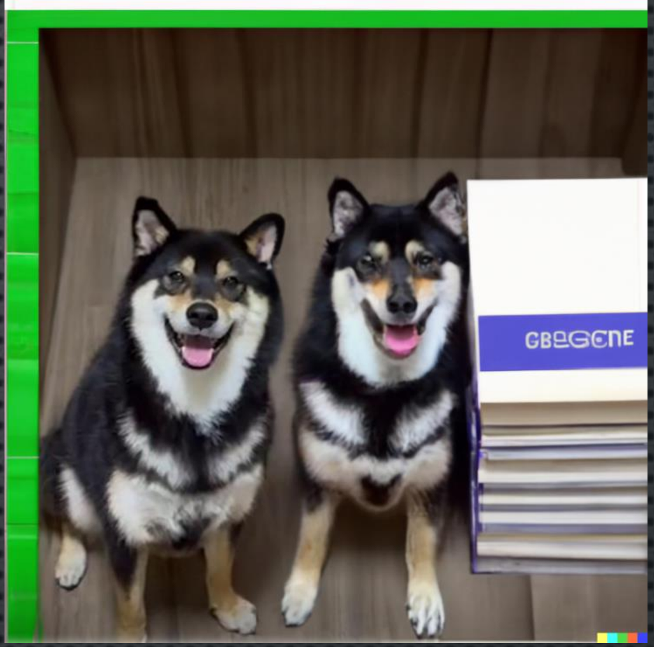
AI人工智慧—
神經網路模型

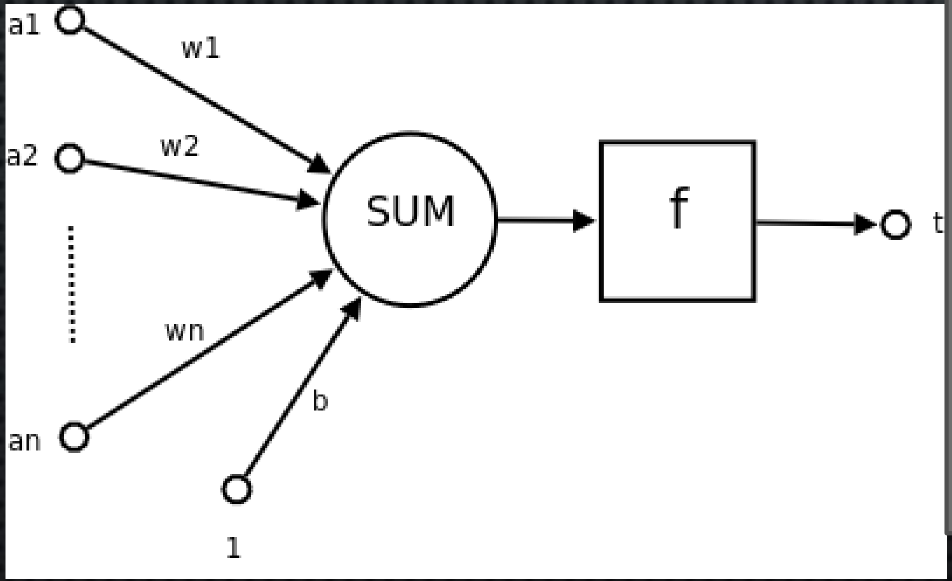


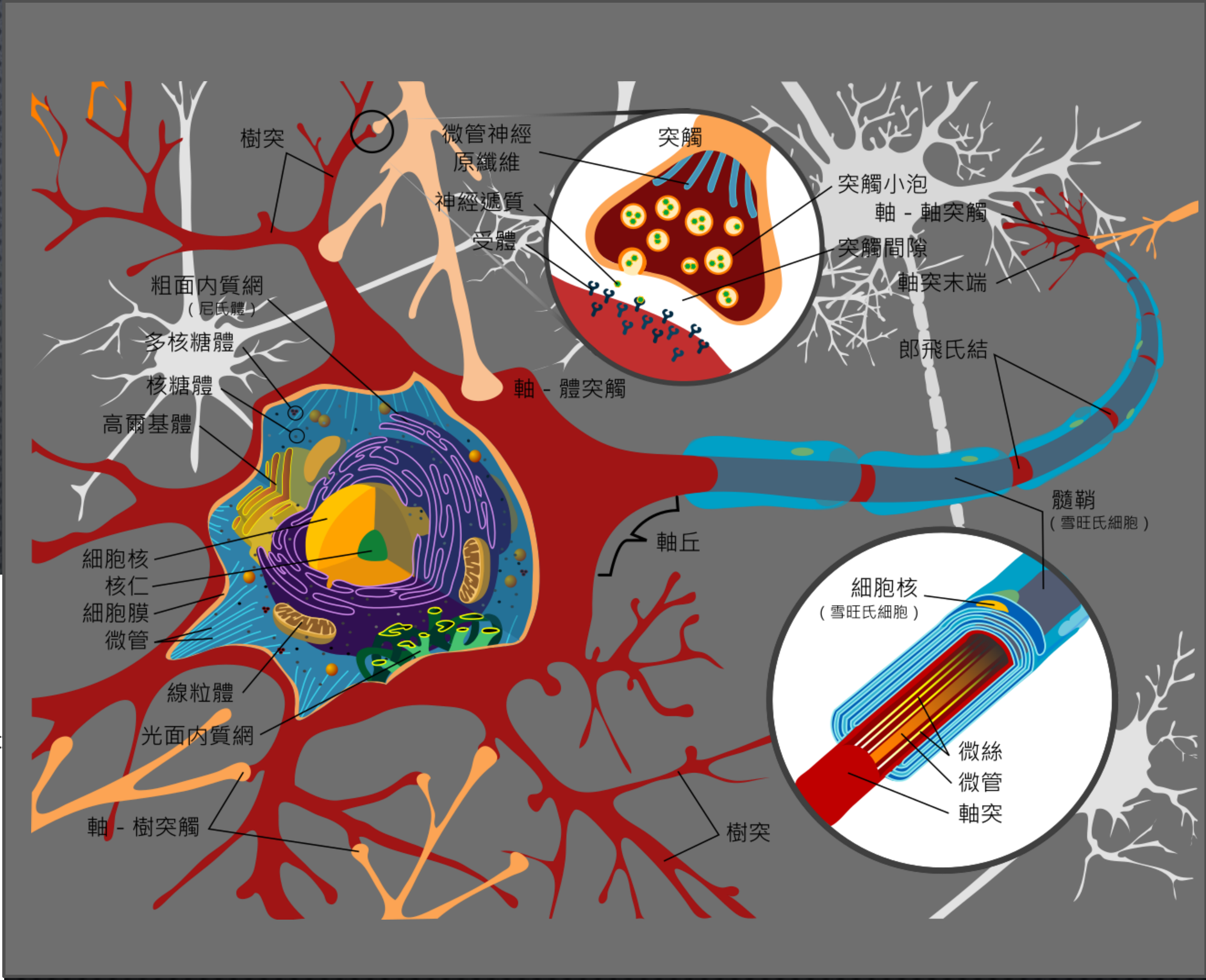
玩過Keras神經網路模型
doing過要learning

von anwendeng

神經元 (wiki)

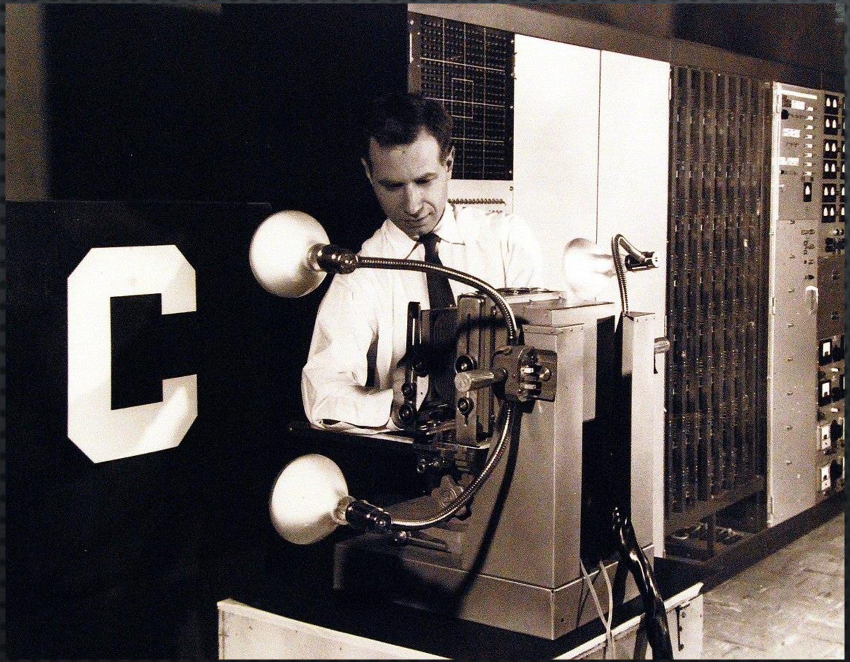
https://upload.wikimedia.org/wikipedia/commons/5/59/Complete_neuron_cell_diagram_zh-hant.svg





