iThome
TechTalk

用 Python 快速實現深層學習

蔡炎龍
政治大學應用數學系

1 瞭解 deep learning 基礎

2 會寫 deep learning 程式

今天的目標

假設今天你在路上撿到一個神燈...

你擦拭了以後果然如傳說出現一位巨人...

為了報答你，他給你兩個選擇，你只能選一個

說好的三個願望呢？

1 任意函數生成器

2 現金三千萬

你會選哪個

當然要選

任意函數生成器!!

[例子] 股票點數預測

f(日期 x) = 某股票 x 當天的收盤價

$$f(x_1, x_2, \ldots, x_{n-1}) = x_n$$

這可能不是最好的描述法。也許我們比較想要用之前的情況預測我們的想知道的那天。也就類似是上面這樣的函數。

[例子] 你是那種類型的人？

假設我們把人分成若干型，例如 9 型 (九型人格)，16 型 (Myers-Briggs) 等等。於是我們想看這是哪一型的人，就是找這個函數：

f(某人 x) = x 這個人的人格型式

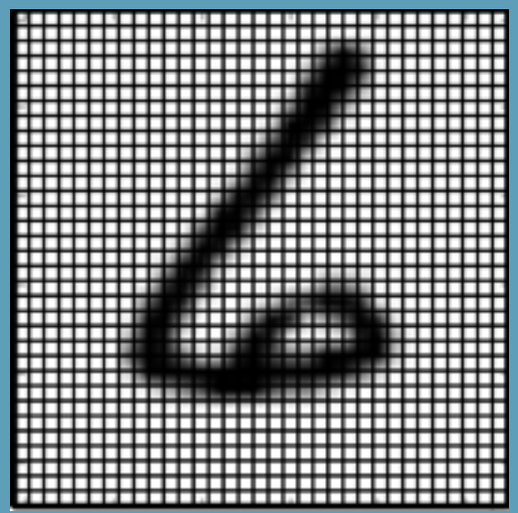
你怎麼「輸入」一個人呢？ 在此例通常是經過心理測驗，得到一串數字，那些數字我們就可以代表這個人。

[例子] 速配指數

f(x, y) = x, y 這兩個人的速配指數

我們再度碰到要把人變成一串描述他的數字。很多可以用，比如星座、血型、興趣、學歷等等我們都可想辦法轉數字。

[例子] 圖形辨識

f() = 6

28x28

$$f : \mathbb{R}^{784} \rightarrow \mathbb{R}$$

其實這裡有兩個問題要討論：

1. 輸入圖形很大的時候呢?
2. 輸出是 0.2, 代表什麼呢?

總之我們就是要函數

$$f : \mathbb{R}^n \to \mathbb{R}^m$$

暗黑學習法

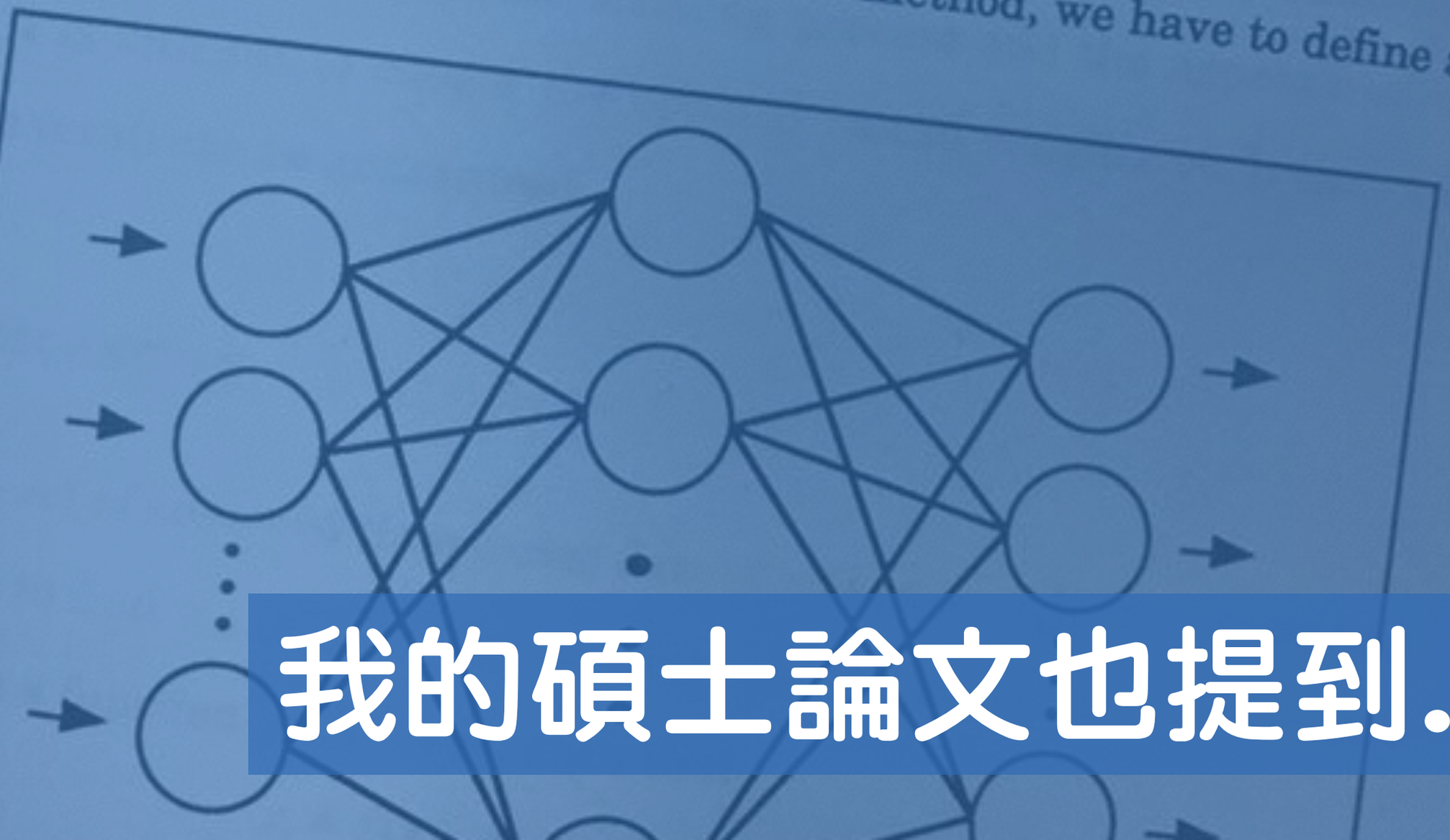
真的有學任意函數的技法

就是神經網路!

