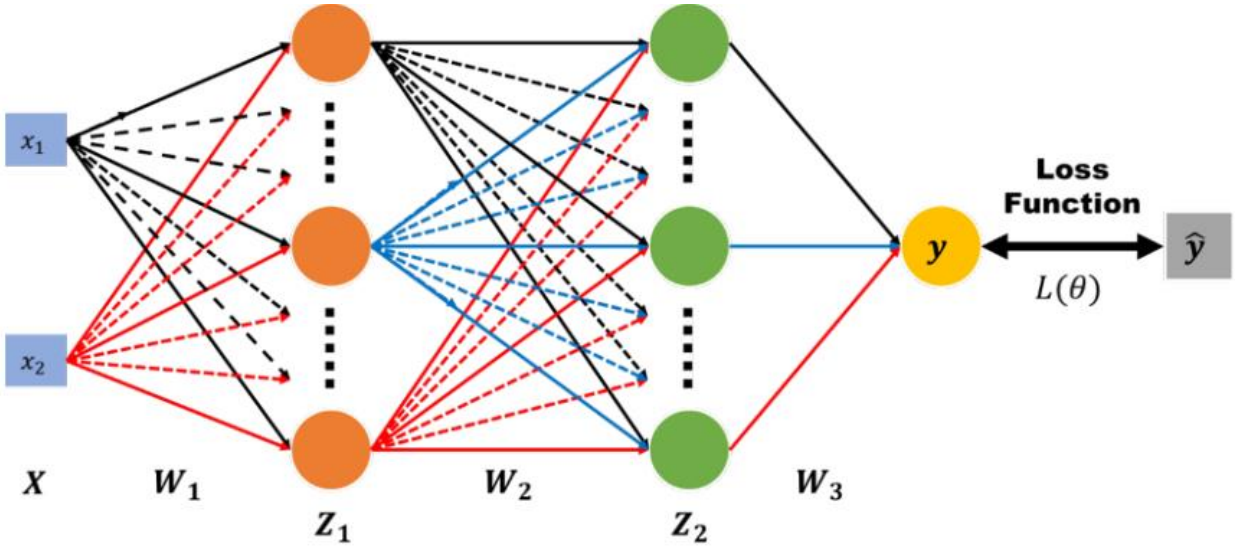
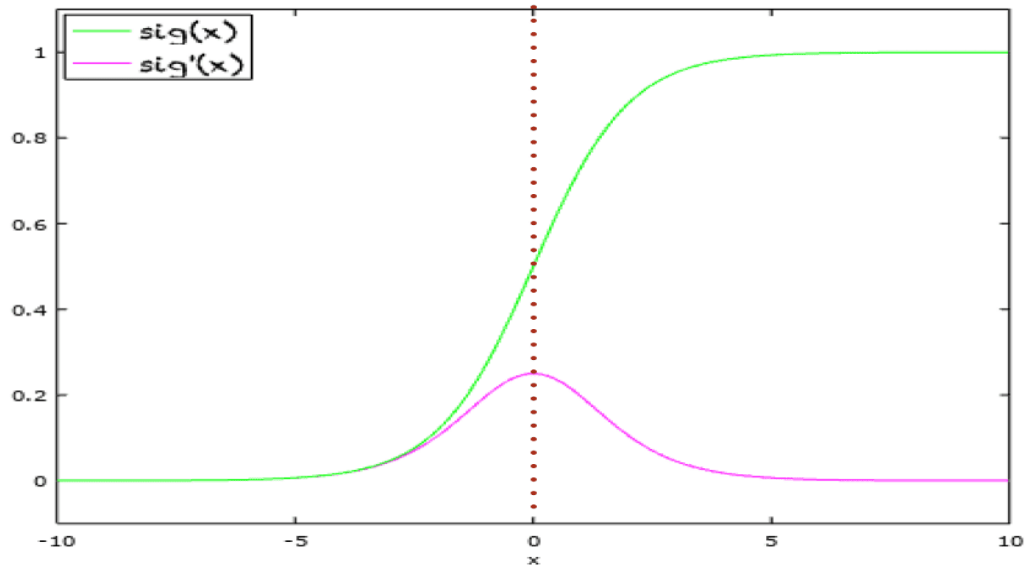
(1) Introduction

Construct a neuron network with two hidden layer to classify input data. It is a logistic regression problem, and the loss function is cross entropy.



(2) Experiment Setup

(a) sigmoid function



Sigmoid(x) = 1 / (1 + e^(-x))

Derivative_sigmoid(x) = sigmoid(x)*(1 – sigmoid(x))

(b) neuron network

Each hidden layer contains 10 neuron, and learning rate has different value according to different optimizer.

