Lab1 - Understand NN and Training Process

Advisor：Tsai, Chia-Chi

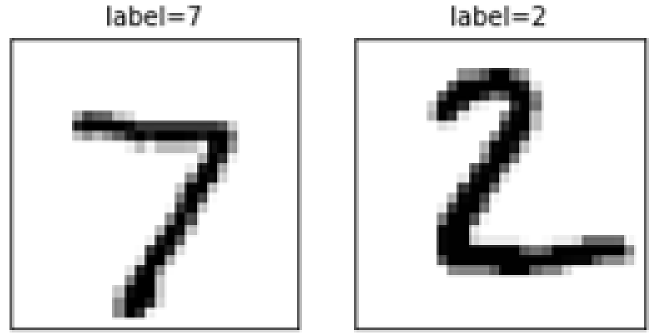
TA：張茹涵

Outline

1. Lab1 task
2. Google colab
3. How NN works
4. MNIST dataset

Tasks

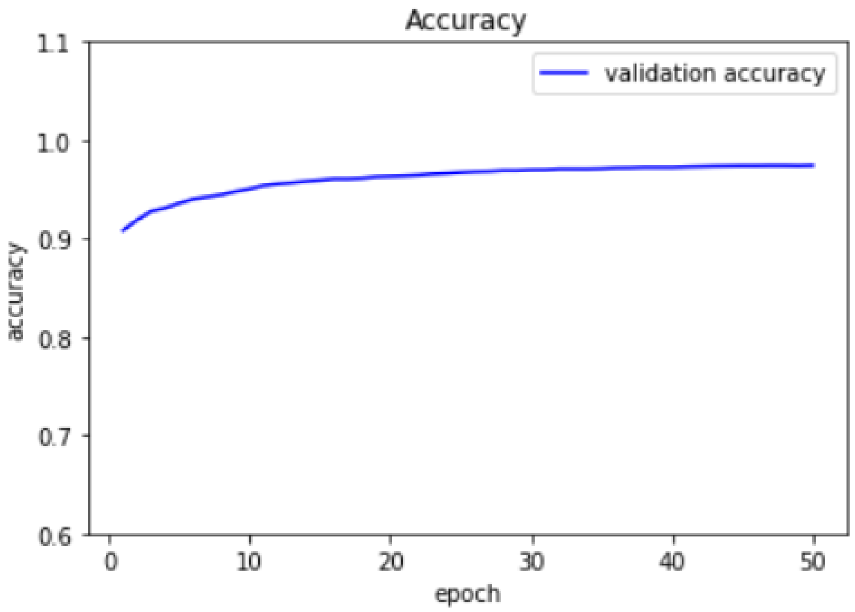
- Implement the following layers as a python function(both forward and backward propagation)
  - Inner-product layer (10%)
  - Activation layer(Sigmoid or Rectified) (10%)
  - Softmax layer (10%)
- Implement training and testing process
  - included cross-validation (5%)
  - use cross-entropy as loss function (5%)
- Build neural network to solve the MNIST classification problem
  - At least one hidden layer neural network (cannot use convolutional layer) (10%)
  - accuracy must > 90%
- Print and Plot accuracy (test)and loss(train&val) curves of the neural networks (10%)
- Report (40%)

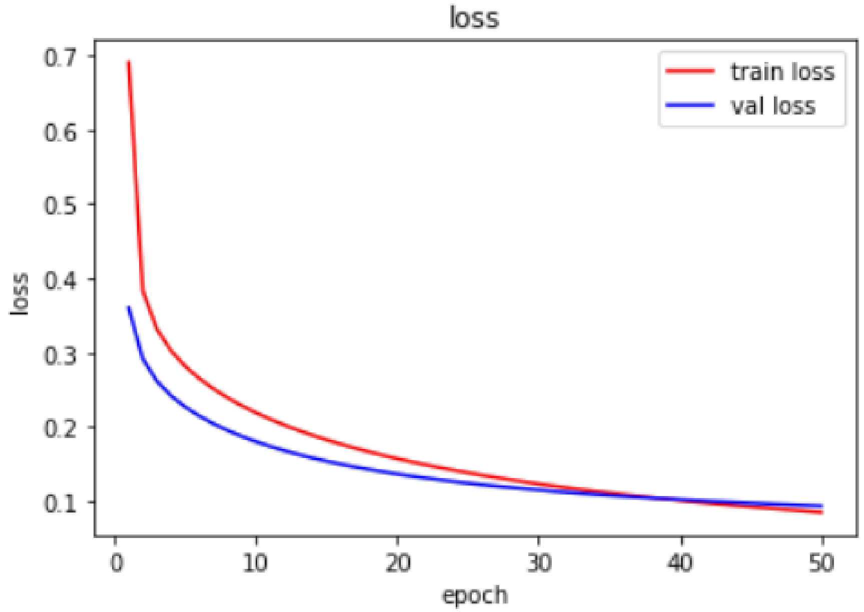


Report your result

- print val accuracy of each epoch & final test accuracy
- plot epoch-accuracy and epoch-loss

```
epoch:  45
val accuracy:  0.9738333333333333
epoch:  46
val accuracy:  0.9738333333333333
epoch:  47
val accuracy:  0.974
epoch:  48
val accuracy:  0.974
epoch:  49
val accuracy:  0.9738333333333333
epoch:  50
val accuracy:  0.9741666666666666
test accuracy:  0.9697
```



