1. 硬體規格

- 筆電
  - CPU : Intel(R) Core(TM) i7-10510U CPU @ 1.60GHz (8核)
  - 顯卡 : NVIDIA GeForce MX250
  - RAM : 12GB
- 桌電
  - CPU : Intel(R) Core(TM) i7-4790 CPU @ 3.60GHz (8核)
  - 顯卡：NVDIA GeForce GTX 1060 3GB
  - RAM : 16GB

桌電大概七年前買的，顯卡後來有更新過，筆電是兩年前買的。作業系統皆為 Windows10 ，沒有其他規格的電腦可以選了，如果筆電顯卡跑不動實驗，會讓桌機去跑。

電話：0901228989

2.

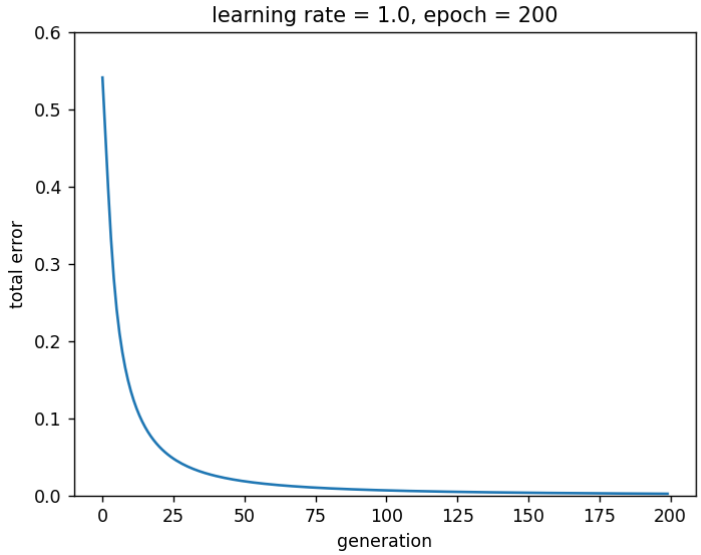
source code:

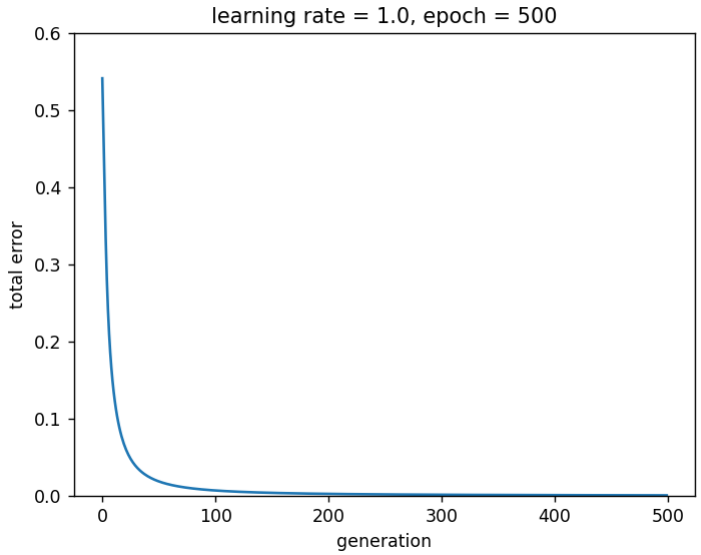
```
from numpy import *
import matplotlib.pyplot as plt
train_in = array([[0,2,0],[0,0,1],[0,1,1],[1,0,1],[1,1,1]])
train_sol = array([[0,0,0,1,1]]).T

# Initialize nn_weights
random.seed(1)
nn_weights = 2 * random.random((3, 1)) - 1

# Initialize the error array to draw the figure
error_points = []

# learning rate and epoch initialize
learning_rate = 1.0
epoch = 200

# Unknown test input
test_in = array([1,0,0])

# Train the network
for i in range(epoch):
    print("\n i= ", i, "nn_weight=")
    print(nn_weights)

    # Calculate the outputs for each training examples
    train_out = 1 / (1 + exp(-(dot(train_in, nn_weights))))
    print("train_out =")
    print(train_out)

    # calculate error
    error = 1.0 - (1/(1 + exp(-(dot(test_in, nn_weights)))))
    error_points.append(error)

