score. To make it clear, stop epoch represents the first epoch that the model learns to classify all samples correctly. The number of failure is the number of trials that the model fails to converge within 100000 epochs, and for those trials that fails, their stop epoch would be recorded as 99999.

### 4.1 Different Learning Rates

From table 1, we can see that with different learning rates, the model can still achieve perfect accuracy on both datasets. It is worthy to note that as the learning rate decreases, the model needs more epochs to converge, which is reasonable.

### 4.2 Different Numbers of Hidden Units

From the results shown in table 1, models with different numbers of hidden units can all do perfect classification. However, I observe that the stop epoch does not always decrease as the number of hidden units increases, though such trend still exists to some degree. I consider the reason is that when the amount of model parameters increases, the model may have gain its ability to fit more complex data, but at the same time, it may also lose some stability during training.

### 4.3 Without Activation Functions

The lower part of table 1 demonstrates the results without activation functions. I found out that the model fails in all trials on xor dataset, while it can still correctly classify all linear data points. This is probably due to the data distribution. Since removing activation functions makes the model lose non-linearity, it can no longer fit perfectly on the xor dataset that has a non-linear data distribution.

## 5 Extra

### 5.1 Implement different activation functions.

I also implement ReLU as an activation function, its curve is shown in figure 6. The results are in the last row of table 1. I observe that using ReLU fails in 2 trials on linear dataset and 5 trials on xor dataset. This is probably because of the model designed in this lab having only one output, and the target value should be $0$ or $1$. Since ReLU function cannot convert the output to the range between $0$ and $1$, the model may not be easy to converge if the output value is far away from $0$ or $1$. To prevent such situation, I initialize smaller model weight while using ReLU function. On the other hand, we can obviously see that using sigmoid function indeed has a higher success rate than using ReLU.

$$ReLU(x) = \max(0, x)$$

$$ReLU'(x) = \begin{cases} 1, & \text{if } x > 0 \\ 0, & \text{otherwise} \end{cases}$$

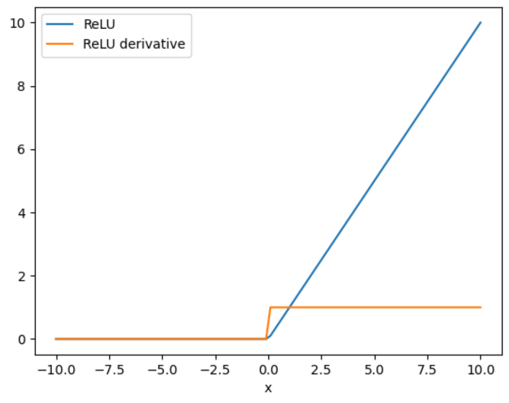


**Figure 6:** Curves of ReLU function and its derivative.