NCTU DLP
Lab1: Back-propagation Implementation Report

廖家鴻 0786009
2020/4/7

Outline

- Introduction
- Experimental Setup
  - a. Sigmoid functions
  - b. Neural network
  - c. Back-propagation
- Experimental Result
  - a. Screenshot and comparison figure
  - b. Anything you want to share
- Discussion and extra experiments

Introduction

- **Experimental Setup**
  - a. Sigmoid functions
  - b. Neural network
  - c. Back-propagation
- **Experimental Result**
  - a. Screenshot and comparison figure
  - b. Anything you want to share
- **Discussion and extra experiments**

在Experimental Setup中，會說明

1. sigmoid函數的微分&coding
2. 神經網絡的架構、參數初始化、forward的設置及loss function
3. 會推導back propagation的過程及相對應的coding。

在Experimental Result中，會呈現

1. Spec所指定的loss &accuracy隨著epoch變化的過程、model prediction的機率輸出、prediction與ground truth的比較圖
2. 用linear data plot來呈現分類訓練的階段過程
3. 用XOR data plot來實驗不同的hidden unit數的相關影響與探討。
4. Test data觀察 & 分類邊界探討

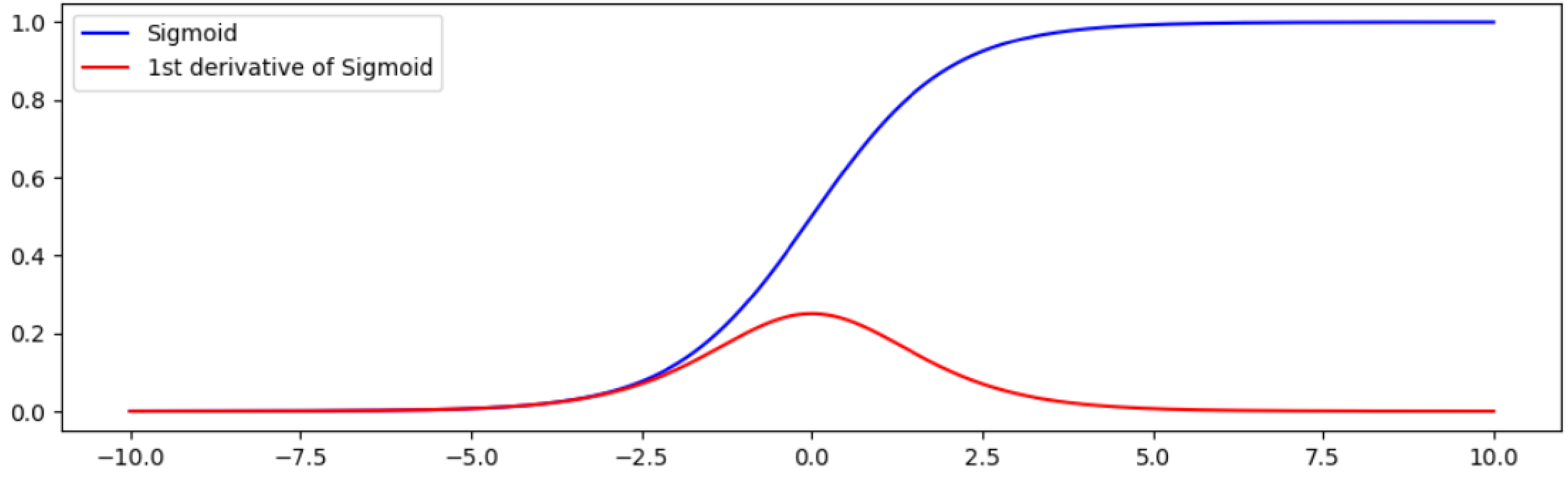
最後，會以accuracy history plot來觀察不同的hidden unit數、learning rate、XOR&Linear data的影響在收斂性。

以及用new test data輸入到trained model來觀察此兩層NN分類器的分類行為。

Experimental Setup

Experimental Setup-Sigmoid Function

右圖為Sigmoid function本身與其一階微分的curve plot



以下為Sigmoid function的一階微分推導：

$$y=\frac{1}{1+e^{-x}} \Rightarrow y'=\frac{-1}{(1-e^{-x})^2}(-e^{-x}) = y^2 e^{-x} \quad \text{—— ①}$$

$$\text{又 } y(1+e^{-x})=1 \Rightarrow e^{-x}=\frac{1-y}{y} \quad \text{—— ②}$$

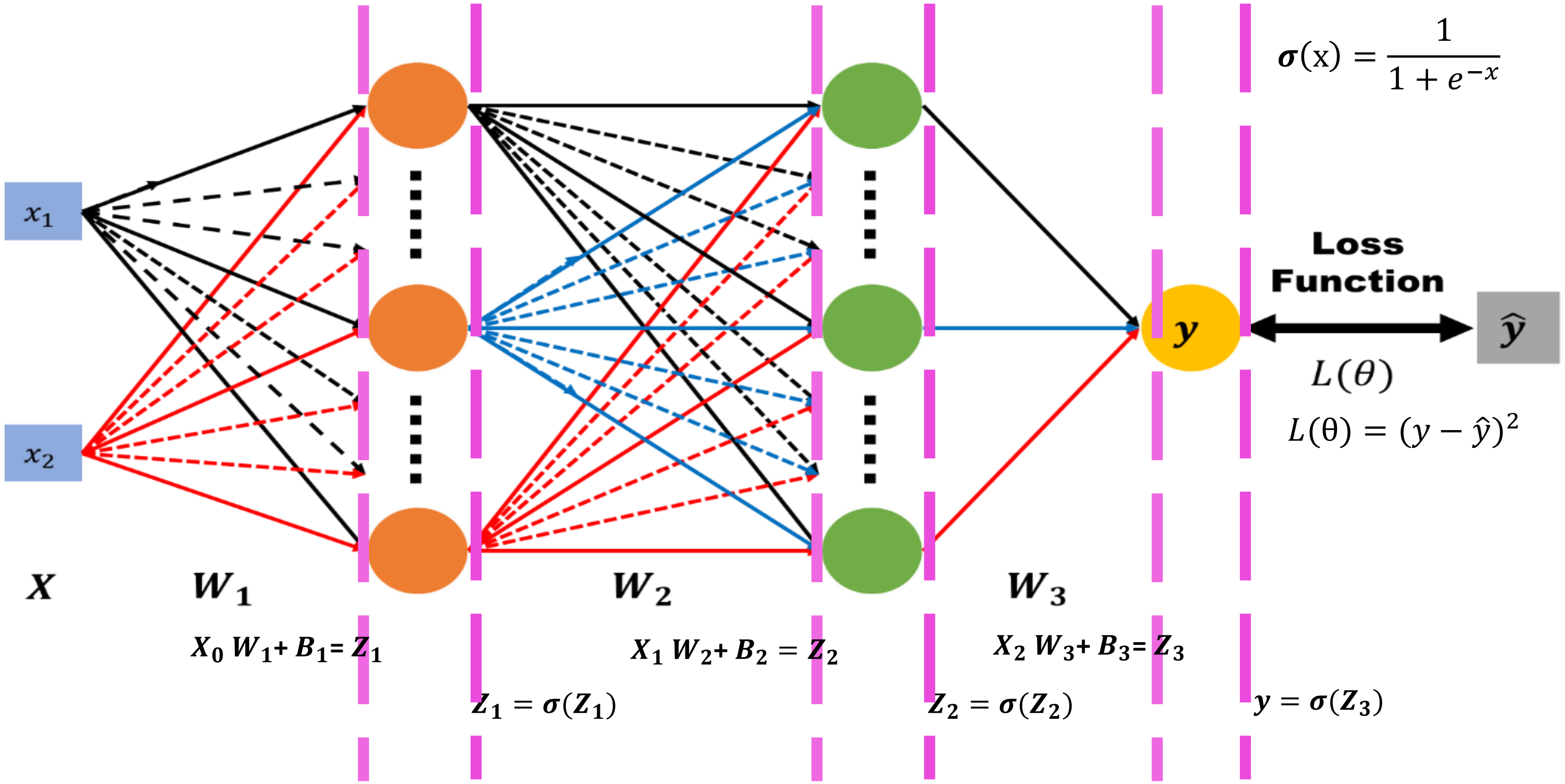
$$\therefore \text{由 ① and ②} \Rightarrow y'=y^2\cdot\frac{1-y}{y} = y\cdot(1-y)_{\#}$$

以下為Sigmoid function的一階微分推導：

```python
def sigmoid(x):
    """ Sigmoid function.
    This function accepts any shape of np.ndarray object
    as input and perform sigmoid operation.
    """
    return 1 / (1 + np.exp(-x))

def der_sigmoid(y):
    """ First derivative of Sigmoid function.
    The input to this function should be the value that
    output from sigmoid function.
    """
    return y * (1 - y)
```