Cross entropy :

$$C(f(x^n), \hat{y}^n) = -\left[\hat{y}^n ln f(x^n) + (1-\hat{y}^n) ln\left(1 - f(x^n)\right)\right]$$

(c) back-propagation

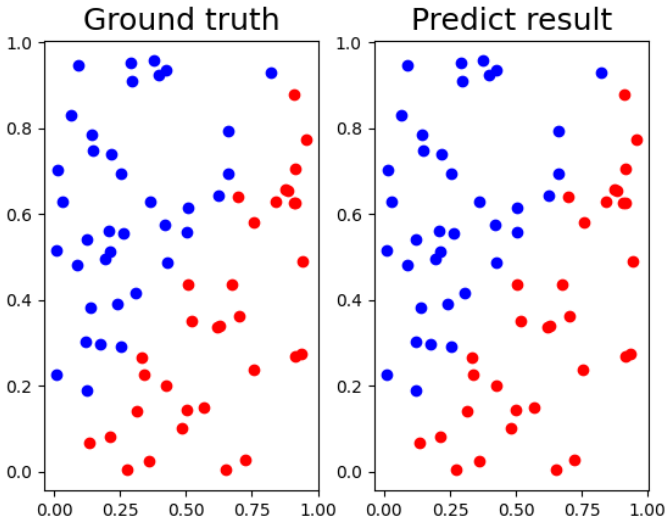
In the back-propagation, the error is propagated back through the network from the output layer to the input layer. The partial derivation of the error with respect to each weight and bias is calculated using the chain rule of calculus. The derivative indicate the parameter contribute to the error and provide the information to update parameters. The parameters are updated using optimizer.

1. Forward Pass: In the forward pass, an input is fed through the neural network, propagating it through the layers from the input layer to the output layer. Each neuron applies its activation function to the weighted sum of its inputs, producing an output signal.
2. Loss Computation: Once the forward pass is complete, the output of the network is compared to the desired output using a chosen loss function (e.g., mean squared error, cross-entropy). The loss function quantifies the discrepancy between the predicted output and the expected output.
3. Backward Pass: In the backward pass, the gradients of the loss function with respect to the parameters (weights and biases) of the network are calculated. The gradients represent the sensitivity of the loss function to changes in the parameters.
4. Weight Update: The calculated gradients are then used to update the weights and biases of the network, typically using an optimization algorithm such as stochastic gradient descent (SGD) or one of its variants. The weights and biases are adjusted in the opposite direction of the gradients to minimize the loss function.
5. Iteration: Steps 1 to 4 are repeated for a certain number of iterations or until the desired level of convergence is reached. This iterative process gradually improves the network's ability to make accurate predictions by adjusting the parameters in a way that reduces the overall loss.

(3) Experiment result

(a) screenshot and comparison figure

Linear:



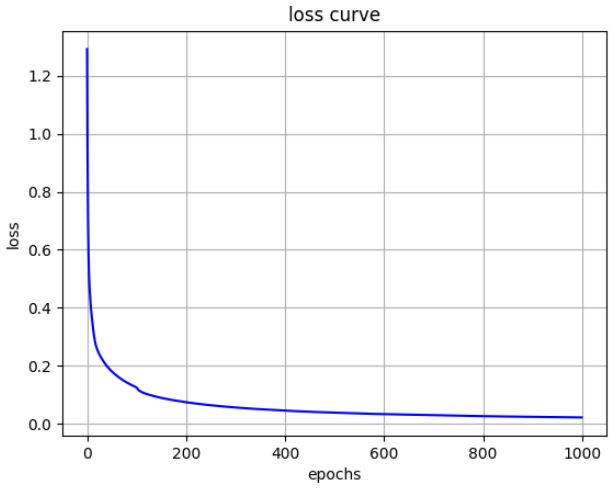
▲The comparison figure between predictions and ground truth

```
Epochs 980: loss=0.01357 accuracy=100.00%
Epochs 981: loss=0.01356 accuracy=100.00%
Epochs 982: loss=0.01354 accuracy=100.00%
Epochs 983: loss=0.01353 accuracy=100.00%
Epochs 984: loss=0.01351 accuracy=100.00%
Epochs 985: loss=0.01350 accuracy=100.00%
Epochs 986: loss=0.01348 accuracy=100.00%
Epochs 987: loss=0.01347 accuracy=100.00%
Epochs 988: loss=0.01345 accuracy=100.00%
Epochs 989: loss=0.01344 accuracy=100.00%
Epochs 990: loss=0.01342 accuracy=100.00%
Epochs 991: loss=0.01341 accuracy=100.00%
Epochs 992: loss=0.01340 accuracy=100.00%
Epochs 993: loss=0.01338 accuracy=100.00%
Epochs 994: loss=0.01336 accuracy=100.00%
Epochs 995: loss=0.01335 accuracy=100.00%
Epochs 996: loss=0.01334 accuracy=100.00%
Epochs 997: loss=0.01332 accuracy=100.00%
Epochs 998: loss=0.01331 accuracy=100.00%
Epochs 999: loss=0.01329 accuracy=100.00%
```

