22.1.2 Backpropagation algorithm for training NN

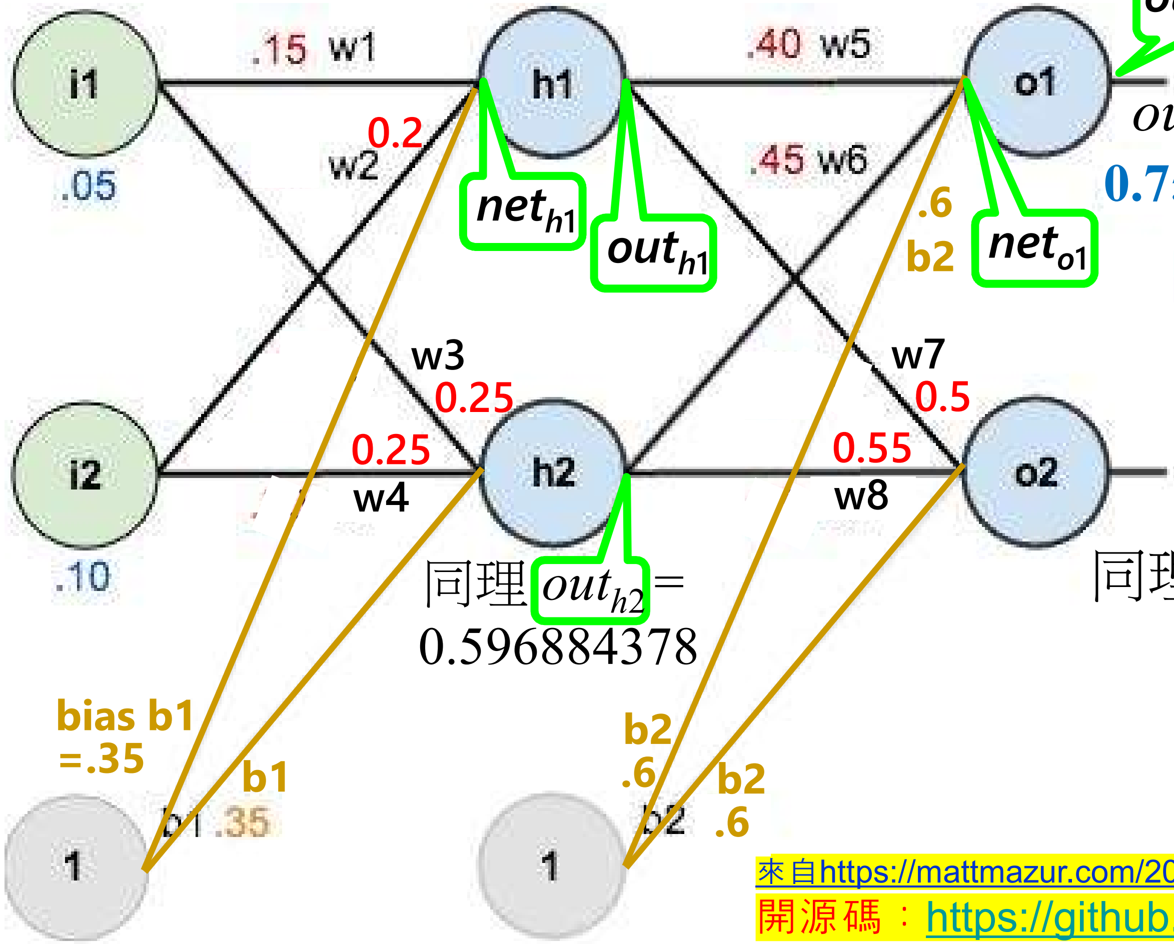
Step 1：隨機設定初始的權重w1~w8及偏置b1~b2。

1963年Vapnik提出

Step 2：輸入某個訓練資料i1~i2，求o1~o2。正確輸出是0.01和0.99，求誤差。

$net_{h1}=0.05*0.15+0.1*0.2+0.35*1=0.3775$

$$out_{h1}=\frac{1}{1+e^{-net_{h1}}}=\frac{1}{1+e^{-0.3775}}=0.593269992$$



同理 out_{h2}= 0.596884378

net_{O1}= 0.593269992*0.4+ 0.596884378*0.45+ 1*0.6=1.105905967

$$out_{o1}=\frac{1}{1+e^{-net_{o1}}}$$

=0.75136507

out_{O1}= **0.75136507**

$$E_{O1}=\frac{1}{2}(target-output)^2=\frac{1}{2}(0.01-0.75136507)^2=0.274811083$$

稱為least mean square error

同理 out_{O2}= **0.772928465**

$$E_{O2}=0.023560026$$

$E_{total}=E_{O1}+E_{O2}=0.298371109$

來自https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/

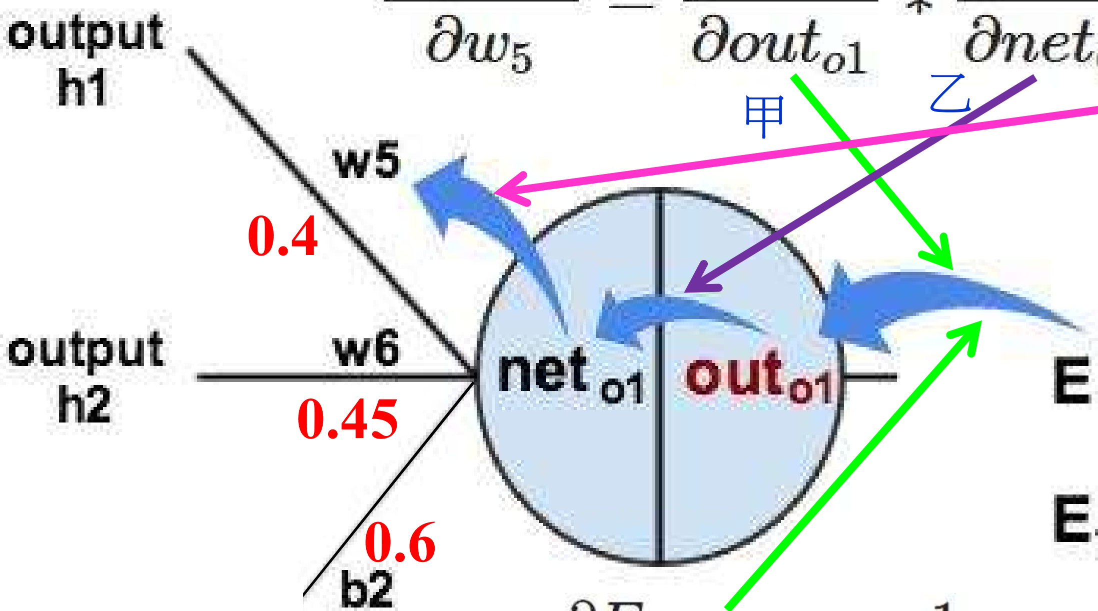
開源碼：https://github.com/mattm/simple-neural-network

