

Deep-Learning-for-Autonomous-Driving

Lab1 : back propagation

ID : 310605007

Member : 鄭晴立

Department : 機器人學程

目錄

表目錄	2
圖目錄	2
1. Introduction	3
2. Experiment setups	4
2.1. Activation Functions	4
2.2. Neural network.....	5
2.3. Backpropagation and Optimizer	6
2.4. Data augmentation	7
3. Results of your testing	8
4. Discussion	9
4.1. Data augmentation	9
4.2. 不同 learning rate.....	9
4.3. 將模型加深加大.....	10
4.4. 不同的 Batch size	11
4.5. Sigmoid Function.....	12

表目錄

表 一 Neural Network 相關資料.....	3
表 二 最終模型參數.....	8

圖目錄

圖 一 Sigmoid functions code.....	4
圖 二 Softmax functions code.....	4
圖 三 Relu functions code.....	5
圖 四 利用 numpy 創建三層 Network	5
圖 五 所有 layer 的 Backpropagation	6
圖 六 利用 Optimizer 更新參數	6
圖 七 Data augmentation code.....	7
圖 八 Data augmentation	7
圖 九 Learning rate = 0.005	9
圖 十 Learning rate = 0.01	10
圖 十一 Learning rate = 0.1	10
圖 十二 加大網路.....	10
圖 十三加深網路.....	11
圖 十四 batch size = 1024	11
圖 十五 batch size = 512	11
圖 十六 全部改為 sigmoid learning rate 為 0.0001.....	12
圖 十七 全部改為 sigmoid learning rate 為 0.01.....	12
圖 十八 2 兩層改為 sigmoid learning rate 為 0.01.....	12

1. Introduction

本次實驗要分辨 47 類 28*28 的圖片，並用 numpy 去實作網路，程式部分大多按照助教所給提示及檔案去完成，盡量不去做大幅度的修改。

表 一 Neural Network 相關資料

Parameter	Neural Network
Number of input layer	784
Number of hidden layer	2~5
Number of output layer	47
Activation function	Relu、Sigmoid、Softmax
Optimizer	Adam、SDG
Learning rate	0.01~0.0001
Epoch	50~1000
Loss function	Mean Squared Error
Batch size	256~1024

2. Experiment setups

2.1. Activation Functions

本實驗最後一層採用 softmax，其餘則嘗試 relu 及、sigmoid，由於本次為 47 分類的圖像辨識問題，relu 的結果應該會較佳，最後與預期符合，以下為 Activation Functions 的說明與實作。

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

```
class sigmoid():
    def __init__(self):
        pass

    def forward(self, x:np.array):
        self.inputs = x
        return self.sigmoid(x)

    def sigmoid(self, x:np.array):
        return 1.0 / (1.0 + np.exp(-x))

    def backward(self, dy:np.array):
        dx = self.sigmoid(self.inputs) * (1 - self.sigmoid(self.inputs)) * dy
        return dx
```

圖 一 Sigmoid functions code

```
class SoftmaxWithloss(_Layer):
    def __init__(self):
        self.soft_x=None

    def forward(self, input):

        self.soft_x = np.zeros(shape = (input.shape[0], input.shape[1]))
        for i in range(input.shape[0]):
            self.soft_x[i] = (np.exp(input[i]-np.max(input[i]))/sum(np.exp(input[i]-np.max(input[i]))))
        predict = self.soft_x

        return predict
    def loss(self,target):
        diff = self.soft_x - target
        differences_squared = diff ** 2
        your_loss = differences_squared.mean()
        return your_loss

    def backward(self, a3:np.array, target:np.array):
        input_grad = (a3-target) /len(target)
        return input_grad
```

圖 二 Softmax functions code

```

class relu(_Layer):
    def __init__(self):
        pass

    def forward(self, input):
        output = (np.maximum(0, input))
        return output

    def backward(self, z):
        dz = np.heaviside(z,1)
        return dz