```

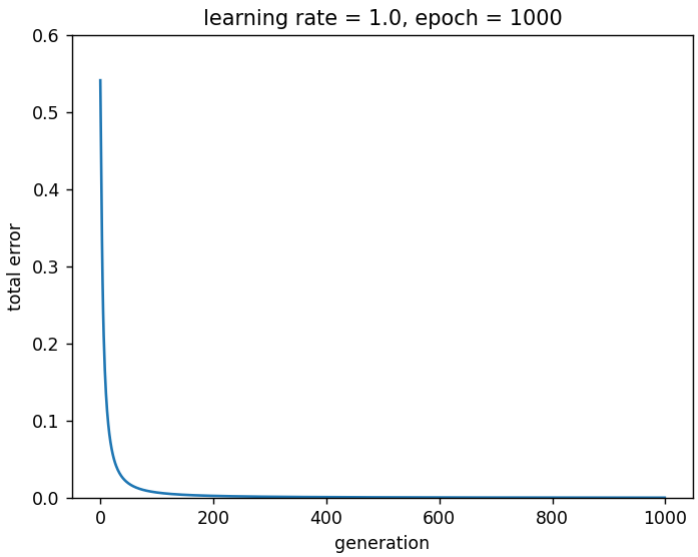
```
    # Run the NN adjustments
    nn_weights += learning_rate * dot(train_in.T, (train_sol-
train_out)*train_out*(1-train_out))

# draw figure
plt.plot(error_points)
plt.ylim(0.0, 0.6)
plt.title("learning rate = " + str(learning_rate) + ", epoch = " +
str(epoch))
plt.xlabel("generation")
plt.ylabel("total error")
plt.show()

# Print the result for our unknown test input
print('\nThe final prediction is ',1/(1 + exp(-(dot(test_in, nn_weights)))))
```

result figure:

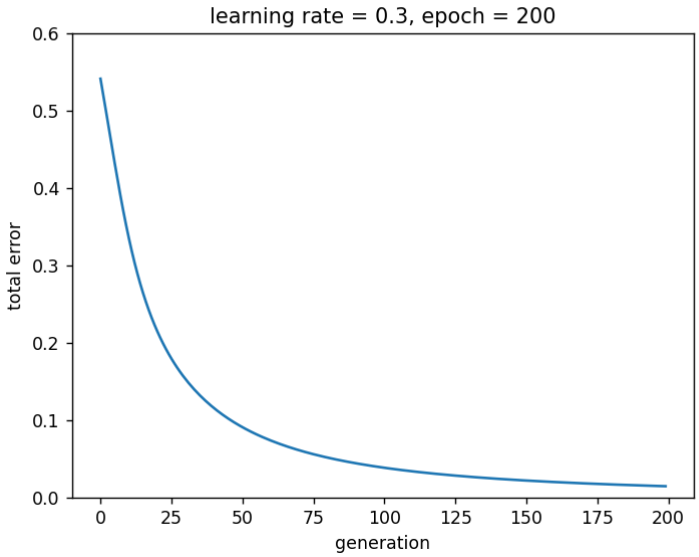
- 200 epoch vs. 500 epoch vs. 1000 epoch (learning rate = 1.0)