Step 3：更新網路中的每一個權重，使得最終的輸出更接近**target**。

先看$\mathbf{w_5}$，我們想知道$\mathbf{w_5}$的改變會影響總誤差多少，也就是$\frac{\partial E_{total}}{\partial w_5}$。
根據**Chain rule**，我們可以得到：

$$\frac{\partial E_{total}}{\partial w_5} = \frac{\partial E_{total}}{\partial out_{o1}} * \frac{\partial out_{O1}}{\partial net_{O1}} * \frac{\partial net_{O1}}{\partial w_5}$$

甲 乙 丙



$E_{o1} = ½(target_{o1} - out_{o1})^2$

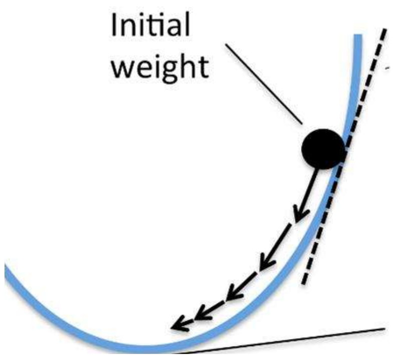
$E_{total} = E_{o1} + E_{o2}$

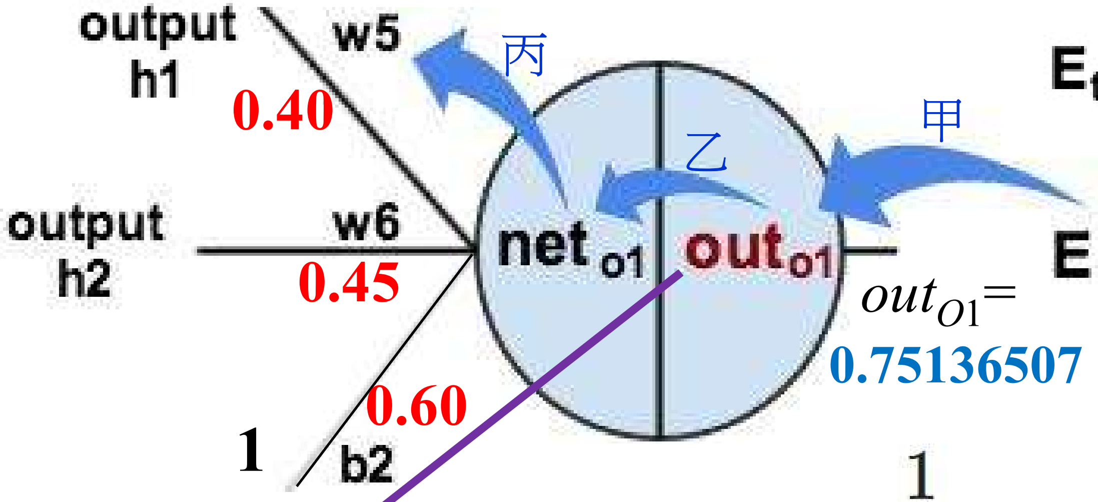
甲 $$\frac{\partial E_{total}}{\partial out_{O1}} = 2 * \frac{1}{2}(target_{O1} - \mathrm{out}_{O1})^{2-1} * -1 + 0$$

$$= -(target_{O1} - \mathrm{out}_{O1}) = -(0.01 - 0.75136507)$$

$$= 0.74136507$$

偏微分是正數(斜率是正的)表示$\mathbf{out_{O1}}$增加，$\mathbf{E_{total}}$也跟著增加。



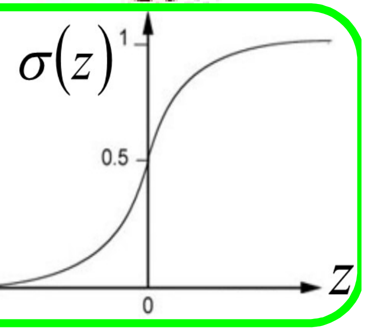


$$E_{total} = E_{o1} + E_{o2}$$

$$E_{o1} = \tfrac{1}{2}(target_{o1} - out_{o1})^2$$

Sigmoid Function

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



$$out_{O1} = \frac{1}{1+e^{-net_{O1}}}$$

乙 $$\boxed{\frac{\partial out_{O1}}{\partial net_{O1}}}=\frac{e^{-net_{O1}}}{(1+e^{-net_{O1}})^2}=\frac{1}{1+e^{-net_{O1}}}\bullet\frac{e^{-net_{O1}}}{1+e^{-net_{O1}}}=out_{O1}\bullet\left(\frac{1+e^{-net_{O1}}}{1+e^{-net_{O1}}}-\frac{1}{1+e^{-net_{O1}}}\right)$$

$$= out_{O1}(1-out_{O1}) = 0.75136507(1-0.75136507) = 0.186815602$$

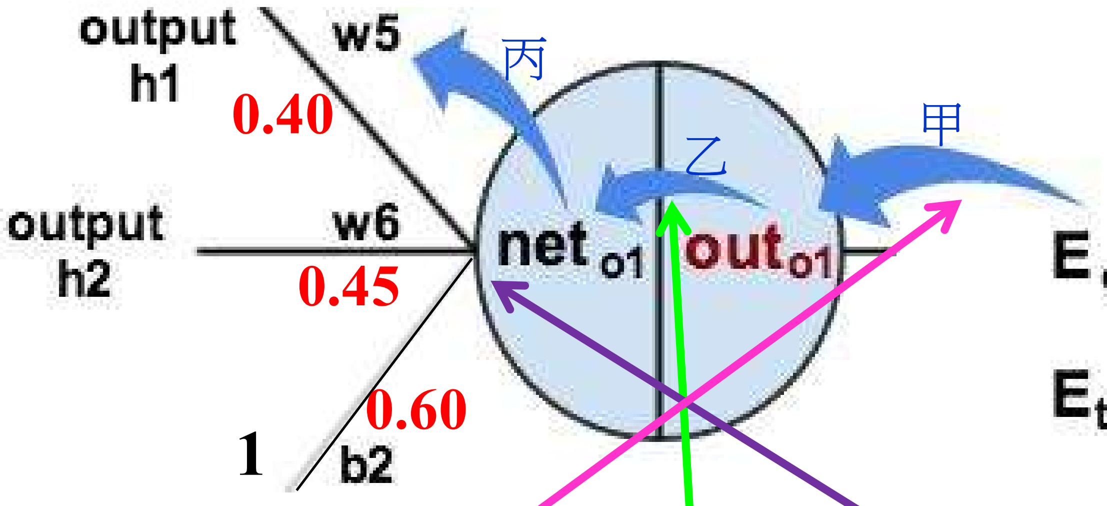
丙 $$net_{O1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

$$\frac{\partial net_{O1}}{\partial w_5} = out_{h1} = \mathbf{0.593269992}$$

$$\frac{\partial E_{total}}{\partial w_5} = \overset{甲}{\frac{\partial E_{total}}{\partial out_{o1}}} * \overset{乙}{\boxed{\frac{\partial out_{O1}}{\partial net_{O1}}}} * \overset{丙}{\frac{\partial net_{O1}}{\partial w_5}}$$

$$= \overset{甲}{-(target_{O1} - out_{O1})} * \overset{乙}{out_{O1}(1-out_{o1})} * \overset{丙}{out_{h1}}$$

$$= \overset{甲}{0.74136507} * \overset{乙}{0.186815602} * \overset{丙}{0.593269992} = \color{red}{0.082167041}$$



