Lab1 : back-propagation

110550095 王皓平

1 Introduction

In this assignment, we implemented a simple neural network to classify a set of 2D points based on their position relative to a decision boundary only using Numpy. The model was trained using gradient descent and different activation functions. This report will show the detail of the implementation and discuss about the performance of the model with different training parameters.

2 Implementation Details

2.1 Sigmoid Function

We use Sigmoid as the activate function of the hidden layers.

$$\sigma(x) = \frac{1}{1+e^{-x}}, \quad 0 < \sigma(x) < 1, \quad \forall x \in \mathbb{R} \tag{1}$$

The sigmoid function is a widely used activation function in neural networks , particularly in the output layer for binary classification tasks. The function maps any value to a range between 0 and 1.

2.2 Neural Network Architecture

The neural network consists of:

- **Input layer**: 2 neurons (x_1, x_2)
- **Hidden layers**: 2 layers, 4 neurons each
- **Output layer**: 1 neuron

2.3 Backpropagation

Using the chain rule, we compute the gradients for each layer:

$$dA_3 = A_3 - y \tag{2}$$
$$dW_3 = \frac{1}{m} A_2^T (dA_3 \cdot \sigma'(A_3)) \tag{3}$$
$$dW_2 = \frac{1}{m} A_1^T (dA_2 \cdot \sigma'(A_2)) \tag{4}$$
$$dW_1 = \frac{1}{m} X^T (dA_1 \cdot \sigma'(A_1)) \tag{5}$$

where $\sigma'(x) = \sigma(x)(1 - \sigma(x))$.

2.4 Weight Update (Gradient Descent)

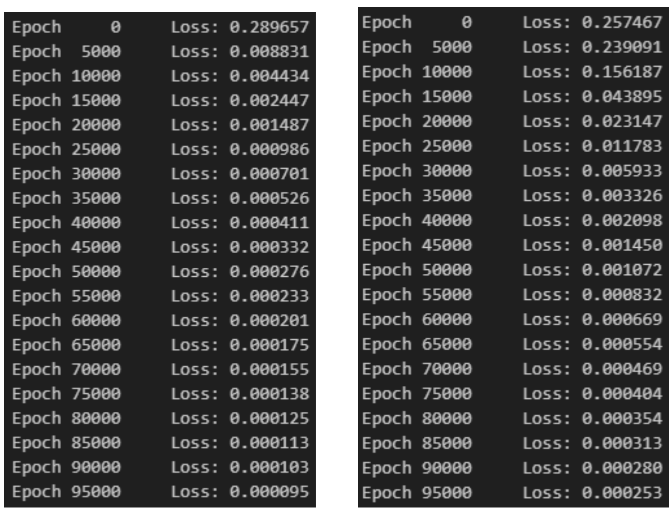
The weight matrices are updated using gradient descent:

$$W_i = W_i - \alpha dW_i \tag{6}$$

where: α is the learning rate, dW_i is the gradient of the weight matrix.

3 Experimental Results

3.1 Screenshot and comparison figure



Epoch 0	Loss: 0.289657
Epoch 5000	Loss: 0.008831
Epoch 10000	Loss: 0.004434
Epoch 15000	Loss: 0.002447
Epoch 20000	Loss: 0.001487
Epoch 25000	Loss: 0.000986
Epoch 30000	Loss: 0.000701
Epoch 35000	Loss: 0.000526
Epoch 40000	Loss: 0.000411
Epoch 45000	Loss: 0.000332
Epoch 50000	Loss: 0.000276
Epoch 55000	Loss: 0.000233
Epoch 60000	Loss: 0.000201
Epoch 65000	Loss: 0.000175
Epoch 70000	Loss: 0.000155
Epoch 75000	Loss: 0.000138
Epoch 80000	Loss: 0.000125
Epoch 85000	Loss: 0.000113
Epoch 90000	Loss: 0.000103
Epoch 95000	Loss: 0.000095