- 感知器（Perceptron）是弗蘭克·羅森布拉特(Frank Rosenblatt (July 11, 1928 – July 11, 1971))在1957年就職於康奈爾航空實驗室（Cornell Aeronautical Laboratory）時所發明的一種人工神經網路。
- Frank Rosenblatt 開發並探索了當今深度學習系統的所有基本要件，他應該被視為深度學習之父

取自https://commons.wikimedia.org/
330-PSA-80-60 (USN 710739): Experimental Machine Able To Identify Letters of Alphabet Announced By Navy. This photo shows the Mark I Perceptron, an experimental machine which can be trained to automatically identify objects or patterns, such as letters of the alphabet. Originated by Dr. Frank Rosenblatt



- 感知器一台機器，它的首次實現是在IBM 704的軟體，但隨後它在定制硬體中實現，稱為“Mark I 感知器”，專為圖像辨識而設計。

By Rosenblatt, F. - Rosenblatt, F. The Design of an Intelligent Automaton, Research Reviews, Office of Naval Research. Washington, October 1958, 5-13, Public Domain, https://commons.wikimedia.org/w/index.php?curid=139658945

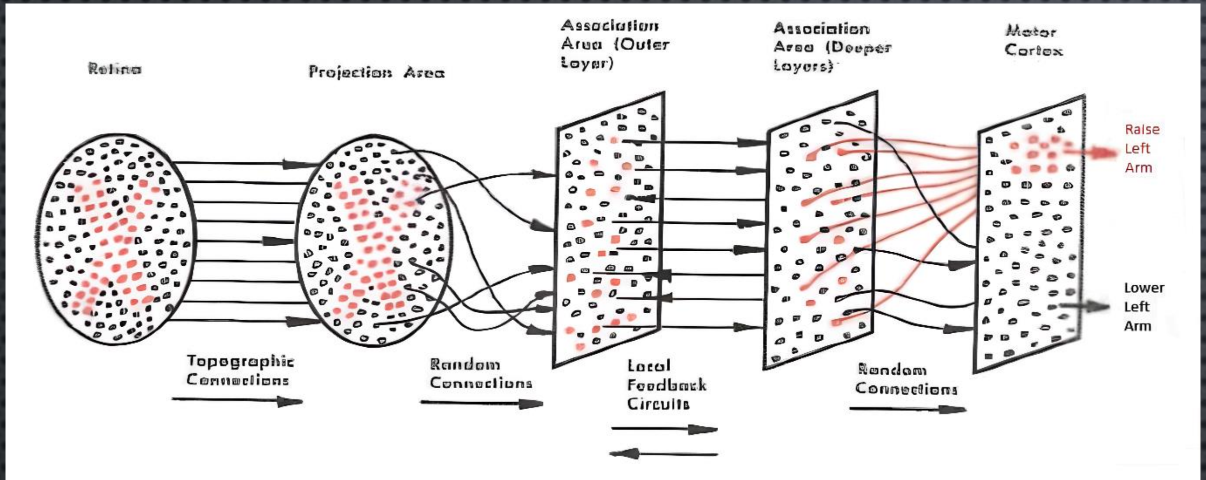


FIG. 1 — Organization of a biological brain. (Red areas indicate active cells, responding to the letter X.)

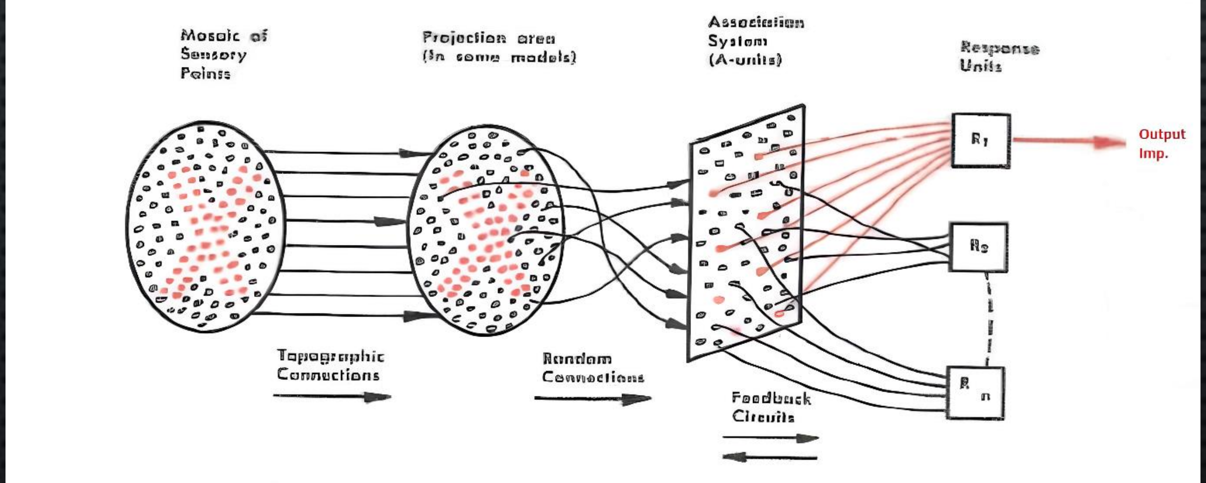
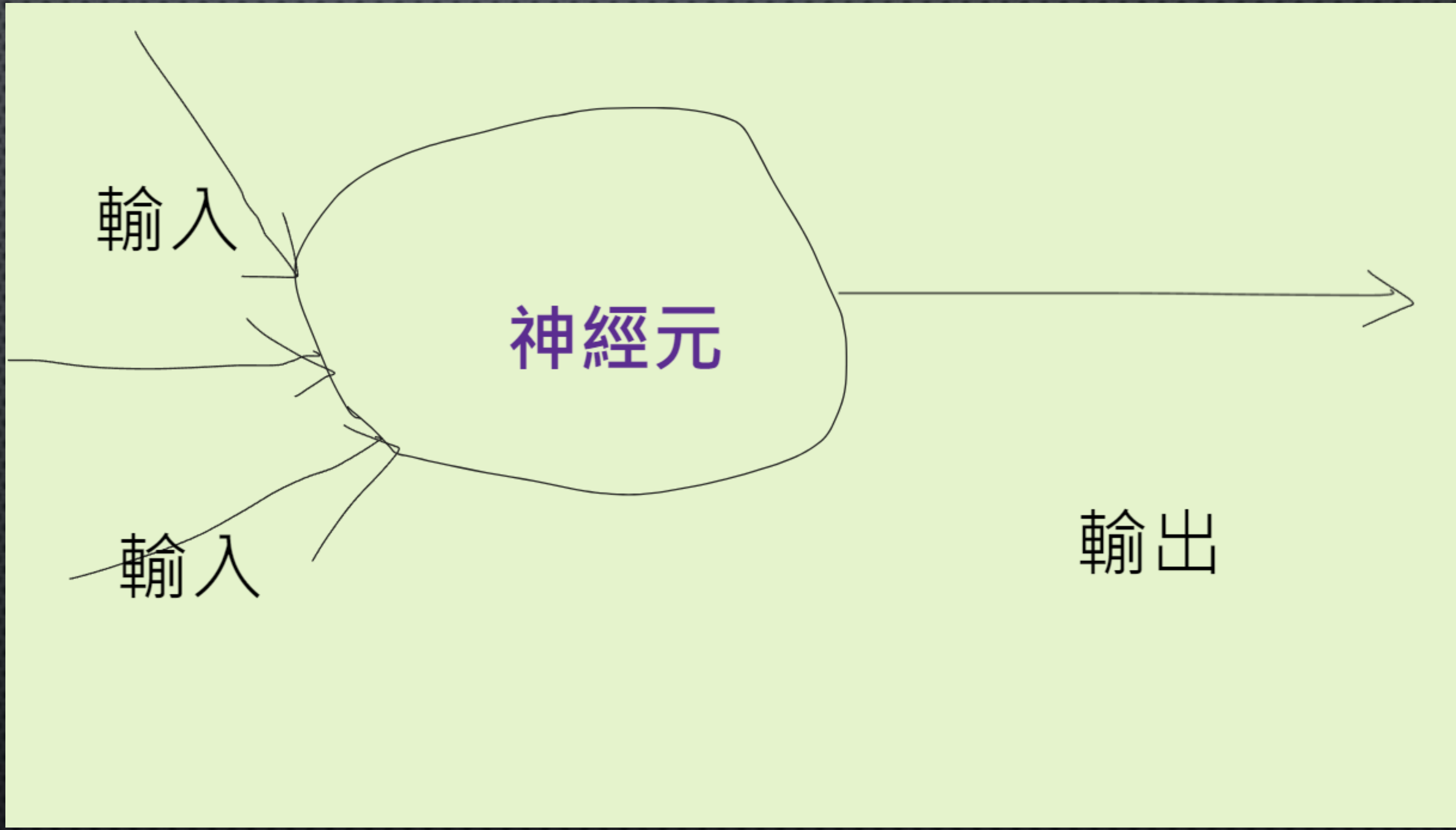