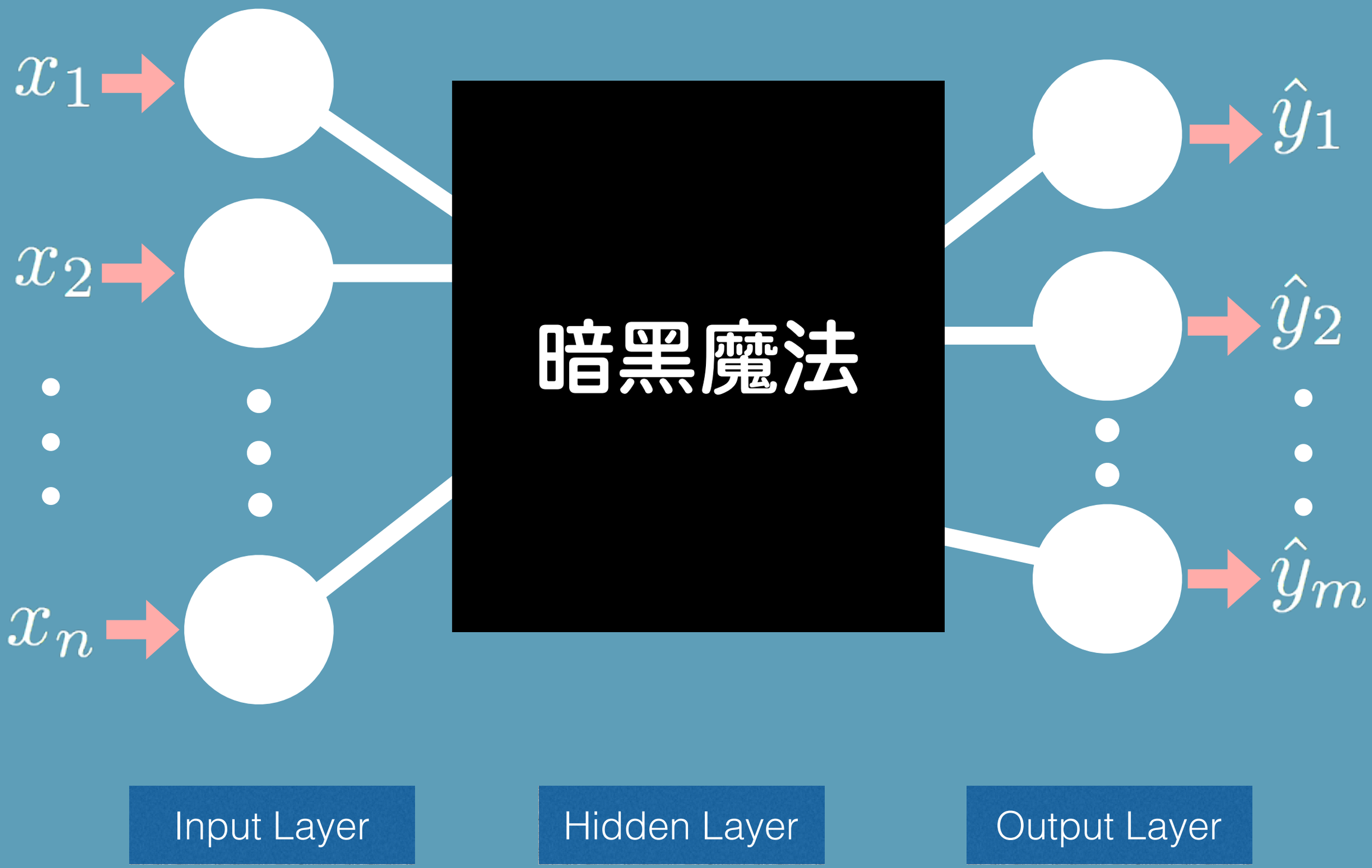
所以在 1980-1990 變成很潮的東西

2.3 Back Propagation Networks

The back propagation network is a kind of the feedforward networks. The network uses "gradient descent" (Rumelhart and McClelland, 1986) method as the learning algorithm. Using this method, we have to define a



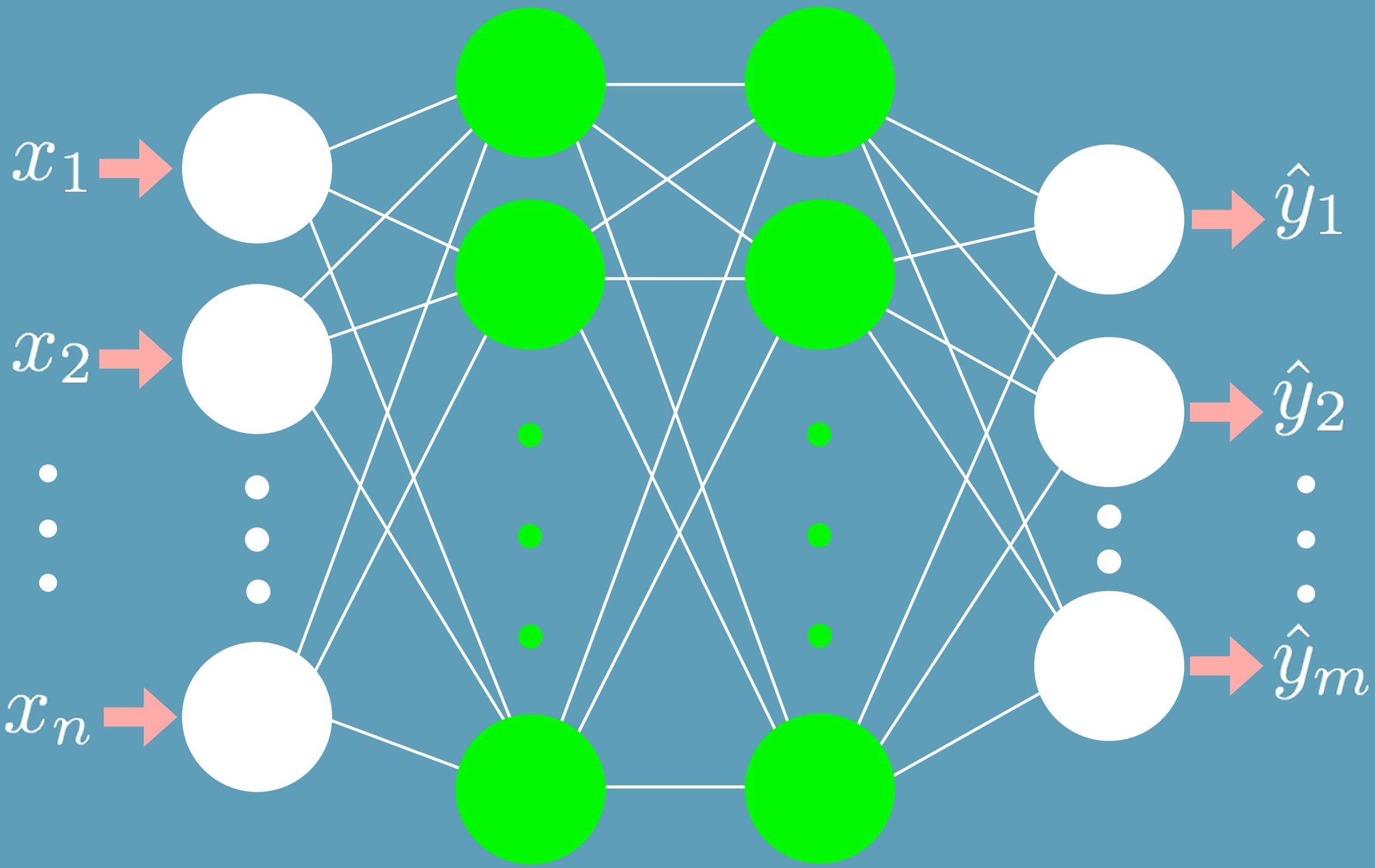
我的碩士論文也提到...


x_1
x_2
x_n
暗黑魔法
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_m$
Input Layer
Hidden Layer
Output Layer

厲害的是神經網路什麼都學得會!