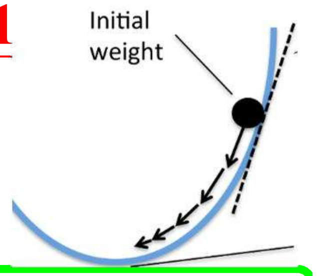
$$E_{o1} = \frac{1}{2}(target_{o1} - out_{o1})^2$$

$$E_{total} = E_{o1} + E_{o2}$$

$$\frac{\partial E_{total}}{\partial w_5} = \overset{甲}{\frac{\partial E_{total}}{\partial out_{o1}}} * \overset{乙}{\frac{\partial out_{O1}}{\partial net_{O1}}} * \overset{丙}{\frac{\partial net_{O1}}{\partial w_5}}$$

$$= \overset{甲}{-}(\boxed{target_{O1}} - \boxed{out_{O1}}) * \overset{乙}{out_{O1}}(1 - out_{o1}) * \overset{丙}{\boxed{out_{h1}}}$$

$$= \underline{0.74136507 * 0.186815602} * 0.593269992 = \underline{0.082167041}$$



偏微分是正數，即斜率是正的，表示w_5增加，總誤差E_{total}也會跟著增加。因此我們應減少w_5之值， 使總誤差E_{total}降低：

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = \boxed{0.4} - 0.5 * 0.082167041 = \boxed{0.35891648}$$

learning rate

這個規則稱為**LMS(Least Mean Squares)**更新規則， 或**Widrow-Hoff learning rule**。

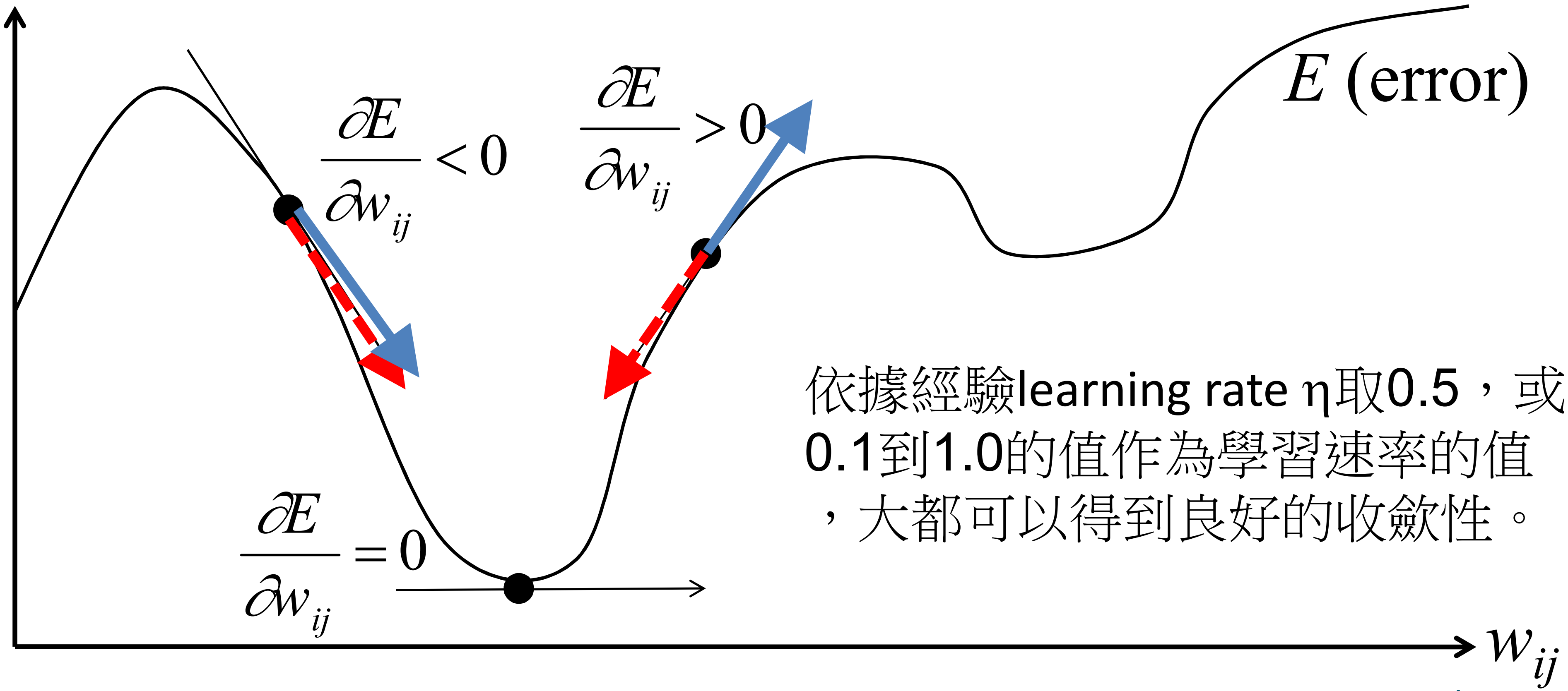
Gradient Descent(梯度下降法)

Gradient direction:

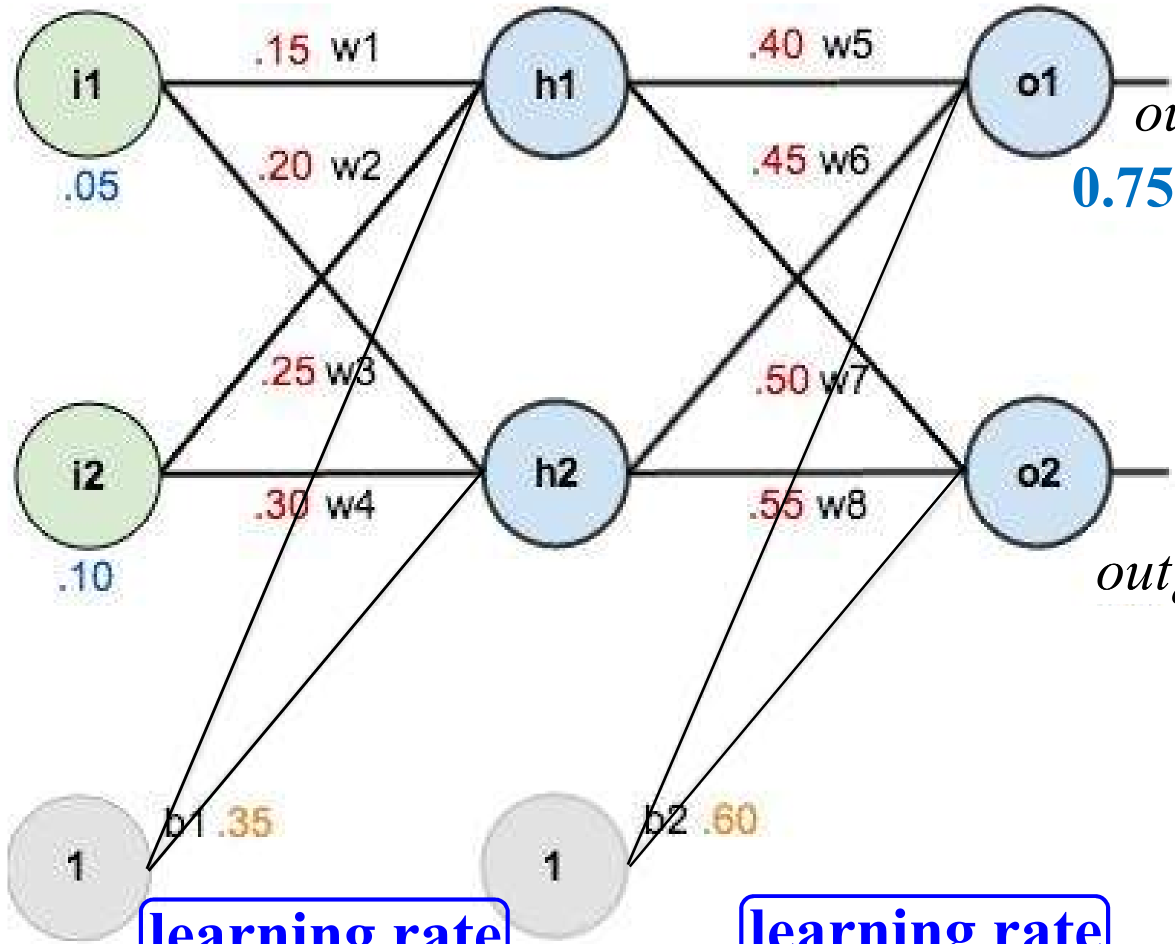
Descent direction:

$$\Delta w_{ij} = -\eta \frac{\partial E}{\partial w_{ij}}$$

(η is learning rate)



依據經驗learning rate η取0.5，或0.1到1.0的值作為學習速率的值，大都可以得到良好的收斂性。



out_{o1}= **0.75136507**

out_{o2}= **0.772928465**

learning rate

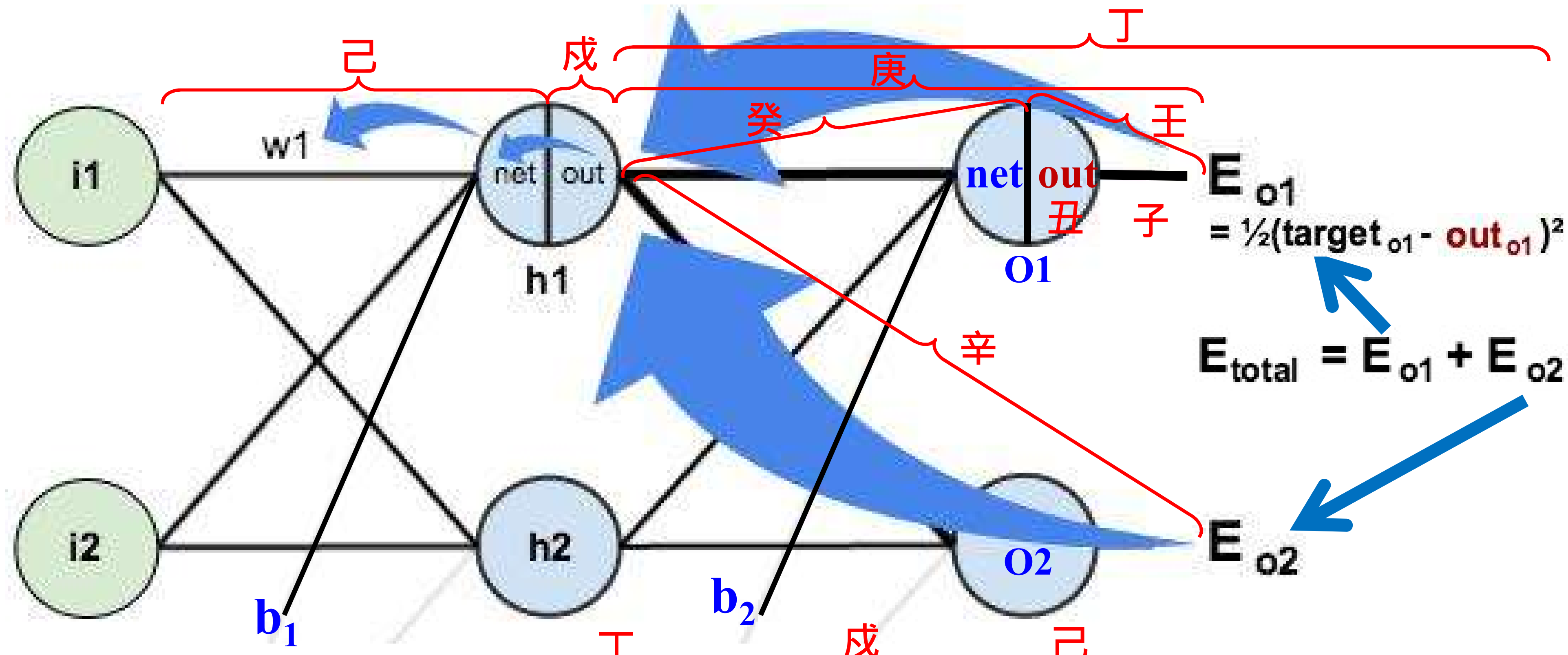
learning rate

$$w_5^+ = w_5 - \eta * \frac{\partial E_{total}}{\partial w_5} = 0.4 - 0.5 * 0.082167041 = \boxed{0.35891648}$$

同理更新

$$\left.\begin{aligned} w_6^+ &= 0.408666186 \\ w_7^+ &= 0.511301270 \\ w_8^+ &= 0.561370121 \end{aligned}\right\}$$

