Machine Learning Techniques

Shyi-Chyi Cheng (鄭錫齊)
Email:csc@mail.ntou.edu.tw
Tel: 02-24622192-6653

章節目錄

誤差反向傳播法

學習重點

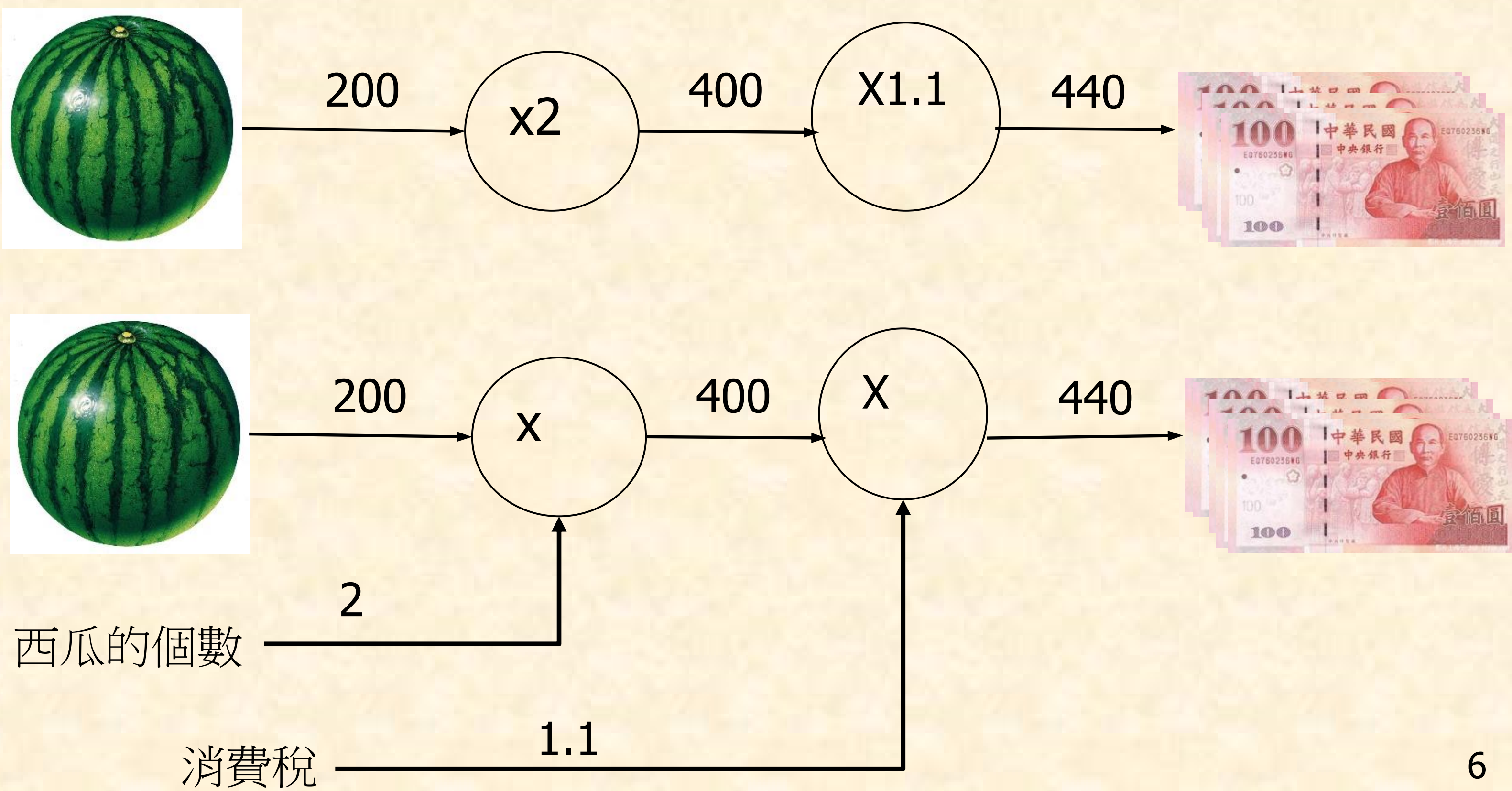
- 使用計算圖，使用視覺化方式掌握計算過程
- 計算圖的節點由局部計算圖構成，組合所有局部計算圖構成可完成全部計算
- 利用計算圖的正向傳播進行一般計算；利用計算圖的反向傳播計算各節點的微分。
- 把神經網路的構成元素當成'層'來執行處理，可以快速計算梯度(誤差反向傳播法)
- 比較數值微分與誤差反向傳播，可以確認誤差反向傳播法的執行過程有沒有錯誤。

反向傳播(Backpropagation)

- Phase 1: Propagation
  - Forward pass, generating output values
  - Backward pass: calculating the gradients
- Phase 2: Weight update
  - Updating the weights by a ratio of the weight’s gradient. This ratio is also called learning rate, η.
  - $\Delta w_{ij} = -\eta \quad E / \quad w_{ij}$

計算圖(Computational Graph)