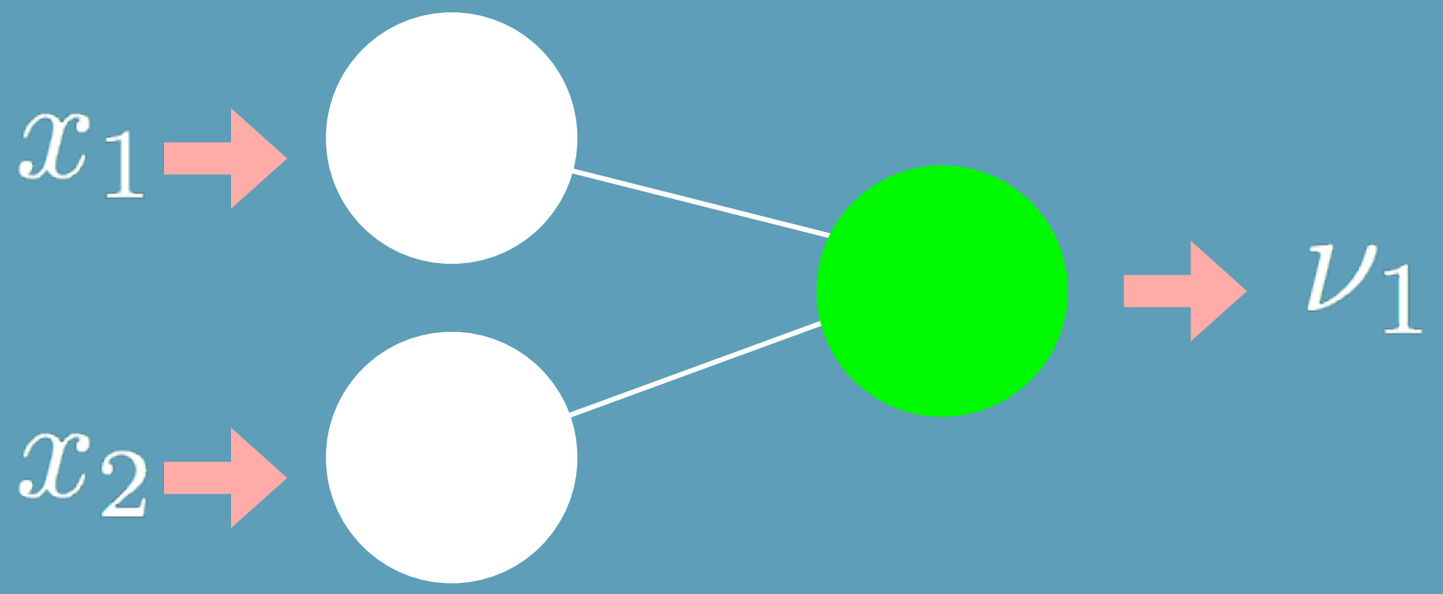
而且你完全不用告訴它函數應該長什麼樣子: 線性啦、二次多項式啦等等。

打開暗黑世界

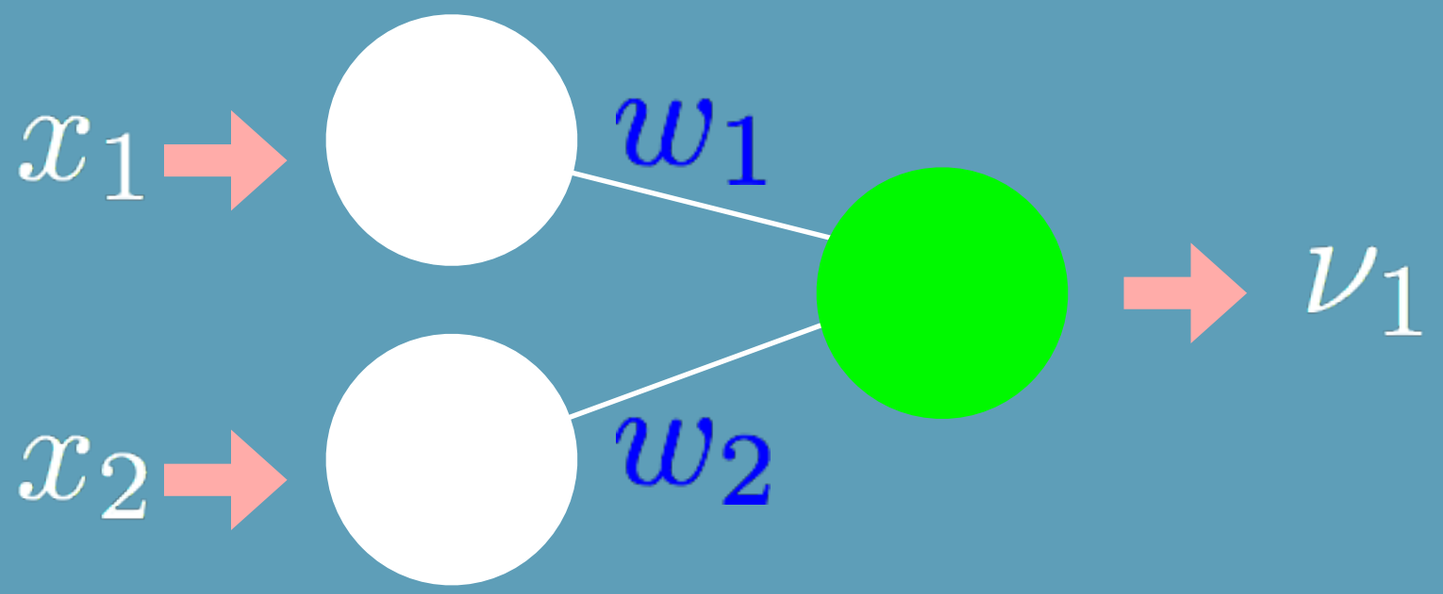

x_1
x_2
x_n
$\hat{y}_1$
$\hat{y}_2$
$\hat{y}_m$
Input Layer
Hidden Layer
Output Layer

每個神經元動作基本上是一樣的!

