DL Lab 1 – Backpropagation

曾大衛

1. Introduction

在 Lab 1 中實作兩層 hidden layers 的 neural network，透過程式呈現 forward, backpropagation,不同的 activation functions 以及 optimizers，並使用 linear data 和 XOR data 對 neural network 進行訓練，嘗試不同的訓練設定與其後續對模型產生的影響。

2. Implementation Details

以下是 activation function, model, forward 跟 backpropagation 的詳細設定

2.1. Sigmoid function

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

2.2. Neural network architecture

```
class Model:
    def __init__(self, input_size= 2, hidden_size= 10, output_size= 1, lr=0.01,
                 optim = "SGD", activate = "ReLU", show_epoch = 10000, decay = 0.9):
        self.layer1 = Layer(input_size, hidden_size, activate)
        self.layer2 = Layer(hidden_size, hidden_size, activate)
        if activate == "None":
            self.output = Layer(hidden_size,output_size, activate)
        else:
            self.output = Layer(hidden_size,output_size, "Sigmoid")
        self.lr = lr
        self.loss = []
        self.show_epoch = show_epoch
        self.epoch = 0
        self.optim = optim
        self.decay = decay

    def train(self,x,y,epoch = 100000):
        self.epoch += epoch
        for i in range(epoch):
            output = self.output.forward(self.layer2.forward(self.layer1.forward(x)))
            loss, grad = MSELoss(output, y)
            self.layer1.backward(self.layer2.backward(self.output.backward(grad, self.lr, self.optim, self.decay)
                                 , self.lr, self.optim, self.decay), self.lr, self.optim, self.decay)
            self.loss.append(loss)
            if i%self.show_epoch == 0:
                print(f"epoch {i} loss : {loss}")
        self.prediction = output
        plt.subplot(2,1,1)
        plt.title("Learning Curve", fontsize = 18)
        plt.xlabel("Epoch")
        plt.ylabel("Loss")
        plt.plot(loss)
        return output

    def show_result(self,x,y):
        import matplotlib.pyplot as plt
        plt.plot(self.loss)
        print(f"Accuracy : {sum((self.prediction > 0.5)== (y==1))/y.size}")
        print("Prediction : ")
        for i in range(y.size):
            print(f"Iter{i+1} |     Ground truth: {y[i]} |     prediction: {self.prediction[i]} |")
        show_result(x,y,self.prediction>0.5)
```