在用到***w5***、***w6***、***w7***和***w8***的地方，仍使用舊的權重，而不是逐一更新後的權重。



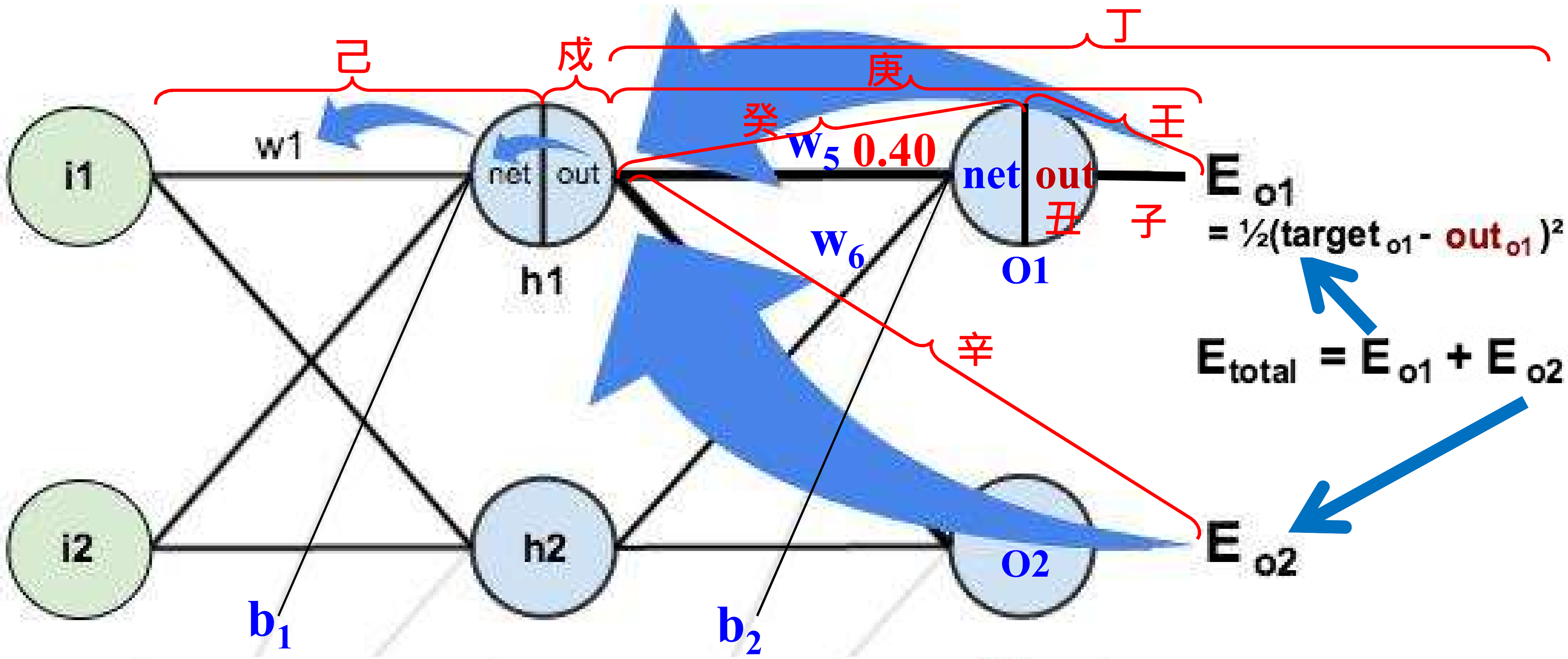
再看w_1如何更新: $\frac{\partial E_{total}}{\partial w_1} = \overset{\text{丁}}{\frac{\partial E_{total}}{\partial out_{h_1}}} * \overset{\text{戊}}{\frac{\partial out_{h_1}}{\partial net_{h1}}} * \overset{\text{己}}{\frac{\partial net_{h1}}{\partial w_1}}$ //根據Chain rule

$$\text{丁}\frac{\partial E_{total}}{\partial out_{h1}} = \text{庚}\frac{\partial E_{O1}}{\partial out_{h_1}} + \text{辛}\frac{\partial E_{O2}}{\partial out_{h1}}$$

注意w_1有兩條路徑影響E_{total}

$$\text{庚}\frac{\partial E_{O1}}{\partial out_{h1}} = \text{壬}\frac{\partial E_{O1}}{\partial net_{O1}} * \text{癸}\frac{\partial net_{O1}}{\partial out_{h1}}$$

$$\text{壬}\frac{\partial E_{O1}}{\partial net_{O1}} = \text{子}\frac{\partial E_{O1}}{\partial out_{O1}} * \text{丑}\frac{\partial out_{O1}}{\partial net_{O1}} = \underline{0.74136507 * 0.186815602} = 0.138498562$$



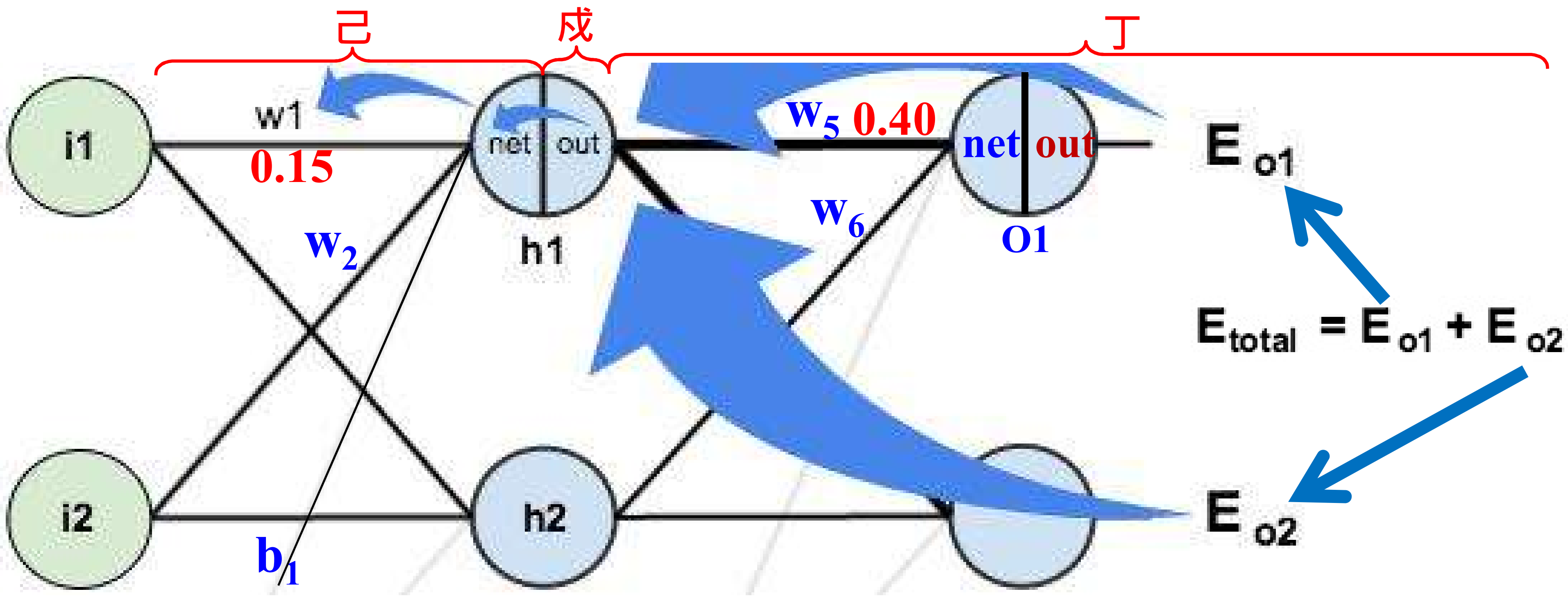
$$net_{O1} = w_5 * out_{h1} + w_6 * out_{h2} + b_2 * 1$$