Experimental Setup – Structure & Initialization

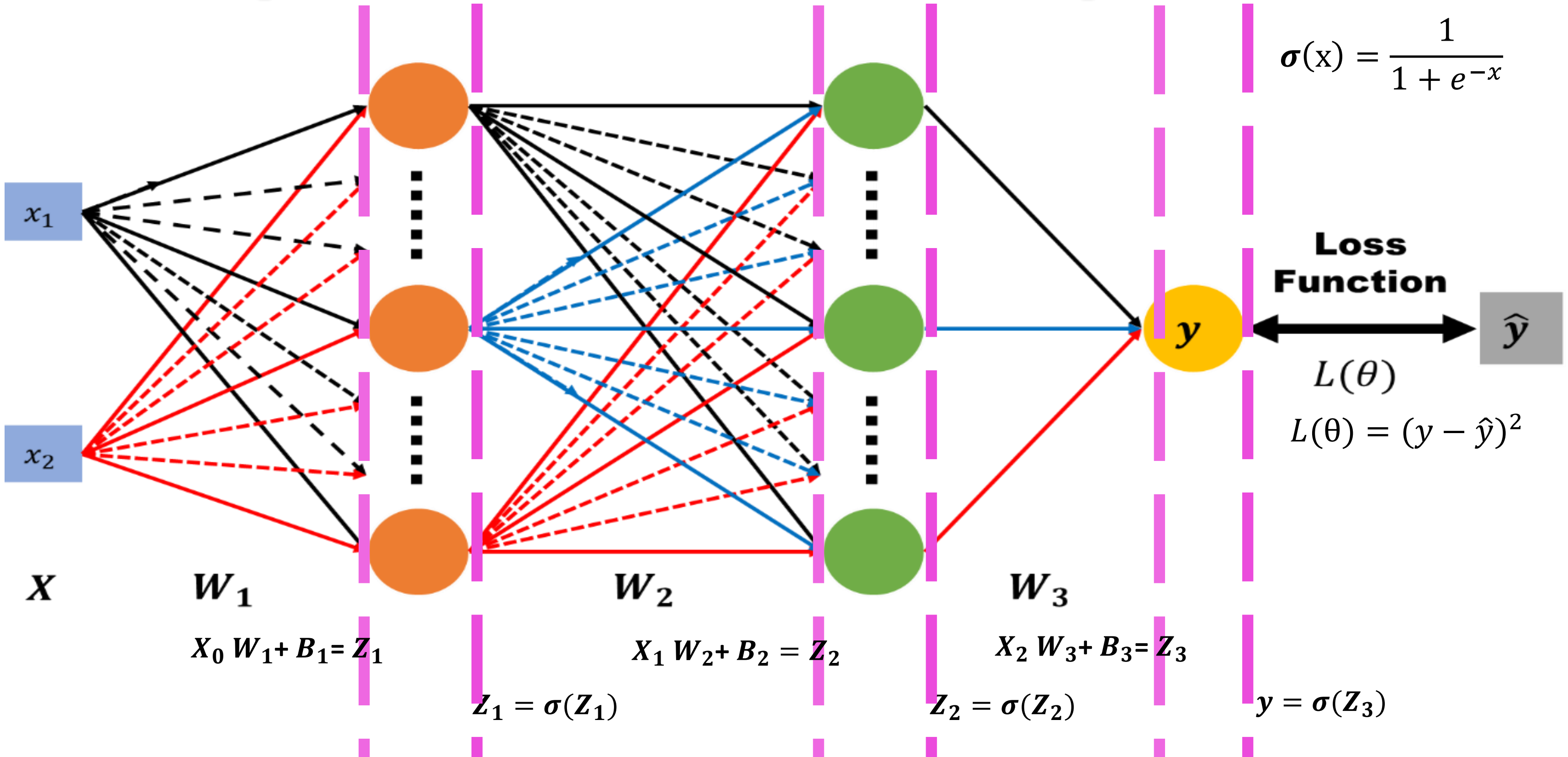


W1,W2,W3的初始化如右圖：

```python
self.hidden_weights = {
    'W1': np.random.normal(loc=0, scale=1, size=(input_size, hidden1_size)),
    'W2': np.random.normal(loc=0, scale=1, size=(hidden1_size, hidden2_size)),
    'W3': np.random.normal(loc=0, scale=1, size=(hidden2_size, output_size))}

self.hidden_biases = {
    'B1': np.random.normal(loc=0, scale=1, size=(hidden1_size)),
    'B2': np.random.normal(loc=0, scale=1, size=(hidden2_size)),
    'B3': np.random.normal(loc=0, scale=1, size=(output_size))}
```

Experimental Setup – Forward propagation



後面實驗中，我將選擇不同的(h1,h2)神經元個數組合來進行測試：

(h1,h2)= (64,64)
(128,128)
(256,256)
(512,512)
(256,512)
(512,256)

Forward propagation 的設置如右圖

```
def forward(self, inputs):
    """ Implementation of the forward pass."""
    ##### 1st hidden layer
    Z1 = np.dot(inputs, self.hidden_weights['W1']) + self.hidden_biases['B1']
    self.X1 = sigmoid(Z1)
    ##### 2nd hidden layer
    Z2 = np.dot(self.X1, self.hidden_weights['W2']) + self.hidden_biases['B2']
    self.X2 = sigmoid(Z2)
    ##### output layer
    Z3 = np.dot(self.X2, self.hidden_weights['W3']) + self.hidden_biases['B3']
    NNpred = sigmoid(Z3)
    return NNpred
```

用chain rule解開整個backward propagation

Input data

$$\frac{\partial Z}{\partial W} = X$$

註：$\frac{\partial Z}{\partial B} = 1$

- 一開始為解gradient decent： $W^{t+1} = W^{t} - \eta \frac{\partial L}{\partial W^{t}}$
- 為了得到W^{t+1}， $\frac{\partial L}{\partial W}$是必要解的方程式，做chain rule變成： $\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z} \boxed{\frac{\partial Z}{\partial W}}$
- 其中， $\frac{\partial L}{\partial Z}$ 為backward pass，可做chain rule變成： $\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial X} \boxed{\frac{\partial X}{\partial Z}}$
- 再接著下去解 $\frac{\partial L}{\partial X}$，再做chain rule變成： $\frac{\partial L}{\partial X} = \sum_{m=1}^{D_{l+1}=2} \frac{\partial L}{\partial Z_m^t} \boxed{\frac{\partial Z_m^t}{\partial X^t}}$
- 最後，$\because L = (y - \hat{y})^2$
- 最後一次chain rule變成： $\frac{\partial L}{\partial Z^{t+1}} = \frac{\partial L}{\partial y} \frac{\partial y}{\partial Z^{t+1}} = 2(y - \hat{y}) \times \boldsymbol{\sigma}'(Z^{t+1})$

$$\frac{\partial X}{\partial Z} = \boldsymbol{\sigma}'(Z)$$

$$\frac{\partial Z_m^t}{\partial X_n^t} = W_{nm}^t$$

註：

W^t的t即代表iteration數

D_l即為第 l 隱藏層neuron個數

如此即可把Backward Propagation解出來！

Known

$$\frac{\partial L}{\partial Z^t} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial Z^t} = 2(y-\hat{y})\times \boldsymbol{\sigma}'(Z^t)$$

Known

$$\frac{\partial Z_m^t}{\partial X_n^t} = W_{nm}^t$$

$$\frac{\partial L}{\partial X} = \sum_{m=1}^{D_{l+1}=2} \frac{\partial L}{\partial Z_m^{t+1}}\frac{\partial Z_m^t}{\partial X^t}$$

Known

$$\frac{\partial X}{\partial Z} = \boldsymbol{\sigma}'(Z)$$

$$\frac{\partial L}{\partial Z} = \frac{\partial L}{\partial X}\frac{\partial X}{\partial Z}$$

Known (input data)

$$\frac{\partial Z}{\partial W} = X$$

$$\frac{\partial L}{\partial W} = \frac{\partial L}{\partial Z}\frac{\partial Z}{\partial W}$$

$$W^{t+1} = W^t - \eta\frac{\partial L}{\partial W^t}$$

DONE!

註：

W^t的t即代表iteration數。

D_l即為第 l 隱藏層neuron個數。

B的update以此類推。

Experimental Setup – Back propagation

```python
def backward(self, inputs, grad_Loss, NNpred):
    """ Implementation of the backward pass. """
    ############ Calculate Gradients ############
    ###### output layer
    grad_Z3 = grad_Loss*grad_sigmoid(NNpred)
    grad_W3 = grad_Z3*(self.X2).T
    grad_B3 = grad_Z3.squeeze()

    ###### 2nd hidden layer
    grad_Z2 = np.dot(grad_Z3, self.hidden_weights['W3'].T)*grad_sigmoid(self.X2)
    grad_W2 = np.dot(self.X1.T, grad_Z2)
    grad_B2 = grad_Z2.squeeze()

    ###### 1st hidden layer
    grad_Z1 = np.dot(grad_Z2, self.hidden_weights['W2'].T)*grad_sigmoid(self.X1)
    grad_W1 = np.dot(inputs.T, grad_Z1)
    grad_B1 = grad_Z1.squeeze()

    ############ Update Model Parameters ############
    self.hidden_weights['W1'] -= self.learn_rate*grad_W1
    self.hidden_weights['W2'] -= self.learn_rate*grad_W2
    self.hidden_weights['W3'] -= self.learn_rate*grad_W3
    self.hidden_biases['B1'] -= self.learn_rate*grad_B1
    self.hidden_biases['B2'] -= self.learn_rate*grad_B2
    self.hidden_biases['B3'] -= self.learn_rate*grad_B3
```

$$\mathrm{grad\_Z3} = \frac{\partial L}{\partial y}\frac{\partial y}{\partial Z} = 2(y-\hat{y})\times \boldsymbol{\sigma}'(Z^{t+1})$$

