國立高雄科技大學
National Kaohsiung University of Science and Technology

Deep Learning
A Neuron to Many Neurons

陳俊豪 教授

Outline

1、A Neuron & Case Study: Iris

2、 A Neuron to Many Neurons

3、Case Study: MINIST

4、Conclusions

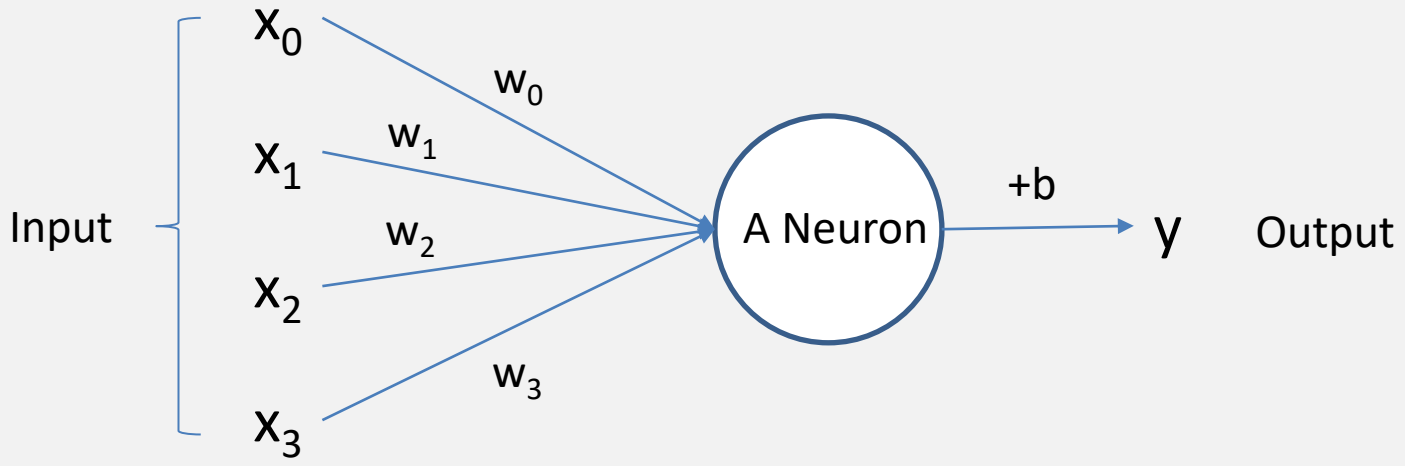
1、A Neuron & Case Study: Iris

Basic Unit of Deep Learning – A Neuron

✓A Neuron

- The smallest unit of neuron network
- Based on the input values to predict the outcome

✓ How a neuron work, e.g.

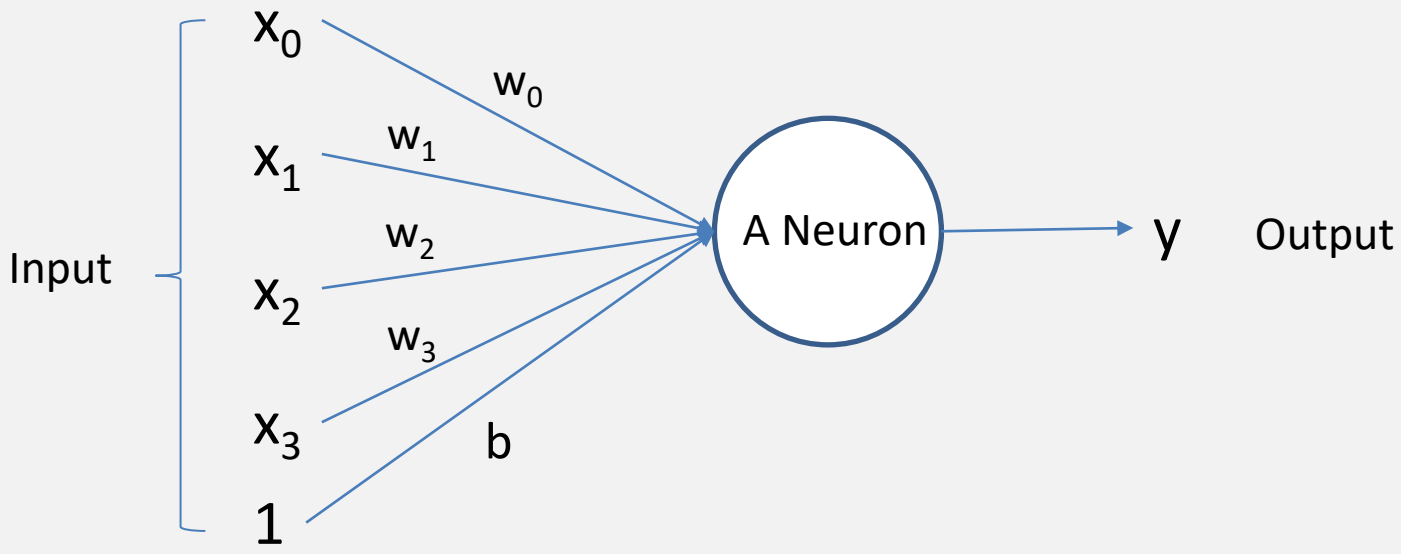


x_i: input value
w_i: weight
b: bias
y: output

$$y = x_0*w_0 + x_1*w_1 + x_2*w_2 + x_3*w_3 + b$$

Bias – Another Input Attribute

- ✓The bias can be represented as another attribute
- ✓The neuron can be re-construct as follows



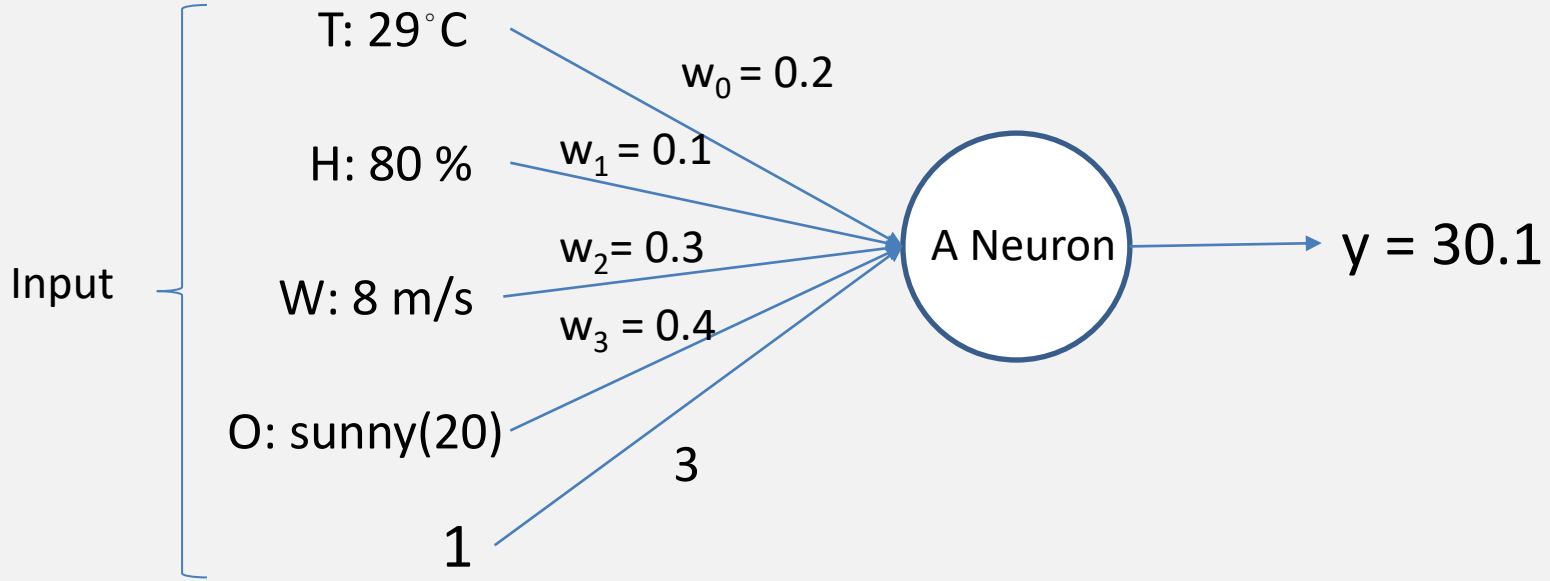
x_i: input value
w_i: weight
b: bias
y: output

$$y = x_0*w_0 + x_1*w_1 + x_2*w_2 + x_3*w_3 + b$$

An Example

✓Take “play tennis” as an example

- Input attributes: temperature (T), humidity (H), wind (W), outlook (O)
- Output target: Play tennis, Not Play tennis

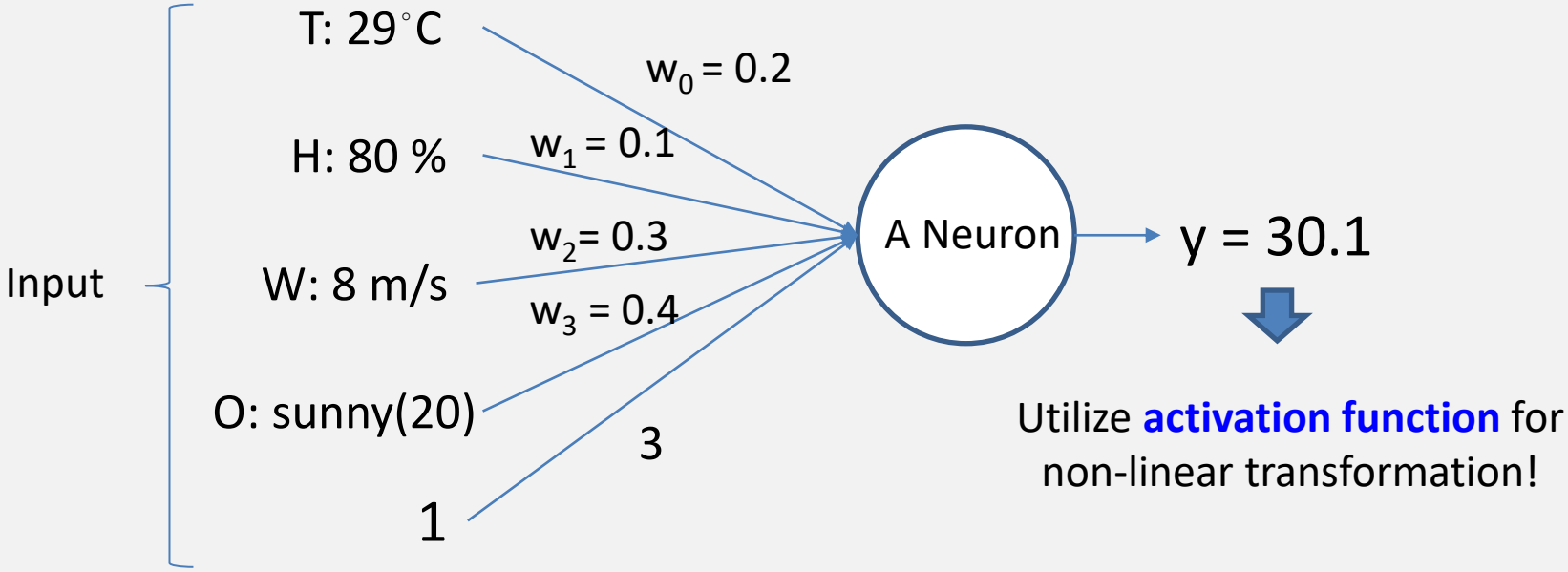


y = 29*0.3 + 80*0.1 + 8*0.3 + 20*0.4 + 1*3 = 30.1

An Example (Cont.)