```

圖 三 Relu functions code

2.2. Neural network

本實驗使用三到五層網路，其模型示意圖，經過實驗發現，加深網路及加大網路，皆無法使結果更好，其實程式說明如圖 四。

```

layers = [
    layer.FullyConnected(28*28, 1024) ,
    layer.relu() ,
    layer.FullyConnected(1024, 512),
    layer.relu() ,
    layer.FullyConnected(512, 256),
    layer.relu() ,
    layer.FullyConnected(256, 47),
    layer.SoftmaxWithloss()
]
net = Network(layers, Learning_rate, opt)

```

圖 四 利用 numpy 創建三層 Network

2.3. Backpropagation and Optimizer

有別於助教給的 code，我把 Optimizer 放在 Backpropagation 後面，實作上相對容易，backward 之後直接更新參數，其程式如圖 五、圖 六。

```
def backward(self, target):  
  
    self.h_last_grad = self.layers[-1].backward(self.pred, target)  
    grad=self.h_last_grad  
    for i in range(int((len(self.layers))/2)-1):  
        act_grad=self.layers[-3-2*i].backward(self.act_f[i+1])  
        grad=self.layers[-2-2*i].backward(act_grad,grad,self.lr,self.opt)
```

圖 五 所有 layer 的 Backpropagation

```
def backward(self, act_grad3,h4_grad,lr,opt):  
  
    self.d = np.multiply(act_grad3, self.weight.dot(h4_grad.transpose()).transpose())  
  
    if opt.name == 'adam':  
        beta1 = 0.9  
        beta2 = 0.999  
        epsilon = 1e-8  
        if self.trained == 0:  
            self.m = np.zeros((self.in_features, self.out_features))  
            self.v = np.zeros((self.in_features, self.out_features))  
  
            self.m_b = np.zeros((1, self.out_features))  
            self.v_b = np.zeros((1, self.out_features))  
        self.m = beta1 * self.m + (1 - beta1) * self.input.transpose().dot(h4_grad)  
        self.v = beta2 * self.v + (1 - beta2) * (self.input.transpose().dot(h4_grad)** 2)  
        m_hat = self.m / (1 - beta1)  
        v_hat = self.v / (1 - beta2)  
        self.weight = self.weight - lr * m_hat / np.sqrt(v_hat + epsilon)  
  
        self.m_b = beta1 * self.m_b + (1 - beta1) * np.sum(h4_grad, axis=0)  
        self.v_b = beta2 * self.v_b + (1 - beta2) * ( np.sum(h4_grad, axis=0) ** 2)  
        m_b_hat = self.m_b / (1 - beta1)  
        v_b_hat = self.v_b / (1 - beta2)  
        self.bias = self.bias - lr * m_b_hat / np.sqrt(v_b_hat + epsilon)  
    elif opt.name == 'SGD':  
        for i in range(len(self.weight)):  
            self.weight-= lr*(self.input.transpose().dot(h4_grad))  
            self.bias-= lr*np.sum(h4_grad, axis=0)  
    self.trained = 1  
    return self.d
```

圖 六 利用 Optimizer 更新參數

2.4. Data augmentation

我嘗試使用 Data augmentation 去改善 overfitting，我將圖片往八個角落移動 1 個 pixel，使我的資料有九種變化，最終使精準度有效提升。

```
print("After shift")
shiftpoint=3
imageleft=np.zeros((28,28), dtype=int)
imageright=np.zeros((28,28), dtype=int)
magedown=np.zeros((28,28), dtype=int)
mageupleft=np.zeros((28,28), dtype=int)
magedownleft=np.zeros((28,28), dtype=int)
mageupright=np.zeros((28,28), dtype=int)
magedownright=np.zeros((28,28), dtype=int)
train_data_aug1=np.zeros((100000, 784), dtype=int)
train_data_aug2=np.zeros((100000, 784), dtype=int)
train_data_aug3=np.zeros((100000, 784), dtype=int)
train_data_aug4=np.zeros((100000, 784), dtype=int)
for index in range(len(train_data)):
    image = train_data[index].reshape(28,28)

    for i in range(28-shiftpoint):
        imageleft[i]=image[i+shiftpoint]
        imageright[i]=image[i-shiftpoint]

    for i in range(28-shiftpoint):
        mageupleft[:,i]=imageleft[:,i+shiftpoint]
        magedownleft[:,i]=imageleft[:,i-shiftpoint]
        mageupright[:,i]=imageright[:,i+shiftpoint]
        magedownright[:,i]=imageright[:,i-shiftpoint]
    train_data_aug1[index]=mageupleft.reshape(784,)
    train_data_aug2[index]=magedownleft.reshape(784,)
    train_data_aug3[index]=mageupright.reshape(784,)
    train_data_aug4[index]=magedownright.reshape(784,)
```

