

Gradient descent 導傳遞 Weight Initialization

黃志勝 (Tommy Huang)

義隆電子 人工智慧研發部

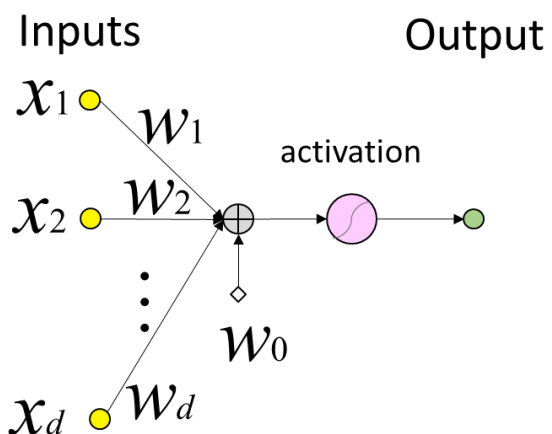
國立陽明交通大學 AI學院 合聘助理教授

國立台北科技大學 電資學院合聘助理教授



Introduction

神經網路求解方式為利用倒傳遞(back-propagation)方式來更新權重。



權重就是 $w_0, w_1, w_2, \dots, w_d$

怎麼用Gradient descent求解?

此份投影片會利用一些簡單的解釋怎麼利用導數(Derivative)/梯度(Gradient)求最佳解。



微積分求極值

一階微分=0找解，求得的解可能為最大或最小。
二階微分判斷，一階微分找到的解為最大或是最小。

$$f(x) = x^2 - 10x + 1$$

$$f'(x) = \frac{\partial f(x)}{\partial x} = 2x - 10 = 0$$
$$\Rightarrow x = 5$$

$$f''(x) = \frac{\partial f'(x)}{\partial x} = 2 > 0$$

此範例有 $x=5$ 有最小值-24。



微積分求極值

上述範例為close-form可以找到解。

$$f(\mathbf{x}) = 0.5x_1 + 2x_2 + 4x_3 + 10$$

$f(\mathbf{x})$ 為三元一次方程式，則此函數的梯度為一個向量方程式：

$$\nabla f = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \\ \frac{\partial f(\mathbf{x})}{\partial x_3} \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 4 \end{bmatrix}$$



微積分求極值

$f(x)$ 為一元多次方程式，其對一元的參數作微分稱為求此參數的導數 (Derivative)。

$$f'(x)$$

$f(\mathbf{x})$ 為多元多次方程式，其對多元的參數作微分稱為求此參數的梯度 (Gradient)。

$$\nabla f = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d} \end{bmatrix}$$



Hessian Matrix

剛提到梯度順便提一下Hessian Matrix

牛頓法求解用，但相對計算量大，目前還沒被廣泛使用。

$$H(\mathbf{x}) = \nabla^2 f = \nabla \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d} \end{bmatrix} = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1 \partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_1 \partial x_d} \\ \vdots & \ddots & \vdots \\ \frac{\partial f(\mathbf{x})}{\partial x_d \partial x_1} & \cdots & \frac{\partial f(\mathbf{x})}{\partial x_d \partial x_d} \end{bmatrix}$$



微積分求極值

上述範例為close-form可以找到解。

現實狀況

$$f(\mathbf{x}) = x_1^2 + x_1 - 4x_1x_2 + x_1^3x_2$$

$$\nabla f = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 + 1 - 4x_2 + 3x_1^2x_2 \\ x_1^3 - 4x_1 \end{bmatrix}$$



微積分求極值

上述範例為close-form可以找到解。

現實狀況

$$f(\mathbf{x}) = x_1^2 + x_1 - 4x_1x_2 + x_1^3x_2$$

$$\nabla f = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 + 1 - 4x_2 + 3x_1^2x_2 \\ -4x_1 + x_1^3 \end{bmatrix} = 0$$

$$x_1 = 0, -2, 2$$

$$(x_1, x_2) = (0, 0.25), (-2, 0.125), (2, -0.625)!$$



微積分求極值

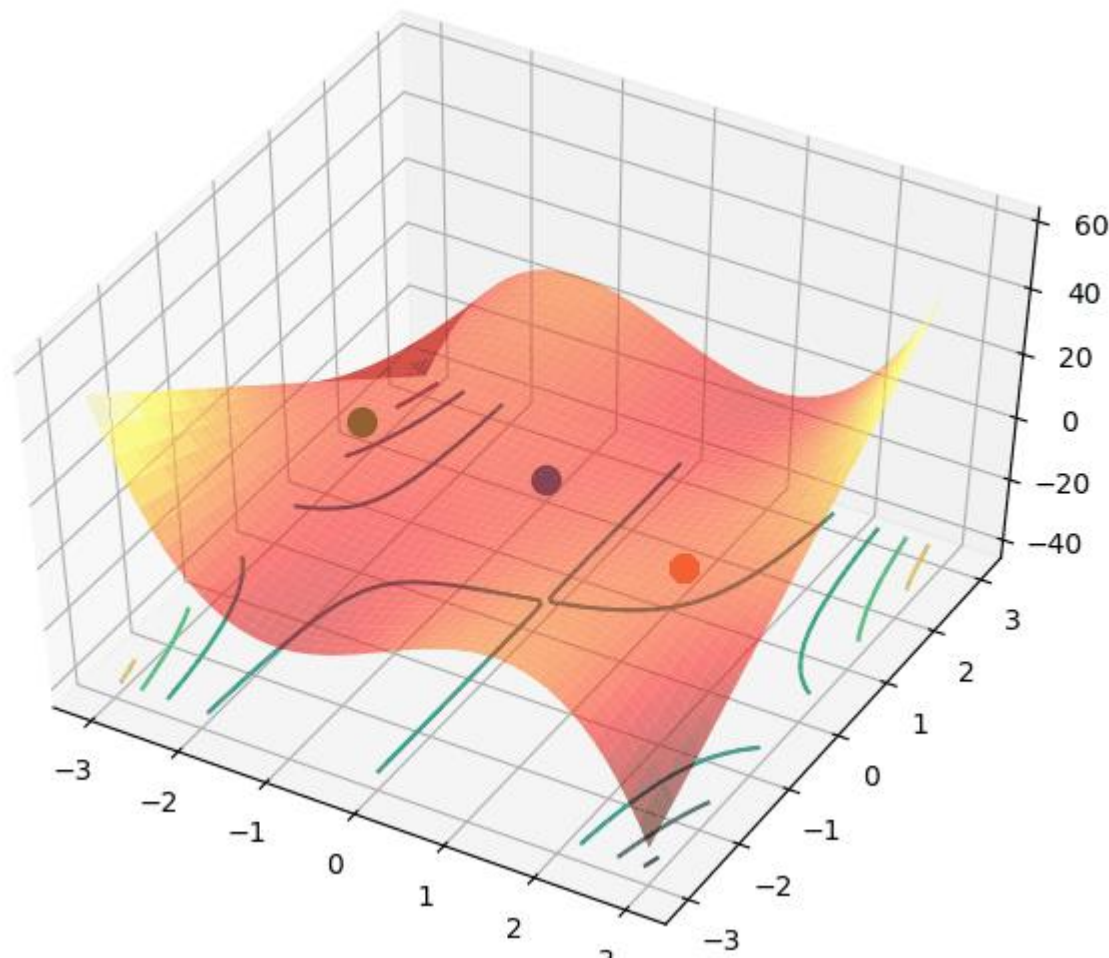
$$f(\mathbf{x}) = x_1^2 + x_1 - 4x_1x_2 + x_1^3x_2$$

$$f(0, 0.25) = 0$$

$$f(-2, 0.125) = 2$$

$$f(2, -0.625) = 6$$

所以微分得到的解不一定是極值，有可能是鞍點、反曲點。



導數(Derivative)/梯度(Gradient)

- 假設有一個一元的函數為：

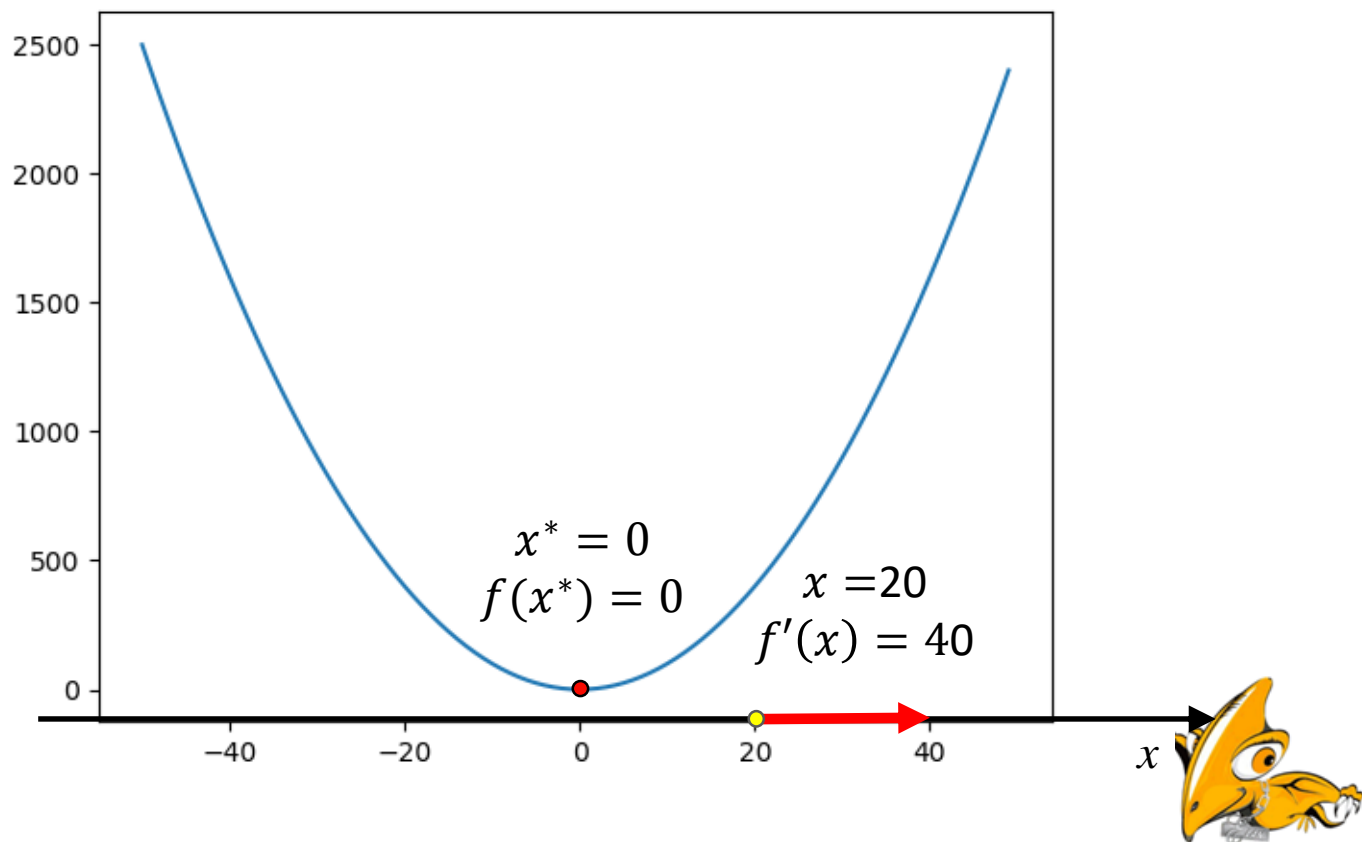
$$f(x) = x^2$$

其導數: $f'(x) = 2x$ (導數有方向性)

$x > 0 \rightarrow$ 往右(正)的方向

$x < 0 \rightarrow$ 往左(負)的方向

導數(Derivative)/梯度(Gradient)往極大值的方向走



導數(Derivative)/梯度(Gradient)

- 假設有一個一元的函數為:

$$f(x) = x^2$$

其導數: $f'(x) = 2x$ (導數有方向性)

$x > 0 \rightarrow$ 往右(正)的方向

$x < 0 \rightarrow$ 往左(負)的方向

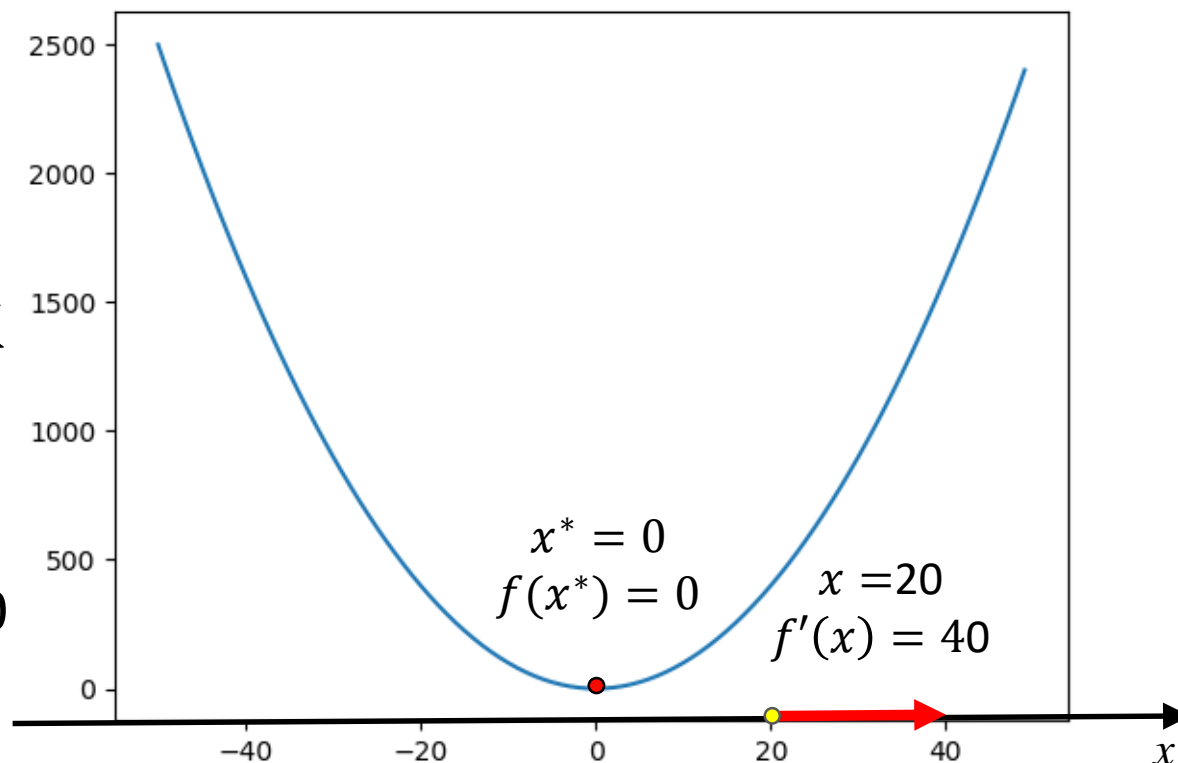
往導數(Derivative)/梯度(Gradient)往極大值的反方向走，則是往極小值

$$x^{(t+1)} = x^{(t)} - \alpha f'(x)$$

$$\alpha = 0.5$$

$$x^{(0)} = 20, f'(x) = 20, x^{(1)} = 20 - 10 = 10$$

梯度下降法(Gradient descent)



梯度(Gradient)

$$f(\mathbf{x}) = (x_1 - 5)^2 + (x_2)^2$$

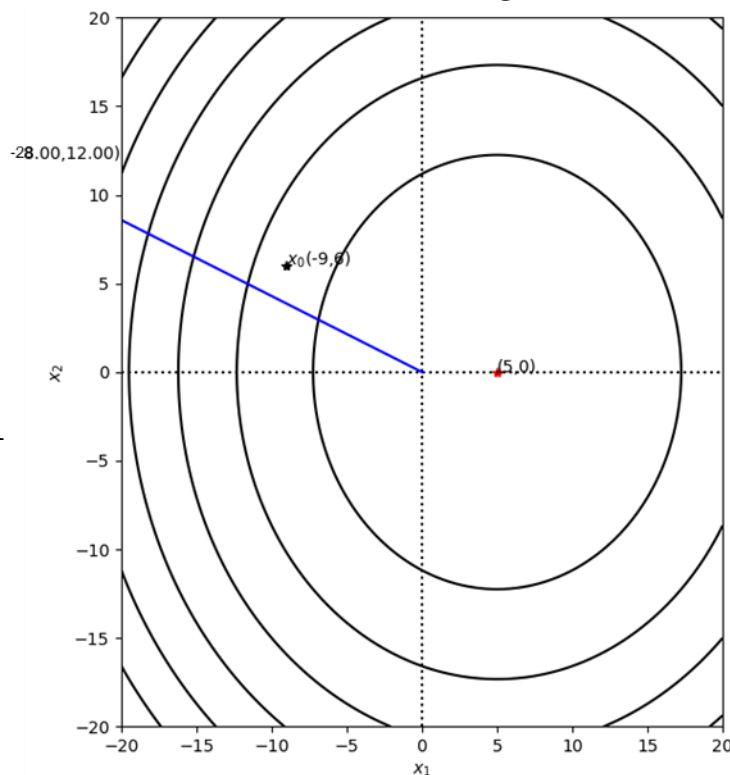
$$\nabla f = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 - 10 \\ 2x_2 \end{bmatrix}$$

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)}), \alpha = 0.1$$

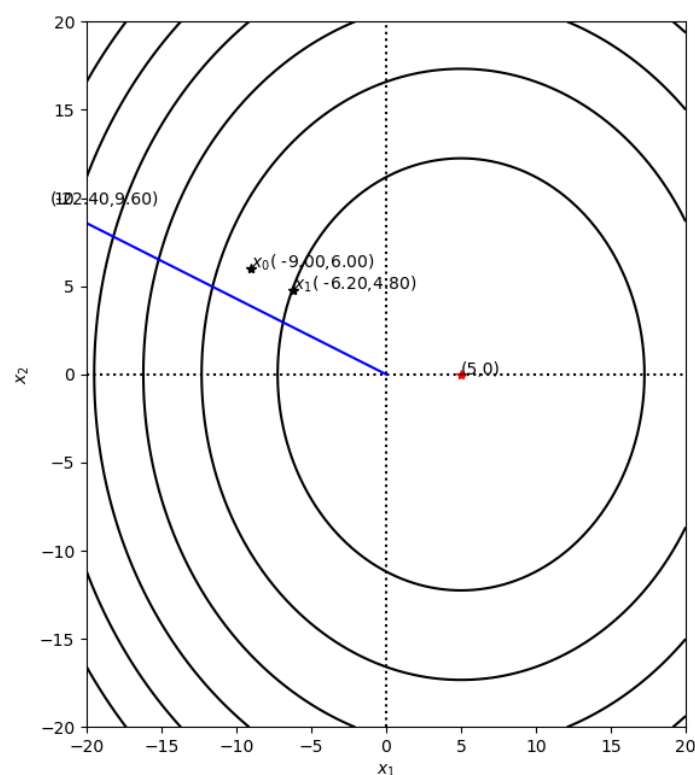
$$\mathbf{x}^{(0)} = \begin{bmatrix} -9 \\ 6 \end{bmatrix}, \nabla f(\mathbf{x}^{(0)}) = \begin{bmatrix} -28 \\ 12 \end{bmatrix}$$

$$\mathbf{x}^{(1)} = \begin{bmatrix} -6.2 \\ 4.8 \end{bmatrix}, \nabla f(\mathbf{x}^{(1)}) = \begin{bmatrix} -22.4 \\ 9.6 \end{bmatrix}$$

$$\mathbf{x}^{(0)} = \begin{bmatrix} -9 \\ 6 \end{bmatrix}$$



$$\mathbf{x}^{(1)} = \begin{bmatrix} -6.2 \\ 4.8 \end{bmatrix}$$



藍線為 f 的Gradient



梯度(Gradient)