✓The output value y = 30.1

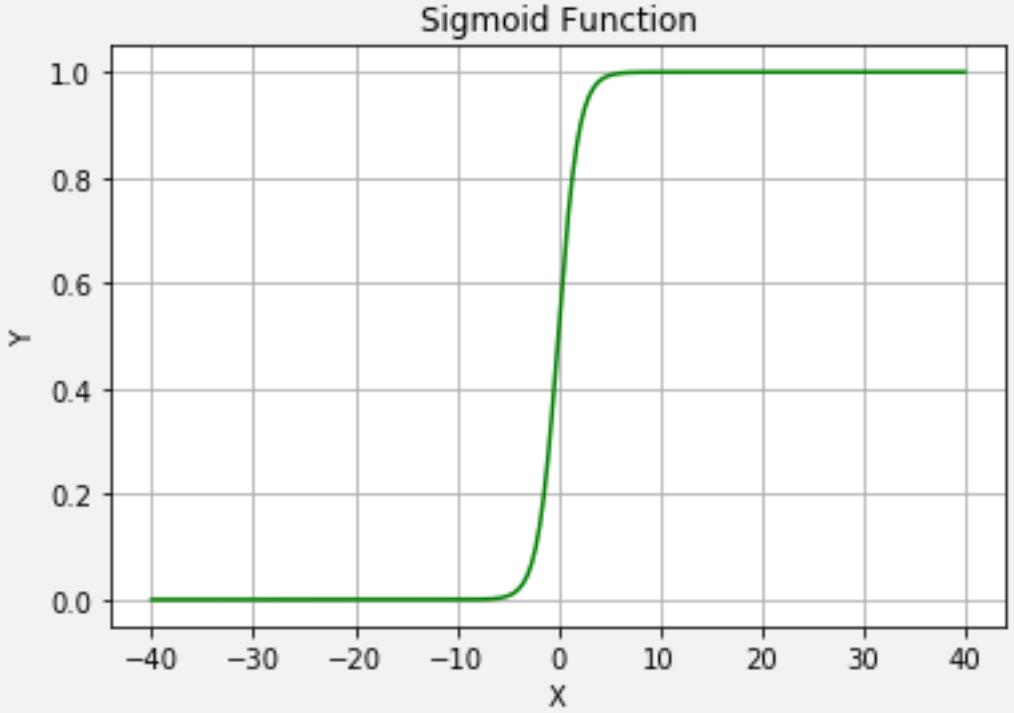
- Not easy to identify “Play Tennis” or “Not Play Tennis”



Activation Function

✓Use to enhance the learning ability for non-linear rules

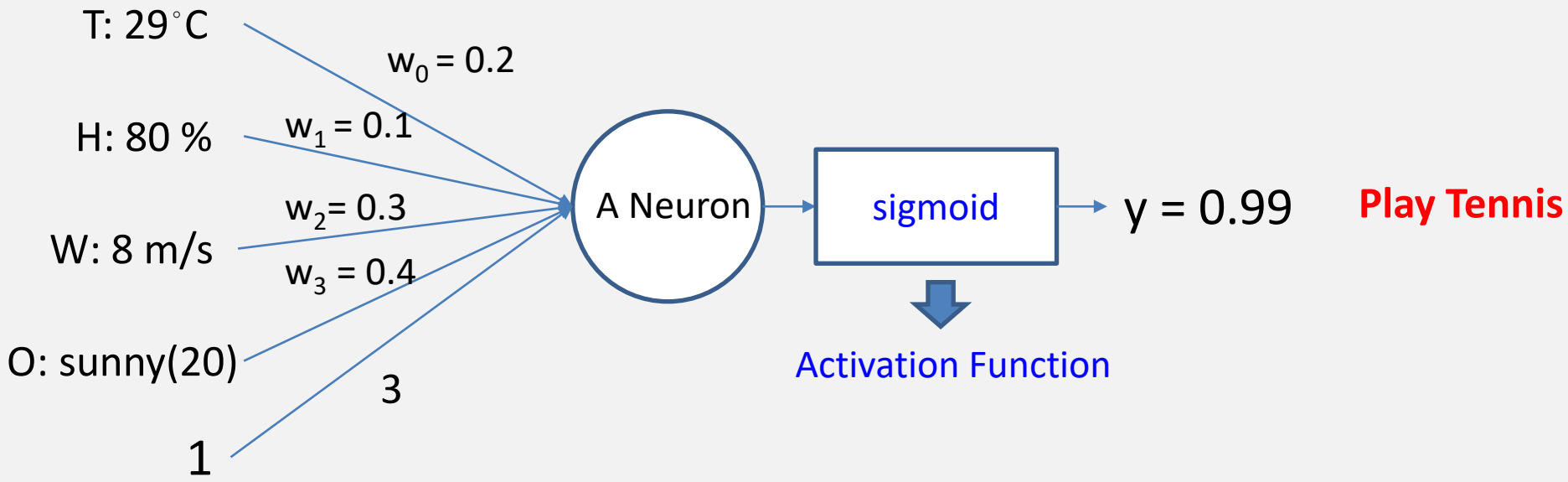
✓e.g. ➔ Sigmoid function: $sig(x) = 1/(1 + e^{-x})$



- Range of sigmoid is between [0, 1]
- e.g.,
  - sig(10) = 0.999954
  - sig(1) = 0.731
  - sig(0) = 0.5
  - sig(-1) = 0.268
  - sig(-10) = 0.000045

Consider Activation Function

✓The neuron is changed as follows



$$y = \sum_{i=0}^{n} x_i \; w_i$$

y = f_{sig}(29*0.3 + 80*0.1 + 8*0.3 + 20*0.4 + 1*3)
= f_{sig}(30.1) = 0.999999999999154

Loss Function

✓Evaluate the Goodness of A Neuron

- For every instance ➔ (A_1, A_2, …, A_n, y)
- For a neuron ➔ Based on the weights, a predicted value y’ can be genereated
- Sum of Distance between the y and y’ of all instances

✓ e.g.

- Mean square error (MSE)

$$L = \frac{1}{2N}\sum_{i=0}^{n}(y_i' - y_i\ )^2$$

An Example

- ✓Given the following 3 instances

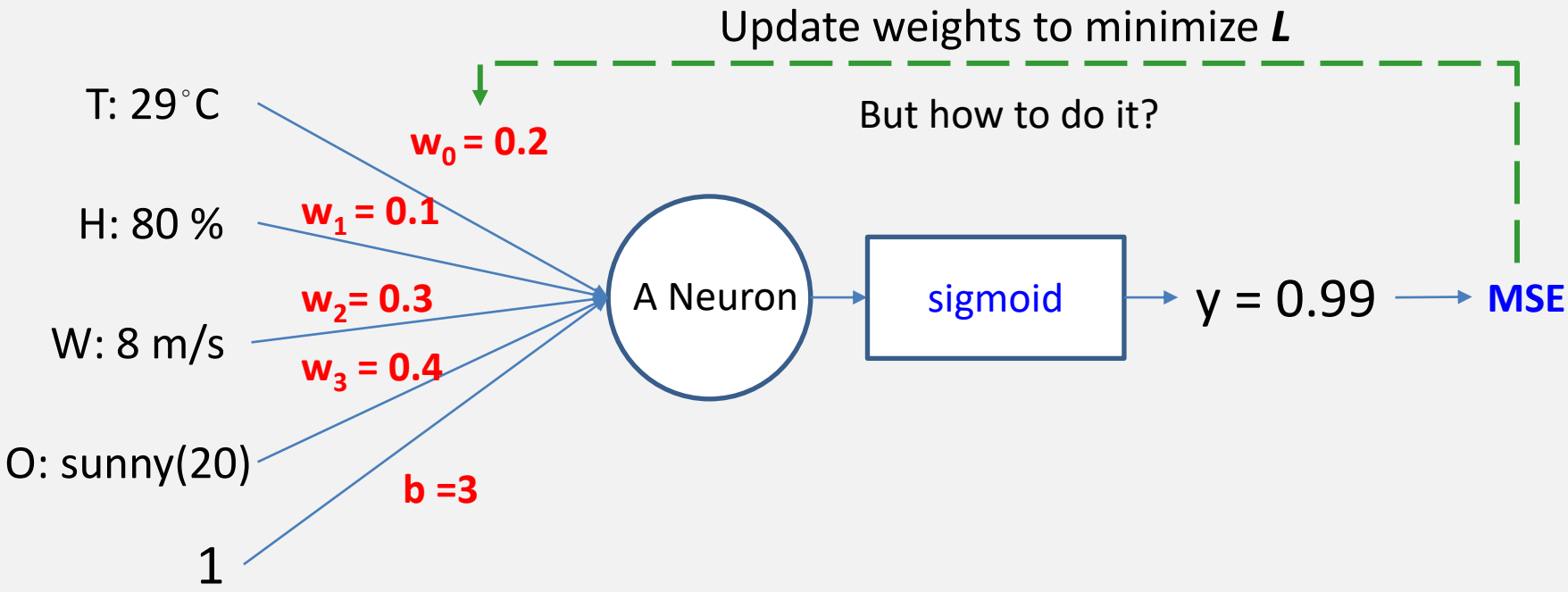
	x_0	x_1	x_2	x_3	y
$Instance_1$	5.1	3.5	1.4	0.2	0
$Instance_2$	4.9	3.0	1.4	0.2	1
$Instance_3$	4.7	3.2	1.3	0.2	0

- ✓Let weights are set at **0.1** for all attributes
- ✓MSE
  - ✓$y_0' = f_s(5.1*0.1+3.5*0.1*1.4*0.1+0.2*0.1+b) = f_s(1.12) = 0.7539$
  - ✓$y_1' = f_s(1.05) = 0.7407$
  - ✓$y_2' = f_s(1.04) = 0.7388$
  - ✓**MSE** = $[(0.7539 - 0)^2 + (0.7407 - 1)^2 + (0.7388 - 0)^2] / 2*3 =$ **0.19169**

Goal of Learning A Neuron

✓Train a good neuron means ➔ Minimize the loss function ***L***

- Obtain the **weights**: (w_0, w_1, …, w_n, b) that can **minimize** the loss function



How to Update Weights of A Neuron

- ✓ Two steps
  - Step 1: Partial derivative of the loss function ***L*** on every weight w_i
  - Step 2: Update weight using the value after partial derivative
- ✓ Take w_0 as an example

Step 1

$$L = \frac{1}{2N}\sum_{i=0}^{n}(y_i' - y_i\ )^2$$

Partial derivative of ***L*** on w_0

$$\frac{\partial L}{\partial w_0} = \frac{1}{N}\sum_{i=0}^{n}\{(y_i' - y_i\ ) * y_i' * (1- y_i') * A_i^1\}$$

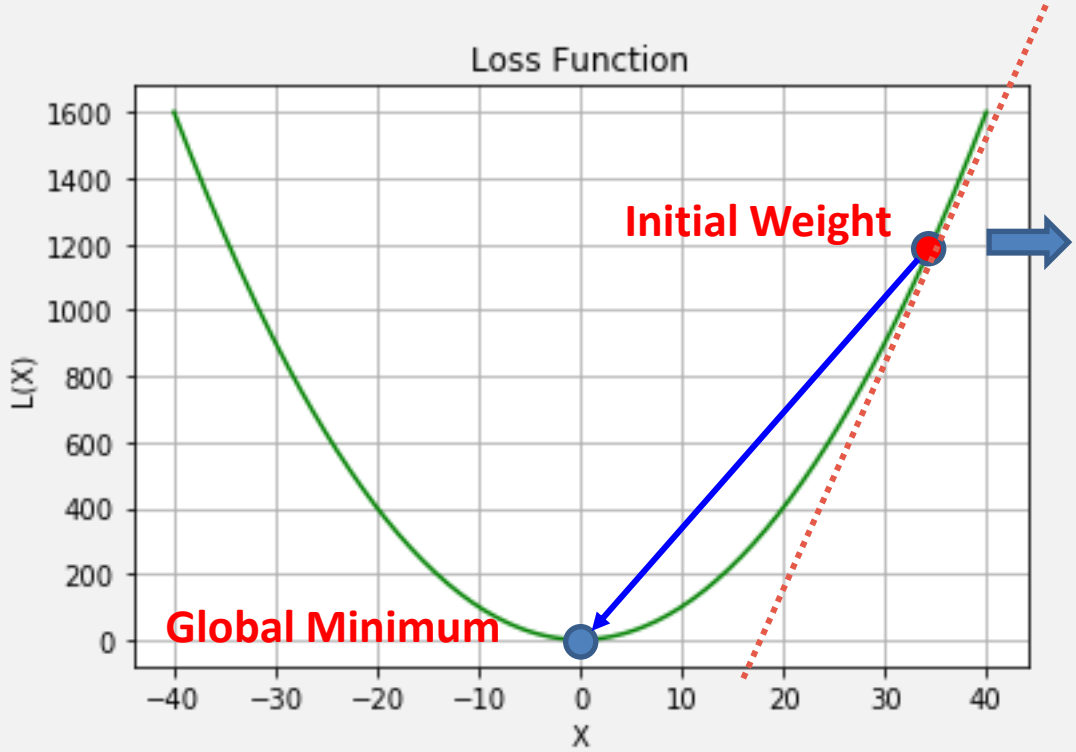
Step 2

$$\text{new } w_0 = w_0 - \eta * \frac{\partial L}{\partial w_0}$$

Learning rate (eta): [0, 1]

