Deep Learning, SUMMER 2022

LAB1: BACKPROPAGAION

1. Introduction

Backpropagation is an efficient method to update the weights of each layer in multi-layer neural networks. It is usually implemented by the chain rule. The chain rule simplified the complex expression, $\partial L/\partial w_i$, to several easy-calculated partial derivatives. By computing and multiplying these partial derivatives $\partial L/\partial w_i$ can be solved. After that, we can apply the Generalized Delta Rule to update the weights of the multi-layer neural network.

2. Experiment setups:

A. Sigmoid functions

A Sigmoid function is a general term for a mathematical function that has the characteristic of S-shaped curve. In Deep Learning, the term "Sigmoid function" often refer to the Logistic function,

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

The Sigmoid function is a popular activation function. It can be applied in the input layer, hidden layers and the output layer of binary classification. It is continuous, bounded between 1 and -1 and differentiable at all points.

The derivative of the Sigmoid function is

$$\frac{\partial S(x)}{\partial x} = S(x)(1-S(x))$$

which is a part of $\partial L/\partial w_i$ which we will talk about soon.

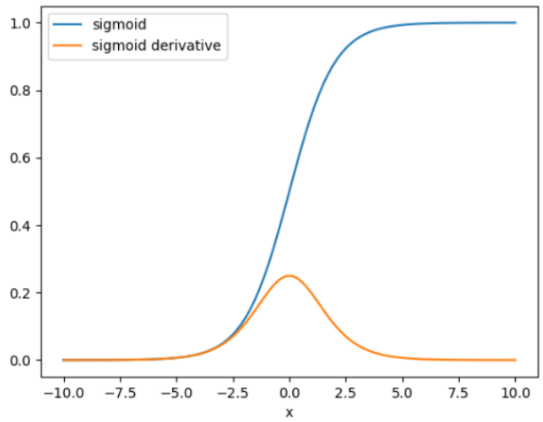


Figure 1. The curve of sigmoid (logistic) and its derivative.

B. Neural network

According to the requirements of this lab, the neural network consists of two hidden layers. For both the Linear and the XOR Data, the input will be 2-dimension and the output will be a simple 0 or 1, indicating two different classes.

We will then change the number of neurons, activation function of each layer and the learning rate to compare the results.

C. Backpropagation

To decompose the expression, we first look at the output layer:

$$\frac{\partial L}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial L}{\partial z} = x\frac{\partial L}{\partial z} = x\left(\frac{\partial y}{\partial z}\frac{\partial L}{\partial y}\right)$$

where “L” represents the loss function. “x” is the output of the previous layer. “w” is the weights of the output layer. "z” is the x multiply by w. “y” is the activation function output.

We can calculate x, $\frac{\partial y}{\partial z}$ and $\frac{\partial L}{\partial y}$, therefore, $\frac{\partial L}{\partial w}$ from the feedforward process.

Then, we look at the hidden and input layers:

$$\frac{\partial L}{\partial w} = \frac{\partial z}{\partial w}\frac{\partial L}{\partial z} = x\left(\frac{\partial y}{\partial z}\frac{\partial L}{\partial y}\right) = x\left(\frac{\partial y}{\partial z}\frac{\partial L}{\partial x_{next}}\right)$$

$$= x\left(\frac{\partial y}{\partial z}\left(\frac{\partial z_{next}}{\partial x_{next}}\frac{\partial L}{\partial z_{next}}\right)\right)$$

$$= x\left(\frac{\partial y}{\partial z}\left(w_{next}\frac{\partial L}{\partial z_{next}}\right)\right)$$

where "x" will be the input of the network for input layer and the output of the previous layer for hidden layers. We already calculate $\frac{\partial L}{\partial z_{next}}$ from previous calculation due to the nature of backpropagation. Furthermore, we have calculated the $\frac{\partial L}{\partial z}$ which is the $\frac{\partial L}{\partial z_{next}}$ for the next calculation.

As a result, we successfully decompose the partial derivative of the loss function to an easier expression via chain rule.