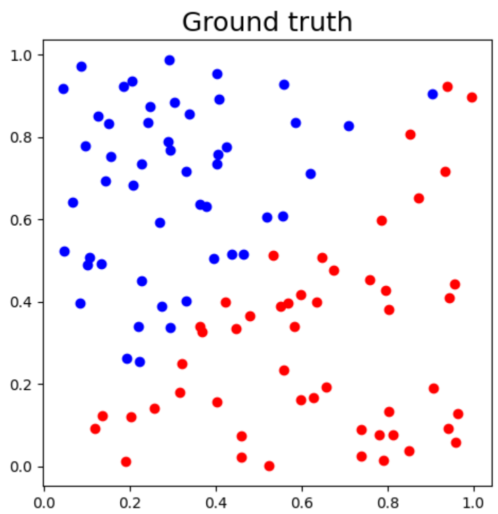
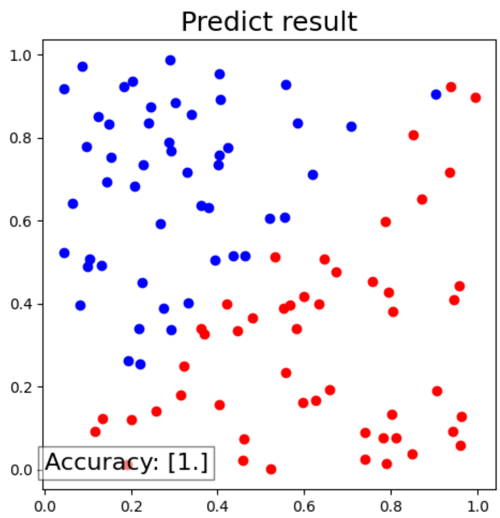
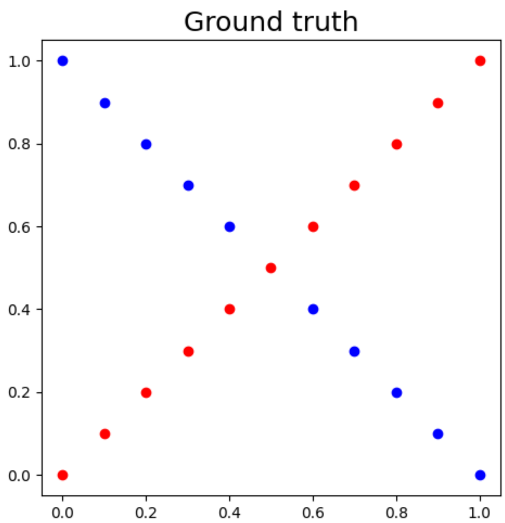
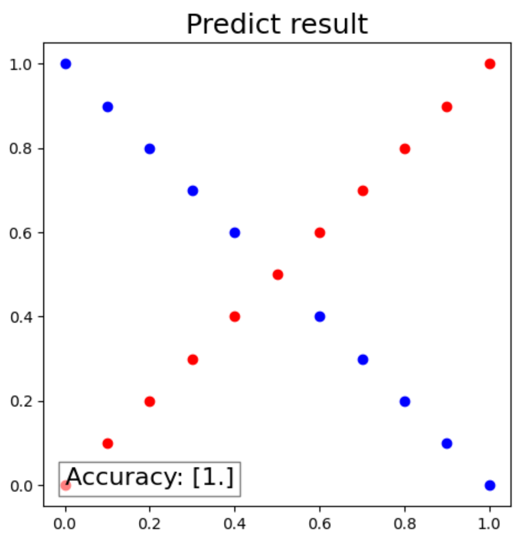
2.3. Backpropagation

```
class Layer:
        def __init__(self, input_size, output_size, activate = "Sigmoid"):
                self.input_size = input_size
                self.output_size = output_size
                self.activate = activate
                self.v_w = 0
                self.v_b = 0
                self.total_w = 0
                self.total_b = 0
                #initialize weight and bias
                self.w = np.random.randn(input_size, output_size) #shape(2,n)
                self.b = np.random.randn(1, output_size)/100 #shape(1,n)
        def forward(self,x):
                self.x = x
                z = np.dot(x,self.w) +self.b
                if self.activate == "Sigmoid":
                        z = sigmoid(z)
                elif self.activate == "ReLU":
                        z = ReLU(z)
                elif self.activate == "tanh":
                        z = tanh(z)
                else:
                        pass
                self.z = z

                return z
        def backward(self, upstream_grad, lr=0.01, optim = "momentum", decay = 0.9):
                if self.activate == "Sigmoid":
                        grad = upstream_grad * derivative_sigmoid(self.z)
                elif self.activate == "ReLU":
                        grad = upstream_grad * derivative_ReLU(self.z)
                elif self.activate == "tanh":
                        grad = upstream_grad * derivative_tanh(self.z)
                else:
                        grad = upstream_grad

                if optim == "SGD":
                        self.b -= np.sum(grad) * lr
                        self.w -= np.dot(self.x.T, grad) *lr
                elif optim == "momentum":
                        self.v_w = decay * self.v_w + lr * np.dot(self.x.T, grad)
                        self.v_b = decay * self.v_b + lr * np.sum(grad)
                        self.b -= self.v_b
                        self.w -= self.v_w
                elif optim == "AdaGrad":
                        self.total_w += np.dot(self.x.T, grad) ** 2
                        self.total_b += np.sum(grad) ** 2
                        self.b -= np.sum(grad) * lr / (np.sqrt(self.total_b) + 1e-8)
                        self.w -= np.dot(self.x.T, grad) * lr / (np.sqrt(self.total_w) + 1e-8)

                return np.dot(grad,self.w.T)
```

3. Experimental Results

以下是實驗的結果，包含 linear data 與 XOR data 的預測資料與實際資料的點陣圖，以及實驗設定跟模型的準確率，最後是模型的學習曲線。

3.1. Screenshot and comparison figure

<table>
<tr><td colspan="2">Linear</td></tr>
<tr><td></td><td></td></tr>
<tr><td colspan="2">XOR</td></tr>
<tr><td></td><td></td></tr>
</table>