圖 七 Data augmentation code





圖 八 Data augmentation

3. Results of your testing

經過反覆測試，利用此模型，最高可達 86.679% 左右的精確度。

表 二 最終模型參數

Parameter	Neural Network
Number of input layer	784
Number of hidden layer	3
Number of output layer	47
Activation function	Relu 、 Softmax
Optimizer	Adam
Learning rate	0.0001
Epoch	500
Loss function	Mean Squared Error
Batch size	256

#	Team	Members	Score	Entries	Last	Code	Join
1	0810885		0.87570	9	1h		
2	310605007		0.86679	20	10h		


 Your Best Entry!
Your submission scored 0.85921, which is not an improvement of your previous score. Keep trying!

圖 九 最高 Accuracy

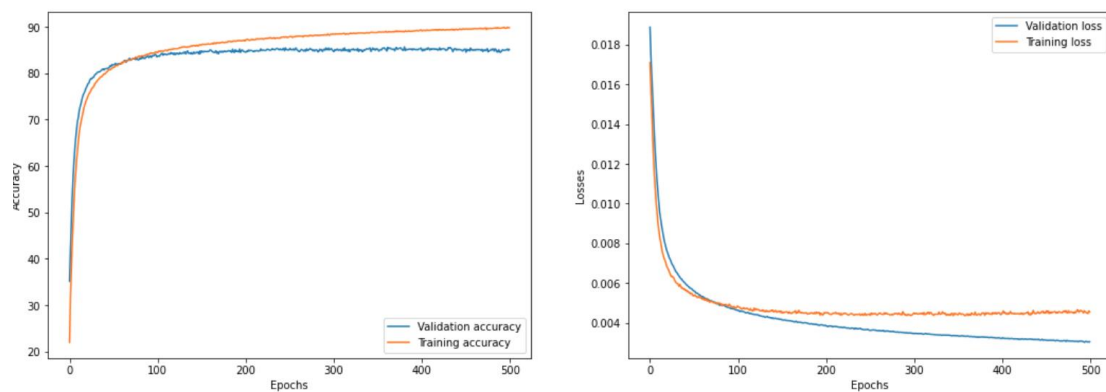


圖 十 Loss 及 Accuracy

4. Discussion

因為時間關係，大多沒有完成訓練，僅訓練到大致看得出結果為止。

4.1. Data augmentation

大幅改善 overfitting 的問題。

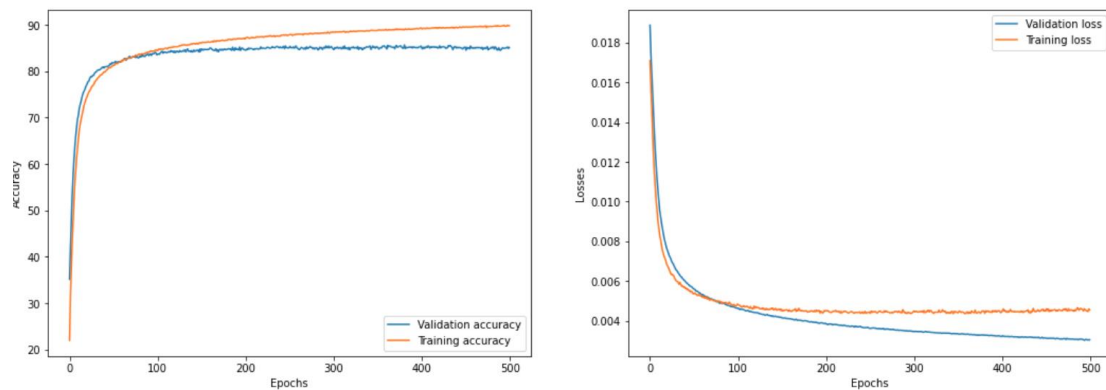


圖 十一利用 Data augmentation 去改善模型

4.2. 不同 learning rate

我嘗試了不同的 learning rate，在 learning rate > 0.001 之後，Accuracy 幾乎都很小，很快就發生，也很容易趨向發生 overfitting。