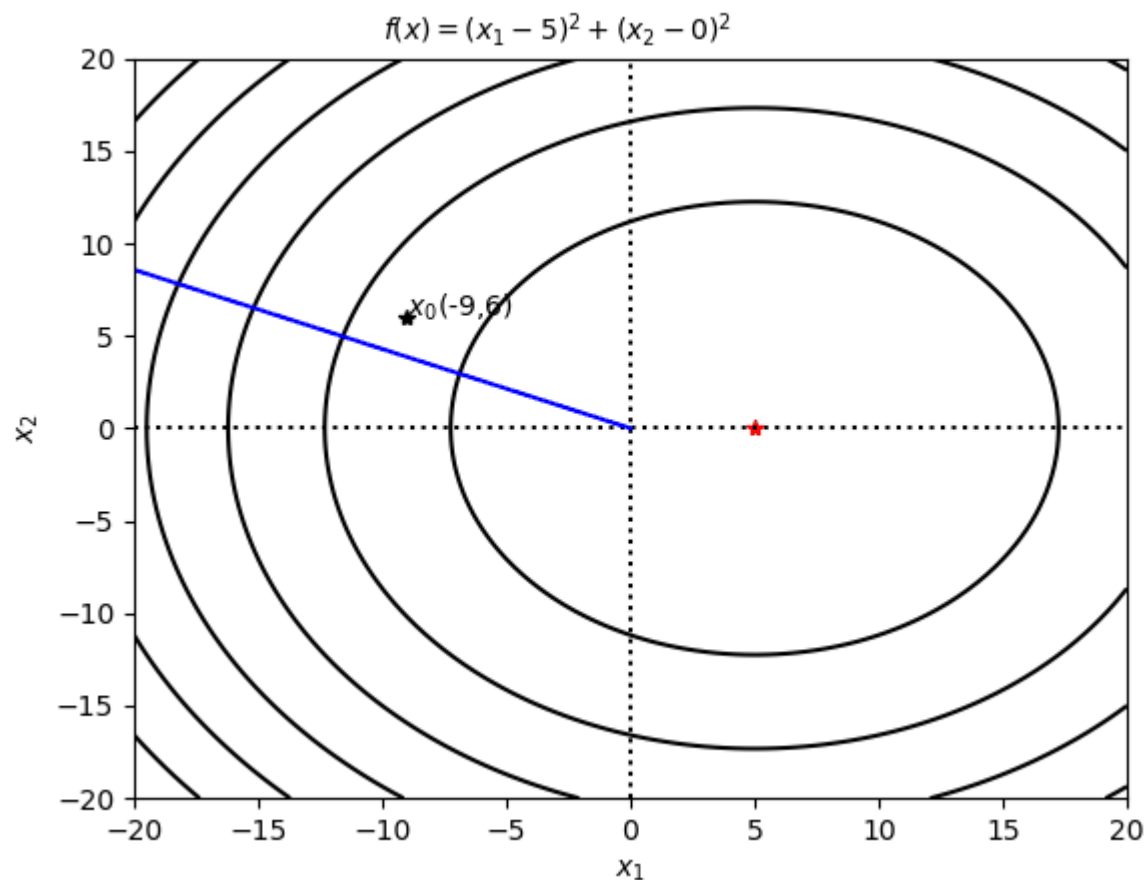
$$f(\mathbf{x}) = (x_1 - 5)^2 + (x_2 - 0)^2$$

$$\nabla f = \begin{bmatrix} \frac{\partial f(\mathbf{x})}{\partial x_1} \\ \frac{\partial f(\mathbf{x})}{\partial x_2} \end{bmatrix} = \begin{bmatrix} 2x_1 - 10 \\ 2x_2 \end{bmatrix}$$

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \alpha \nabla f(\mathbf{x}^{(t)}), \alpha = 0.1$$

$$\mathbf{x}^{(0)} = \begin{bmatrix} -9 \\ 6 \end{bmatrix}, \nabla f(\mathbf{x}^{(0)}) = \begin{bmatrix} -28 \\ 12 \end{bmatrix}$$

$$\mathbf{x}^{(1)} = \begin{bmatrix} -6.2 \\ 4.8 \end{bmatrix}, \nabla f(\mathbf{x}^{(0)}) = \begin{bmatrix} -22.4 \\ 9.6 \end{bmatrix}$$



藍線為 f 的Gradient



梯度下降法(Gradient descent)

梯度下降法(Gradient descent)公式:

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \gamma \nabla f(\mathbf{x}^{(t)})$$

t : 第 t 次迭代

∇f : 函數 f 的Gradient

γ : learning rate



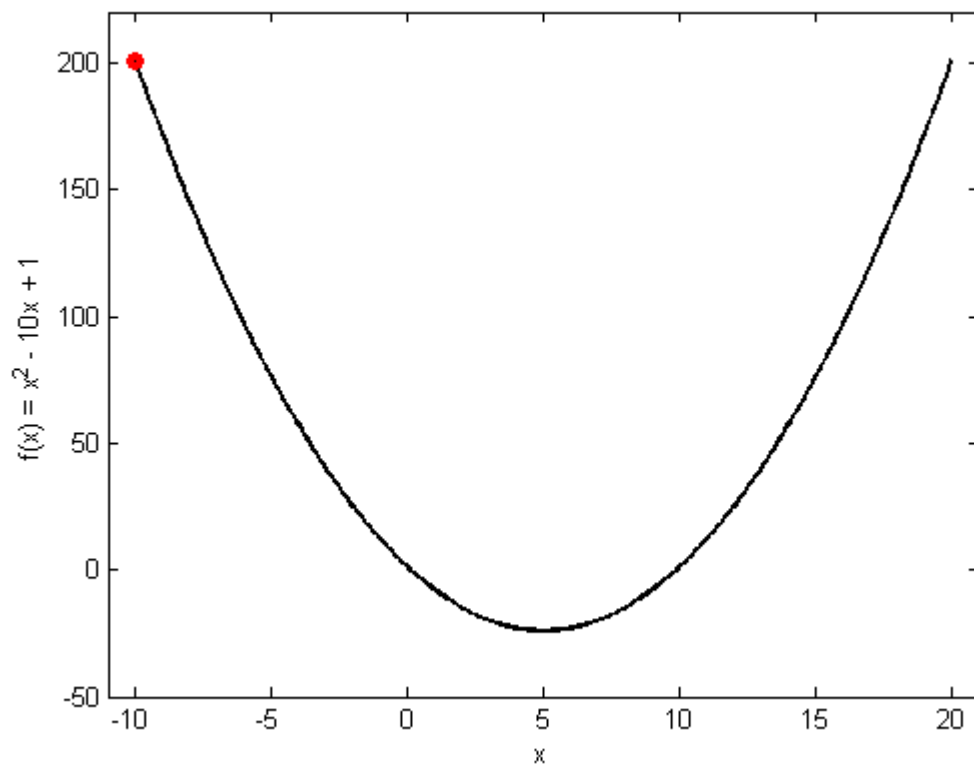
梯度下降法-範例

$$f(x) = x^2 - 10x + 1$$
$$f'(x) = 2x - 10$$

$$x^{(t+1)} = x^{(t)} - \gamma f'(x^{(t)})$$

$$\Rightarrow x^{(t+1)} = x^{(t)} - \gamma(2x^{(t)} - 10) = (1 - 2\gamma)x^{(t)} + 10\gamma$$

$$\Rightarrow x^{(t+1)} = (1 - 2\gamma)x^{(t)} + 10\gamma$$

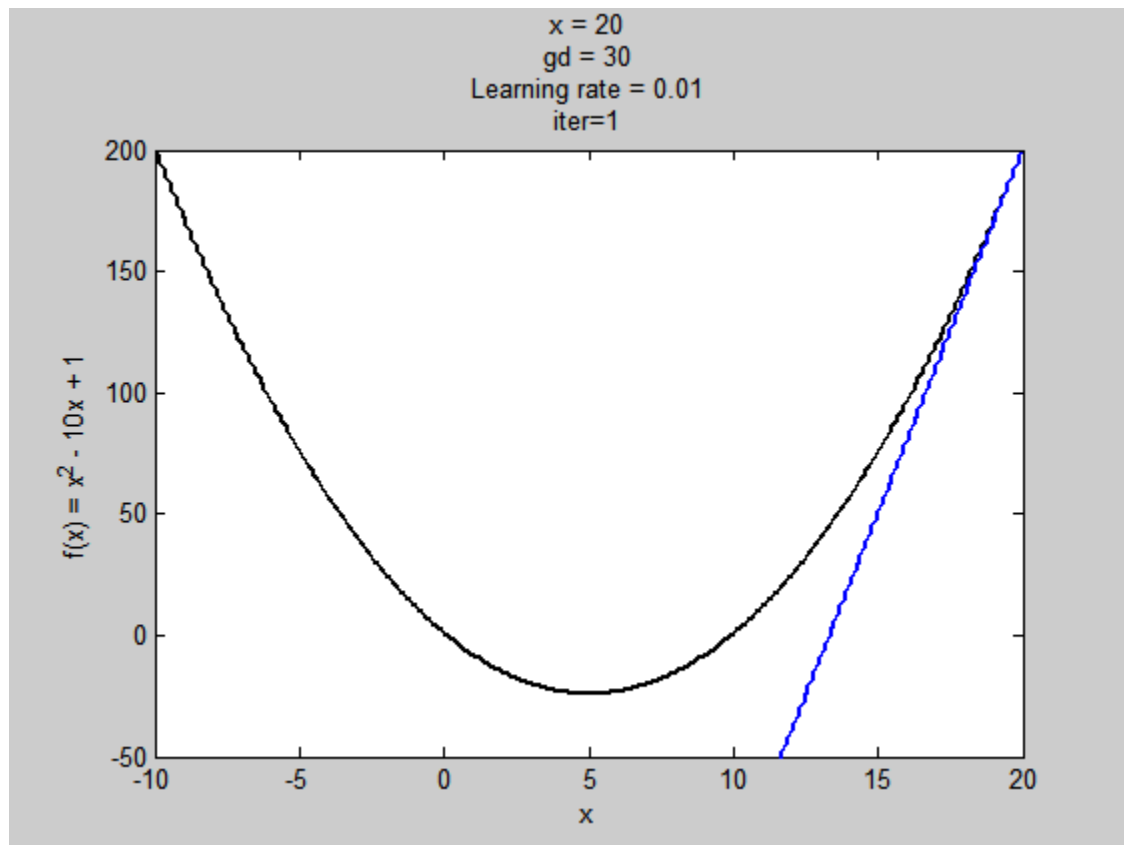


梯度下降法-範例

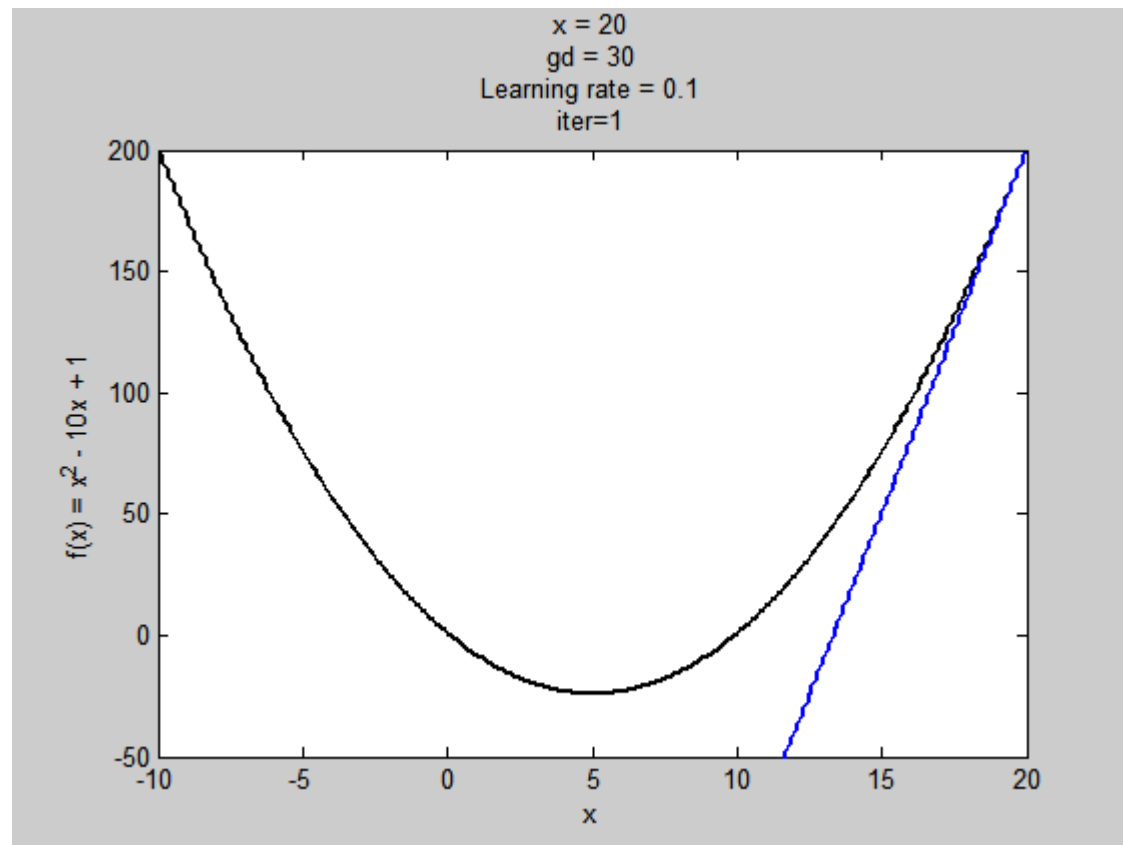
$$f(x) = x^2 - 10x + 1$$

**Learning rate 越
大越快找到解**

Learning rate = 0.01

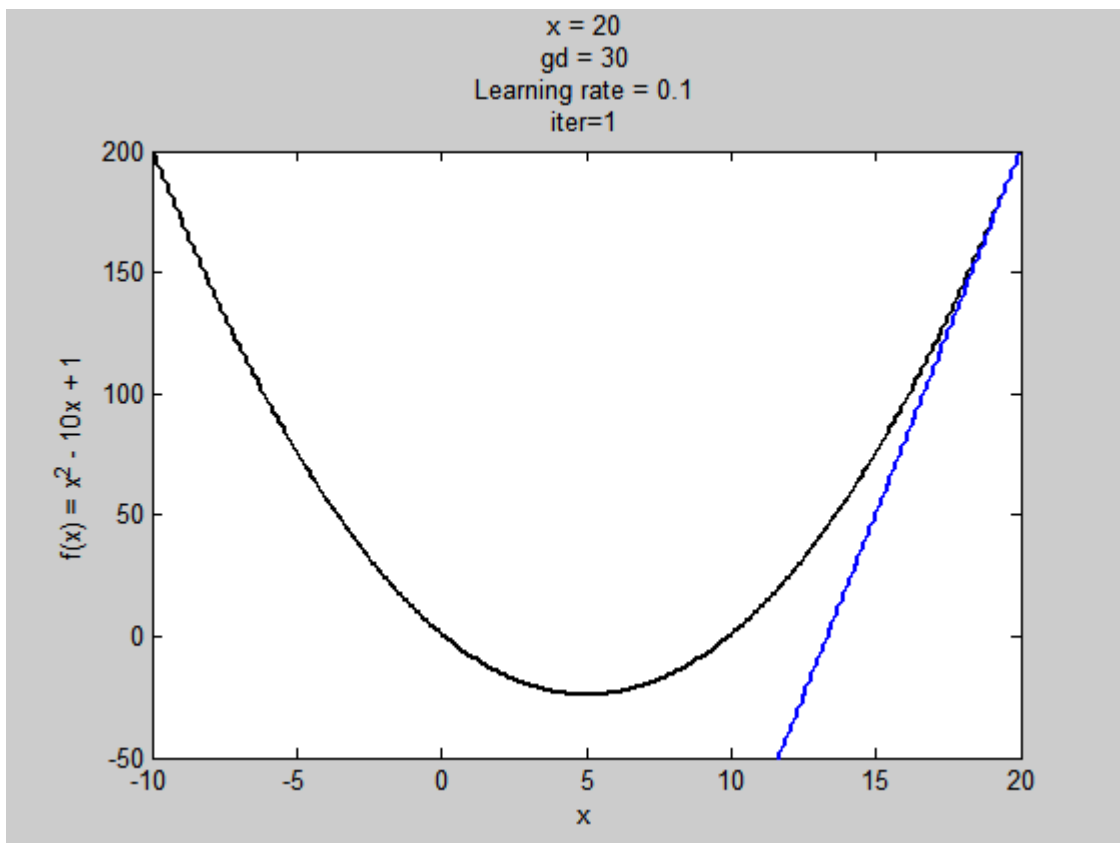


Learning rate = 0.1

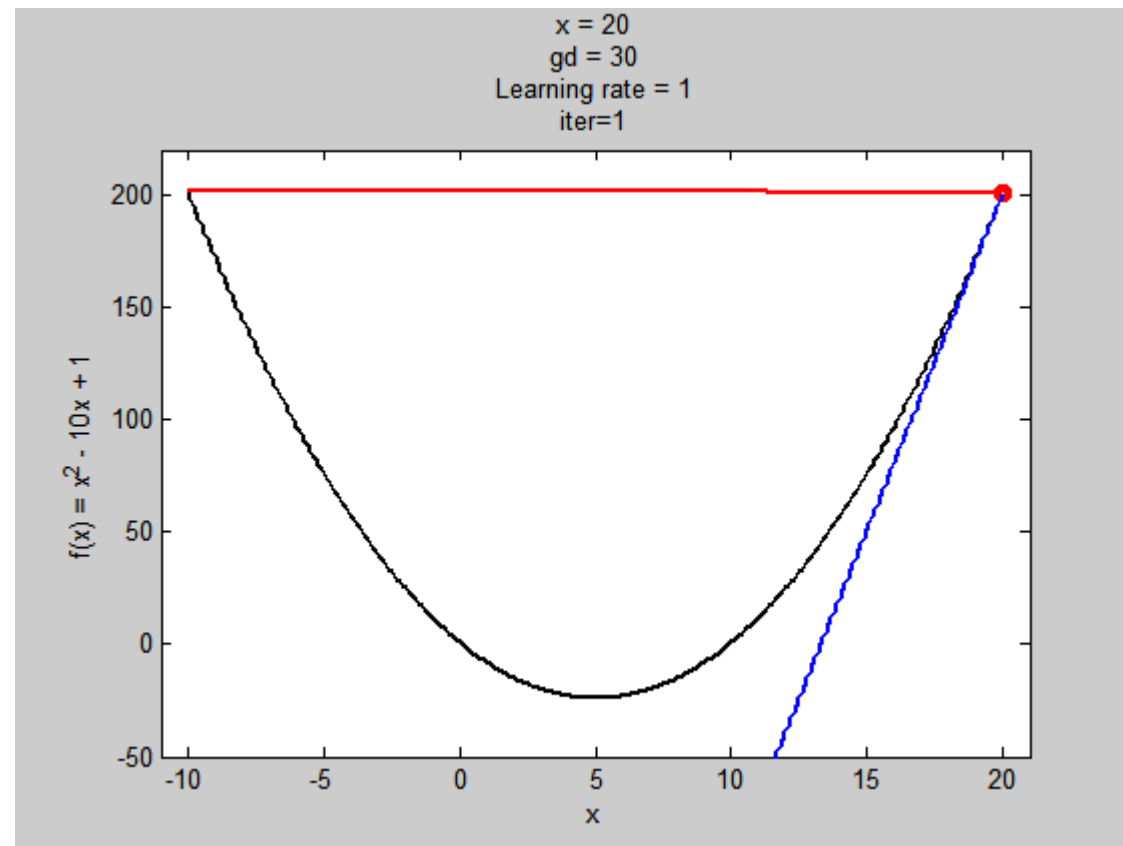


梯度下降法 (學習率過大)