Physical Meaning of Partial Derivative

✓ Find the gradient (slop) of the weight $\boldsymbol{w_i}$ on the loss function L



$$gradient = \frac{\partial L}{\partial w_0}$$

Continue Previous Example

- ✓ Take updating weight w_0 as an example
  - Let learning rate η = 0.5

	x_0	x_1	x_2	x_3	y	y'
$Instance_1$	**5.1**	3.5	1.4	0.2	0	0.75
$Instance_2$	**4.9**	3.0	1.4	0.2	1	0.74
$Instance_3$	**4.7**	3.2	1.3	0.2	0	0.73

$$\frac{\partial L}{\partial w_0} = \frac{1}{N}\sum_{i=0}^{n}\{(y_i' - y_i\ ) * y_i' * (1- y_i') * A_i^0\}$$

$$= [(0.75 - 0) * 0.75 * (1 - 0.75) * 5.1 + (0.74 - 1) * 0.74 * (1 - 0.74) * 4.9 + (0.73 - 0) * 0.73 * (1 - 0.73) * 4.7] / 3 = \mathbf{0.3827}$$

new w_0 = 0.1 - 0.5 * $\frac{\partial L}{\partial w_0}$ = $0.1 - 0.5 * 0.3827 = 0.1 - 0.1913 = \mathbf{-0.0913}$

After Update All Weights

- ✓ The original neuron
  - $y' = f_s(0.1 * x_0 + 0.1 * x_1 + 0.1 * x_2 + 0.1 * x_3 + 0.1)$
- ✓ The updated neuron
  - $y' = f_s(0.0724 * x_0 + 0.795 * x_1 + 0.0952 * x_2 + 0.10008 * x_3 + 0.0943)$

From the original neuron ➔ To get the updated neuron

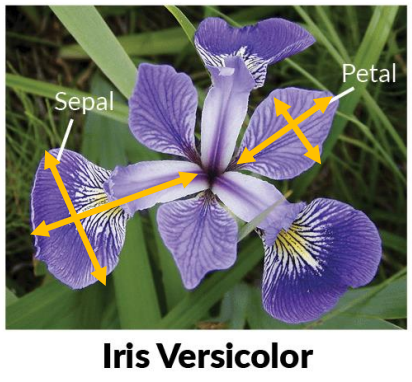
The processes call **a Epoch (週期)**

Implementation by Python

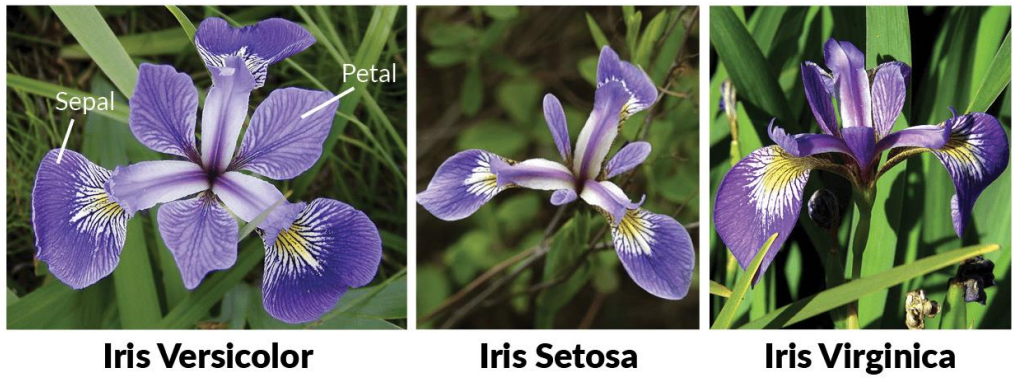
- ✓ Three steps
  - Step 1: Prepare training and testing datasets
  - Step 2: Strat to Train the neuron
  - Step 3: Test the accuracy of the neuron

Step 1: Prepare Training & Testing Datasets

- ✓ Iris dataset (https://archive.ics.uci.edu/ml/datasets/iris)
  - Four attributes ➔ Length and width of sepals (花萼) and petals(花瓣)
  - Three classes
    - ✓ Iris versicolor(雜色鳶尾), Iris setosa(山鳶尾), Iris virginica(維吉尼亞鳶尾)
  - Use 100 instances contain two classes in this example



Three classes



http://www.lac.inpe.br/~rafael.santos/Docs/CAP394/WholeStory-Iris.html

Read Irsi Datasets

1. import numpy as np
2. import pandas as pd
3. df = pd.read_csv('iris.data', header=None) # use padas
4. print(df) # The dataset contains 150 instances

	0	1	2	3	4
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa
5	5.4	3.9	1.7	0.4	Iris-setosa
6	4.6	3.4	1.4	0.3	Iris-setosa
7	5.0	3.4	1.5	0.2	Iris-setosa
8	4.4	2.9	1.4	0.2	Iris-setosa
9	4.9	3.1	1.5	0.1	Iris-setosa
10	5.4	3.7	1.5	0.2	Iris-setosa
11	4.8	3.4	1.6	0.2	Iris-setosa
12	4.8	3.0	1.4	0.1	Iris-setosa
13	4.3	3.0	1.1	0.1	Iris-setosa
14	5.8	4.0	1.2	0.2	Iris-setosa
15	5.7	4.4	1.5	0.4	Iris-setosa
16	5.4	3.9	1.3	0.4	Iris-setosa
17	5.1	3.5	1.4	0.3	Iris-setosa
18	5.7	3.8	1.7	0.3	Iris-setosa
19	5.1	3.8	1.5	0.3	Iris-setosa

125	7.2	3.2	6.0	1.8	Iris-virginica
126	6.2	2.8	4.8	1.8	Iris-virginica
127	6.1	3.0	4.9	1.8	Iris-virginica
128	6.4	2.8	5.6	2.1	Iris-virginica
129	7.2	3.0	5.8	1.6	Iris-virginica
130	7.4	2.8	6.1	1.9	Iris-virginica
131	7.9	3.8	6.4	2.0	Iris-virginica
132	6.4	2.8	5.6	2.2	Iris-virginica
133	6.3	2.8	5.1	1.5	Iris-virginica
134	6.1	2.6	5.6	1.4	Iris-virginica
135	7.7	3.0	6.1	2.3	Iris-virginica
136	6.3	3.4	5.6	2.4	Iris-virginica
137	6.4	3.1	5.5	1.8	Iris-virginica
138	6.0	3.0	4.8	1.8	Iris-virginica
139	6.9	3.1	5.4	2.1	Iris-virginica
140	6.7	3.1	5.6	2.4	Iris-virginica
141	6.9	3.1	5.1	2.3	Iris-virginica
142	5.8	2.7	5.1	1.9	Iris-virginica
143	6.8	3.2	5.9	2.3	Iris-virginica
144	6.7	3.3	5.7	2.5	Iris-virginica
145	6.7	3.0	5.2	2.3	Iris-virginica
146	6.3	2.5	5.0	1.9	Iris-virginica
147	6.5	3.0	5.2	2.0	Iris-virginica
148	6.2	3.4	5.4	2.3	Iris-virginica
149	5.9	3.0	5.1	1.8	Iris-virginica

[150 rows x 5 columns]

Data Processing on the Irsi Datasets

1. import numpy as np
2. import pandas as pd
3. df = pd.read_csv('iris.data', header=None)
4. x = df.iloc[0:100,[0, 1, 2, 3]].values # iloc() use to determine selected range
5. y = df.iloc[0:100,4].values # .values() use to get attribute values
6. y = np.where(y=='Iris-setosa', 0, 1) # if 'Iris-setosa’ => 0, else 1

print(x)

```
[[ 5.1  3.5  1.4  0.2]
 [ 4.9  3.   1.4  0.2]
 [ 4.7  3.2  1.3  0.2]
 [ 4.6  3.1  1.5  0.2]
 [ 5.   3.6  1.4  0.2]
 [ 5.4  3.9  1.7  0.4]
 [ 4.6  3.4  1.4  0.3]
 [ 5.   3.4  1.5  0.2]
 [ 4.4  2.9  1.4  0.2]
 [ 4.9  3.1  1.5  0.1]
 [ 5.4  3.7  1.5  0.2]
 [ 4.8  3.4  1.6  0.2]
 [ 4.8  3.   1.4  0.1]
```

```
['Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-setosa' 'Iris-setosa' 'Iris-setosa' 'Iris-setosa
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor' 'Iris-versicolor'
 'Iris-versicolor' 'Iris-versicolor']
```

print(y)//line 5

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
```

print(y)//line 6

After Data Processing

- ✓ The extracted 100 instances
  - x = df.iloc[0:100,[0, 1, 2, 3]].values # iloc() use to determine selected range
  - y = df.iloc[0:100,4].values # .values() use to get attribute values

	X (Features)	Y (Classes)	
Class = 0	50 instances	50 instances	Number 0 to 49
Class = 1	50 instances	50 Instances	Number 50 to 99

Divide Dataset Into Training and Testing

- ✓ How to get suitable training and testing datasets
- ✓ e.g.
  - Let the proportion of training and testing is 8:2

	X (Features)	Y (Classes)
C = 0	50 instances	50 instances
C = 1	50 instances	50 Instances

80 instances for training
20 instances for testing

	X	Y
0~39	40	40
40~49	10	10
50~89	40	40
90~99	10	10

x_train 80 instances

y_train 80 instances

x_test 20 instances

y_test 20 instances

Using Python to Get Training and Testing

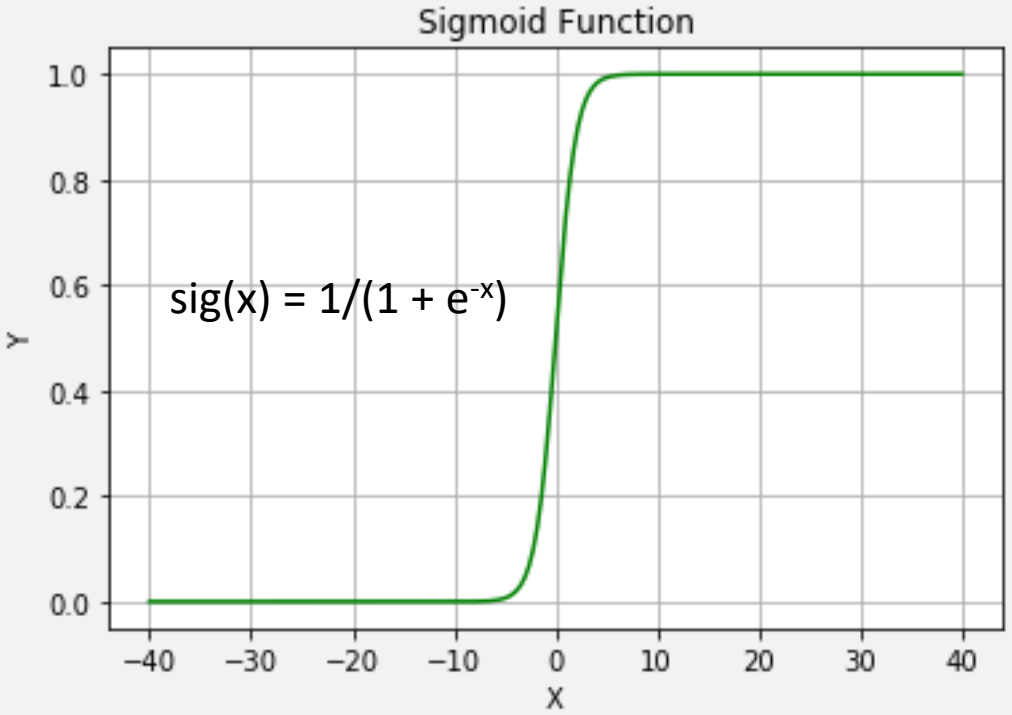
```
#prepare empty arrays
x_train = np.empty((80, 4))# size: 80*4 to store attribute values
x_test = np.empty((20, 4)) # size: 20*4
y_train = np.empty(80) # size: 80*1 to store class values
y_test = np.empty(20) # size: 20*1