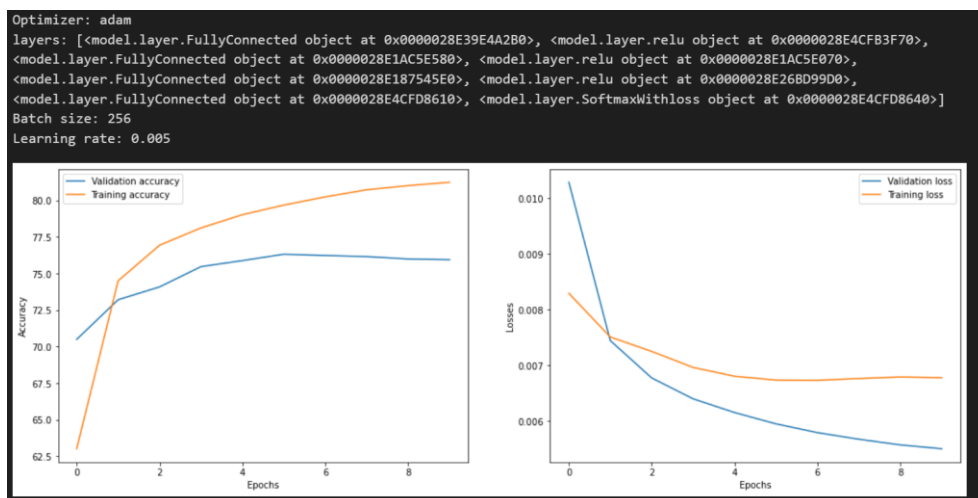


圖 十二 Learning rate = 0.005

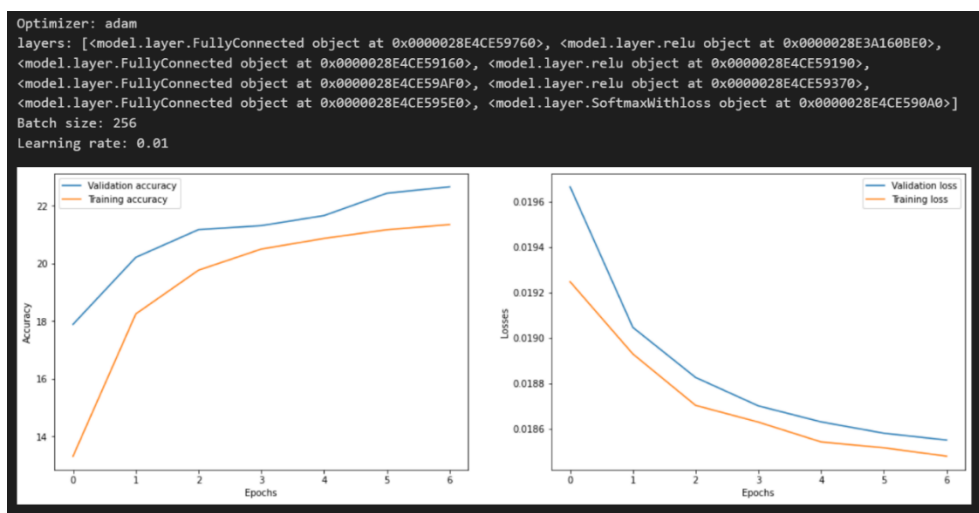


圖 十三 Learning rate = 0.01

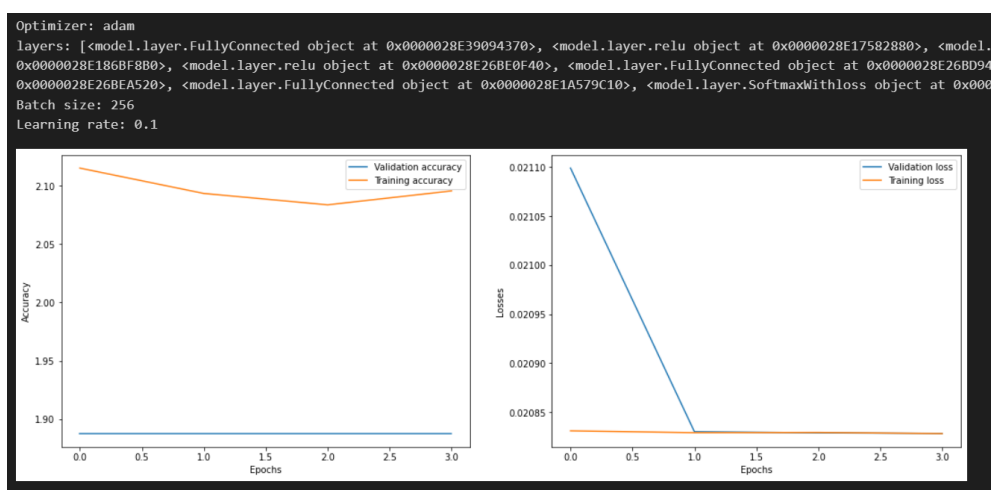


圖 十四 Learning rate = 0.1

4.3. 將模型加深加大