Learning rate = 0.1



Learning rate = 1



梯度下降法-範例2

$$f(x) = x^4 - 50x^3 - x + 1$$

$$f'(x) = 4x^3 - 150x^2 - 1$$

$$x^{(t+1)} = x^{(t)} - \gamma f'(x^{(t)})$$

$$\Rightarrow x^{(t+1)} = x^{(t)} - \gamma (4x^{(t)3} - 150x^{(t)2} - 1)$$

$$\Rightarrow x^{(t+1)} = -4\gamma x^{(t)3} + 150\gamma x^{(t)2} + x^{(t)} + \gamma$$

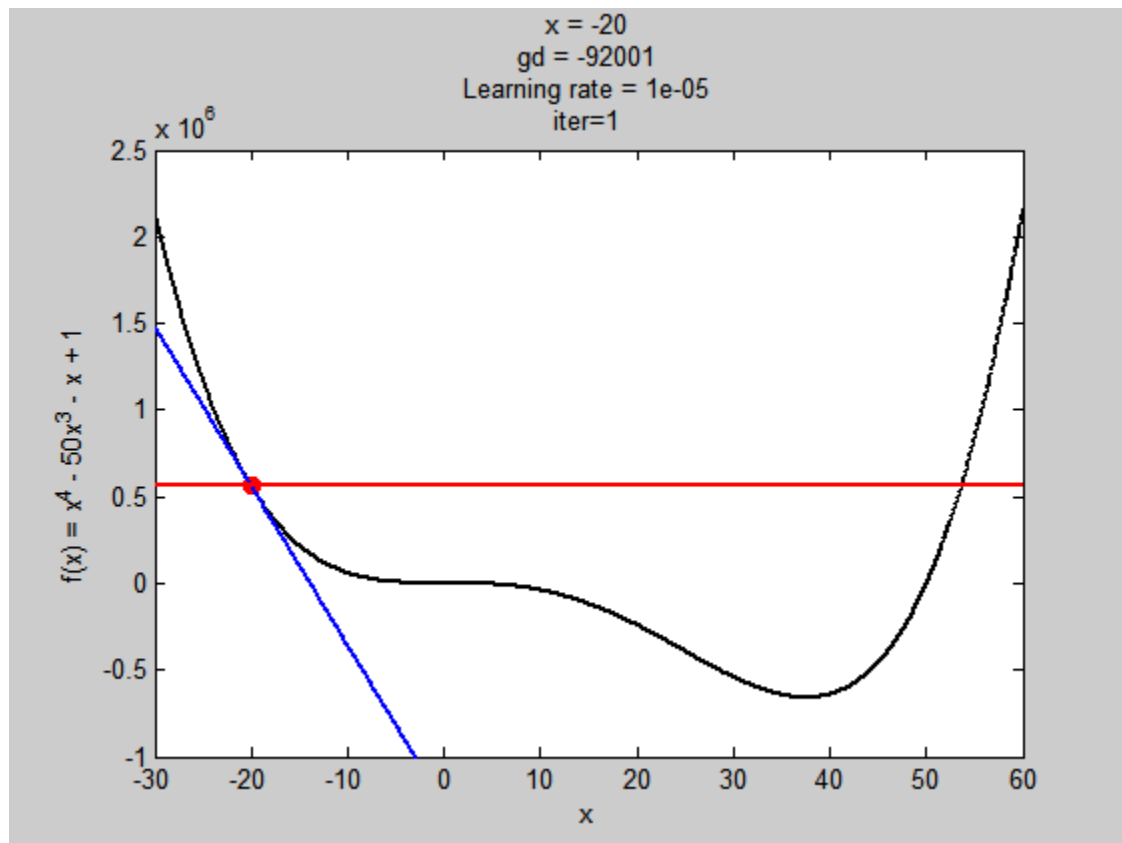


梯度下降法-範例2

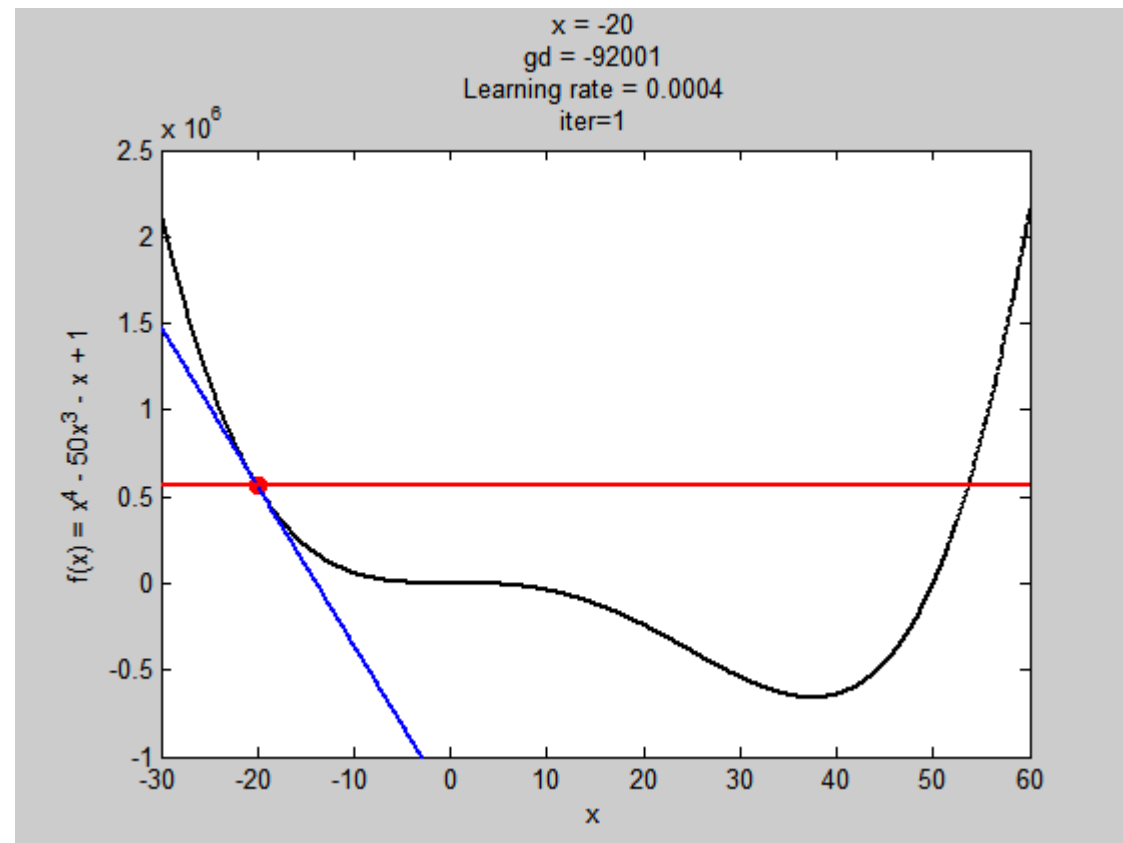
Learning rate 過
小容易掉到local
minima

$$f(x) = x^4 - 50x^3 - x + 1$$

Learning rate = 0.00001

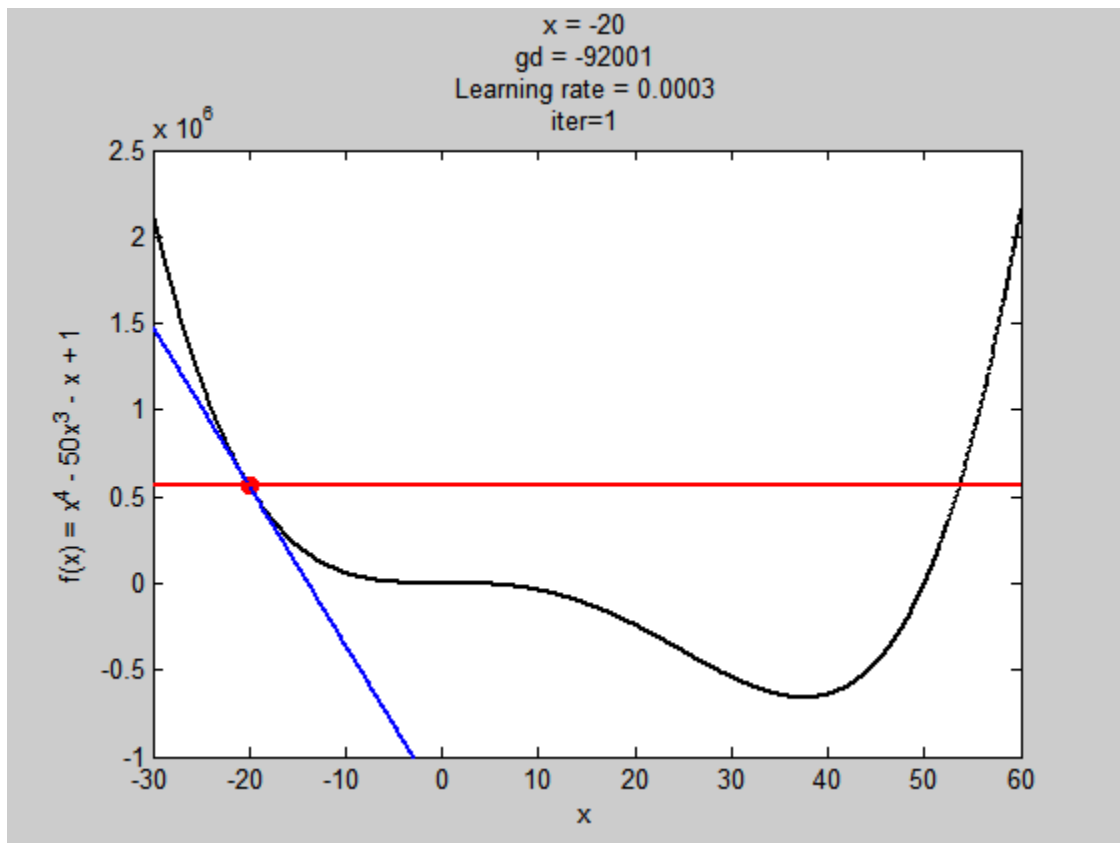


Learning rate = 0.0004



梯度下降法-範例2

Learning rate = 0.0003



Learning rate = 0.00001

過小掉到local minima

Learning rate = 0.0004

過大雖然跳出local minima，但走不進global minima

Learning rate = 0.0003

最合適



梯度下降法

梯度下降法: 學習率是非常重要的一个參數因子。

因此梯度下降研究就可以展開了



梯度下降法

大家比較常看到的function為Stochastic gradient descent (SGD)、Momentum、Adagrad、RMSProp、Adam。

SGD跟前面提到GD的差異。

一般在神經網路/深度學習，訓練數量可能達到幾百萬筆

一筆資料就update模型一次(很沒有效率): 一筆資料訓練假設要1秒，一萬筆都跑過模型就要1萬秒(2.7個小時)

Mini-batch update模型一次: 假設100筆一個mini-batch，訓練要兩秒，一萬筆只需要3.3分鐘。

SGD: 就是一次跑一個小批次(Mini-batch)後的平均梯度模型即更新一次，這個mini-batch為隨機抽取出，所以用Stochastic。



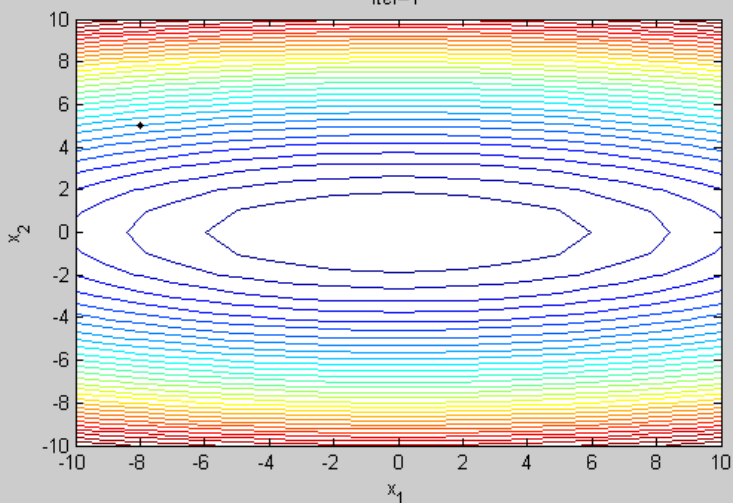
SGD

$$f(x_1, x_2) = 0.1x_1^2 + x_2^2$$

Initial point=[-8,5]

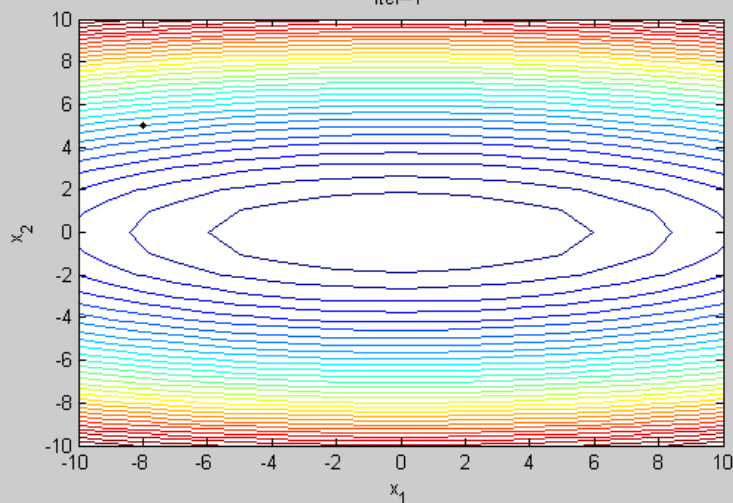
Learning rate = 0.9

x = -8.5
gradient norm = 0
loss = 31.4
Learning rate = 0.9
iter=1



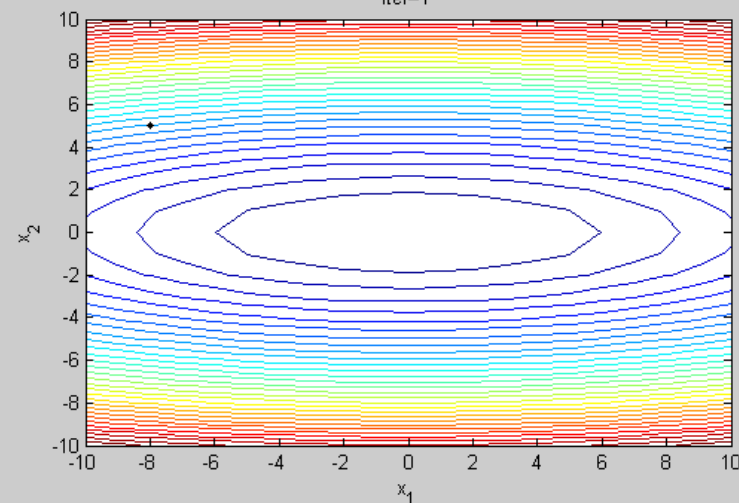
Learning rate = 0.5

x = -8.5
gradient norm = 0
loss = 31.4
Learning rate = 0.5
iter=1



Learning rate = 1

x = -8.5
gradient norm = 0
loss = 31.4
Learning rate = 1
iter=1



Momentum

Momentum是架構在SGD上的算法

$$\begin{aligned} \mathbf{v}^{(t)} &= \begin{cases} \gamma \mathbf{g}_t & t = 0 \\ m\mathbf{v}^{(t-1)} + \gamma \mathbf{g}_t & t \geq 1 \end{cases} \\ \mathbf{x}^{(t+1)} &= \mathbf{x}^{(t)} - \mathbf{v}^{(t)} \\ \mathbf{g}_t &= \nabla f(\mathbf{x}^{(t)}) \end{aligned}$$

