深度學習 作業二

學號：312831002　姓名：廖健棚

實作 backpropagation

SimpleNet 網路架構程式碼

```
class SimpleNet:
    def __init__(self, num_step=6000, print_interval=100, learning_rate=1e-2):
        ...
        # Model hyperparameter
        self.num_step = num_step
        self.print_interval = print_interval
        self.learning_rate = learning_rate
        self.lr_gamma = 0.5 # learning rate schedule
        self.lr_epoch = 1500 # learning rate schedule
        self.momentum = 0.9 # momentum
        # Model parameters initialization
        self.error = None
        self.inputs = None # Initialize input layer
        self.hidden1_weights = np.random.randn(2, 100)  # Initialize weights for hidden layer 1
        self.hidden1_biases = np.zeros((1, 100))  # Initialize biases for hidden layer 1
        self.hidden1_output = None
        self.update1 = np.zeros_like(self.hidden1_weights) # update part for hidden layer 1

        self.hidden2_weights = np.random.randn(100, 50)  # Initialize weights for hidden layer 2
        self.hidden2_biases = np.zeros((1, 50))  # Initialize biases for hidden layer 2
        self.hidden2_output = None
        self.update2 = np.zeros_like(self.hidden2_weights) # update part for hidden layer 2

        self.hidden3_weights = np.random.randn(50, 10)  # Initialize weights for hidden layer 3
        self.hidden3_biases = np.zeros((1, 10))  # Initialize biases for hidden layer 3
        self.hidden3_output = None
        self.update3 = np.zeros_like(self.hidden3_weights) # update part for hidden layer 3

        self.output_weights = np.random.randn(10, 1)  # Initialize weights for the output layer
        self.output_biases = np.zeros((1, 1))  # Initialize biases for the output layer
        self.output = None
        self.updateo = np.zeros_like(self.output_weights)
```

SimpleNet forward 程式碼

```
def forward(self, inputs):
    ...
   # input layer
    self.inputs = inputs
    self.hidden1_output = sigmoid(np.dot(inputs, self.hidden1_weights) + self.hidden1_biases)
    self.hidden2_output = sigmoid(np.dot(self.hidden1_output, self.hidden2_weights) + self.hidden2_biases)
    self.hidden3_output = sigmoid(np.dot(self.hidden2_output, self.hidden3_weights) + self.hidden3_biases)
    output = sigmoid(np.dot(self.hidden3_output, self.output_weights) + self.output_biases)

    return output
```

SimpleNet backward 程式碼

```
def backward(self):
    ...
    # Compute the gradient of the error with respect to the output
    d_error_output = self.error * der_sigmoid(self.output)
    # Backpropagate the gradient through the network (Chain rule)
    d_error_hidden3 = d_error_output.dot(self.output_weights.T) * der_sigmoid(self.hidden3_output)
    d_error_hidden2 = d_error_hidden3.dot(self.hidden3_weights.T) * der_sigmoid(self.hidden2_output)
    d_error_hidden1 = d_error_hidden2.dot(self.hidden2_weights.T) * der_sigmoid(self.hidden1_output)
    # Update the weights and biases using the gradients
    self.updateo = self.momentum * self.updateo + self.learning_rate *
                   self.hidden3_output.T.dot(d_error_output)
    self.output_weights -= self.updateo
    self.output_biases -= self.learning_rate * np.ones_like(d_error_output).dot(d_error_output.T)
    self.update3 = self.momentum * self.update3 + self.learning_rate *
                   self.hidden2_output.T.dot(d_error_hidden3)
    self.hidden3_weights -= self.update3
    self.hidden3_biases -= self.learning_rate * np.ones_like(d_error_hidden3).dot(d_error_hidden3.T)
    self.update2 = self.momentum * self.update2 + self.learning_rate *
                   self.hidden1_output.T.dot(d_error_hidden2)
    self.hidden2_weights -= self.update2
    self.hidden2_biases -= self.learning_rate * np.ones_like(d_error_hidden2).dot(d_error_hidden2.T)
   self.update1 = self.momentum * self.update1 + self.learning_rate * self.inputs.T.dot(d_error_hidden1)
    self.hidden1_weights -= self.update1
    self.hidden1_biases -= self.learning_rate * np.ones_like(d_error_hidden1).dot(d_error_hidden1.T)
```

SimpleNet learning rate schedule 程式碼

```python
def update_lr(self):
    self.learning_rate *= 1 - self.lr_gamma

def train(self, inputs, labels):
    ...

    # LR schedule
    if epochs % self.lr_epoch == 0 and epochs != 0:
        self.update_lr()
```