(a) Linear Data

Epoch 0	Loss: 0.257467
Epoch 5000	Loss: 0.239091
Epoch 10000	Loss: 0.156187
Epoch 15000	Loss: 0.043895
Epoch 20000	Loss: 0.023147
Epoch 25000	Loss: 0.011783
Epoch 30000	Loss: 0.005933
Epoch 35000	Loss: 0.003326
Epoch 40000	Loss: 0.002098
Epoch 45000	Loss: 0.001450
Epoch 50000	Loss: 0.001072
Epoch 55000	Loss: 0.000832
Epoch 60000	Loss: 0.000669
Epoch 65000	Loss: 0.000554
Epoch 70000	Loss: 0.000469
Epoch 75000	Loss: 0.000404
Epoch 80000	Loss: 0.000354
Epoch 85000	Loss: 0.000313
Epoch 90000	Loss: 0.000280
Epoch 95000	Loss: 0.000253

(b) XOR Data

Figure 1: Comparison of Training: Linear vs. XOR

3.2 Accuracy of Predictions

```
Iter 80 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 81 |     Ground Truth: 1 |     Prediction: 1.000000 |
Iter 82 |     Ground Truth: 1 |     Prediction: 0.999990 |
Iter 83 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 84 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 85 |     Ground Truth: 1 |     Prediction: 0.993082 |
Iter 86 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 87 |     Ground Truth: 0 |     Prediction: 0.000006 |
Iter 88 |     Ground Truth: 1 |     Prediction: 1.000000 |
Iter 89 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 90 |     Ground Truth: 0 |     Prediction: 0.000006 |
Iter 91 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 92 |     Ground Truth: 0 |     Prediction: 0.000004 |
Iter 93 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 94 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 95 |     Ground Truth: 1 |     Prediction: 1.000000 |
Iter 96 |     Ground Truth: 0 |     Prediction: 0.000003 |
Iter 97 |     Ground Truth: 1 |     Prediction: 0.999998 |
Iter 98 |     Ground Truth: 0 |     Prediction: 0.051907 |
Iter 99 |     Ground Truth: 0 |     Prediction: 0.000004 |
Test Accuracy: 100.00%
```

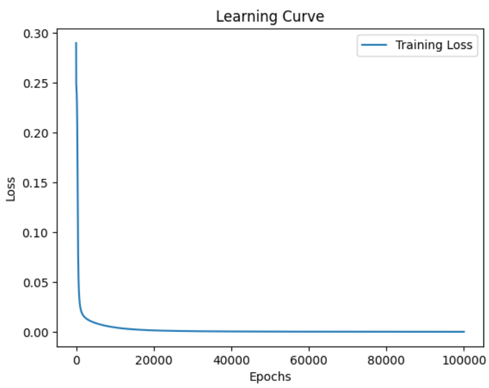
(a) Linear Data

```
Iter  0 |     Ground Truth: 0 |     Prediction: 0.004559 |
Iter  1 |     Ground Truth: 1 |     Prediction: 0.999995 |
Iter  2 |     Ground Truth: 0 |     Prediction: 0.000392 |
Iter  3 |     Ground Truth: 1 |     Prediction: 0.999995 |
Iter  4 |     Ground Truth: 0 |     Prediction: 0.000165 |
Iter  5 |     Ground Truth: 1 |     Prediction: 0.999994 |
Iter  6 |     Ground Truth: 0 |     Prediction: 0.000635 |
Iter  7 |     Ground Truth: 1 |     Prediction: 0.999973 |
Iter  8 |     Ground Truth: 0 |     Prediction: 0.013752 |
Iter  9 |     Ground Truth: 1 |     Prediction: 0.966362 |
Iter 10 |     Ground Truth: 0 |     Prediction: 0.045010 |
Iter 11 |     Ground Truth: 0 |     Prediction: 0.015394 |
Iter 12 |     Ground Truth: 1 |     Prediction: 0.965270 |
Iter 13 |     Ground Truth: 0 |     Prediction: 0.002843 |
Iter 14 |     Ground Truth: 1 |     Prediction: 0.999966 |
Iter 15 |     Ground Truth: 0 |     Prediction: 0.000650 |
Iter 16 |     Ground Truth: 1 |     Prediction: 0.999994 |
Iter 17 |     Ground Truth: 0 |     Prediction: 0.000214 |
Iter 18 |     Ground Truth: 1 |     Prediction: 0.999996 |
Iter 19 |     Ground Truth: 0 |     Prediction: 0.000096 |
Iter 20 |     Ground Truth: 1 |     Prediction: 0.999997 |
Test Accuracy: 100.00%
```