\mathbf{g}_t :第t次迭代的gradient。

$\mathbf{v}^{(t)}$:第t次參數要更新的幅度(歷史的梯度和)。

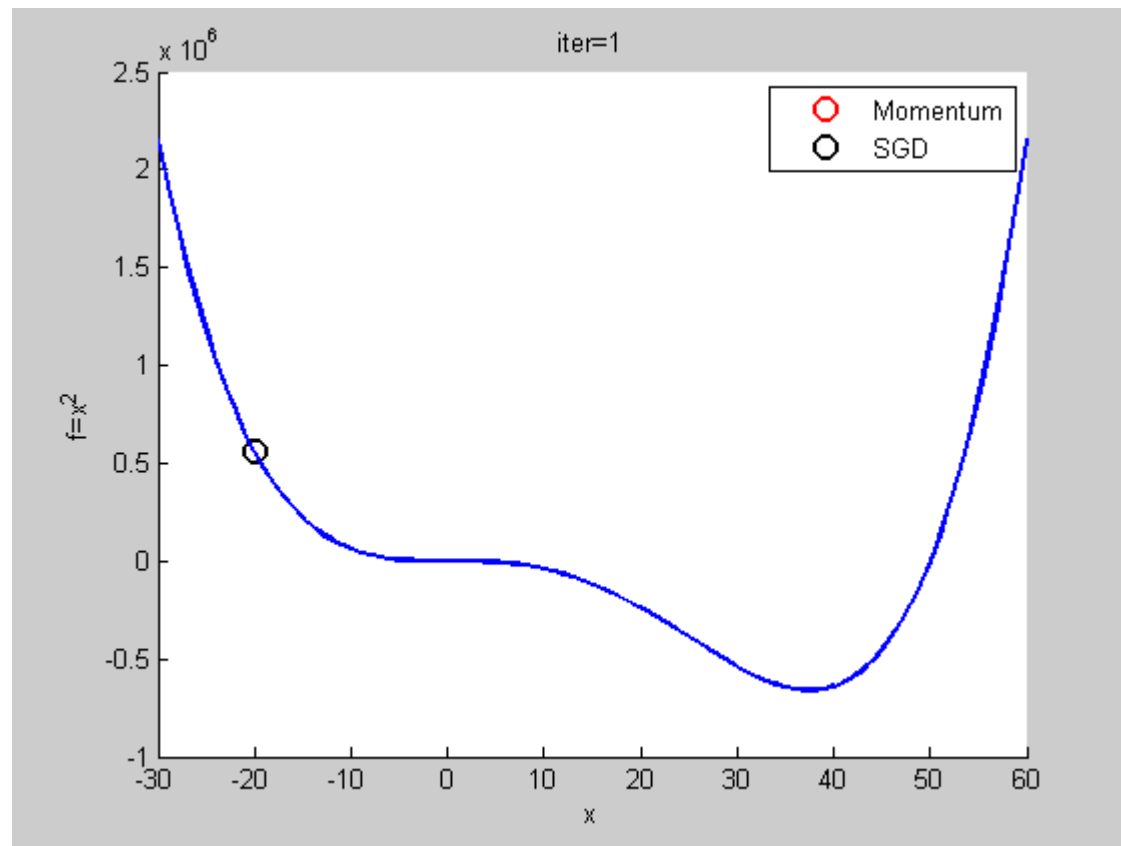
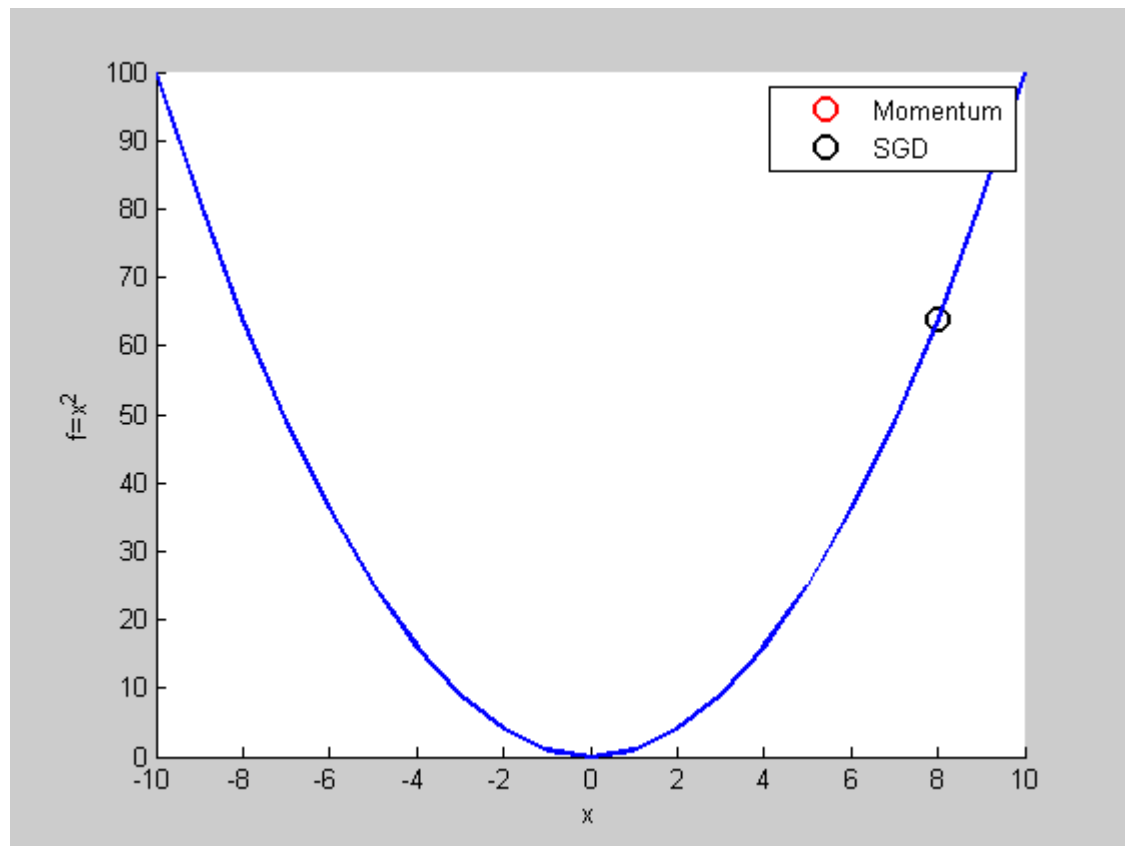
如果過去的梯度方向和當下的梯度方向一致，代表這個更新方向是對的，會增強這個方向的梯度。

如果過去的和當下方向不一致，則梯度會有消融作用(衰退)，只會微幅調整參數。



Momentum

Learning rate = 0.00001



基本上API用的SGD都有Momentum可以設定。



Adagrad

- SGD和momentum在更新參數時，都是用同一個學習率(γ)
- Adagrad算法則是在學習過程中對學習率不斷的調整，這種技巧叫做「學習率衰減(Learning rate decay)」。
- Ada這個字跟是Adaptive縮寫。

Gradient: $g_{t,i} = \nabla_{x_i} f(x_i^{(t)})$

SGD: $x_i^{(t+1)} = x_i^{(t)} - \gamma g_{t,i}$

Adagrad : $x_i^{(t+1)} = x_i^{(t)} - \frac{\gamma}{\sqrt{G_{t,ii} + \varepsilon}} g_{t,i}$



Adagrad

$$x_i^{(t+1)} = x_i^{(t)} - \frac{\gamma}{\sqrt{G_{t,ii}} + \varepsilon} g_{t,i} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^{d \times 1}$$

$$G_t = \begin{bmatrix} \sum_{t'=1}^t (g_{t',1} \times g_{t',1}) & 0 & 0 \\ 0 & \ddots & 0 \\ 0 & 0 & \sum_{t'=1}^t (g_{t',d} \times g_{t',d}) \end{bmatrix} \in \mathbb{R}^{d \times d}$$

$$G_{t,ii} = \sum_{t'=1}^t (g_{t',i} \times g_{t',i})$$



Adagrad

$$x_i^{(t+1)} = x_i^{(t)} - \frac{\gamma}{\sqrt{G_{t,ii}} + \varepsilon} g_{t,i} \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_d \end{bmatrix} \in \mathbb{R}^{d \times 1}$$
$$G_{t,ii} = \sum_{t'=1}^t (g_{t',i} \times g_{t',i})$$

- 隨著迭代次數越多，分母(Gradient平方和開根號)越大，學習率會越低，達到前期學習率高，後期學習率低的目的。
- 缺點: 學習中後期，分母有可能因為累積過大導致最後更新的參數趨近於0，所以無法有效學習。



RMSProp

- RMSprop是Geoff Hinton 提出未發表的方法，和Adagrad一樣是自適應的方法，但**Adagrad**的分母是從第1次梯度到第t次梯度的和，所以和可能過大，**RMSprop**則是算對應的平均值，因此可以緩解Adagrad學習率下降過快的問題。

Adagrad

$$x_i^{(t+1)} = x_i^{(t)} - \frac{\gamma}{\sqrt{G_{t,ii}} + \varepsilon} g_{t,i}$$

$$G_{t,ii} = \sum_{t'=1}^t (g_{t',i} \times g_{t',i})$$

RMSprop

$$x_i^{(t+1)} = x_i^{(t)} - \frac{\gamma}{\sqrt{E[g_i^2]_t} + \varepsilon} g_{t,i}$$

$$E[g_i^2]_t = \rho E[g_i^2]_{t-1} + (1 - \rho) g_{t,i}^2$$



Adam

- **Momentum**: 考慮過去梯度的方向和當前梯度的方向做合成 (沒直接修改learning rate)
- **Adagrad、RMSprop**: 考慮過去梯度的大小用來修改learning rate (Learning rate decay)
- **Adam(Adaptive Moment Estimation)**則是兩者合併加強版本(Momentum+RMSprop+各自做偏差的修正)



Adam

Momentum

$$\mathbf{v}^{(t)} = \begin{cases} \gamma \mathbf{g}_t & t = 0 \\ \mathbf{m} \mathbf{v}^{(t-1)} + \gamma \mathbf{g}_t & t \geq 1 \end{cases}$$

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \mathbf{v}^{(t)}$$

$$\mathbf{g}_t = \nabla f(\mathbf{x}^{(t)})$$

$$\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1) \mathbf{g}_t$$

$$\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2) \mathbf{g}_t^2$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1 - \beta_1^t}$$

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_2^t}$$

RMSProp

$$x_i^{(t+1)} = x_i^{(t)} - \frac{\gamma}{\sqrt{E[g_i^2]_t + \varepsilon}} g_{t,i}$$

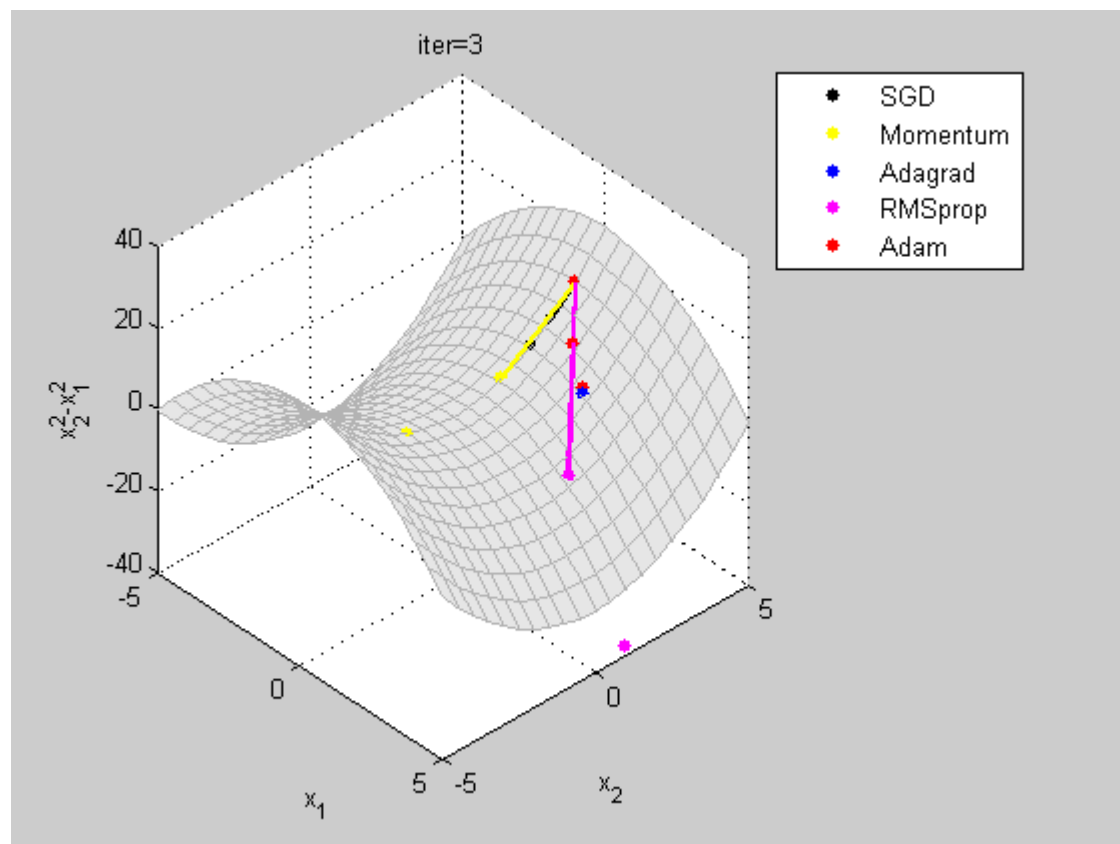
$$E[g_i^2]_t = \rho E[g_i^2]_{t-1} + (1 - \rho) g_{t,i}^2$$

$$\mathbf{x}^{(t+1)} = \mathbf{x}^{(t)} - \frac{\gamma}{\sqrt{\hat{\mathbf{v}}_t + \varepsilon}} \hat{\mathbf{m}}_t$$



Example

$$f(x_1, x_2) = x_2^2 - x_1^2$$



導傳遞

- 神經網路如何利用導傳遞找解
- 神經網路太深層的問題:

梯度消失問題(Vanishing gradient)/梯度爆炸問題 (exploding gradient problem) 。

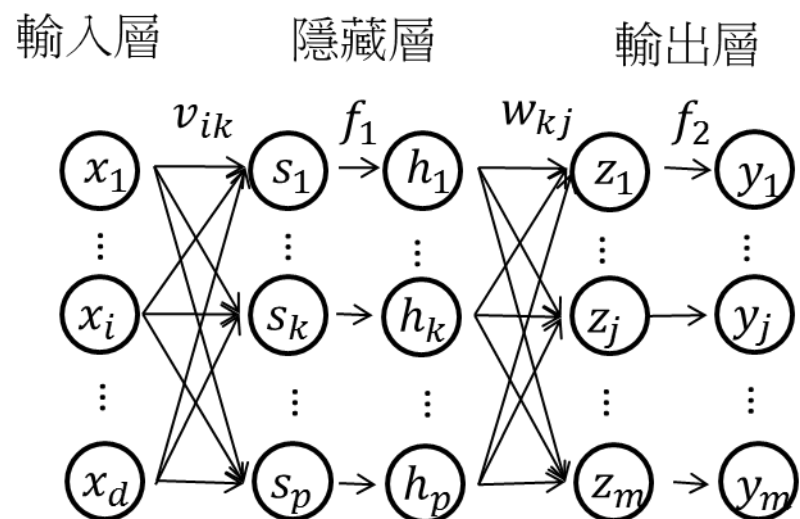
Activation Function為什麼要採用ReLU，而不是用Sigmoid。

Residual block克服神經網路不能太深層的問題。

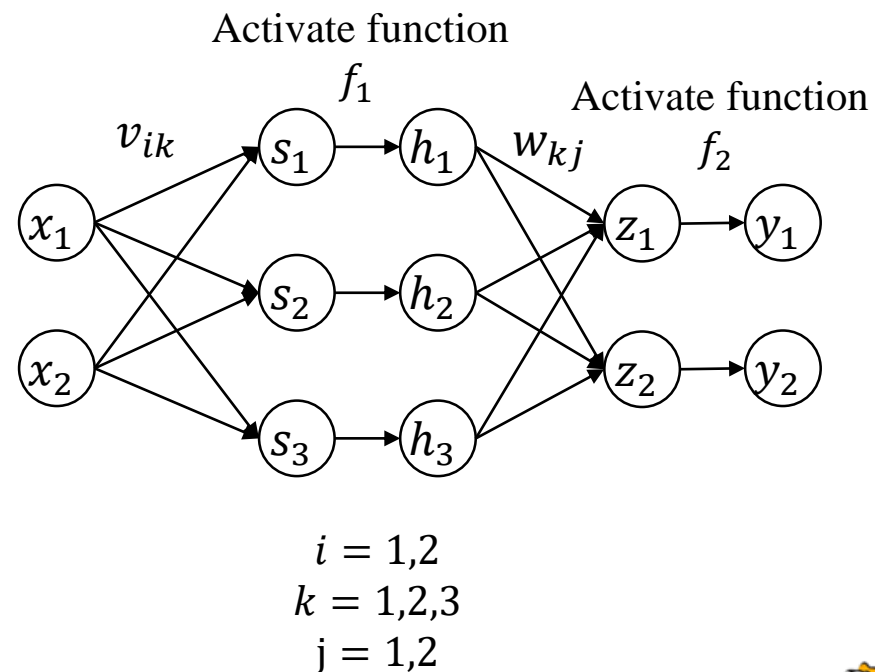
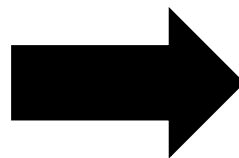


神經網路如何利用導傳遞找解

- 下圖為一個三層的MLP

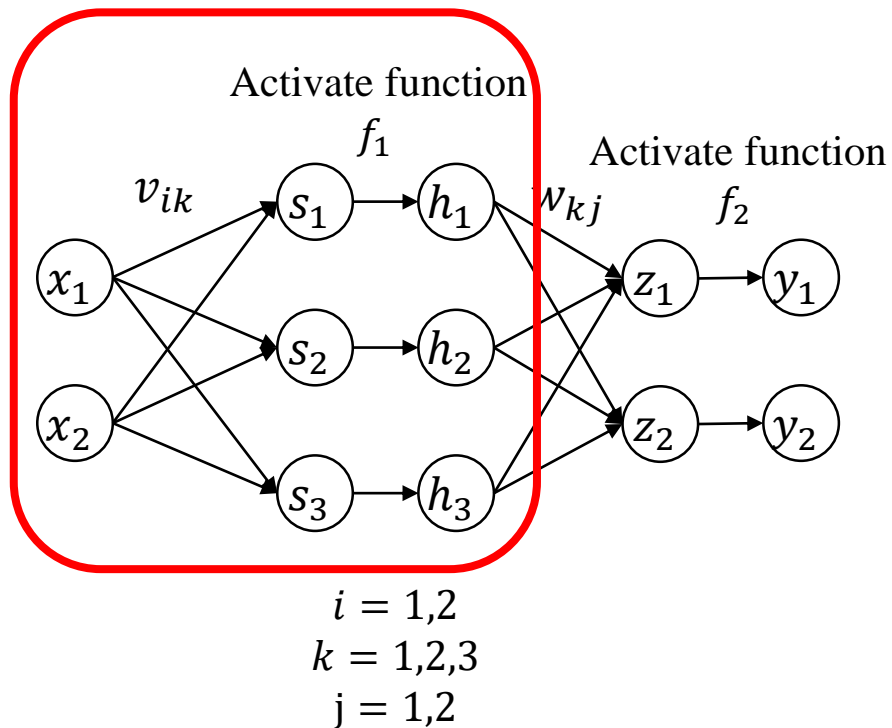


簡化



Forward propagation

輸入層→隱藏層



$$\begin{aligned} s_1 &= v_{11}x_1 + v_{21}x_2 = \mathbf{v}_1^T \mathbf{x} \\ s_2 &= v_{12}x_1 + v_{22}x_2 = \mathbf{v}_2^T \mathbf{x} \\ s_3 &= v_{13}x_1 + v_{23}x_2 = \mathbf{v}_3^T \mathbf{x} \end{aligned}$$

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \mathbf{v}_k = \begin{bmatrix} v_{1k} \\ v_{2k} \end{bmatrix}, k = 1, 2, 3$$



$$s_k = \sum_{i=1}^2 v_{ik}x_i$$

$$\mathbf{s} = \mathbf{v}^T \mathbf{x}$$

$$\mathbf{s} = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \end{bmatrix}$$

$$\mathbf{v} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \mathbf{v}_3 \end{bmatrix}$$

$$= \begin{bmatrix} v_{11} & v_{12} & v_{13} \\ v_{21} & v_{22} & v_{23} \end{bmatrix}$$

Activate function

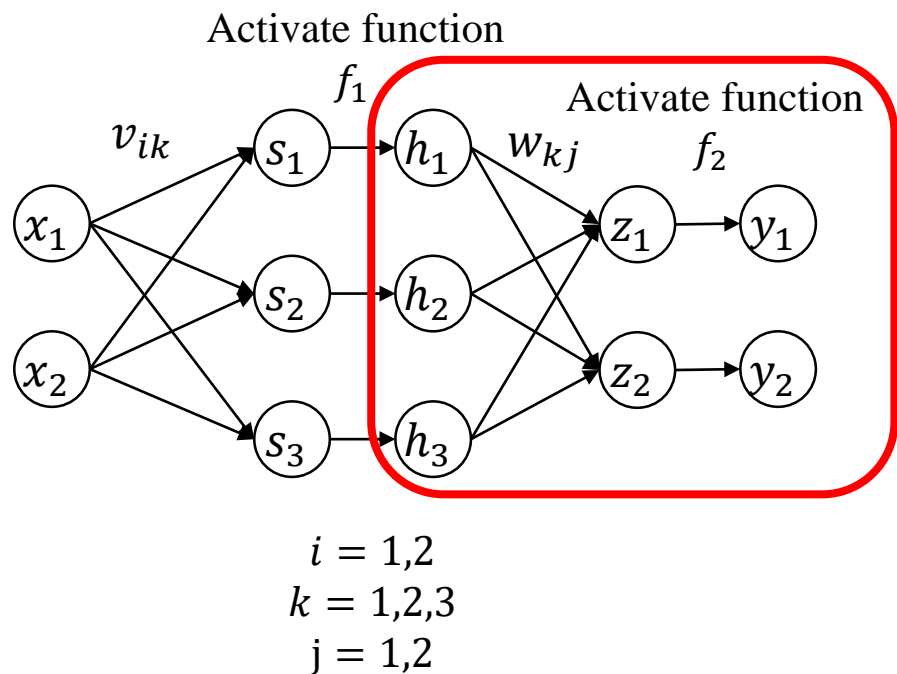
$$\mathbf{h} = f_1(\mathbf{s})$$

$$\mathbf{h} = \begin{bmatrix} h_1 \\ h_2 \\ h_3 \end{bmatrix}$$



Forward propagation

隱藏層→輸出層



$$z_1 = w_{11}h_1 + w_{21}h_2 + w_{31}h_3 = \mathbf{w}_1^T \mathbf{h}$$

$$z_2 = w_{12}h_1 + w_{22}h_2 + w_{32}h_3 = \mathbf{w}_2^T \mathbf{h}$$

$$\mathbf{w}_j = \begin{bmatrix} w_{1j} \\ w_{2j} \\ w_{3j} \end{bmatrix}, j = 1, 2$$