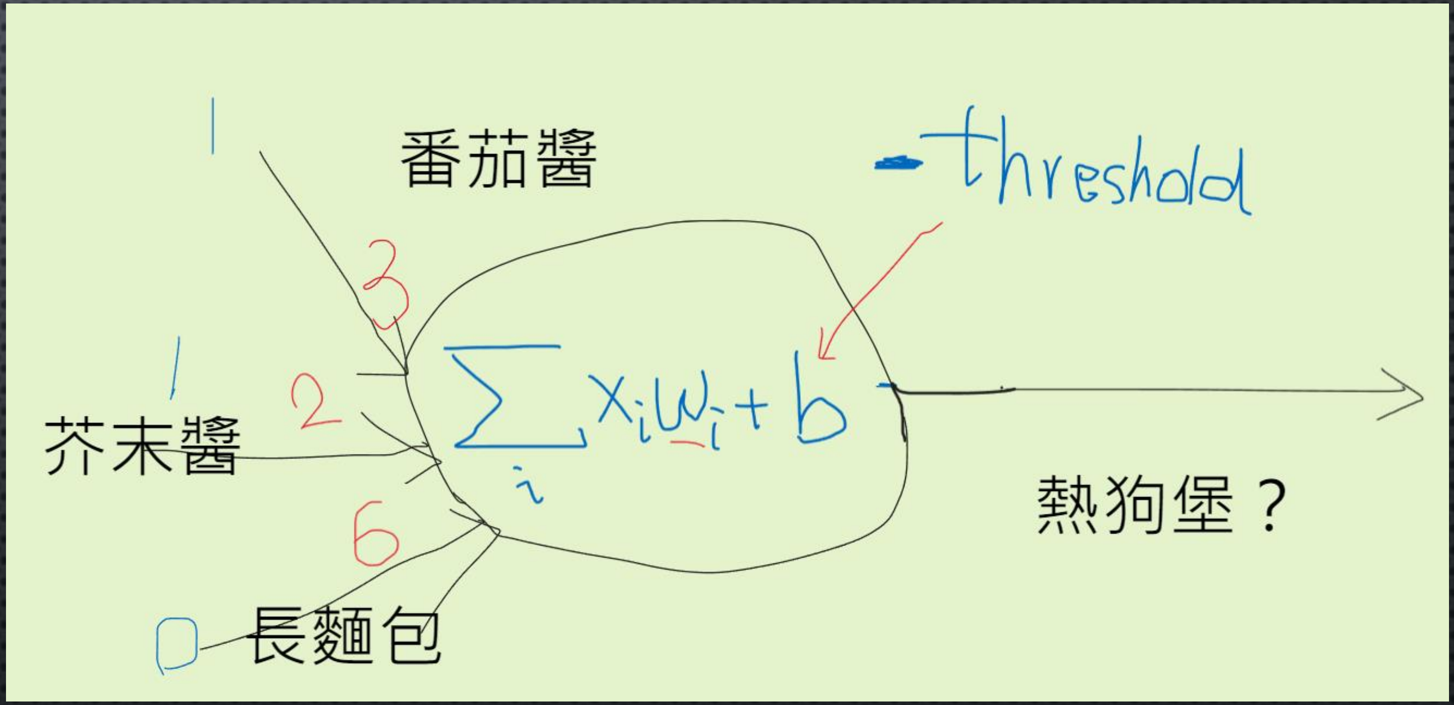
FIG. 2 — Organization of a perceptron.

- Mark I 感知器有 3 層。
- 排列成 20x20 網格的 400 個光電管陣列，稱為「感覺單元」（S 單元）或「輸入視網膜」。
- 感知器的隱藏層，稱為「關聯單元」（A 單元）。
- 感知器的輸出層，稱為「響應單元」（R-units）。
- S 單元隨機連接到 A 單元，以「消除感知器中任何特定的故意偏差」。 連結權重是固定的，不是學習的。

神經網路的結構



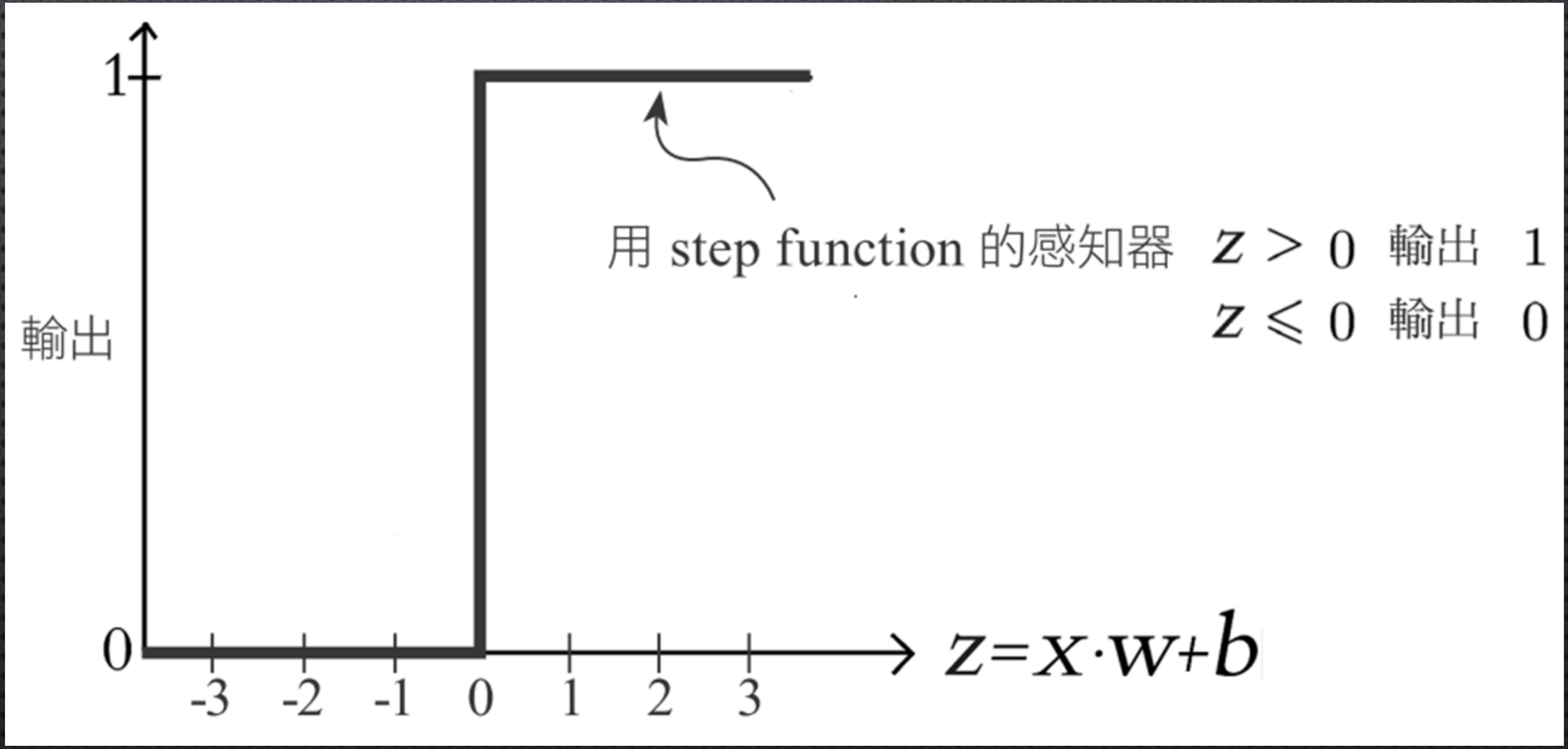
Weight, threshold? How about inner product?