反正就是接受輸入經某種運算輸出

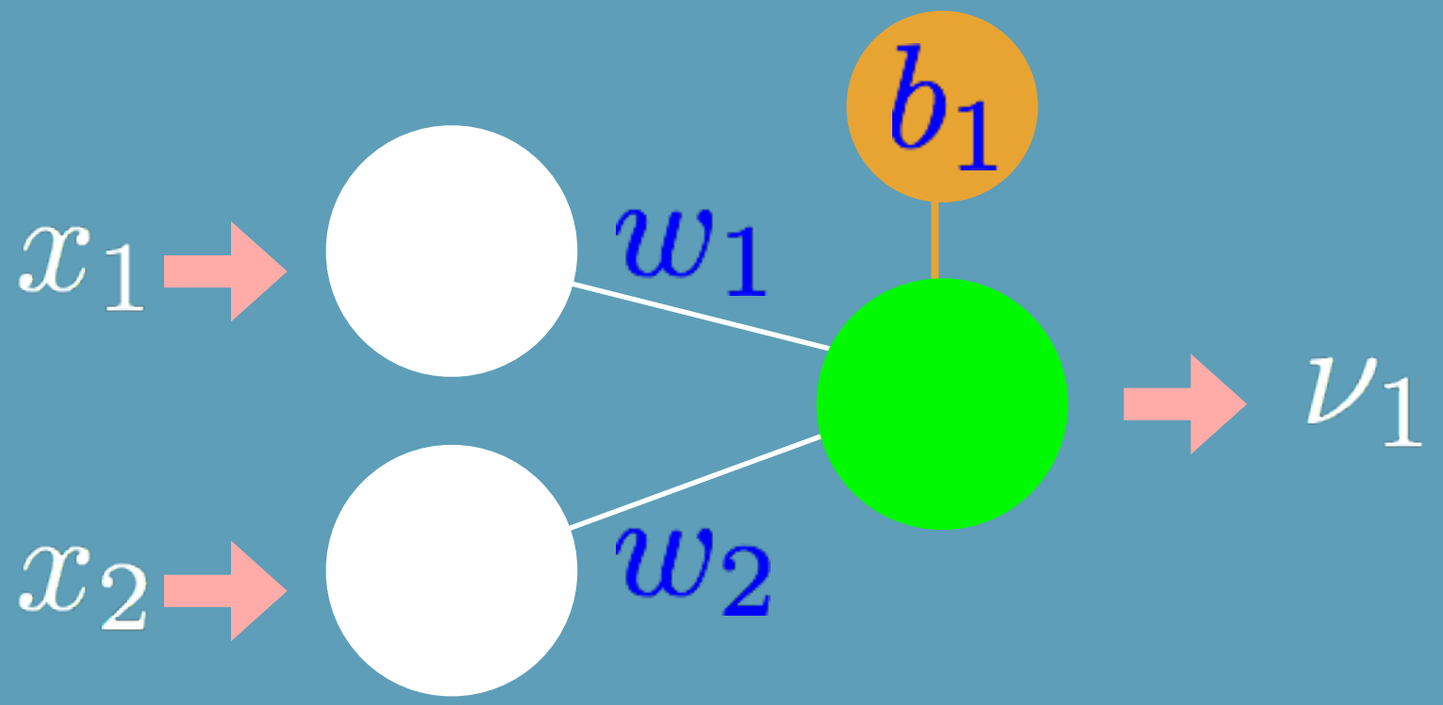


(1) 經過 weight 加權



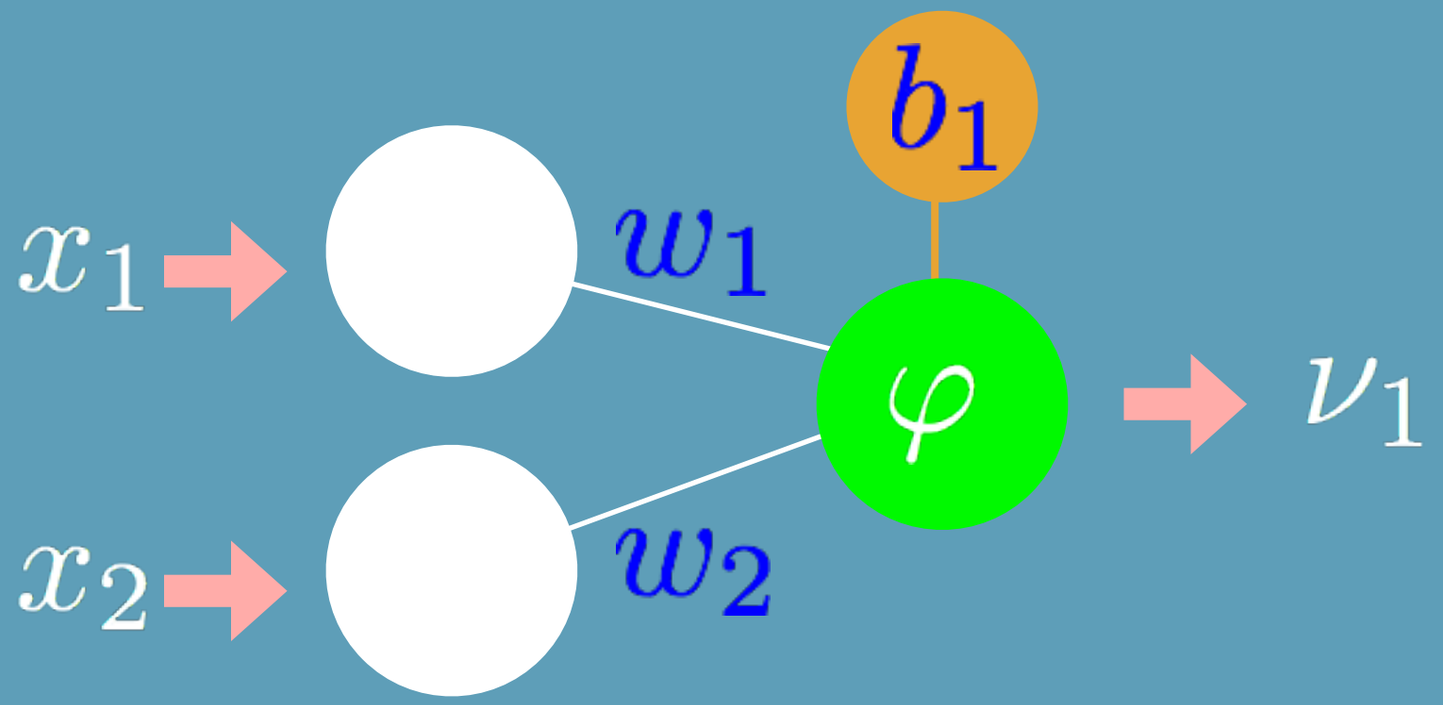
$$w_1 x_1 + w_2 x_2$$

(2) 每個神經元有個 bias



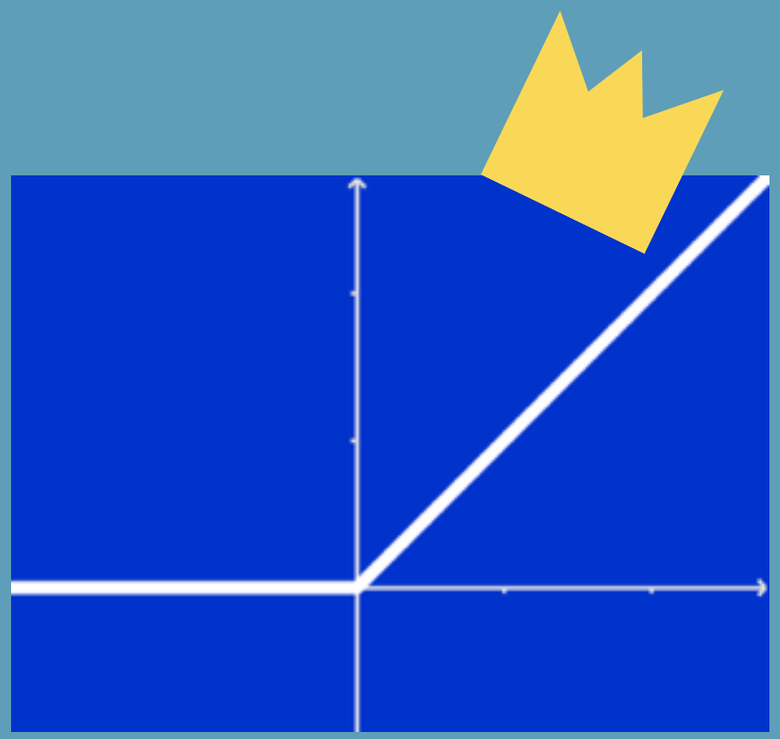
$$w_1 x_1 + w_2 x_2 + b_1$$

(3) 最後代入 activation function

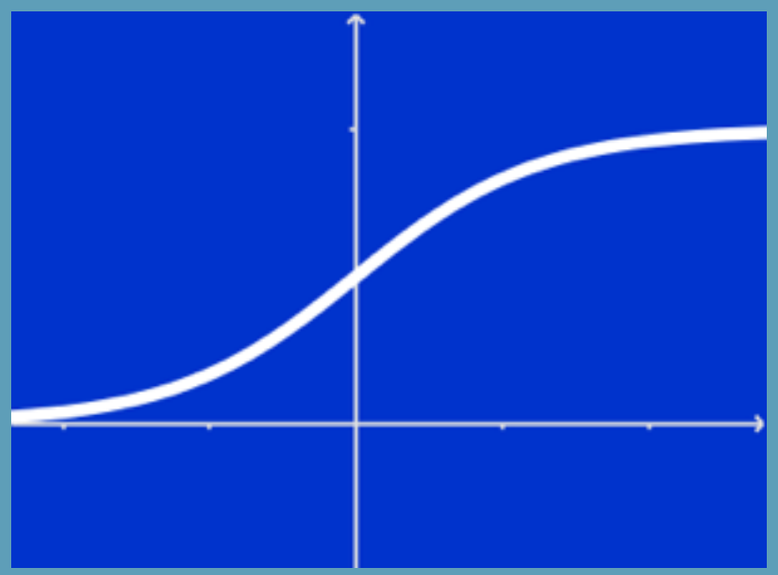


$$\varphi(w_1x_1 + w_2x_2 + b_1) = \nu_1$$

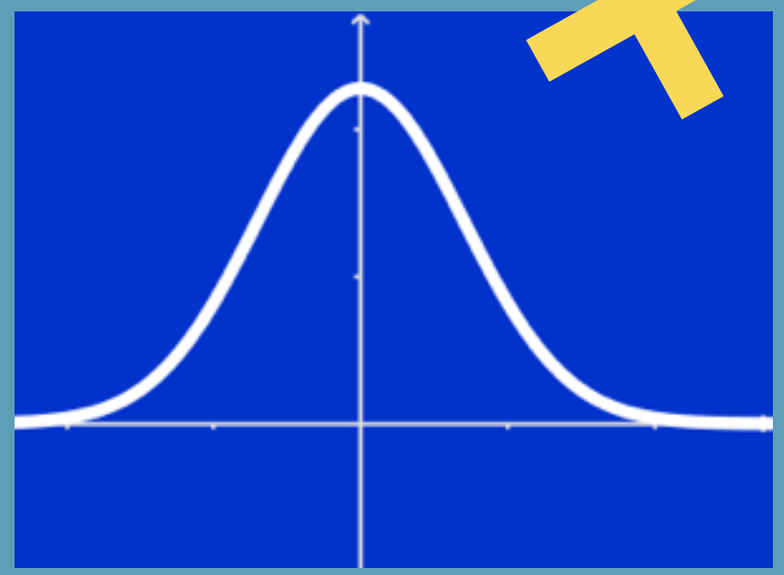
(3) 幾個 activation functions



ReLU

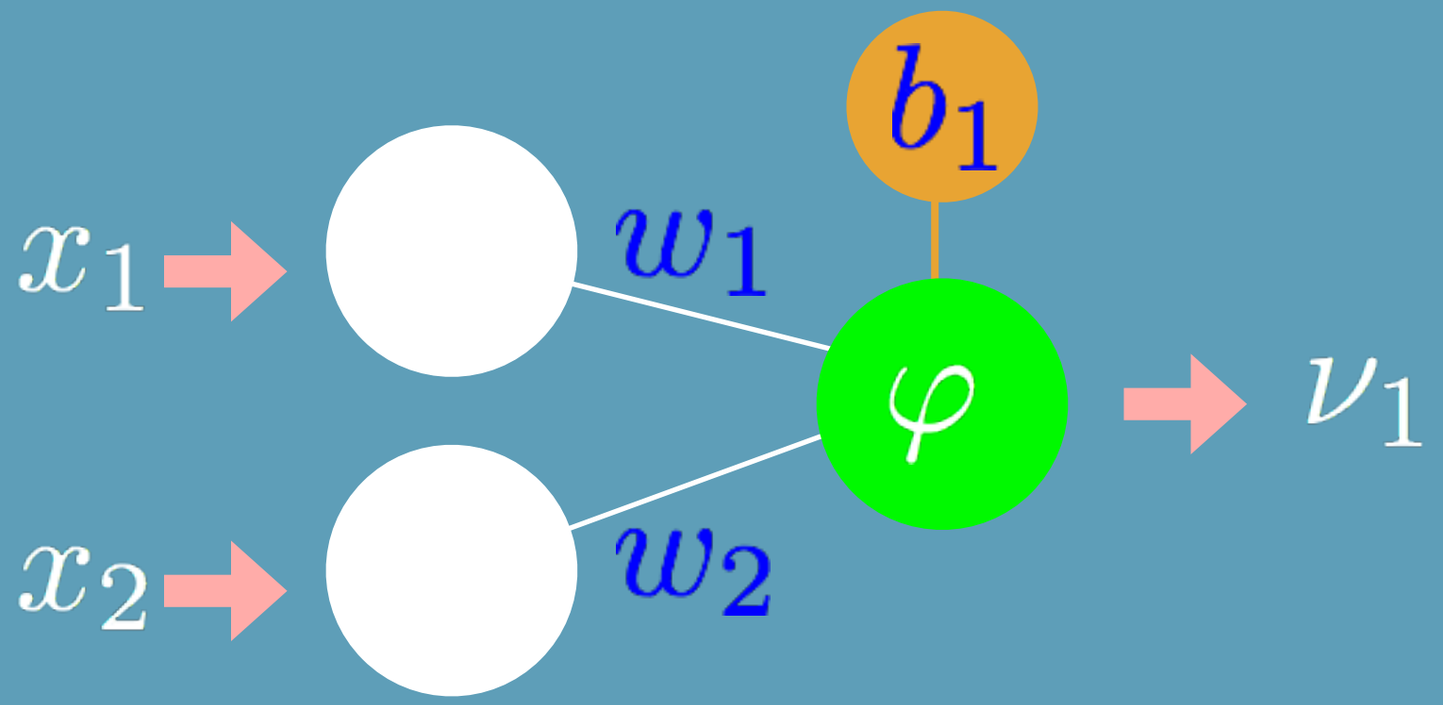


Sigmoid



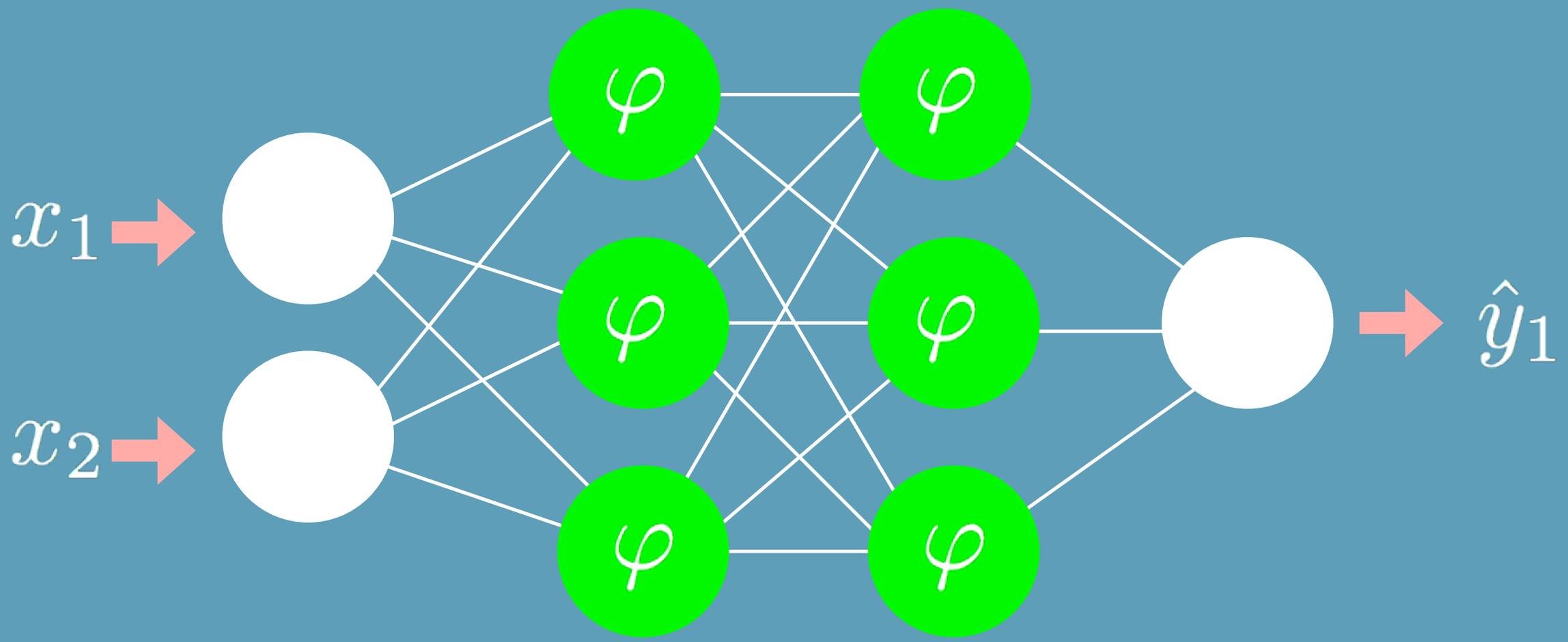
Gaussian

變數就是 weights, biases



$$\varphi(w_1 x_1 + w_2 x_2 + b_1) = \nu_1$$

[練習] 這樣子的神經網路變數有幾個呢?



固定結構神經網路的函數空間

當一個神經網路結構決定、activation functions 也決定, 那可以調的就是 weights, biases。我們把這些參數的集合叫 θ, 每一個 θ 就定義一個函數, 我們把它看成一個集合。

$$\{F_\theta\}$$

我們就是要找θ^*

使得F_{θ^*}和目標函數最接近

「最近」是什麼意思

就是“loss function”最小

假設我們有訓練資料

$$\{(\mathbf{x}_1, \mathbf{y}_1), (\mathbf{x}_2, \mathbf{y}_2), \cdots, (\mathbf{x}_k, \mathbf{y}_k)\}$$

最常用 loss function

$$L(\theta) = \frac{1}{2}\sum_{i=1}^{k} \|\mathbf{y}_i - F_\theta(\mathbf{x}_i)\|^2$$

我們希望它越小越好

基本上這樣調

learning rate

$$\eta \frac{\partial L}{\partial w_{ij}}$$

有個很潮的名字

Gradient Descent

你也會常聽到

Backpropagation

一層 hidden layer 的神經網路就是 Universal Approximator!

設 $\varphi: X \to Y$ 是一個有界、非常數連續函數, 其中 $X \subset \mathbb{R}^n$, 且 $Y \subset \mathbb{R}^m$ (φ 將是我們的 activation fuction)。設 I_n 是一個 n 維超立方體, 且 $\mathcal{C}(I_n)$ 為所有定義在 I_n 上的連續函數。則對於任意的 $f \in \mathcal{C}(I_n)$, 與任意的 $\varepsilon > 0$, 都存在 $N > 0$, 及

$$\alpha_i, \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix}, b_i, i = 1, 2, \ldots, N$$