$$grad\_W3 = \frac{\partial L}{\partial W3} = X2 \times grad\_Z3$$

$$grad\_B3 = \frac{\partial L}{\partial B3}$$

$$\mathrm{grad\_Z2} = \frac{\partial L}{\partial Z3}\boldsymbol{\sigma}'(Z2) = \frac{\partial L}{\partial Z3}\times(X2)(1-X2)$$

$$grad\_W2 = \frac{\partial L}{\partial W2} = X1\times\frac{\partial L}{\partial Z3}\times\boldsymbol{\sigma}'(Z2)$$

$$grad\_B2 = \frac{\partial L}{\partial B2}$$

$$\mathrm{grad\_Z1} = \frac{\partial L}{\partial Z2}\boldsymbol{\sigma}'(Z1) = \frac{\partial L}{\partial Z2}\times(X1)(1-X1)$$

$$grad\_W1 = \frac{\partial L}{\partial W1} = \mathrm{input}\times\frac{\partial L}{\partial Z2}\times\boldsymbol{\sigma}'(Z1)$$

$$grad\_B1 = \frac{\partial L}{\partial B1}$$

$$W^{t+1} = W^{t} - \eta\frac{\partial L}{\partial W^{t}}$$

$$B^{t+1} = B^{t} - \eta\frac{\partial L}{\partial B^{t}}$$

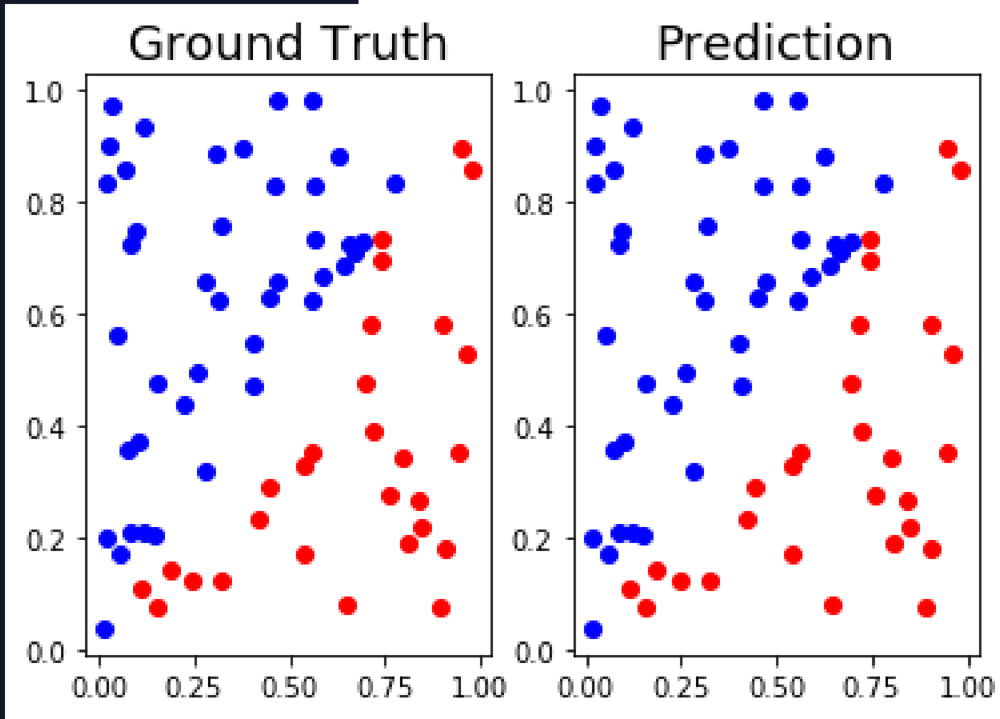
Experimental Result

part1. basic results

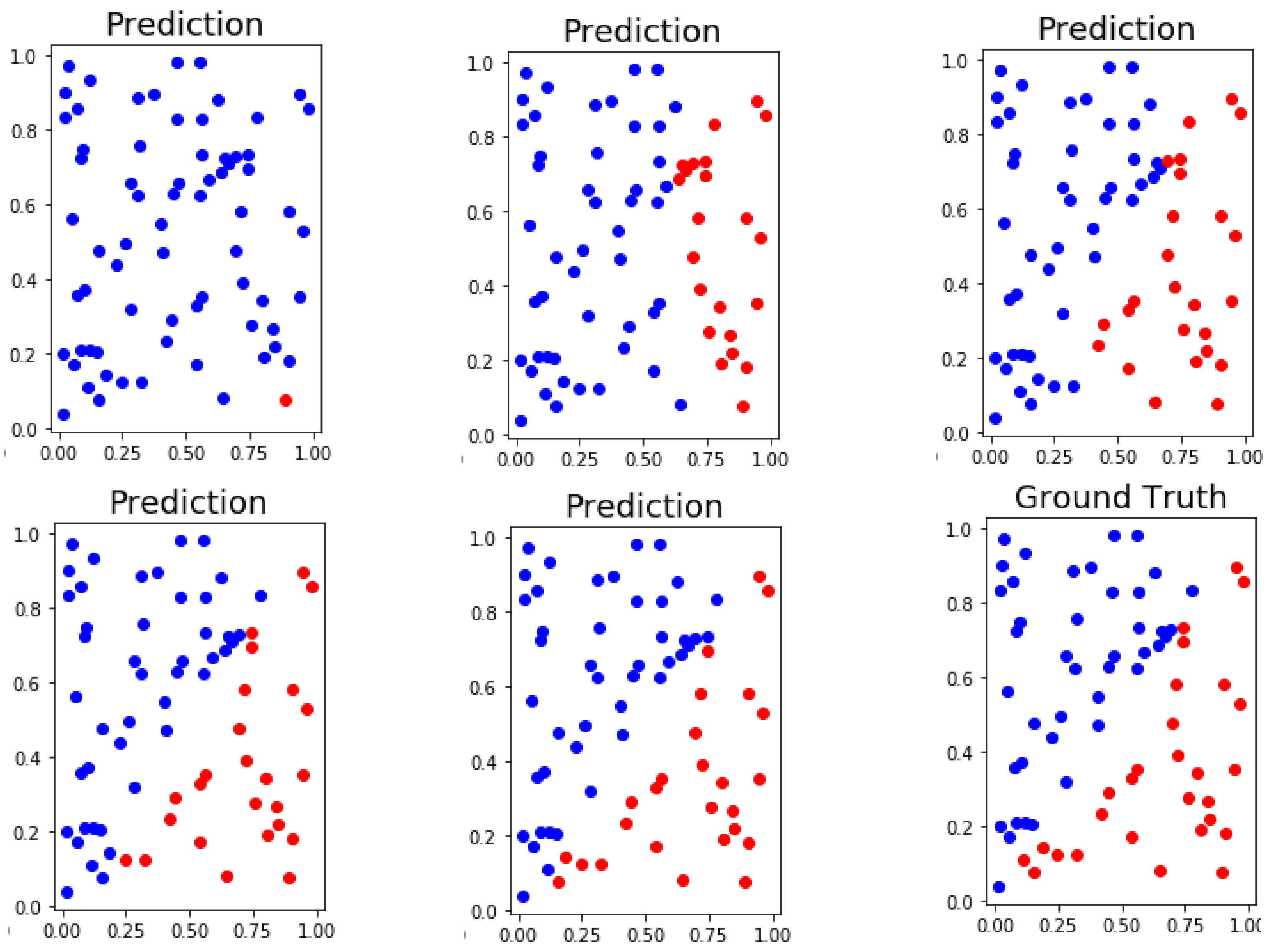
Experimental Result – Linear data

```
Epoch    0, loss: 15.3336, accuracy: 53.34%
Epoch  250, loss: 3.2052, accuracy: 84.74%
Epoch  500, loss: 2.3631, accuracy: 88.26%
Epoch  750, loss: 1.9154, accuracy: 90.20%
Epoch 1000, loss: 1.6212, accuracy: 91.50%
Epoch 1250, loss: 1.4103, accuracy: 92.46%
Epoch 1500, loss: 1.2513, accuracy: 93.19%
Epoch 1750, loss: 1.1284, accuracy: 93.76%
Epoch 2000, loss: 1.0299, accuracy: 94.22%
Epoch 2250, loss: 0.9478, accuracy: 94.59%
Epoch 2500, loss: 0.8777, accuracy: 94.92%
Epoch 2750, loss: 0.8169, accuracy: 95.20%
Epoch 3000, loss: 0.7636, accuracy: 95.45%
Epoch 3250, loss: 0.7163, accuracy: 95.67%
Epoch 3500, loss: 0.6740, accuracy: 95.87%
Epoch 3750, loss: 0.6360, accuracy: 96.05%
Epoch 4000, loss: 0.6016, accuracy: 96.21%
Epoch 4250, loss: 0.5702, accuracy: 96.36%
Epoch 4500, loss: 0.5414, accuracy: 96.50%
Epoch 4750, loss: 0.5150, accuracy: 96.62%
***** Training finished *****
Epoch 5000, loss: 0.4907, accuracy: 96.74%
Model Training time taken: 0.0 minutes 20.0 seconds
```

```
SimpleNet Predicted Probabilities:
[[9.99999909e-01]
 [2.90238228e-03]
 [9.99997420e-01]
 [9.99999418e-01]
 [9.54098604e-01]
 [9.99999936e-01]
 [9.99998832e-01]
 [9.99999182e-01]
 [9.99992950e-01]
 [9.71861101e-01]
 [9.95699861e-01]
 [9.99716492e-01]
 [3.33892433e-08]
 [9.78007645e-05]
 [9.99999829e-01]
 [9.99997163e-01]
 [9.99331448e-01]
 [1.88177925e-03]
 [1.21050264e-07]
 [2.32139591e-04]
 [9.78765304e-01]
 [2.61307546e-06]
 [8.39284473e-01]
 [8.92354463e-01]
```