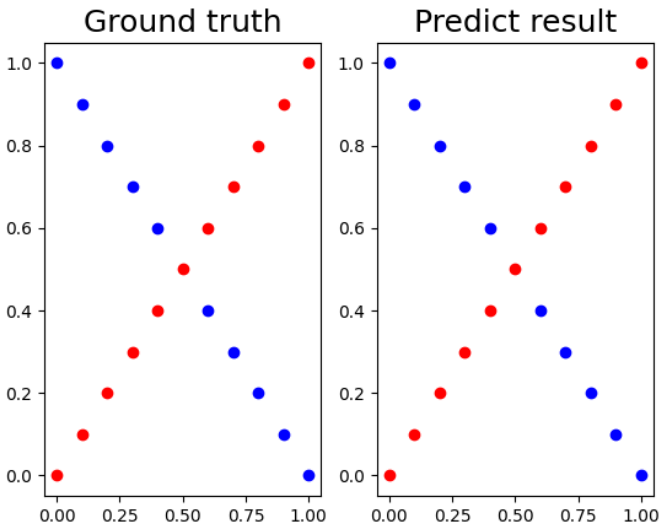
▲Training loss and the accuracy

```
testing:
[[9.99861286e-01 8.23880380e-01 4.61815166e-04 9.96748413e-01
  9.99954411e-01 3.47662647e-03 1.51182526e-03 2.78192248e-03
  7.26257405e-02 2.52809520e-04 9.99874727e-01 8.98536628e-01
  9.99620987e-01 9.99894697e-01 9.97849647e-01 9.99849612e-01
  2.16652682e-04 2.34077643e-04 9.99862825e-01 2.04401375e-03
  1.01025248e-03 9.99901845e-01 9.99956615e-01 9.65548016e-04
  1.38537451e-01 9.99004390e-01 9.79744748e-01 1.31151404e-04
  9.99923592e-01 9.99720480e-01 2.59657662e-04 9.99918923e-01
  9.92968530e-01 4.19821331e-02 9.99953916e-01 8.52574438e-02
  7.75185996e-04 9.99877613e-01 9.99952959e-01 9.99856864e-01
  1.14845553e-02 2.28049455e-04 9.99856377e-01 2.15719670e-04
  9.96294290e-01 9.22636310e-04 1.45892764e-04 3.39485087e-04
  9.99881084e-01 9.35677043e-01 9.99876285e-01 2.17698103e-04
  9.99935817e-01 7.87289188e-04 1.74887995e-04 4.82113403e-04
  9.86683178e-01 9.99419017e-01 2.06966431e-04 9.99954445e-01
  9.97894387e-01 8.20903639e-02 9.99925636e-01 9.99136855e-01
  4.52678419e-04 7.23952429e-04 1.39164067e-03 4.33509565e-04
  9.99931950e-01 9.69426396e-01]]
loss=0.01328 accuracy=100.00%
finished
```

▲Testing prediction



Xor:



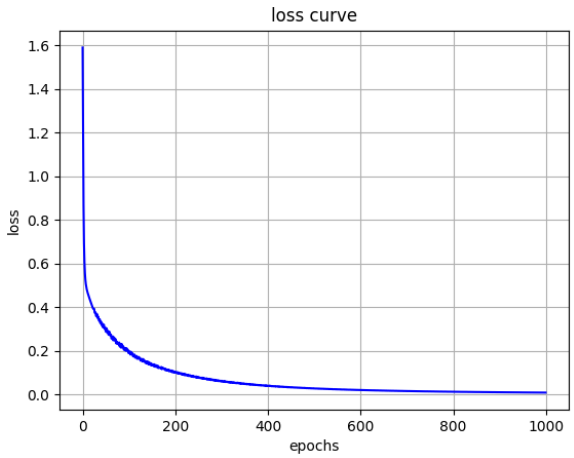
The comparison figure between predictions and ground truth

```
Epochs 979: loss=0.02735 accuracy=100.00%
Epochs 980: loss=0.02730 accuracy=100.00%
Epochs 981: loss=0.02725 accuracy=100.00%
Epochs 982: loss=0.02721 accuracy=100.00%
Epochs 983: loss=0.02716 accuracy=100.00%
Epochs 984: loss=0.02711 accuracy=100.00%
Epochs 985: loss=0.02706 accuracy=100.00%
Epochs 986: loss=0.02702 accuracy=100.00%
Epochs 987: loss=0.02697 accuracy=100.00%
Epochs 988: loss=0.02693 accuracy=100.00%
Epochs 989: loss=0.02688 accuracy=100.00%
Epochs 990: loss=0.02683 accuracy=100.00%
Epochs 991: loss=0.02679 accuracy=100.00%
Epochs 992: loss=0.02674 accuracy=100.00%
Epochs 993: loss=0.02670 accuracy=100.00%
Epochs 994: loss=0.02665 accuracy=100.00%
Epochs 995: loss=0.02661 accuracy=100.00%
Epochs 996: loss=0.02659 accuracy=100.00%
Epochs 997: loss=0.02651 accuracy=100.00%
Epochs 998: loss=0.02647 accuracy=100.00%
Epochs 999: loss=0.02643 accuracy=100.00%
```

Training loss and the accuracy

```
testing:
[[0.01088776 0.99984636 0.0102724  0.99951322 0.0099516  0.9982796
  0.01125564 0.99248704 0.02833437 0.92851197 0.06126605 0.04663397
  0.85588982 0.0320821  0.98142445 0.02428786 0.99170201 0.01988112
  0.99427254 0.01723879 0.99530274]]
loss=0.02638 accuracy=100.00%
finished
```