3. Results of your testing

A. Screenshot and comparison figure

As the picture below shown, with the same parameters, linear data only required 5000 epochs to train to 100% accuracy. On the other hand, XOR data 110000 epochs, which is 22 times that of linear. This is because linear data is less complicated than XOR. One only need a linear model to classify linear data.

```
Epoch 0, loss = 0.33775127546065065, accuracy = 0.54
Epoch 500, loss = 0.24057087342676683, accuracy = 0.54
Epoch 1000, loss = 0.23261890202724064, accuracy = 0.58
Epoch 1500, loss = 0.22027419803200277, accuracy = 0.73
Epoch 2000, loss = 0.20014681057636335, accuracy = 0.81
Epoch 2500, loss = 0.16910824620996212, accuracy = 0.87
Epoch 3000, loss = 0.13036916226948617, accuracy = 0.91
Epoch 3500, loss = 0.09556134985299453, accuracy = 0.94
Epoch 4000, loss = 0.07125527110531853, accuracy = 0.98
Epoch 4500, loss = 0.05548587544455839, accuracy = 0.99
Epoch 5000, loss = 0.045134269485221444, accuracy = 1.0
```

Figure 2. Linear training loss and accuracy.

```
Epoch 55000, loss = 0.24107853938874374, accuracy = 0.67
Epoch 60000, loss = 0.2370912380524325, accuracy = 0.71
Epoch 65000, loss = 0.23066750769278455, accuracy = 0.71
Epoch 70000, loss = 0.22041767314221006, accuracy = 0.71
Epoch 75000, loss = 0.2041480263713043, accuracy = 0.71
Epoch 80000, loss = 0.17708559992630798, accuracy = 0.81
Epoch 85000, loss = 0.13375788027848262, accuracy = 0.86
Epoch 90000, loss = 0.09318129065270722, accuracy = 0.9
Epoch 95000, loss = 0.06785553371489017, accuracy = 0.9
Epoch 100000, loss = 0.05100724272245462, accuracy = 0.95
Epoch 105000, loss = 0.03833031818656298, accuracy = 0.95
Epoch 110000, loss = 0.02821773744890706, accuracy = 1.0
```

Figure 3. XOR training loss and accuracy.

Below is the prediction of each point in linear and XOR data.

```
[[0.05459897]
 [0.2087982 ]
 [0.89331451]
 [0.98579612]
 [0.05685221]
 [0.96669633]
 [0.66985986]
 [0.88612257]
 [0.0214853 ]
 [0.07547361]
 [0.04711679]
 [0.97632779]
 [0.72541673]
 [0.76818688]
 [0.39974963]
 [0.99032723]
 [0.97266857]
 [0.82676184]
 [0.61744924]
 [0.03529778]
 [0.98840045]
 [0.98540054]]
[[0.05836576]
 [0.02345705]
 [0.01807929]
 [0.90939659]
 [0.04354551]
 [0.74687362]
 [0.9932305 ]
 [0.94245865]
 [0.94939076]
 [0.93409842]
 [0.99301765]
 [0.21504556]
 [0.77590694]
 [0.41141609]
 [0.94760716]
 [0.06790458]
 [0.95421311]
 [0.37503208]
 [0.99066962]
 [0.89739263]
 [0.99155958]]
[[0.21310694] [0.01772007]
 [0.73450572] [0.77709306]
 [0.87656286] [0.15846059]
 [0.98941097] [0.97351775]
 [0.92902291] [0.74739995]
 [0.97318942] [0.02873004]
 [0.44772904] [0.97668929]
 [0.8941882 ] [0.9792251 ]
 [0.03850453] [0.21144171]
 [0.98709455] [0.80618354]
 [0.99409261] [0.98524056]
 [0.6902598 ] [0.94965935]
 [0.98491266] [0.991029  ]
 [0.99370929] [0.93274974]
 [0.02133599] [0.99173766]
 [0.06178496] [0.03063195]
 [0.97659866] [0.87755189]
 [0.99275991] [0.02164029]
 [0.01798484] [0.91312356]
 [0.89029245] [0.03105708]]
[[0.81775716]
 [0.06567402]
 [0.96123036]
 [0.99208966]
 [0.04491499]
 [0.99096144]
 [0.81300764]
 [0.76524174]
 [0.02883218]
 [0.92405329]
 [0.98546659]
 [0.92492405]
 [0.02600527]
 [0.91708106]
 [0.06552601]
 [0.61489405]
 [0.75430144]]
[[0.27589914]
 [0.79861871]
 [0.31651455]
 [0.75082956]
 [0.35529196]
 [0.66501812]
 [0.38348042]
 [0.54417452]
 [0.39574272]
 [0.43494745]
 [0.39157893]
 [0.3740538 ]
 [0.4329561 ]
 [0.34788202]
 [0.54441636]
 [0.31785535]
 [0.66589838]
 [0.28780824]
 [0.74945369]
 [0.26018901]
 [0.79746814]]
```