Epoch=2, acc=55.39% Epoch=10, acc=57.43% Epoch=20, acc=69.38%



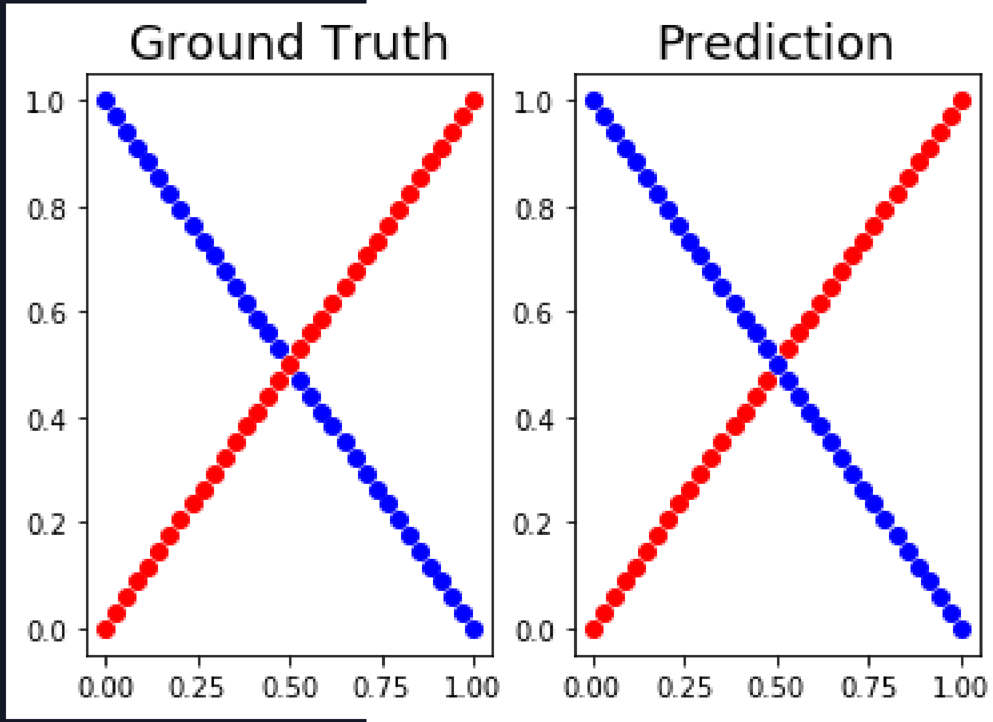
Epoch=30, acc=72.72% Epoch=180, acc=86.32% Ground Truth

本頁show出Linear data points的訓練過程，可以看出隨著訓練的epoch增加，此兩層類神經網絡的分類器愈來愈準確。

Experimental Result – XOR datas

```
Epoch    0, loss: 29.2697, accuracy: 50.45%
Epoch  250, loss: 10.4599, accuracy: 62.36%
Epoch  500, loss: 7.2477, accuracy: 70.99%
Epoch  750, loss: 5.7383, accuracy: 76.14%
Epoch 1000, loss: 4.8718, accuracy: 79.43%
Epoch 1250, loss: 4.2925, accuracy: 81.74%
Epoch 1500, loss: 3.8694, accuracy: 83.47%
Epoch 1750, loss: 3.5424, accuracy: 84.83%
Epoch 2000, loss: 3.2795, accuracy: 85.93%
Epoch 2250, loss: 3.0618, accuracy: 86.84%
Epoch 2500, loss: 2.8775, accuracy: 87.62%
Epoch 2750, loss: 2.7188, accuracy: 88.28%
Epoch 3000, loss: 2.5801, accuracy: 88.87%
Epoch 3250, loss: 2.4575, accuracy: 89.38%
Epoch 3500, loss: 2.3480, accuracy: 89.84%
Epoch 3750, loss: 2.2495, accuracy: 90.26%
Epoch 4000, loss: 2.1601, accuracy: 90.63%
Epoch 4250, loss: 2.0785, accuracy: 90.97%
Epoch 4500, loss: 2.0036, accuracy: 91.28%
Epoch 4750, loss: 1.9345, accuracy: 91.57%
***** Training finished *****
Epoch 5000, loss: 1.8708, accuracy: 91.84%
Model Training time taken: 0.0 minutes 19.5 seconds
```

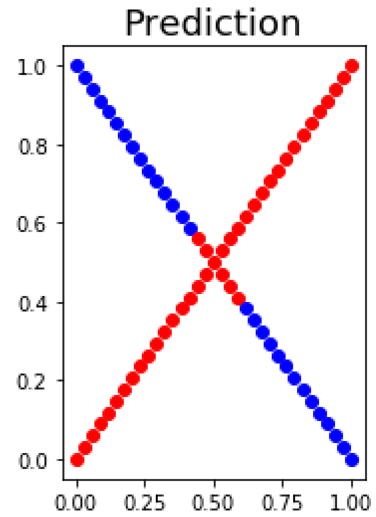
```
SimpleNet Predicted Probabilities:
[[1.57695465e-03]
 [9.98310029e-01]
 [1.58227846e-03]
 [9.98138901e-01]
 [1.60990699e-03]
 [9.97894987e-01]
 [1.66568335e-03]
 [9.97549391e-01]
 [1.75957893e-03]
 [9.97056244e-01]
 [1.90925254e-03]
 [9.96340569e-01]
 [2.14746913e-03]
 [9.95274992e-01]
 [2.53865425e-03]
 [9.93632420e-01]
 [3.21890190e-03]
 [9.90983755e-01]
 [4.50163191e-03]
 [9.86464094e-01]
 [7.18241856e-03]
 [9.78219015e-01]
 [1.34778768e-02]
 [9.62119242e-01]
```



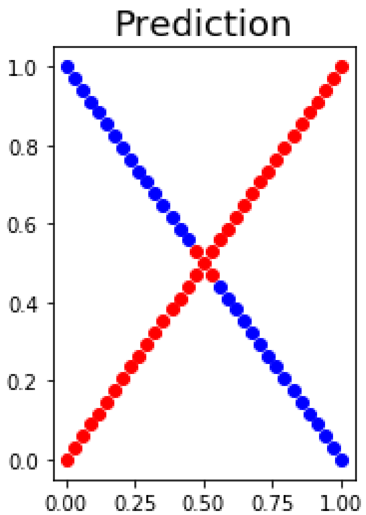
Experimental Result

part2. different # of hidden units

```
Epoch    0, loss: 17.9294, accuracy: 52.10%
Epoch  500, loss: 8.7622, accuracy: 67.29%
Epoch 1000, loss: 6.2196, accuracy: 74.73%
Epoch 1500, loss: 4.9643, accuracy: 78.98%
***** Training finished *****
Epoch 2000, loss: 4.2003, accuracy: 81.81%
Model Training time taken: 0.0 minutes 7.1 seconds
```



(64,64)



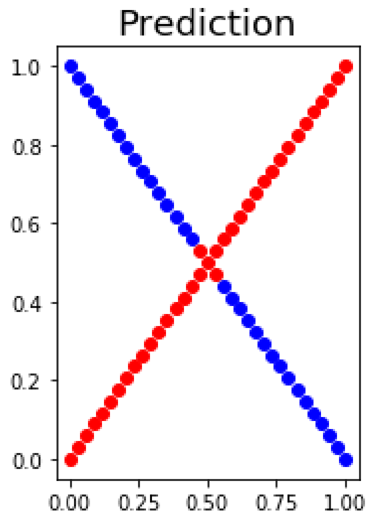
(128,128)

```
Epoch    0, loss: 34.9410, accuracy: 49.29%
Epoch  500, loss: 4.9069, accuracy: 79.03%
Epoch 1000, loss: 3.8776, accuracy: 83.14%
Epoch 1500, loss: 3.3353, accuracy: 85.40%
***** Training finished *****
Epoch 2000, loss: 2.9671, accuracy: 86.96%
Model Training time taken: 0.0 minutes 31.4 seconds
```

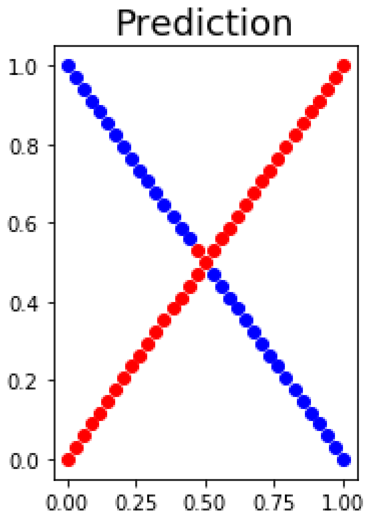
本頁show出XOR data points在不同的神經元個數的分類器，經訓練2000個epoch後的分類效能比較。