Activate function

$$\mathbf{y} = f_2(\mathbf{z})$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

$$\mathbf{z} = \mathbf{w}^T \mathbf{h}$$

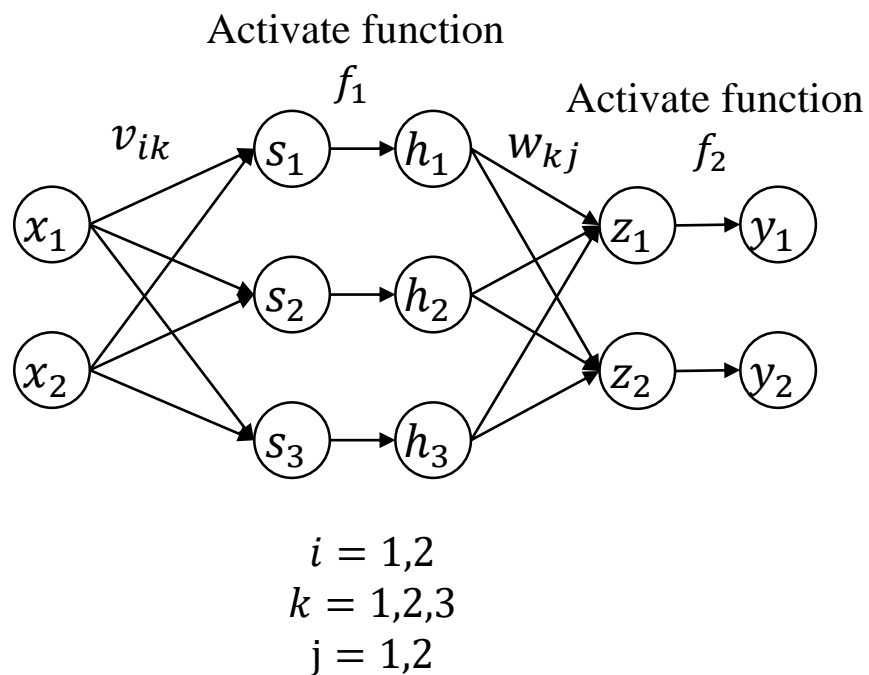
$$\mathbf{z} = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$$

$$\mathbf{w} = [\mathbf{w}_1 \quad \mathbf{w}_2]$$

$$= \begin{bmatrix} w_{11} & w_{12} \\ w_{21} & w_{22} \\ w_{31} & w_{32} \end{bmatrix}$$



Forward propagation



輸入層→隱藏層

$$\mathbf{s} = \mathbf{v}^T \mathbf{x}$$

$$\mathbf{h} = f_1(\mathbf{s})$$

隱藏層→輸出層

$$\mathbf{z} = \mathbf{w}^T \mathbf{h}$$

$$\mathbf{y} = f_2(\mathbf{z})$$

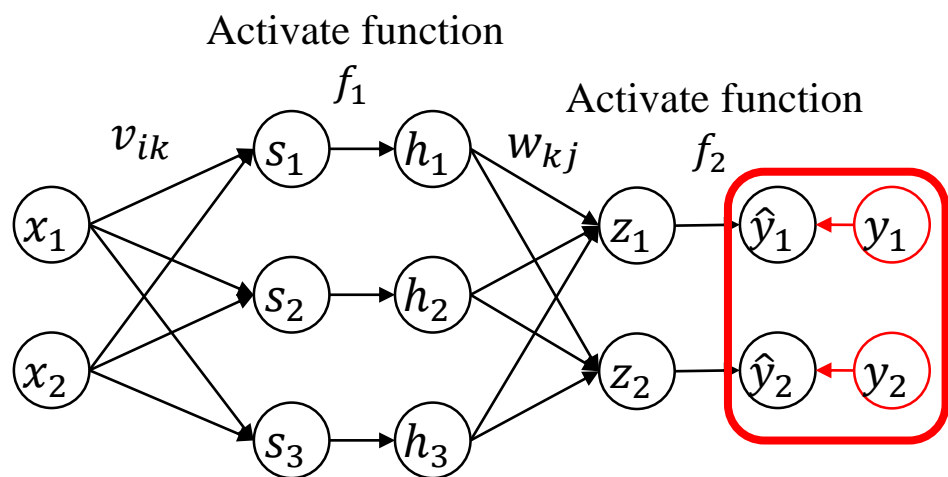
||

$$\mathbf{y} = f_2(\mathbf{w}^T f_1(\mathbf{v}^T \mathbf{x}))$$

參數為 \mathbf{v} 和 \mathbf{w} 。



Back-propagation



輸入層→隱藏層

$$\mathbf{s} = \mathbf{v}^T \mathbf{x}$$

$$\mathbf{h} = f_1(\mathbf{s})$$

隱藏層→輸出層

$$\mathbf{z} = \mathbf{w}^T \mathbf{h}$$

$$\mathbf{y} = f_2(\mathbf{z})$$

$$\hat{\mathbf{y}} = f_2(\mathbf{w}^T f_1(\mathbf{v}^T \mathbf{x}))$$

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix}$$

我們用SSE來當作loss function

$$\text{loss}(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} (\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}})$$

利用gradient descent找最佳參數解(參數只有 \mathbf{w} 和 \mathbf{v})

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha \nabla \mathbf{w}^{(t)}$$

$$\mathbf{v}^{(t+1)} = \mathbf{v}^{(t)} - \alpha \nabla \mathbf{v}^{(t)}$$

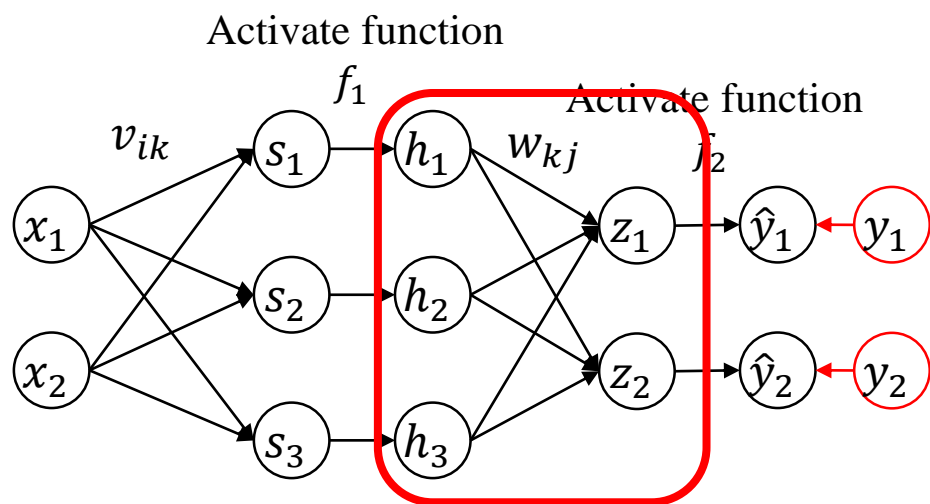
$$\nabla \mathbf{w} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{w}} \quad \text{輸出層} \rightarrow \text{隱藏層參數為} \mathbf{w}, \text{對loss進行} \mathbf{w} \text{的偏微分}$$

$$\nabla \mathbf{v} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{v}} \quad \text{隱藏層} \rightarrow \text{輸入層參數為} \mathbf{v}, \text{對loss進行} \mathbf{v} \text{的偏微分}$$



Back-propagation

輸出層→隱藏層



隱藏層→輸出層

$$\mathbf{z} = \mathbf{w}^T \mathbf{h}$$

$$\mathbf{y} = f_2(\mathbf{z})$$

參數為 \mathbf{w} ，因此我們對loss進行 \mathbf{w} 的偏微分

$$\begin{aligned} \text{loss}(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{2} (\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}}) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{z}))^T (\mathbf{y} - f_2(\mathbf{z})) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{w}^T \mathbf{h}))^T (\mathbf{y} - f_2(\mathbf{w}^T \mathbf{h})) \end{aligned}$$

Chain rule

$$\Delta \mathbf{w} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{w}} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{w}}$$

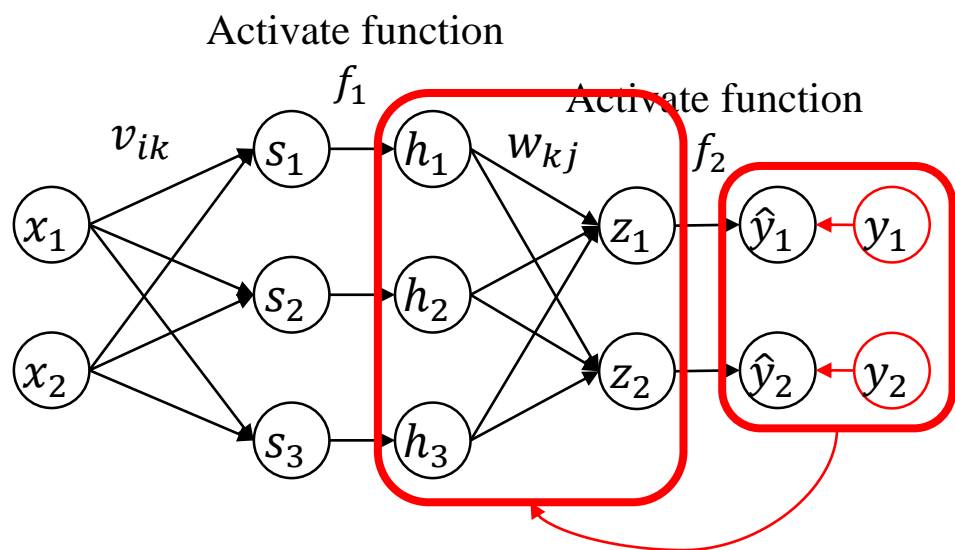
$$\begin{aligned} \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}} &= \frac{\frac{1}{2} (\mathbf{y} - f_2(\mathbf{z}))^T (\mathbf{y} - f_2(\mathbf{z}))}{\partial \mathbf{z}} \\ &= (\mathbf{y} - f_2(\mathbf{z})) \frac{\partial f_2(\mathbf{z})}{\partial \mathbf{z}} = (\mathbf{y} - \hat{\mathbf{y}}) f_2'(\mathbf{z}) \end{aligned}$$

$$\frac{\partial \mathbf{z}}{\partial \mathbf{w}} = \frac{\partial \mathbf{w}^T \mathbf{h}}{\partial \mathbf{w}} = \mathbf{h}$$



Back-propagation

輸出層→隱藏層



隱藏層→輸出層

$$\mathbf{z} = \mathbf{w}^T \mathbf{h}$$

$$\mathbf{y} = f_2(\mathbf{z})$$

參數為 \mathbf{w} ，因此我們對loss進行 \mathbf{w} 的偏微分

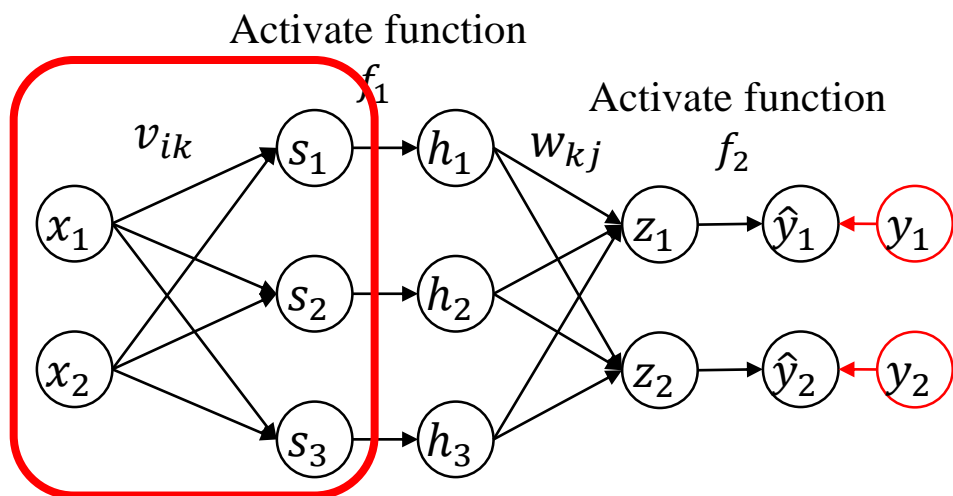
$$\begin{aligned} \text{loss}(\mathbf{y}, \hat{\mathbf{y}}) &= \frac{1}{2} (\mathbf{y} - \hat{\mathbf{y}})^T (\mathbf{y} - \hat{\mathbf{y}}) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{z}))^T (\mathbf{y} - f_2(\mathbf{z})) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{w}^T \mathbf{h}))^T (\mathbf{y} - f_2(\mathbf{w}^T \mathbf{h})) \end{aligned}$$

$$\begin{aligned} \Delta \mathbf{w} &= \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{w}} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{w}} \\ &= (\mathbf{y} - \hat{\mathbf{y}}) f_2'(\mathbf{z}) \mathbf{h} \end{aligned}$$



Back-propagation

隱藏層→輸入層



隱藏層→輸入層

$$\mathbf{s} = \mathbf{v}^T \mathbf{x}$$

$$\mathbf{h} = f_1(\mathbf{s})$$

參數為 \mathbf{v} ，因此我們對loss進行 \mathbf{v} 的偏微分

$$\begin{aligned} \text{loss}(\mathbf{y}, \hat{\mathbf{y}}) &= (\mathbf{y} - f_2(\mathbf{z}))^T (\mathbf{y} - f_2(\mathbf{z})) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{s})))^T (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{s}))) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{v}^T \mathbf{x})))^T (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{v}^T \mathbf{x}))) \end{aligned}$$

Chain rule

$$\Delta \mathbf{v} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{v}} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{v}} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{v}}$$

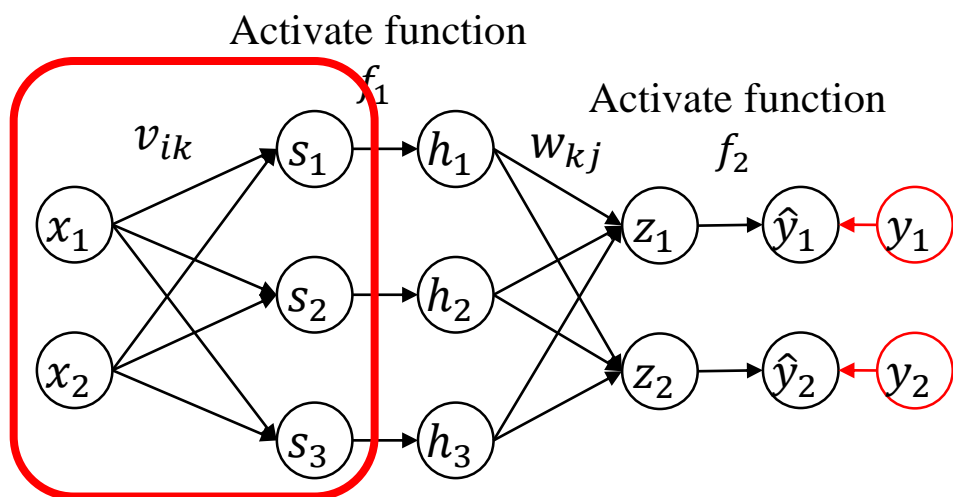
前面解過了

$$\frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}} = (\mathbf{y} - \hat{\mathbf{y}}) f_2'(\mathbf{z})$$



Back-propagation

隱藏層→輸入層



隱藏層→輸入層

$$\mathbf{s} = \mathbf{v}^T \mathbf{x}$$

$$\mathbf{h} = f_1(\mathbf{s})$$

參數為 \mathbf{v} ，因此我們對loss進行 \mathbf{v} 的偏微分

$$\begin{aligned} \text{loss}(\mathbf{y}, \hat{\mathbf{y}}) &= (\mathbf{y} - f_2(\mathbf{z}))^T (\mathbf{y} - f_2(\mathbf{z})) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{s})))^T (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{s}))) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{v}^T \mathbf{x})))^T (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{v}^T \mathbf{x}))) \end{aligned}$$

Chain rule

$$\Delta \mathbf{v} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{v}} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{v}} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{v}}$$

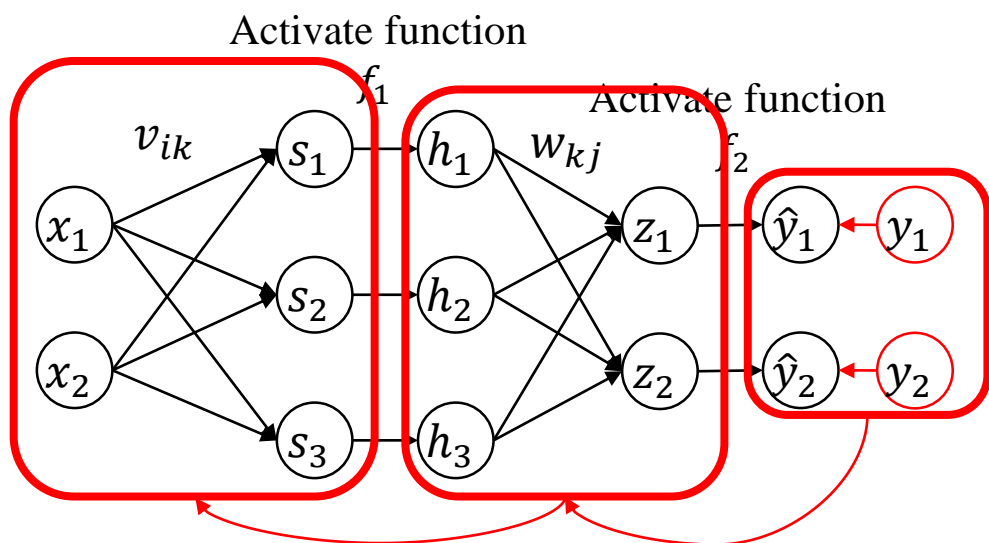
$$\frac{\partial \mathbf{z}}{\partial \mathbf{s}} = \frac{\partial \mathbf{w}^T f_1(\mathbf{s})}{\partial \mathbf{s}} = \mathbf{w}^T f_1'(\mathbf{s})$$

$$\frac{\partial \mathbf{s}}{\partial \mathbf{v}} = \frac{\partial \mathbf{v}^T \mathbf{x}}{\partial \mathbf{v}} = \mathbf{x}$$