f>0 output=1

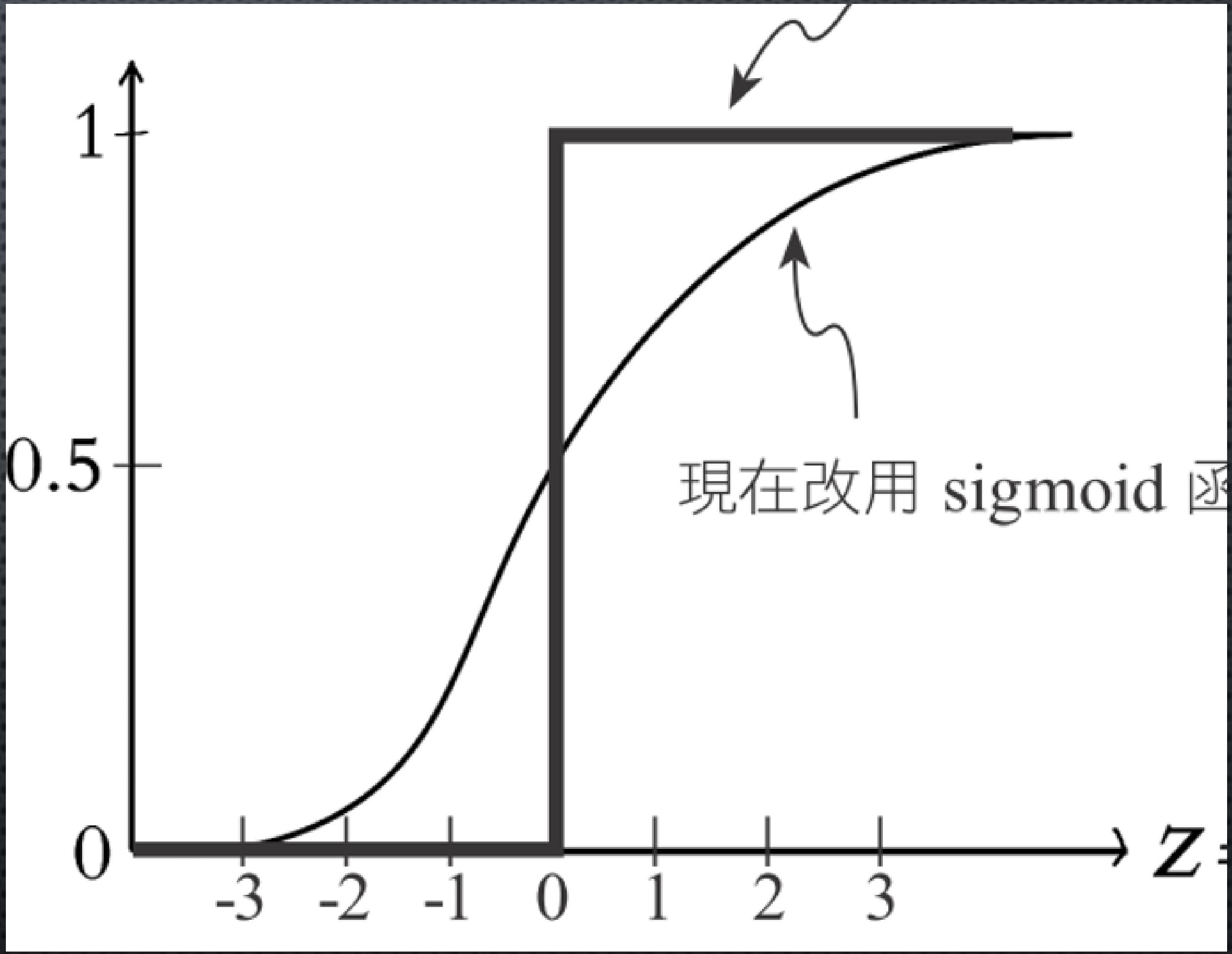
f<=0 output=0

Step function
not good



activation函數的選擇

- sigmoid 函數
- Tanh
- ReLU
- tf.Keras 提供了 leaky ReLU 、參數化 ReLU (parametric ReLU) ... 等「進階」啟動函數


1
0.5
0
-3
-2
-1
0
1
2
3
現在改用 sigmoid 函
z

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

$$\tanh = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$$

$$\boxed{\tanh(x) = 2\sigma(2x) - 1}$$

$$\frac{2 - (1+e^{-2x})}{1+e^{-2x}} = \frac{1 - e^{-2x}}{1+e^{-2x}}$$

on

```python
import numpy as np
import math
```

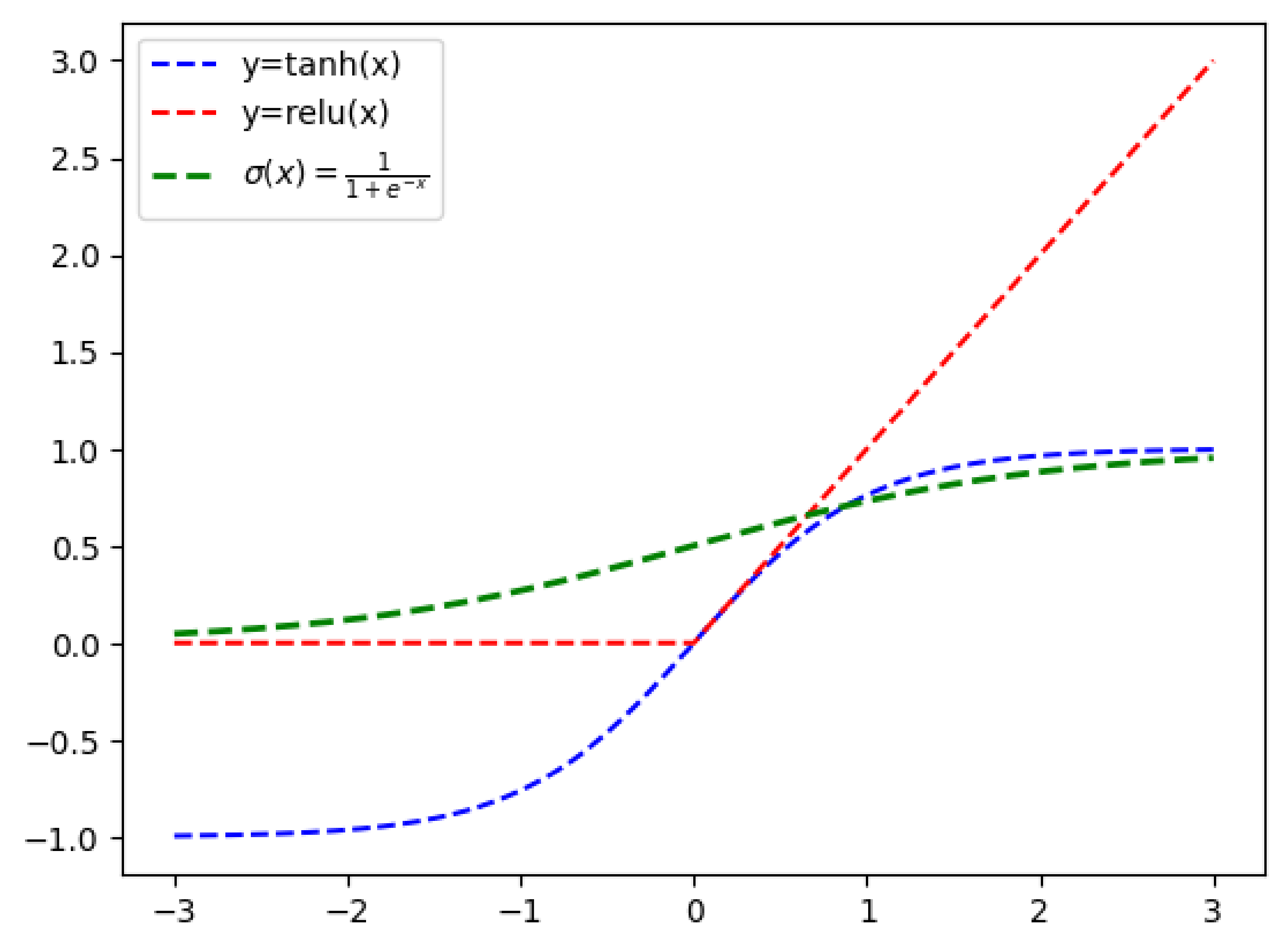
```python
def sigmoid(x):
    return 1/(1+np.exp(-x))
```

```python
def relu(x):
    return np.where(x<0, 0, x)
```

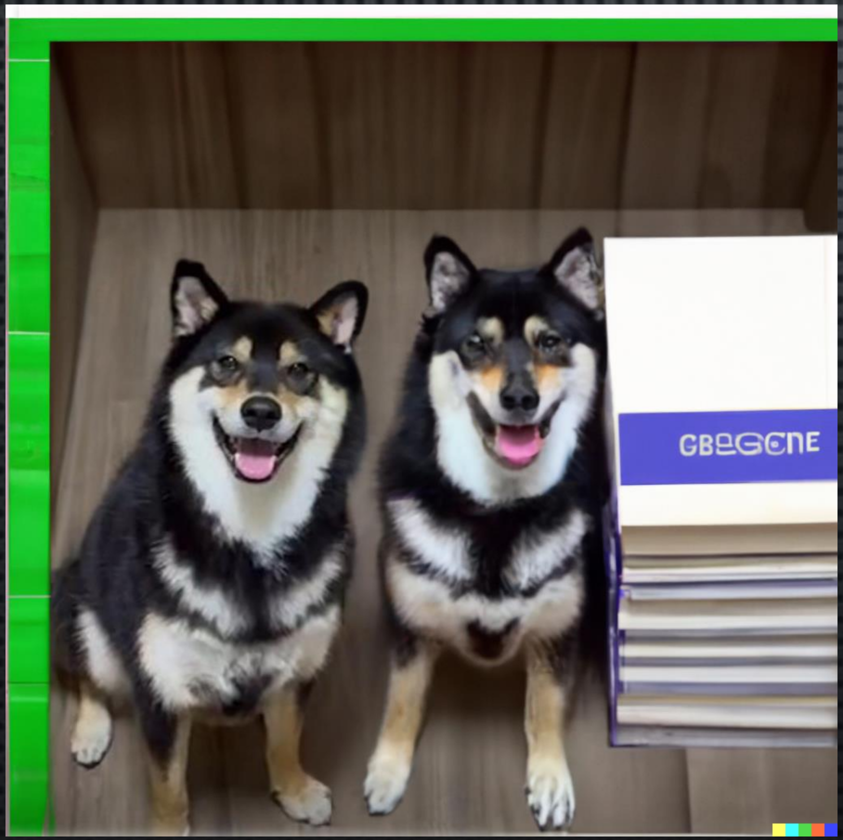
```python
from matplotlib import pyplot as plt
```

```python
x = np.linspace(-3, 3,1000)
y = np.tanh(x) #hyperbolic tangent function
yy =sigmoid(x)
y2 =relu(x)

plt.plot(x,y, 'b--', label='y=tanh(x)')
plt.plot(x, y2, 'r--', label='y=relu(x)' )
plt.plot(x, yy, 'g--', linewidth=2, label=r"$\sigma(x)=\frac{1}{1+e^{-x}}$")