- ❖ 計算圖是利用圖表呈現計算過程。
- ❖ 圖形資料結構由節點(Node)及節點對的連接邊(Edge)組成。
- ❖ 問題1:用計算圖表達1個購物支付流程

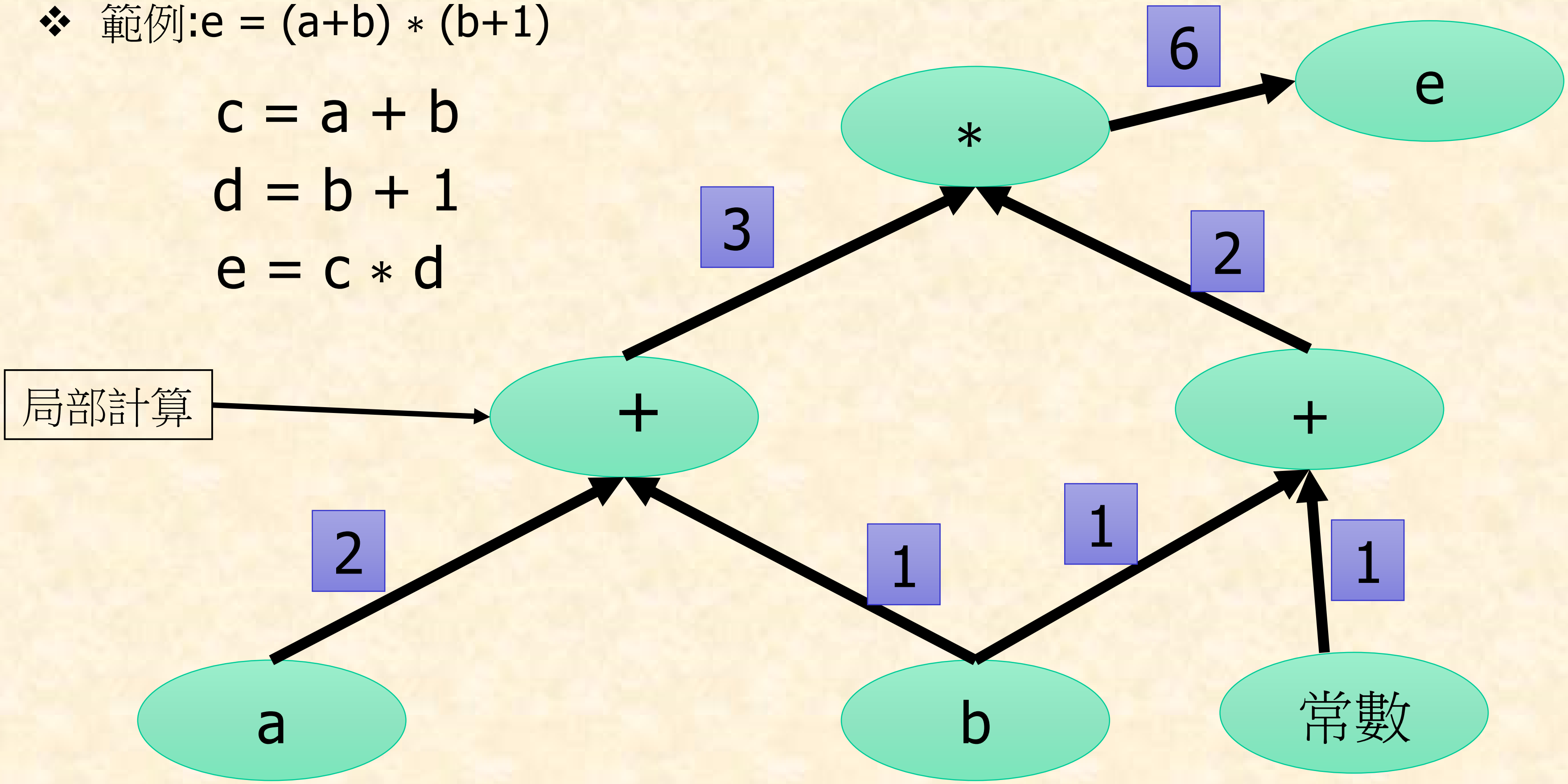


計算圖使用流程

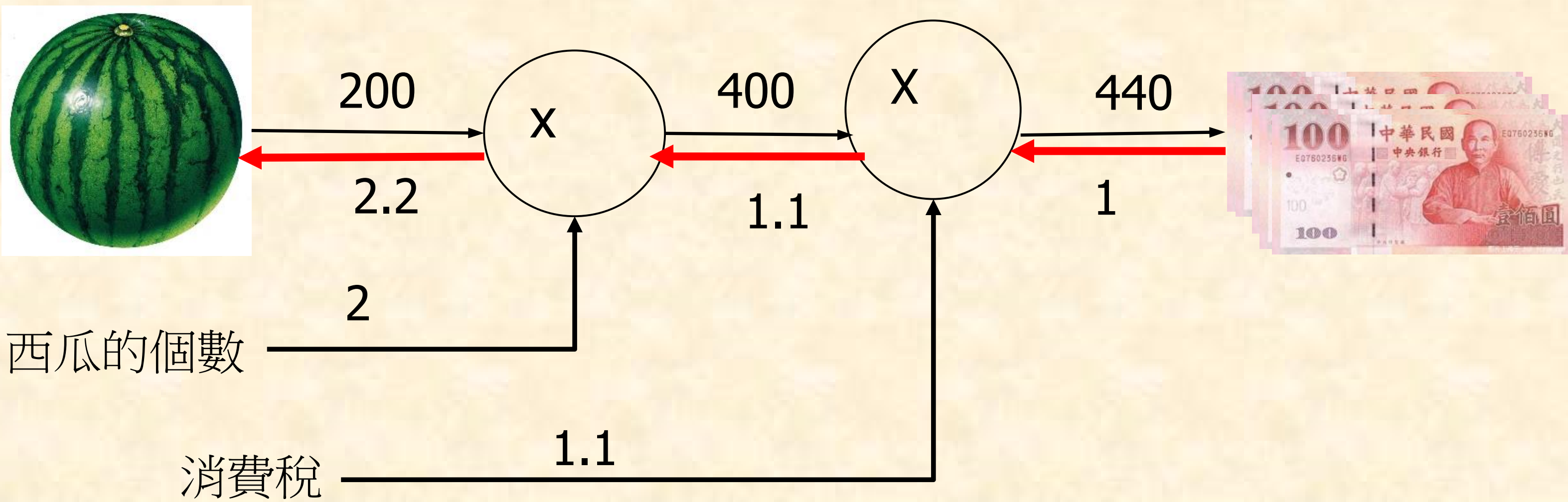
1. 建立計算圖
2. 在計算圖上，由左到右進行計算(正向傳播；forward propagation)

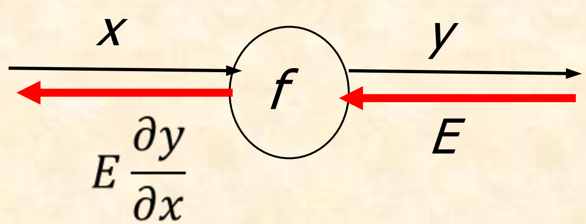
❖ 範例:$e = (a+b) * (b+1)$

$$c = a + b$$
$$d = b + 1$$
$$e = c * d$$



反向傳播計算圖





反向傳播法用來反向傳播微分值

連鎖律(Chain Rule)

Case 1

$$z = f(x) \Rightarrow y = g(x) \quad z = h(y)$$

$$x \xrightarrow{g} y \xrightarrow{h} z$$

$$\Delta x \to \Delta y \to \Delta z$$

$$\frac{dz}{dx} = \frac{dz}{dy}\frac{dy}{dx}$$

Case 2

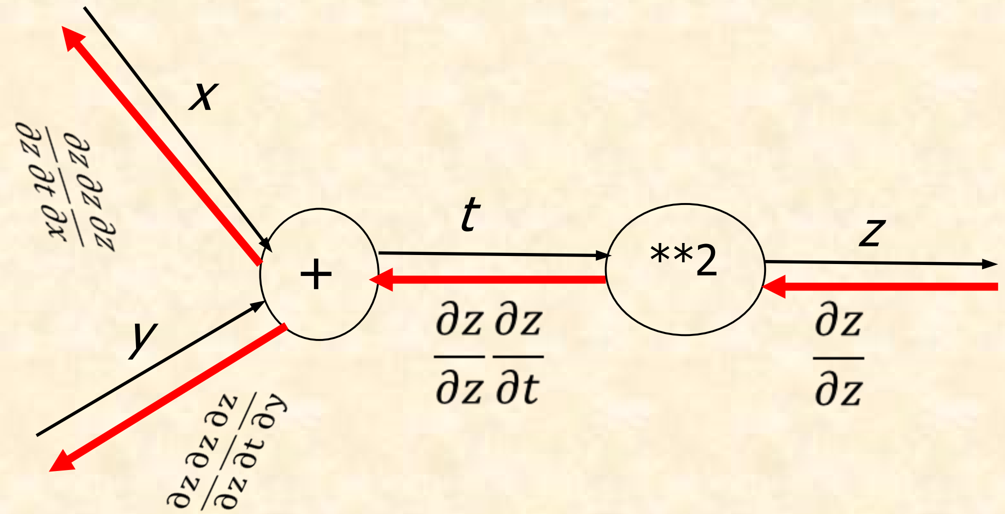
$$z = f(s) \Rightarrow x = g(s) \quad y = h(s) \quad z = k(x, y)$$

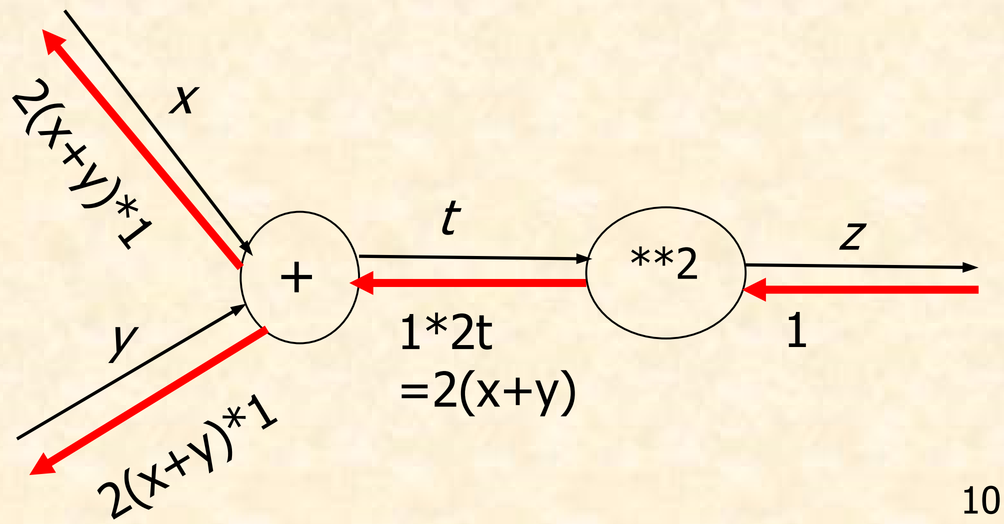
$$\begin{tikzcd}
& x \arrow[dr] & \\
s \arrow[ur, "g"] \arrow[dr, "h"'] & & z \\
& y \arrow[ur, "k"'] &
\end{tikzcd}$$

$$\begin{tikzcd}
& \Delta x \arrow[dr] & \\
\Delta s \arrow[ur] \arrow[dr] & & \Delta z \\
& \Delta y \arrow[ur] &
\end{tikzcd}$$