癸 $$\frac{\partial net_{O1}}{\partial out_{h1}} = w_5 = 0.40$$

庚 $$\frac{\partial E_{O1}}{\partial out_{h1}} = \overset{壬}{\frac{\partial E_{O1}}{\partial net_{O1}}} * \overset{癸}{\frac{\partial net_{O1}}{\partial out_{h1}}} = 0.138498562 * 0.40 = 0.055399425$$

同理可得

丁 $$\frac{\partial E_{total}}{\partial out_{h1}} = \overset{庚}{\frac{\partial E_{O1}}{\partial out_{h_1}}} + \overset{辛}{\frac{\partial E_{O2}}{\partial out_{h1}}} = 0.055399425 + (-0.019049119) = 0.036350306$$



$$out_{h1} = \frac{1}{1+e^{net_{h1}}}$$

戊 $\frac{\partial out_{h1}}{\partial net_{h1}} = out_{h1}(1 - out_{h1}) = 0.59326999(1 - 0.59326999) = 0.241300709$

$net_{h1} = w_1 * i_1 + w_2 * i_2 + b_1 * 1$ 故 $\frac{\partial net_{h1}}{\partial w_1} = i_1 = 0.05$

$$\frac{\partial E_{total}}{\partial w_1} = \overset{丁}{\frac{\partial E_{total}}{\partial out_{h_1}}} * \overset{戊}{\frac{\partial out_{h_1}}{\partial net_{h1}}} * \overset{己}{\frac{\partial net_{h1}}{\partial w_1}} = 0.036350306 * 0.241300709 * 0.05 = 0.000438568$$

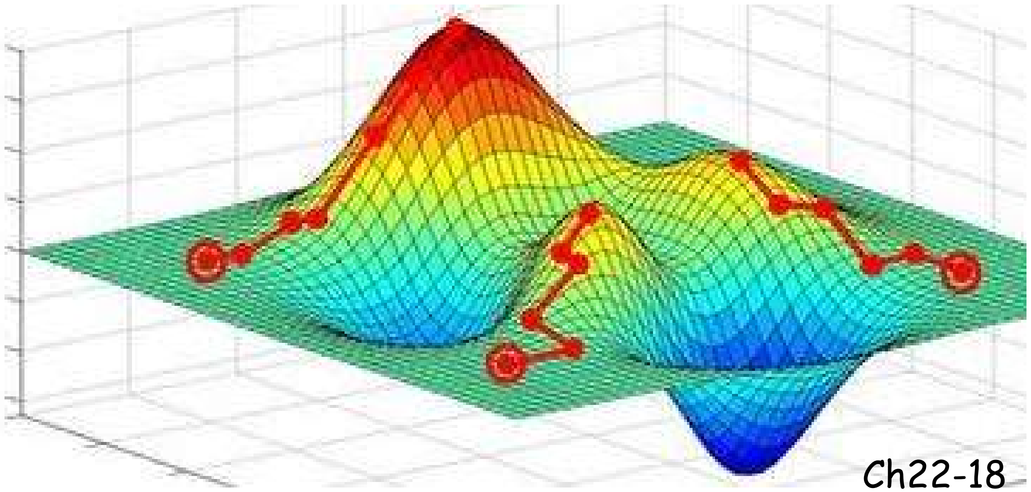
更新w_1： $w_1^+ = w_1 - \eta * \frac{\partial E_{total}}{\partial w_1} = 0.15 - 0.5 * 0.000438568 = 0.149780716$

同理更新：

w_2=0.2 **w_3=0.25** **w_4=0.3**

$$w_2^+ = 0.19956143 \quad w_3^+ = 0.24975114 \quad w_4^+ = 0.29950229$$

- 現在已經更新所有的**w**權重。**(**我們沒有更動**b_1**及**b_2)** 練習
- 在最初，輸入為**0.05**和**0.1**的時候，網路的誤差為**0.298371109**。 repl.it 親自檢視、並執行開源碼看看。
- 經過第一次 **Backpropagation** 後，誤差降低到了**0.291027924**，看來降得不多。
- 在重複這個過程**10000**次以後，網路的誤差就降到了**0.0000351085**。 開源碼：https://github.com/mattm/simple-neural-network
- 這個時候，當我們把**0.05**和**0.1**再輸入進去，兩個神經元的輸出為**0.015912196**(理想值**0.01**)和**0.984065734**(理想值 **0.99**)。



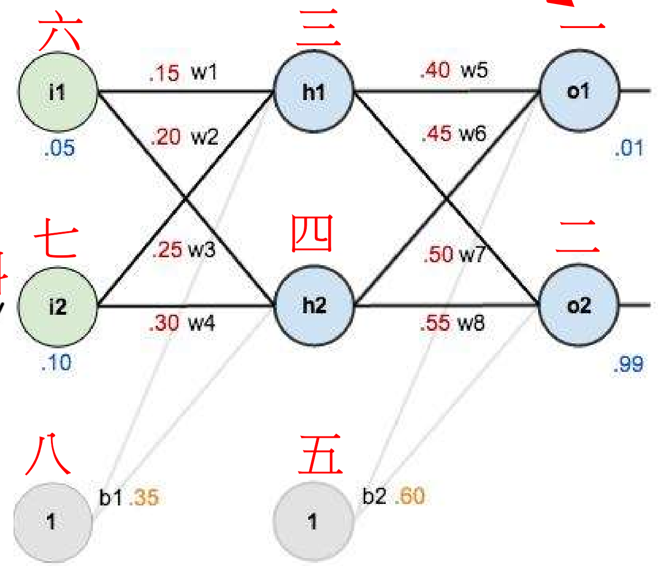
function BACK-PROP-LEARNING(*examples*, *network*) **returns** a neural network //反向傳播學習法

inputs: *examples*, a set of examples, each with input vector $\mathbf{x}$ and output vector $\mathbf{y}$

network, a multilayer network with L layers, weights $w_{i,j}$, activation function g