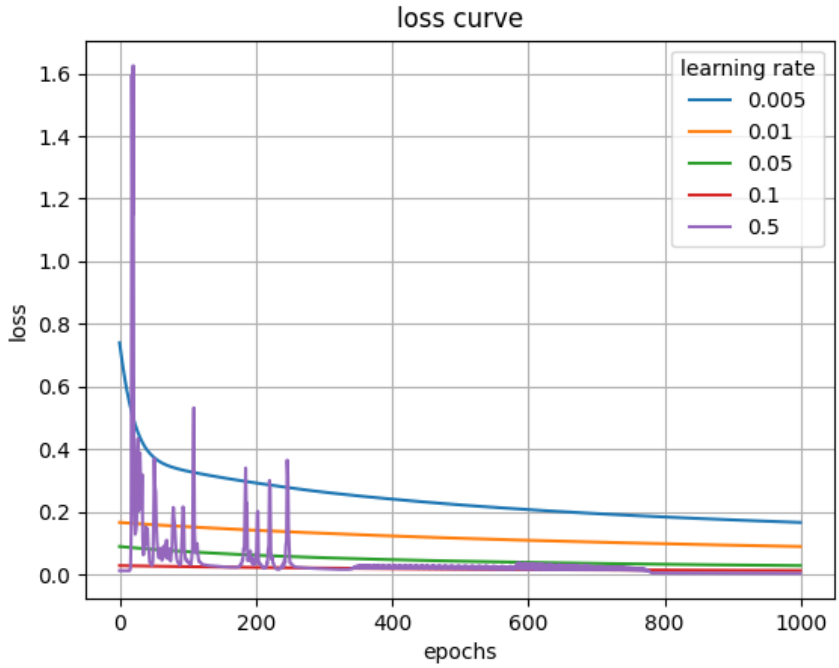
Testing prediction



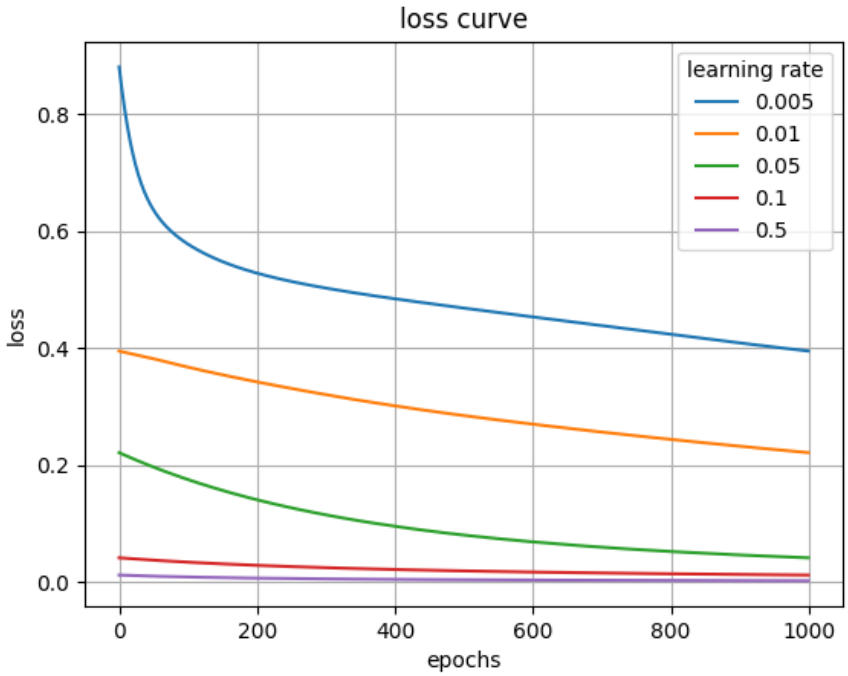
(4) Discussion

(a)Try different learning rate

Linear

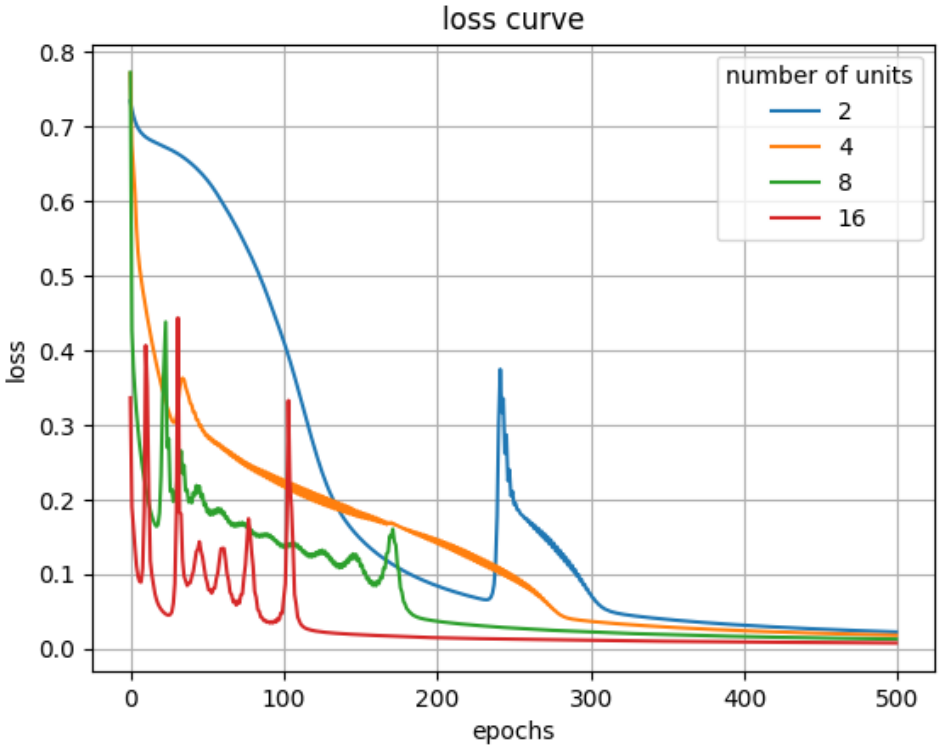


Xor

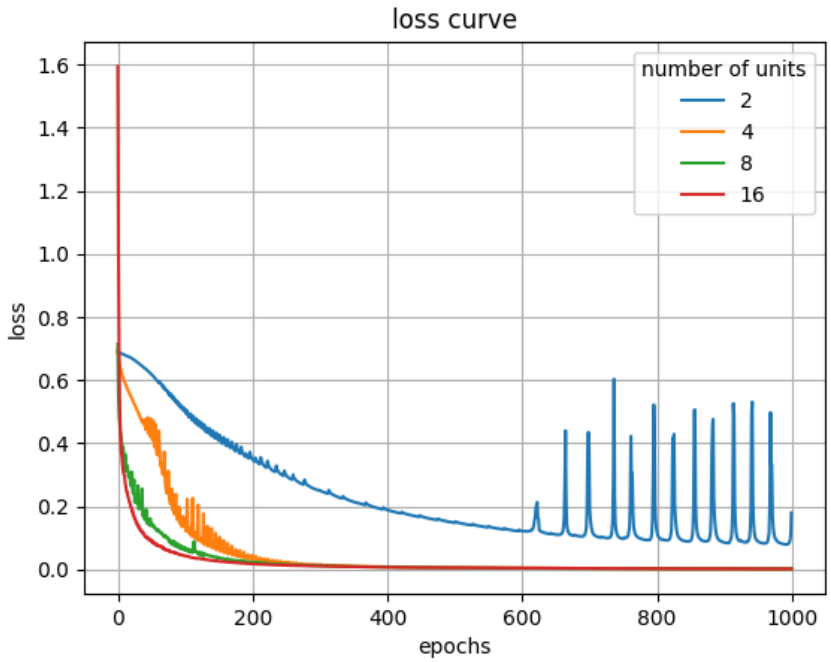


(b)different number of hidden units

Linear:



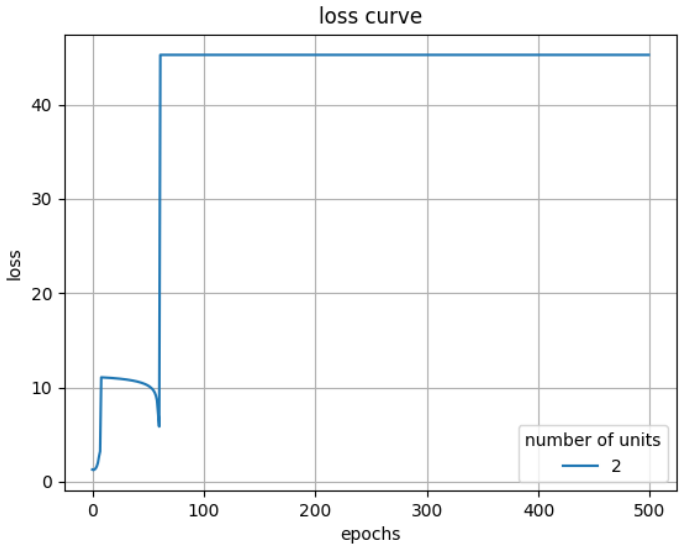
Xor:



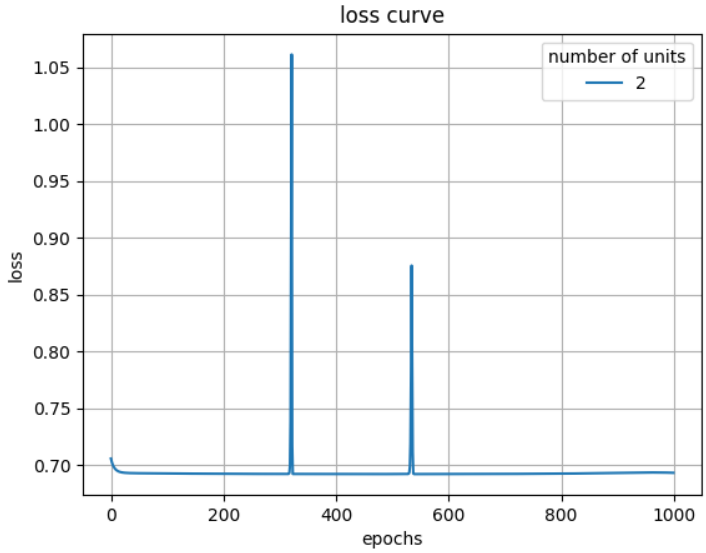
When there are two units in each layer, and training is becoming stable, the loss will sometimes become unstable. The problem is that when the input is 0.5, the model has to guess the output, so the parameters have to be very precise. The problem is too complex to use two units to solve.

(c)without activation function

Linear:



Xor:



If a neural network does not have an activation function, the network would only be capable of performing linear transformations. It introduces nonlinearity, allowing the network to learn and represent more complex function relationships.