```

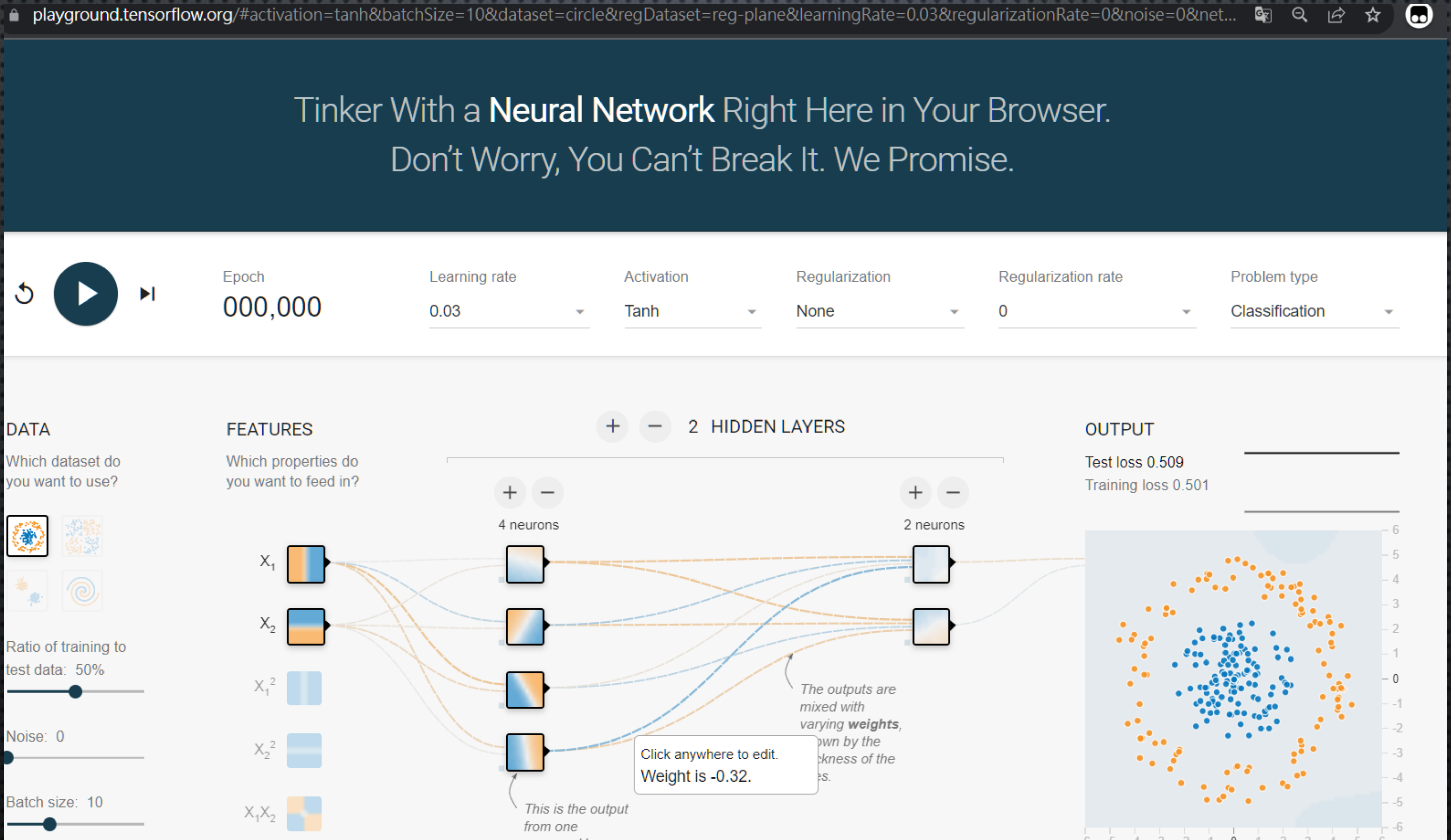
用python的numpy & matplotlib.pyplot



多神經元組成的神經網路



https://playground.tensorflow.org/

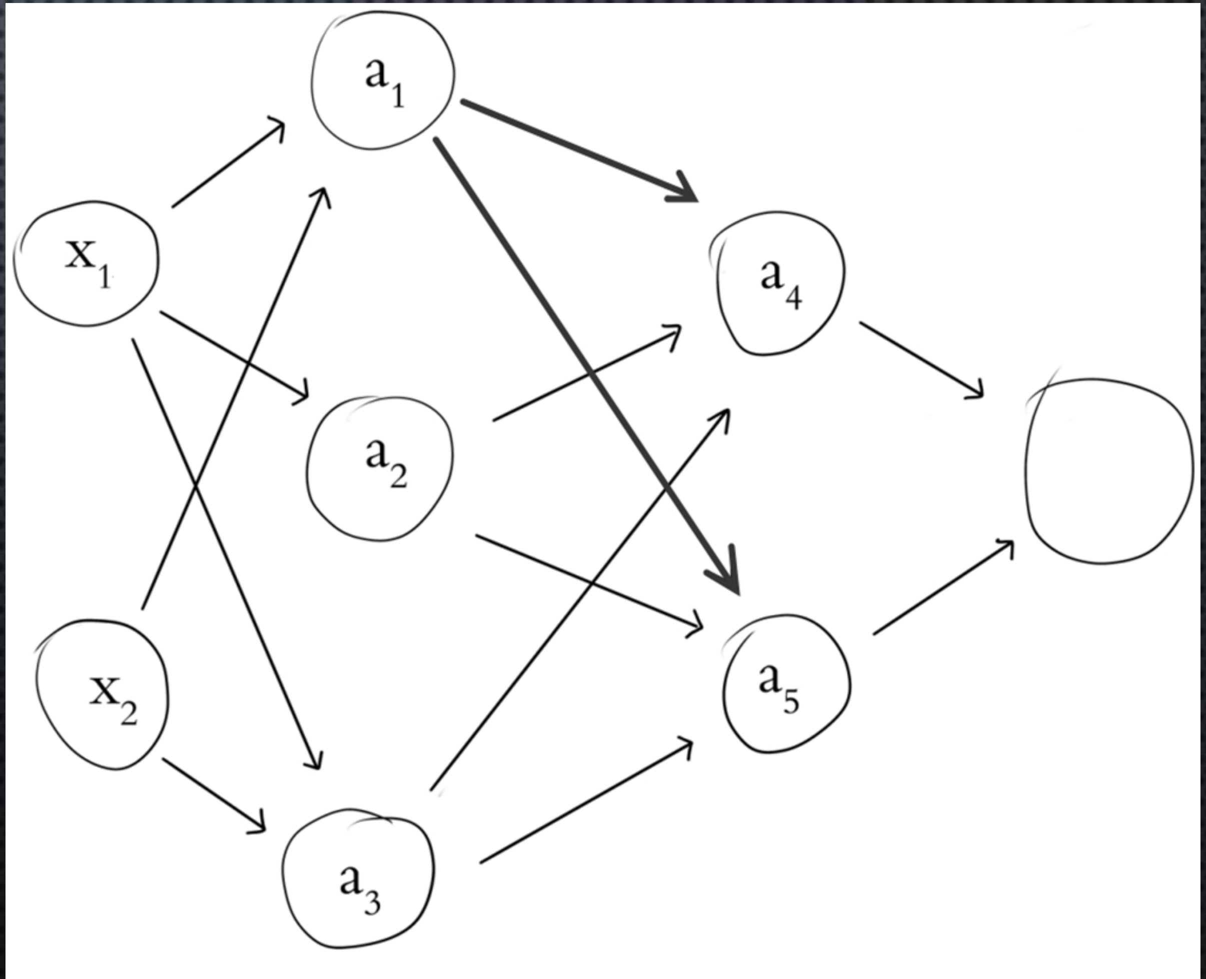


密集神經網路辨識

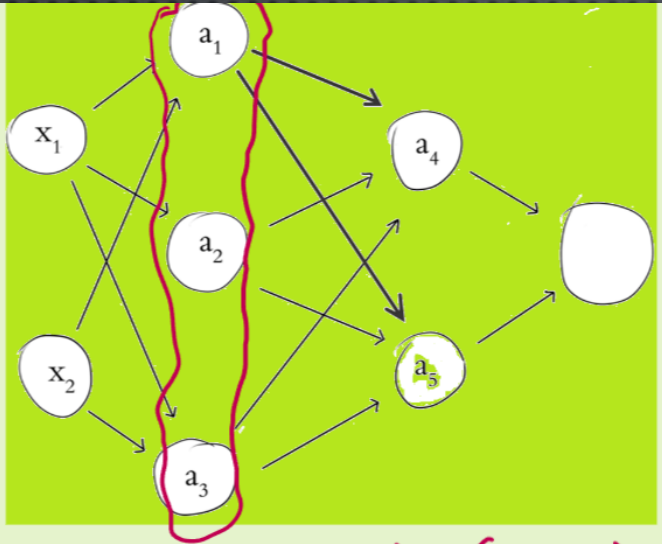
輸入層 → 第 1 隱藏層(前向傳播運算)