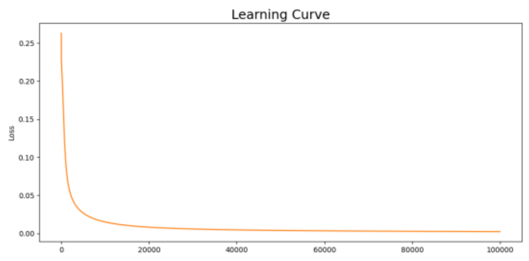
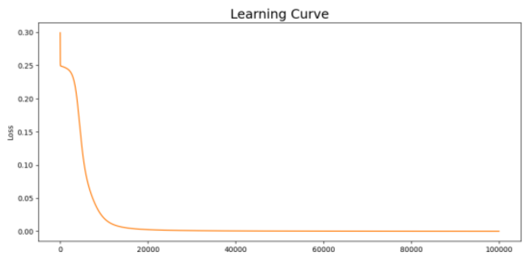
3.2. Show the accuracy of your prediction

Learning rate=0.1, hidden unit=10, optimizer=SGD, activation function: sigmoid

```
Accuracy : [1.]
Prediction :
Iter1 |    Ground truth: [0] |     prediction: [3.57137962e-05] |
Iter2 |    Ground truth: [0] |     prediction: [6.08074965e-08] |
Iter3 |    Ground truth: [0] |     prediction: [4.13789073e-08] |
Iter4 |    Ground truth: [1] |     prediction: [0.99999955] |
Iter5 |    Ground truth: [1] |     prediction: [0.66280268] |
Iter6 |    Ground truth: [0] |     prediction: [4.3006947e-06] |
Iter7 |    Ground truth: [1] |     prediction: [0.99993138] |
Iter8 |    Ground truth: [1] |     prediction: [0.99999969] |
Iter9 |    Ground truth: [1] |     prediction: [0.99999832] |
Iter10 |    Ground truth: [1] |     prediction: [0.99944955] |
```

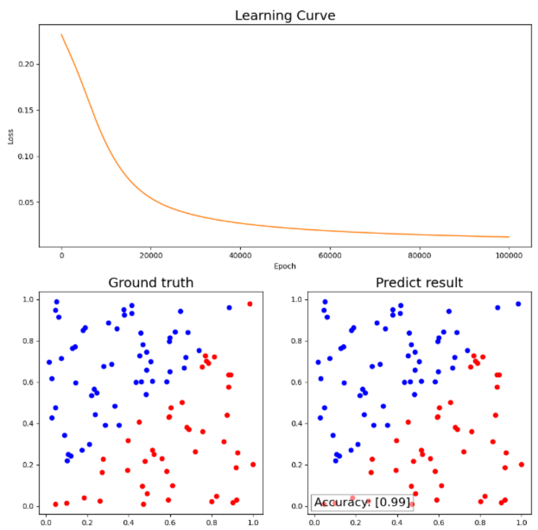
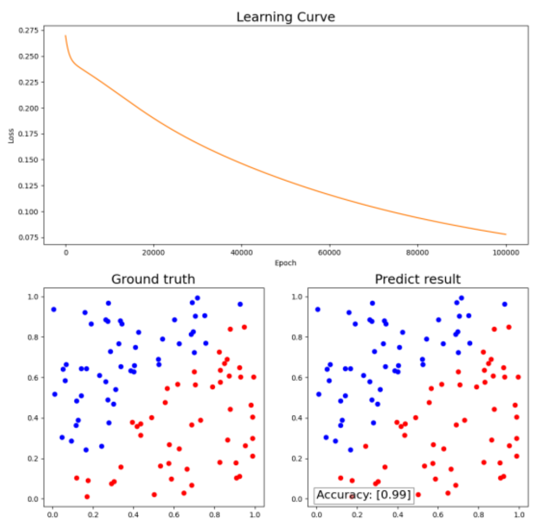
```
Accuracy : [1.]
Prediction :
Iter1 |    Ground truth: [0] |     prediction: [0.01421017] |
Iter2 |    Ground truth: [1] |     prediction: [0.99721193] |
Iter3 |    Ground truth: [0] |     prediction: [0.01416393] |
Iter4 |    Ground truth: [1] |     prediction: [0.99702673] |
Iter5 |    Ground truth: [0] |     prediction: [0.01410663] |
Iter6 |    Ground truth: [1] |     prediction: [0.99655806] |
Iter7 |    Ground truth: [0] |     prediction: [0.01404075] |
Iter8 |    Ground truth: [1] |     prediction: [0.99485846] |
Iter9 |    Ground truth: [0] |     prediction: [0.01396904] |
Iter10 |    Ground truth: [1] |     prediction: [0.9655604] |
```