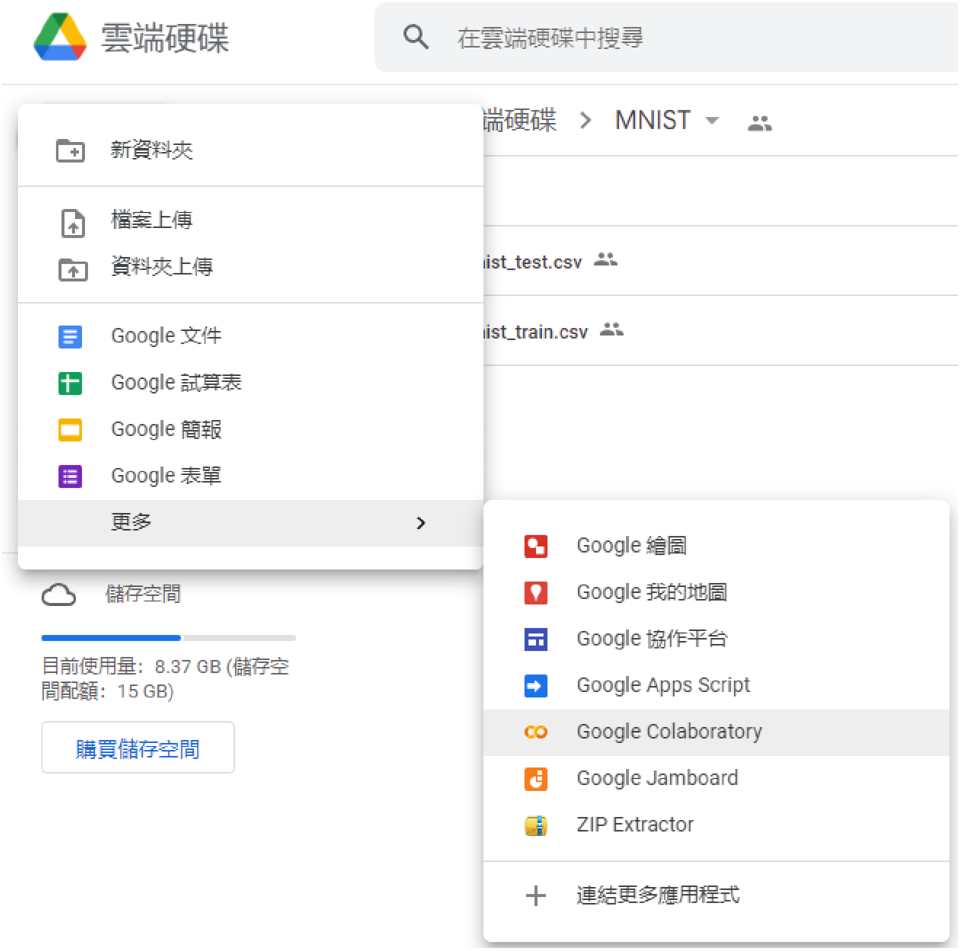


Outline

1. Lab1 task
2. Google colab
3. How NN works
4. MNIST dataset

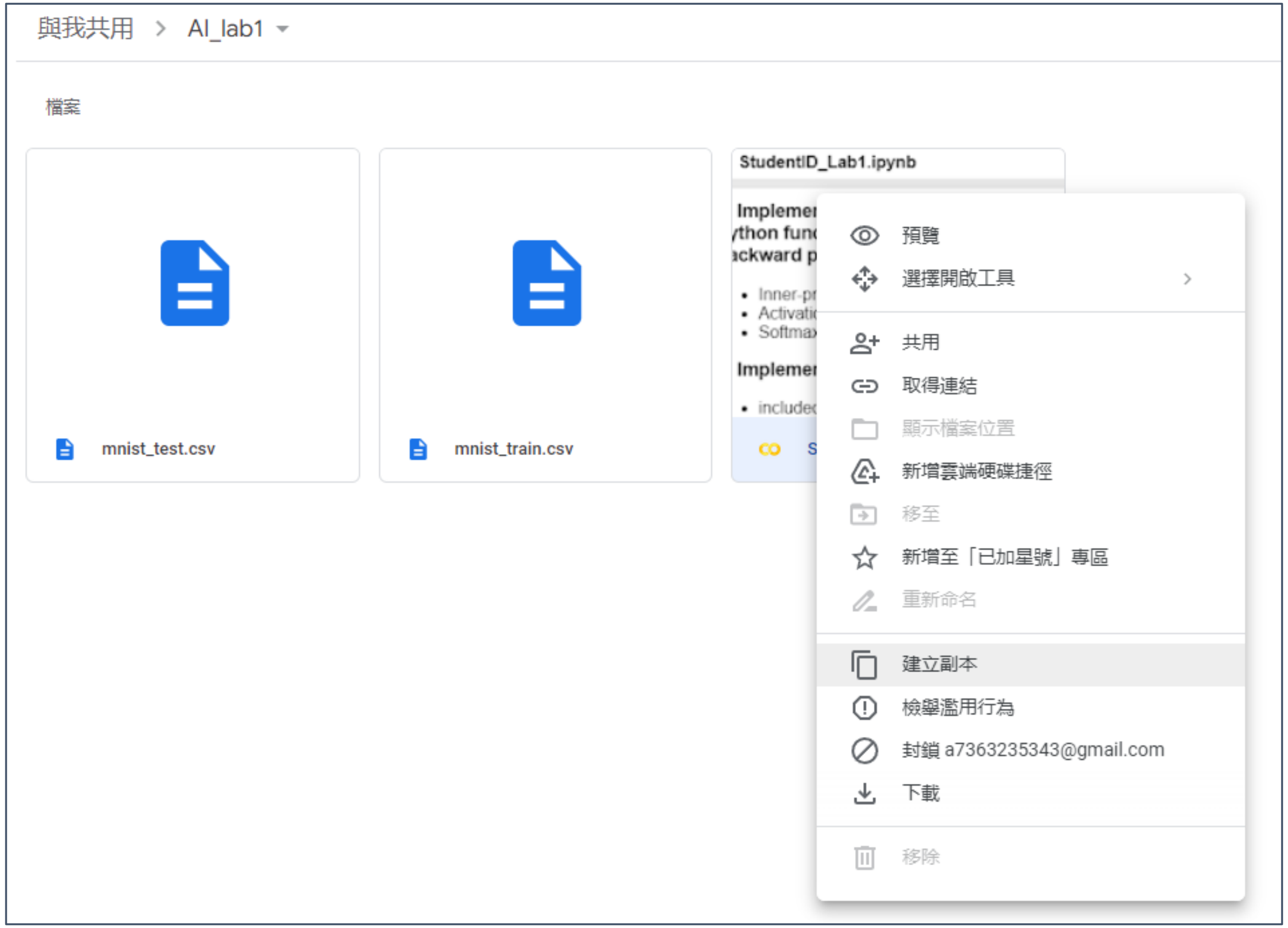
Google colab

- Go to Google Drive and create a Google Colaboratory file



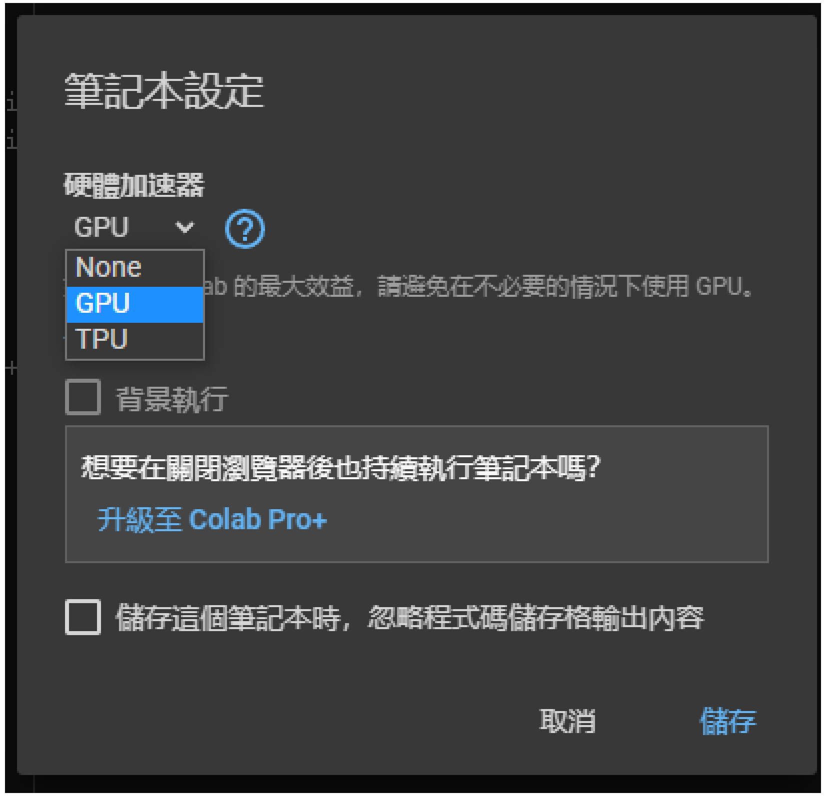
Google colab

- 直接在sample code點選 "在雲端硬碟中儲存副本"