Back-propagation

隱藏層→輸入層



隱藏層→輸入層

$$\mathbf{s} = \mathbf{v}^T \mathbf{x}$$

$$\mathbf{h} = f_1(\mathbf{s})$$

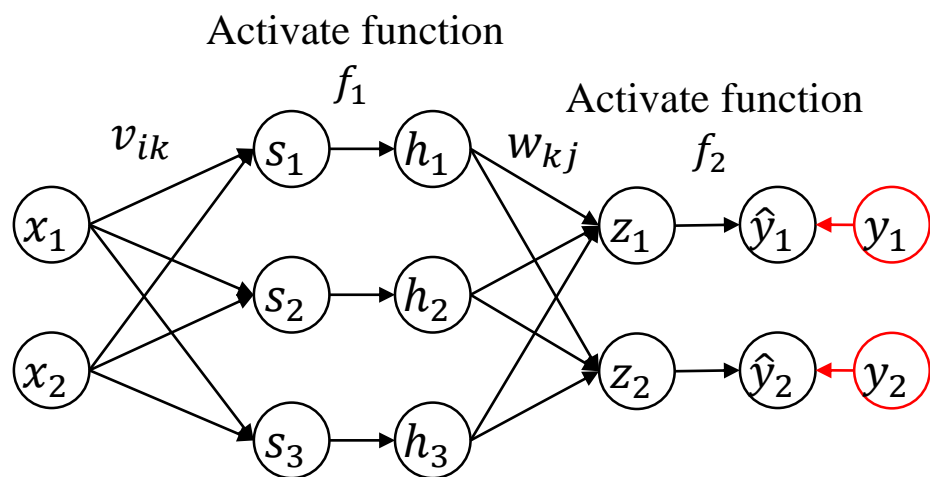
參數為 \mathbf{v} ，因此我們對 loss 進行 \mathbf{v} 的偏微分

$$\begin{aligned} \text{loss}(\mathbf{y}, \hat{\mathbf{y}}) &= (\mathbf{y} - f_2(\mathbf{z}))^T (\mathbf{y} - f_2(\mathbf{z})) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{s})))^T (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{s}))) \\ &= \frac{1}{2} (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{v}^T \mathbf{x})))^T (\mathbf{y} - f_2(\mathbf{w}^T f_1(\mathbf{v}^T \mathbf{x}))) \end{aligned}$$

$$\begin{aligned} \Delta \mathbf{v} &= \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{v}} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{v}} = \frac{\partial \text{loss}(\mathbf{y}, \hat{\mathbf{y}})}{\partial \mathbf{z}} \frac{\partial \mathbf{z}}{\partial \mathbf{s}} \frac{\partial \mathbf{s}}{\partial \mathbf{v}} \\ &= [(\mathbf{y} - \hat{\mathbf{y}}) f_2'(\mathbf{z})] [\mathbf{w}^T f_1'(\mathbf{s})] \mathbf{x} \end{aligned}$$



Back-propagation



輸入層→隱藏層

$$\begin{aligned} s &= v^T x \\ h &= f_1(s) \end{aligned}$$

隱藏層→輸出層

$$\begin{aligned} z &= w^T h \\ y &= f_2(z) \end{aligned}$$

$$w^{(t+1)} = w^{(t)} - \alpha \Delta w^{(t)}$$

$$v^{(t+1)} = v^{(t)} - \alpha \Delta v^{(t)}$$

輸出層→隱藏層參數為 w

$$\Delta w = (y - \hat{y}) f_2'(z) h$$

隱藏層→輸入層參數為 v

$$\Delta v = [(y - \hat{y}) f_2'(z)] [w^T f_1'(s)] x$$

前面層的gradient
會是由後面所有層的gradient和現在這層的疊乘。



神經網路太深層的問題

神經網路太深層的問題



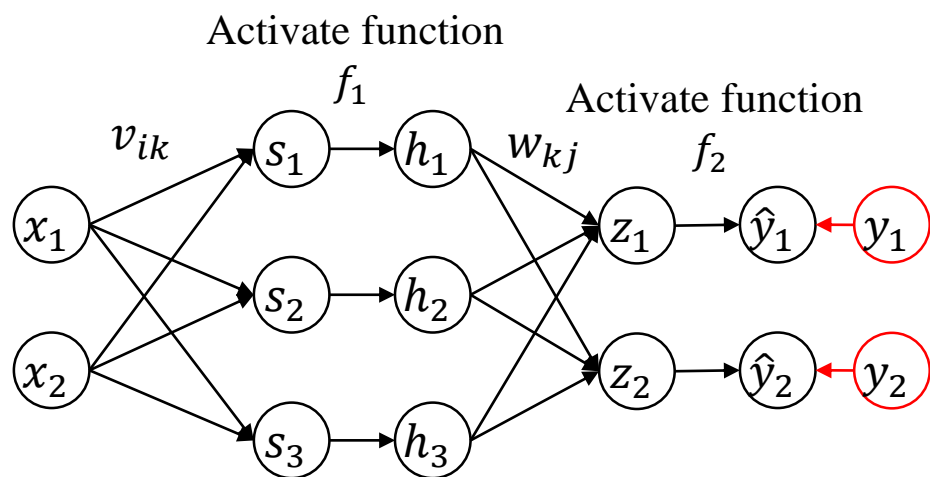
Sigmoid在神經網路太深層的問題

輸出層→隱藏層參數為 w

$$\Delta w = (y - \hat{y}) f_2'(z) h$$

隱藏層→輸入層參數為 v

$$\Delta v = [(y - \hat{y}) f_2'(z)] [w^T f_1'(s)] x$$



輸入層→隱藏層

$$s = v^T x$$

$$h = f_1(s)$$

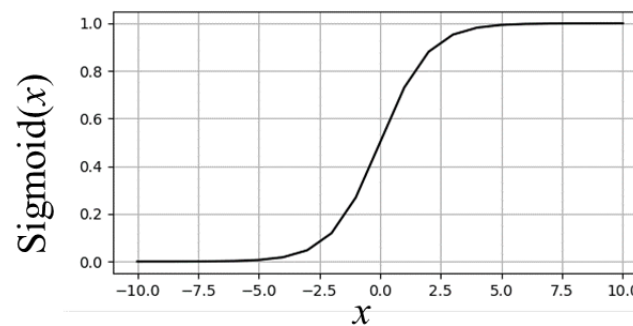
隱藏層→輸出層

$$z = w^T h$$

$$y = f_2(z)$$

$$\text{Sigmoid} : f(x) = \frac{1}{1+e^{-x}}$$

$$f' = f(x)(1 - f(x))$$

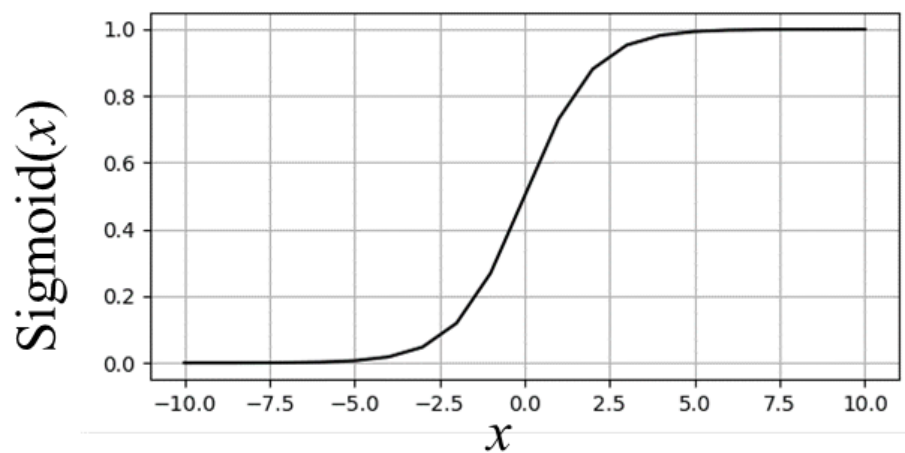


Sigmoid在神經網路太深層的問題

Sigmoid : $f(x) = \frac{1}{1+e^{-x}}$

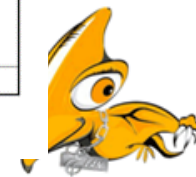
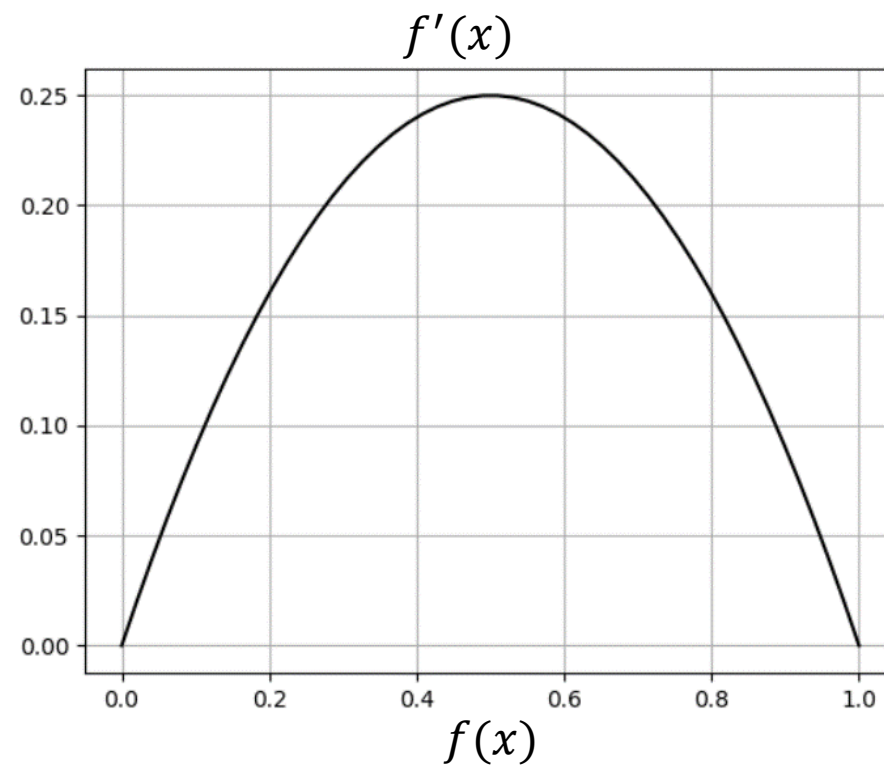
$$f'(x) = f(x)(1 - f(x))$$

$f(x)$ 輸出介於0~1



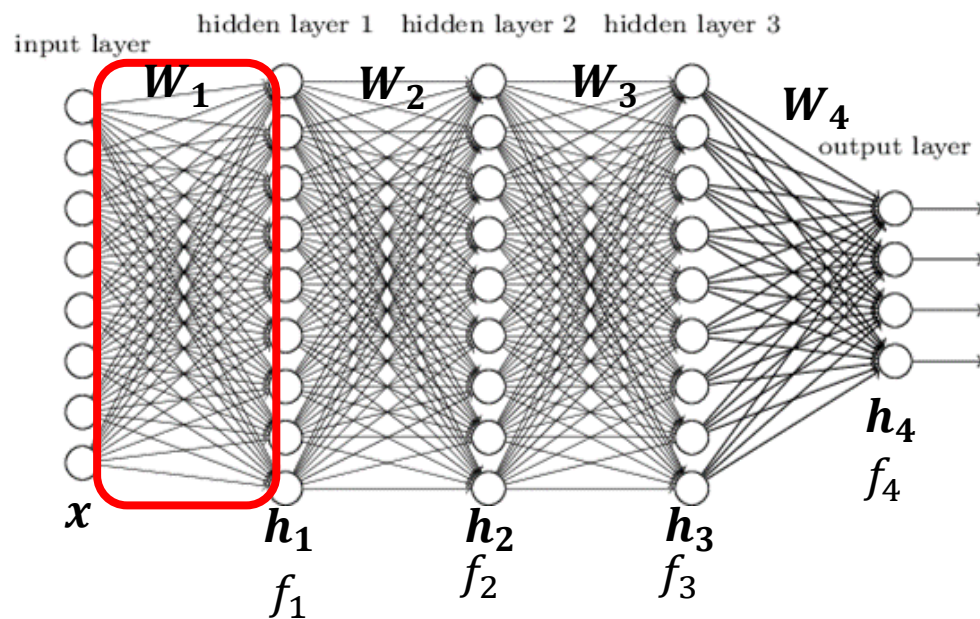
$f'(x)$ 會更小，最大值為0.25

$$f(x) = 0.5, f'(x) = 0.5 \times 0.5 = 0.25$$



Sigmoid在神經網路太深層的問題

MLP



$$\Delta W_1 = [(y - \hat{y})f'_4(h_4)][W_3^T f'_3(h_3)][W_2^T f'_2(h_2)][W_1^T f'_1(h_1)]x$$

所以當層數到100層時候，對於第一層的gradient會有100個activate function的導數相乘。

剛剛sigmoid已經說了，其導數最大為0.25。

所以 $0.25^{100} \approx 0$

這就是梯度消失問題(Vanishing gradient)

如果導數值單調大於1時，就會發生**梯度爆炸問題 (exploding gradient problem)**。



如何舒緩Gradient造成的問題

- 1. 重新設計網路架構: 更少的層。
- 2. Rectified Linear Activation (ReLU)
- 3. Gradient Clipping (Keras預設 clipnorm = 1.0和clipvalue = 0.5。)
- 4. Weight Regularization (L1 or L2 Regularizers)

實際解決神經網路太深層的方法: Residual block。

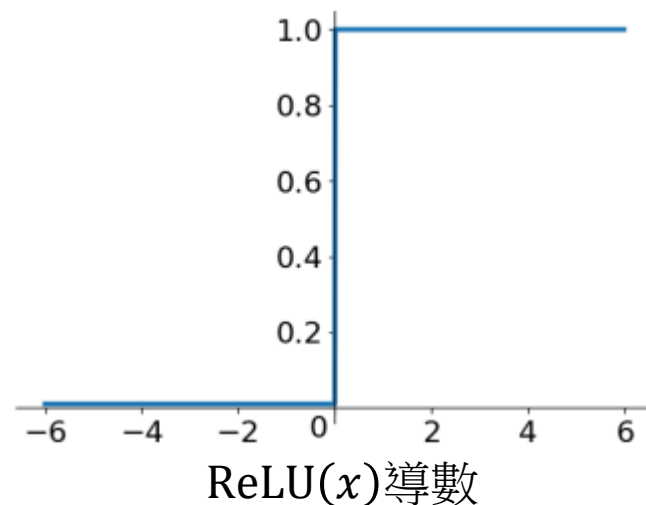
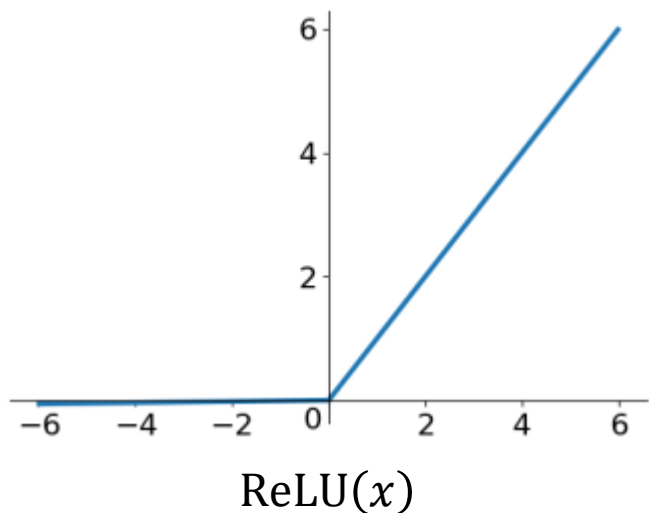
本次介紹不考慮RNN系列的設計。



ReLU在神經網路如何舒緩Gradient的問題

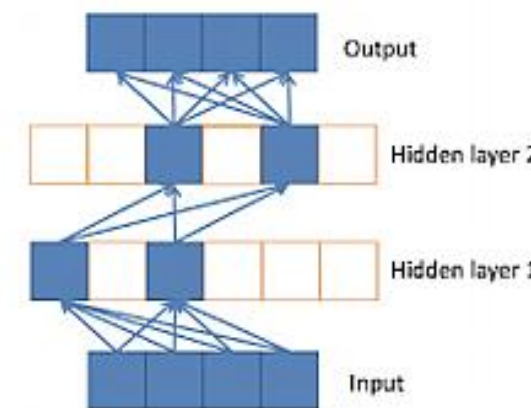
- $\text{ReLU}(x) = \max(0, x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{O.W.} \end{cases}$
- $\text{ReLU}(x)$ 的導數 = $\begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{O.W.} \end{cases}$

ReLU函數並不是全區間皆可微分，但是不可微分的部分可以使用Sub-gradient進行取代



ReLU在神經網路如何舒緩Gradient的問題

- ReLU激勵函數會使負數部分的神經元輸出為0，可以讓網路變得更加多樣性，如同Dropout的概念，可以緩解過擬合(Over fitting)之問題。
- **衍生Dead ReLU的問題**，當某個神經元輸出為0後，就難以再度輸出值，當遇到以下兩種情形時容易導致dead ReLU發生。
 - 初始化權重設定為不能被激活的數值。
 - 學習率設置過大，在剛開始進行誤差反向傳遞時，容易修正權重值過大，導致權重梯度為0，神經元即再也無法被激活。



ResNet

此Residual block有兩層，第一層權重為 w_1 ，第二層權重為 w_2
第一層輸出:

$$\text{relu}(w_1 x)$$

第二層輸出:

$$F(x) = w_2 \text{relu}(w_1 x)$$

最後element-add後結果:

$$H(x) = F(x) + x$$

假設神經網路太深對應用沒有幫助

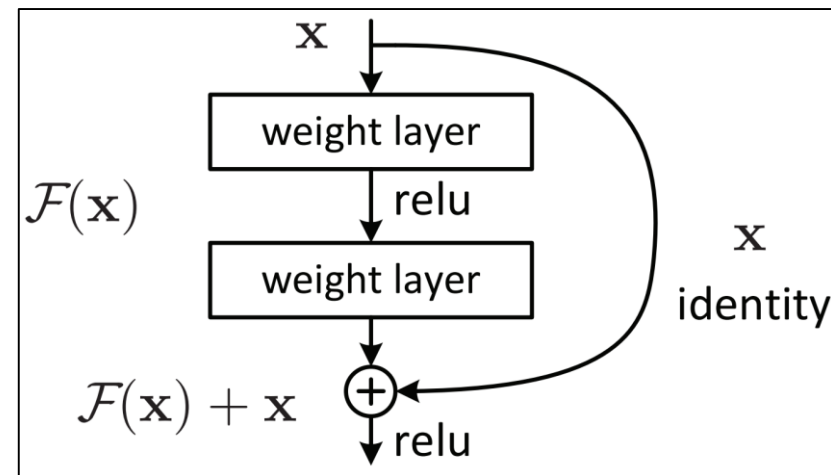
1. 沒有shortcut connection (residual) ，兩層的輸出結果叫做 $H(x)$

最理想狀況就是 $H(x) = \mathbf{F(x)} = \mathbf{x}$ (identity mapping)

2. 有shortcut connection ， $H(x) = F(x) + x \rightarrow \mathbf{F(x) = H(x) - x}$

最理想狀況就是 $\mathbf{F(x) = 0}$

2的優化找weight解會比1容易。



ResNet

假設我們的 *residual block* 只有一層權重為 w

1. 沒有 shortcut connection(residual) , 兩層的輸出結果叫做 $H(x)$

$$F(x) = wx = x$$

2. 有 **shortcut connection** , $H(x) = F(x) + x \rightarrow F(x) = H(x) - x$

$$F(x) = wx = 0$$

Gradient:

$$1. \frac{\partial E}{\partial w} = \frac{(\hat{y} - y)^2}{\partial w} = \frac{(wx - y)^2}{\partial w} = 2(wx - y)x$$