嘗試將模型加大加深，也是非常容易 overfitting，但結果相對好。

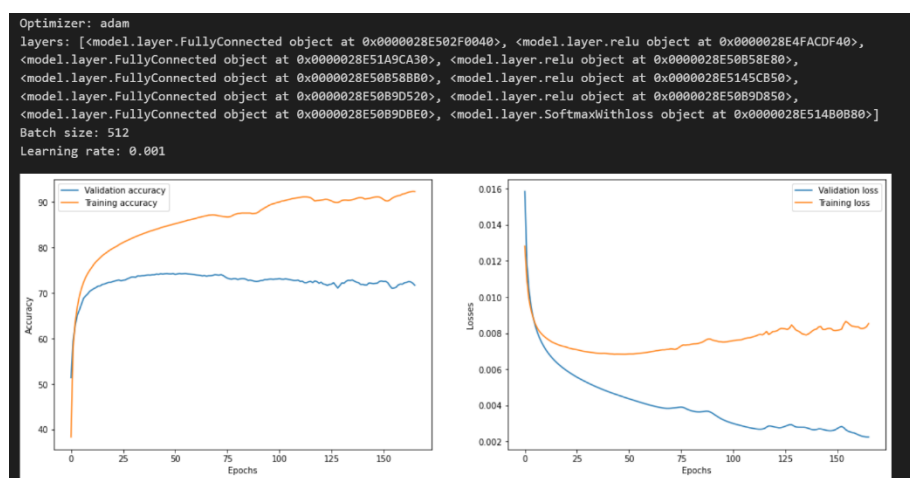


圖 十五 加大網路

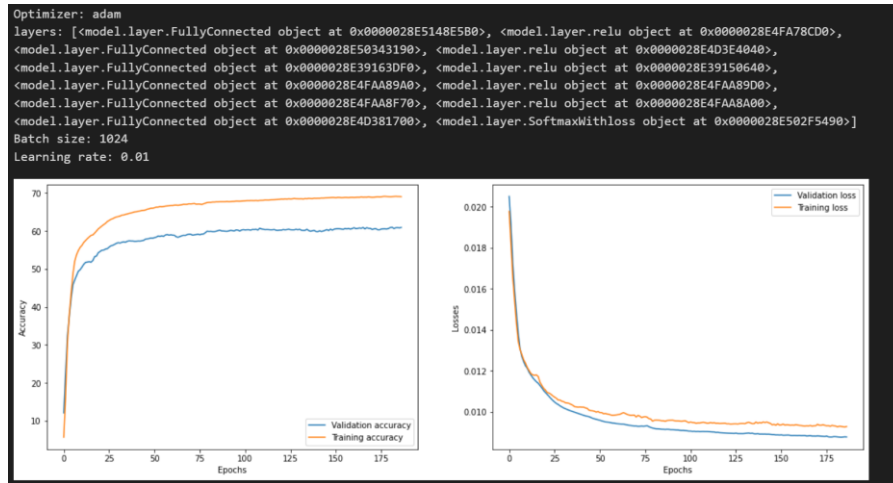


圖 十六加深網路

4.4. 不同的 Batch size

經過多次嘗試，不同 batch size 對結果影響不大，主要還是在模型架構的調整。

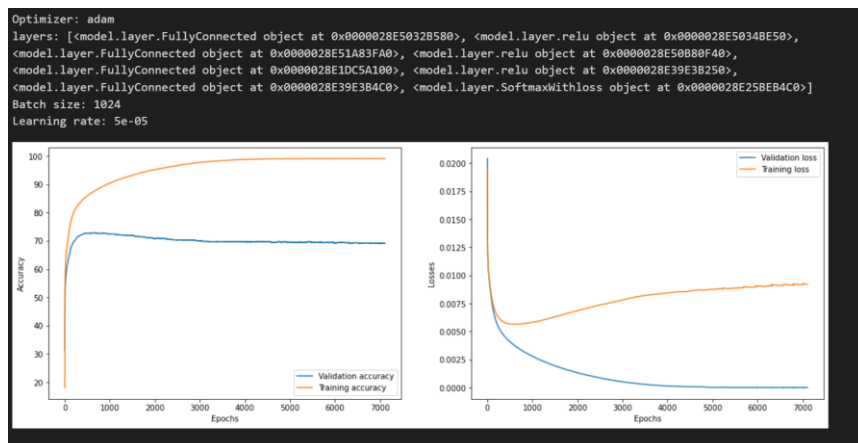