Without an activation function, the neural network would consist of a series of linear layers, and the combination of multiple linear layers would still result in a linear transformation. This means that regardless of the number of layers in the neural network, their combined effect can be simplified to a linear transformation, incapable of capturing nonlinear features and patterns.

(4) Extra

(a)Implement different optimizers:

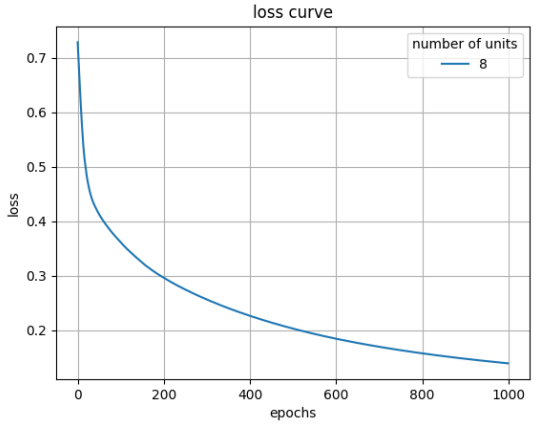
I use three optimizer:

1. GD: the method of gradient decent to find the parameters gradient. And updating weight according to gradient.

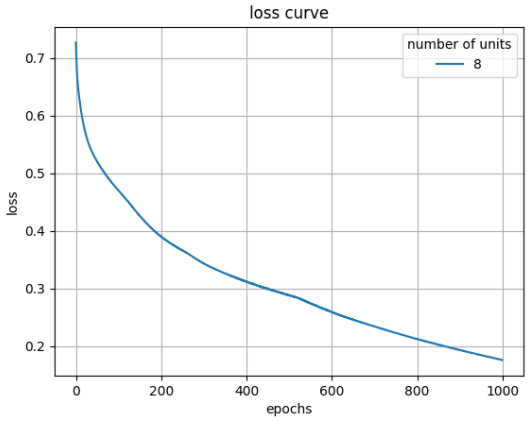
$$W \leftarrow W - \eta \frac{\partial L}{\partial W}$$

W: weight, L: loss function, η:learning rate

Linear



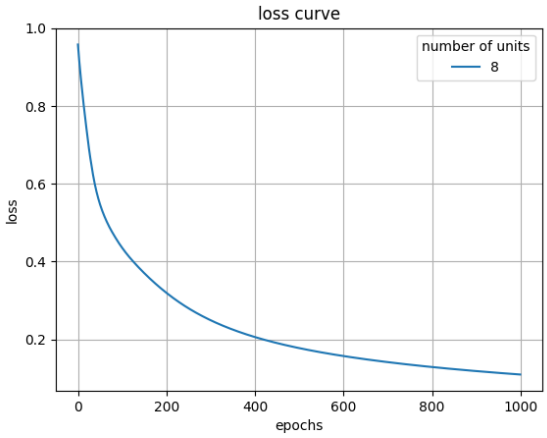
Xor



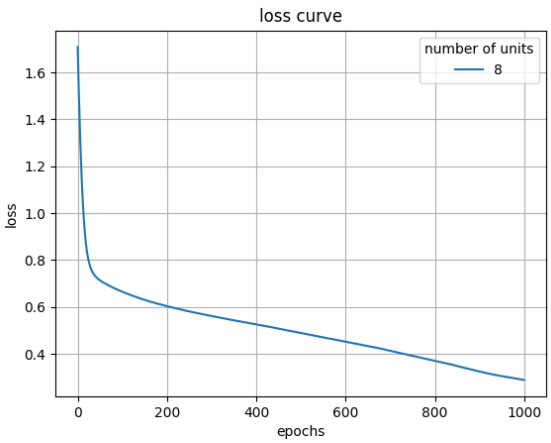
2. Momentum: If the direction is the same to the last update, the learning will become faster.

Vt: direction speed, β: often is 0.9

Linear



Xor



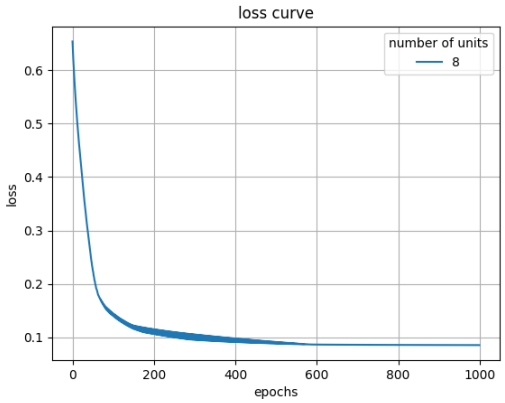
3. Adagrad: When the gradient norm is large and n is large, it can constrain the learning rate, but the accumulation of squared gradients in the denominator will keep increasing, causing the gradient to approach zero.

$$W \leftarrow W - \eta \frac{1}{\sqrt{n+\epsilon}} \frac{\partial L}{\partial W}$$