第1 隱藏層 → 第 2 隱藏層

第 2 隱藏層 → 輸出層



Numpy實作前向傳播計算



forward propagation

$$\underbrace{\begin{bmatrix} -0.5 & 1.5 \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}}_{3\times 2} \underbrace{\begin{bmatrix} 4 \\ 3 \end{bmatrix}}_{2\times 1} + \underbrace{\begin{bmatrix} -0.9 \\ b_2 \\ b_3 \end{bmatrix}}_{3\times 1} =$$

$$(-0.5, 1.5)\cdot(4,3) - 0.9$$
$$= -2 + 4.5 - 0.9$$
$$= 2.5 - 0.9$$
$$= 1.6$$

$$\begin{bmatrix} 1.6 \\ z_2 \\ z_3 \end{bmatrix} \xmapsto{\text{Relu}} \begin{bmatrix} 1.6 \\ a_2 \\ a_3 \end{bmatrix}$$

Numpy 矩陣乘法

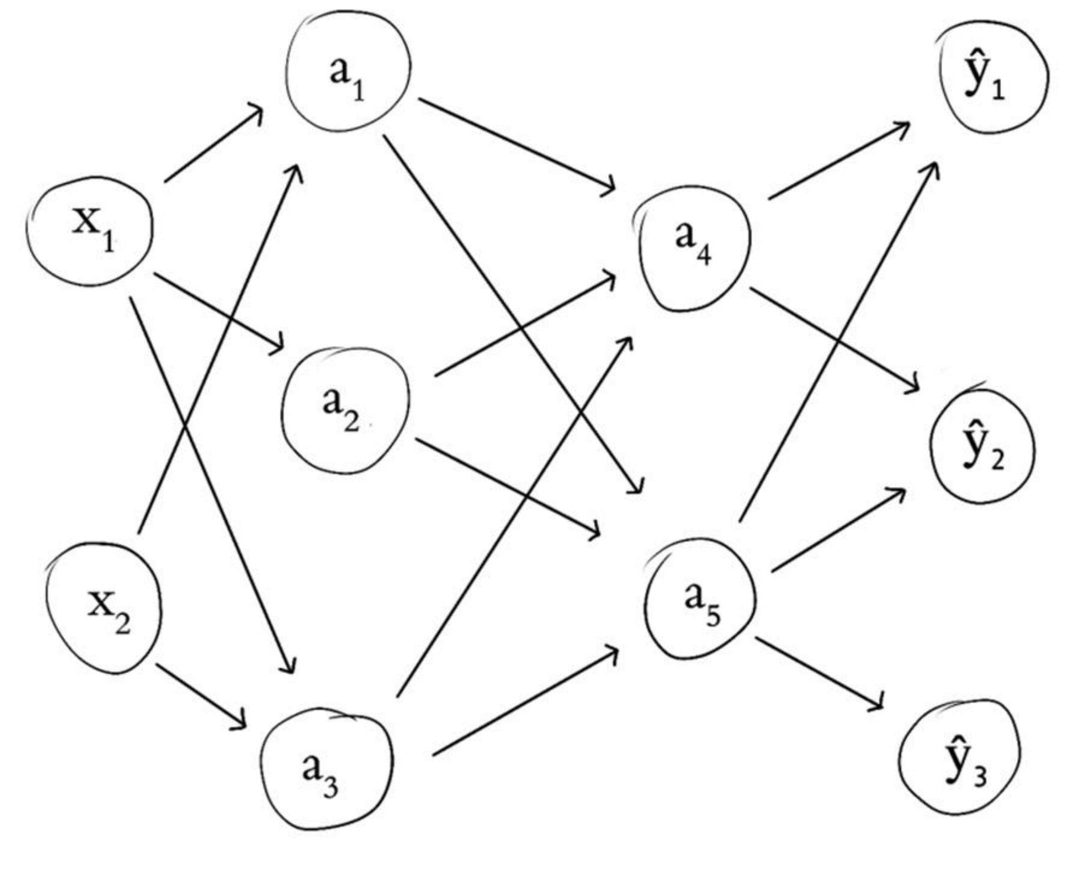
```python
#矩陣乘法
z=w@x.T+b.transpose()
print(z)
print(type(z))
print(w.shape, x.shape, b.shape, z.shape)
print(np.dot(w, x.T)+b.T)
print(np.matmul(w,x.T)+b.T)
```

```
[ 1.6  2.4 -0.4]
<class 'numpy.ndarray'>
(3, 2) (2,) (3,) (3,)
[ 1.6  2.4 -0.4]
[ 1.6  2.4 -0.4]
```

```python
print(w)
w1.append(b[0])
w2.append(b[1])
w3.append(b[2])
w=[w1, w2, w3]
x=np.append(x,1)
print(w, x)
w@x.T
```

```
[[-0.5  1.5]
 [-0.1  1.1]
 [ 0.1  0.9]]
[[-0.5, 1.5, -0.9], [-0.1, 1.1, -0.5], [0.1, 0.9, -3.5]] [4 3 1]
array([ 1.6,  2.4, -0.4])
```