圖 十七 batch size = 1024

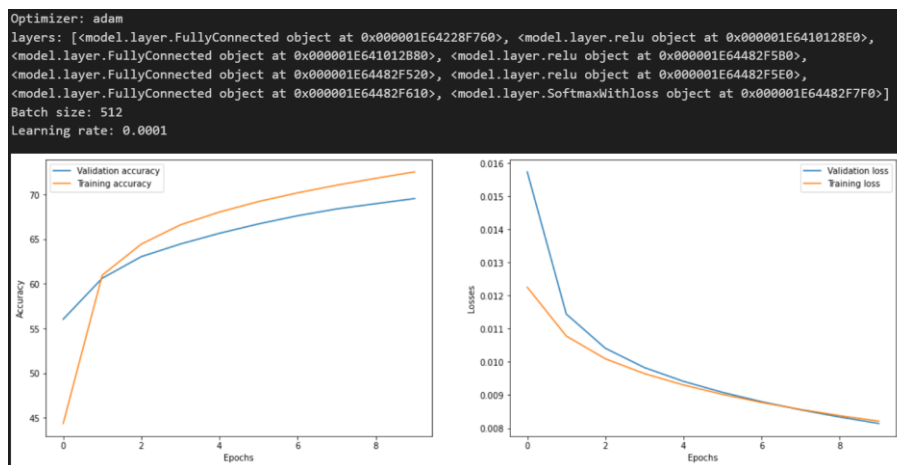


圖 十八 batch size = 512

4.5. Sigmoid Function

嘗試使用 sigmoid 的結果，accuracy 皆無法有效提升，因此，最後使用 relu 作為 Activation Function。

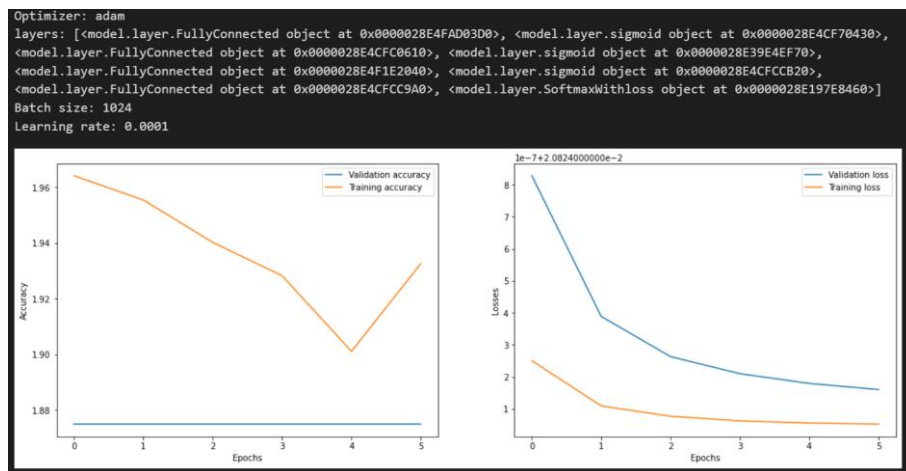