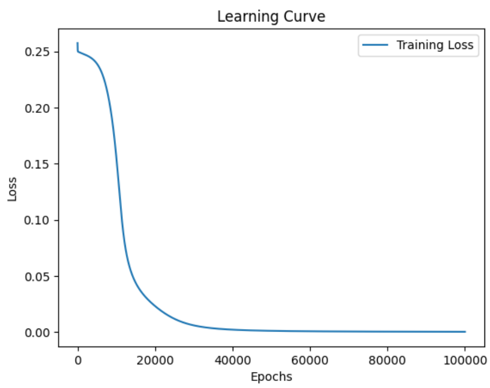
(b) XOR Data

Figure 2: Comparison of Accuracy: Linear vs. XOR

3.3 Learning Curve



(a) Linear Data



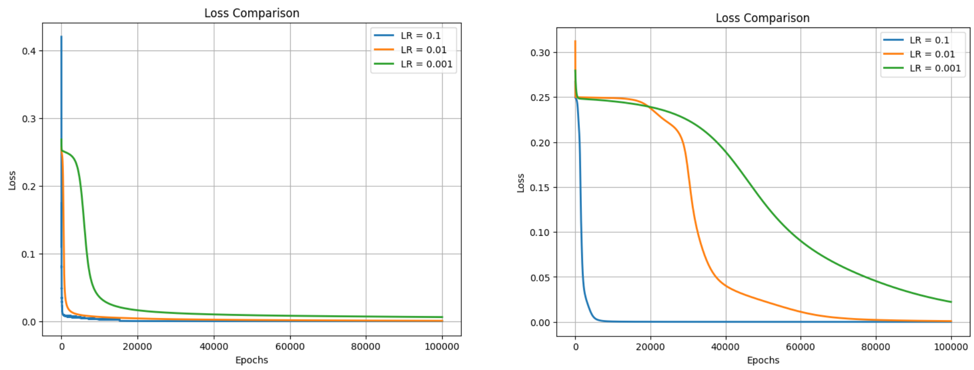
(b) XOR Data

Figure 3: Comparison of Learning Curve: Linear vs. XOR

The convergence speed of XOR curve takes longer to converge, reflecting the higher complexity of non-linear data. Both curves eventually approach near-zero loss, suggesting that the network can effectively learn both linear and non-linear tasks given enough epochs.