#Divide 100 instances into 80 instances for training and 20 for testing
x_train[:40], x_train[40:] = x[:40], x[50:90] # 0~39 -> C=1 & 50~89 -> C=2
x_test[:10], x_test[10:] = x[40:50],x[90:100]
y_train[:40], y_train[40:] = y[:40],y[50:90]
y_test[:10], y_test[10:] = y[40:50],y[90:100]
```

Step 2: Strat to Train the neuron network

- ✓ Prepare the needed functions
  - 1. Sigmoid function
  - 2. Activation function
  - 3. Updating weight function
- ✓ Start to train the neuron

1. Define the Sigmoid function

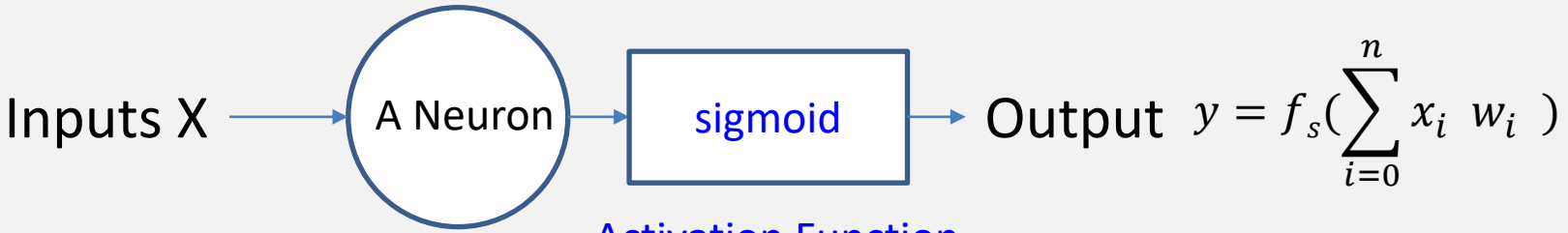
✓ Sigmoid function is defined as follows




```
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

2. Define the Activation function

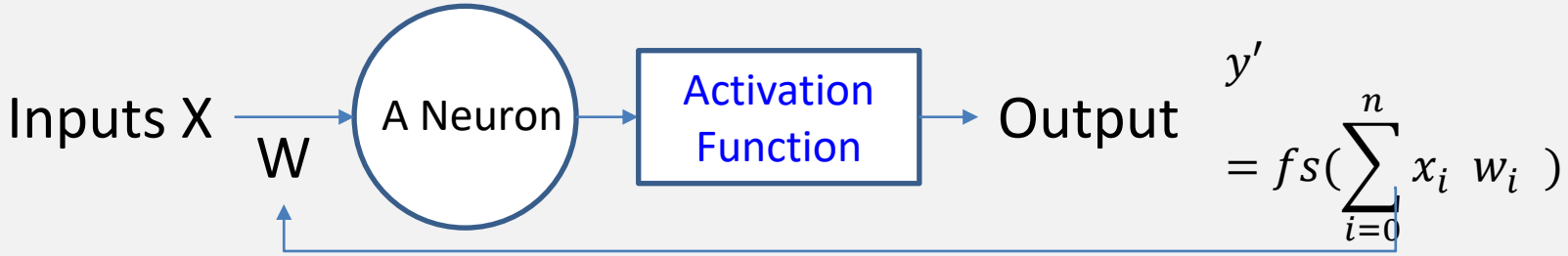
✓ Activation function is defined as follows




```
def activation(x, w, b):
    return sigmoid(np.dot(x, w)+b)
```

3. Updating Weight function

✓ Updating weight function is defined as follows




```
def update(x, y_train, w, b, eta):
    y_pred = activation(x, w, b)
    a = (y_pred - y_train) * y_pred * (1- y_pred)
    for i in range(4):
        w[i] = w[i] - eta * 1/float(len(y)) * np.sum(a*x[:,i])
    b -= eta * 1/float(len(y))*np.sum(a)
    return w, b
```

Step 2: Start to Train the Neuron Network

✓ Let epoch = 15, learning rate = 0.1, and initial weights are 0.1

```
1. weights = np.ones(4)/10  #Initialize weights
2. bias = np.ones(1)/10 #Initialize bias
3. eta=0.1 #Set learning rate
4. for _ in range(15): #epoch = 15
5.         weights, bias = update(x_train, y_train, weights, bias, eta=0.1)
6. print('weights = ', weights, 'bias = ', bias)
```

```
weights = [-0.1490311 -0.38795727 0.64200298 0.34623798]
bias = [ 0.00359432]
```

Step 3: Test the Trained Neuron (epoch=15)

- ✓ True 20 Test data class: y_test
  - array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    1., 1., 1.,1., 1., 1., 1., 1., 1., 1.])

Different of them are large

- ✓ Use **x_test, weights, bias** as parameters for activation function
  - Predicted y values
    - ✓ array([0.46589332, 0.48973998, 0.47429981, 0.4886412 , 0.48846777,
      0.47984259, 0.46916786, 0.47593903, 0.46403066, 0.46949766,
      0.62927546, 0.63098025, 0.61014931, 0.58951514, 0.62221951,
      0.61360253, 0.61827101, 0.61628812, 0.57673708, 0.61575116])

Step 3: Test the Trained Neuron (Epoch = 999)

- ✓ True Test data: y_test
  - array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
    1., 1., 1.,1., 1., 1., 1., 1., 1., 1.])

They are similar now

- ✓ Use x_test, weights, bias as parameters for activation function
  - Predicted y values
    - array([0.05395003, 0.17599855, 0.07863967, 0.10059897, 0.0991089
      0.10288262, 0.05693354, 0.0853523 , 0.05103666, 0.06962544,
      0.96078244, 0.95579566, 0.92428386, 0.86254368, 0.94491158,
      0.92028125, 0.93194228, 0.93172487, 0.77415845, 0.92844899])

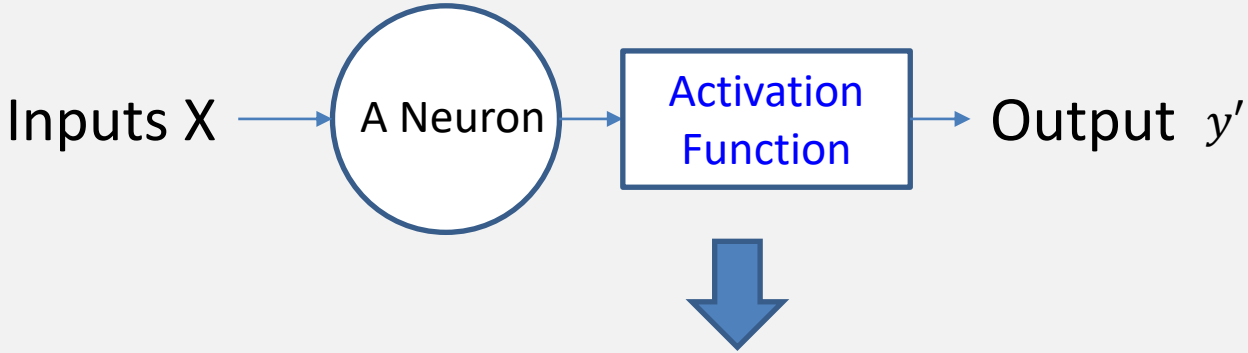
Recap

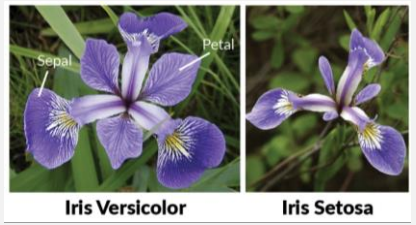
- ✓ So far, you need to know the following terms
  - A neuron
  - Training dataset
  - Testing dataset
  - Weights
  - Activation function
  - Loss function
  - Updating weight function

2、A Neuron to Many Neurons

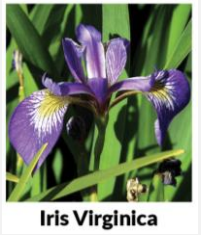
Question

- ✓ In the previous slides
  - The neuron ➔ Identify two classes (Binary Classification)





If the dataset contains **many classes**, what can we suppose to do to solve the problem? e.g., the iris dataset contains three classes.