Figure 4.5. Linear and XOR predictions.

Below is the result after trained to 100 accuracy.

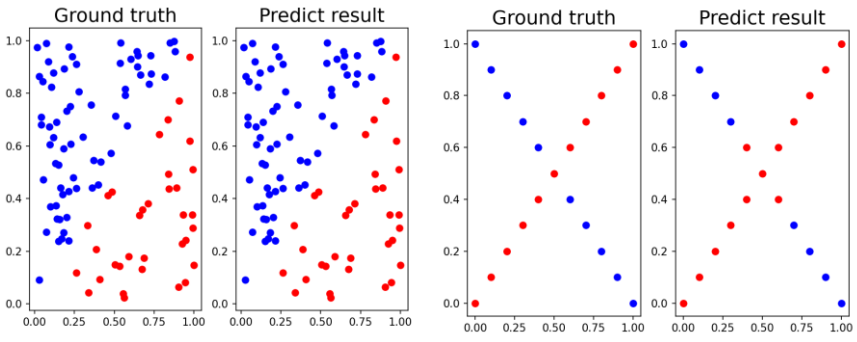
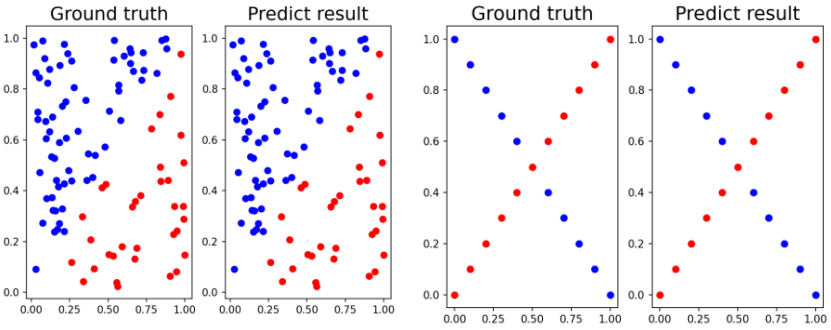


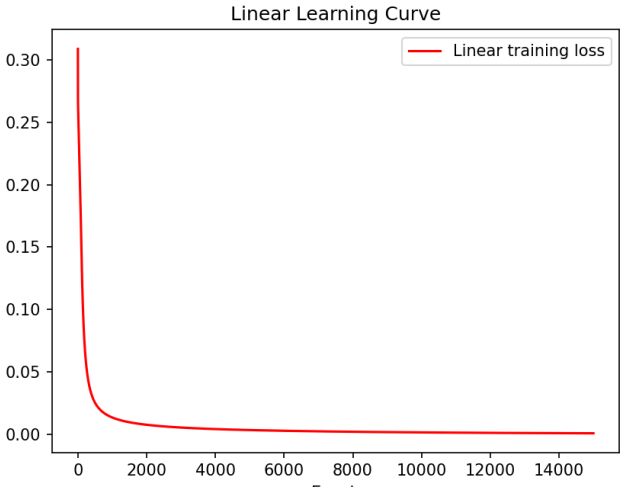
Figure 6.7. Linear and XOR visualized result comparisons.