3.3. Learning curve (loss-epoch curve)

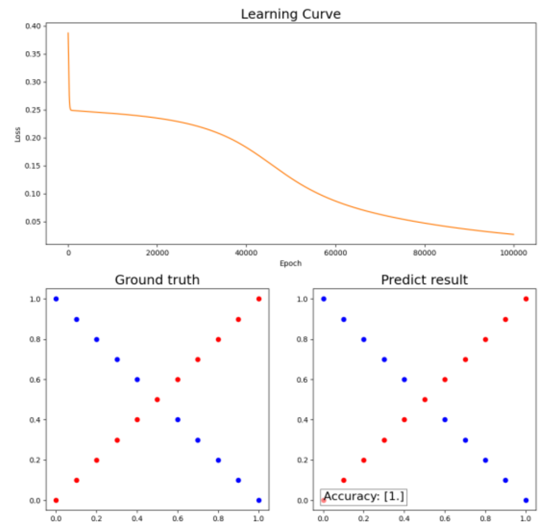
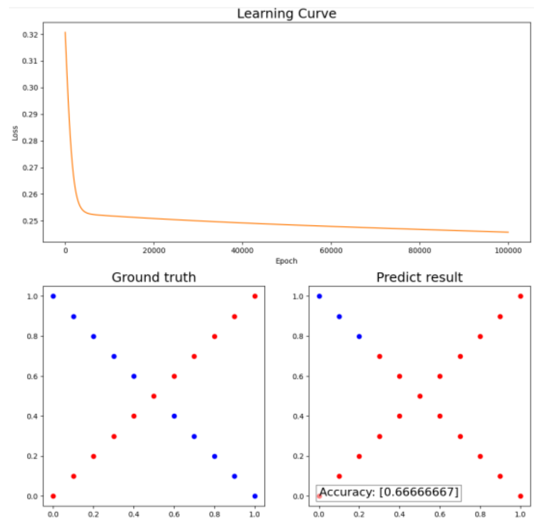
Linear	XOR
	
epoch 0 loss : 0.29946435457863946 epoch 10000 loss : 0.01979503002117869 epoch 20000 loss : 0.0026897322267163083 epoch 30000 loss : 0.0012163680854239457 epoch 40000 loss : 0.0007556068693309222 epoch 50000 loss : 0.0005391599957320184 epoch 60000 loss : 0.00041549212719210304 epoch 70000 loss : 0.000336190540220543 epoch 80000 loss : 0.0002813161950521223 epoch 90000 loss : 0.00024123864075245013	epoch 0 loss : 0.2628302901660786 epoch 10000 loss : 0.015063095652393065 epoch 20000 loss : 0.008229608051337748 epoch 30000 loss : 0.0058514940625119406 epoch 40000 loss : 0.004654812328842541 epoch 50000 loss : 0.003928908665779767 epoch 60000 loss : 0.0034358184166474117 epoch 70000 loss : 0.0030745004378831924 epoch 80000 loss : 0.0027948940663429863 epoch 90000 loss : 0.0025693919279855797