4 Discussion

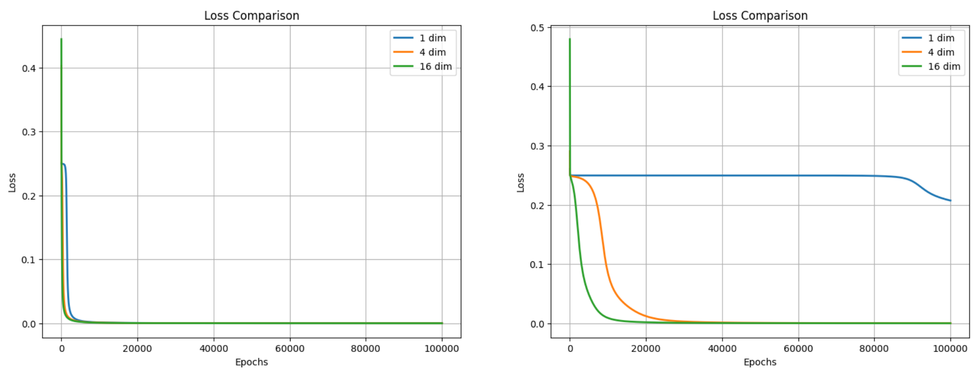
4.1 Different Learning Rates



(a) Linear Data (b) XOR Data

For the linear dataset, both learning rate of 0.1 and 0.01 converges quickly, while a rate of 0.001 takes longer but also reaches the same loss.
For the non-linear dataset, a learning rate of 0.1 shows the fastest initial drop, while a rate of 0.01 converges more smoothly. A rate of 0.001 is too slow to reach the same loss within the same number of epochs.
A larger learning rate shows the fastest decrease in loss for both datasets, but it can risk overshooting in more complex tasks.

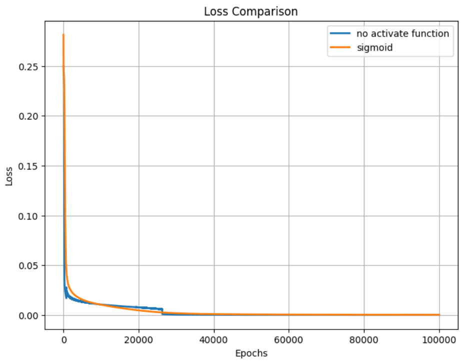
4.2 Different Numbers of Hidden Units



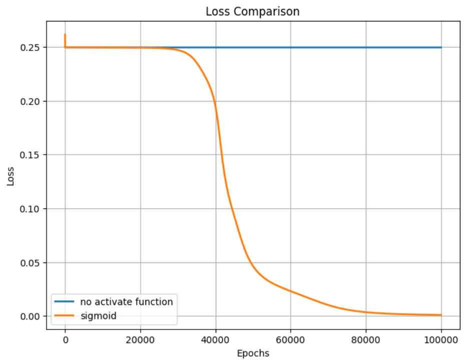
(a) Linear Data (b) XOR Data

For the linear dataset, all curves converge very quickly to near-zero loss, since linear data is easy to separate. Even a small network can learn a suitable boundary.
For the non-linear dataset, a larger number of hidden units allows the network to learn more complex patterns, leading to faster convergence and better accuracy. A smaller network struggles to fit the data, resulting in slower convergence and lower accuracy.
The hidden layer size doesn't significantly improve final performance when the data is linearly separable. However, a larger network can learn more complex patterns and achieve better performance for the complicated non-linear dataset,.

4.3 With and Without Activation Functions



(a) Linear Data



(b) XOR Data

The loss without activation functions quickly declines and eventually converges to zero for the linear dataset. However, the model fails to learn the non-linear pattern, resulting in poor accuracy. The activation functions introduce non-linearity into the network, allowing it to learn complex patterns beyond simple linear relationships. Without an activation function, the network behaves like a linear regression model, limiting its ability to solve complex tasks.

5 Questions

5.1 What is the purpose of activation functions?

Activation functions introduce **non-linearity** into the neural network, allowing it to learn complex patterns beyond simple linear relationships. Without an activation function, a neural network would behave like a linear regression model, limiting its ability to solve complex tasks.

5.2 What might happen if the learning rate is too large or too small?

- **if the learning rate is too large:**
  - The model may fail to converge or oscillate around the minimum.
  - It may skip over the optimal solution, preventing proper learning.
- **If the learning rate is too small:**
  - Training will be very slow, requiring many iterations to reach a good solution.
  - The model may get stuck in a local minimum, failing to generalize well.

5.3 What is the purpose of weights and biases in a neural network?

Weights and biases are the key parameters that allow a neural network to learn from data.

- **Weights (W):**
  - Represent the strength of connections between neurons.
  - Higher weights indicate stronger influence of one neuron on another.
  - Are updated during training using gradient descent.
- **Biases (b):**
  - Allow the activation function to shift, helping the model fit better.
  - Ensure neurons activate even when all inputs are zero.

The neural network learns **optimal weights and biases** during training to minimize the loss function and improve predictions.