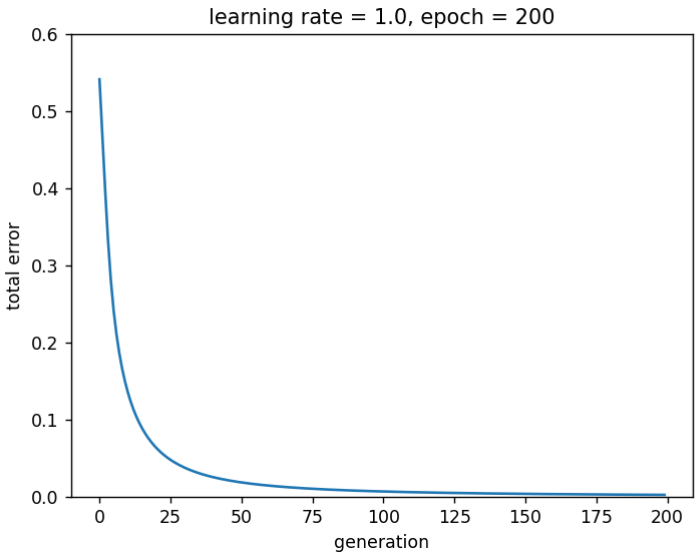


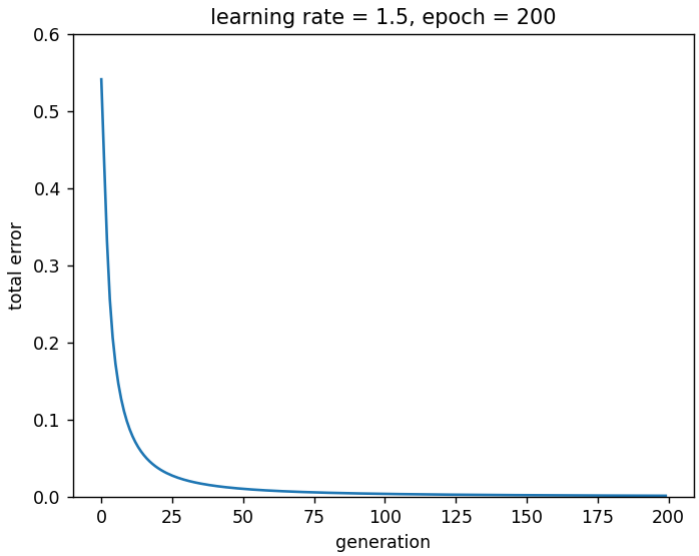




- learning rate 0.3 vs. 1.0 vs. 1.5 (200 epoch)







增加 epoch 的數量後，誤差會越來越接近 0 但不為 0 。learning rate 會影響整個學習的速度，雖然最後學習的成果差不多。

3.

L1 和 L2 正規化是為了讓訓練過程中，不要有某些特徵太過於突出，導致帶偏了整個訓練的方向，或是某個特徵權重太高，後面改不回來，也就是所謂的「Overfitting」。

$$L1: ||w||_1 = \Sigma_{j=1}^{m}|w_j|$$

L1 正規化會**將模型裡所有的參數都取絕對值**，經過運算後將過於複雜的模型簡化，也就是有辦法將沒有用的權重設為0，留下訓練模型認為重要的權重，讓整個模型稀疏。

$$L2: ||w||^2 = \Sigma_{j=1}^{m}w_j^2$$

L2 正規化會**將模型裡所有的參數都取平方求和**，在經過運算後一樣可以簡化模型。但只會進行削弱權重，使所有神經元都處於活動狀態，但所有權重值都保持在較小的狀態。

4.

7 種形狀分別為 $X_1, X_2, X_1^2, X_2^2, X_1X_2, \sin(X_1), \sin(X_2)$，藍色為正，橘色為負。在測試四種分類資料時，每個都有其運用之處。

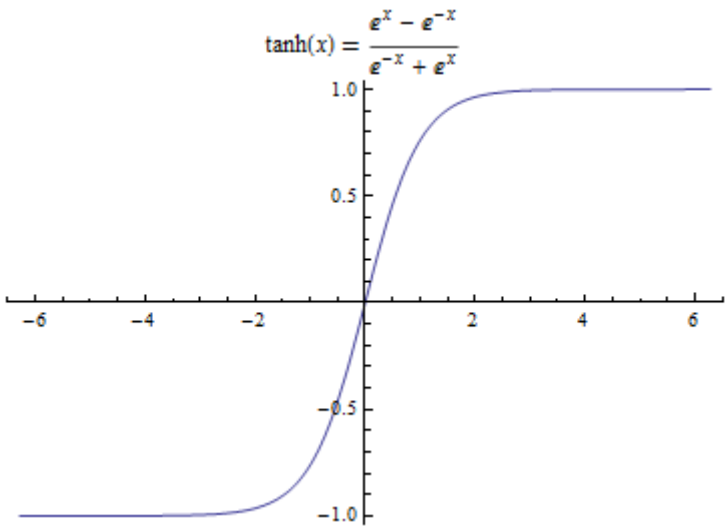
X_1^2 和 X_2^2 的組合可以在視覺化的資料中產生類似圓形的圖案；$\sin(X_1)$ 和 $\sin(X_2)$ 可以讓資料分段出現，而不是連續出現，在螺旋狀訓練時可以用到；X_1X_2 的向量甚至可以直接用來訓練第二種圖形；X_1, X_2 則可以簡單的讓訓練神經有上和下、左和右的概念。

5.

迴歸模型在機器學習裡，也就是特徵與目標的關係。透過梯度下降去找出成本函數算出來最低誤差值的最佳迴歸模型。像我們在 TensorFlow Playground 試著訓練的就是最佳的迴歸模型，找出每種形狀與目標的關係權重。

