Deep Learning Lab1

110550085 房天越

1. Introduction

In this lab, we implement Neural Network, and get the predicted answer by forward propagation. Then, we measure the difference between the prediction and the ground truth, and do backpropagation to propagate it back, and update the weights in the layers with it. This way, we improve the model and get better results.

We also implement different activation functions and optimizers, so that we can observe the difference with different techniques and the change of hyperparameters.

2. Implementation Details

A. Sigmoid function

The sigmoid function and its derivative are set like the following picture, the sigmoid function is used in forward propagation, and the derivative of it is used in backpropagation. The output is set to sigmoid because it maps to (0, 1), which is a great choice for binary classification.

```
def sigmoid(x):
    return 1.0/(1.0+np.exp(-x))

def dsigmoid(x):
    return np.multiply(x, 1.0-x)
```

B. Neural network architecture

The neural network consists of an input layer with 2 neurons, two hidden layers with a configurable number of neurons, and an output layer with one neuron for binary classification.

For activation functions, we allow a choice of sigmoid, ReLU, or tanh for hidden layers, and for the output, we typically use sigmoid for binary classification. The weights are initialized by He initialization and the biases are initialized by zero initialization.

For the forward pass, each layer computes a linear transformation z = Wx+b, then apply the chosen activation function.

For example, a neural network is set like this:

```python
# Generate Data, Construct NN and Train it
# X, y = generate_linear(n=100)
X, y = generate_XOR_easy()
nn = NeuralNetwork(
    input_size = 2,
    hidden_size1 = 5,
    hidden_size2 = 5,
    output_size = 1,
    lr = 0.1,
    # Available optimizer : sgd, momentum, adam
    optimizer = "sgd",
    # Available activation function : relu, tanh, sigmoid
    act_hidden = "relu",
    act_output = "sigmoid"
)
```

C. Backpropagation

For the loss function, we use binary cross-entropy which is set as:

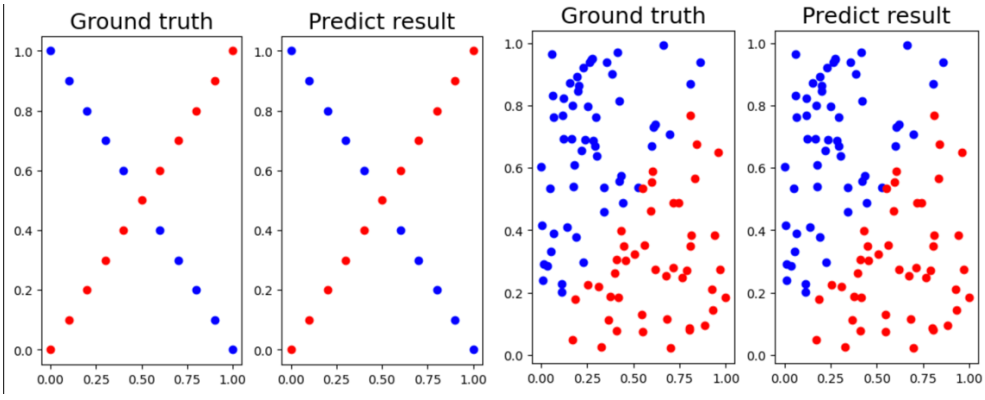
```python
def loss_function(self, y, y_pred):
    m = y.shape[0]
    return -np.sum(y * np.log(y_pred + 1e-8) + (1-y)*np.log(1-y_pred + 1e-8))/m
```

Then, we apply the chain rule to derive the partial derivatives of the loss with respect to each weight and bias, which is the gradient.

After calculating gradients, the weights and biases are updated by the optimizers.

3. Experimental Results

A. Screenshot and comparison figure



We can observe that for both XOR and linear data, the model can predict the answer with an accuracy of 100%.