使得

$$F(\mathbf{x}) = \sum_{i \leq N} \alpha_i \varphi(\mathbf{w}^t \mathbf{x} + b_i) \quad \sup_{\mathbf{x} \in I_n} |f(x) - F(x)| < \varepsilon$$

但是突然大家沒興趣了

差不多近 2000 年

Deep Learning

神經網路再現!

關鍵

CNN / RNN / Deep

CNN

Convolutional Neural Network

1 convolutional layer

2 max-pooling layer

CNN 通常有這兩種
hidden layers

1 Convolutional Layer

我們要做 filters

例如 3x3 的大小

內積

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

filter

這學來的

2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	3	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

35					

想成這是一張圖所成的矩陣

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

filter

還是一樣的矩陣

2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	2	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

右移一格

35	**27**				

filter

$$W = \begin{bmatrix} 1 & 0 & 1 \\ 2 & 1 & 3 \\ 1 & 1 & 2 \end{bmatrix}$$

2	5	5	2	5	2	0	1
2	3	4	0	4	2	1	5
4	3	1	3	5	5	4	3
5	3	4	5	0	2	1	5
2	3	1	1	1	0	1	3
4	4	1	1	5	1	1	4
2	3	2	2	0	4	2	4
0	5	4	5	3	4	1	4

35	27	44	32	36	38
36	36	37	36	36	43
37	37	23	26	17	35
29	25	22	18	14	27
27	25	24	21	24	32
31	38	27	34	25	**40**

一路到最後

- 最後就是一個 6x6 的矩陣
- 有時我們會把它弄成還是 8x8
- 基本上和原矩陣一樣大
- 而且我們通常 filter 會很多!

35	27	44	32	36	38
36	36	37	36	36	43
37	37	23	26	17	35
29	25	22	18	14	27
27	25	24	21	24	32
31	38	27	34	25	40

2 Max-Pooling Layer

基本上就是「投票」

看我們要多大區選一個代表，例如 2x2

35	27	**44**	32	36	38
36	**36**	37	36	36	**43**
37	37	23	**26**	17	**35**
29	25	22	18	14	27
27	25	24	21	24	32
31	**38**	27	**34**	25	**40**

36	**44**	**43**
37	**26**	**35**
38	**34**	**40**

每區投出最大的!!