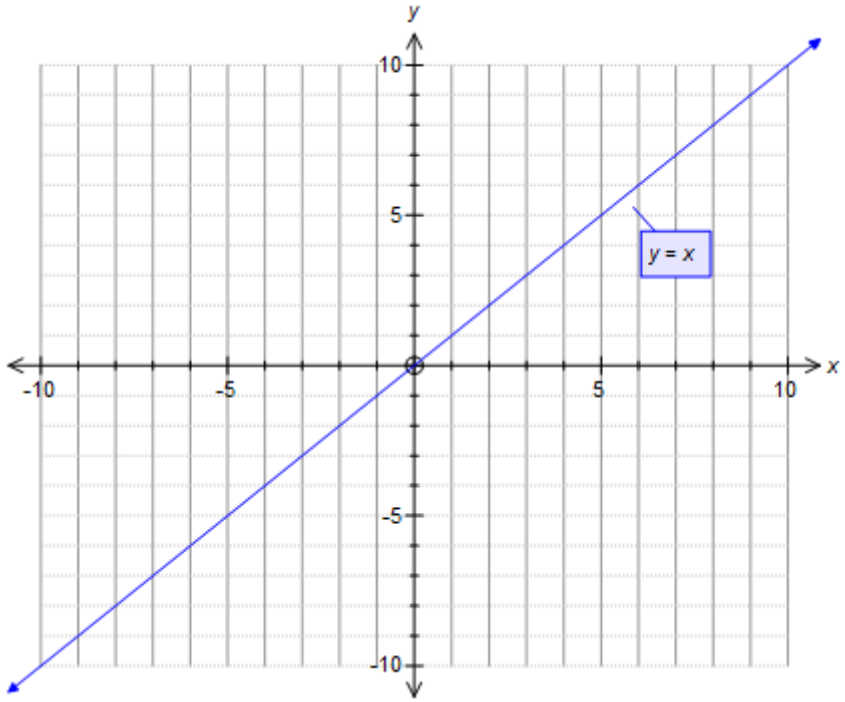
6.

- Tanh : 雙曲正切函數



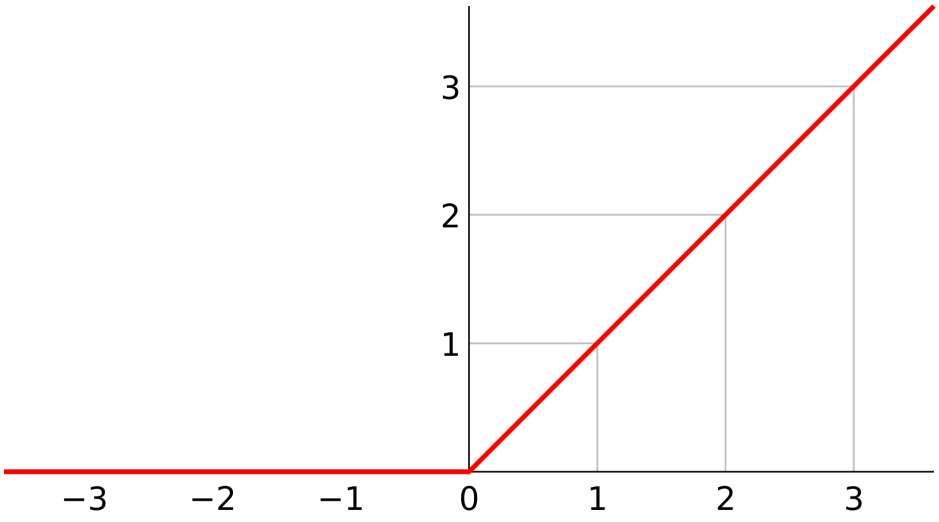
在訓練所有的問題都有不錯的表現，略遜於 ReLU。

- Linear：



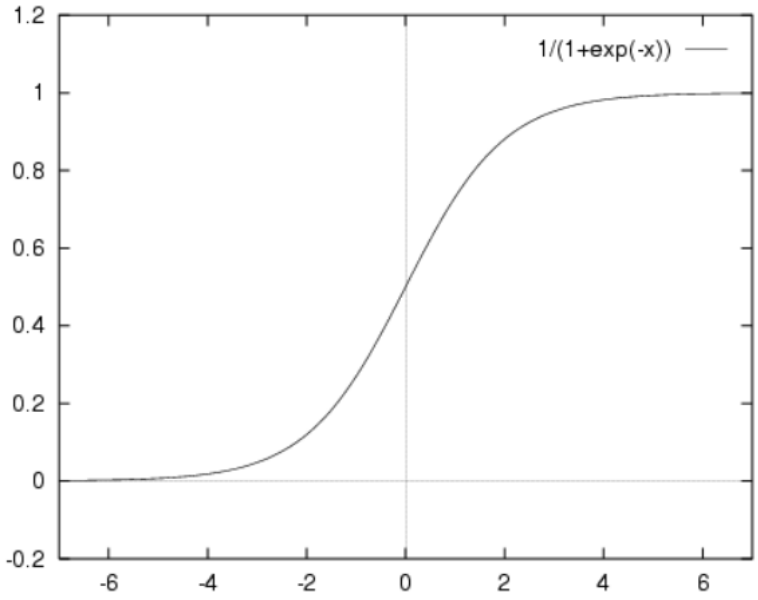
除了螺旋狀的問題，都可以很快跑出結果，螺旋狀問題我自己沒有成功跑出來，幾乎沒有反應。