連鎖律與計算圖

- 計算式: $\begin{cases} z = t^2 \\ t = x + y \end{cases}$

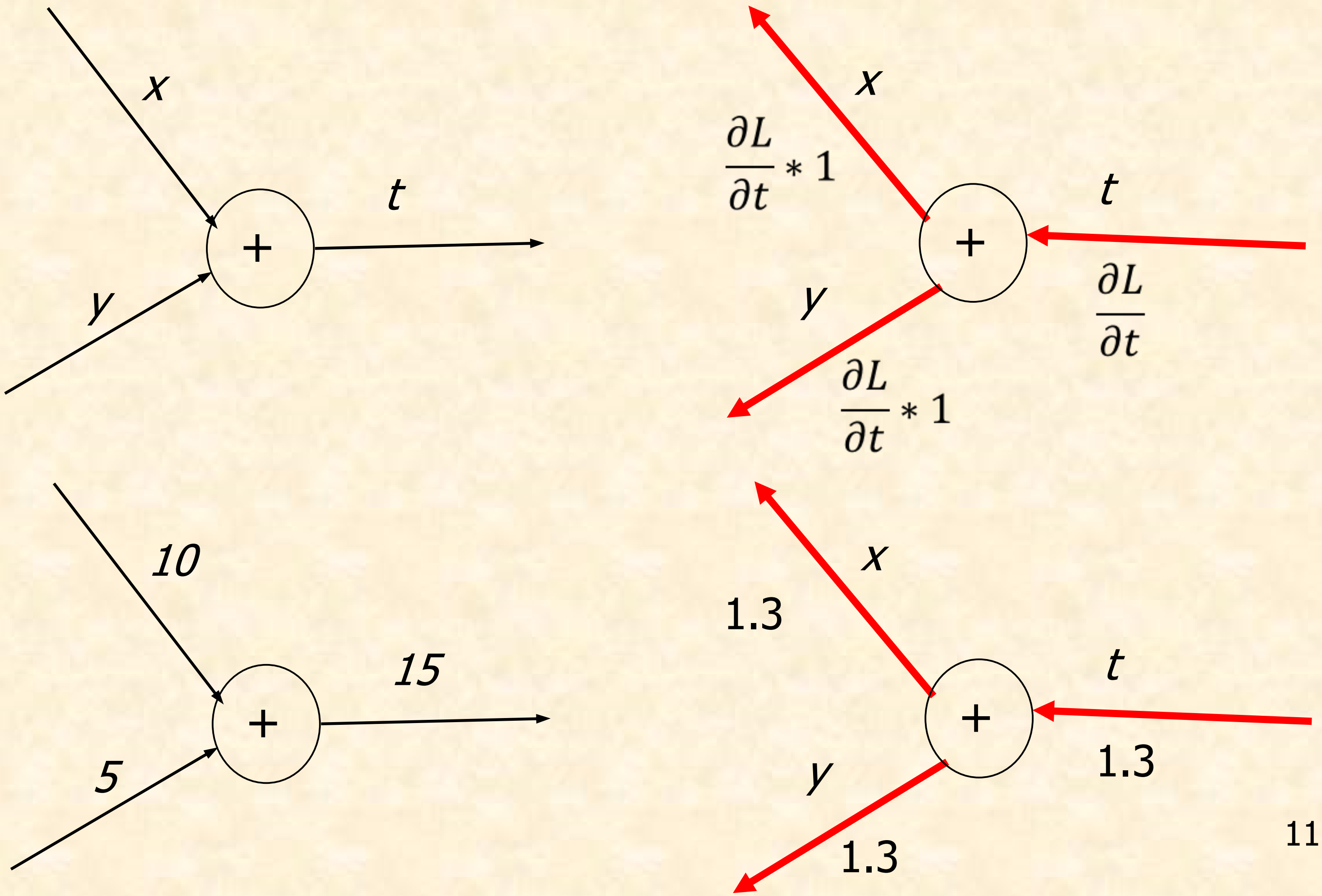




反向傳播(Backpropagation)

$$\begin{cases} z = t^2 \\ t = x + y \end{cases}$$

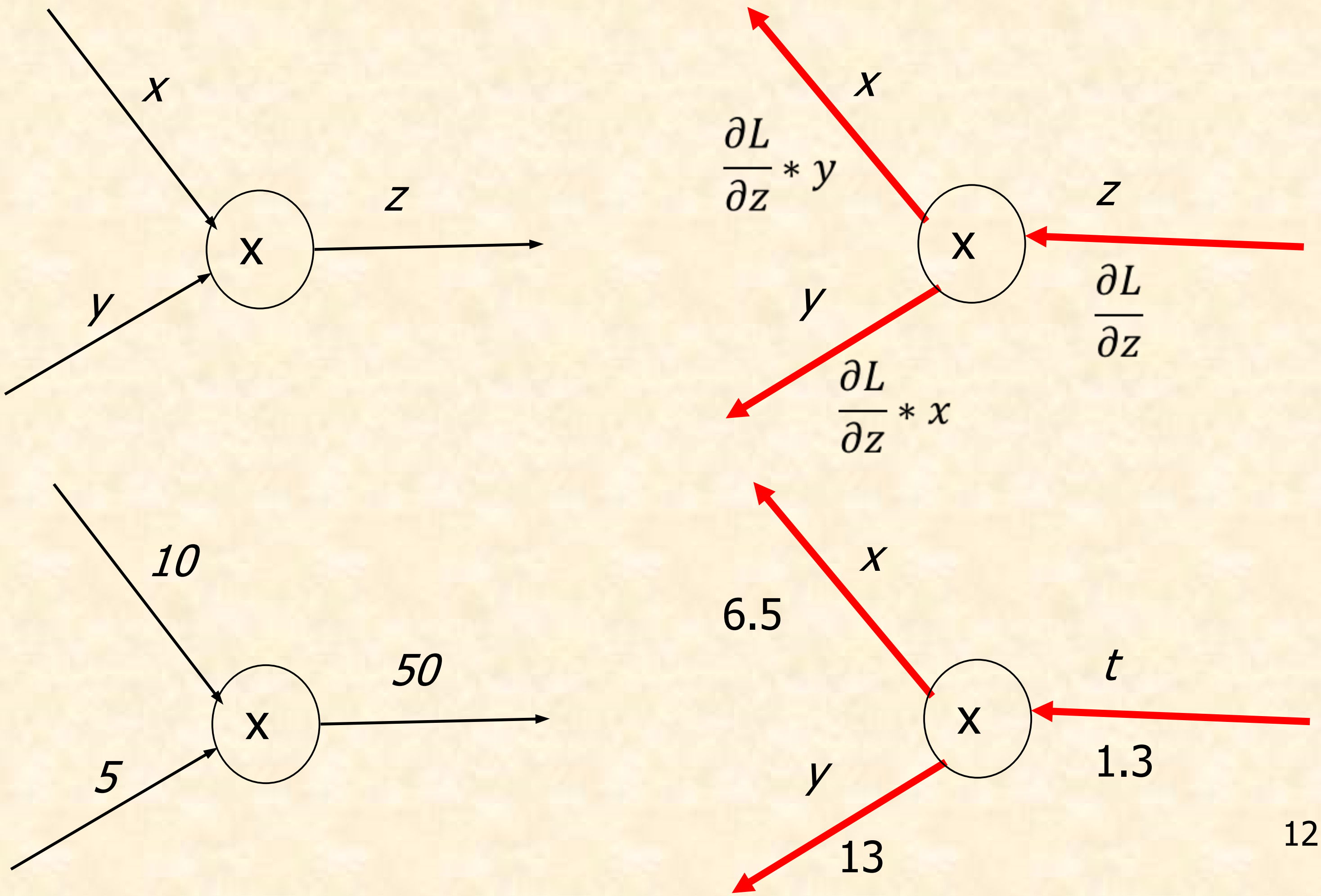
❖加法節點的反向傳播



反向傳播(Backpropagation)

$$\begin{cases} z = xy \\ t = x + y \end{cases}$$

❖ 乘法節點的反向傳播



執行活化函數層

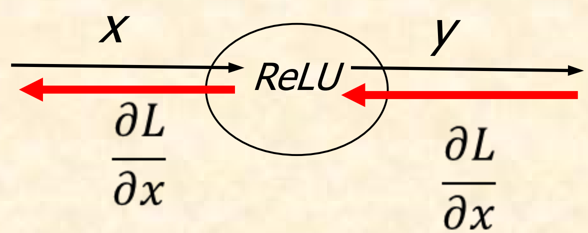
❖ ReLU層

$$y = \begin{cases} x, (x > 0) \\ 0, (x \le 0) \end{cases}$$

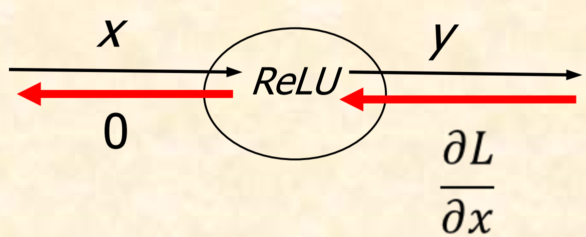
```
class Relu:
    def __init__(self):
        self.mask = None

    def forward(self, x):
        self.mask = (x <= 0)
        out = x.copy()
        out[self.mask] = 0

        return out
    def backward(self, dout):
        dout[self.mask] = 0
        dx = dout

        return dx
```