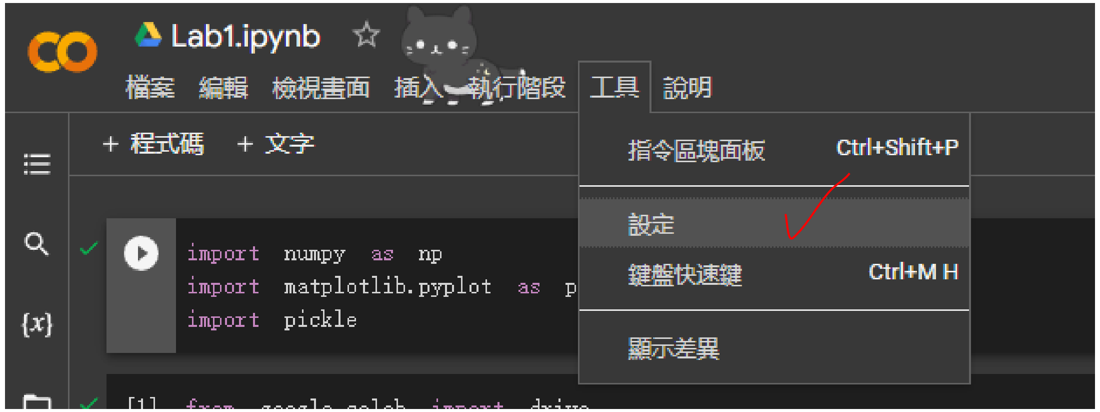
Google colab

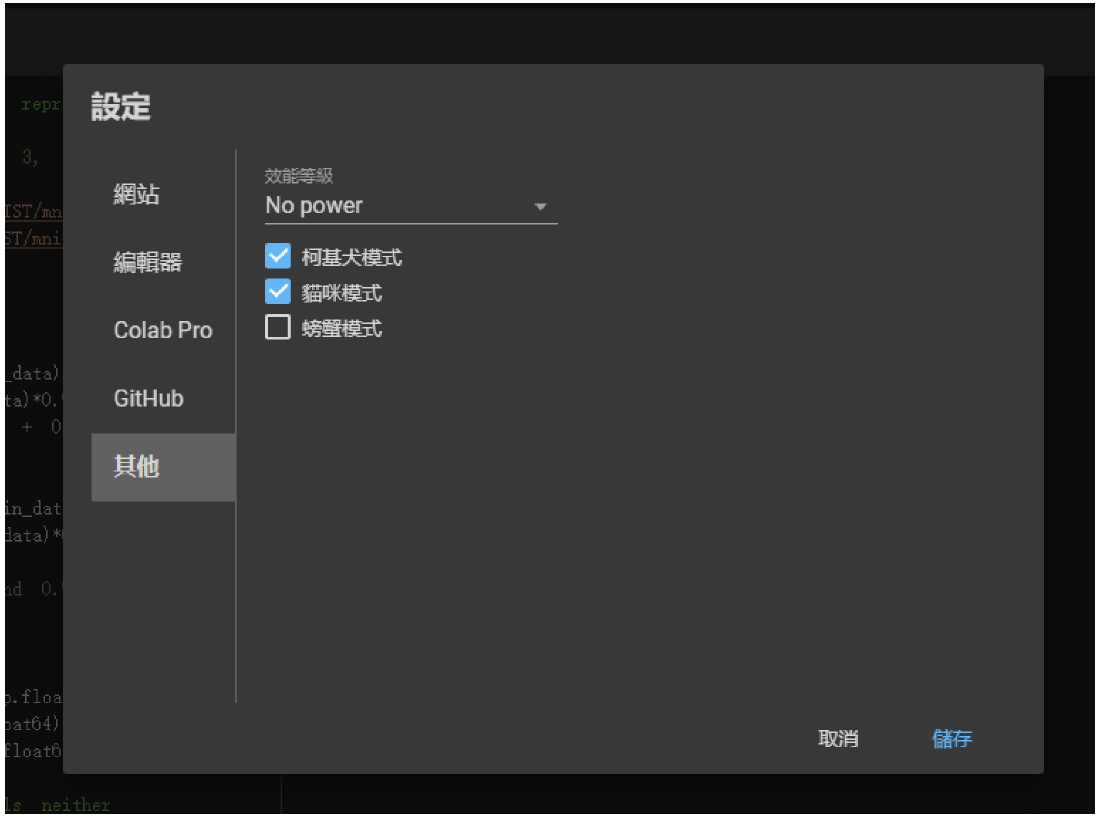
- Choose GPU as runtime(Default : CPU)