- ReLU：線性整流函式



可以成功訓練出所有問題的解，速度也相當快速，能有效克服梯度消失的問題。

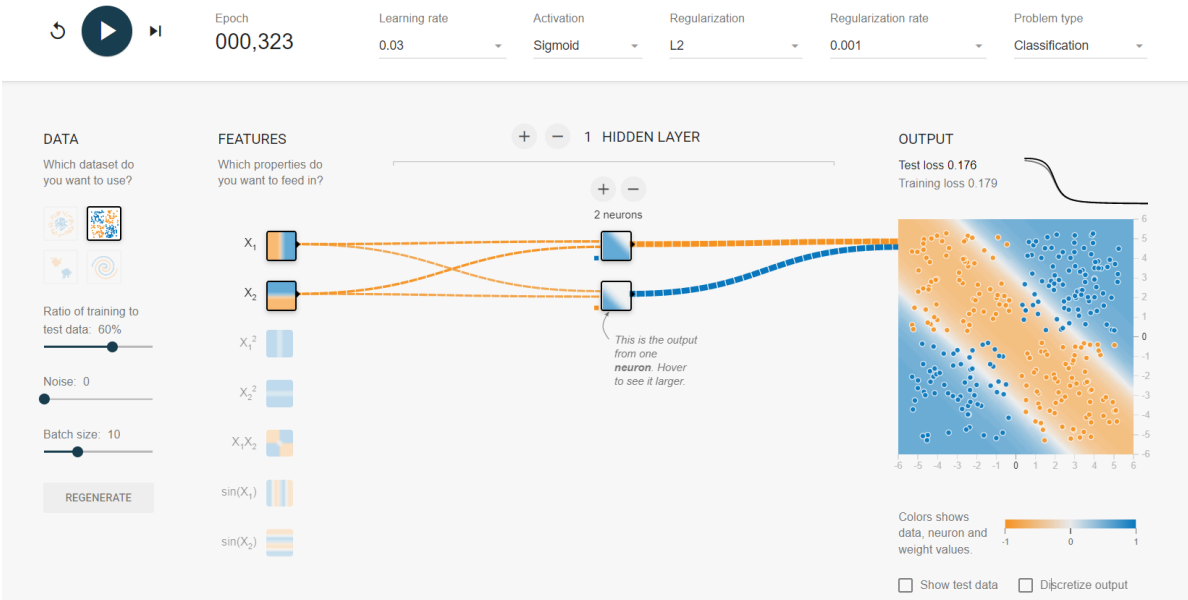
- Sigmoid：二焦點曲線函數