6 Extra

6.1 Implement Different Optimizers

To improve training performance, we implemented different optimization algorithms:

- **Stochastic Gradient Descent (SGD)**:

$$W = W - \alpha \frac{\partial L}{\partial W} \tag{7}$$

- **Adam (Adaptive Moment Estimation)**:

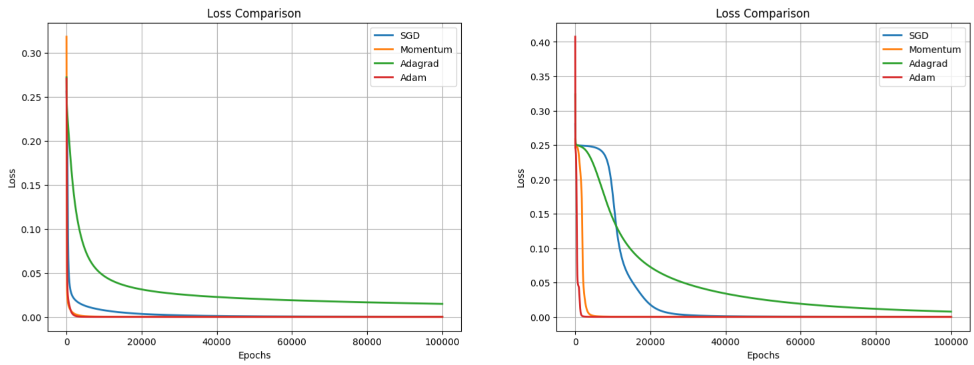
$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{8}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{9}$$

$$W = W - \alpha \frac{m_t}{\sqrt{v_t} + \epsilon} \tag{10}$$

- **RMSprop (Root Mean Square Propagation)**:

$$W = W - \frac{\alpha}{\sqrt{v_t} + \epsilon} g_t \tag{11}$$



(a) Linear Data (b) XOR Data

Adam maintains both first and second moments of gradients and achieves the fastest and smoothest descent, indicating its effectiveness in handling the non-linear dataset.
Momentum improves over SGD by considering the past gradients, helping the model move more decisively toward minimum loss.
Adagrad starts off well but may slow down in later training due to diminishing learning rates.

6.2 Implement Different Activation Functions

To compare the performance of different activation functions, we implemented:

- **Sigmoid**:

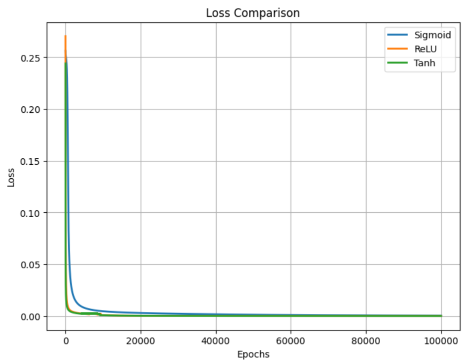
$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{12}$$

- **ReLU (Rectified Linear Unit)**:

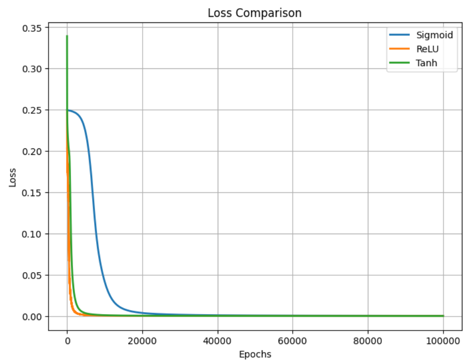
$$f(x) = \max(0, x) \tag{13}$$

- **Tanh (Hyperbolic Tangent)**:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{14}$$



(a) Linear Data



(b) XOR Data

For the linear dataset, all activation functions perform similarly, as the data is easy to separate.
For the non-linear dataset, ReLU and tanh show faster convergence and higher accuracy than sigmoid. Tanh avoids the strict (0,1) output limitation and provides a zero-centered range, while ReLU provides constant gradients for positive values and avoids vanishing gradients.

6.3 Implement Convolutional Layers