local variables: Δ, a vector of errors, indexed by network node

△[j]的計算次序：



repeat

  for each weight $w_{i,j}$ in *network* **do**

    $w_{i,j} \leftarrow$ a small random number // 隨機設定初始權重

  for each example $(\mathbf{x}, \mathbf{y})$ **in** *examples* **do** //輸入某個訓練資料

    /* *Propagate the inputs forward to compute the outputs* */

    for each node i in the input layer **do**

      $a_i \leftarrow x_i$ //讀取輸入層資料

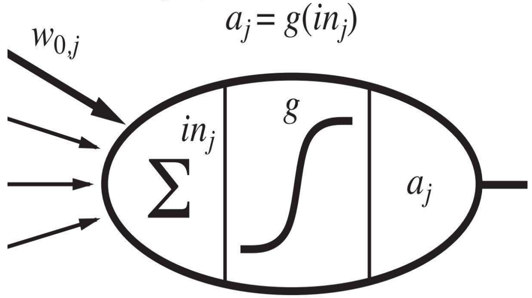
    for $\ell = 2$ **to** L **do**

      for each node j in layer ℓ **do**

        $in_j \leftarrow \sum_i w_{i,j}\, a_i$

        $a_j \leftarrow g(in_j)$

//計算中間層資料，
// 正向傳遞預測的值



    /* *Propagate deltas backward from output layer to input layer* */

    for each node j in the output layer **do**

      $\Delta[j] \leftarrow g'(in_j) \times (y_j - a_j)$

//計算輸出層誤差

    for $\ell = L - 1$ **to** 1 **do**

      for each node i in layer ℓ **do**

        $\Delta[i] \leftarrow g'(in_i) \sum_j w_{i,j}\, \Delta[j]$

//由後往前計算中間層誤差，**backpropate errors**
//反向傳遞誤差的值

    /* *Update every weight in network using deltas* */

// $a_j = g(\sum_{i=0}^{n} w_{i,j} a_i)$