圖 十九 全部改為 sigmoid learning rate 為 0.0001

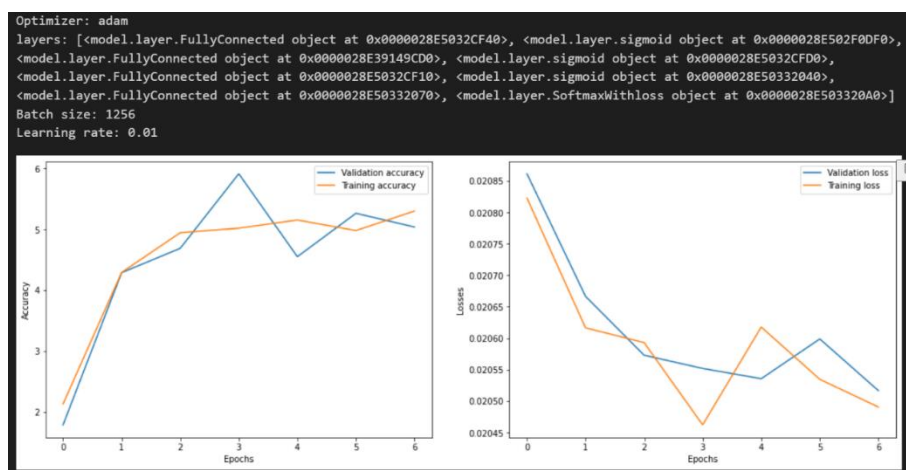


圖 二十 全部改為 sigmoid learning rate 為 0.01

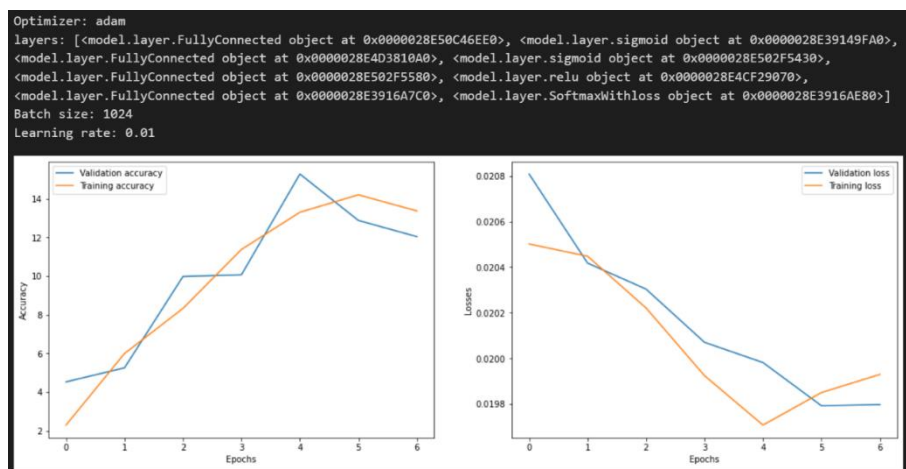


圖 二十一 2 兩層改為 sigmoid learning rate 為 0.01