明顯(512,512)的效能最佳，只是雖然一樣是2000個epoch，其訓練時間多出許多倍。

```
Epoch    0, loss: 33.9902, accuracy: 50.72%
Epoch  500, loss: 3.2241, accuracy: 85.91%
Epoch 1000, loss: 2.4795, accuracy: 89.17%
Epoch 1500, loss: 2.0830, accuracy: 90.89%
***** Training finished *****
Epoch 2000, loss: 1.8126, accuracy: 92.03%
Model Training time taken: 2.0 minutes 0.8 seconds
```



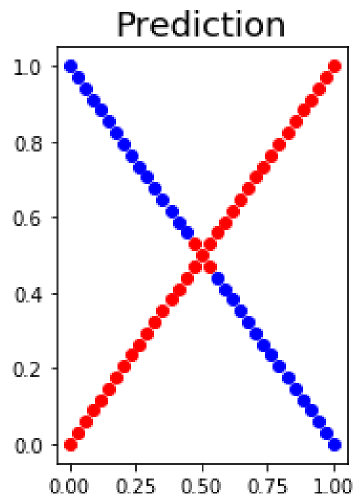
(256,256)



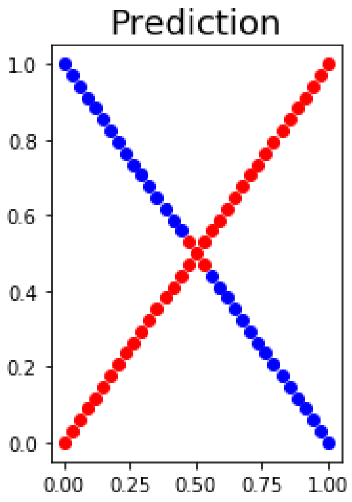
(512,512)

```
Epoch    0, loss: 33.1493, accuracy: 51.18%
Epoch  500, loss: 2.1529, accuracy: 90.57%
Epoch 1000, loss: 1.8284, accuracy: 91.99%
Epoch 1500, loss: 1.6574, accuracy: 92.76%
***** Training finished *****
Epoch 2000, loss: 1.4980, accuracy: 93.42%
Model Training time taken: 6.0 minutes 32.0 seconds
```

```
Epoch    0, loss: 22.1057, accuracy: 52.37%
Epoch  500, loss: 2.7005, accuracy: 88.29%
Epoch 1000, loss: 2.3282, accuracy: 89.96%
Epoch 1500, loss: 2.1118, accuracy: 90.91%
***** Training finished *****
Epoch 2000, loss: 1.9515, accuracy: 91.59%
Model Training time taken: 3.0 minutes 39.2 seconds
```



(256,512)



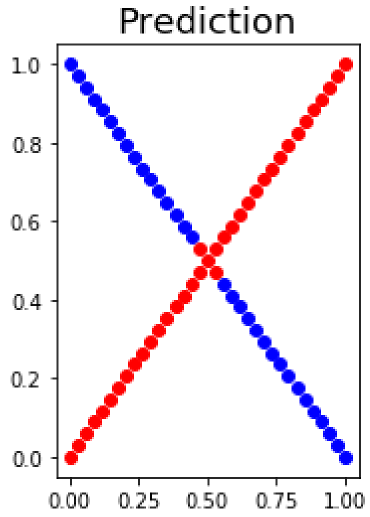
(512,256)

```
Epoch    0, loss: 34.9710, accuracy: 49.26%
Epoch  500, loss: 2.5397, accuracy: 88.36%
Epoch 1000, loss: 2.1156, accuracy: 90.41%
Epoch 1500, loss: 1.8601, accuracy: 91.62%
***** Training finished *****
Epoch 2000, loss: 1.6816, accuracy: 92.41%
Model Training time taken: 3.0 minutes 36.2 seconds
```

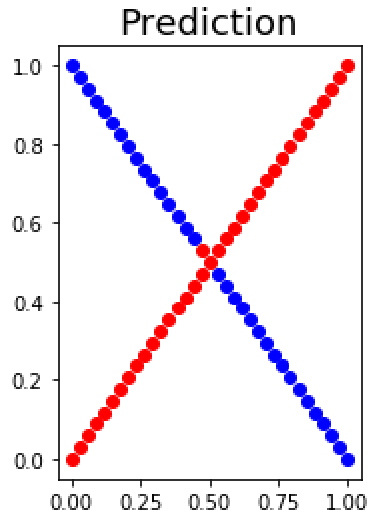
本頁show出XOR data points在兩層神經元個數不一樣的分類器，經訓練2000個epoch後的分類效能比較。

其實四個效能都差不多，唯從(256,256)和(256,512)來看，神經元個數增加，效能就沒有變好了。

```
Epoch    0, loss: 33.9902, accuracy: 50.72%
Epoch  500, loss: 3.2241, accuracy: 85.91%
Epoch 1000, loss: 2.4795, accuracy: 89.17%
Epoch 1500, loss: 2.0830, accuracy: 90.89%
***** Training finished *****
Epoch 2000, loss: 1.8126, accuracy: 92.03%
Model Training time taken: 2.0 minutes 0.8 seconds
```



(256,256)



(512,512)

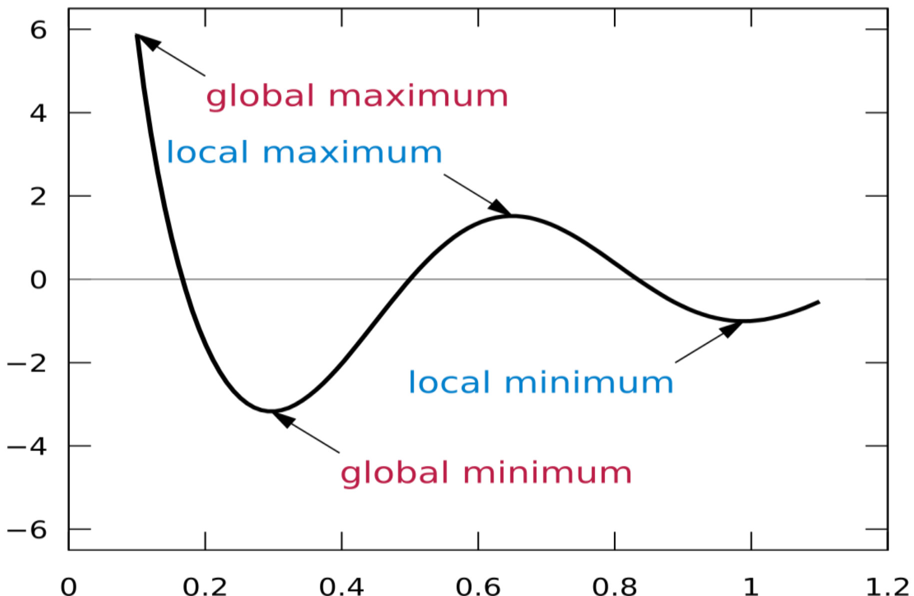
```
Epoch    0, loss: 33.1493, accuracy: 51.18%
Epoch  500, loss: 2.1529, accuracy: 90.57%
Epoch 1000, loss: 1.8284, accuracy: 91.99%
Epoch 1500, loss: 1.6574, accuracy: 92.76%
***** Training finished *****
Epoch 2000, loss: 1.4980, accuracy: 93.42%
Model Training time taken: 6.0 minutes 32.0 seconds
```

疑似陷入local minima或gradient太小

```
Epoch 35800, loss: 34.00, accuracy: 50.72%
Epoch 36000, loss: 34.00, accuracy: 50.72%
Epoch 36200, loss: 34.00, accuracy: 50.72%
Epoch 36400, loss: 34.00, accuracy: 50.72%
Epoch 36600, loss: 34.00, accuracy: 50.72%
Epoch 36800, loss: 34.00, accuracy: 50.72%
Epoch 37000, loss: 34.00, accuracy: 50.72%
Epoch 37200, loss: 34.00, accuracy: 50.72%
Epoch 37400, loss: 34.00, accuracy: 50.72%
Epoch 37600, loss: 34.00, accuracy: 50.72%
Epoch 37800, loss: 34.00, accuracy: 50.72%
Epoch 38000, loss: 34.00, accuracy: 50.72%
Epoch 38200, loss: 34.00, accuracy: 50.72%
Epoch 38400, loss: 34.00, accuracy: 50.72%
Epoch 38600, loss: 34.00, accuracy: 50.72%
Epoch 38800, loss: 34.00, accuracy: 50.72%
Epoch 39000, loss: 34.00, accuracy: 50.72%
Epoch 39200, loss: 34.00, accuracy: 50.72%
Epoch 39400, loss: 34.00, accuracy: 50.72%
Epoch 39600, loss: 34.00, accuracy: 50.72%
Epoch 39800, loss: 34.00, accuracy: 50.72%
Epoch 40000, loss: 34.00, accuracy: 50.72%
```

在實驗過程中，有時候會發現參數都沒調整，但是訓練了上萬個epoch後loss都沒有下降的趨勢。推測也有可能從一開始就陷在local minima。(此處lr=0.0005，epoch終點為40000)