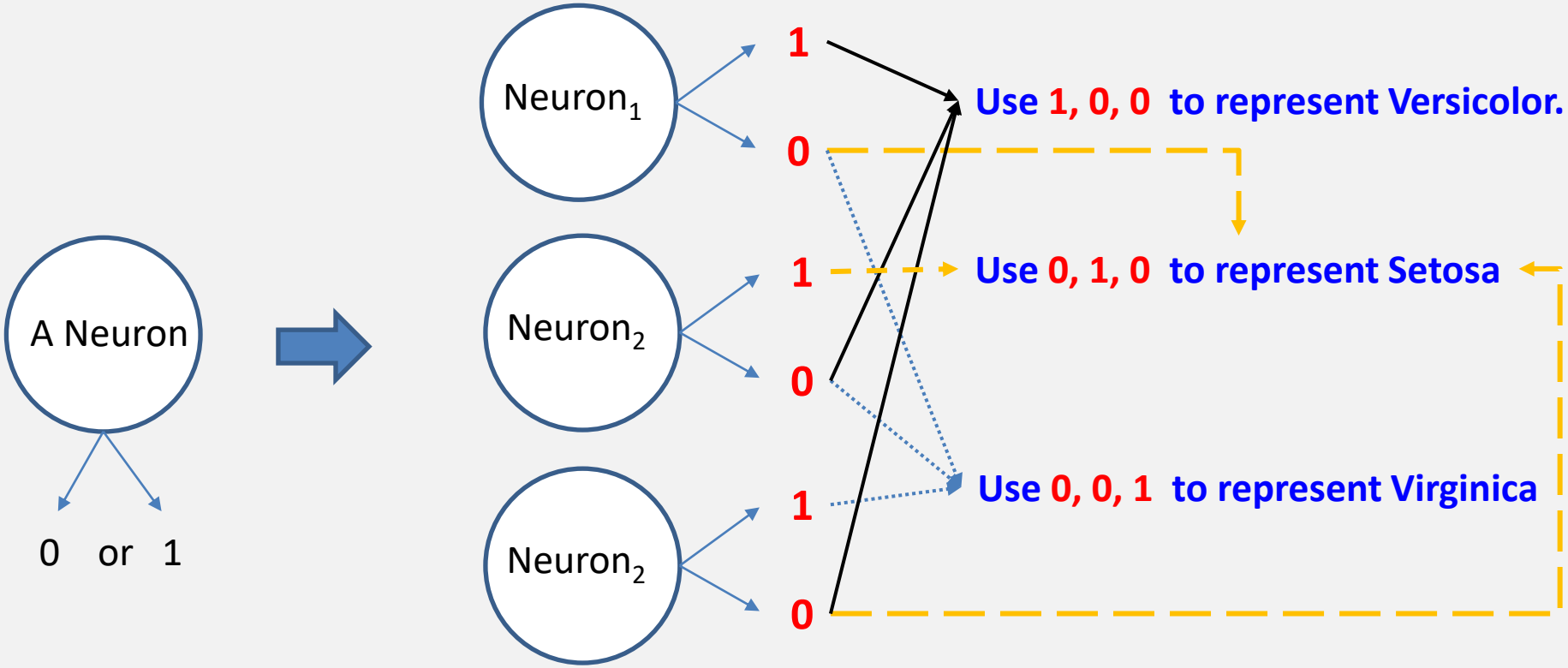


Solution for Handling Multiple Classes

- ✓Step 1: Add neurons
- ✓Step 2: Add layers

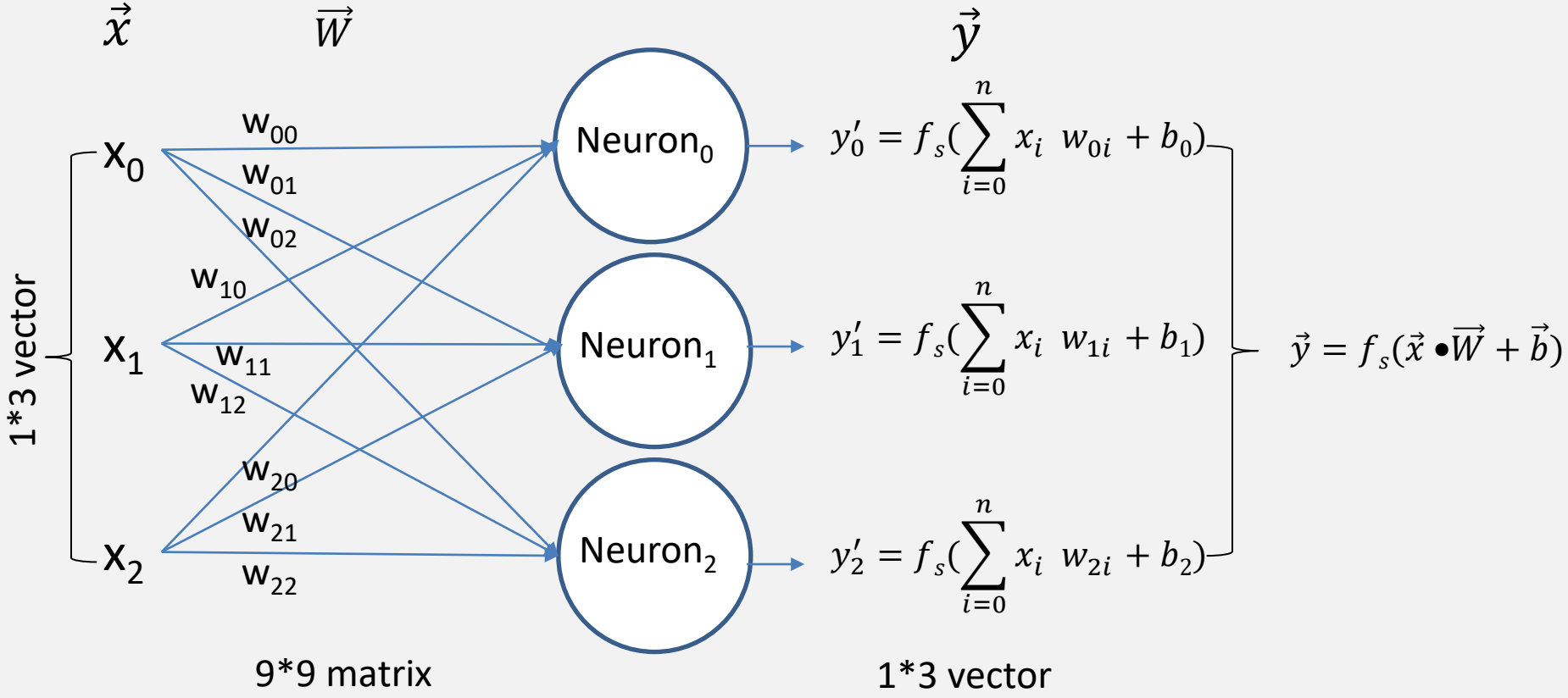
Step 1: Add Neurons

✓From a neuron to three neurons



Changes of the Neuron Structure

✓Let the number of neurons is 3, input attributes is 3, and 3 classes

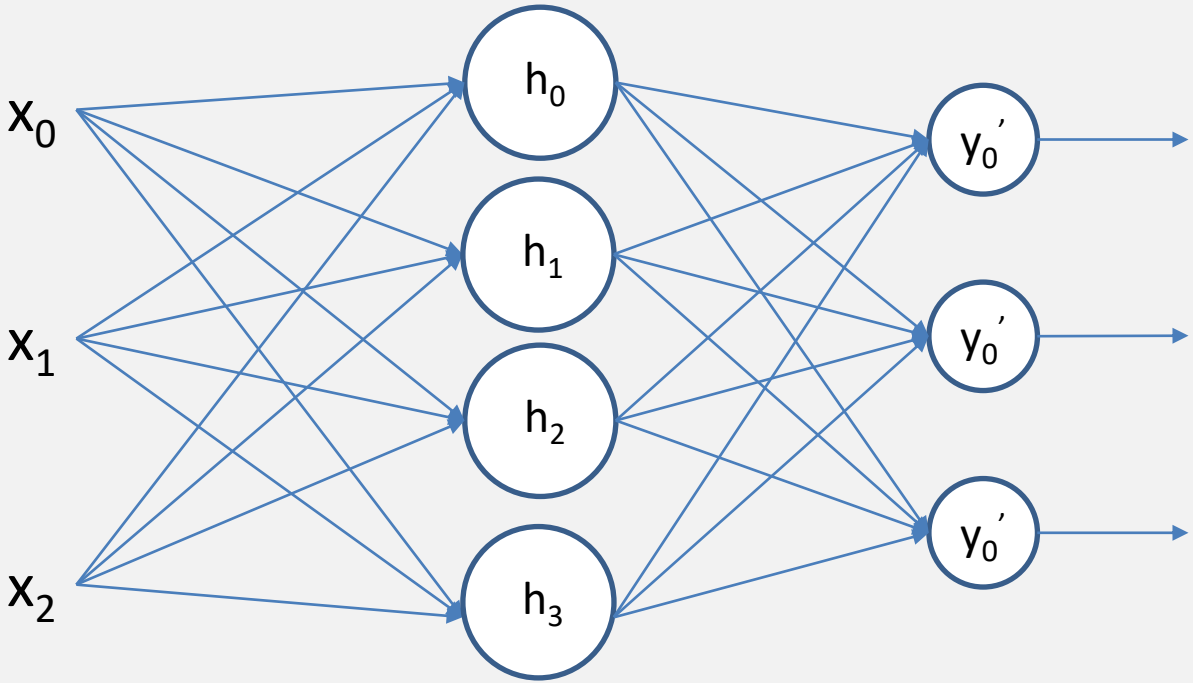


Step 2: Add Layers

✓Between input and output ➔ Add hidden layer (隱藏層)

✓e.g., add a hidden layer with 4 neurons

- Denoted as $\vec{h} = \{h_0, h_1, h_2, h_3\}$



$$\vec{y} = f_s(\vec{h} \bullet W'' + \vec{b''})$$

$$\vec{h} = f_s(\vec{x} \bullet W' + \vec{b'})$$

The Constructed Neuron Network

✓Consist of three layers

- Input layer, Hidden layer and Output layer

Input layer to hidden layer

$$\vec{y} = f_s(\overbrace{f_s(\vec{x} \bullet W' + \vec{b'})}^{} \bullet W'' + \vec{b''})$$

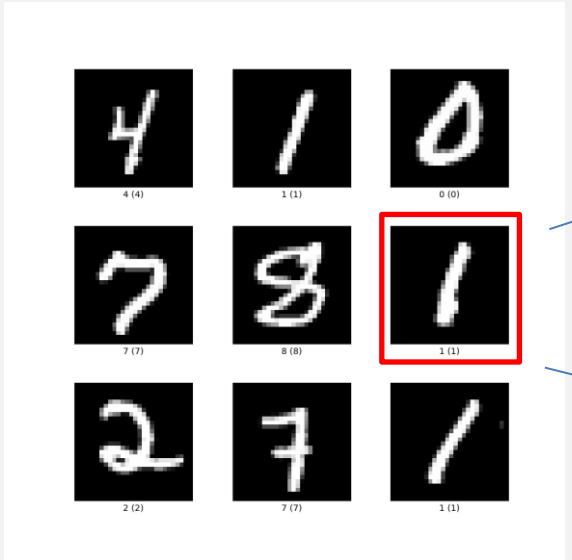
Hidden layer to output layer

3、Case Study: MINIST

An Example - MNIST

✓MNIST(**M**ixed **N**ational **I**nstitute of **S**tandards and **T**echnology dataset)

- Can be download at http://yann.lecun.com/exdb/mnist/
- Handwritten digits for number 0 to 10, and totally 70,000 images
- Size of each image is 28*28 pixels with correct answer



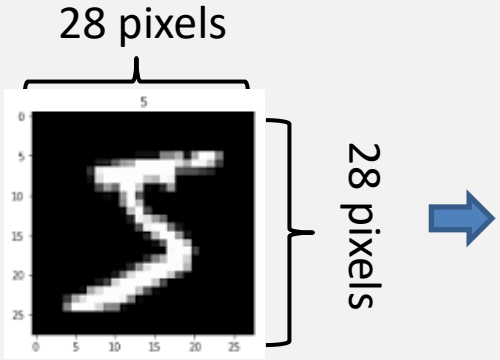
- A image has **784** numbers (attributes)
- Every number is between 0~255

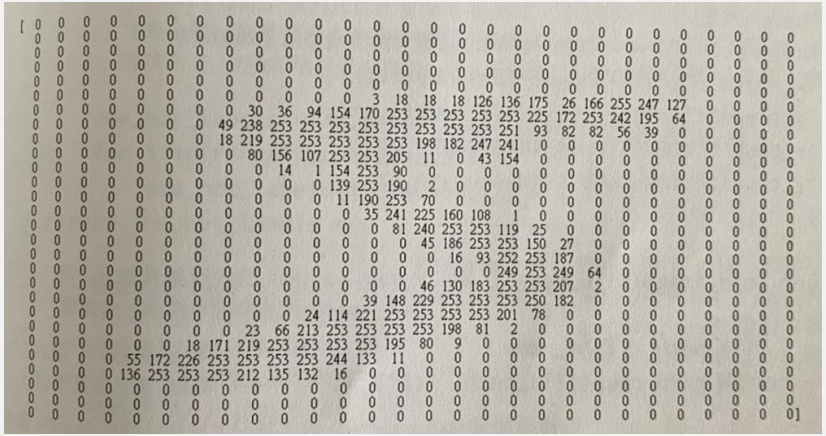
https://www.tensorflow.org/datasets/catalog/mnist

A Image in MNIST Dataset

✓Take image of digit ‘5’ as an example

- 28 * 28 pixels, and value of a pixel is between [0, 255]
- Totally 748 pixels





Dataset Preparation

✓Download the MNIST dataset

- Already divided into four parts: x_train, y_train, x_test, y_test
- Use 60,000 images for training, and 10,000 images for testing

THE MNIST DATABASE

of handwritten digits

Yann LeCun, Courant Institute, NYU
Corinna Cortes, Google Labs, New York
Christopher J.C. Burges, Microsoft Research, Redmond

The MNIST database of handwritten digits, available from this page, has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.