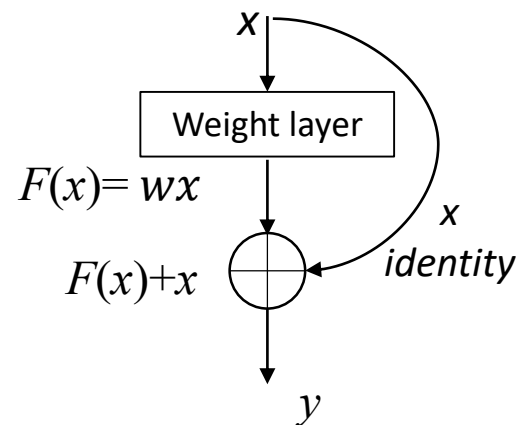
$$2. \frac{\partial E}{\partial w} = \frac{(\hat{y} - y)^2}{\partial w} = \frac{(wx + x - y)^2}{\partial w} = 2(wx - y)x + 2x^2$$

接近0

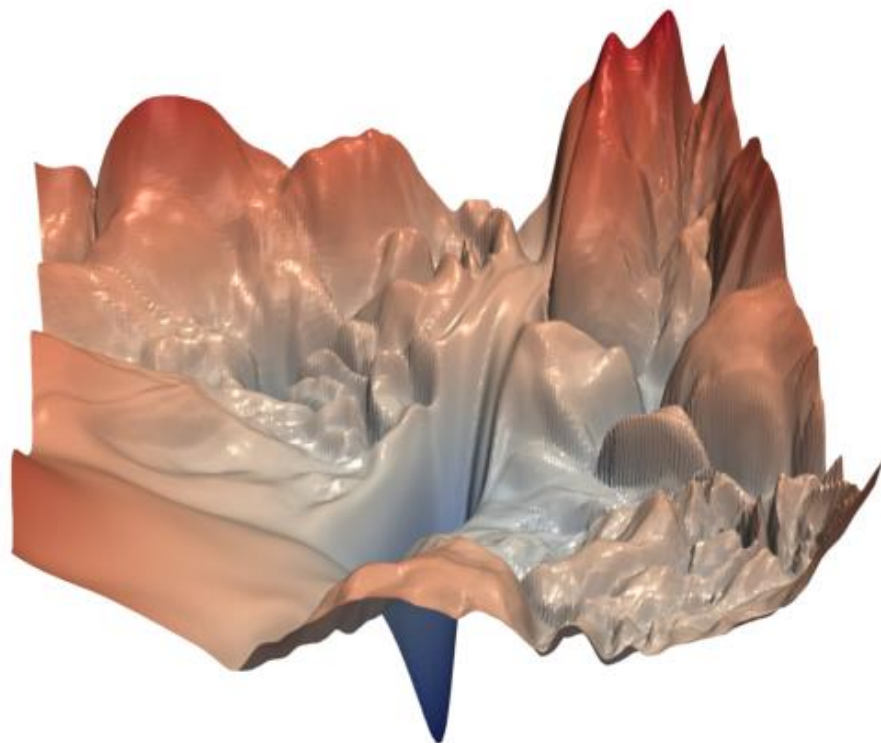
假設越後面層的Gradient很小，甚至Gradient vanish)。

$$\Delta W_1 = [(y - \hat{y})f_4'(h_4)][W_3^T f_3'(h_3)][W_2^T f_2'(h_2)][W_1^T f_1'(h_1)]x$$

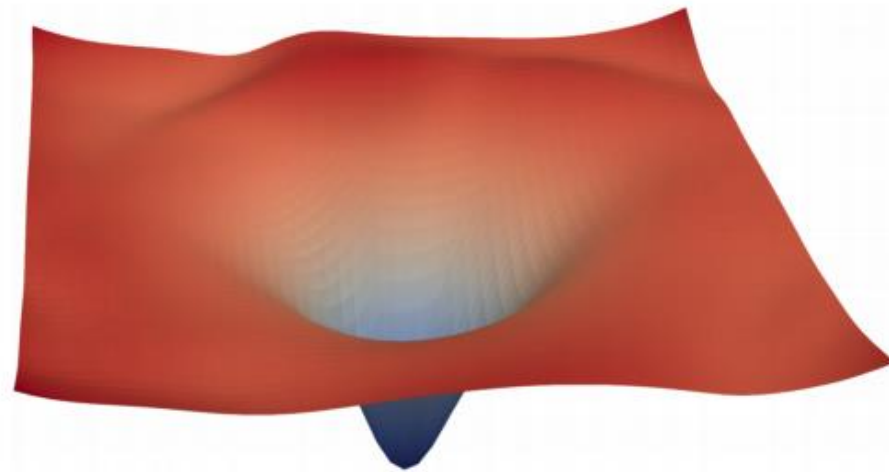
residual block



題外話: 有Residual block的loss space



(a) without skip connections



(b) with skip connections

Figure 1: The loss surfaces of ResNet-56 with/without skip connections. The proposed filter normalization scheme is used to enable comparisons of sharpness/flatness between the two figures.



導傳遞

- 神經網路如何利用導傳遞找解
- 神經網路太深層的問題:

梯度消失問題(Vanishing gradient)/梯度爆炸問題 (exploding gradient problem) 。

Activation Function為什麼要採用ReLU，而不是用Sigmoid。

Residual block克服神經網路不能太深層的問題。



Weight initialization

- 神經網路結構設定完成後，最重要的是
 1. 如何更新權重 (梯度下降法)
 2. 權重如何初始設定
- Weight initialization和Batch Normalization

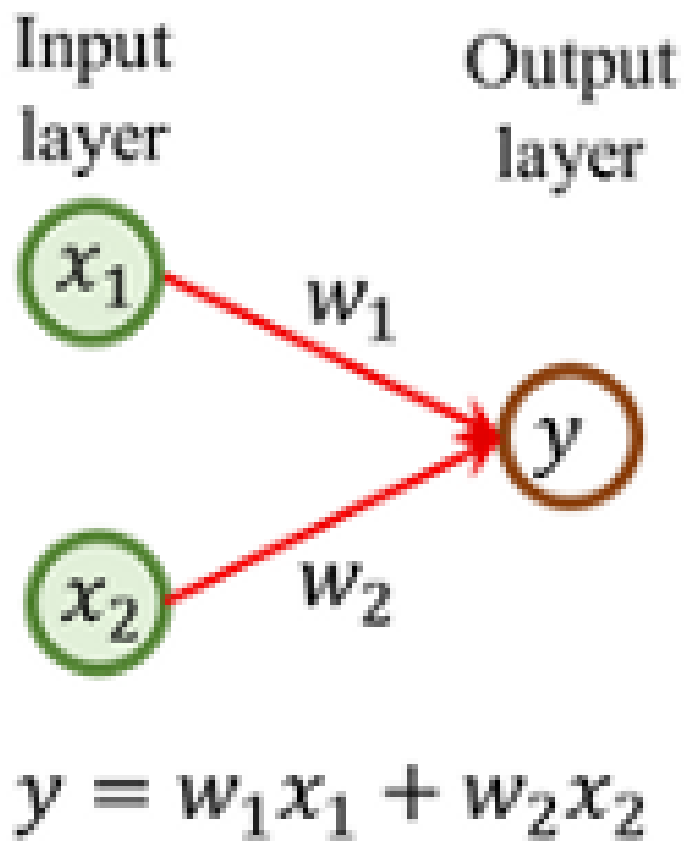


Outline

- 1. weight初始值是0
- 2. Random initialization
- 3. Xavier initialization
- 4. He initialization
- 5. Batch Normalization



weight初始值是0



Backward update:

$$w_i = w_i - \eta \Delta w_i$$

Forward:

$$\hat{y} = w_1x_1 + w_2x_2 = 0$$

weight的gradient

$$\Delta w_i = \frac{\partial E}{\partial w_i} = \frac{\frac{1}{2}y^2}{\partial w_i} = 0$$

$$E = \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}y^2$$

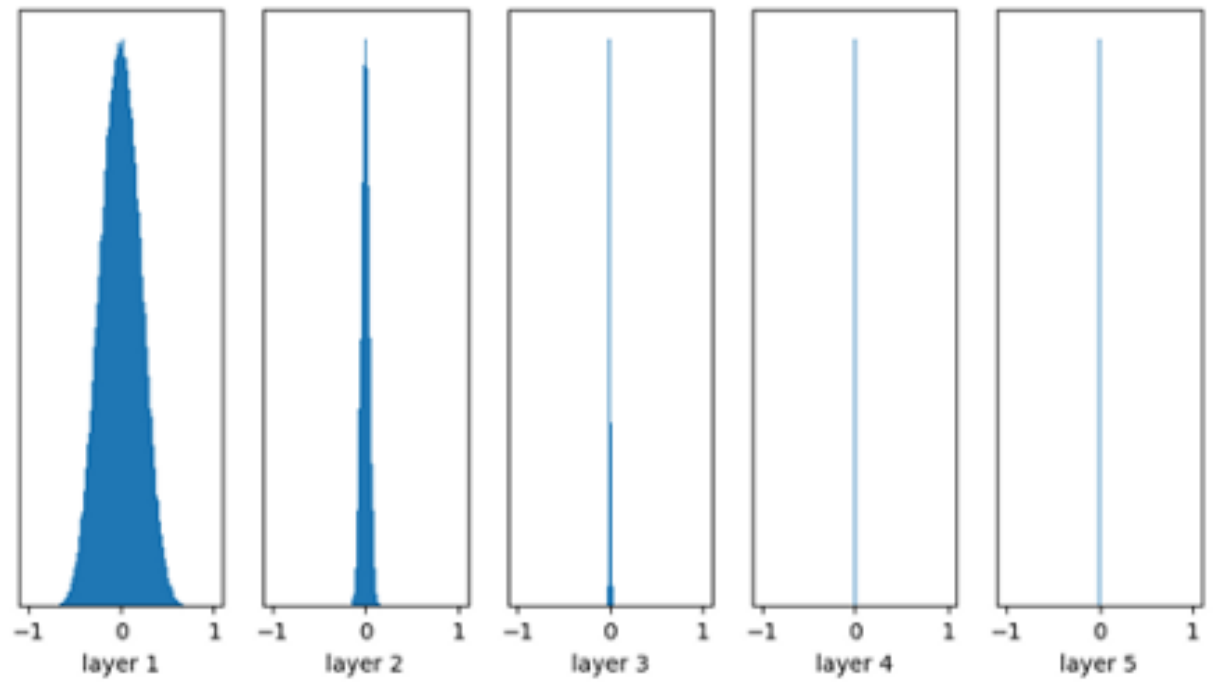


Random initialization

所以第二個最簡單的想法，初始權重用隨機方式建立。

我們建立一個6層的MLP，每一層輸出的activation用tanh。

Weight從常態分佈(平均數為0，標準差為0.01)生成

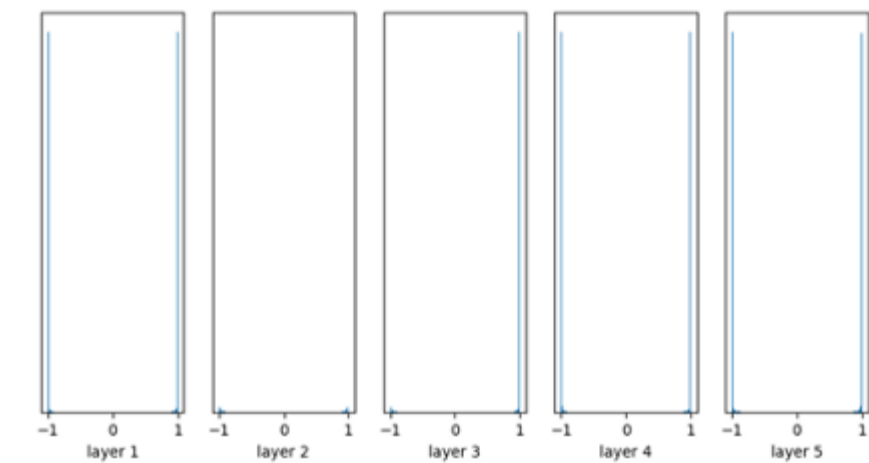


input mean 0.00065 and std 0.99949
 layer 1 mean 0.00025 and std 0.21350
 layer 2 mean -0.00002 and std 0.04516
 layer 3 mean -0.00001 and std 0.00899
 layer 4 mean -0.00000 and std 0.00168
 layer 5 mean -0.00000 and std 0.00029

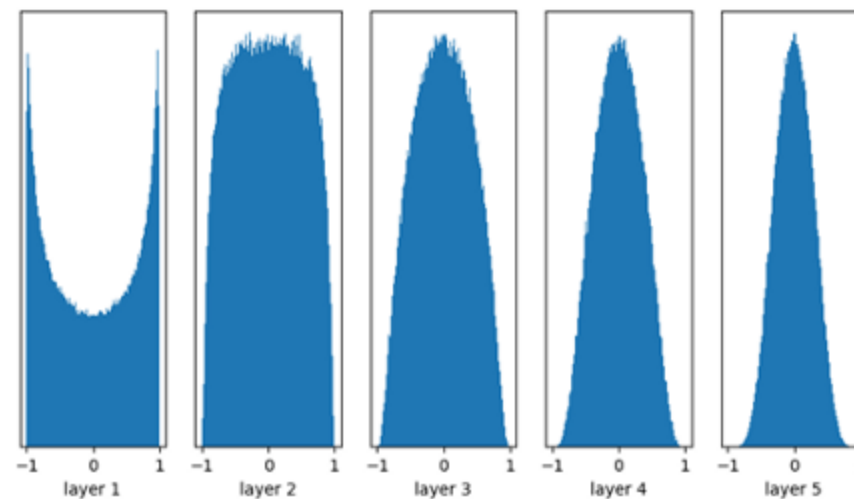


Random initialization

實驗2. 常態分佈(平均數為0，標準差為1)

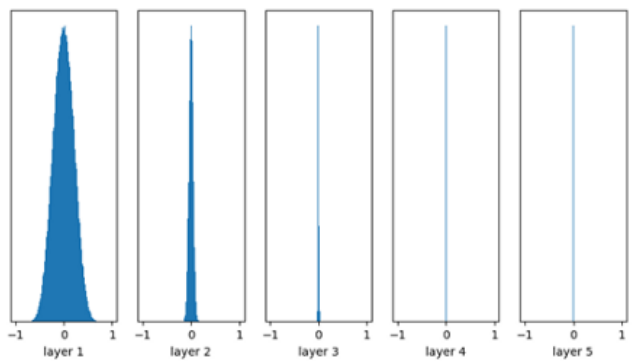


實驗3. 常態分佈(平均數為0，標準差為0.05)

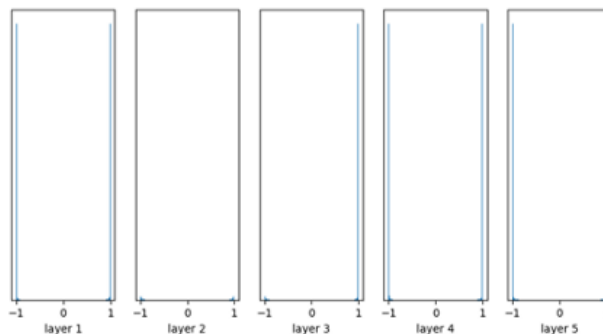


Random initialization

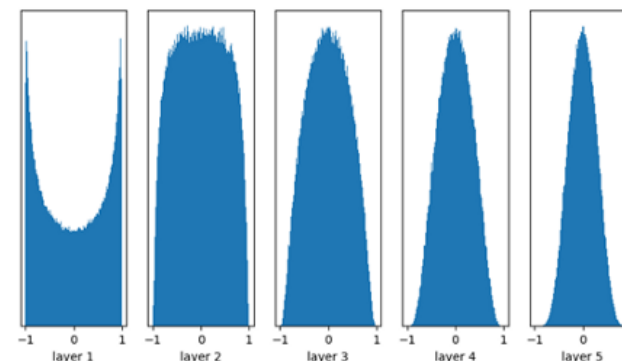
常態分佈
(平均數為0，標準差為0.02)



常態分佈
(平均數為0，標準差為1)



常態分佈
(平均數為0，標準差為0.05)



weight初始值從常態分布(也可以用均勻分布)的標準差變化會影響結果。



Xavier initialization

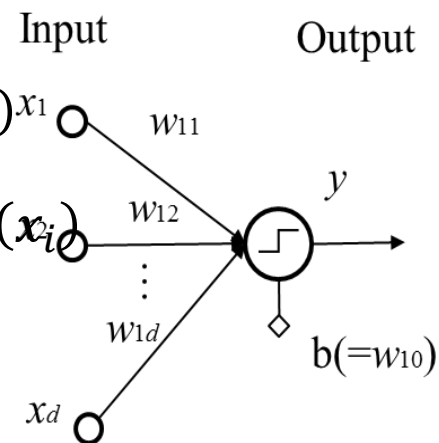
- 由Random initialization得知weight權重的生成會影響神經元的輸出太集中或是過於飽和。
- Xavier的論文(Understanding the difficulty of training deep feedforward neural networks)的想法是希望神經元輸入($x_i, i=1,2,\dots,d$)和輸出(y)的變異數(variance, 標準差的平方)保持一致。

Suppose $x_i, w_i \stackrel{iid}{\sim} \text{Distribution}(\text{mean} = 0)$

$$y = w_1x_1 + w_2x_2 + \dots + w_dx_d \text{ (bias先不考慮)}$$

$$\text{Var}(y) = \text{Var}(w_1x_1 + w_2x_2 + \dots + w_dx_d) = \sum_i^d \text{Var}(w_ix_i)$$

$$\text{Var}(w_ix_i) = E(x_i)^2 \text{Var}(w_i) + E(w_i)^2 \text{Var}(x_i) + \text{Var}(w_i) \text{Var}(x_i)$$



Xavier initialization

$$Var(y) = Var(w_1x_1 + w_2x_2 + \cdots + w_dx_d) = \sum_i^d Var(w_ix_i)$$

$$Var(w_ix_i) = Var(w_i)Var(x_i)$$

$$Var(y) = \sum_i^d Var(w_ix_i) = \sum_i^d Var(w_i)Var(x_i) = d \times Var(w_i)Var(x_i)$$

- Xavier initialization的想法是希望 $Var(y)=1$

$$Var(y) = d \times Var(w_i)Var(x_i) = 1 \Rightarrow Var(w_i) = \frac{1}{d \times Var(x_i)}$$

因為前一層的輸出我們也希望其variance=1 (所以資料前處理很常做z-score處理)

$$Var(w_i) = \frac{1}{d} = \frac{1}{n_{inputnode}}$$



Xavier initialization

同樣得程序在back-propagation也推一遍就可以得到

$$Var(w_i) = \frac{1}{n_{outputnode}}$$

為了希望forward和backward的variance可以一致，除非 $n_{inputnode} = n_{outputnode}$ ，文章就直接取

$$Var(w_i) = \frac{2}{n_{inputnode} + n_{outputnode}}$$

$$Var(y) = d \times Var(w_i) = \frac{2n_{inputnode}}{n_{inputnode} + n_{outputnode}}$$



Xavier initialization

如果採用相加除以2:

$$\frac{\left(\frac{1}{n_{inputnode}} + \frac{1}{n_{outputnode}}\right)}{2} = \left(\frac{n_{inputnode} + n_{outputnode}}{2n_{outputnode}n_{inputnode}}\right)$$