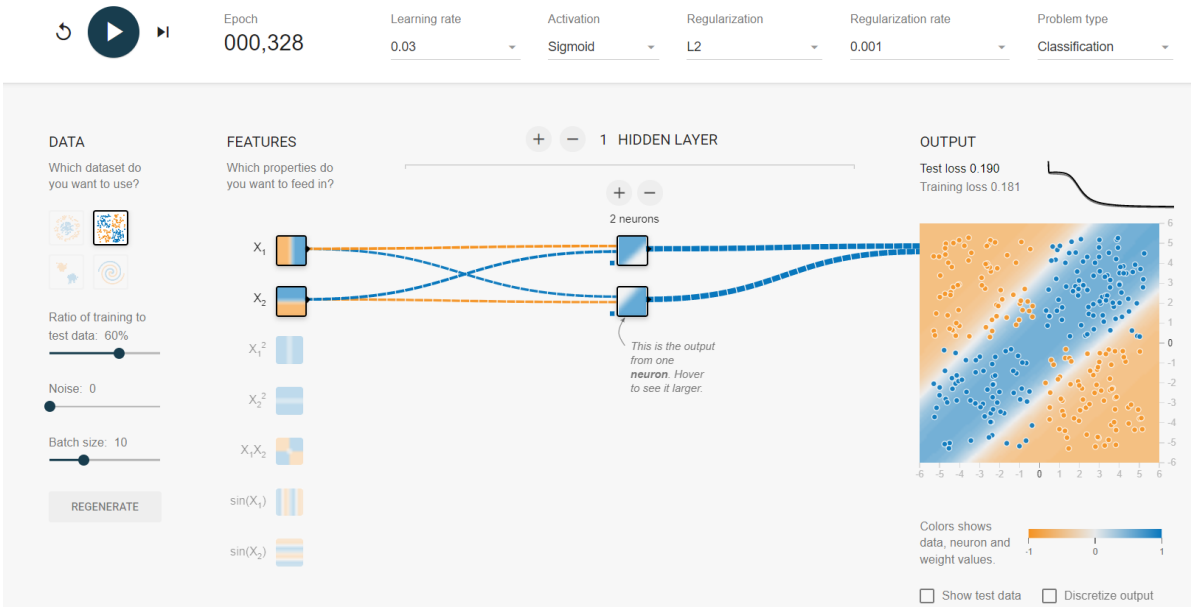


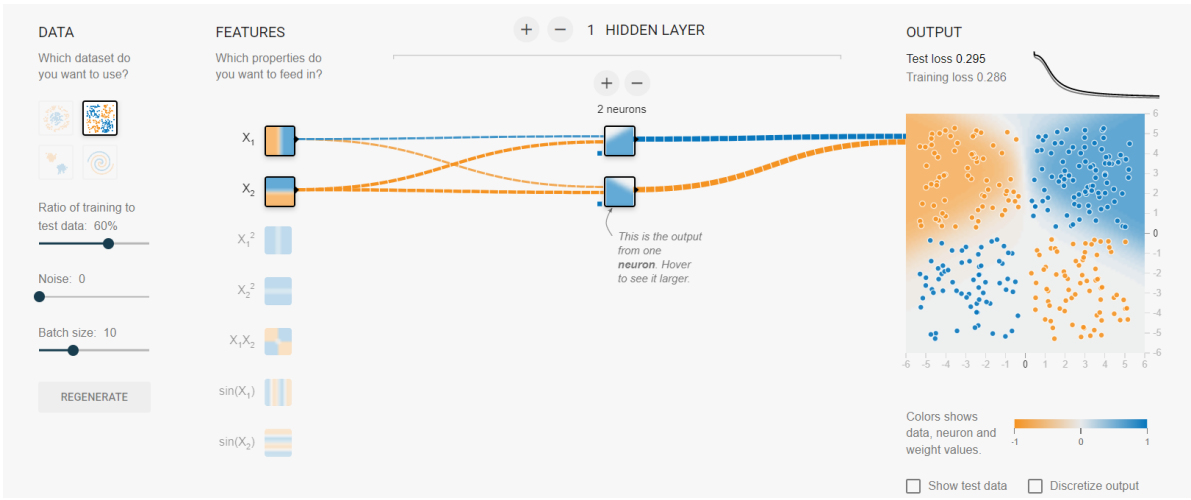
能夠成功訓練四種問題，和 tanh 速度差不多。

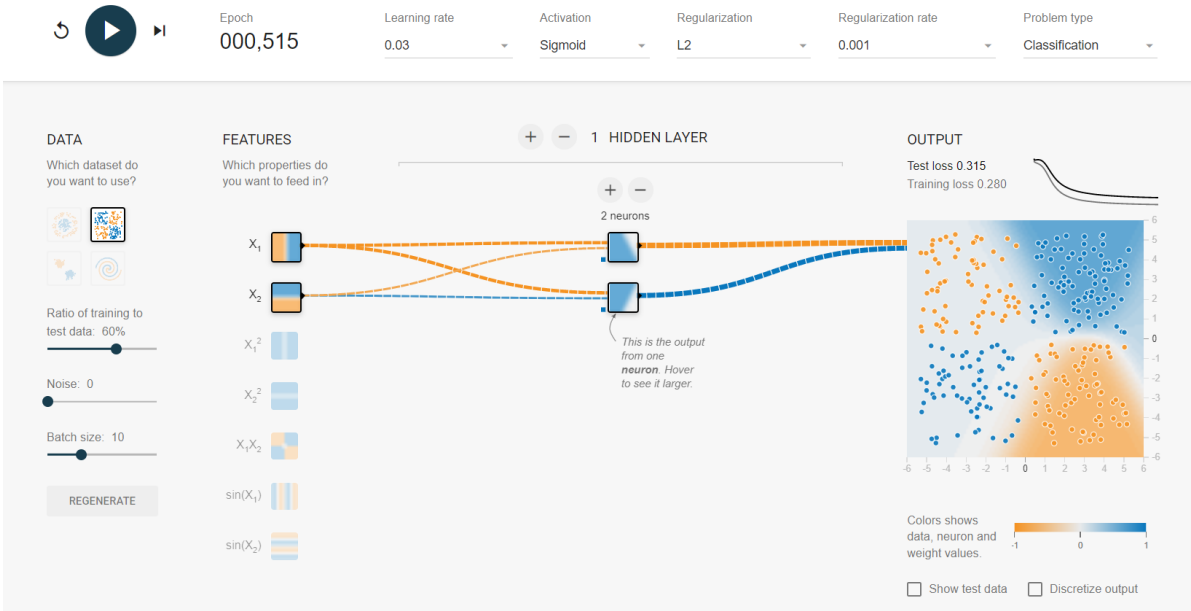
7.