It is a good database for people who want to try learning techniques and pattern recognition methods on real-world data while spending minimal efforts on preprocessing and formatting.

Four files are available on this site:

```
train-images-idx3-ubyte.gz:  training set images (9912422 bytes)
train-labels-idx1-ubyte.gz:  training set labels (28881 bytes)
t10k-images-idx3-ubyte.gz:   test set images (1648877 bytes)
t10k-labels-idx1-ubyte.gz:   test set labels (4542 bytes)
```

Download these four files

Training and Testing Datasets

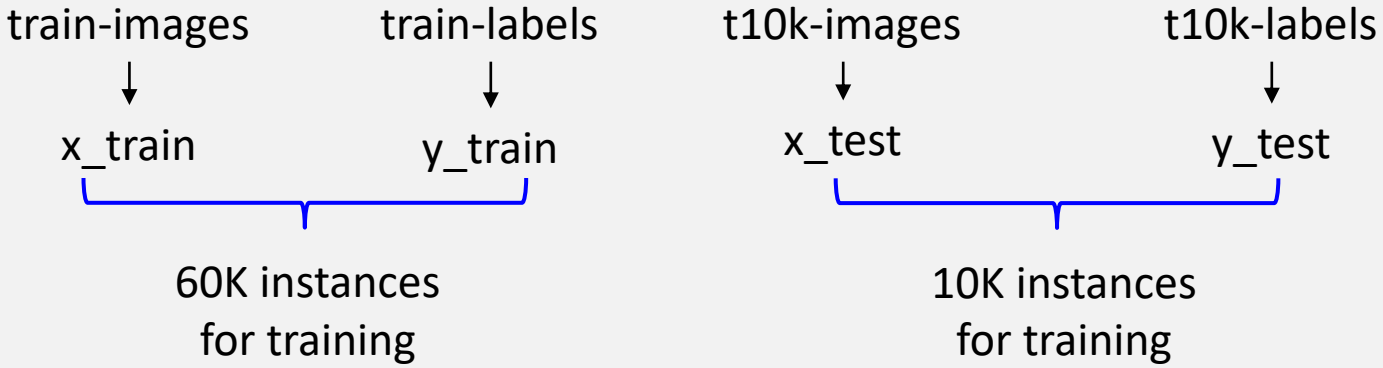
✓The four files

- train-images-idx3-ubyte.gz: training set images (9912422 bytes)
- train-labels-idx1-ubyte.gz: training set labels (28881 bytes)

60000 instances

- t10k-images-idx3-ubyte.gz: test set images (1648877 bytes)
- t10k-labels-idx1-ubyte.gz: test set labels (4542 bytes)

10000 instances



After Download the .gz files

✓Put the four .gz files and the load_mnist.py in the same folder

```
1. #Python code
2. import load_mnist as lm
3. import gzip
4. dataset = lm.load_mnist()
5.     #A dictionary four keys: x_test, x_train, y_test, y_train
6. dataset
```

Result of the “dataset” Dictionary

✓The dataset dictionary contain four pairs

- {'x_test': array(), 'x_train': array(), 'y_test': array(), 'y_train': array() }

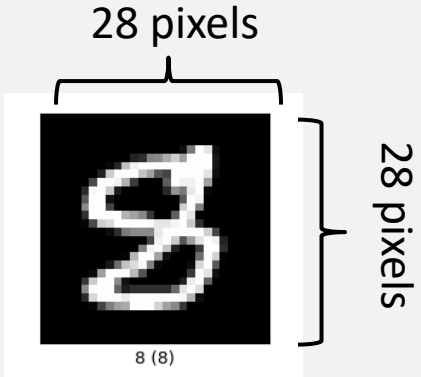
748

'x_train': array([[0., 0., 0., ..., 0., 0., 0.],
[0., 0., 0., ..., 0., 0., 0.],
...,
[0., 0., 0., ..., 0., 0., 0.]],
dtype=float32),

6000

y_train': array([[0., 0., 0., ..., 0., 0., 0.],
[1., 0., 0., ..., 0., 0., 0.],
...,
[0., 0., 0., ..., 0., **1**., 0.]])

Last image



Digit of the last image is ‘8’

Details of the load_mnist.py

✓Contain a key_file dictionary and four functions

```
1.  import numpy as np
2.  import gzip
3.  key_file = {
4.  'x_train':'train-images-idx3-ubyte.gz',
5.  'y_train':'train-labels-idx1-ubyte.gz',
6.  'x_test':'t10k-images-idx3-ubyte.gz',
7.  'y_test':'t10k-labels-idx1-ubyte.gz'
8.  }
9.  def load_image(file_name):
10.     file_path = file_name
11.     with gzip.open(file_path, 'rb') as f:
12.         images = np.frombuffer(f.read(), np.uint8, offset=16)
13.         return images
```

Provide us to open and read .gzip file directly

Dictionary: use to store the file names

Read from the 16th byte

Return a 1D array with length 47,040,000 (= 784 * 60K)

Details of the load_mnist.py (Cont.)

```
14. def load_label(file_name):
15.     file_path = file_name
16.     with gzip.open(file_path, 'rb') as f:
17.         labels = np.frombuffer(f.read(), np.uint8, offset=8)
18.         one_hot_labels = np.zeros((labels.shape[0], 10))
19.         for i in range(labels.shape[0]):
20.             one_hot_labels[i, labels[i]] = 1
21.     return one_hot_labels
22. def convert_into_numpy(key_file):
23.     dataset = {}
24.     dataset['x_train'] = load_image(key_file['x_train'])
25.     dataset['y_train'] = load_label(key_file['y_train'])
26.     dataset['x_test'] = load_image(key_file['x_test'])
27.     dataset['y_test'] = load_label(key_file['y_test'])
28.     return dataset
```

Read from the 8th byte

an array that has 60,000 or 10,000 digits

One-hot encoding

Return a 2D array with length 60,000*10 or 10,000*10

Call load_image() and load_label() to store Image from .gz files to dataset (a dictionary)

Details of the load_mnist.py (Cont.)

```
29. def load_mnist():
30.   dataset = convert_into_numpy(key_file)
31.   dataset['x_train'] = dataset['x_train'].astype(np.float32)
32.   dataset['x_test'] = dataset['x_test'].astype(np.float32)
33.   dataset['x_train'] /= 255.0
34.   dataset['x_test'] /= 255.0
35.   dataset['x_train'] = dataset['x_train'].reshape(-1, 28*28)
36.   dataset['x_test'] = dataset['x_test'].reshape(-1, 28*28)
37.   return dataset
```

Set type from uint8 to np.float32

Reshape the 1D array (47,040,000 = 784 * 60K) to 2D array (60K * 784)

Reshape the 1D array (47,040,000 = 784 * 60K) to 2D array (10K * 784)

Question

✓Now, we can use x_train and y_train to learn a neuron network for the MINST dataset

Input layer to hidden layer

$$\vec{y} = f_s(\underbrace{f_s(\vec{x} \bullet W' + \overrightarrow{b'})}_{} \bullet W'' + \overrightarrow{b''})$$

Hidden layer to output layer

How many parameters should be learned?

Learn A Network for MNIST Dataset

✓The neuron network has three layers

- Input layer: 60K instances, each one has 784 attributes
- Hidden layer: 100 neurons
- Output layer: 10 neurons (since the number of classes is 10)

Input layer to hidden layer: $784 \times 100 + 100 = 78500$

$$\vec{y} = f_s(f_s(\vec{x} \bullet W' + \overrightarrow{b'}) \bullet W'' + \overrightarrow{b''})$$

Hidden layer to output layer: $100 \times 10 + 10 = 1010$

Totally, 79510 parameters

Goal: Minimize the Loss Function

- ✓Previously the loss function is MSE
- ✓Update the weight for a neuron by 2 steps

Step 1

$$L = \frac{1}{2N}\sum_{i=0}^{n}(y_i' - y_i\ )^2$$

Partial derivative of ***L*** on w_0

$$\frac{\partial L}{\partial w_0} = \frac{1}{N}\sum_{i=0}^{n}\{(y_i' - y_i\ ) * y_i' * (1- y_i') * A_i^1\}$$

Step 2

$$\text{new } w_0 = w_0 - \eta * \frac{\partial L}{\partial w_0}$$

Learning rate (eta): [0, 1]

79510 parameters should be learned and how to learn it effectively is an import issue

By back propagation (反向傳遞)

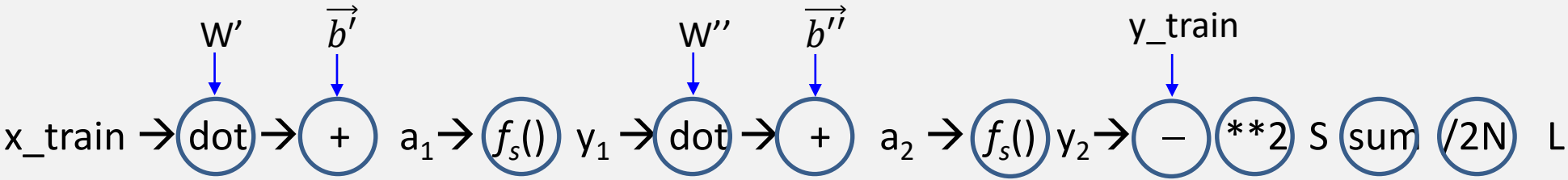
What is Forward Propagation?

✓Computational graph of the loss function

$$\vec{y} = f_s(f_s(\vec{x} \bullet W' + \vec{b'}) \bullet W'' + \vec{b''})$$

y_2 = $f_s(y_1 \bullet W'' + \vec{b''})$, y_1 = $f_s(a_1 \bullet W'' + \vec{b''})$, a_1 = $\vec{x} \bullet W' + \vec{b'}$, a_2 = $y_1 \bullet W'' + \vec{b''}$

$$L = \frac{1}{2N}\sum_{i=0}^{n}(y_i' - y_i\ )^2$$

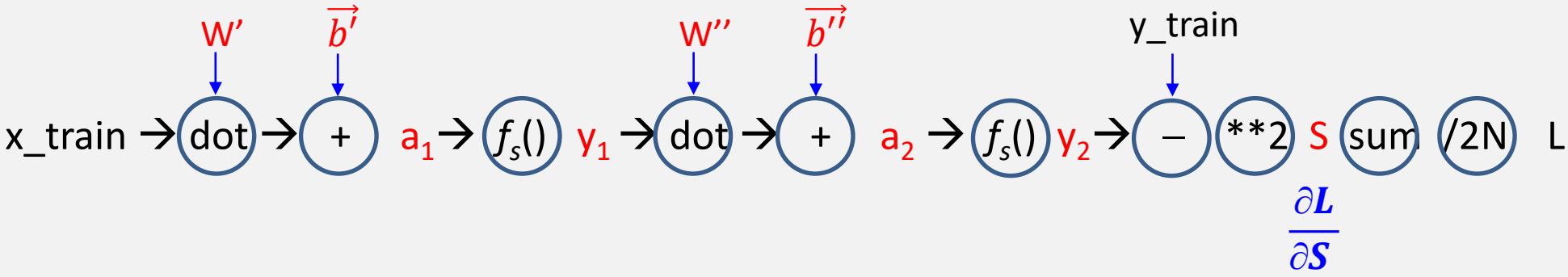


The computational graph shows the forward propagation

9 Variables in the Computational Graph

✓To know an impact of a variable to the loss function

- Use partial derivative
- e.g., to find the impact of variable S on ***L*** ➔ Calculate $\frac{\partial L}{\partial S}$ → Simple



But, how to know the impact of other variables to the Loss value, e.g., y_2 ?