密集神經網路分類



Softmax

$$S(z_i) = \frac{e^{z_i}}{\sum_i e^{z_i}}$$

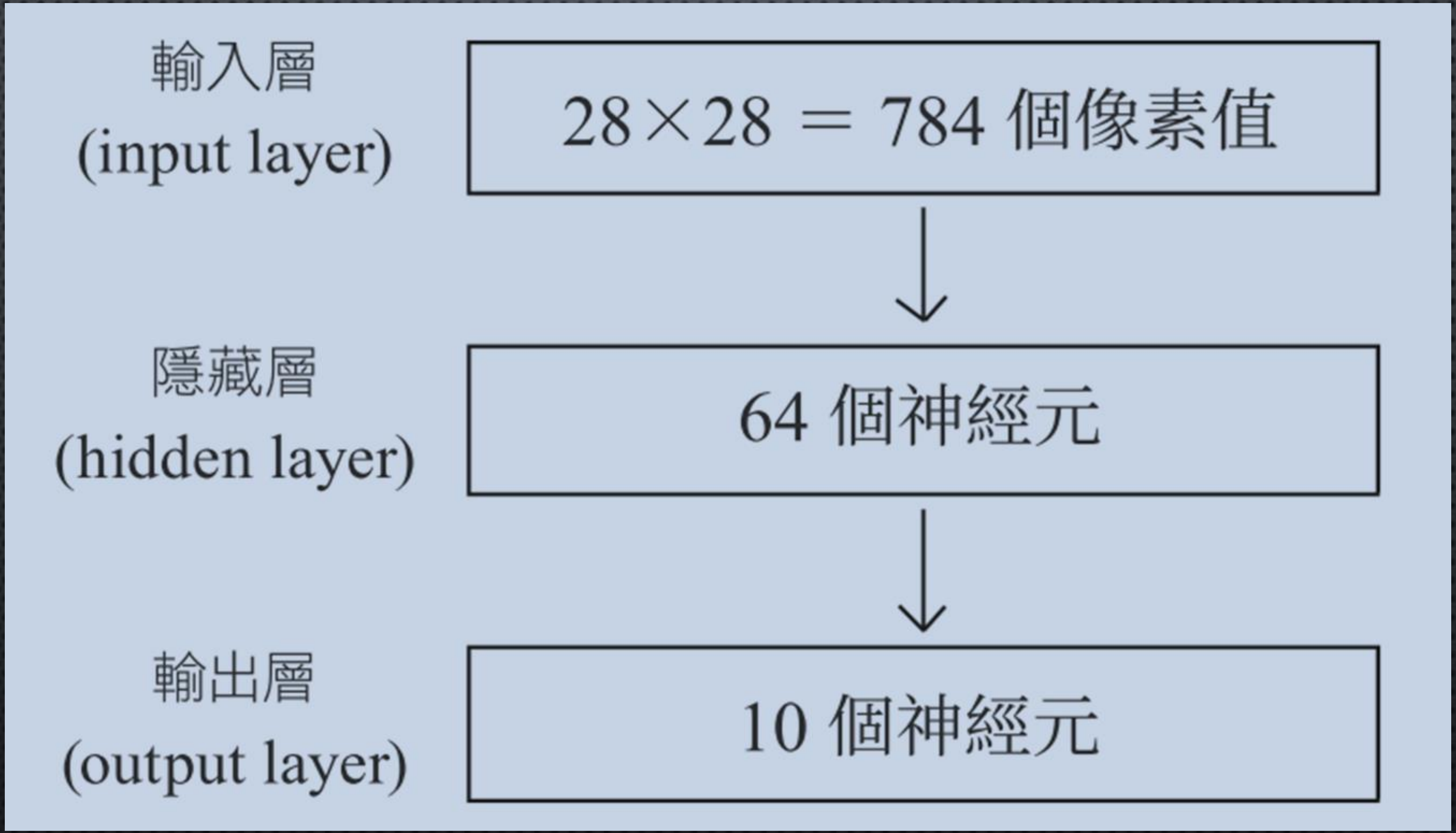
$$\text{exp\_sum} = \sum_i e^{z_i}$$

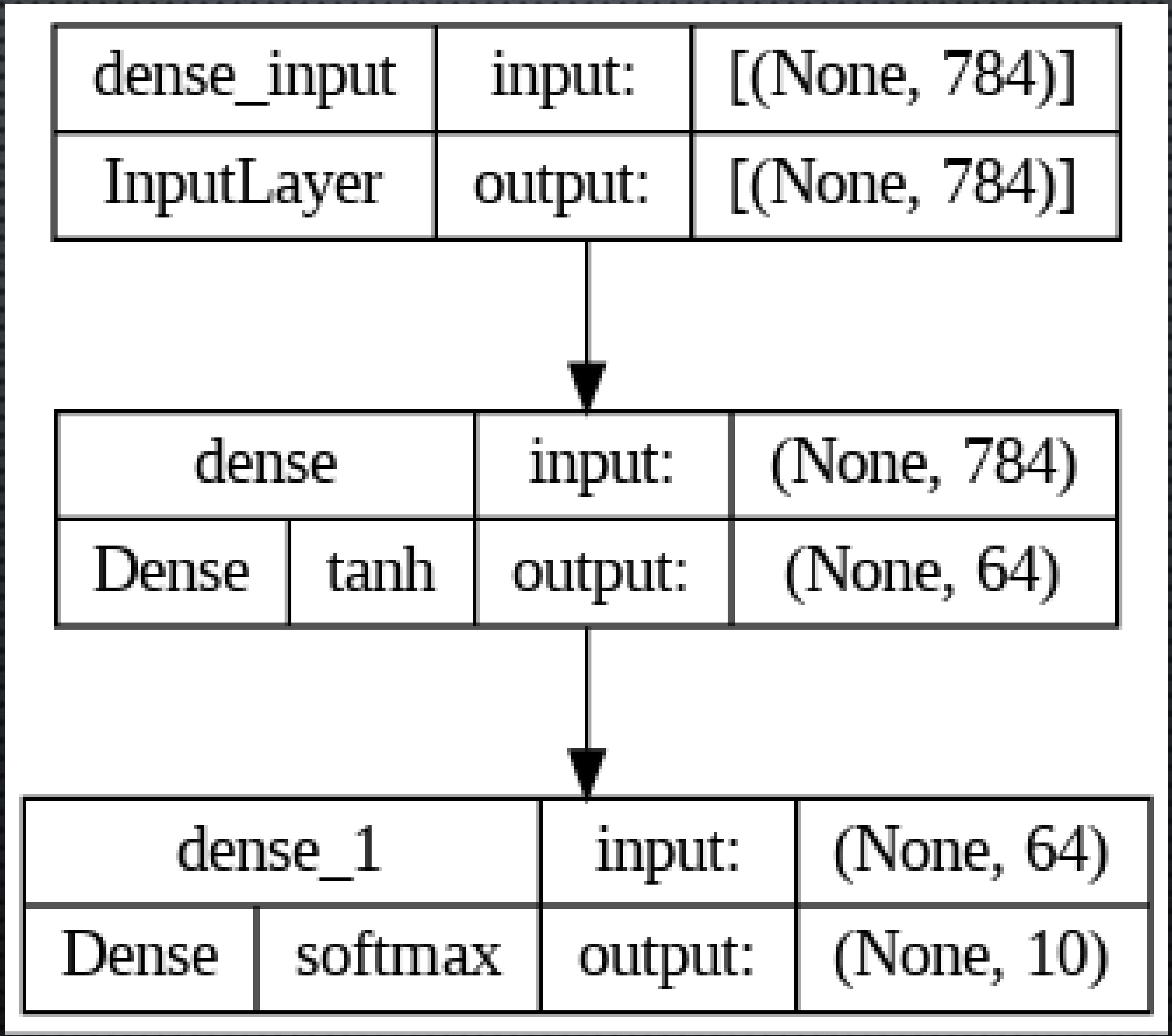
$$\begin{bmatrix}-1\\ 1\\ 5\end{bmatrix} \longrightarrow \begin{bmatrix}\frac{e^{-1}}{\text{sum}}\\ \frac{e^{1}}{\text{sum}}\\ \frac{e^{5}}{\text{sum}}\end{bmatrix}$$

$$\text{exp\_sum} \rightarrow e^{-1}+e^{1}+e^{5}$$

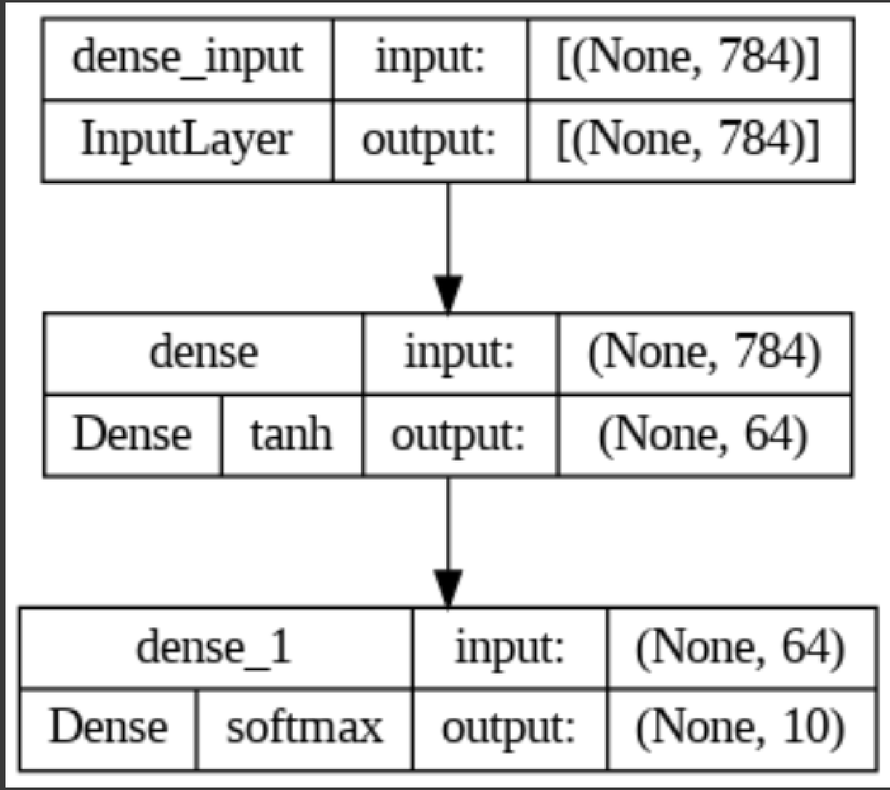
softmax 函數的運算

```
>>> import numpy as np
>>> a = [1.0, 2.0, 3.0, 4.0, 1.0, 2.0, 3.0]
>>> np.exp(a) / np.sum(np.exp(a))
array([0.02364054, 0.06426166, 0.1746813,
0.474833, 0.02364054,0.06426166, 0.1746813])
```


輸入層
(input layer)
28×28 = 784 個像素值
隱藏層
(hidden layer)
64 個神經元
輸出層
(output layer)
10 個神經元


dense_input
input:
[(None, 784)]
InputLayer
output:
[(None, 784)]
dense
input:
(None, 784)
Dense
tanh
output:
(None, 64)
dense_1
input:
(None, 64)
Dense
softmax
output:
(None, 10)

```
1  plot_model(model, show_shapes=True, show_layer_activations=True, sh)
```




```
%%sh                      def named_script_magic(line,
show_dtype=
show_layer_names=
show_trainable=
%shell
%%shell
```