如果只用 X_1, X_2 的話，這是個沒有辦法得到解答的問題，並且會訓練出幾種結果：

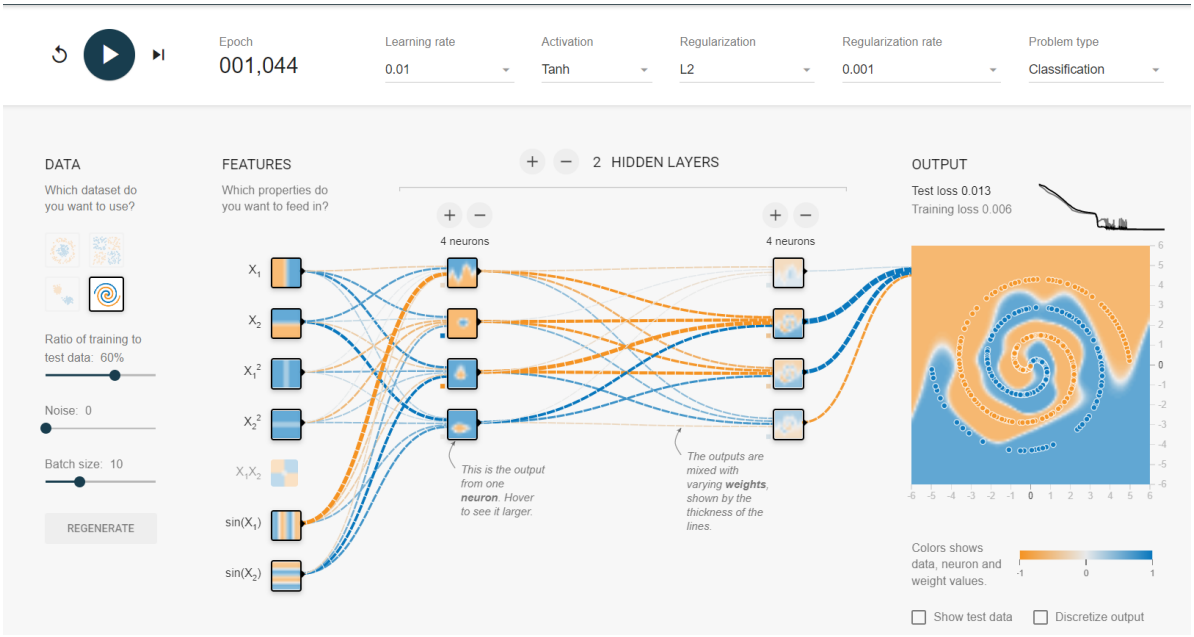



Epoch
000,328
Learning rate
0.03
Activation
Sigmoid
Regularization
L2
Regularization rate
0.001
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 60%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
2 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.190
Training loss 0.181
Colors shows data, neuron and weight values.
Show test data
Discretize output


DATA
Which dataset do you want to use?
Ratio of training to test data: 60%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
2 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.295
Training loss 0.286
Colors shows data, neuron and weight values.
Show test data
Discretize output