Back propagation

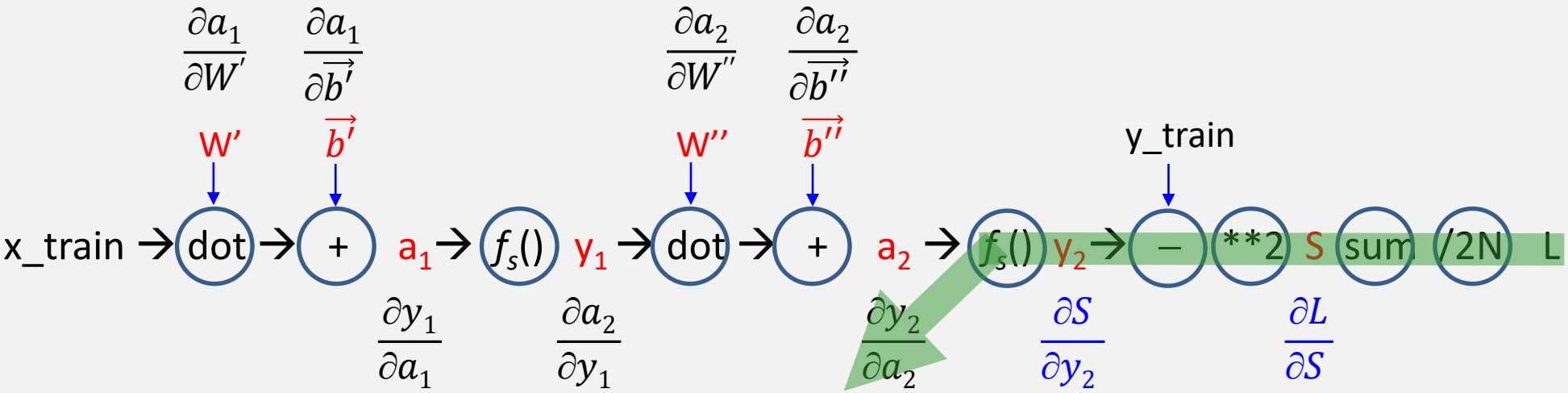
Concept of Back Propagation

✓Every variable only need to focus on the nearest computational result to find its impact on the loss function ***L***

- To find the impact of variable S no ***L*** ➔ Calculate $\frac{\partial L}{\partial S}$
- To find the impact of variable y_2 no ***L*** ➔ Calculate $\frac{\partial S}{\partial y_2}$ → $\frac{\partial L}{\partial y_2} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial y_2}$

Chain rule (連鎖法則) ➡ $\frac{\partial L}{\partial y_2} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial y_2}$

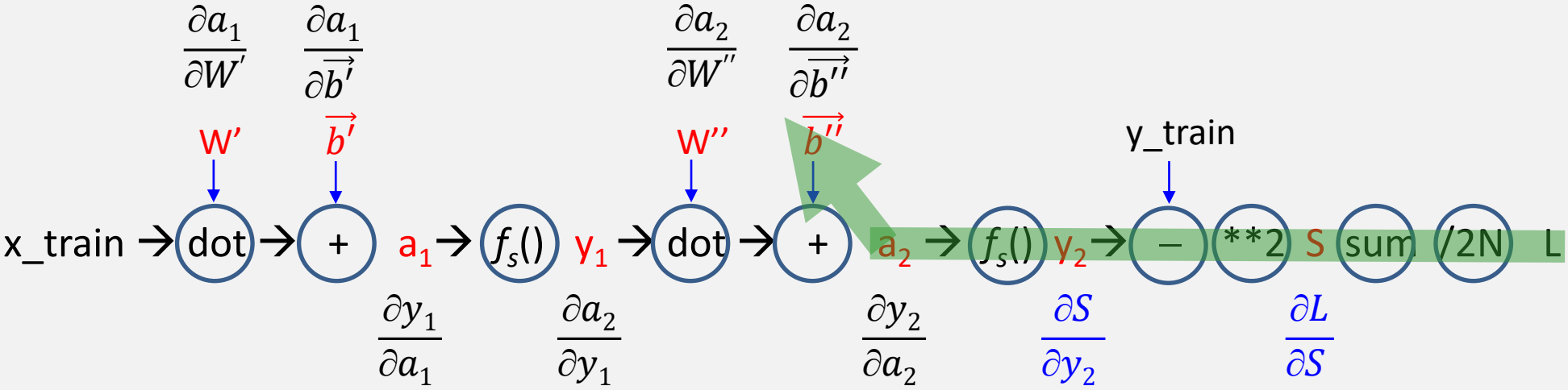
Partial Derivative of L on a_2 ($\frac{\partial L}{\partial a_2}$)



$$\frac{\partial L}{\partial a_2} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial y_2} * \frac{\partial y_2}{\partial a_2}$$

Already know

Partial Derivative of L on $\overrightarrow{b''}$ $(\frac{\partial L}{\partial \overrightarrow{b''}})$



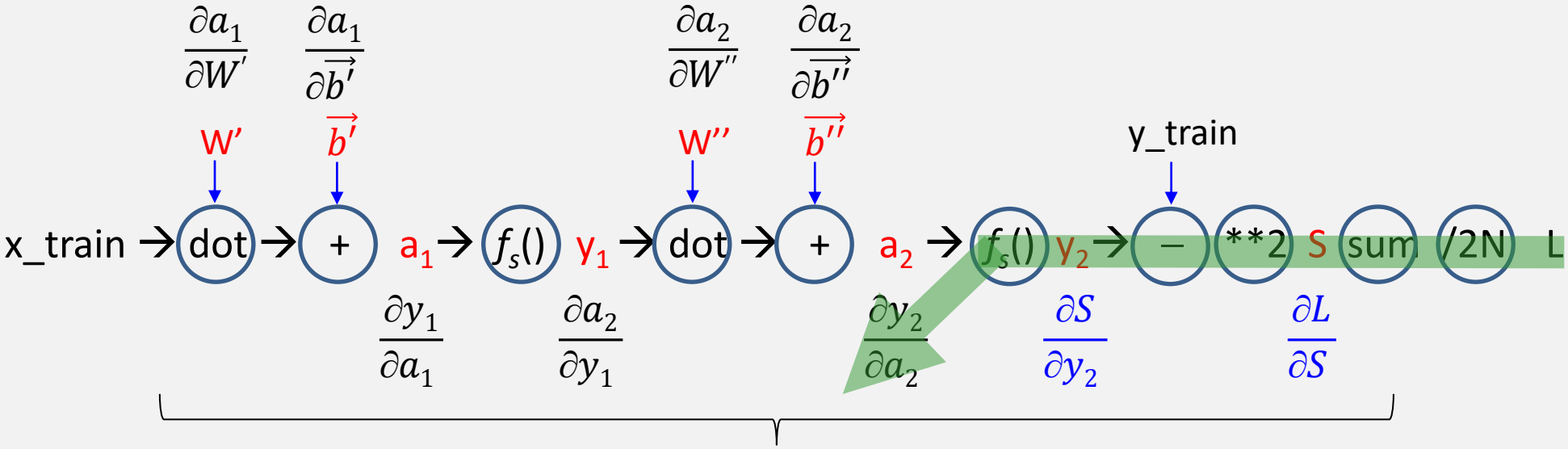
$$\frac{\partial L}{\partial \overrightarrow{b''}} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial y_2} * \frac{\partial y_2}{\partial a_2} * \frac{\partial a_2}{\partial \overrightarrow{b''}}$$

Already know

Back Propagation

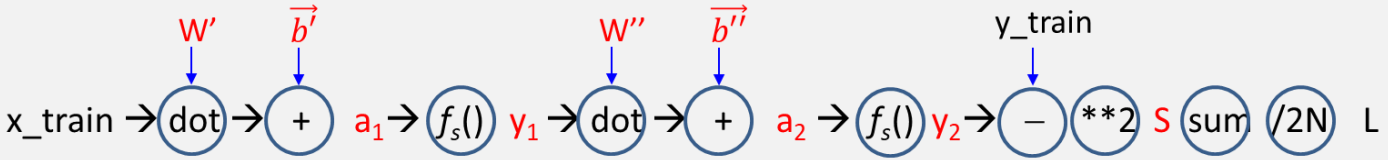
✓From last outcome to fist computational result

- Call back propagation
- Advantage ➔ Effectively calculate the partial derivative



This is back propagation

The Result of 9 Variables



1. $\frac{\partial L}{\partial S} = 1$
2. $\frac{\partial S}{\partial y_2} = \frac{1}{N}(y_2 - y\_train)$
3. $\frac{\partial y_2}{\partial a_2} = y_2 * (1 - y_2)$
4. $\frac{\partial a_2}{\partial \vec{b''}} = 1$
5. $\frac{\partial a_2}{\partial W''} = y_1^T$
6. $\frac{\partial a_2}{\partial y_1} = W''^T$
7. $\frac{\partial y_1}{\partial a_1} = y_1 * (1 - y_1)$
8. $\frac{\partial a_1}{\partial \vec{b'}} = 1$
9. $\frac{\partial a_1}{\partial W'} = x\_train^T$

y_2, y_train → shape(60K, 10)

y_1 → Output of hidden layer shape(60K, 10)

y_1^T → shape(10, 60K)

W'' → hidden to output layer shape(100, 10)

W''^T → shape(10, 100)

x_train → shape(60K, 784)

x_train^T → shape(784, 60K)

Implementation by Python

- ✓Related Python files
  - load_mnist.py (already explained previously)
    - ✓ Read the MNIST dataset
  - neuralnet.py
    - ✓ Prepare needed function for the neuron network
  - learn.py
    - ✓ Use to learn the neuron network

neuralnet.py – Initializing weight

```
1.  import numpy as np
2.  # 輸入每層神經元數目的陣列，例如 shape_list = [784, 100, 10]
3.  def make_params(shape_list):
4.      w_list = []
5.      b_list = []
6.      for i in range(len(shape_list)-1):
7.          # 產生初始值為遵從標準常態分佈的亂數
8.          weight = np.random.randn(shape_list[i], shape_list[i+1])
9.          # 始值全部設定 0.1
10.         bias = np.ones(shape_list[i+1])/10.0
11.         w_list.append(weight)
12.         b_list.append(bias)
13.     return w_list, b_list
```

neuralnet.py – Activation Function

```
14. def sigmoid(x): # sigmoid 函式
15.     return 1/(1+np.exp(-x))

16. def inner_product(x_train, w, b):# 內積再加上偏權值
17.     return np.dot(x_train, w)+ b

18. def activation(x_train, w, b):
19.     return sigmoid(inner_product(x_train, w, b))
```

neuralnet.py – Calculate Function

```
20. def calculate(x_train, w_list, b_list):
21.
22.     val_dict = {}

23.     a_1 = inner_product(x_train, w_list[0], b_list[0]) # (N, 100)
24.     y_1 = sigmoid(a_1) # (N, 100)
25.     a_2 = inner_product(y_1, w_list[1], b_list[1]) # (N, 10)
26.     y_2 = sigmoid(a_2)
27.     y_2 /= np.sum(y_2, axis=1, keepdims=True)  # 這裡進行簡單的正規化
28.     val_dict['y_1'] = y_1
29.     val_dict['y_2'] = y_2
30.
31.     return val_dict
```

neuralnet.py – Update Function

```
32. def update(x_train, w_list, b_list, y_train, eta):
33.     val_dict = {}
34.     val_dict = calculate(x_train, w_list, b_list)
35.     y_1 = val_dict['y_1']
36.     y_2 = val_dict['y_2']
37.     d12_d11 = 1.0 #1
38.     d11_d9 = 1/x_train.shape[0]*(y_2 - y_train) #2
39.     d9_d8 = y_2*(1.0 - y_2) #3
40.     d8_d7 = 1.0 #4
41.     d8_d6 = np.transpose(y_1) #5
42.     d8_d5 = np.transpose(w_list[1]) #6
43.     d5_d4 = y_1 * (1 - y_1) #7
44.     d4_d3 = 1.0 #8
45.     d4_d2 = np.transpose(x_train) #9
```