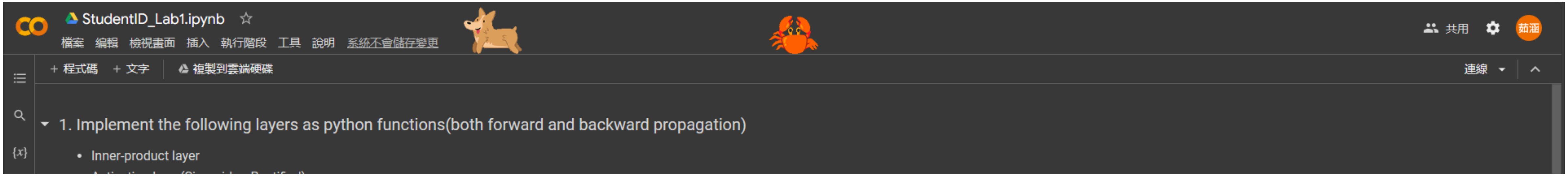


Google colab

- Choose a pet

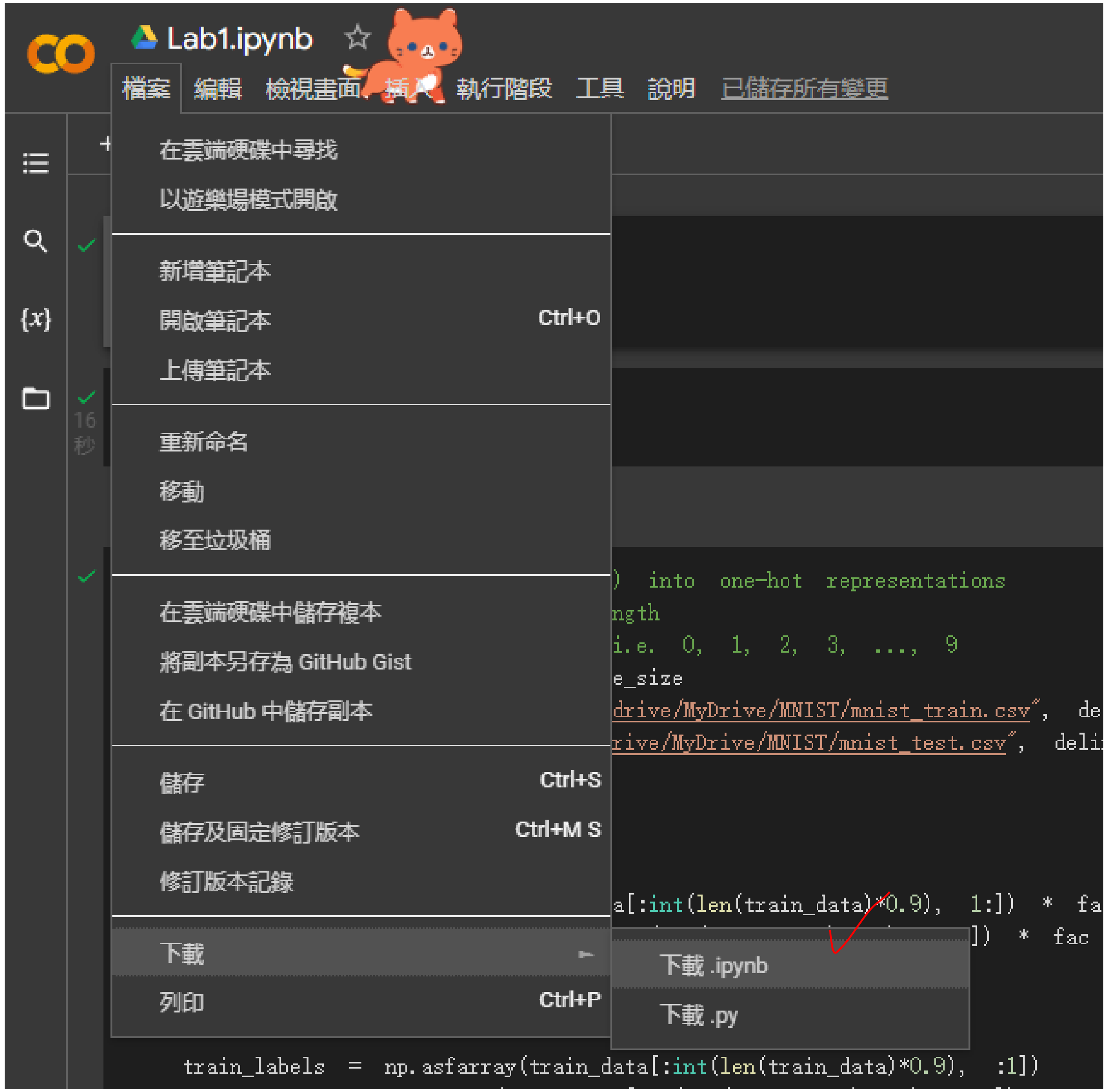






Google colab

- Save : *.ipynb

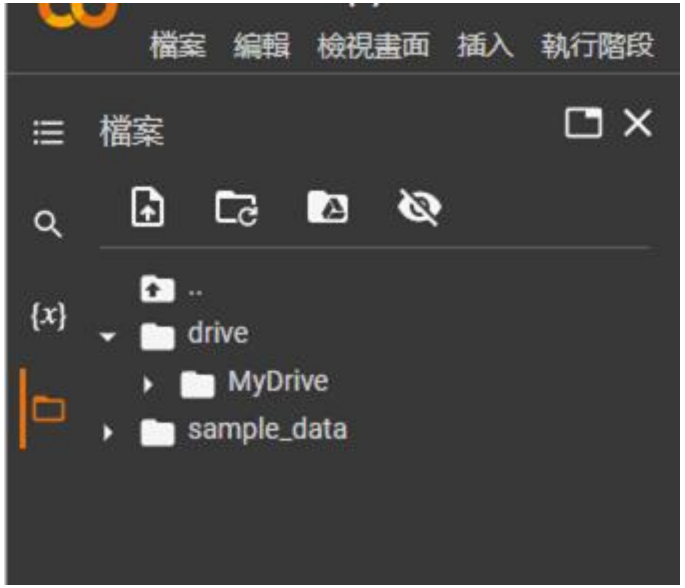


Google colab

- 連結到雲端資料夾並設定當前路徑

```
[ ]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive
```

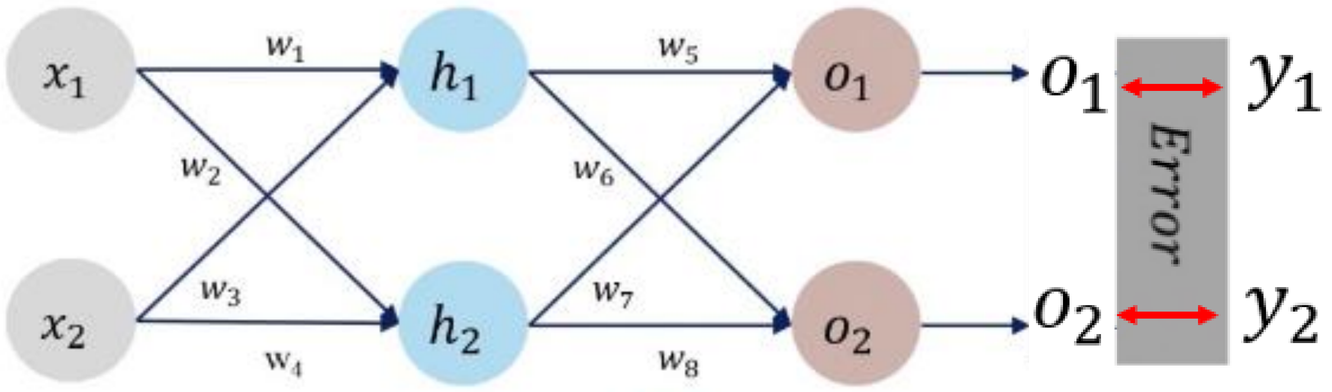
```
[ ]  %cd /content/drive/MyDrive/lab1

     /content/drive/MyDrive/lab1
```