不過也有另一種可能是loss曲面太平坦，導致gradient極小，讓model update速度極慢。也有可能要再訓練個幾萬個epoch才會改善說不定。



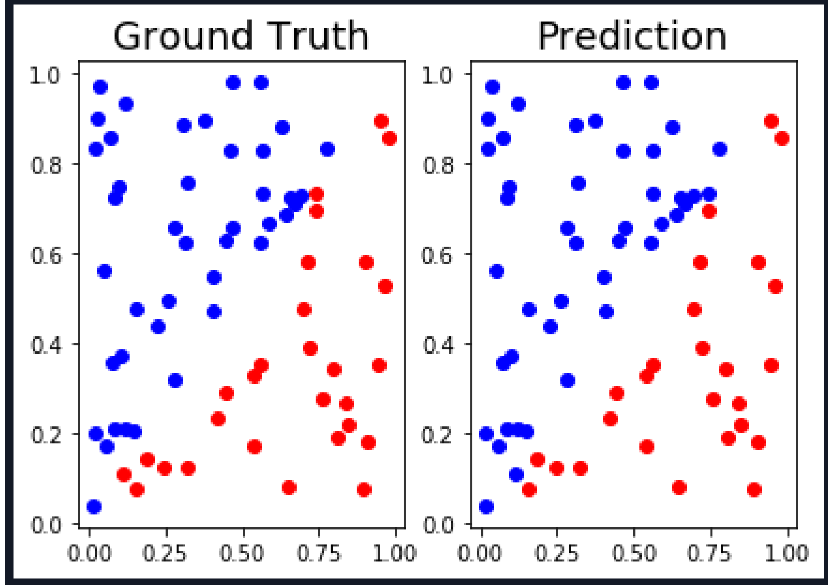
Experimental Result

part3. Observation for test data and decision boundary

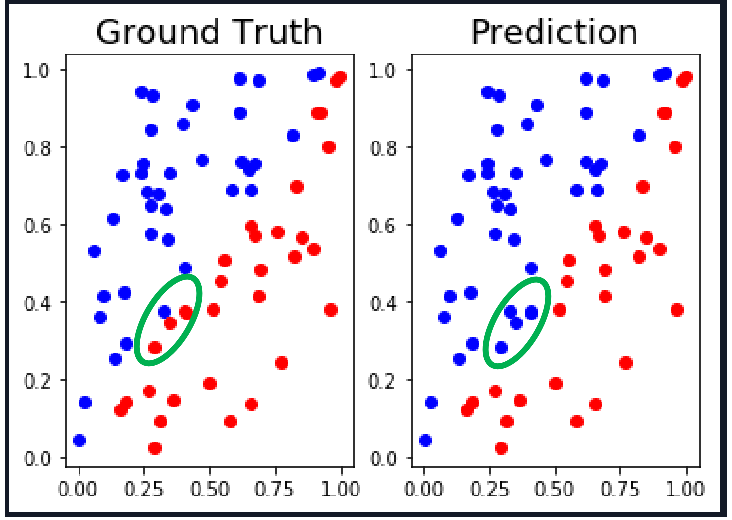
Experiment Results - test data 1

以下為Linear data (70 training data points)訓練2000個epoch後的分類器(64,64)之結果示例：

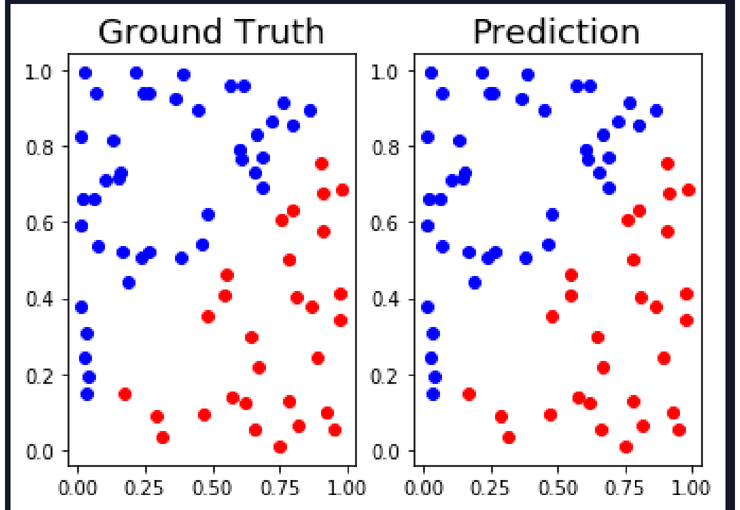
```
***** Training finished *****
Epoch 2000, loss: 2.86, accuracy: 86.29%
Model Training time taken: 0.0 minutes 8.4 seconds
```




```
***** Predicted results of Test data *****
Epoch 2000, loss: 4.46, accuracy: 81.98%
```




```
***** Predicted results of Test data *****
Epoch 2000, loss: 1.65, accuracy: 90.73%
```

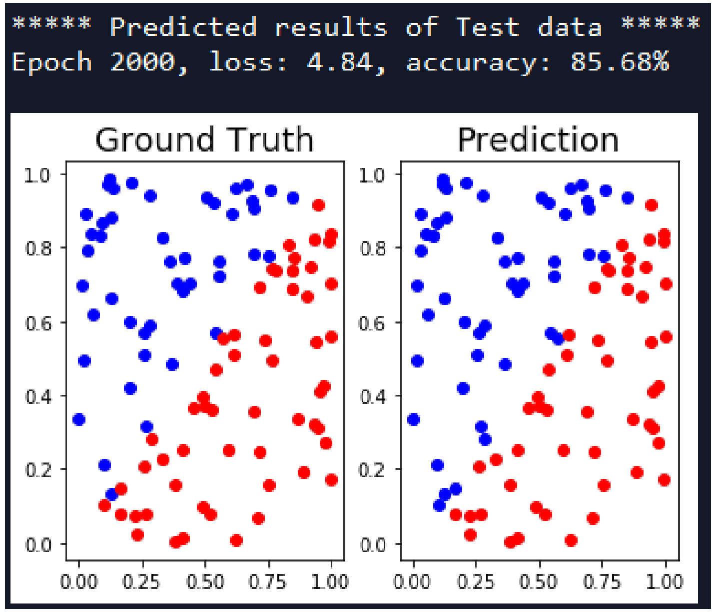


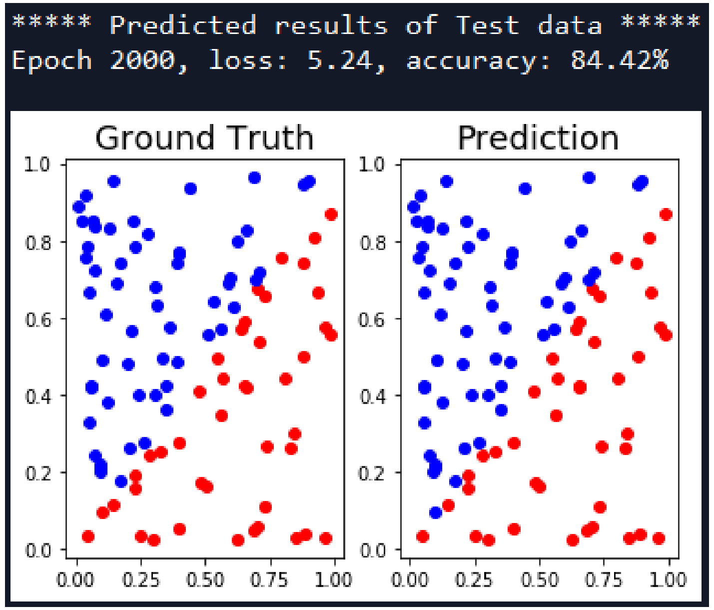
左邊上下二圖均以70 不同於training的test data points來進行trained model的inference觀察。