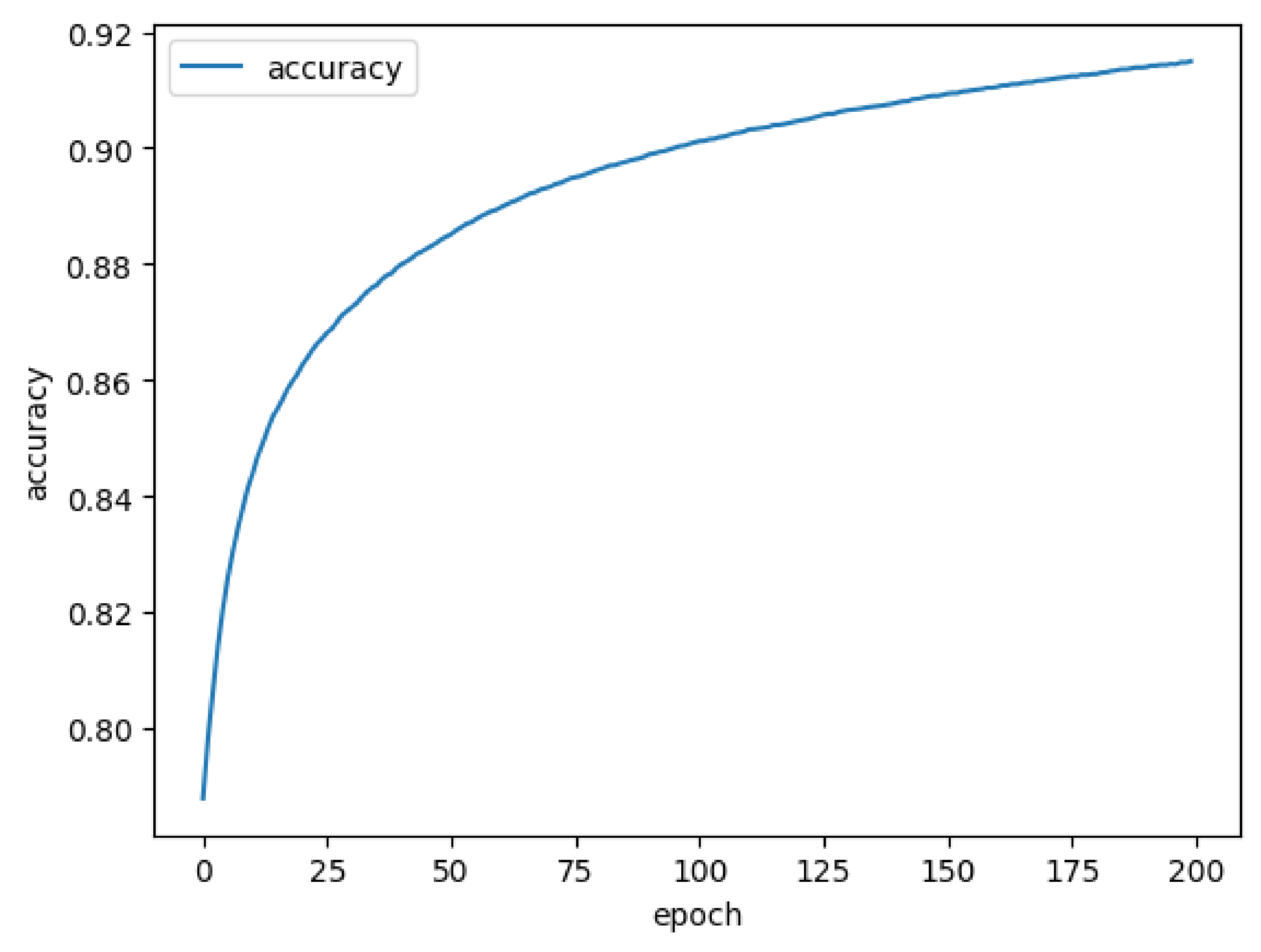
Example:

```
input = tf.keras.Input(shape=(100,), dtype='int32',
name='input')
x = tf.keras.layers.Embedding(
    output_dim=512, input_dim=10000, input_length=100)
(input)
```

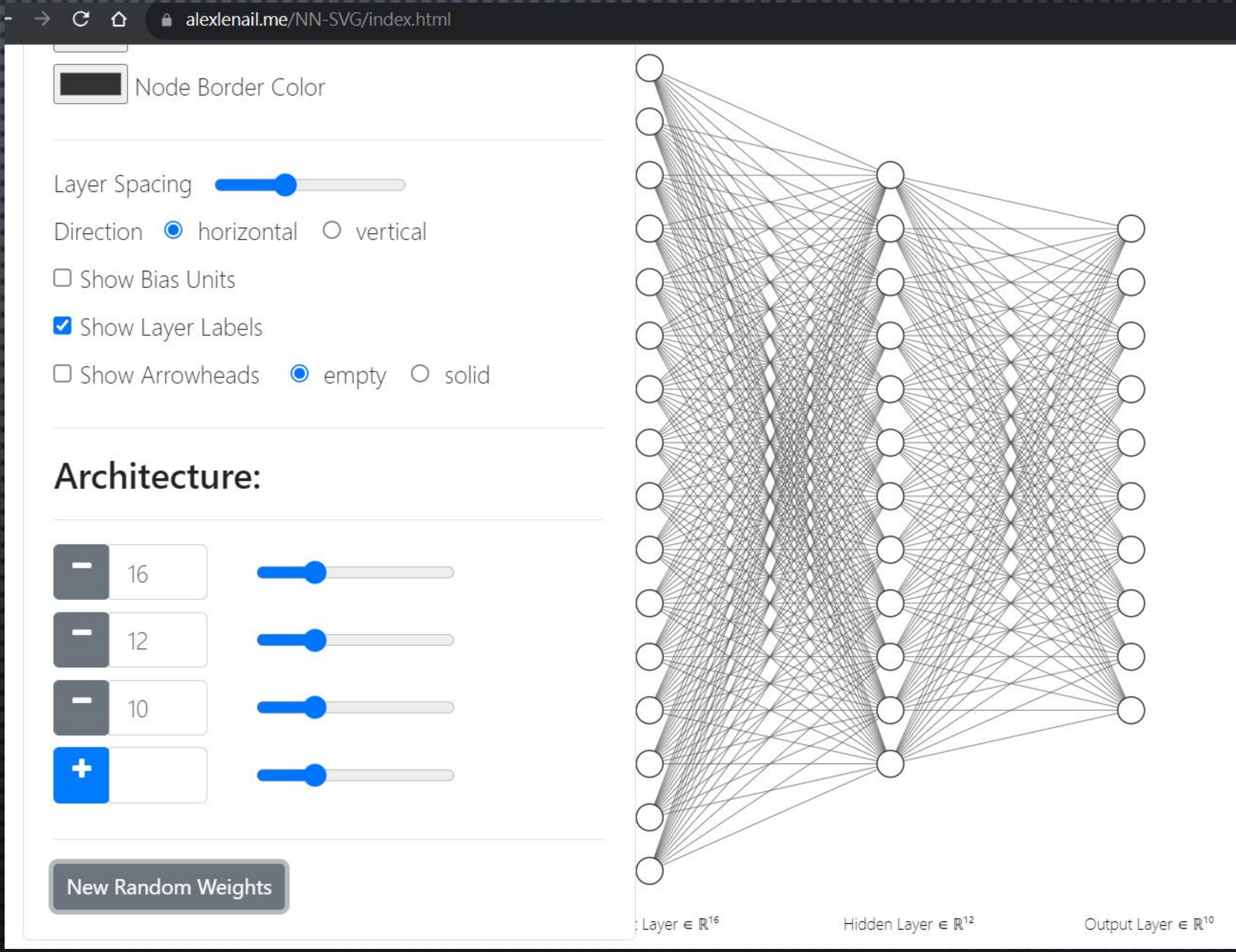
```python
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 64)                50240

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 50890 (198.79 KB)
Trainable params: 50890 (198.79 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Shallow Neural Network
Use tanh

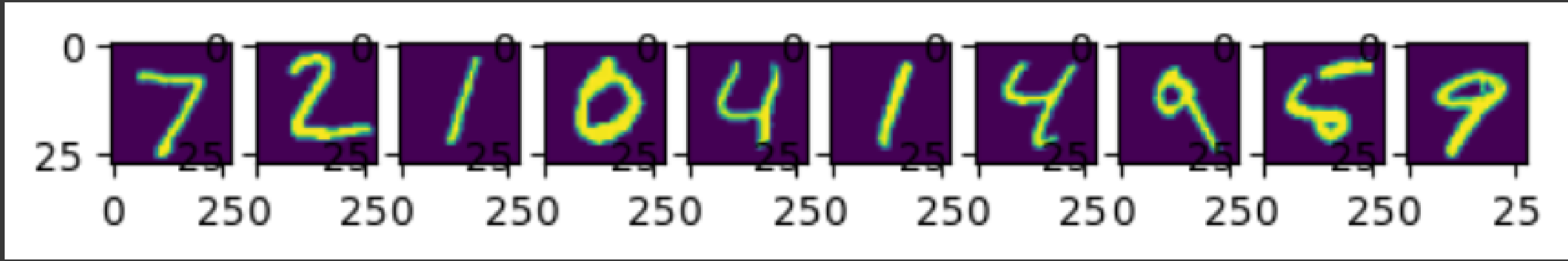


https://alexlenail.me/NN-SVG/index.html



利用pyplot強化之前的程式

```
for i in range(10):
    plt.subplot(1, 10, i+1)
    plt.imshow(X_test[i].reshape((28, 28)))
plt.show()
prediction=np.argmax(model.predict(X_test[0:10]), axis=1)
print(prediction)
```



```
1/1 [==============================] - 0s 65ms/step
[7 2 1 0 4 1 4 9 6 9]
```

- 由神經元組成的神經網路，一步步運算出輸出值$\hat{y}$

軟體實作

von anwendeng

感謝觀賞

Herzlichen Dank für die
Aufmerksamkeit

von anwendeng