Suppose

Layer 1 node=32, Layer 2 node=64, Layer 3 node=128

$$Var(w_i) = \frac{2}{n_{inputnode} + n_{outputnode}}$$

Layer 1 = $2/(32+64)=2/96=0.021$, Layer 2 = $2/(64+128)=2/192=0.01$

Layer 1 = $(32+64)/(2*32*64)=0.023$, Layer 2 = $(32+64)/(2*64*128)=0.00586$

Note how both constraints are satisfied when all layers have the same width.

constraints: $Var(w_i) = \frac{1}{n_{inputnode}}$, $Var(w_i) = \frac{1}{n_{outputnode}}$



Xavier initialization

- 文章提到假設每一層的node數一樣

$$Var(w_i) = \frac{2}{n_{inputnode} + n_{outputnode}}$$

$$\forall i, Var\left[\frac{\partial Cost}{\partial s^i}\right] = \left[nVar[W]\right]^{d-i} Var[x] \quad (13)$$

$$\forall i, Var\left[\frac{\partial Cost}{\partial w^i}\right] = \left[nVar[W]\right]^d Var[x] Var\left[\frac{\partial Cost}{\partial s^d}\right] \quad (14)$$

所以這個方法只能盡量緩和gradient vanish/explode，不能避免。



Xavier initialization

$$w_i \stackrel{iid}{\sim} U(a, b)$$

$$var(w_i) = \frac{(b - a)^2}{12}$$

一般用均匀分布

$$w_i \stackrel{iid}{\sim} U\left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right)$$

$$var(w_i) = \frac{\left(\frac{2}{\sqrt{n}}\right)^2}{12} = \frac{1}{3n} = n \cdot var(w_i) = \frac{1}{3}$$

論文建議用他們提出來的normalized initialization

$$w \sim U\left(-\sqrt{\frac{6}{n_{inputnode} + n_{outputnode}}}, \sqrt{\frac{6}{n_{inputnode} + n_{outputnode}}}\right)$$



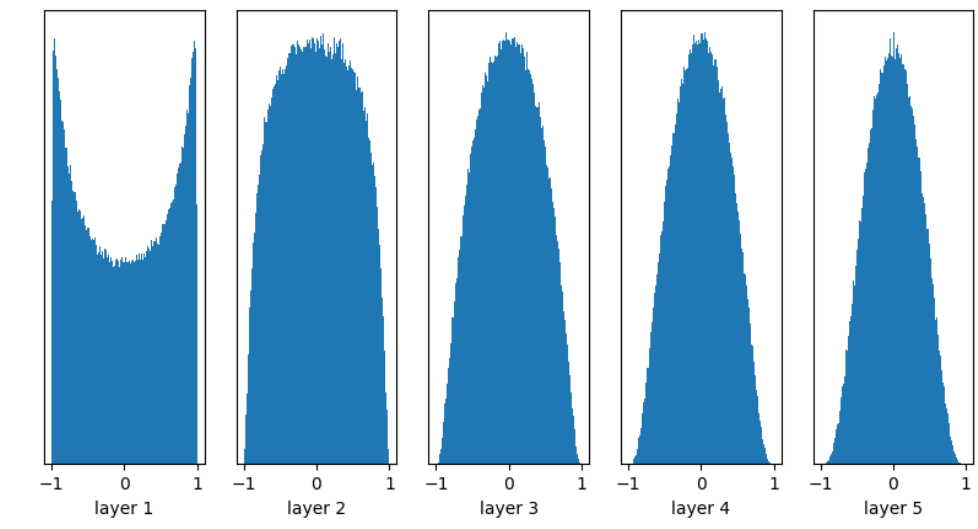
Xavier initialization

$$w_i \stackrel{iid}{\sim} U(-b, b)$$
$$\text{var}(w_i) = \frac{(b - a)^2}{12} = \frac{b^2}{3} = \frac{2}{n_{\text{inputnode}} + n_{\text{outputnode}}}$$
$$\Rightarrow b^2 = \frac{6}{n_{\text{inputnode}} + n_{\text{outputnode}}}$$
$$\Rightarrow b = \sqrt{\frac{6}{n_{\text{inputnode}} + n_{\text{outputnode}}}}$$



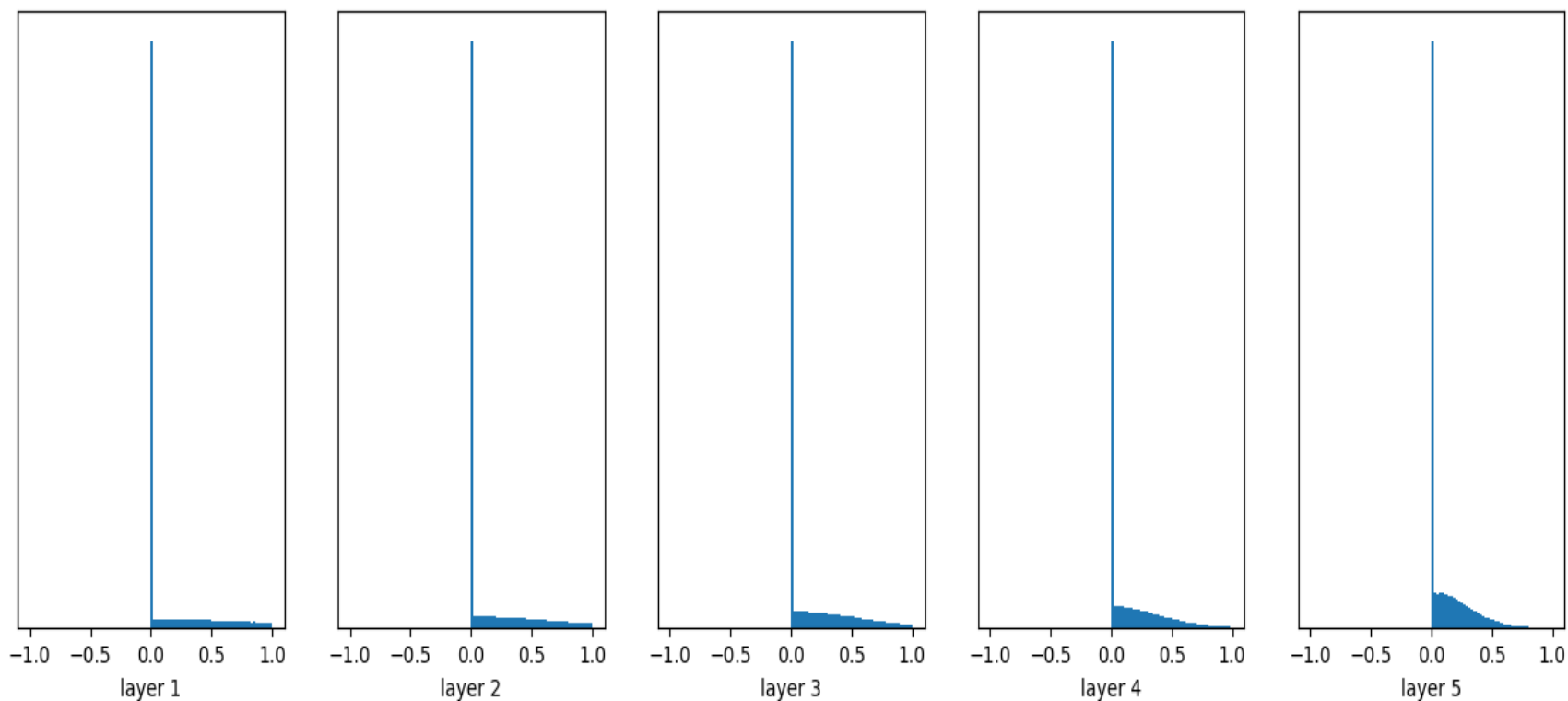
Xavier initialization

實驗一：用Xavier initialization，神經網路activation function採用tanh輸出。



Xavier initialization

實驗二: 用Xavier initialization，神經網路activation function採用ReLU輸出。後深層值會越接近0。



He initialization

He initialization為何鎧明的文章[Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification](#)

$$Var(y) = d \times Var(w_i)Var(x_i)$$

因為He initialization的推導稍微複雜一點，需要引入前層和當層的關係，所以我將公式修改成

$$Var(y_l) = n_l \times Var(w_l)Var(x_l)$$

- y_l : 第 l 層的輸出
- n_l : 第 l 層輸入神經元數量
- w_l : 第 l 層的權重
- x_l : 第 l 層的輸入，且 $x_l = f(y_{l-1})$, f : ReLU



He initialization

w_{l-1} 是對稱於0的分佈，所以 y_{l-1} 的結果也是對稱於0的分佈(平均數等於0)

$$Var(x_l) = Var(f(y_{l-1})) = \frac{1}{2} Var(y_{l-1})$$

$$Var(y_l) = \frac{n_l}{2} \times Var(w_l) Var(y_{l-1})$$

$$Var(y_L) = \frac{n_L}{2} \times Var(w_L) Var(y_{L-1})$$

$$= \frac{n_L}{2} \times Var(w_L) \times \frac{n_{L-1}}{2} Var(w_{L-1}) Var(y_{L-2}) = \dots$$

$$= Var(y_1) \left(\prod_{l=2}^L \frac{n_l}{2} Var(w_l) \right)$$

只要這個variance大於1或是小於1都會因為層數增加造成vanish或是explosion



He initialization

$$\frac{n_l}{2} \text{Var}(w_l) = 1 \Rightarrow \text{Var}(w_l) = \frac{2}{n_l}, \forall l$$

$$w_i \stackrel{iid}{\sim} U(-b, b)$$

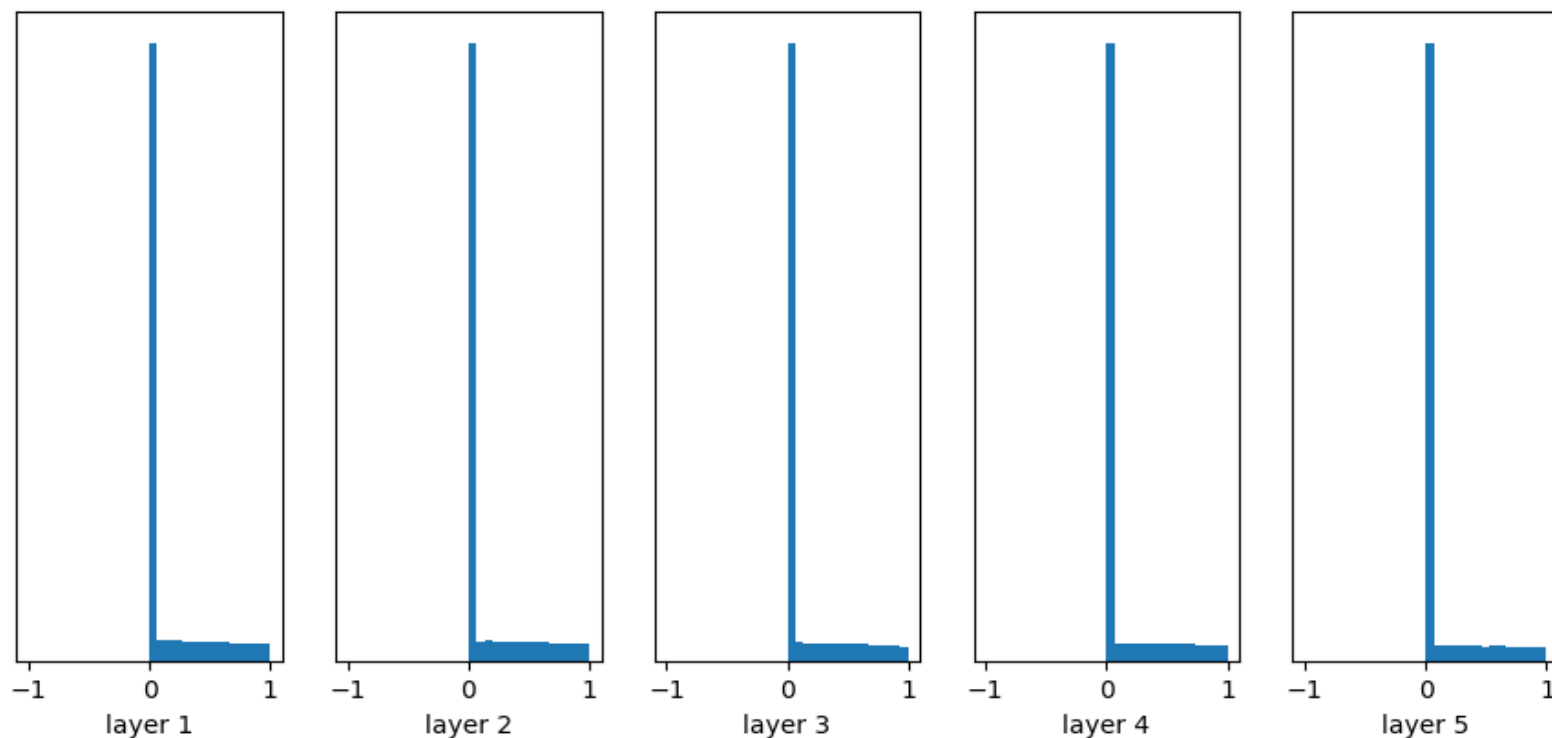
$$\text{var}(w_i) = \frac{(b - (-b))^2}{12} = \frac{2}{n_l} \Rightarrow b^2 = \frac{6}{n_l} \Rightarrow b = \sqrt{\frac{6}{n_l}}$$

$$w \sim U\left(-\sqrt{\frac{6}{n_l}}, \sqrt{\frac{6}{n_l}}\right)$$



He initialization

實驗: 剛剛 Xavier initialization 發生問題的例子，改用 He initialization，神經網路 activation function 採用 ReLU 輸出。



Batch Normalization

- **Weight Initialization**前面說了這麼多，不外乎是要怎麼有效決定 **weight** 生成時分佈的參數。
- 有沒有不管參數生成方法都可以避免發生問題的方法: **Batch Normalization(BN)**。 ([Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift](#))
- 我們剛剛前面都希望輸出的變異數等於1 ($Var(y) = 1$)

最簡單的方式統計學的z-score

假設資料是 $x \sim N(\mu, \sigma)$

$$\frac{x - \mu}{\sigma} \sim N(0, 1)$$



Batch Normalization

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

強制把值拉到標準常態: $\frac{x - \mu}{\sigma} \sim N(0, 1)$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

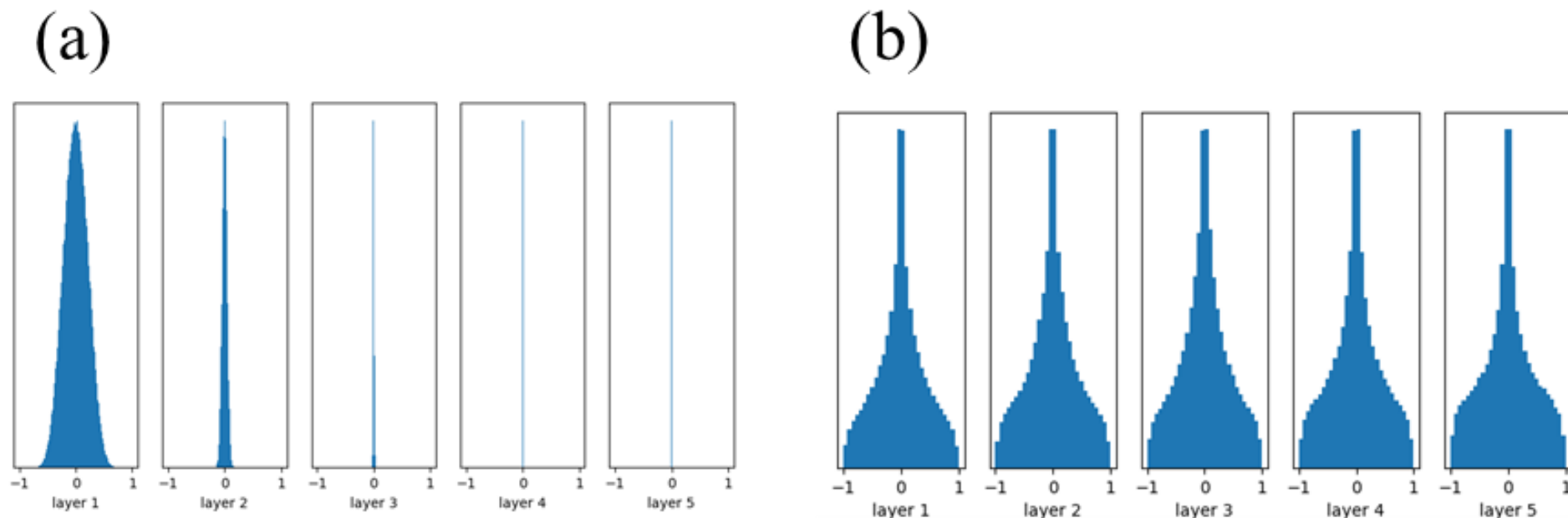
$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$



Batch Normalization

實驗1. Weight是由常態分佈隨機生成(平均數為0，標準差為0.01)。

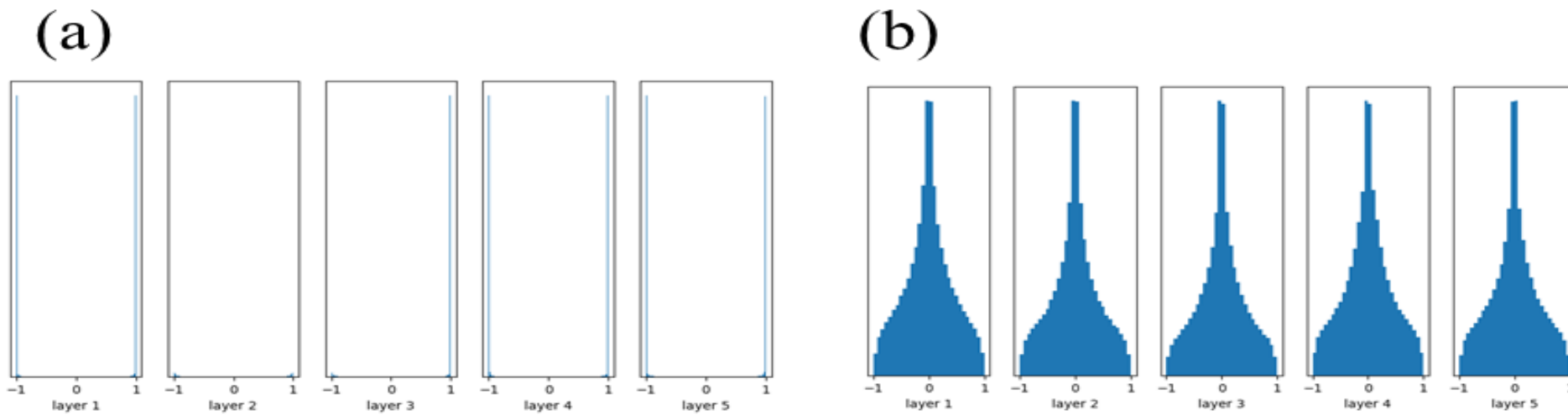


(a) 神經網路沒有加Batch normalization，(b)神經網路加入Batch normalization。



Batch Normalization

實驗2. Weight是由常態分佈隨機生成(平均數為0，標準差為1)。



(a) 神經網路沒有加Batch normalization，(b)神經網路加入Batch normalization。