    for each weight $w_{i,j}$ in *network* **do**

      $w_{i,j} \leftarrow w_{i,j} + \alpha \times a_i \times \Delta[j]$

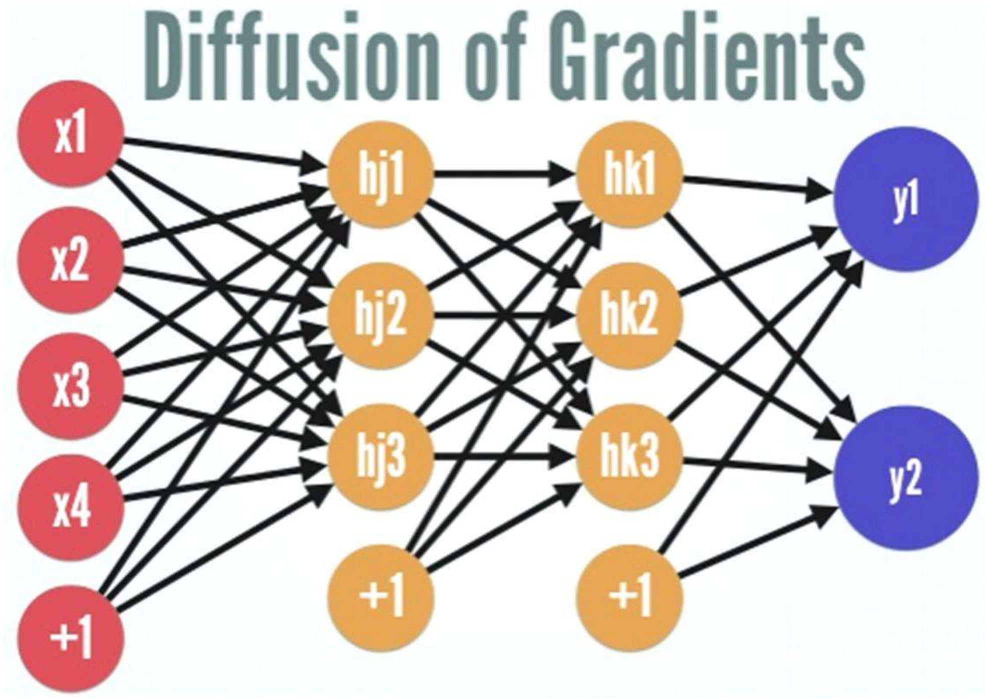
//更新網路中的每一個權重， **α 是learning rate**

until some stopping criterion is satisfied //結束條件：總誤差夠小、權值變化很小、執行逾時。

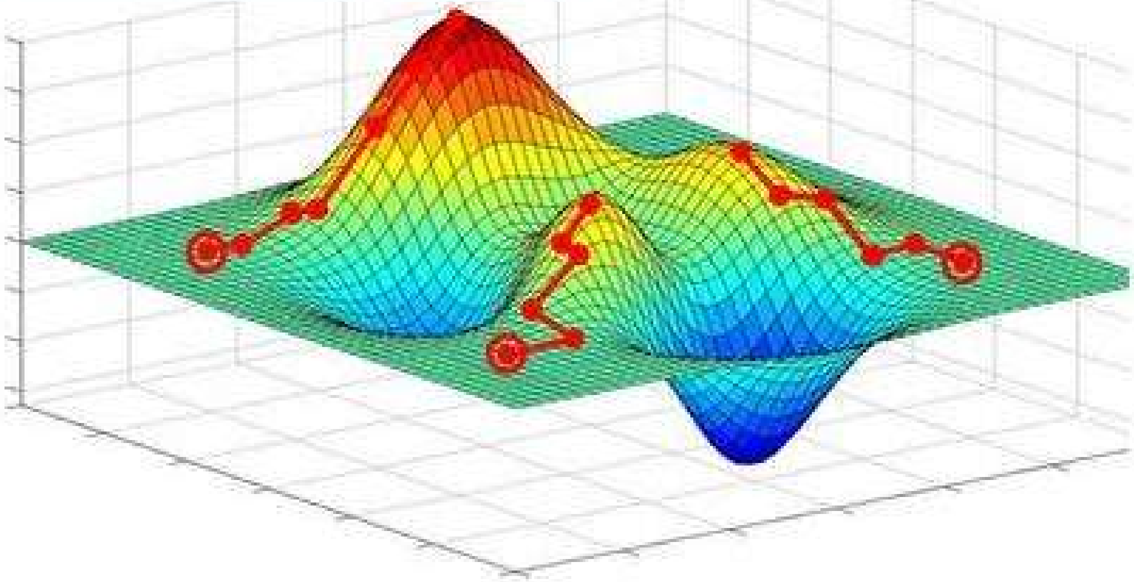
return *network*

1963年Vapnik 提出

BP演算法的問題

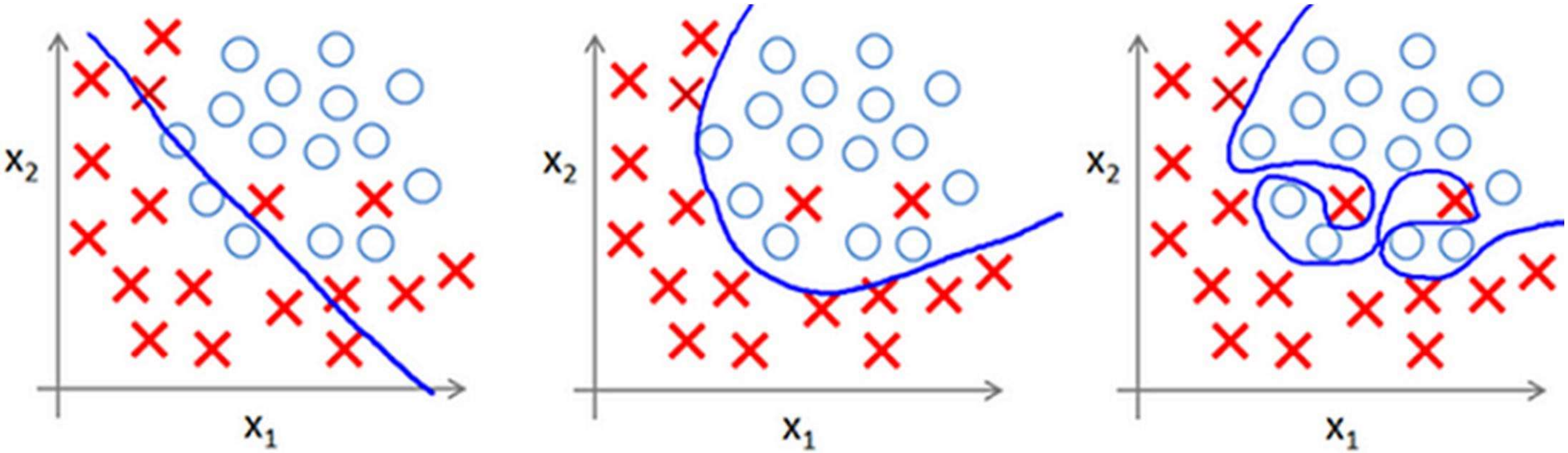


◆gradient diffusion（梯度擴散，又稱 vanishing gradient）：誤差傳播到前面的layer將使梯度的幅度會急劇減小，導致前幾層神經元的權重更新非常緩慢，變成了前幾層相對固定，只能改變最後幾層的結果，很容易陷入局部最優解。



BP演算法的問題

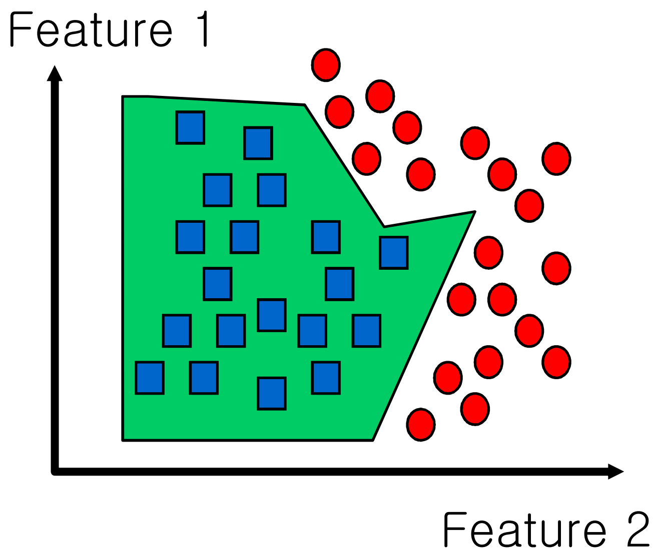
◆overfitting（過度擬合）：訓練進行的太徹底，把訓練資料的所有特徵幾乎都學到了。只針對訓練資料集合的局部特徵，在測試時反而結果不佳。例如：訓練天鵝(或非天鵝)的特徵，知道了天鵝是有翅膀的，天鵝的嘴巴是長的彎的，脖子是長的有點曲度，體型像一個"2"。但很不巧訓練的天鵝全是白色的，於是機器會認為白的(局部特徵)才是天鵝，以後看到黑的天鵝就會認為那不是天鵝。

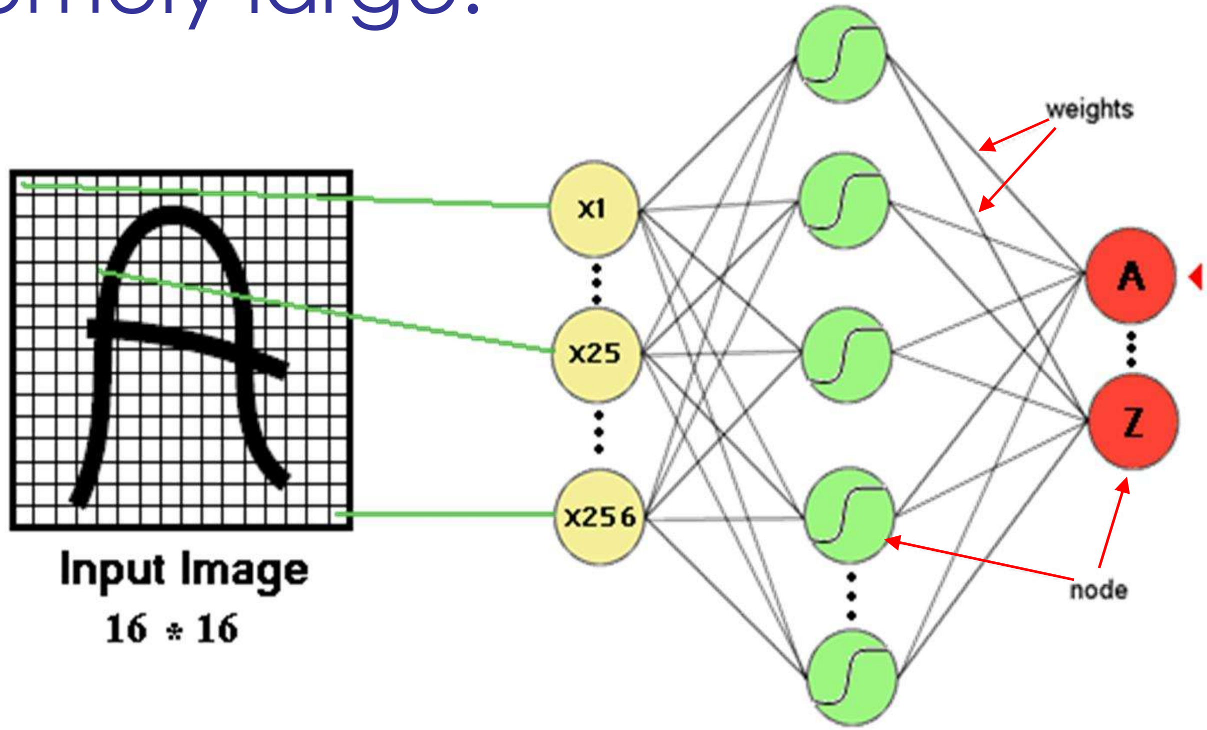


underfitting　balanced　overfitting

Drawbacks of **Multi-layer** networks

- The number of trainable parameters becomes extremely large.





- Fully connected network of sufficient size can produce outputs that are invariant with respect to such variations.
  - But training time? Network size? Free parameters?