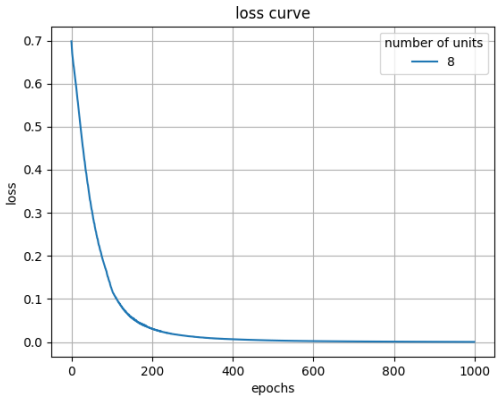
$$n = \sum_{r=1}^{t} (\frac{\partial L_r}{\partial W_r})^2$$

$$W \leftarrow W - \eta \frac{1}{\sqrt{\sum_{r=1}^{t} (\frac{\partial L_r}{\partial W_r})^2 + \epsilon}} \frac{\partial L}{\partial W}$$

Linear



Xor



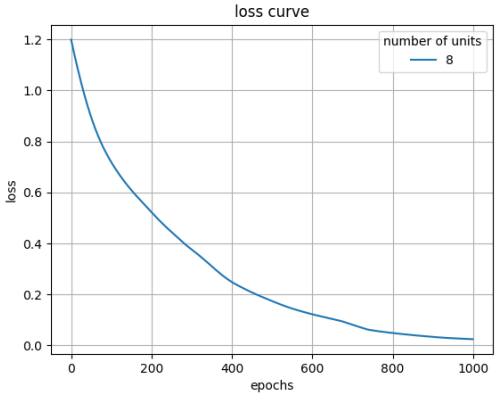
4. Adam: Momentum + AdaGrad, Adam retains Momentum to adjust the gradient speed of the direction of the past gradient and Adam adjusts the learning rate of the square value of the past gradient. Besides, Adam gas a "deviation correction for parameters, so the each learning rate will be determined. The range will be determined. The range will make the update of the parameters more stable.

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\frac{\partial L_t}{\partial W_t} \qquad \hat{m}_t = \frac{m_t}{1-\beta_1^t}$$

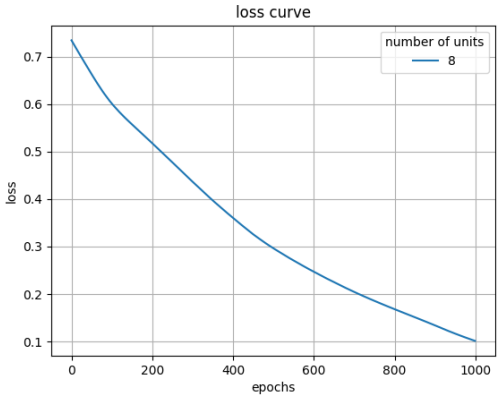
$$v_t = \beta_1 v_{t-1} + (1-\beta_2)(\frac{\partial L_t}{\partial W_t})^2 \qquad \hat{v}_t = \frac{v_t}{1-\beta_2^t}$$

$$W \leftarrow W - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

Linear



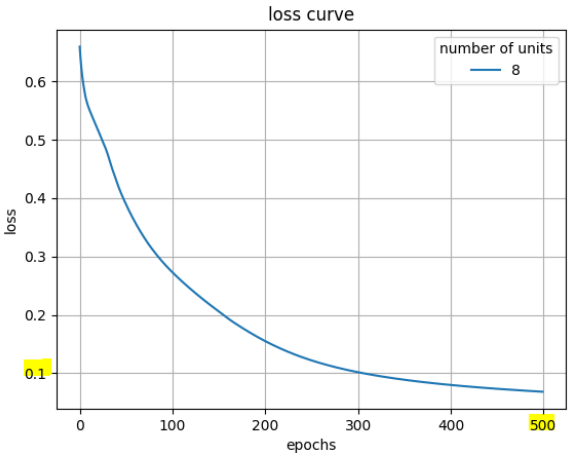
Xor



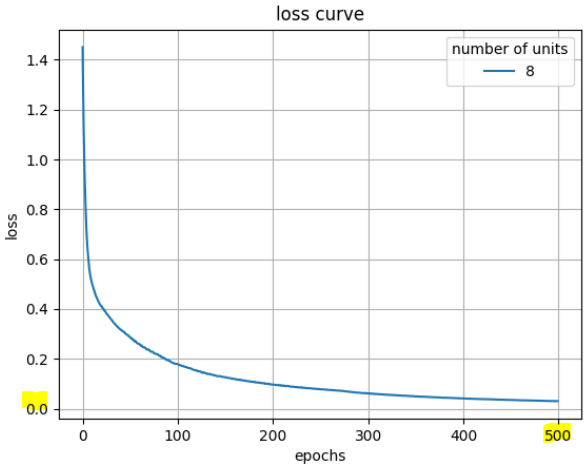
(b)Implement different activation function:

First: ReLU Second: ReLU Output: Sigmoid

Linear



Xor



Advantage:

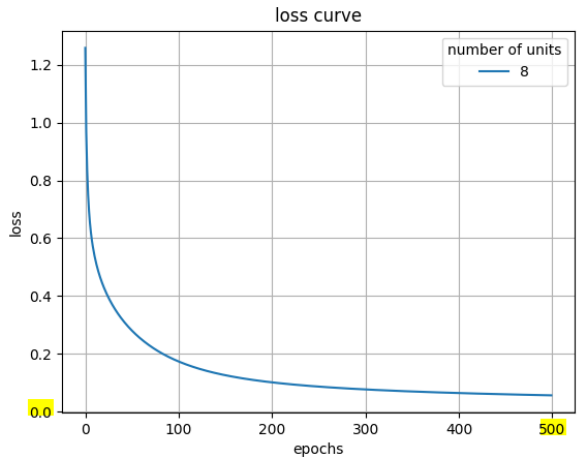
1. ReLU introduces non-linearity to the neural network, allowing it to learn and represent complex patterns and relationships in the data.
2. Since ReLU has a constant positive gradient for positive inputs, it avoids the diminishing gradient problem and facilitates effective gradient flow.
3. As ReLU sets negative inputs to zero, it promotes sparse activation where only a subset of neurons gets activated, resulting in a more efficient and expressive network representation.