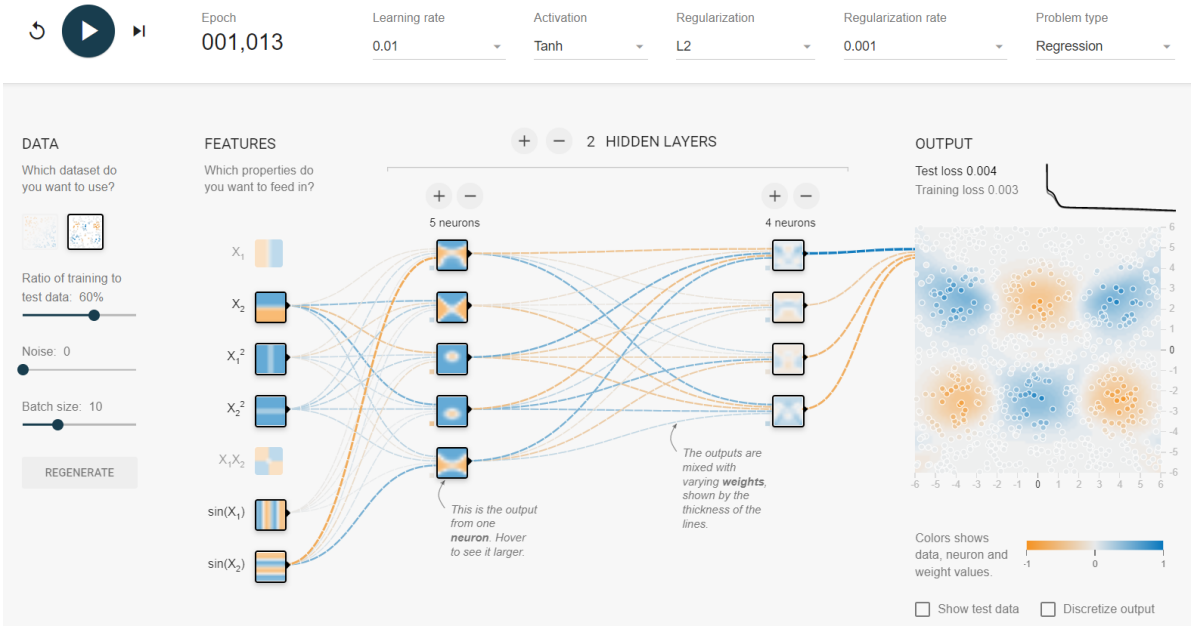

Epoch
000,515
Learning rate
0.03
Activation
Sigmoid
Regularization
L2
Regularization rate
0.001
Problem type
Classification
DATA
Which dataset do you want to use?
Ratio of training to test data: 60%
Noise: 0
Batch size: 10
REGENERATE
FEATURES
Which properties do you want to feed in?
1 HIDDEN LAYER
2 neurons
This is the output from one neuron. Hover to see it larger.
OUTPUT
Test loss 0.315
Training loss 0.280
Colors shows data, neuron and weight values.
Show test data
Discretize output

8.



從原本全部加滿，慢慢刪減，試著用最少的 layers 和 neurons，正確率參考他右上角的 Training loss。大概花了一個小時在玩這個圖形訓練，我覺得已經算滿意了。

9.



試了幾次就得到了還不錯的結果，正確率一樣參考他算的 Training loss。整個訓練和測試過程大概10分鐘。

10.

在跑上面兩個訓練的時候，都是使用 Tahn 的 activation function，不過後來回去寫第六題並試著找出差異的時候，發現 ReLU 的表現普遍比較好，在 Regression 訓練沒有 Tahn 快，但在 Classification 的速度就比較快。而 Linear 則是沒辦法訓練出螺旋狀的問題， sigmoid 則是和 Tahn 速度差不多。

11.

L1 和 L2 蠻難解釋的，在找資料時自己先看懂都很困難了。

TensorFlow Playground 裡的 Linear Activation 函數到底是什麼，也找了很久，過程中還一直看到線性回歸和羅吉斯回歸，後來詢問同學後才發現原來只是簡單的 $x = y$。

看不太懂第7題想要做什麼。

螺旋狀的分類問題比想像中還難達成，有看到同學曾經比我用更少的 neurons 得到比我還要好的訓練結果。

12.

1. Regularization in Deep Learning – L1, L2, and Dropout：https://towardsdatascience.com/regularization-in-deep-learning-l1-l2-and-dropout-377e75acc036
2. Machine Learning — 給自己的機器學習筆記 — Linear Regression — 迴歸模型介紹與公式原理教學：https://chwang12341.medium.com/machine-learning-%E7%B5%A6%E8%87%AA%E5%B7%B1%E7%9A%84%E6%A9%9F%E5%99%A8%E5%AD%B8%E7%BF%92%E7%AD%86%E8%A8%98-linear-regression-%E8%BF%B4%E6%AD%B8%E6%A8%A1%E5%9E%8B%E4%BB%8B%E7%B4%B9%E8%88%87%E5%85%AC%E5%BC%8F%E5%8E%9F%E7%90%86%E6%95%99%E5%AD%B8-35e34ad8c690
3. 深度學習：使用激勵函數的目的、如何選擇激勵函數 Deep Learning: the role of the activation function：https://mropengate.blogspot.com/2017/02/deep-learning-role-of-activation.html