ReLU層的微分計算

$$\frac{\partial y}{\partial x} = \begin{cases} 1, (x > 0) \\ 0, (x \le 0) \end{cases}$$



(x>0)

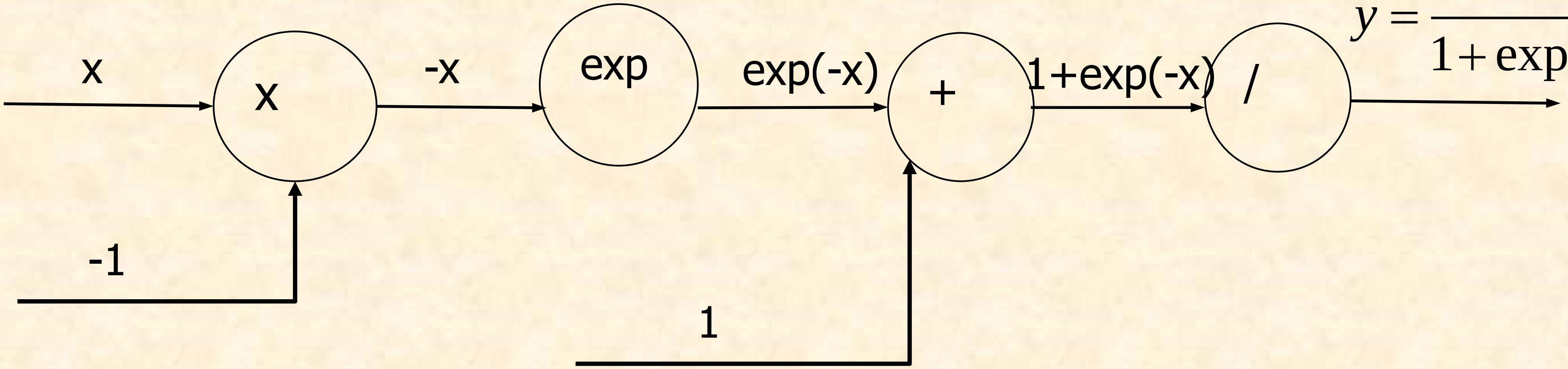


$(x \le 0)$

執行活化函數Sigmoid層

❖ Sigmoid函數

$$f(x) = \frac{1}{1+\exp(-x)}$$

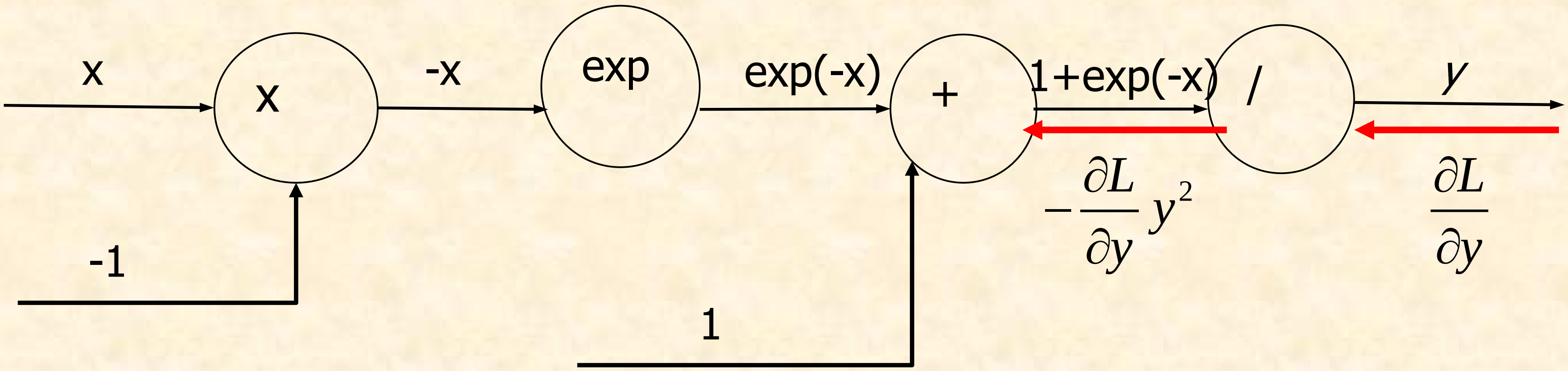


$$y = \frac{1}{1+\exp(-x)}$$

執行活化函數Sigmoid層

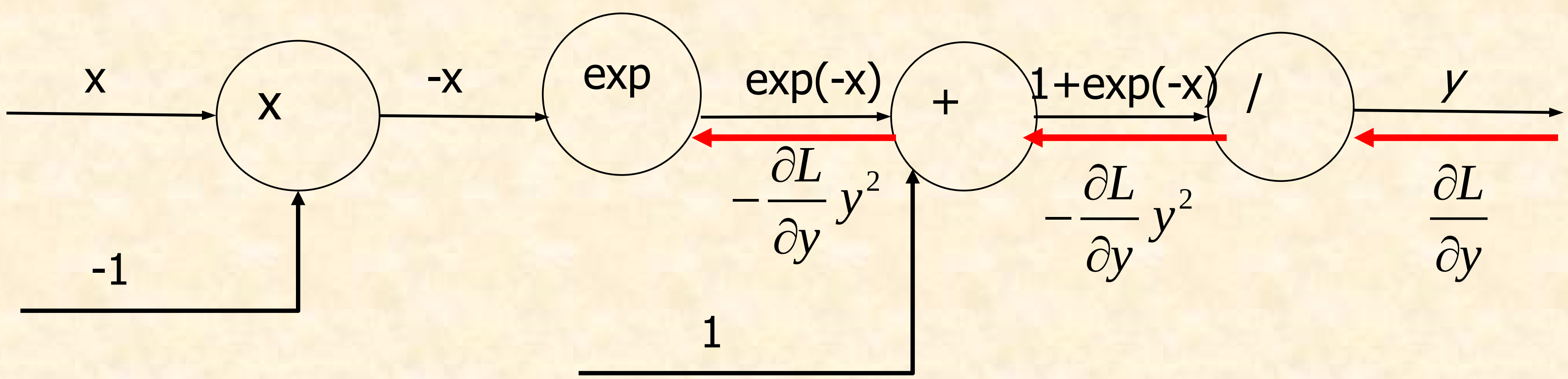
- 步驟一
  - 節點'/'計算y=1/x，其微分算式為

$$\frac{\partial y}{\partial x} = -\frac{1}{x^2} = -y^2$$



執行活化函數Sigmoid層

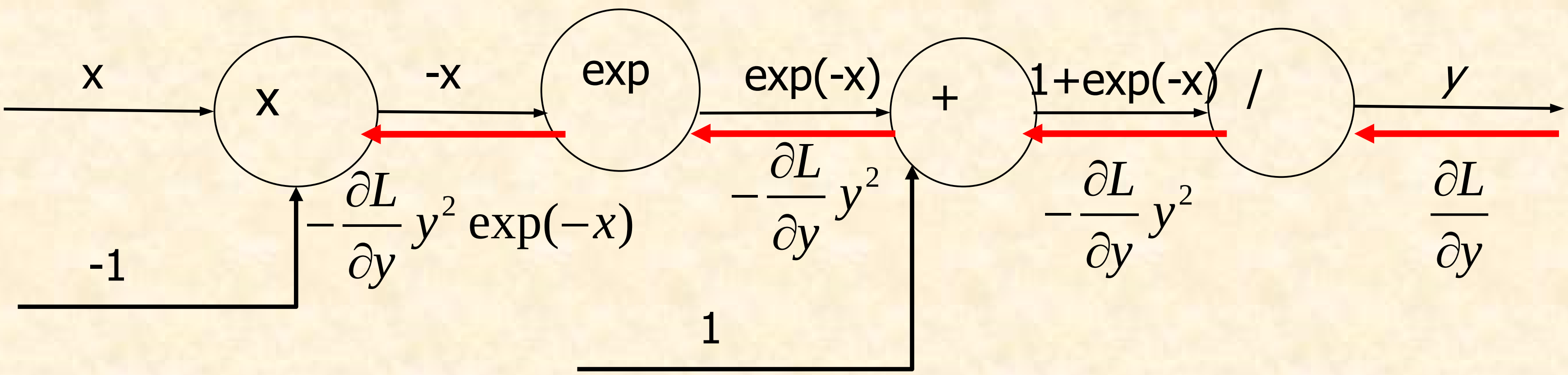
- 步驟二
  - 節點'+'直接將上層值傳給下層



執行活化函數Sigmoid層

- 步驟三
  - 節點'exp'計算y=exp(x)，其微分算式為

$$\frac{\partial y}{\partial x} = \exp(x)$$



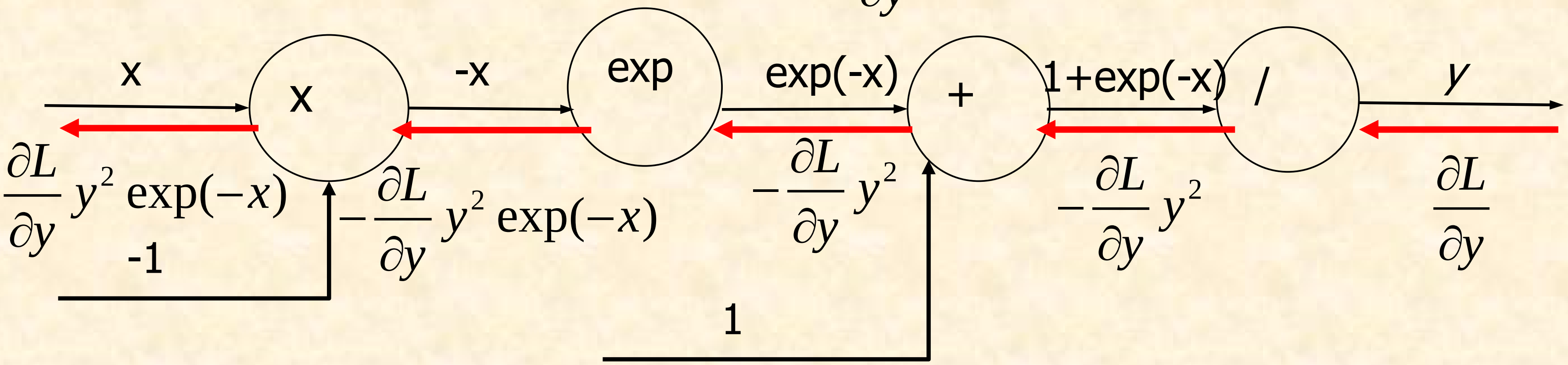