SimpleNet 網路架構程解說

這次訓練主要使用到 SGD 的 optimize 並且額外使用 Momentum 及 Learning Rate Schedule 用來提升模型表現。

- lr_gamma = 0.5 用來控制學習率的數值，更新後 $lr = lr \times (1 - lr\_gamma)$
- lr_epoch = 1500 當代次數到達 lr_epoch，更新 lr
- momentum = 0.9 相較於純 SGD 方式，使用動量方法可以讓模型收斂得更快更好

Momentum

模擬物理動量的概念，在同方向的維度上學習會變快，反方向則學習變慢，可以讓模型不會卡在 local minima 的地方。

- inputs: 輸入層，會在 forward 時更新 input
- hidden_layer [100, 50, 10] 取第一層做舉例
  - weights: 權重會是串接上一層輸出的維度及下一層輸入維度的矩陣 $W_{2\times100}$
  - biases: 偏差是在神經元輸出後加上的與 nodes 數量一樣的 1 維向量 $b_{1\times100}$
  - hidden_output: 在 back propagation 時需要用到，需要紀錄 forward 的過程
  - update: momentum 用來更新 weights，與 weights 的矩陣大小一樣

SimpleNet forward 解說

- self.inputs = input: 初始化 input
- forward: $\sigma(xw + bias)$

  sigmoid(np.dot(inputs, self.hidden_weights) + self.hidden1_biases)
- np.dot: 矩陣相乘
- output: forward 輸出層

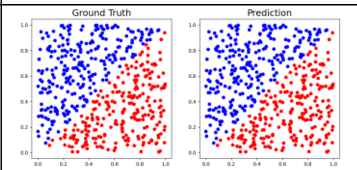
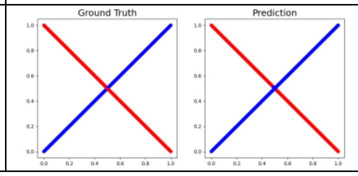
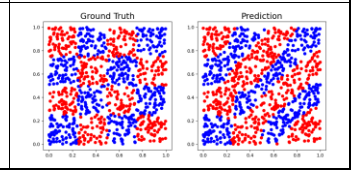
SimpleNet backward 解說

- d_error_output: 輸出的梯度，error_output 乘上 sigmoid 的偏微分
- Chain Rule gradient (weight)
  - backward pass: d_error_output 乘上 weight 再 * sigmoid 的偏微分
  - forward pass: hidden3_output
  - Gradient: (forward pass)乘上(backward pass)
- Update 參數:
  - update (momentum): $U_t = m * U_{t-1} + \eta \Delta W_t$ 保留部分上一次的 gradient
  - weight: $W_t = W_t - U_t$
  - bias: $b_t = b_t - \eta \Delta b_t$

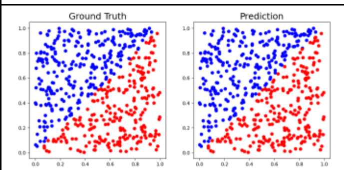
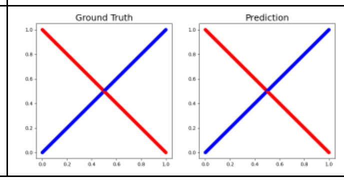
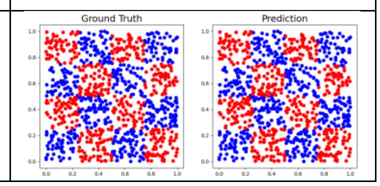
learning rate schedule 解說

- 學習率更新: $lr = lr * (1 - gamma)$
- 因為模型與資料較為簡單，加上訓練代次比較少，所以不需要太頻繁更新學習率，因此我只有在 1500 代時更新學習率

訓練結果

SGD	Linear	XOR	Chess board
Test Accuracy	98.96%	96.58%	75.60%
Graph			

只有使用單純的 SGD 做訓練，訓練結果在 Linear 及 XOR 都有不錯的表現，然而在 Chess board 的分類表現就差強人意。

Momentum	Linear	XOR	Chess board
Test Accuracy	99.12%	98.81%	96.57%
Graph			

使用 Momentum 過後，在 Linear 及 XOR 的表現有些微上升，尤其是在 Chess board 的表現上有卓越的提升，這說明 momentum 能夠克服較為複雜的訓練。

參考資料

i. Papers with code - SGD with momentum explained. Explained | Papers With Code. (n.d.-a). https://paperswithcode.com/method/sgd-with-momentum

ii. GGWithRabitLIFE. (2018, August 5). [機器學習 ML note]sgd, momentum, AdaGrad, Adam Optimizer. Medium. https://medium.com/%E9%9B%9E%E9%9B%9E%E8%88%87%E5%85%94%E5%85%94%E7%9A%84%E5%B7%A5%E7%A8%8B%E4%B8%96%E7%95%8C/%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92ml-note-sgd-momentum-adagrad-adam-optimizer-f20568c968db

iii. OpenAI. (2023). ChatGPT (Mar 14 version) [Large language model]. https://chat.openai.com/chat