4. Discussion

4.1. Try different learning rates

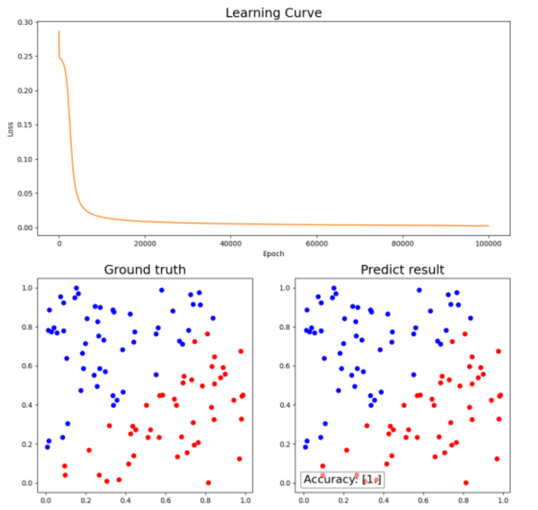
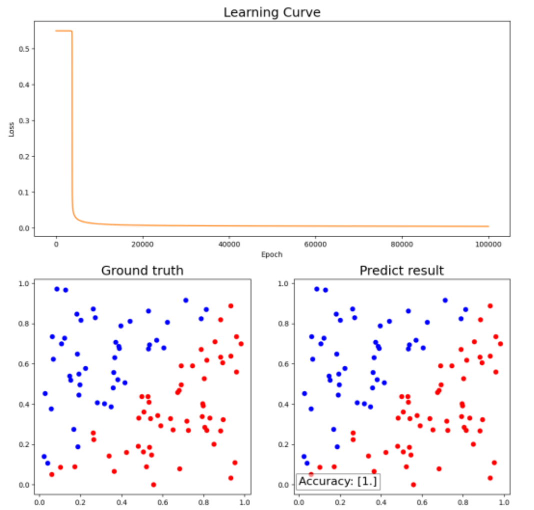
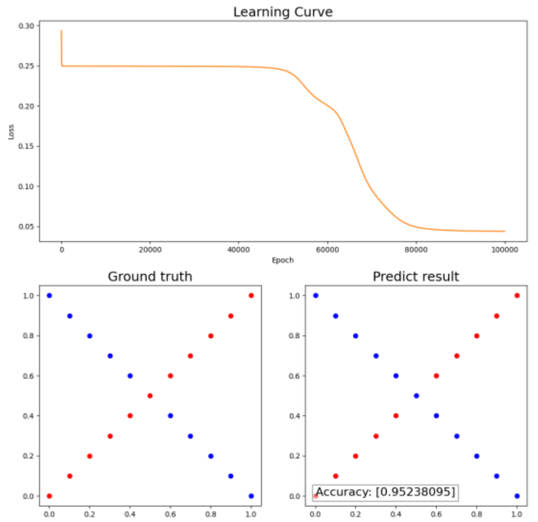
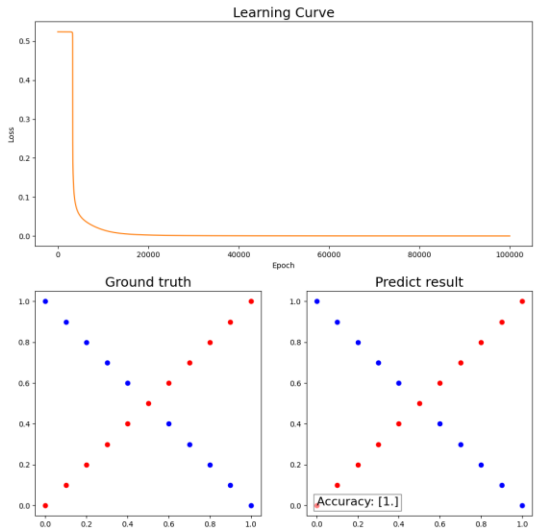
Linear	
Lr=1e-2	Lr=1e-3
	

Learning rate 越小，隨著 epoch 增加，loss 也下降的越慢，導致學習曲線下降得更慢

<table>
<tr><th colspan="2">XOR</th></tr>
<tr><td>Lr=1e-2</td><td>Lr=1e-3</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>

XOR data 訓練量較大，導致 learning rate 越小，模型預測的準確率就越低，雖然在 lr=1e-3 時，loss 已經迅速下降，但因訓練次數造成過擬和，導致模型的準確率下降

4.2. Try different numbers of hidden units

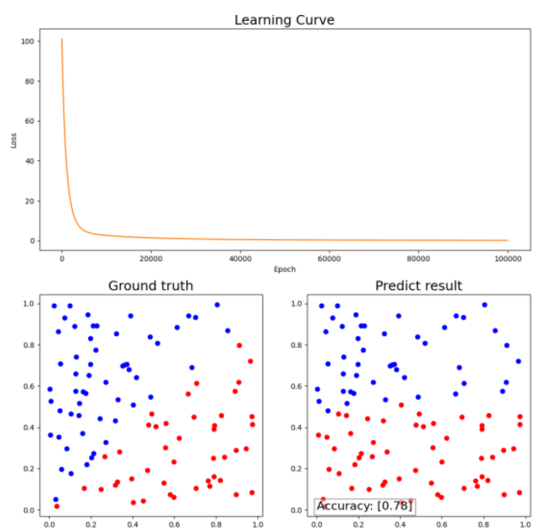
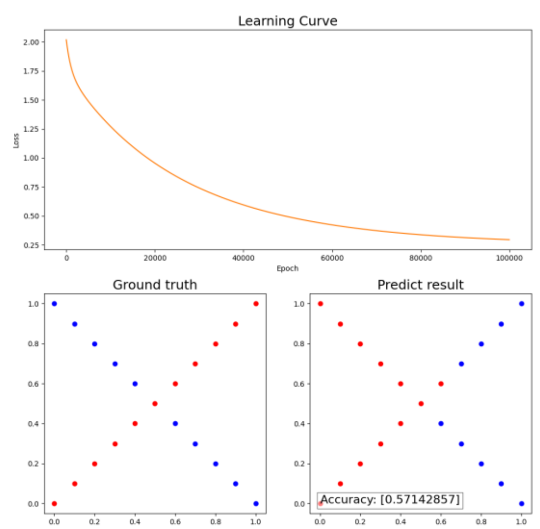
<table>
<tr><td colspan="2">Linear</td></tr>
<tr><td>hidden unit=2</td><td>hidden unit =100</td></tr>
<tr><td>
</td><td>
</td></tr>
<tr><td colspan="2">XOR</td></tr>
<tr><td>hidden unit =2</td><td>hidden unit =100</td></tr>
<tr><td>
</td><td>
</td></tr>
</table>