執行活化函數Sigmoid層

- 步驟四
  - 節點'x'是乘上正向傳播的相反值

$$\frac{\partial L}{\partial y} y^2 \exp(-x) = \frac{\partial L}{\partial y} \frac{1}{(1+\exp(-x))^2} \exp(-x)$$

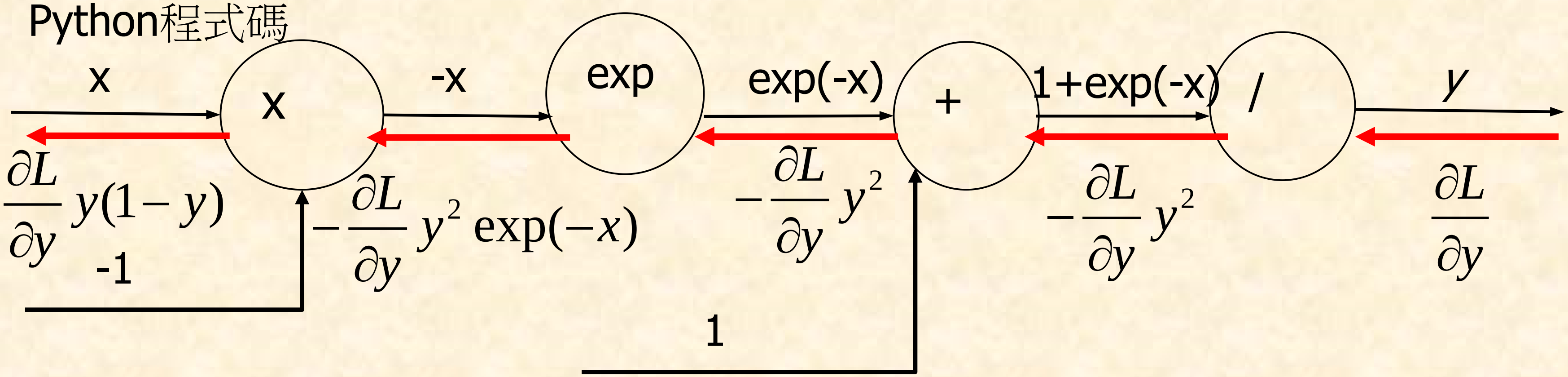
$$= \frac{\partial L}{\partial y} \frac{1}{1+\exp(-x)} \frac{\exp(-x)}{1+\exp(-x)}$$

$$= \frac{\partial L}{\partial y} y(1-y)$$



執行活化函數Sigmoid層

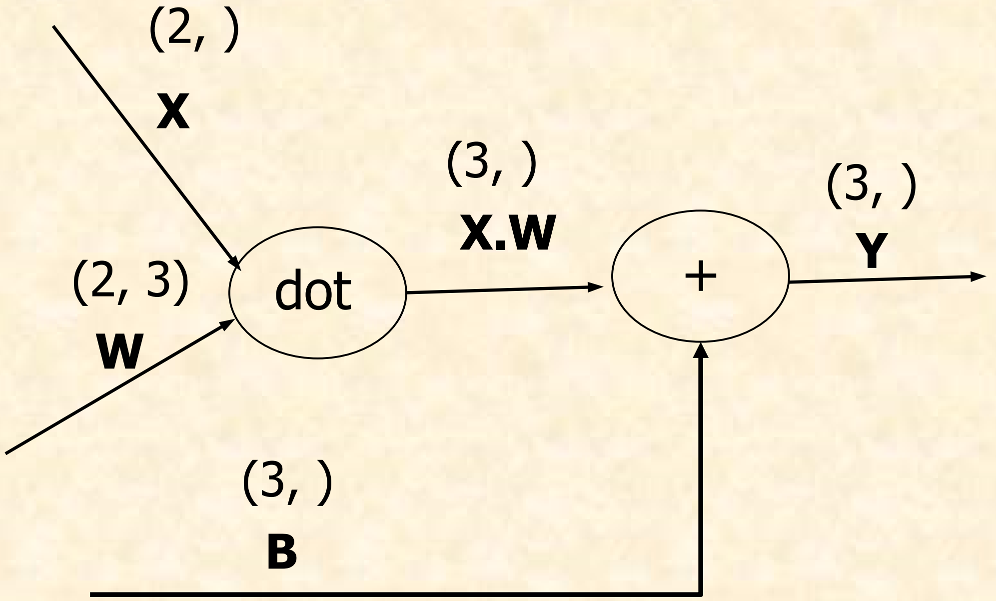
- Python程式碼




```
class Sigmoid:
    def __init__(self):
        self.out = None

    def forward(self, x):
        out = sigmoid(x)
        self.out = out
        return out

    def backward(self, dout):
        dx = dout * (1.0 - self.out) * self.out

        return dx
```