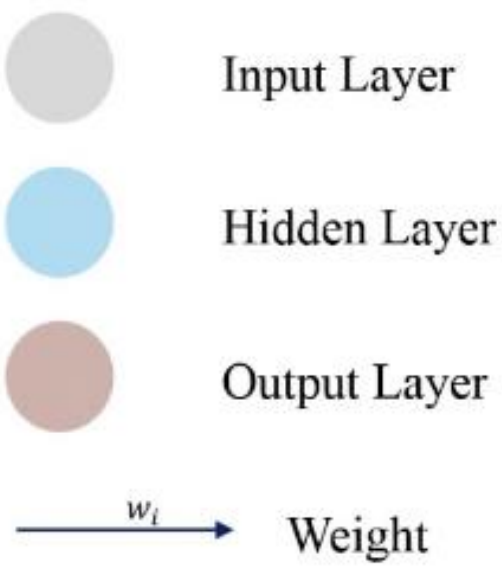


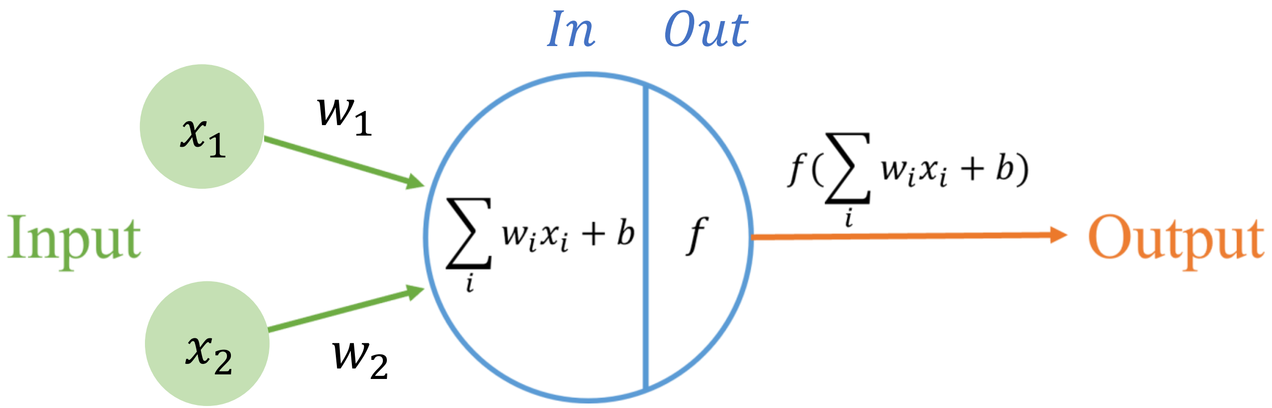
Outline

1. Lab1 task
2. Google colab
3. How NN works
4. MNIST dataset

NN training





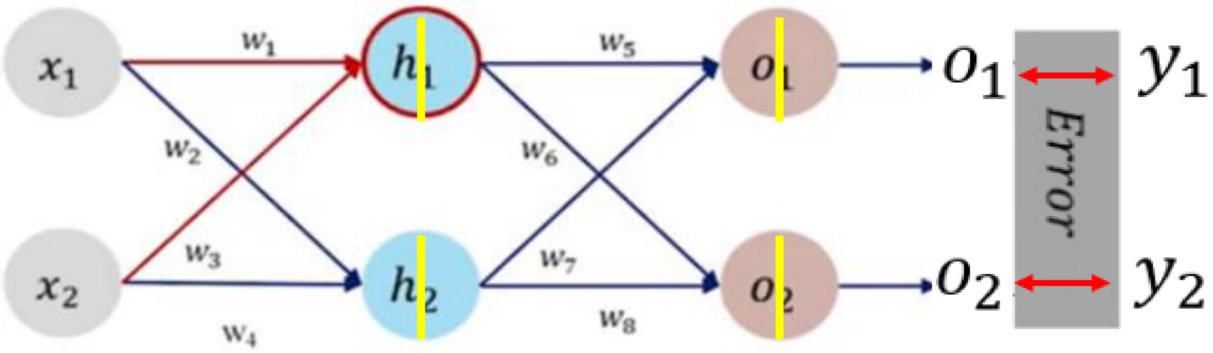


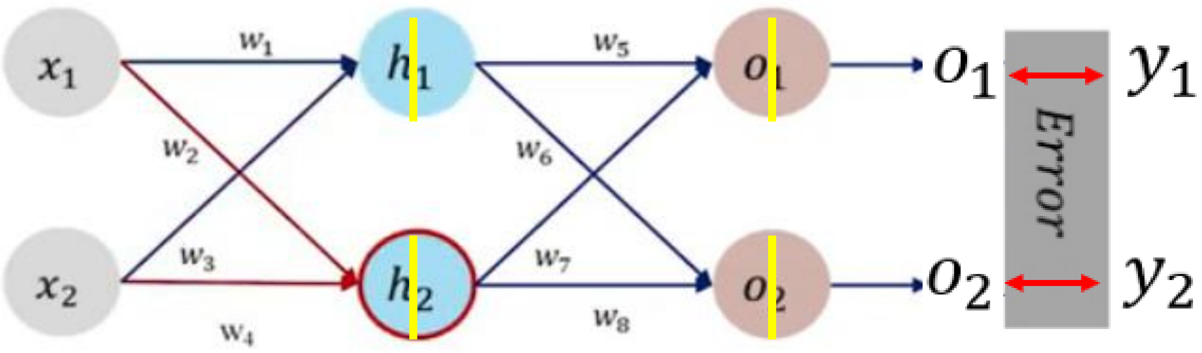
$$y = \sum_{i=1}^{N} w_i x_i + b$$

f *is activation function*

$$f(x) = Sigmoid(x)$$

NN training - forward propagation





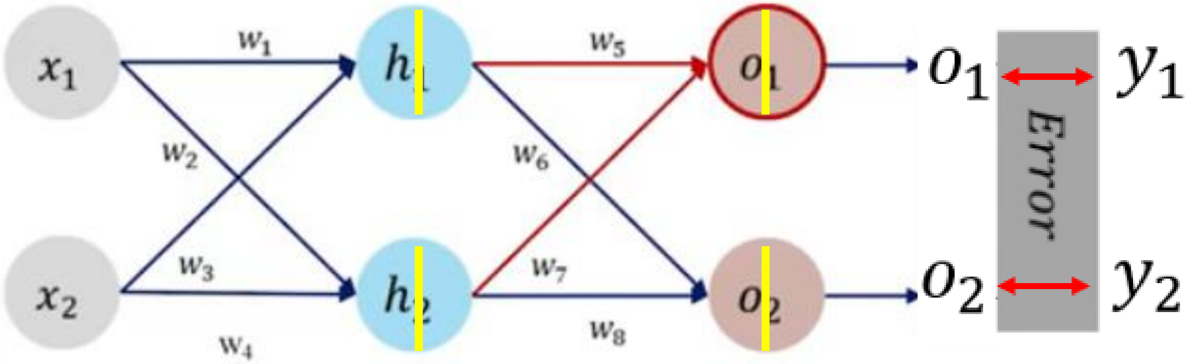
$$In_{h_1} = w_1 * x_1 + w_3 * x_2$$

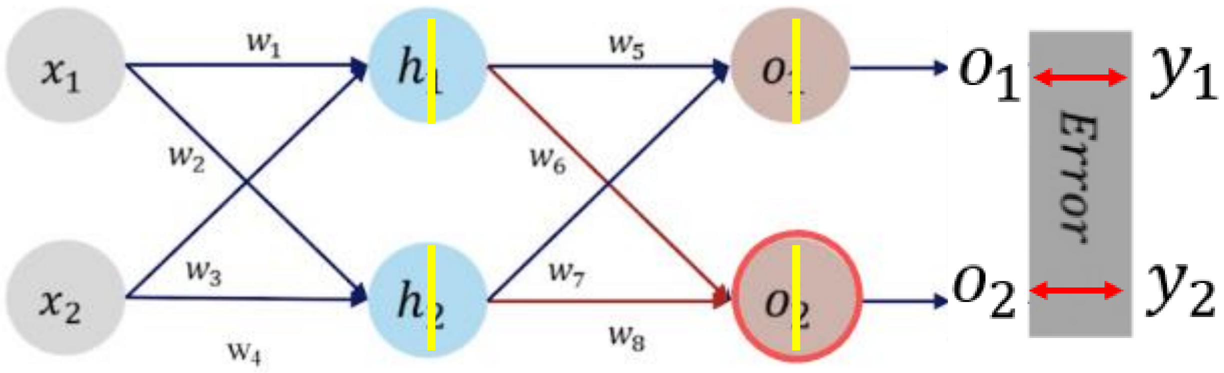
$$h_1 = Out_{h_1} = Sigmoid(In_{h_1})$$

$$In_{h_2} = w_2 * x_1 + w_4 * x_2$$

$$h_2 = Out_{h_2} = Sigmoid(In_{h_2})$$

NN training - forward propagation





$$In_{o_1} = w_5 * h_1 + w_7 * h_2$$

$$o_1 = Out_{o_1} = Sigmoid(In_{o_1})$$

$$In_{o_2} = w_6 * h_1 + w_8 * h_2$$

$$o_2 = Out_{o_2} = Sigmoid(In_{o_2})$$

$$MSE: \quad Error = \frac{1}{2}\sum_{i=1}^{2}(o_i - y_i)^2$$

Gradient Descent (梯度下降)

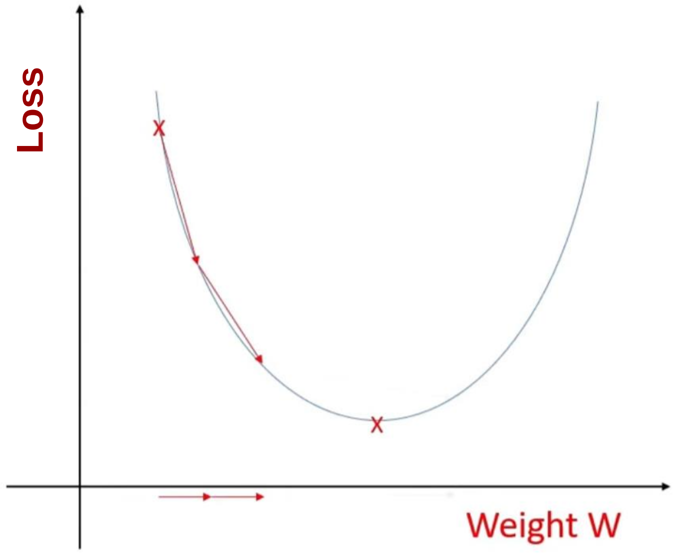
- In the neural networks, there are only the inner-product layers have parameters to be optimized
- Using gradient descent to optimize parameters in inner-product layers

$$\boldsymbol{W}^{new} = \boldsymbol{W}^{old} - \eta \nabla_{\boldsymbol{W}} E$$

$$\boldsymbol{b}^{new} = \boldsymbol{b}^{old} - \eta \nabla_{\boldsymbol{b}} E$$

↑

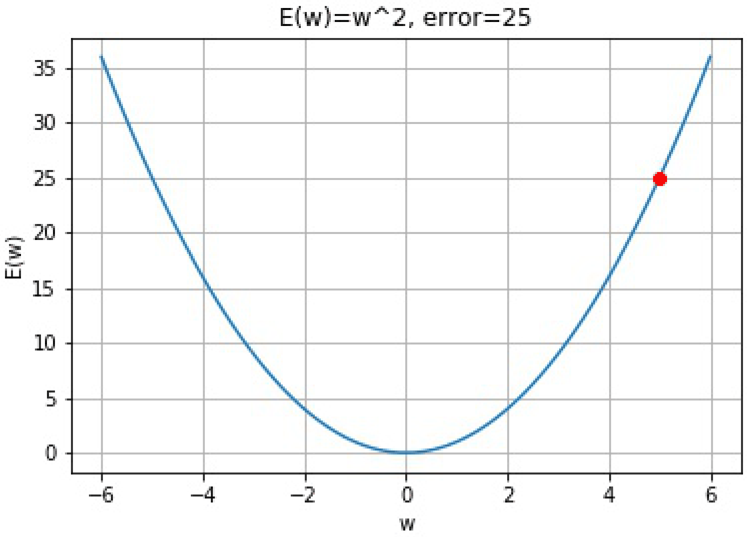
learning rate (步長)