可以發現左上圖的acc =81.98% 較training的86.29%低，原因是如綠色圈處有三個points被分類器誤判。

左下圖的acc=90.73%反而比train_acc來的高，推測原因是這次generate出來的point分佈離中間的對角線較遠。

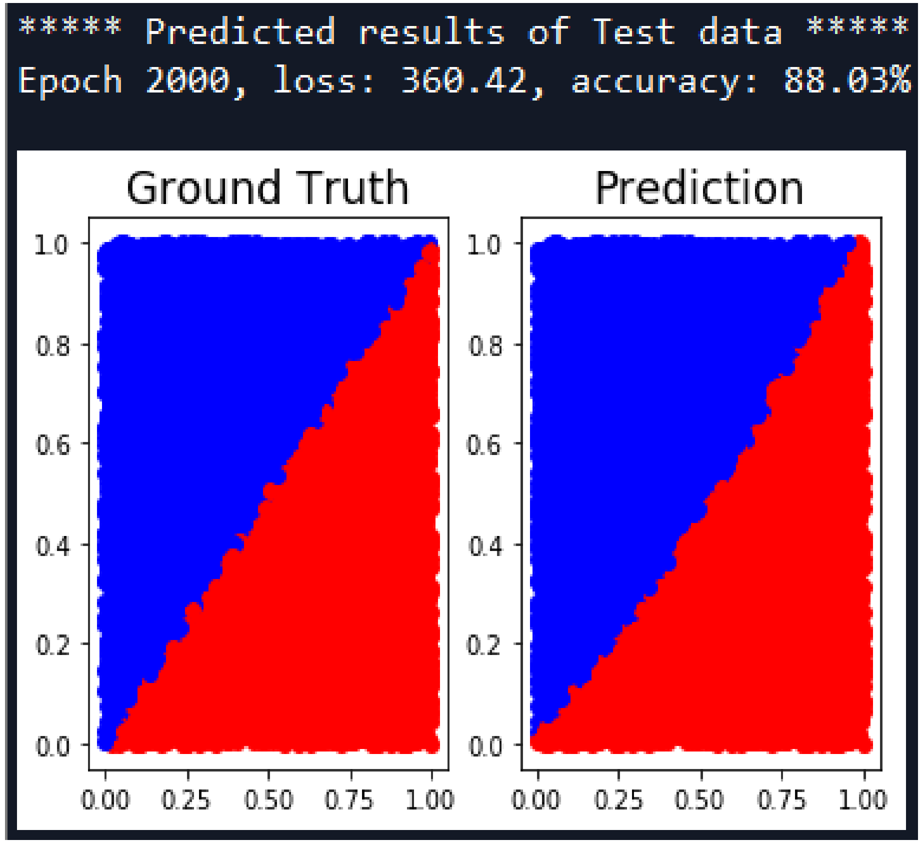
Experiment Results- test data 2





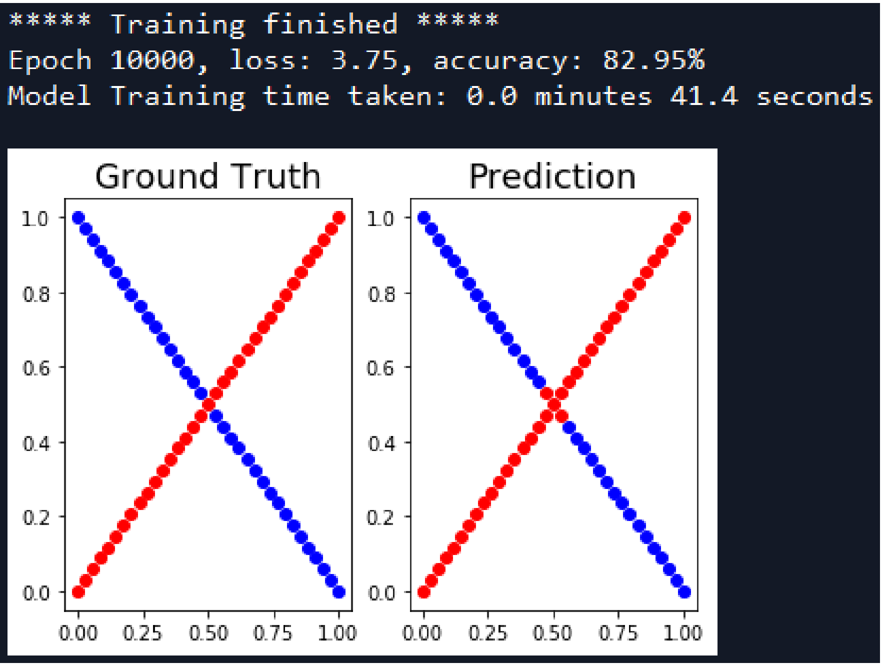
左邊上下二圖均以上一張slide的trained分類器，來進行新的100 test points的分類預測，一樣會發現在靠近中間對角線的附近較會有分類錯誤的情形發生

下圖索性用10000個test points來確認分類邊界的樣子，結果發現是接近對角線沒錯。



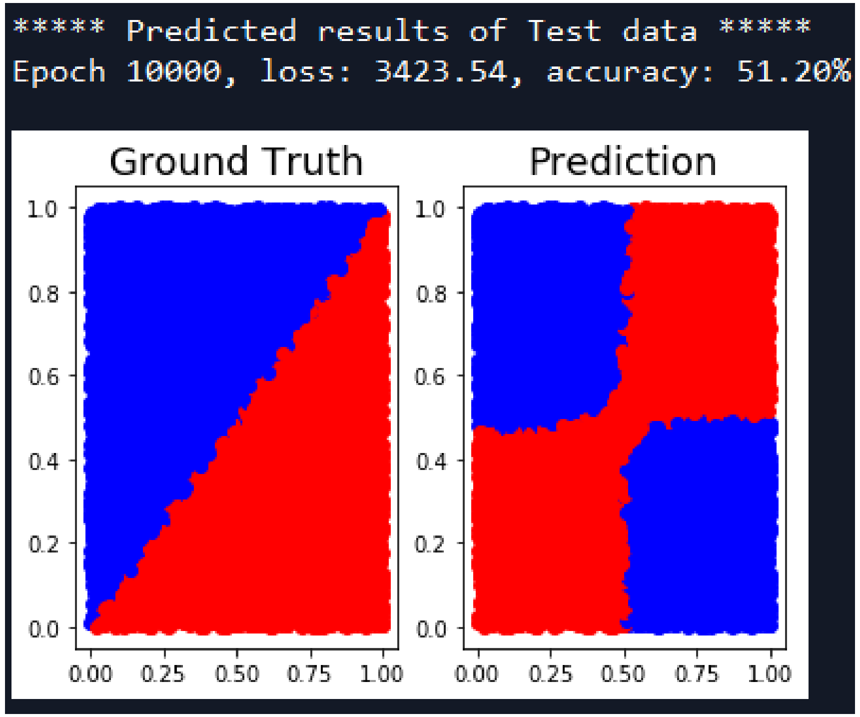
Experiment Results- test data 3

以下為XOR data (70 training data points)訓練10000個epoch後的分類器(64,64)之結果示例：

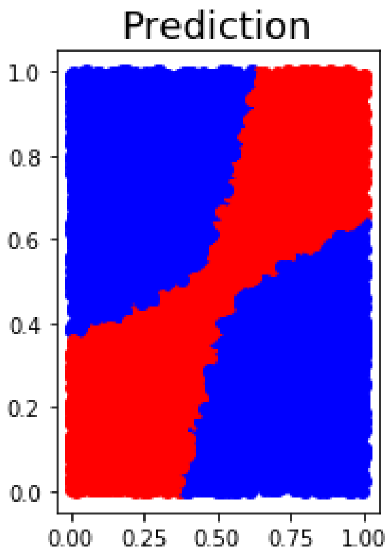


由於XOR data都是在兩條對角線上分佈，所以並不知道trained-XOR分類器的分類邊界大概如何？

因此嚐試將10000個Linear data points餵進此XOR分類器來觀察，發現其分類邊界大致如下圖所示：



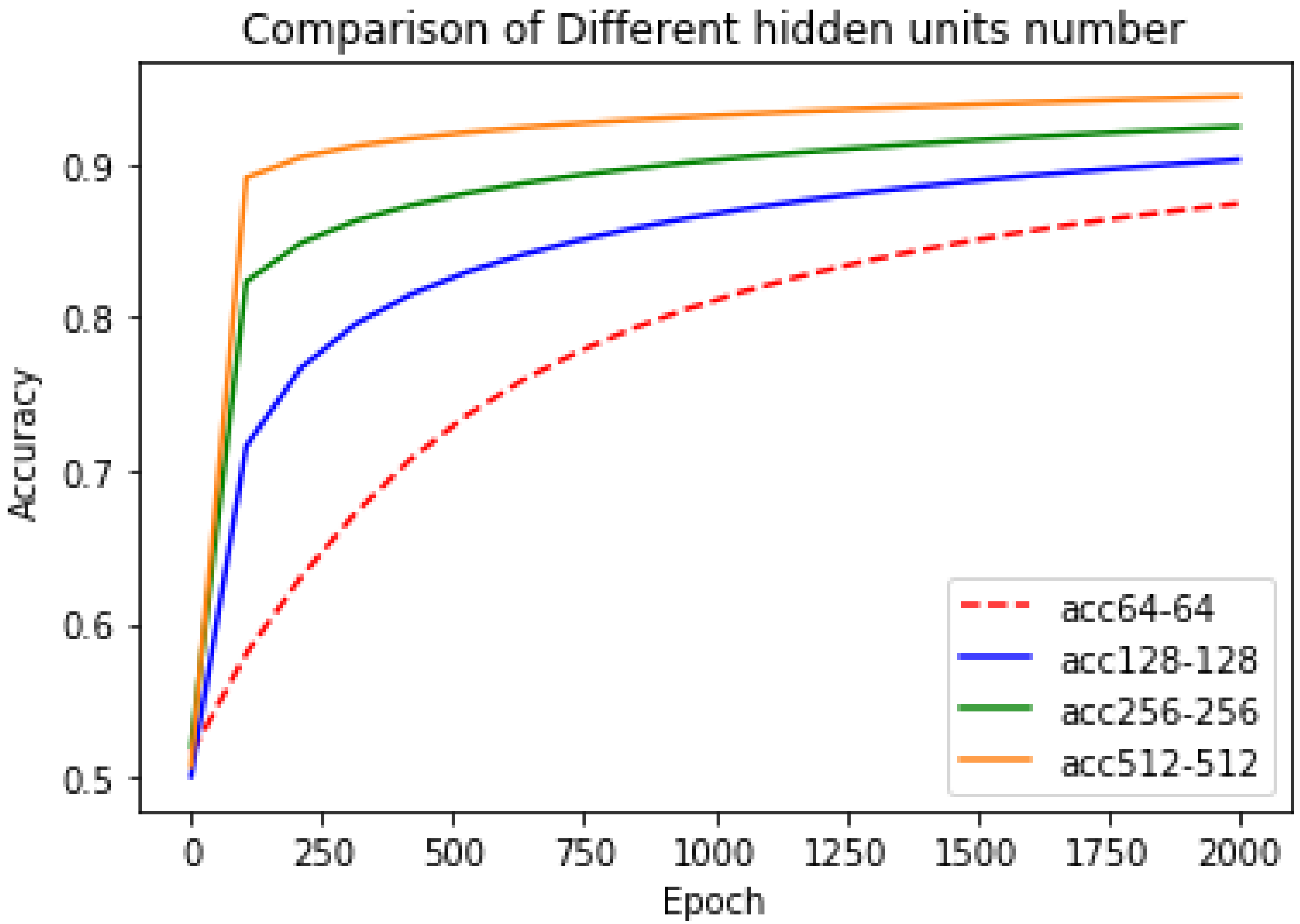
(64,64)