可以不斷重覆

convolution, max-pooling, convolution,
max-pooling

做完再送到「正常的」
神經網路

ImageNet 2012 冠軍

AlexNet 拿下，之後大家幾乎都用某種形式的 ConvNet

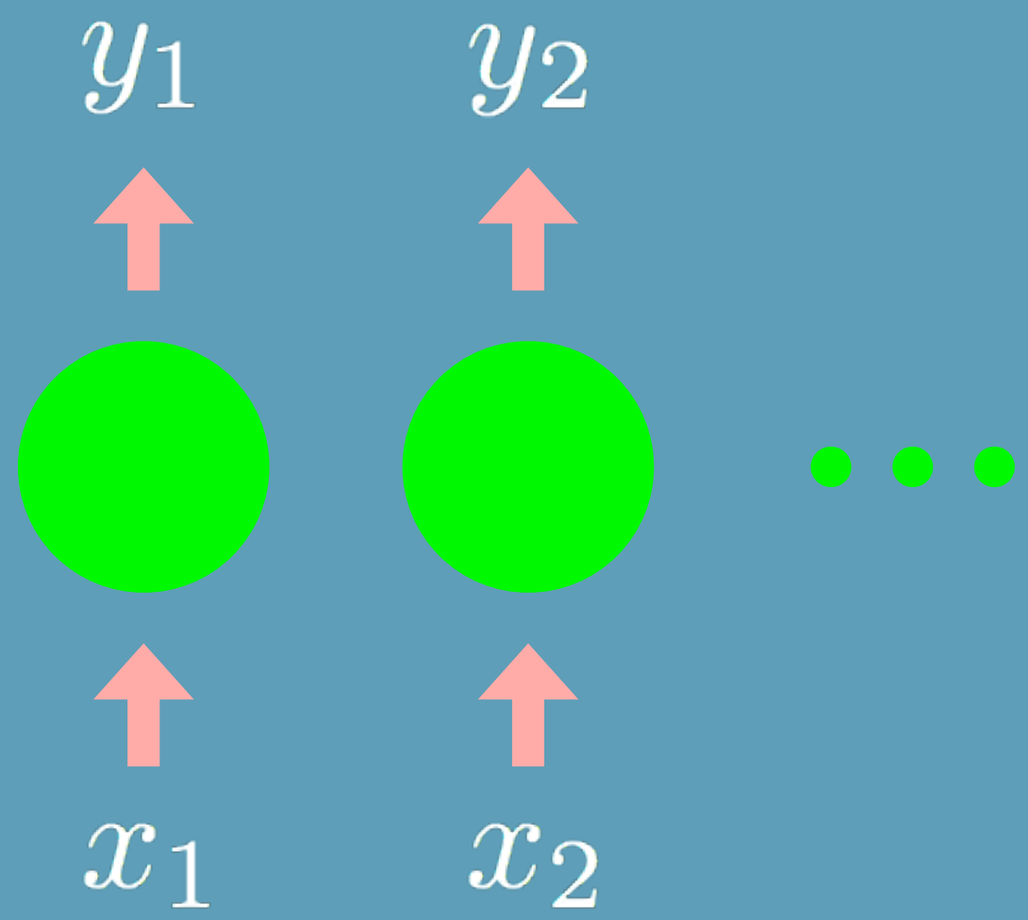
RNN

Recurrent Neural Network

有記憶的神經網路

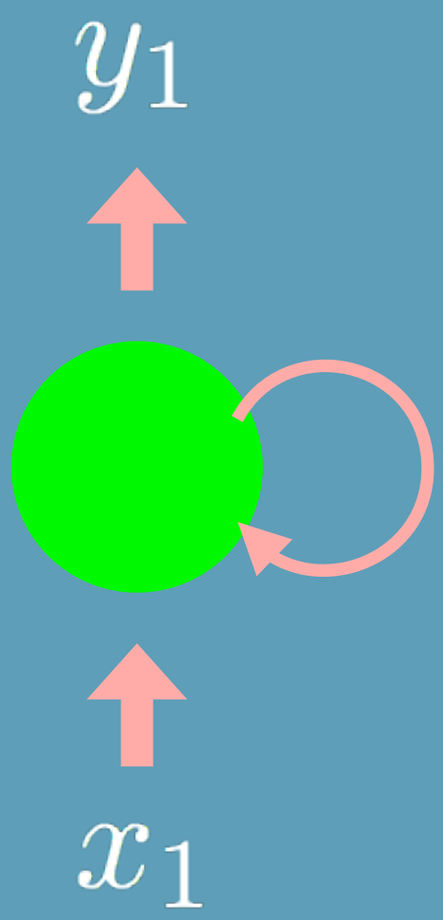
一般的神經網路

一筆輸入和下一筆是沒有關係的...