執行Affine/Softmax層

❖ Affine層



Y = np.dot(X, W) + B

執行Affine/Softmax層

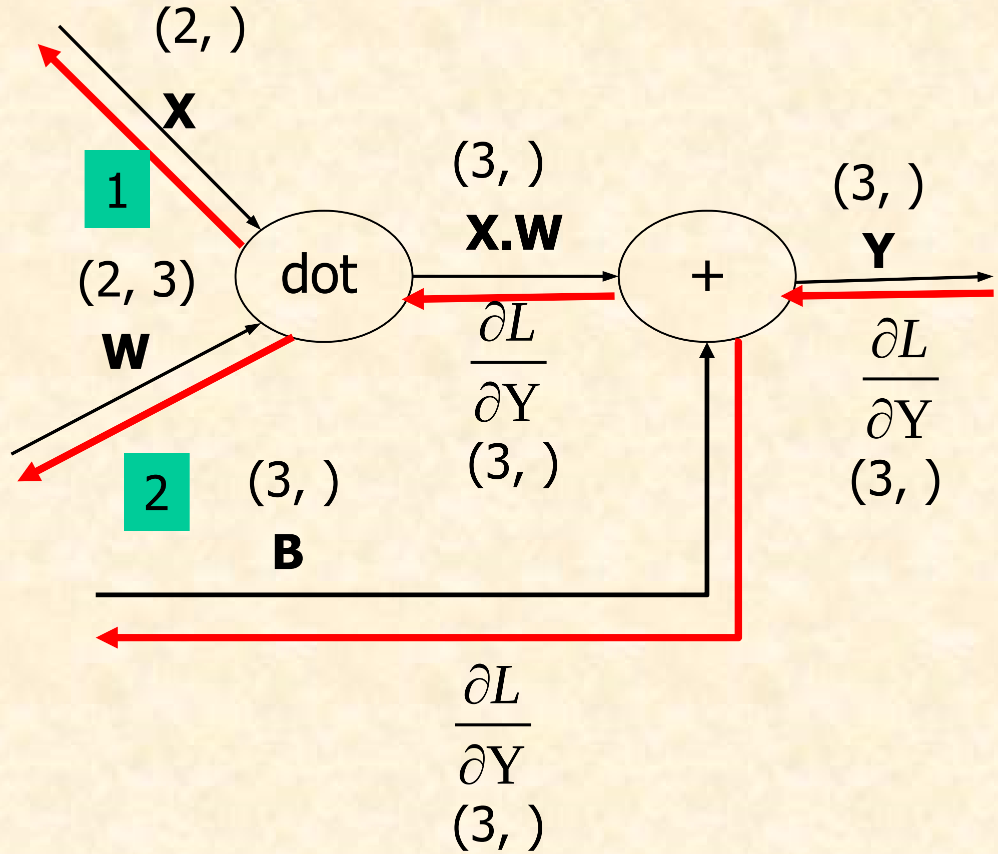
❖ Affine層

1 $$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot W^T$$

(2,) (3,) (3, 2)

2 $$\frac{\partial L}{\partial W} = X^T \cdot \frac{\partial L}{\partial Y}$$

(2,3) (2, 1) (1, 3)



執行Affine/Softmax層

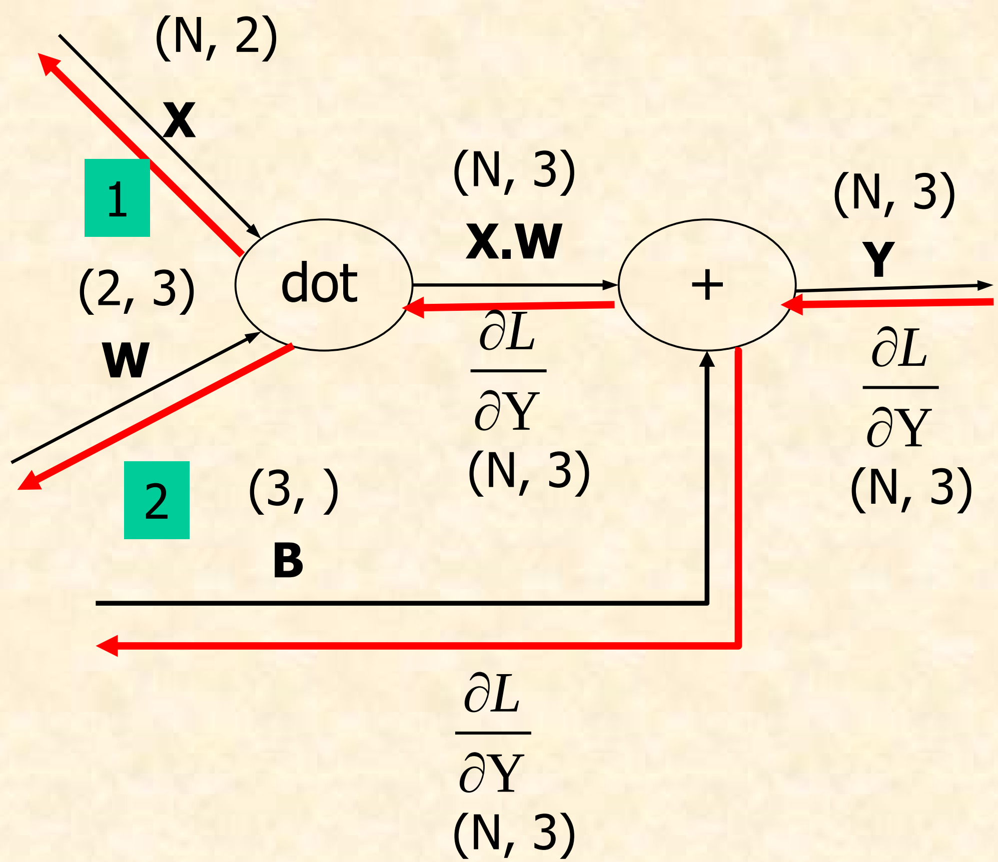
❖ 批次版Affine層

1 $$\frac{\partial L}{\partial X} = \frac{\partial L}{\partial Y} \cdot W^T$$

(N, 2) (N,3) (3, 2)

2 $$\frac{\partial L}{\partial W} = X^T \cdot \frac{\partial L}{\partial Y}$$

(2,3) (2, 1) (1, 3)



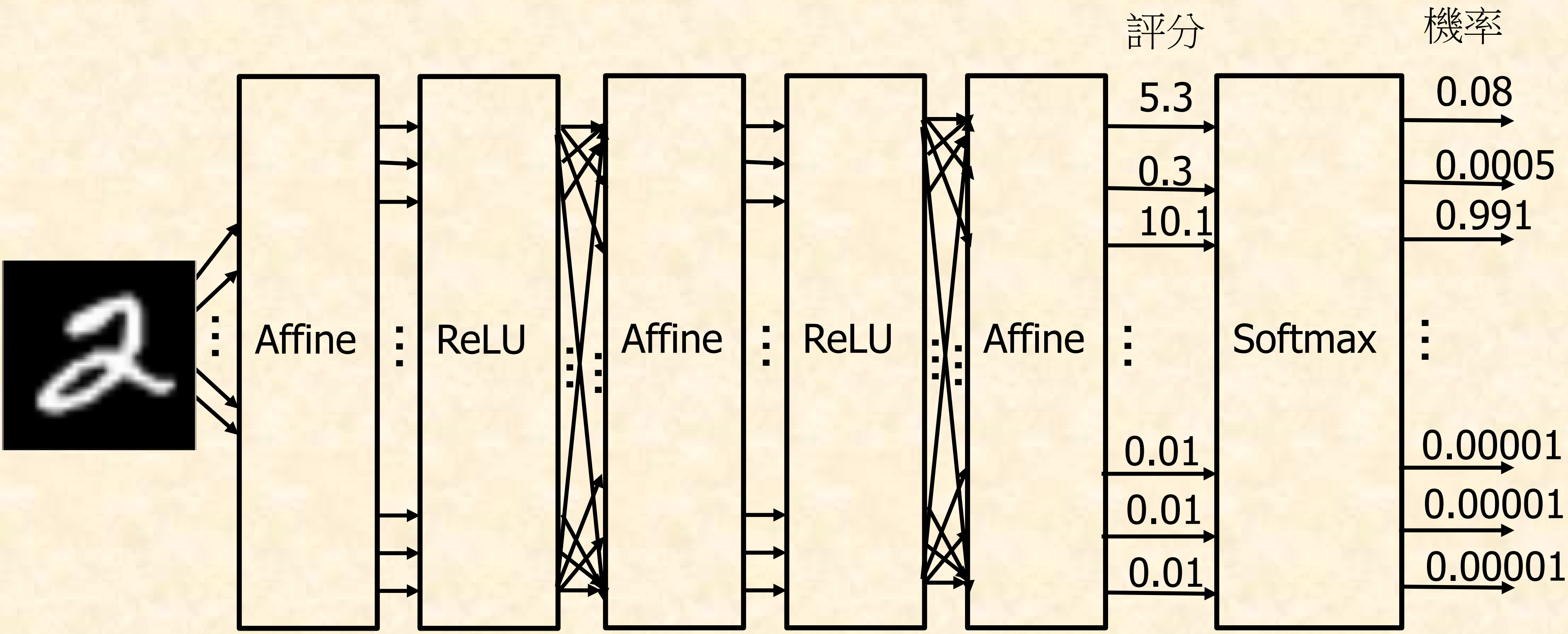
執行Affine/Softmax層

❖ Python程式碼

```
class Affine:
    def __init__(self, W, b):
        self.W =W
        self.b = b
        self.x = None
        self.original_x_shape = None
        self.dW = None
        self.db = None

    def forward(self, x):
        self.original_x_shape = x.shape
        x = x.reshape(x.shape[0], -1)
        self.x = x
        out = np.dot(self.x, self.W) + self.b

        return out

    def backward(self, dout):
        dx = np.dot(dout, self.W.T)
        self.dW = np.dot(self.x.T, dout)
        self.db = np.sum(dout, axis=0)

        dx =.dx.reshape(*self.original_x_shape.
        return dx
```