(256,256)

Discussion and Extra experiments

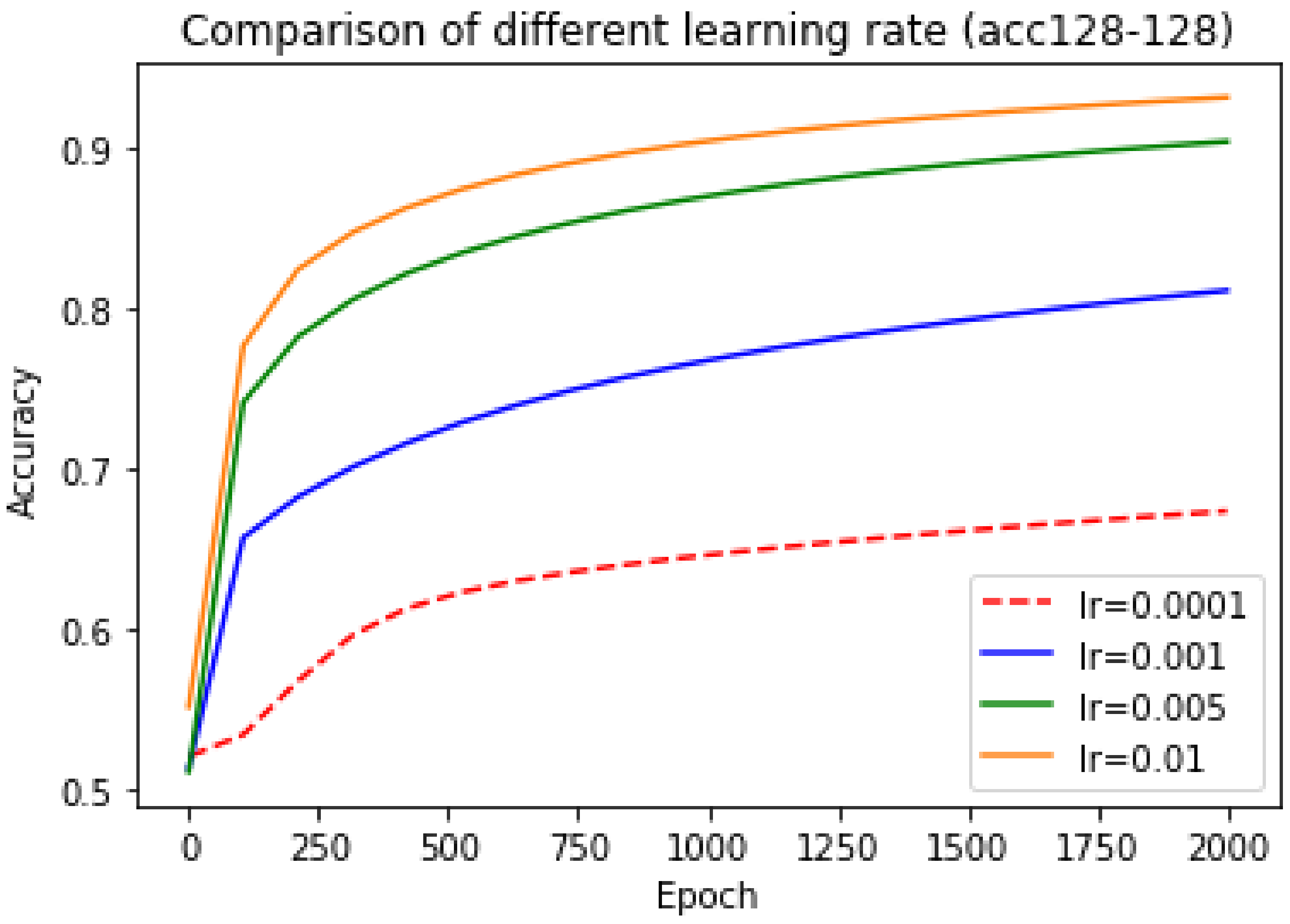
Discussion and extra experiments-1



左圖以XOR data points在不同的神經元個數的分類器，經訓練2000個epoch的訓練過程比較。

明顯**(512,512)**的隨epoch收斂速度最快(但實際訓練秒數是最久的)。

Discussion and extra experiments-2

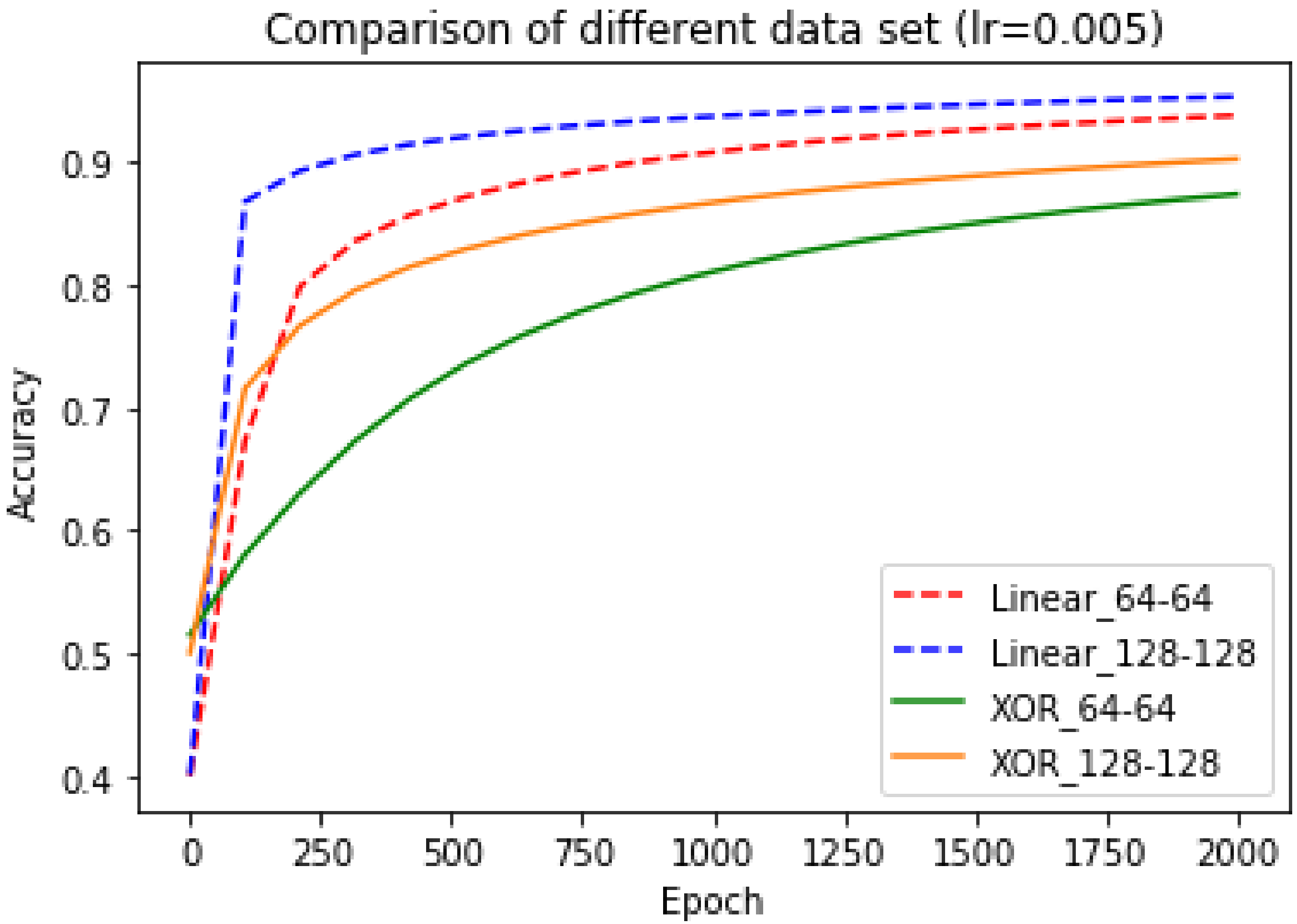


左圖以XOR data points在不同的learning rate，經訓練2000個epoch的訓練過程比較，明顯learning rate愈大，收斂速度愈快。

但是要注意，雖然lr愈大收斂速度快，但也有可能因為跨越的step大，而難以收斂到最佳最小的loss。

反之，lr愈小，收斂速度慢，卻有可能收斂到最小的loss。不過小的lr，也可能容易陷在loss的local minima。

Discussion and extra experiments-3



左圖比較XOR和Linear兩種data points在不同的神經元個數的分類器，經訓練2000個epoch的訓練過程比較。

明顯Linear的收斂速度較快，這也很好解釋，因為以data二維平面分佈來說，要訓練出將linear data成功分成兩邊的分類器是相對容易的。反觀XOR因為中間的交叉點有重疊，較難成功完全分類，而這是造成訓練loss平均較linear分佈的data來的高的原因。

The End