從 linear data 可發現 hidden size 從 2 到 100 時，學習曲線下降得更陡峭，而在 XOR data 的 hidden unit=2 時，學習曲線大約在 epoch=60000 才開始下降，但

在 hidden unit =100 時，學習曲線下降的幅度就跟 linear data 一樣，由此可知，hidden unit 越大時，模型的學習速度更快。

4.3. Try without activation functions

Lr=1e-6, hidden unit=10, optimizer=SGD

Linear	XOR
	

從圖中可以發現，若缺少 activation function，不論是 linear data 或 XOR data，他們的準確率都會下降。

5. Questions

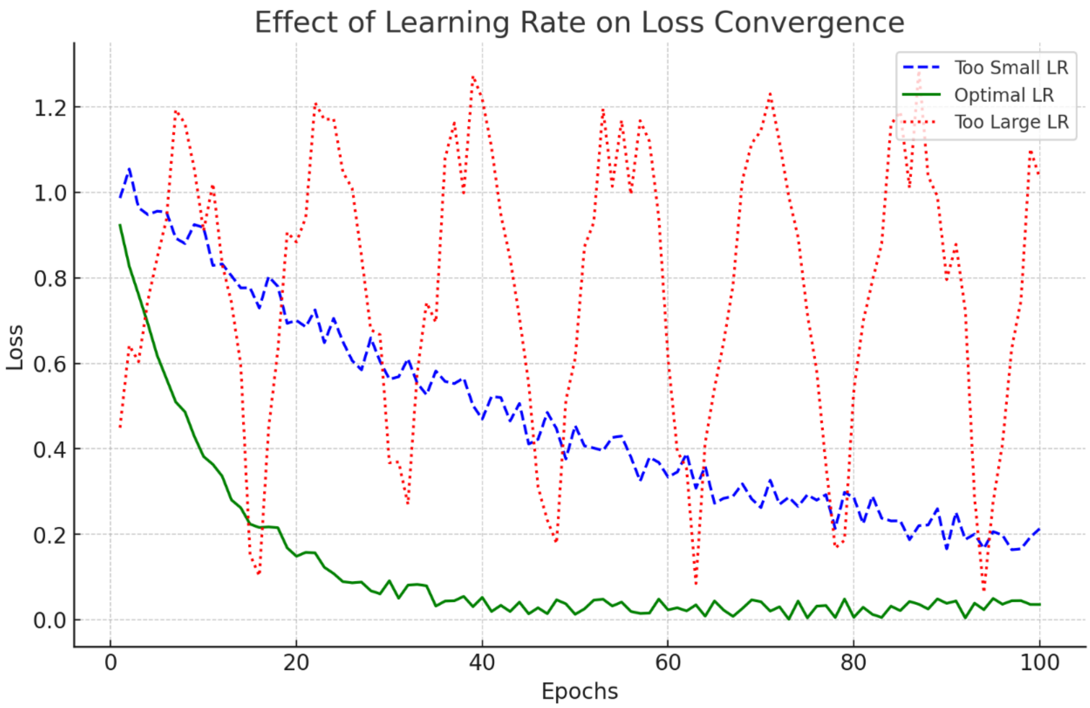
5.1. What is the purpose of activation functions?

主要目的是引入非線性特性，使神經網路能夠學習複雜模式，提供特徵選擇能力，提升模型的泛化能力，讓梯度計算更有效率，避免梯度消失與更加收斂模型，控制輸出範圍使數據穩定。

Sigmoid	ReLU	Tanh
將輸出壓縮到$(0,1)$，適用於概率輸出	輸出範圍 $(-1,1)$，可以使數據均值接近 0，有助於梯度下降	非負輸出，有助於梯度傳播並加速訓練

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

當學習率太大時，模型可能會不斷跳過最優解（optimal solution），甚至發散（diverge），導致損失（loss）值不斷上升，而不是減少，當學習率過小時，每次更新的幅度非常小，模型需要更多的迭代次數才能接近最優解，這會導致訓練時間過長，尤其是在深度學習中，可能需要數千個 epoch 才能達到良好的收斂。