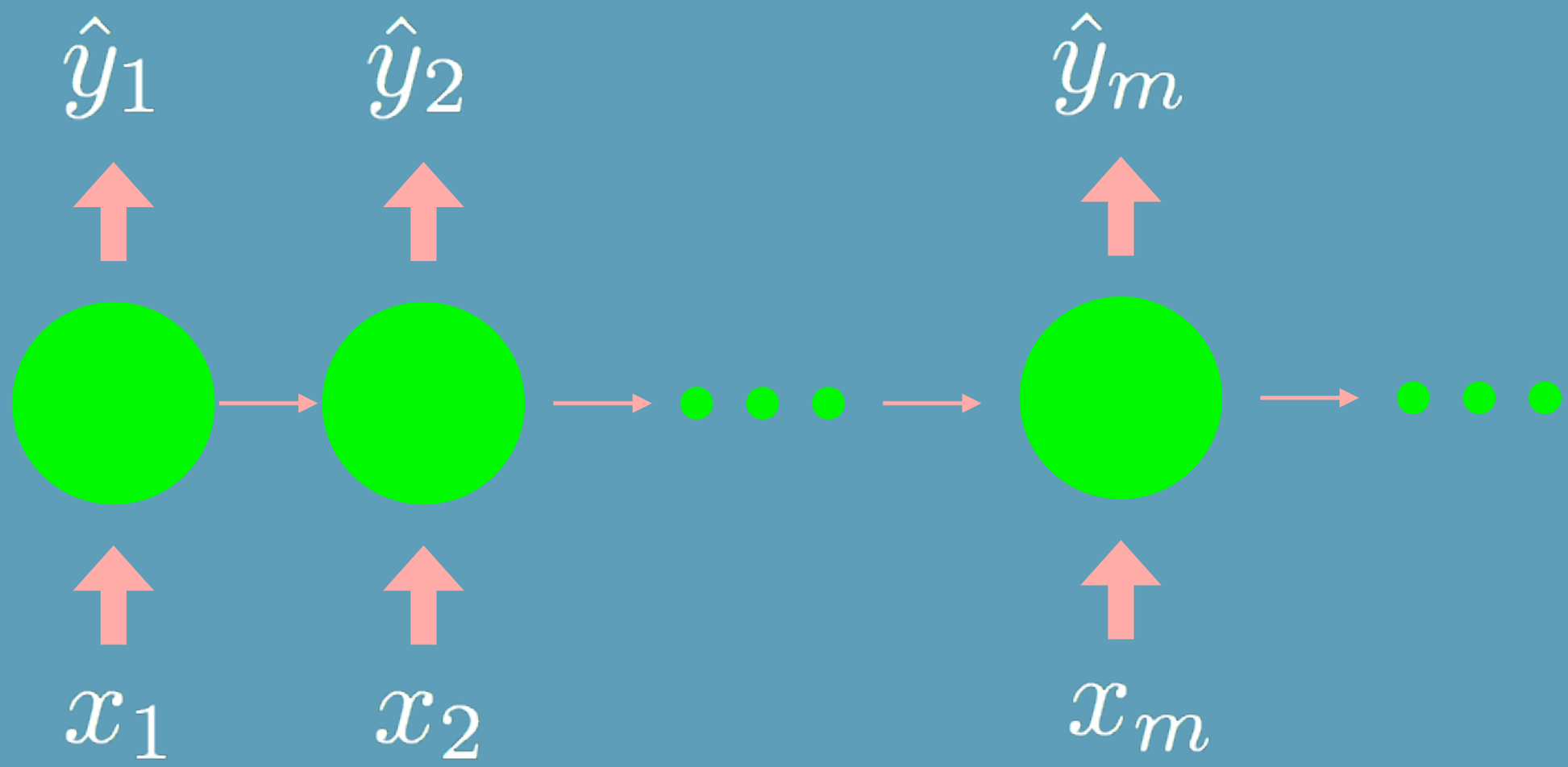


RNN

有「記憶」的神經網路

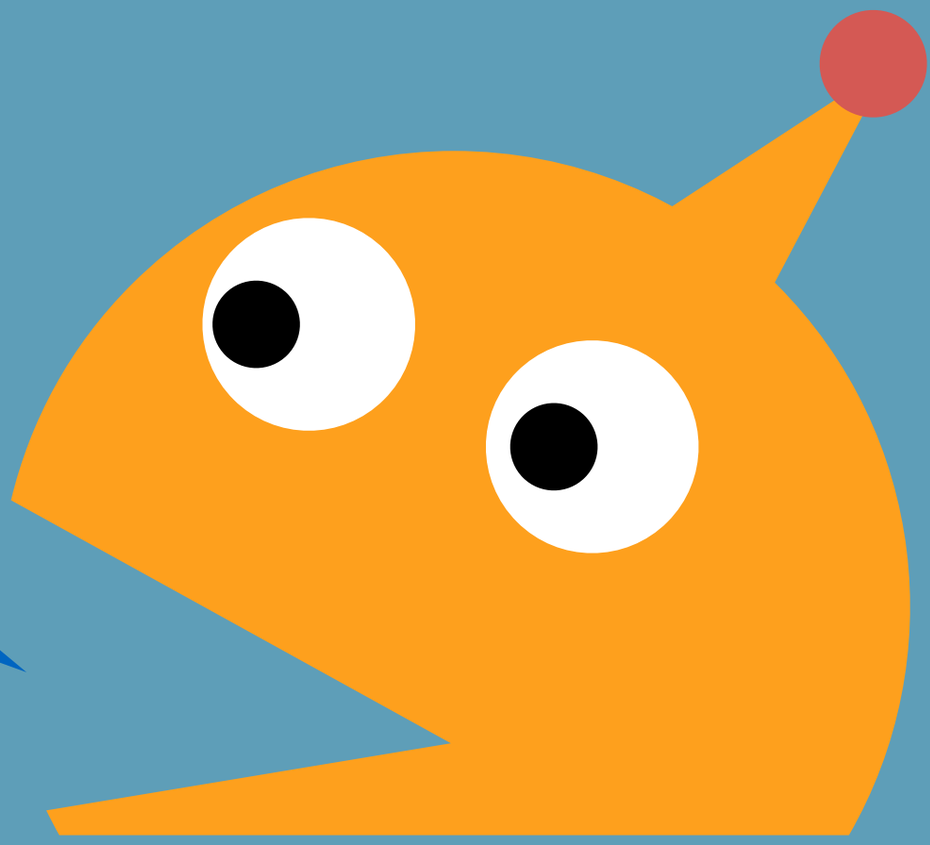


很多人畫成這樣

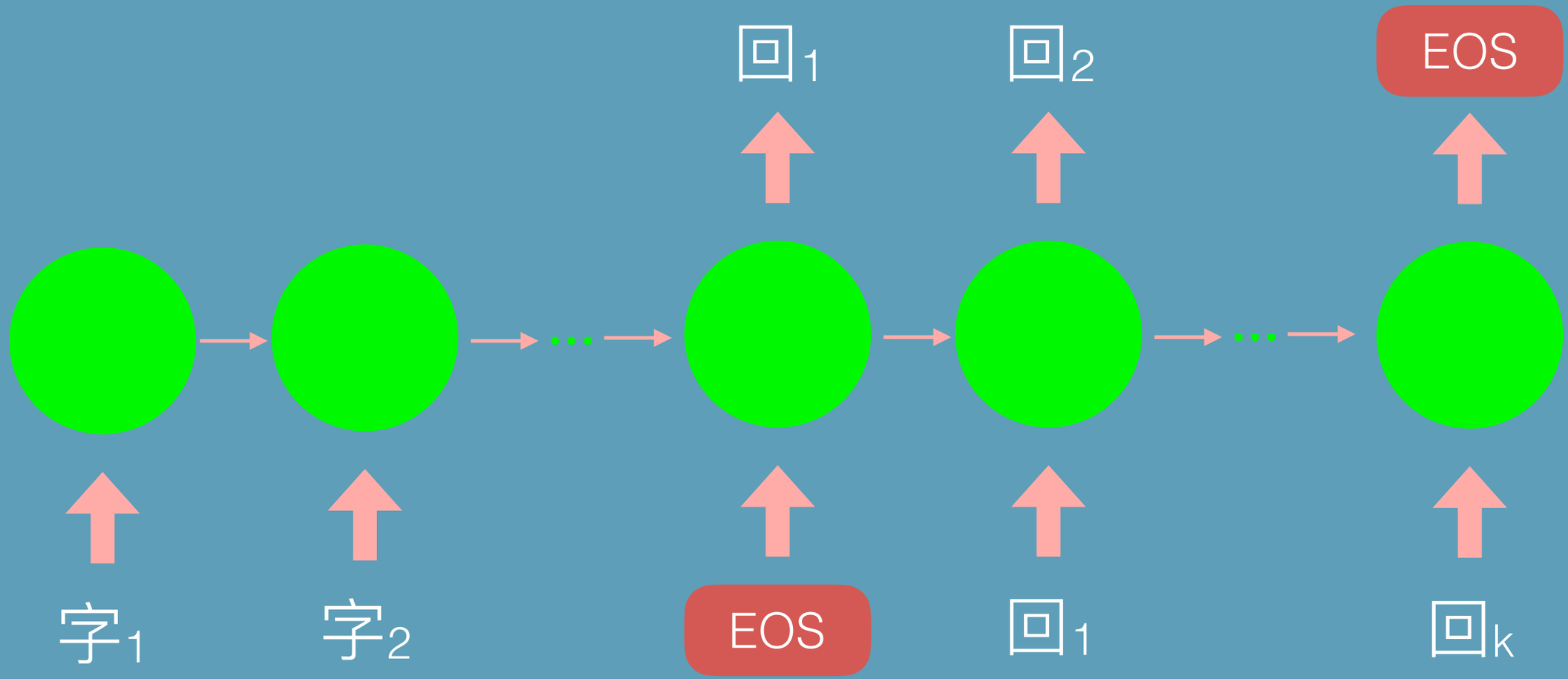


這能有什麼用?