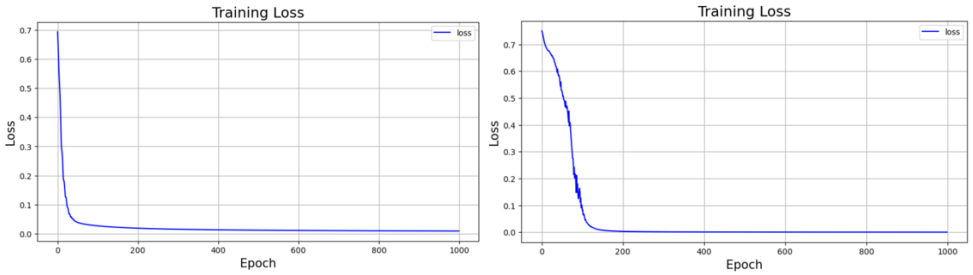
B. Show the accuracy of your prediction

```
...
Iter97 |     Ground truth: 0 |     prediction: 0.0000 |
Iter98 |     Ground truth: 1 |     prediction: 1.0000 |
Iter99 |     Ground truth: 0 |     prediction: 0.0000 |
loss=0.00300 accuracy=100.00%
```

```
Iter18 |     Ground truth: 1 |     prediction: 1.0000 |
Iter19 |     Ground truth: 0 |     prediction: 0.0003 |
Iter20 |     Ground truth: 1 |     prediction: 1.0000 |
loss=0.00029 accuracy=100.00%
```

The above one is the loss and accuracy of the linear one, and the bottom one is that of the XOR one, we can see that the network can predict both of them with 100% of accuracy.

C. Learning curve (loss-epoch curve)

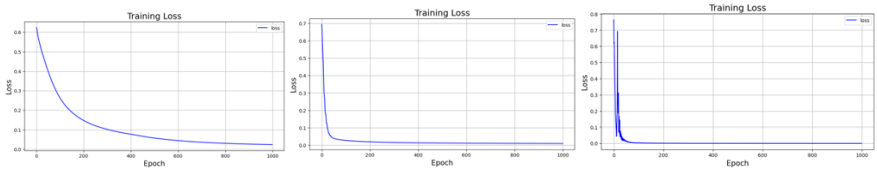


The left curve is the learning curve of the linear data, and the right curve is the learning curve of the XOR data. We can clearly see that both of their losses quickly converge to near 0, but there are some vibrations for the XOR one, this is because that it is more complicated compared to the linear one.

4. Discussion

A. Try different learning rates

a. Linear

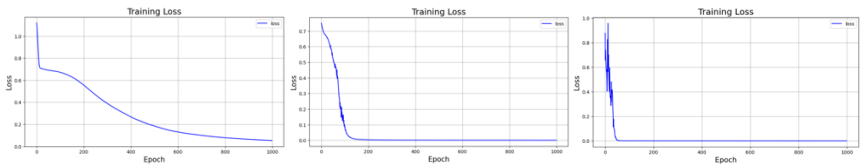


The three curves have learning rates of 0.01, 0.1, and 0.9.

We can see that for 0.01, it converges slower than the other ones.

And for 0.9, it converges but has a severe vibration, sometimes it even fails to converge.

b. XOR

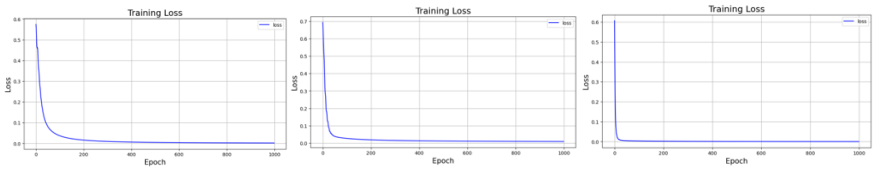


In these three graphs we can also observe the tentation like that of the

linear one.

B. Try different numbers of hidden units

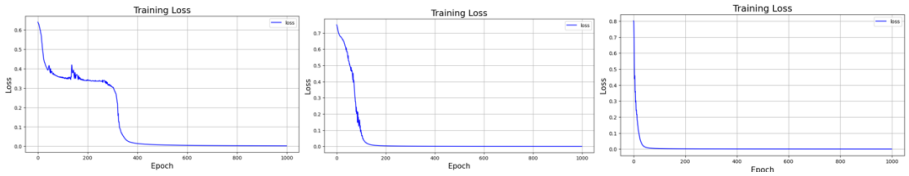
a. Linear



The three curves show the number of hidden units by 2, 8, and 200.