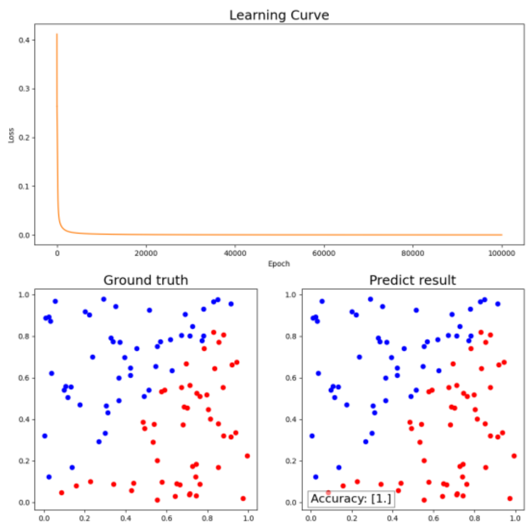
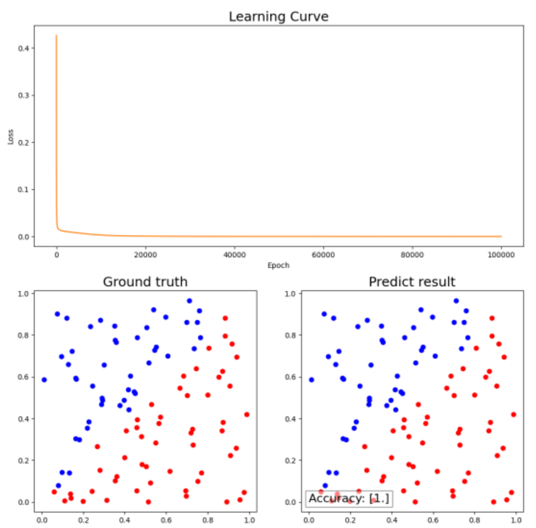
5.3. What is the purpose of weights and biases in a neural network?

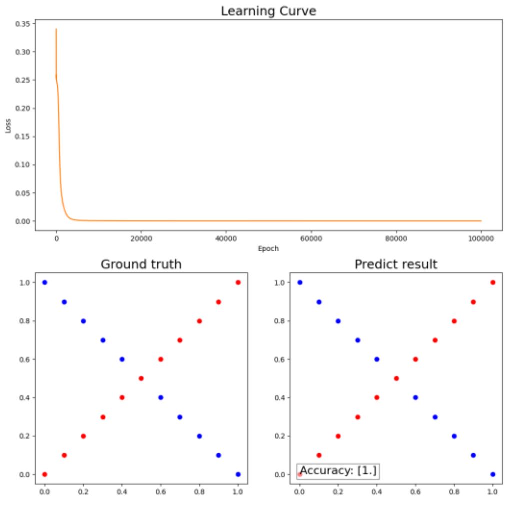
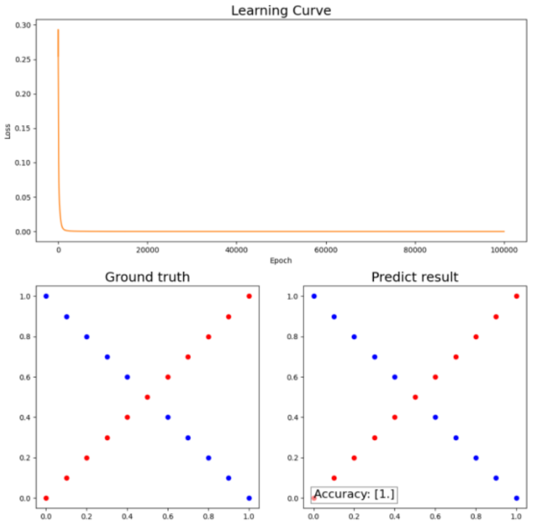
權重是連接神經元之間的係數，用來學習資料中的特徵，並調整神經網路的行為，偏差是一個額外的可調整參數，直接加在權重輸入的總和上，調整神經網路的輸出，提高靈活性和學習能力。