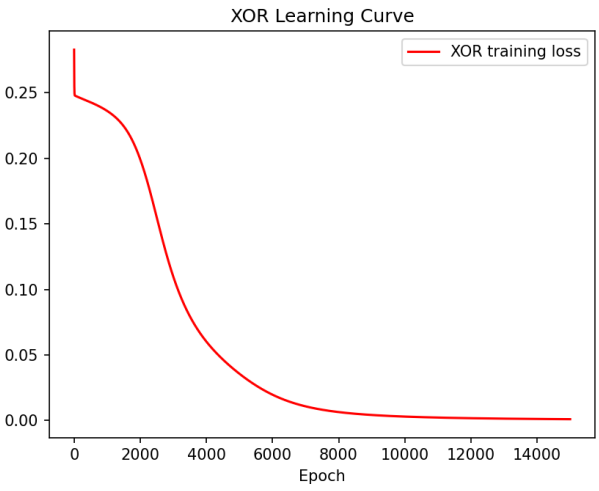
B. Show the accuracy of your prediction



	Testing loss	Testing Accuracy
Linear	< 0.001	100%
XOR	0.012	100%

C. Learning curve (loss, epoch curve)





D. anything you want to present

The results shown above are all trained by Sigmoid activation function. It turns out ReLU activation function is much difficult to train in a two-hidden layer model. Especially when the bias is not added. It is probably because ReLU is simpler than Sigmoid, and it turns every negative value to zero. Therefore, gradients vanished.

4. Discussion

A. Try different learning rates

The table shows that when the learning rates become smaller, it is proportional to the number of epochs to achieve 100% accuracy. For both linear and XOR data, the phenomenon is obvious in learning rate 0.1, 0.001 and 0.0001. Notice that the number of neurons of each layer does not matter in this case. To elaborate, learning rates determine the speed of updating weights, and the learning rate that is ten times smaller need ten times epochs to reach the same state.

LR	Neurons	Activation	~100%Epoch(linear/XOR)
0.1	20	Sigmoid	55 / 314
0.01	20	Sigmoid	2166 / 4431
0.001	20	Sigmoid	21679 / 44299
0.0001	20	Sigmoid	216805 / 442979
0.1	10	Sigmoid	256 / 1371
0.01	10	Sigmoid	542 / 16674
0.001	10	Sigmoid	5411 / 166682
0.0001	10	Sigmoid	54095 / 1666763

B. Try different numbers of hidden units

The number of epochs is highly affected by the number of neurons. There is a sweet spot for linear data when the number of neurons is 20. However, the sweet spot of XOR data seems to locate at higher number of neurons.

Neurons	LR	Activation	~100%Epoch(linear/XOR)

5	0.1	Sigmoid	208 / 3132
10	0.1	Sigmoid	256 / 1371
20	0.1	Sigmoid	55 / 314
40	0.1	Sigmoid	13774 / 312
80	0.1	Sigmoid	1000000+(54%) /120

C. Try without activation functions

Models for linear data classification can be trained without activation function. Since the activation function provide non-linear function into the model, XOR data, which is clearly non-linear cannot be trained without it. The 71% accuracy is the best a linear function can achieve.

LR	Neurons	Activation	~100%Epoch(linear/XOR)
0.1	10	Sigmoid	X (46%) / X (52%)
0.01	10	Sigmoid	3119 / X (48%)
0.001	10	Sigmoid	77 / X (71%)
0.0001	10	Sigmoid	591 / X (71%)
0.00001	10	Sigmoid	5880 / X (71%)

D. Anything you want to share

When the learning rate is too high, models usually stuck at a terrible accuracy, around 50%. Also, if the number of neurons is too large such as 500, 1000, a two-hidden-layer model can learning nothing.

5. Extra - Implement different activation functions.

As the table shown below, ReLU function is not a better choice for this simple model. When the learning rate is too high, it fail to predict correctly comparing to Sigmoid function with the same parameters. However, when the learning rate is small enough, it can achieve similar results as Sigmoid function in linear data classification. XOR data seems to be too complicated for ReLU to make a 100% accuracy in this case.

LR	Neurons	Activation	~100%Epoch(linear/XOR)
0.1	20	ReLU	X (54%) / X (48%)
0.01	20	ReLU	X (54%) / 1000000+ (76%)
0.001	20	ReLU	21962 / 1000000+ (76%)
0.0001	20	ReLU	226657 / 1000000+(76%)
0.1	20	Sigmoid	55 / 314
0.01	20	Sigmoid	2166 / 4431
0.001	20	Sigmoid	21679 / 44299
0.0001	20	Sigmoid	216805 / 442979