We can see that they all converges, but faster with more hidden units.

b. XOR



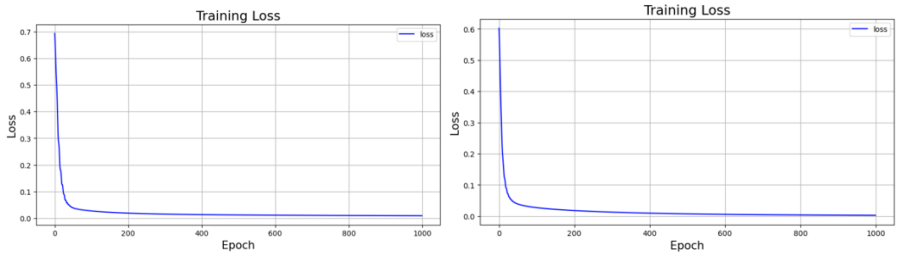
The three curves show the number of hidden units by 2, 8, and 200.

We can see that they all converges, but faster with more hidden units.

Also, we can notice that for 2 hidden units, the curve has a severe vibration, and sometimes even fail to converge.

C. Try without activation functions

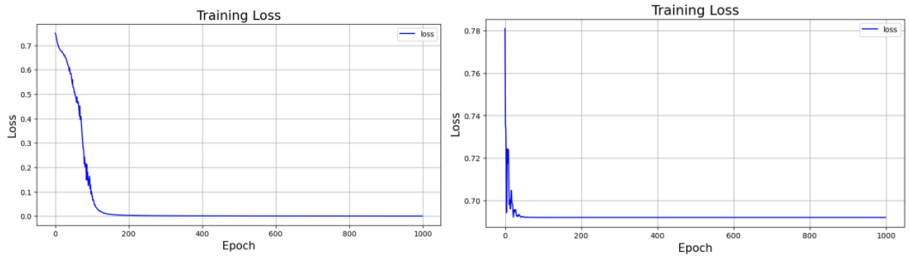
a. Linear



The left one is with activation functions, and the right one is without activation functions.

We can see both of them converges fast.

b. XOR



The left one is with activation functions, and the right one is without activation functions.

We can see that it fails to converge for the one without activation functions due to its non-linearity.

5. Questions
   A. What is the purpose of activation functions?

      Activation functions introduce non-linearity into the network. Without them, multiple layers of linear transformations would still behave as a single linear layer, and the network would fail to learn complex and non-linear decision boundaries.

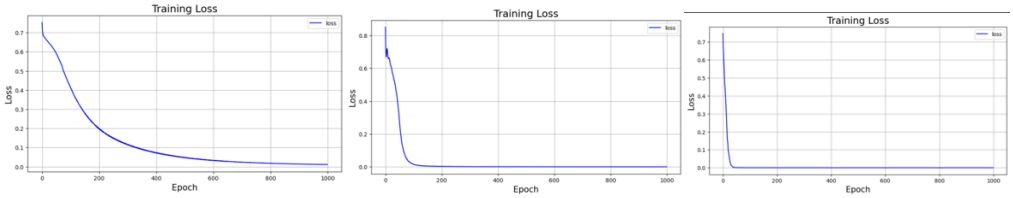
   B. What might happen of the learning rate is too large or too small?

      If the learning rate is too large, the network would become unstable or even diverge. If the learning rate is too small, the convergence would become slow and need more epochs for it to converge.

   C. What is the purpose of weights and biases in neural network?

      Weights determine how strongly each input neuron influences the neuron in the next layer, and biases allow shifting the activation function to fit the data better. Together, they represent the learnable parameters of the model, enabling the network to approximate complex functions.

6. **Extra**
   A. Implement different optimizers

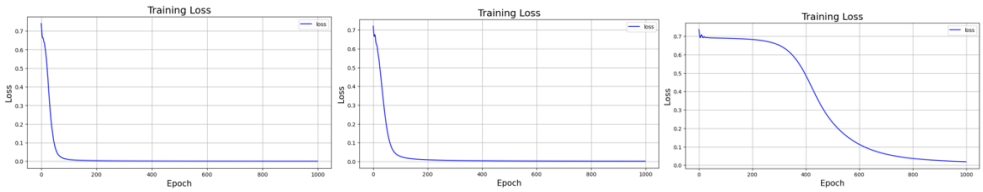


I implemented three optimizers, SGD, momentum, and Adam.

The three curves show them respectively, we can see Adam has the best performance between them, followed by momentum, and finally SGD.

This comparison is for XOR.

   B. Implement different activation functions



The three curves show the results for relu, tanh, and sigmoid functions, respectively. We can see for sigmoid function, the convergence is the slowest, followed by tanh and relu, but they can finally converge and has 100% of accuracy in prediction.

This comparison is for XOR.