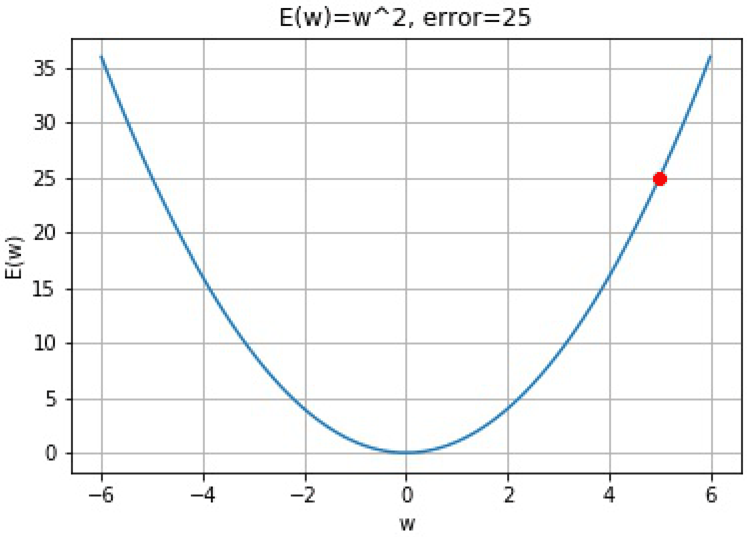
Gradient Descent (梯度下降)

- The larger the learning rate, the longer the distance of each movement
- If the learning rate is too large, there will often be instability
- If the learning rate is too small, more iterations are needed to reach the minimum value

Learning rate = 0.9



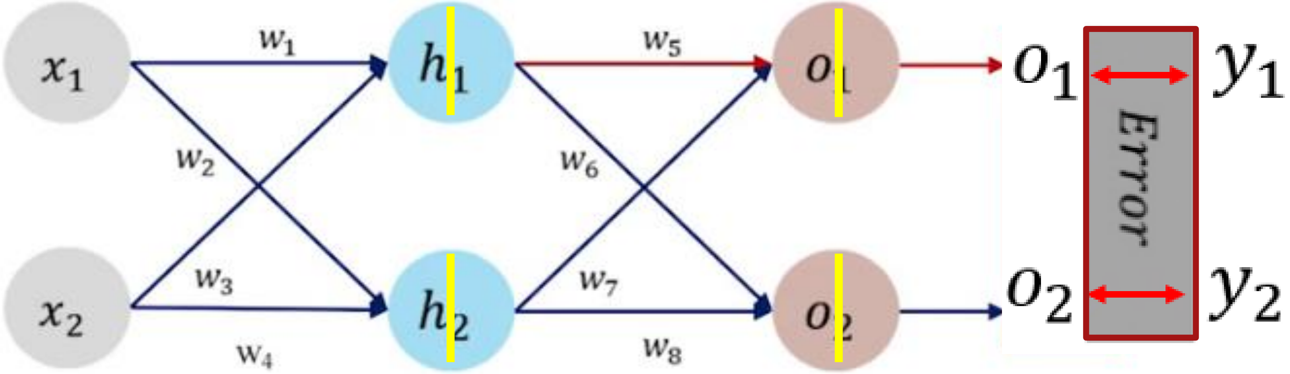
Learning rate = 0.1



NN training - backward propagation

- 計算 w_5 到 w_8 的梯度 (誤差對每個權重的變化)

(以 w_5為例)



$$\delta_5 = \frac{\partial Error}{\partial w_5} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial w_5}$$

$$\frac{\partial Error}{\partial o_1} = o_1 - y_1 \quad \Leftarrow \quad Error = \frac{1}{2}\sum_{i=1}^{2}(o_i - y_i)^2$$

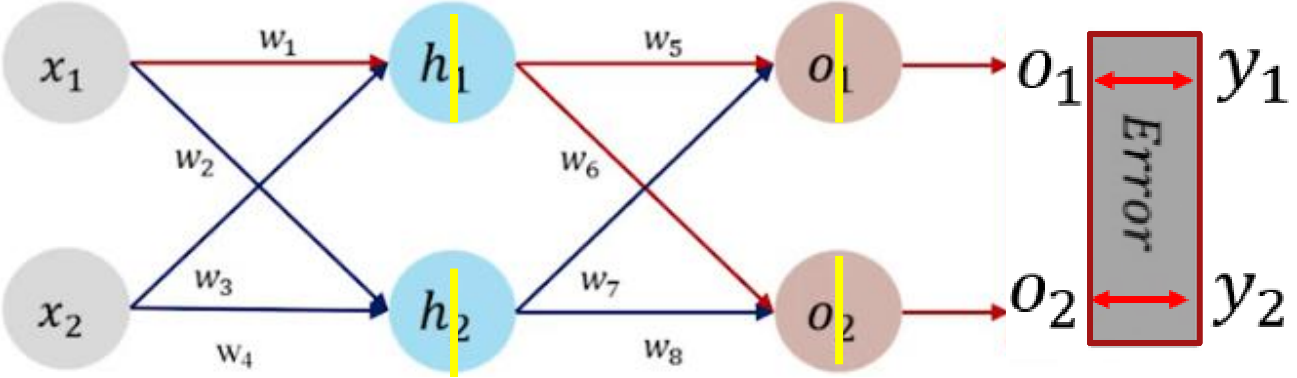
$$\frac{\partial o_1}{\partial In_{o_1}} = \text{Sigmoid}'(in_{o1}) \quad \Leftarrow \quad o_1 = Out_{o_1} = Sigmoid(In_{o_1})$$

$$\frac{\partial In_{o_1}}{\partial w_5} = h_1 \quad \Leftarrow \quad In_{o_1} = w_5 * h_1 + w_7 * h_2$$

NN training - backward propagation

- 計算 w_1 到 w_4 的梯度 (誤差對每個權重的變化)

(以 w_1 為例)



$$\delta_1 = \frac{\partial Error}{\partial w_1} = \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial w_1} + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial w_1}$$

$$= \frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial h_1} * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_1} + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial In_{o_2}} * \frac{\partial In_{o_2}}{\partial h_1} * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_1}$$

$$= (\frac{\partial Error}{\partial o_1} * \frac{\partial o_1}{\partial In_{o_1}} * \frac{\partial In_{o_1}}{\partial h_1} + \frac{\partial Error}{\partial o_2} * \frac{\partial o_2}{\partial In_{o_2}} * \frac{\partial In_{o_2}}{\partial h_1}) * \frac{\partial h_1}{\partial In_{h_1}} * \frac{\partial In_{h_1}}{\partial w_1}$$

$$= [(o_1 - y_1) \times \text{sigmoid}'(In_{o1}) \times w_5 + (o_2 - y_2) \times \text{sigmoid}'(In_{o2}) \times w_6] \times \text{sigmoid}'(In_{h1}) \times x_1$$

NN training - backward propagation

- 更新權重

$$\boldsymbol{W}^{new} = \boldsymbol{W}^{old} - \eta\nabla_{\boldsymbol{W}}E$$

$\delta_1 \sim \delta_8$

$$\boldsymbol{b}^{new} = \boldsymbol{b}^{old} - \eta\nabla_{\boldsymbol{b}}E$$

Inner-product layer

- Forward propagation

$$X = \begin{bmatrix} x_1 & \dots & x_i \end{bmatrix} \quad W = \begin{bmatrix} w_{11} \dots w_{1j} \\ \vdots \quad \ddots \quad \vdots \\ w_{i1} \dots w_{ij} \end{bmatrix} \quad B = \begin{bmatrix} b_1 & \dots & b_j \end{bmatrix}$$

$$Y = XW + B$$

i : the number of input neurons
j : the number of output neurons

Inner-product layer

- Backward propagation

$$\nabla_x E = \left[\frac{\partial E}{\partial x_1} \quad \frac{\partial E}{\partial x_2} \quad \cdots \quad \frac{\partial E}{\partial x_i} \right]$$

$$= \left[\left(\frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_1} + \ldots + \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_1} \right) \quad \cdots \quad \left(\frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_i} + \ldots + \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial x_i} \right) \right] = \left[\frac{\partial E}{\partial y_1} \quad \cdots \quad \frac{\partial E}{\partial y_j} \right] \mathbf{A}$$

$$\nabla_{\boldsymbol{w}} E = \begin{bmatrix} \frac{\partial E}{\partial w_{11}} & \cdots & \frac{\partial E}{\partial w_{1j}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial w_{i1}} & \cdots & \frac{\partial E}{\partial w_{ij}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{11}} & \cdots & \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial w_{1j}} \\ \vdots & \ddots & \vdots \\ \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial w_{i1}} & \cdots & \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}} \end{bmatrix} = \mathbf{B} \left[\frac{\partial E}{\partial y_1} \cdots \frac{\partial E}{\partial y_j} \right]$$

$$\nabla_b E = \left[\frac{\partial E}{\partial b_1} \quad \cdots \quad \frac{\partial E}{\partial b_j} \right] = \left[\frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial b_1} \quad \cdots \quad \frac{\partial E}{\partial y_j}\frac{\partial y_j}{\partial b_j} \right] = \left[\frac{\partial E}{\partial y_1} \quad \cdots \quad \frac{\partial E}{\partial y_j} \right] \mathbf{C}$$

$$Y = XW + B$$

$$\frac{\partial E}{\partial X} = \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial X} = \frac{\partial E}{\partial Y} W$$

$$\frac{\partial E}{\partial W} = \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial W} = X \frac{\partial E}{\partial Y}$$