Disadvantage:

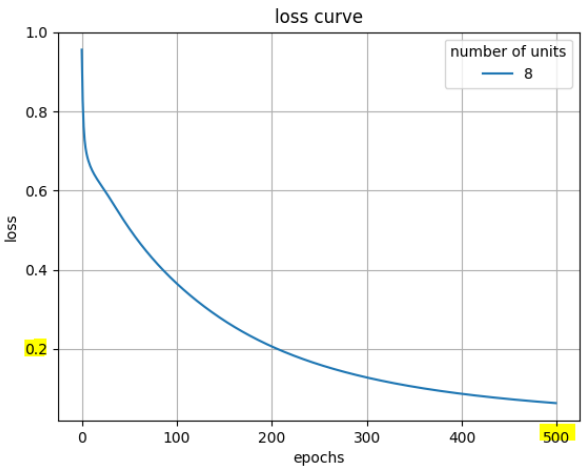
1. When a neuron's output becomes zero or negative for all inputs during training, it is considered "dead" and remains non-responsive to any further training. Dead neurons do not contribute to the learning process, resulting in a decrease in model capacity. This problem is more likely to occur when using a large learning rate or with imbalanced data.
2. Lack of saturation can cause issues when dealing with very large inputs, leading to unbounded activations, which may negatively affect the training process.
3. ReLU is not centered around zero, which means it can introduce a bias in the network's output.

First: tanh Second: tanh Output: Sigmoid

Linear



Xor



Advantage:

1.nonlinear activation function

2. The output of the tanh function ranges from -1 to 1 and is centered around zero. This means that the average output value is close to zero, which aids in convergence and stability during training.

3. tanh is a smooth curve with a continuous derivative across its entire input range. This smoothness helps reduce the risk of gradient spikes and explosions, promoting stability in gradient descent optimization algorithms.

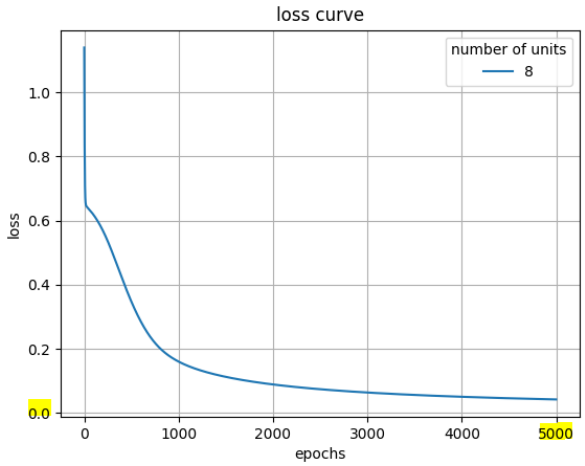
Disadvantage:

1.in the saturation regions, leading to vanishing gradients and reducing the efficiency of gradient descent algorithms.

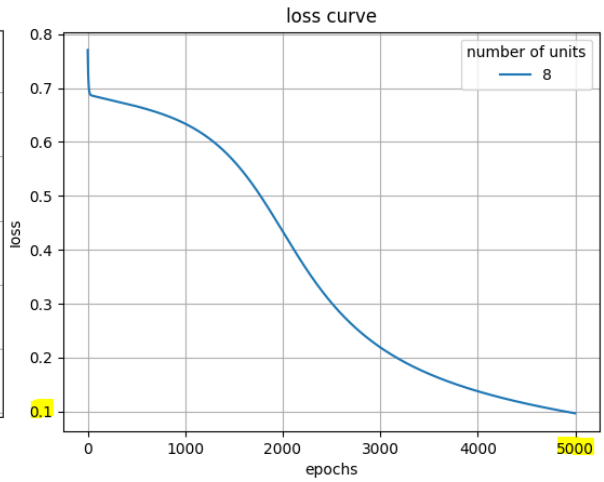
2. It involves exponential operations, which can be computationally expensive, especially in large-scale neural networks.

First: Sigmoid Second: Sigmoid Output: Sigmoid

Linear



Xor



Advantage:

1. The sigmoid function is a nonlinear activation function
2. The smoothness helps prevent abrupt changes in the output and ensures a stable learning process.
3. The sigmoid function maps the input to a range between 0 and 1. This property is useful for tasks where the output needs to be interpreted.

Disadvantage:

1. The derivative of the sigmoid function becomes very small as the input moves away from zero, leading to the vanishing gradient problem.
2. The sigmoid function is not centered around zero, which can introduce a bias in the network's output.
3. The sigmoid function saturates at the extremes of the input range, causing the gradient to approach zero.
4. Cost becomes more significant when dealing with large-scale neural networks and high-dimensional data.