對話機器人



Sequence-to-Sequence
Neural Network


1
2
EOS
字1
字2
EOS
回1
回k

其實我有點騙你

RNN 很不容易訓練

Vanishing

Gradient

LSTM

Long Short Term Memory

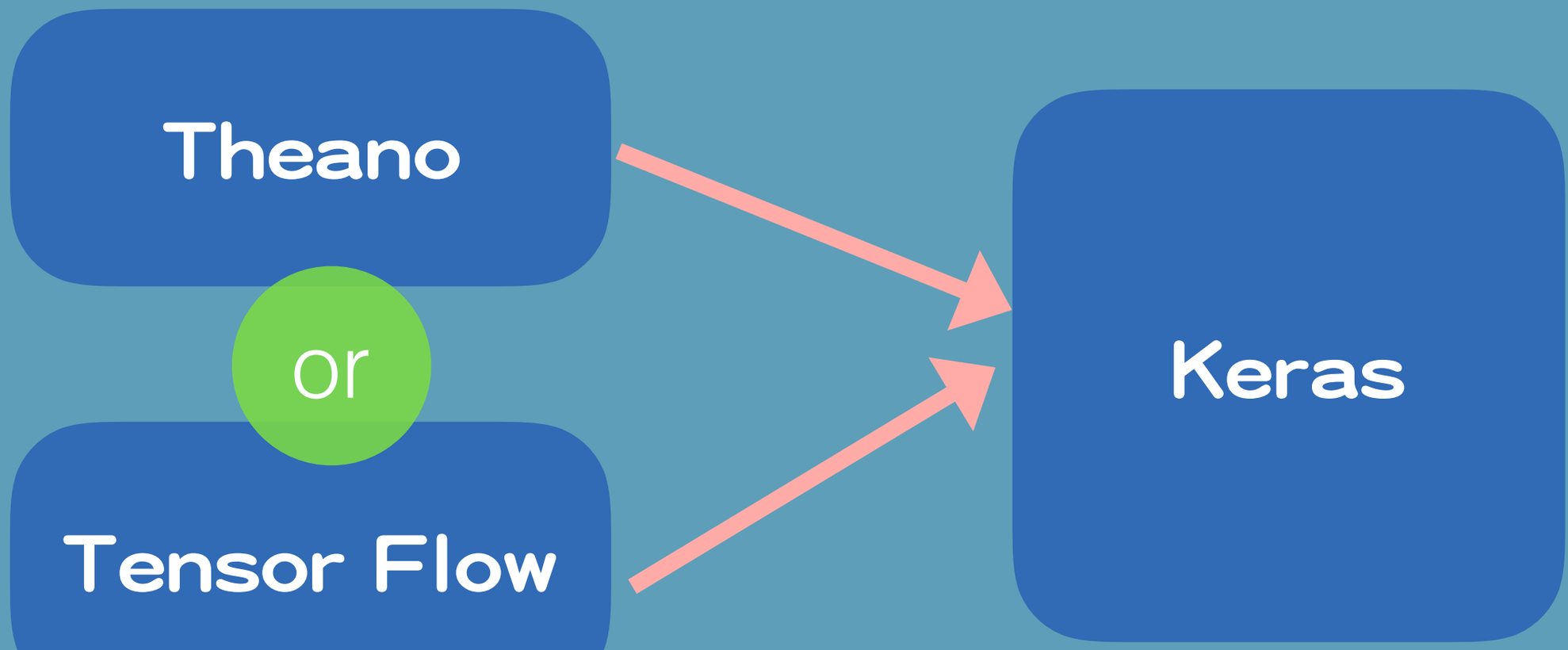
GRU

Gated Recurrent Unit

Deep Learning 心法

1. 要相信電腦學得會
2. 要有夠多的訓練資料
3. Dropout 避免 overfitting

Keras 瞬間完成神經網路



Keras 很方便、快速完成神經網路。

在 Mac 裝很容易

http://goo.gl/Z1RS2K

這裡只秀給你看很簡單

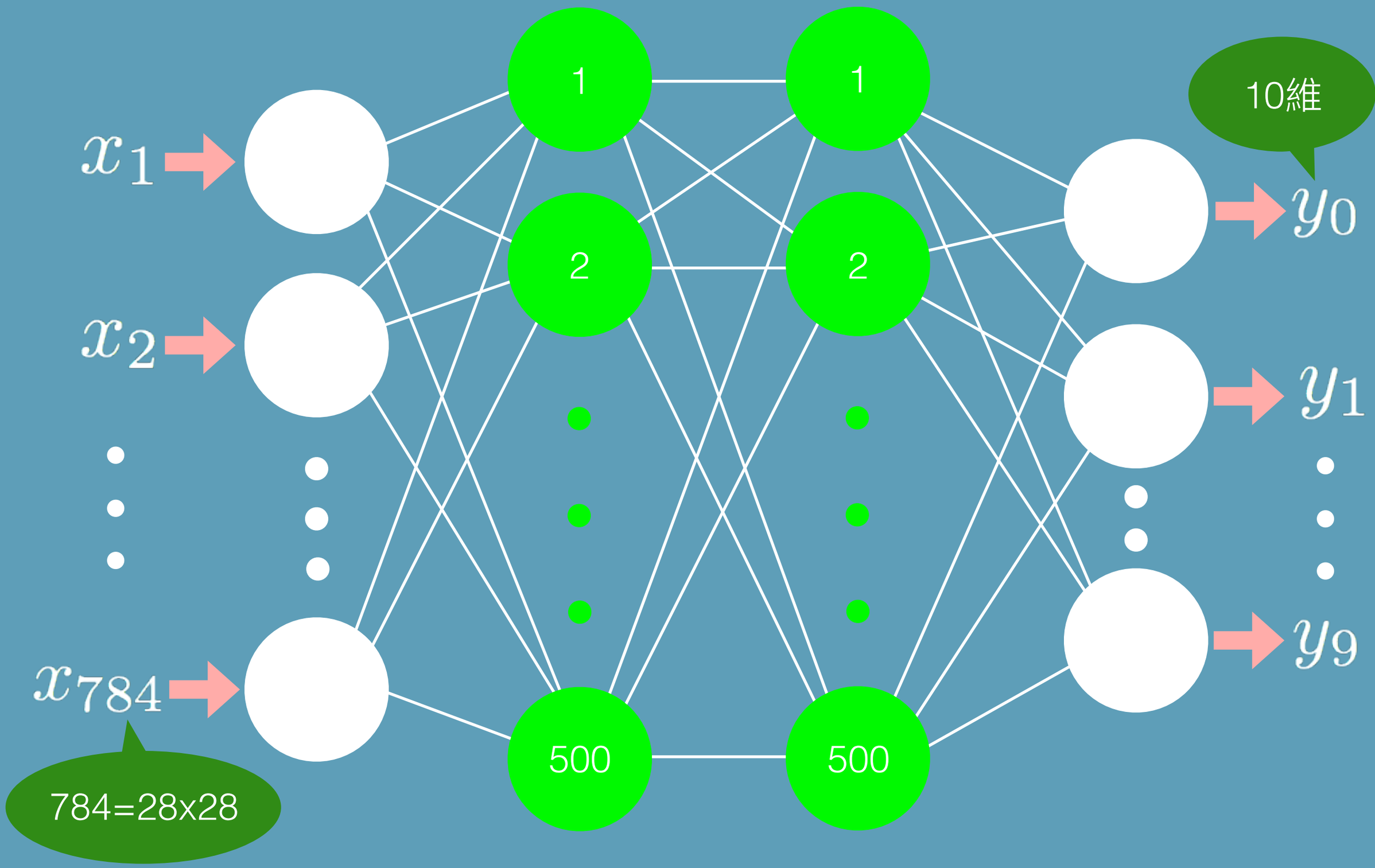
如果你有寫過神經網路的程式會哭

例如我們要做手寫辨識

雖然現在大家都用 CNN, 但我們借李宏毅
老師第一個例子用最傳統的

架構是這樣

- 輸入層有 28x28 = 784 個節點
- 兩個隱藏層, 各有 500 個節點
- 10 個輸出節點


x_1
x_2
x_{784}
784=28x28
1
2
500
1
2
500
10維
y_0
y_1
y_9
Input Layer
Hidden Layer
Output Layer

開始

和很多機器學習一樣, 弄個機器出來, 「正常」的 feedforward 神經網路就用 Sequential。

```
model = Sequential()
```

輸入層+隱藏層1

是不是很簡單, 基本上只要指出有幾個節點, 選哪個 activation function 就好。

```
model.add(Dense(input_dim=28*28, output_dim=500))
model.add(Activation('sigmoid'))
```

隱藏層2

輸入就是 500 個不用說, 直接說要輸出多少個, 也就是多少個節點。

```
model.add(Dense(output_dim=500))
model.add(Activation('sigmoid'))
```

輸出層

10 個輸出, 分別代表辨識為 0, 1, 2, ⋯, 9 的機率。

```
model.add(Dense(output_dim=10))
model.add(Activation('softmax'))
```

然後就組裝

說一下要用什麼當 loss function, 還有學習法。

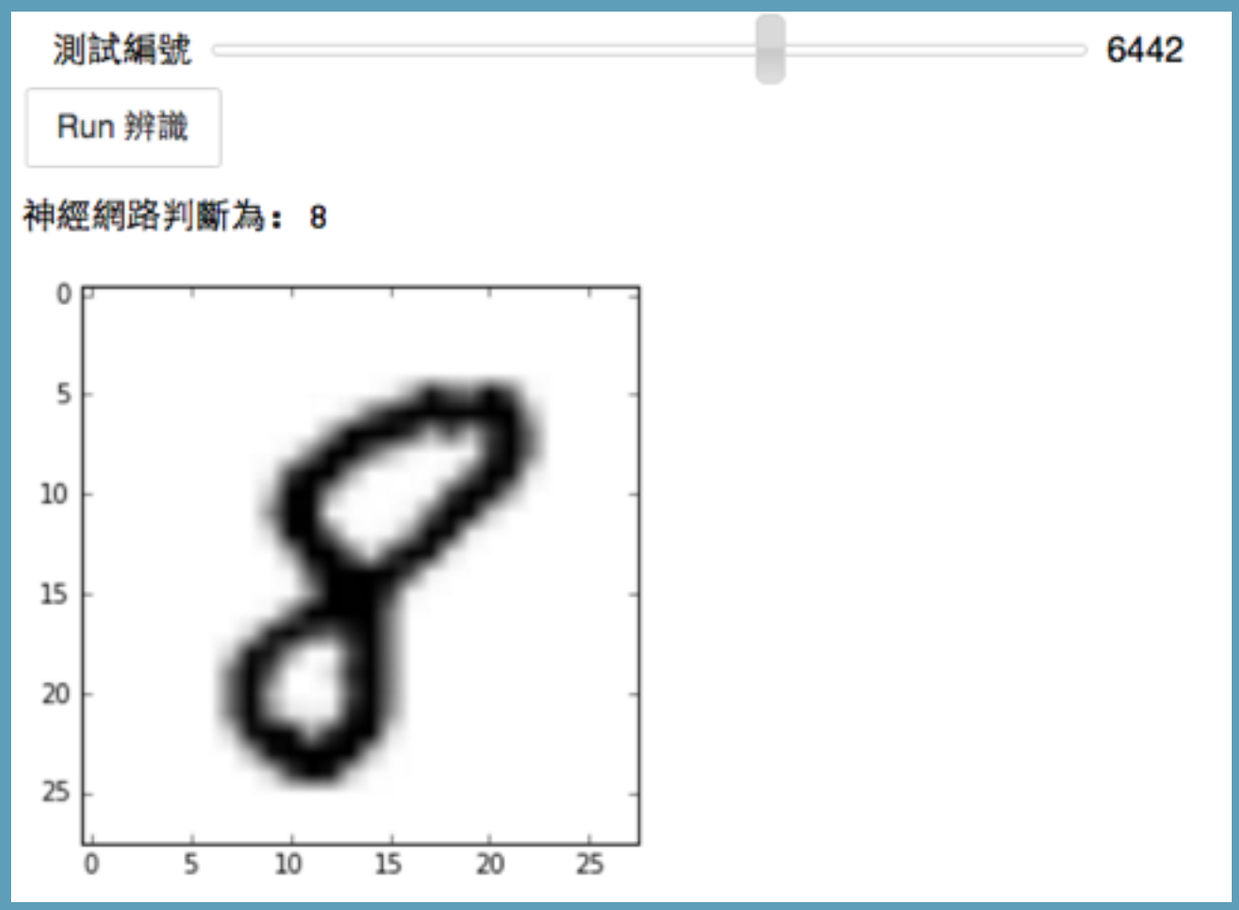
```
model.compile(loss='mse', optimizer=SGD(lr=0.1),
metrics=['accuracy'])
```

基本上就好了

當然這只有架構部份，詳請麻煩看 code

檢測學習成果

又可以用 Jupyter 的互動功能, 快速做出一個 GUI 界面!



李宏毅老師很棒的介紹

一天搞懂深度學習

http://goo.gl/appfNa