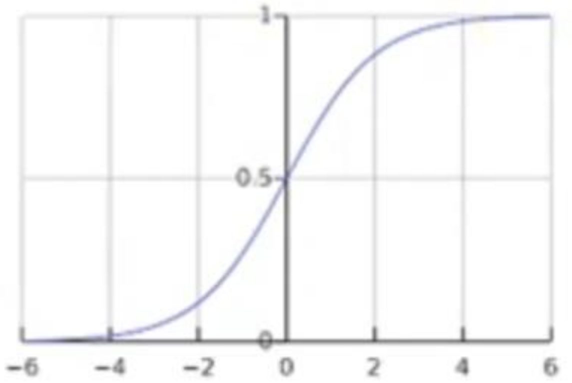
$$\frac{\partial E}{\partial B} = \frac{\partial E}{\partial Y}\frac{\partial Y}{\partial B} = \frac{\partial E}{\partial Y} 1$$

Please find the matrices **A**, **B** and **C**

Activation layer- Sigmoid function

- forward propagation

$$\boldsymbol{y}(\boldsymbol{x}) = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1/(1+e^{-x_1}) \\ \vdots \\ 1/(1+e^{-x_n}) \end{bmatrix}$$



- backward propagation

$$\nabla_{\boldsymbol{x}} E = \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial y_n}\frac{\partial y_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_1} \\ \vdots \\ \frac{\partial E}{\partial y_n} \end{bmatrix} \boldsymbol{D}$$

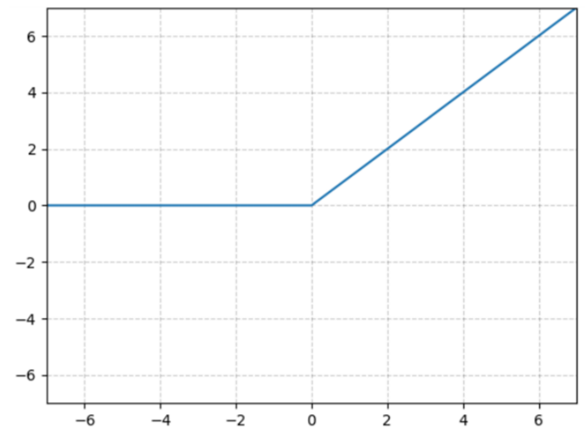
$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial E}{\partial y}\underbrace{\frac{d}{dx}f(x)}_{\boldsymbol{D}}$$

Please find the matrix $\boldsymbol{D}$

Activation layer- Rectified function

- forward propagation

$$\boldsymbol{y}(\boldsymbol{x}) = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} [x_1 > 0] \cdot x_1 \\ \vdots \\ [x_n > 0] \cdot x_n \end{bmatrix}$$



- backward propagation

$$\nabla_{\boldsymbol{x}} E = \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_1}\frac{\partial y_1}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial y_n}\frac{\partial y_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_1} \\ \vdots \\ \frac{\partial E}{\partial y_n} \end{bmatrix} \boldsymbol{E}$$

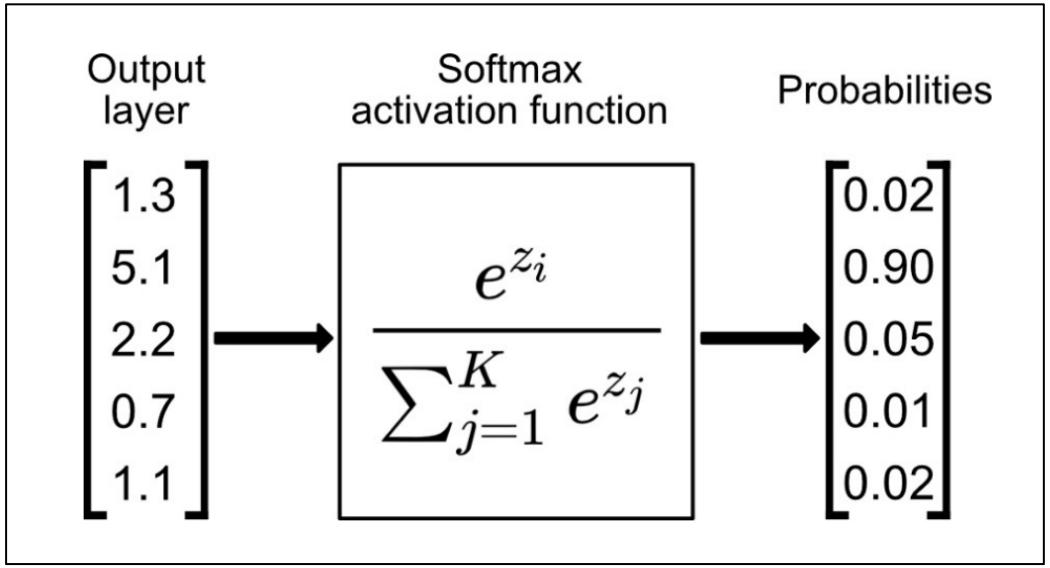
$$\frac{\partial E}{\partial x} = \frac{\partial E}{\partial y}\frac{\partial y}{\partial x} = \frac{\partial E}{\partial y}\underbrace{\frac{d}{dx} f(x)}_{\boldsymbol{E}}$$