6. Extra

6.1. Implement different optimizers

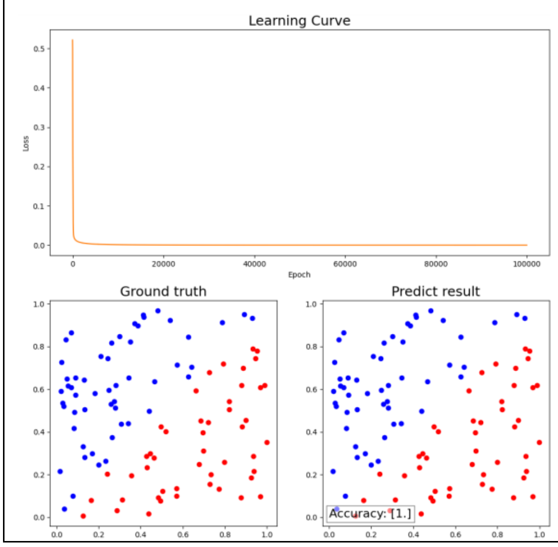
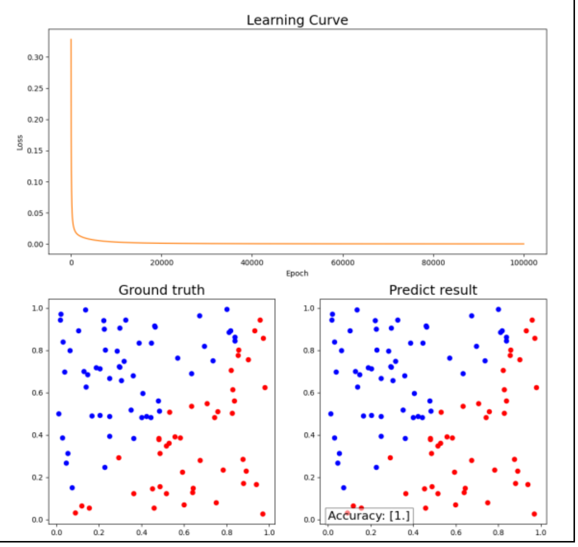
Lr=0.1, hidden unit=10

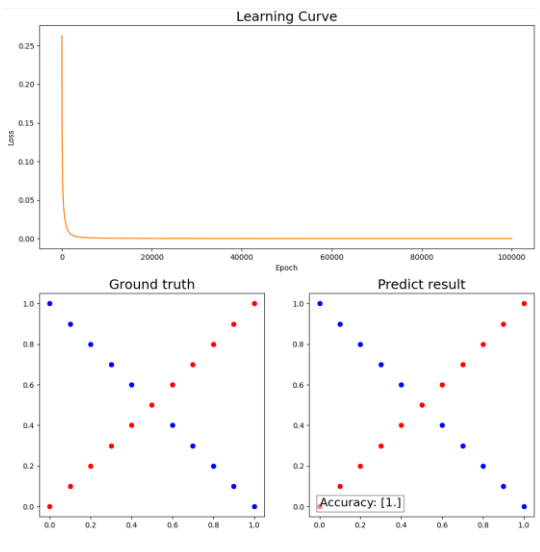
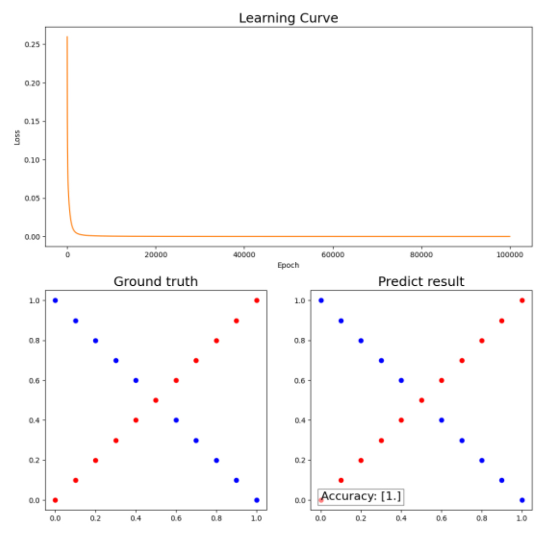
<table>
<tr><th colspan="2">Linear</th></tr>
<tr><th>momentum</th><th>AdaGrad</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

<table>
<tr><th colspan="2">XOR</th></tr>
<tr><th>momentum</th><th>AdaGrad</th></tr>
<tr><td>
</td><td>
</td></tr>
</table>

6.2. Implement different activation functions

Lr=0.1, hidden unit=10

Linear	
ReLU	tanh
	

XOR	
ReLU	tanh
	

6.3. Implement convolutional layer

Input channels=1, img size=5, conv filters=8, kernel size=3, hidden unit=10, optim=SGD, activate=ReLU, output size=1, lr=0.01

Linear