1. $\frac{\partial L}{\partial S} = 1$
2. $\frac{\partial S}{\partial y_2} = \frac{1}{N}(y_2 - y\_train)$
3. $\frac{\partial y_2}{\partial a_2} = y_2 * (1 - y_2)$
4. $\frac{\partial a_2}{\partial \vec{b''}} = 1$
5. $\frac{\partial a_2}{\partial W''} = y_1^T$
6. $\frac{\partial a_2}{\partial y_1} = W''^T$
7. $\frac{\partial y_1}{\partial a_1} = y_1 * (1 - y_1)$
8. $\frac{\partial a_1}{\partial \vec{b'}} = 1$
9. $\frac{\partial a_1}{\partial W'} = x\_train^T$

neuralnet.py – Update Function

```
46.   #(續上)以下計算 d12_d7_&_d12_d6
47.       d12_d8 = d12_d11 * d11_d9 * d9_d8 # 1 × 2 × 3

48.       b_list[1] -= eta*np.sum(d12_d8 * d8_d7, axis=0) # 1 × 2 × 3 × 4
49.       w_list[1] -= eta*np.dot(d8_d6, d12_d8) # 1 × 2 × 3 × 5

50.       #以下計算 d12_d3_&_d12_d2
51.       d12_d8 = d12_d11 * d11_d9 * d9_d8
52.       d12_d5 = np.dot(d12_d8, d8_d5)
53.       d12_d4 = d12_d5 * d5_d4
54.
55.       b_list[0] -= eta * np.sum(d12_d4 * d4_d3, axis=0)
56.       w_list[0] -= eta * np.dot(d4_d2, d12_d4)

57.       return w_list, b_list
```

$$\frac{\partial L}{\partial \vec{b''}} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial y_2} * \frac{\partial y_2}{\partial a_2} * \frac{\partial a_2}{\partial \vec{b''}}$$

$$\frac{\partial L}{\partial \vec{W''}} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial y_2} * \frac{\partial y_2}{\partial a_2} * \frac{\partial a_2}{\partial \vec{W''}}$$

$$\frac{\partial L}{\partial \vec{b'}} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial y_2} * \frac{\partial y_2}{\partial a_2} * \frac{\partial a_2}{\partial y_1} * \frac{\partial y_1}{\partial a_1} * \frac{\partial a_1}{\partial \vec{b'}}$$

$$\frac{\partial L}{\partial \vec{W'}} = \frac{\partial L}{\partial S} * \frac{\partial S}{\partial y_2} * \frac{\partial y_2}{\partial a_2} * \frac{\partial a_2}{\partial y_1} * \frac{\partial y_1}{\partial a_1} * \frac{\partial a_1}{\partial \vec{W'}}$$

Learn.py – Learn the Network

```
1.  import numpy as np
2.  import neuralnet as nl
3.  import load_mnist as lm
4.  np.random.seed(21)

5.  dataset = lm.load_mnist()
6.  x_train = dataset['x_train']
7.  y_train = dataset['y_train']
8.  x_test = dataset['x_test']
9.  y_test = dataset['y_test']
10. w_list, b_list = nl.make_params([784, 100, 10])
11. for epoch in range(1):
12.     ra = np.random.randint(60000,size=60000)
13.     for i in range(60):
14.         x_batch = x_train[ra[i*1000:(i+1)*1000],:]
15.         y_batch = y_train[ra[i*1000:(i+1)*1000],:]
16.         w_list, b_list = nl.update(x_batch, w_list, b_list, y_batch, eta=2.0)
```

1 epoch

60 batches

For a batch, 10K images are used

Learn.py – Verify the Network

The prepared testing dataset

The learned weights

```
17. #驗證成效
18. val_dict = nl.calculate(x_test, w_list, b_list)
19. print(val_dict['y_2'][0:10].round(2))
```

Show results of the first ten images

```
array([[0.07, 0.  , 0.01, 0.  , 0.  , 0.3 , 0.  , 0.59, 0.01, 0.01],
       [0.47, 0.  , 0.01, 0.05, 0.  , 0.  , 0.  , 0.21, 0.25, 0.  ],
       [0.  , 1.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.68, 0.05, 0.07, 0.02, 0.  , 0.  , 0.04, 0.04, 0.09, 0.02],
       [0.01, 0.  , 0.  , 0.02, 0.9 , 0.01, 0.04, 0.02, 0.  , 0.  ],
       [0.  , 0.99, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ],
       [0.2 , 0.  , 0.01, 0.  , 0.24, 0.13, 0.  , 0.08, 0.  , 0.34],
       [0.44, 0.  , 0.02, 0.13, 0.09, 0.12, 0.14, 0.04, 0.  , 0.  ],
       [0.  , 0.04, 0.  , 0.  , 0.4 , 0.01, 0.05, 0.49, 0.  , 0.  ],
       [0.85, 0.  , 0.  , 0.  , 0.02, 0.01, 0.  , 0.  , 0.04, 0.07]])
```

Results of 1 Epoch (60 batches)

✓The network still has space to be improved

Index	0	1	2	3	4	5	6	7	8	9	答案	
array([[	0.07,	0. ,	0.01,	0. ,	0. ,	0.3 ,	0. ,	**0.59**,	0.01,	0.01],	7	△
	[0.47,	0. ,	**0.01**,	0.05,	0. ,	0. ,	0. ,	0.21,	0.25,	0.],	2	✗
	[0. ,	**1.** ,	0. ,	0. ,	0. ,	0. ,	0. ,	0. ,	0. ,	0.],	1	✓
	[	**0.68**,	0.05,	0.07,	0.02,	0. ,	0. ,	0.04,	0.04,	0.09,	0.02],	0	△
	[0.01,	0. ,	0. ,	0.02,	**0.9** ,	0.01,	0.04,	0.02,	0. ,	0.],	4	✓
	[0. ,	**0.99**,	0. ,	0. ,	0. ,	0. ,	0. ,	0. ,	0. ,	0.],	1	✓
	[0.2 ,	0. ,	0.01,	0. ,	**0.24**,	0.13,	0. ,	0.08,	0. ,	0.34],	4	✗
	[0.44,	0. ,	0.02,	0.13,	0.09,	0.12,	0.14,	0.04,	0. ,	**0.**],	9	✗
	[0. ,	0.04,	0. ,	0. ,	0.4 ,	**0.01**,	0.05,	0.49,	0. ,	0.],	5	✗
	[0.85,	0. ,	0. ,	0. ,	0.02,	0.01,	0. ,	0. ,	0.04,	**0.07**]])	9	✗

Results of 200 Epoch (100 batches)

✓The network can predict all most correctly

```
array([[0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 1.   , 0.   , 0.   ], ✓
       [0.   , 0.   , 1.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ], ✓
       [0.   , 1.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ], ✓
       [0.96, 0.   , 0.   , 0.   , 0.   , 0.02, 0.   , 0.02, 0.   , 0.   ], ✓
       [0.   , 0.   , 0.   , 0.   , 0.75, 0.   , 0.   , 0.   , 0.01, 0.23], ✓
       [0.   , 1.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ], ✓
       [0.   , 0.   , 0.   , 0.   , 0.81, 0.02, 0.   , 0.   , 0.   , 0.16], ✓
       [0.   , 0.   , 0.   , 0.03, 0.02, 0.02, 0.   , 0.   , 0.06, 0.87], ✓
       [0.01, 0.   , 0.1  , 0.   , 0.01, 0.07, 0.8  , 0.   , 0.01, 0.   ], ✗
       [0.   , 0.   , 0.   , 0.   , 0.01, 0.   , 0.   , 0.01, 0.   , 0.98]]) ✓
```

Verify the Network by Measurements

✓By Accuracy (準確率) and Loss (總誤差值)

```
1.  def predict(X, w_list, b_list, t): # 這裡進行預測
2.      val_list = calculate(X, w_list, b_list, t)
3.      y_2 = val_list['y_2']
4.      result = np.zeros_like(y_2)
5.      # 對應樣本數
6.      for i in range(y_2.shape[0]):
7.          result[i, np.argmax(y_2[i])] = 1
8.      return result

9.  def accuracy(X, w_list, b_list, t): # 這裡計算準確率
10.     pre = predict(X, w_list, b_list, t)
11.     result = np.where(np.argmax(t, axis=1)==np.argmax(pre, axis=1), 1, 0)
12.     acc = np.mean(result)
13.     return acc

14. def loss(X, w_list, b_list, t): # 這裡計算總損失
15.     L = calculate(X, w_list, b_list, t)['L']
16.     return L
```

ps: 將三函式加到neuralnet.py中

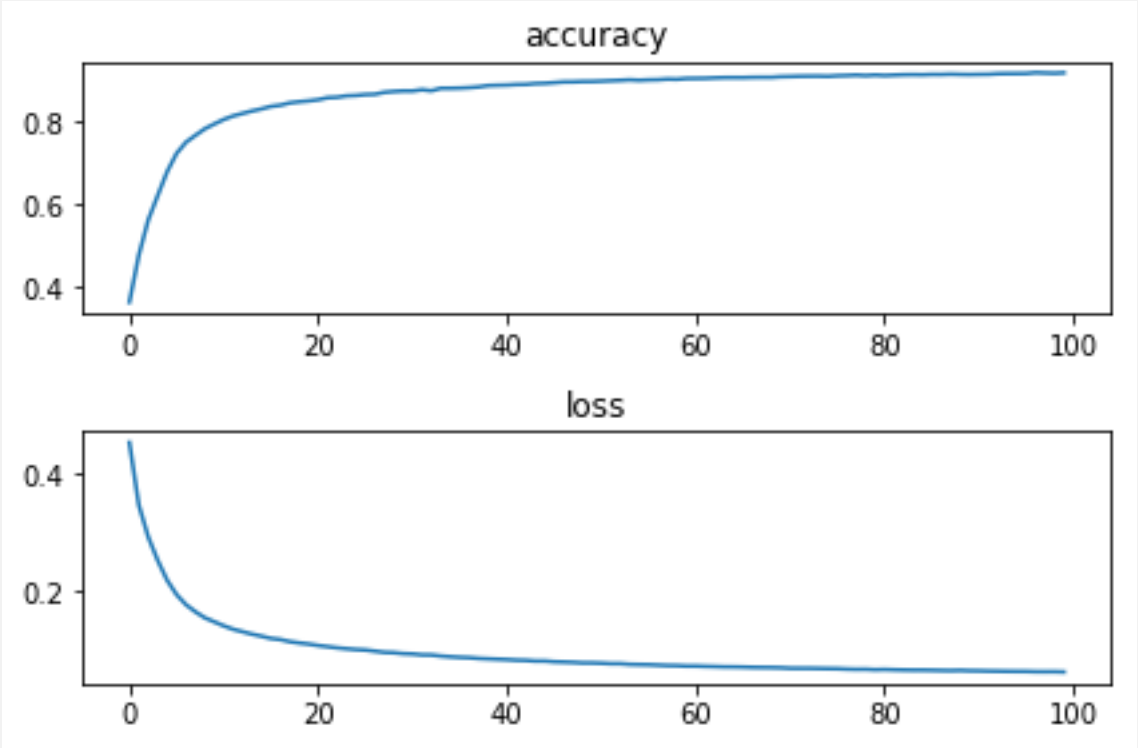
The Updated Python Files

✓Related Python files

- load_mnist.py ➔ load_mnist_acc_loss.py
  - ✓ Read the MNIST dataset
- neuralnet.py ➔ neuralnet_acc_loss.py
  - ✓ Prepare needed function for the neuron network
- learn.py ➔ learn_acc_loss.py
  - ✓ Use to learn the neuron network

After 100 Epochs

✓Accuracy and Loss



epoch:99,
Accuracy: 0.915700,
Loss: 0.063376

Recap

- ✓A neuron to Many neurons (A neural network)
- ✓Implementation through the MINIST dataset
  - load_mnist.py
    - ✓ Read the MNIST dataset
  - neuralnet.py
    - ✓ Prepare needed function for the neuron network
  - learn.py
    - ✓ Use to learn the neuron network