Please find the matrix $\boldsymbol{E}$

Softmax layer

- forward propagation

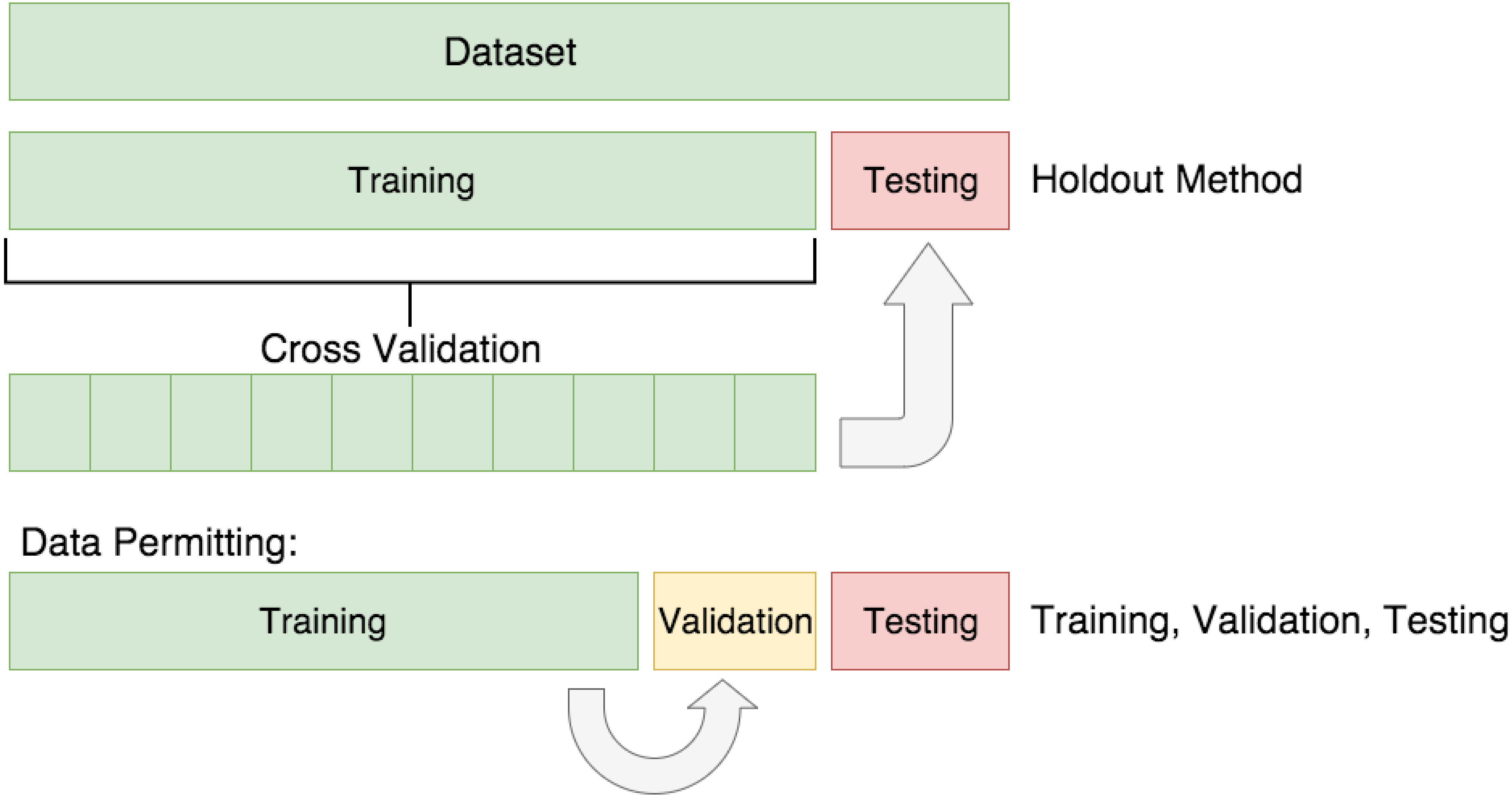
$$\boldsymbol{y}(\boldsymbol{x}) = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \frac{1}{\sum_{i=1}^{n} e^{x_i}} \begin{bmatrix} e^{x_1} \\ \vdots \\ e^{x_n} \end{bmatrix}$$



- backward propagation

$$\nabla_{\boldsymbol{x}} E(\boldsymbol{y}, \boldsymbol{t}) = \begin{bmatrix} \frac{\partial E}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial x_n} \end{bmatrix} = \begin{bmatrix} \frac{\partial E}{\partial y_1} \frac{\partial y_1}{\partial x_1} \\ \vdots \\ \frac{\partial E}{\partial y_n} \frac{\partial y_n}{\partial x_n} \end{bmatrix} = \begin{bmatrix} y_1 - t_1 \\ \vdots \\ y_n - t_n \end{bmatrix} = \boldsymbol{y} - \boldsymbol{t}$$

Cross validation



Outline

1. Lab1 task
1. Google colab
1. How NN works
1. MNIST dataset

MNIST Dataset

- MNIST dataset is a large dataset of handwritten digits
- Train: 60000 images (28*28 pixels) + one label for each image
  - train_images[60000][784] : consist of 60000 images
  - train_labels[60000] : consist of 60000 labels
- Test: 10000 images (28*28 pixels) + one label for each image
  - test_images[10000][784] : consist of 10000 images
  - test_labels[10000] : consist of 10000 labels

原始 Dataset

- How to load ubyte?

t10k-images-idx3-ubyte	2017/9/24 下午 04:46	檔案	7,657 KB
t10k-labels-idx1-ubyte	2017/9/24 下午 04:46	檔案	10 KB
train-images-idx3-ubyte	2017/9/24 下午 04:46	檔案	45,938 KB
train-labels-idx1-ubyte	2017/9/24 下午 04:46	檔案	59 KB

TRAINING SET IMAGE FILE (train-images-idx3-ubyte):

[offset]	[type]	[value]	[description]
0000	32 bit integer	0x00000803(2051)	magic number
0004	32 bit integer	60000	number of images
0008	32 bit integer	28	number of rows
0012	32 bit integer	28	number of columns
0016	unsigned byte	??	pixel
0017	unsigned byte	??	pixel
....			
xxxx	unsigned byte	??	pixel

```python
def convert(imgf, labelf, outf, n):
    f = open(imgf, "rb")
    o = open(outf, "w")
    l = open(labelf, "rb")

    f.read(16)
    l.read(8)
    images = []

    for i in range(n):
        image = [ord(l.read(1))]
        for j in range(28 * 28):
            image.append(ord(f.read(1)))
        images.append(image)

    for image in images:
        o.write(",".join(str(pix) for pix in image) + "\n")
    f.close()
    o.close()
    l.close()

convert("\MNIST\\train-images.idx3-ubyte", "\MNIST\\train-labels.idx1-ubyte",
        "\MNIST\mnist_train.csv", 60000)
convert("\MNIST\\t10k-images.idx3-ubyte", "\MNIST\\t10k-labels.idx1-ubyte",
        "\MNIST\mnist_test.csv", 10000)

print("Convert Finished!")
```

Coding Dataset

- mnist_train.csv & mnist_test.csv
- 讀寫容易，數據可以表格化展示

	A	B	C	D	E	F	G
1	7	0	0	0	0	0	0
2	2	0	0	0	0	0	0
3	1	0	0	0	0	0	0
4	0	0	0	0	0	0	0
5	4	0	0	0	0	0	0
6	1	0	0	0	0	0	0

label

pixel information

	MK	ML	MM	MN	MO	MP	MQ	MR	MS	MT
1	0	0	0	0	0	0	22	233	255	83
2	176	246	253	159	12	0	0	0	0	0
3	0	0	0	140	254	77	0	0	0	0
4	188	20	0	0	0	0	0	109	251	253
5	0	0	0	0	0	0	32	232	250	66
6	0	0	58	254	254	237	0	0	0	0

One-hot encoding

- easy to show predicted probability of each label

```
lr = np.arange(no_of_different_labels)

# transform labels into one hot representation
train_labels_one_hot = (lr==train_labels).astype(np.float)
val_labels_one_hot = (lr==val_labels).astype(np.float)
test_labels_one_hot = (lr==test_labels).astype(np.float)
```

```
label:  0  in one-hot representation:  [1 0 0 0 0 0 0 0 0 0]
label:  1  in one-hot representation:  [0 1 0 0 0 0 0 0 0 0]
label:  2  in one-hot representation:  [0 0 1 0 0 0 0 0 0 0]
label:  3  in one-hot representation:  [0 0 0 1 0 0 0 0 0 0]
label:  4  in one-hot representation:  [0 0 0 0 1 0 0 0 0 0]
label:  5  in one-hot representation:  [0 0 0 0 0 1 0 0 0 0]
label:  6  in one-hot representation:  [0 0 0 0 0 0 1 0 0 0]
label:  7  in one-hot representation:  [0 0 0 0 0 0 0 1 0 0]
label:  8  in one-hot representation:  [0 0 0 0 0 0 0 0 1 0]
label:  9  in one-hot representation:  [0 0 0 0 0 0 0 0 0 1]
```

Upload to moodle

- 學號_lab1.zip
  - 學號_lab1.ipynb
    - IPython notenook 須包含程式碼跟結果
  - 學號_lab1.pdf
    - 說明如何實作各個layer
    - 用自己的話說明forward / backward如何進行
    - 截圖並說明各項結果(包含accuracy和loss圖表(plot)的結果)
    - 實作過程中遇到的困難及你後來是如何解決的(optional)

END

Advisor：Tsai, Chia-Chi

雲端連結:

https://drive.google.com/drive/folders/1iaHWh042C6TRjOw4ybh6udM9JtHeQO3u?usp=sharing