Softmax-with-Loss層

❖ Softmax函數通常落在最後一層，用以正規化輸出值

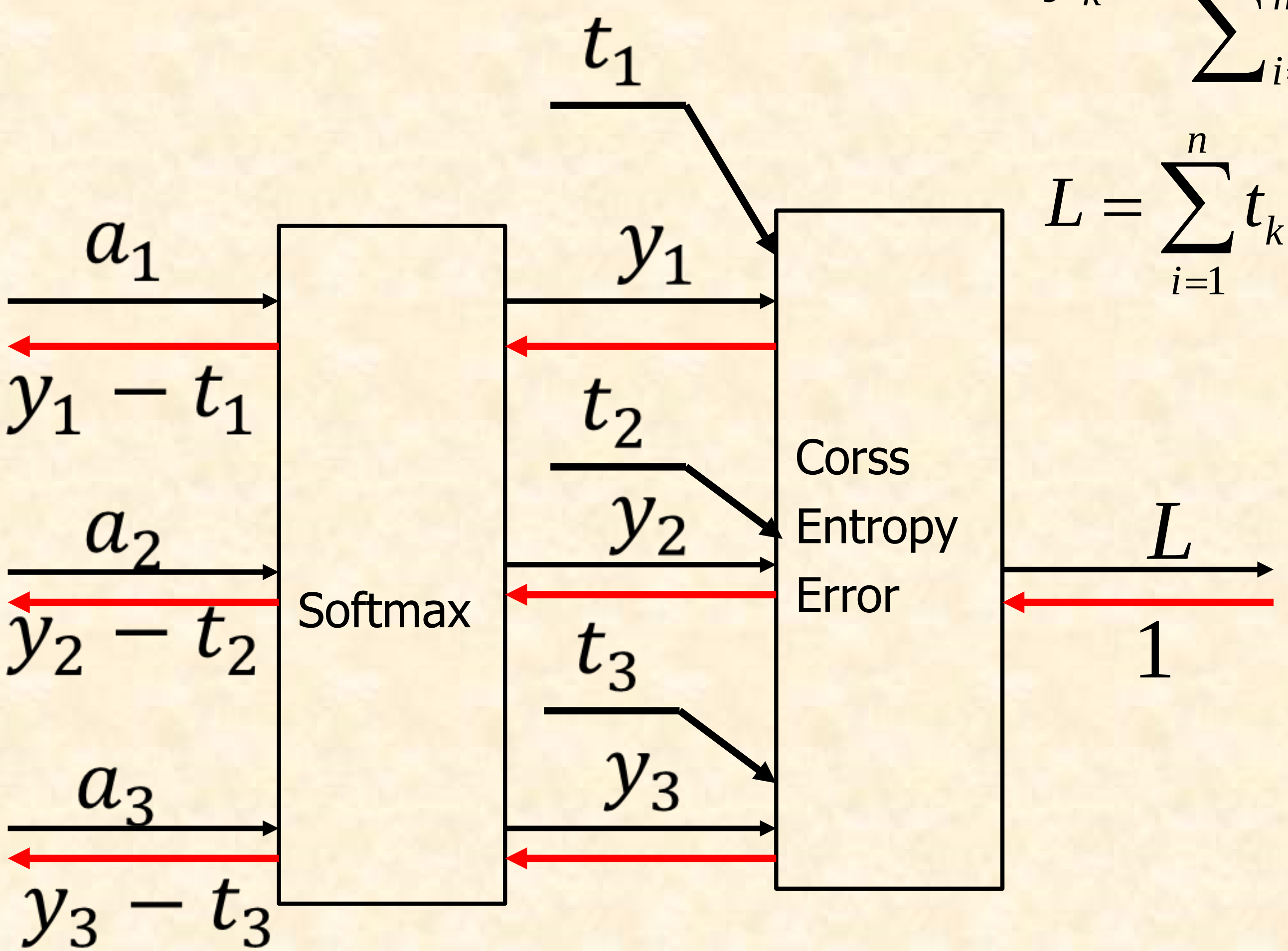


Softmax-with-Loss層

- Softmax-with-Loss層的計算圖層

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$

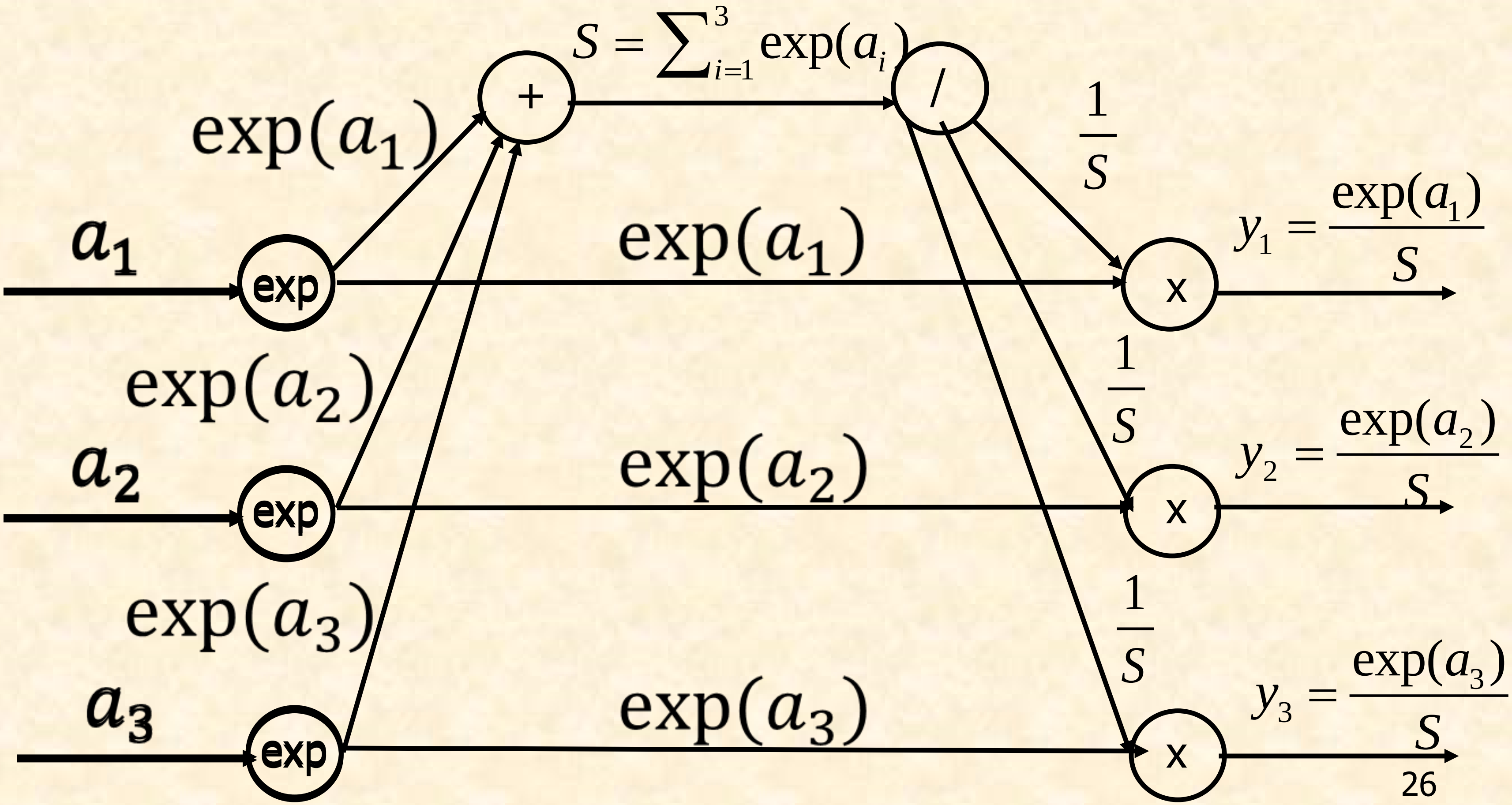
$$L = \sum_{i=1}^{n} t_k \log y_k$$



Softmax-with-Loss層

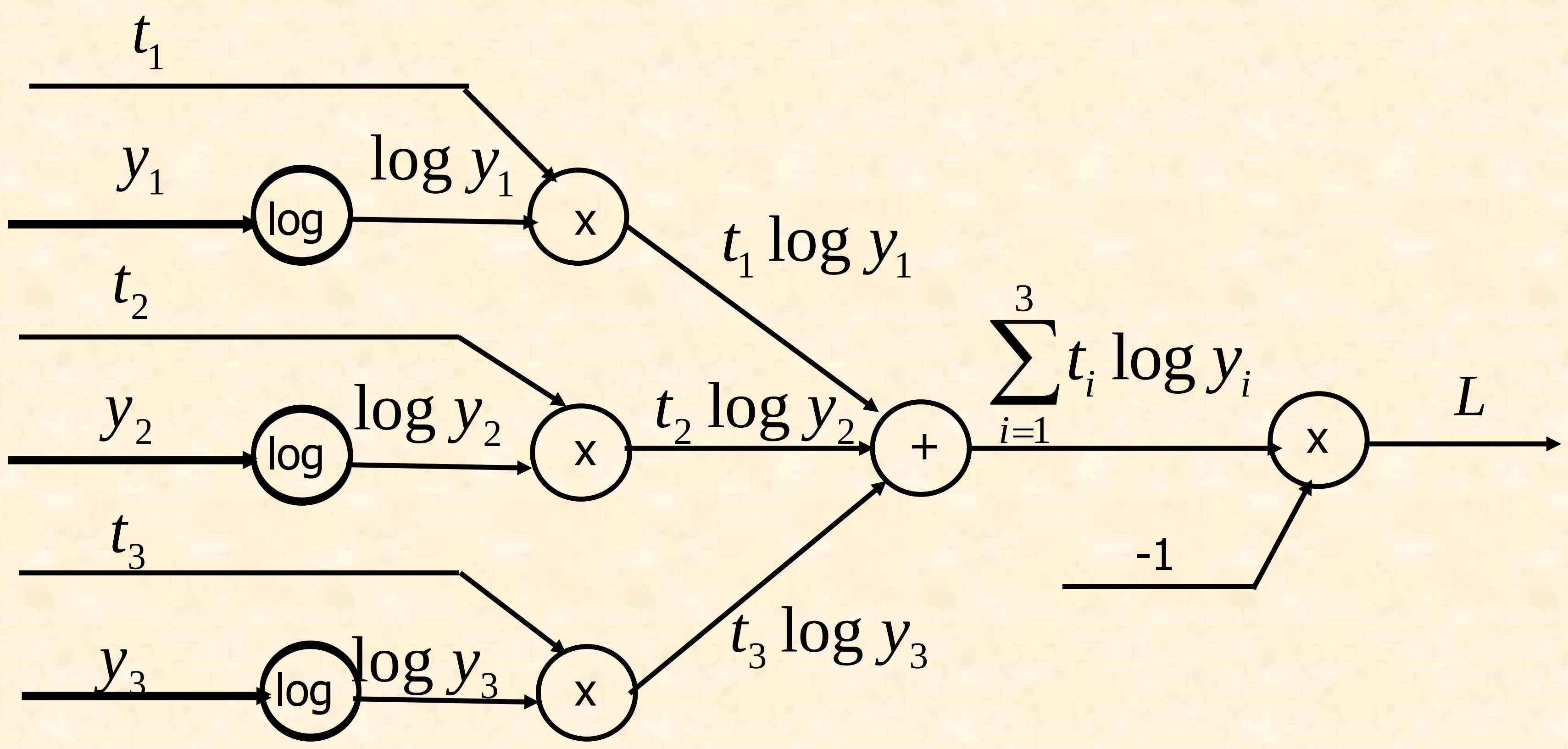
- Softmax層的正向傳播

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$



Softmax-with-Loss層

❖ Cross Entropy Error層的正向傳播



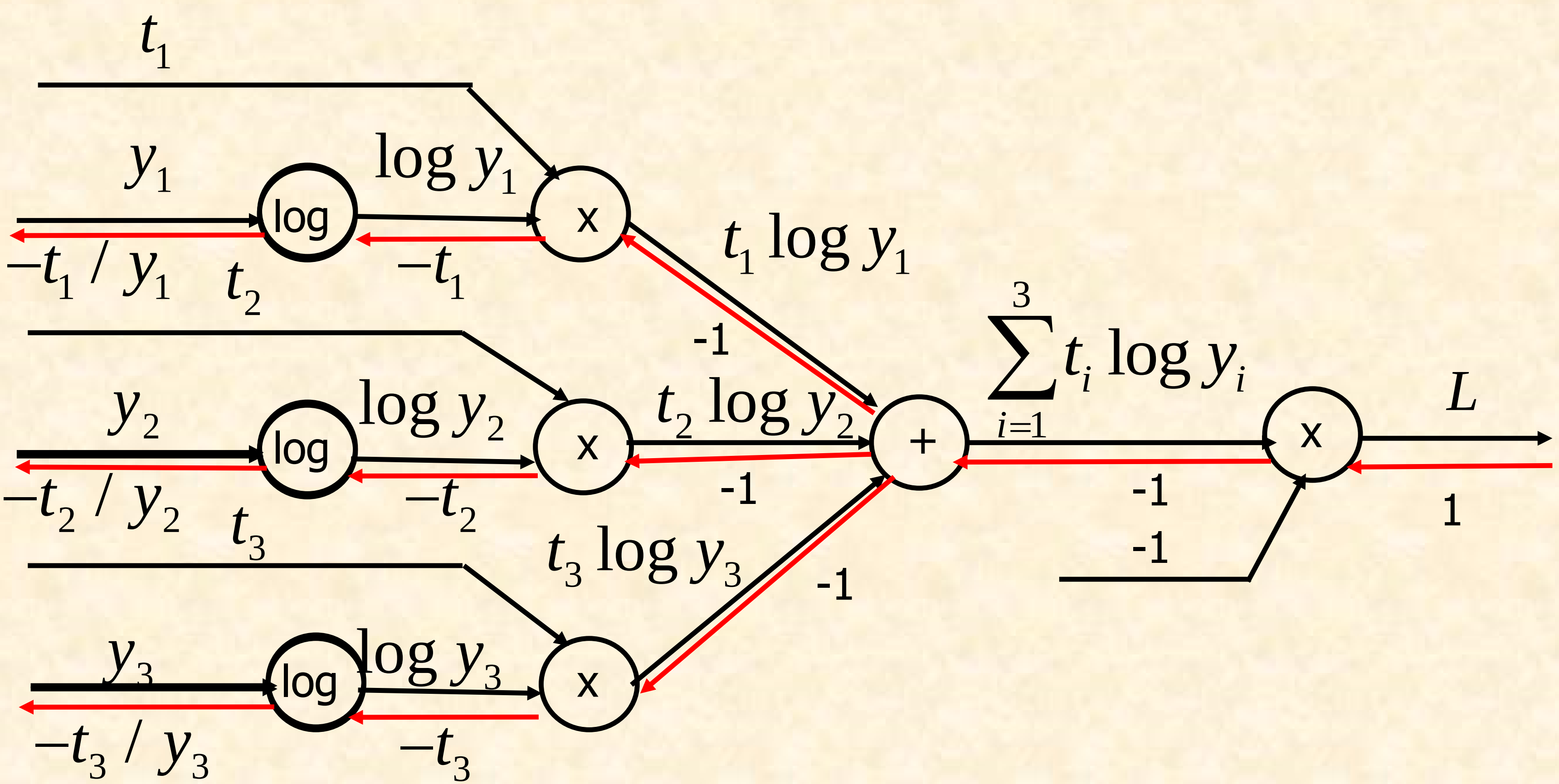
Softmax-with-Loss層

❖ 執行Softmax-with-Loss層的正向傳播

```
class SoftmaxWithLoss:
    def __init__(self):
        self.loss = None
        self.y = None # softmax的輸出
        self.t = None # 訓練資料(one-hot vector)

    def forward(self, x, t):
        self.t = t
        self.y = softmax(x)
        self.loss = cross_entropy_error(self.y, self.t)

        return self.loss

    def backward(self, dout=1):
        batch_size = self.t.shape[0]
        if self.t.size == self.y.size:
        dx = (self.y - self.t) / batch_size
        else:
            dx = self.y.copy()
            dx[np.arange(batch_size), self.t] -= 1
            dx = dx / batch_size

        return dx
```

Softmax-with-Loss層

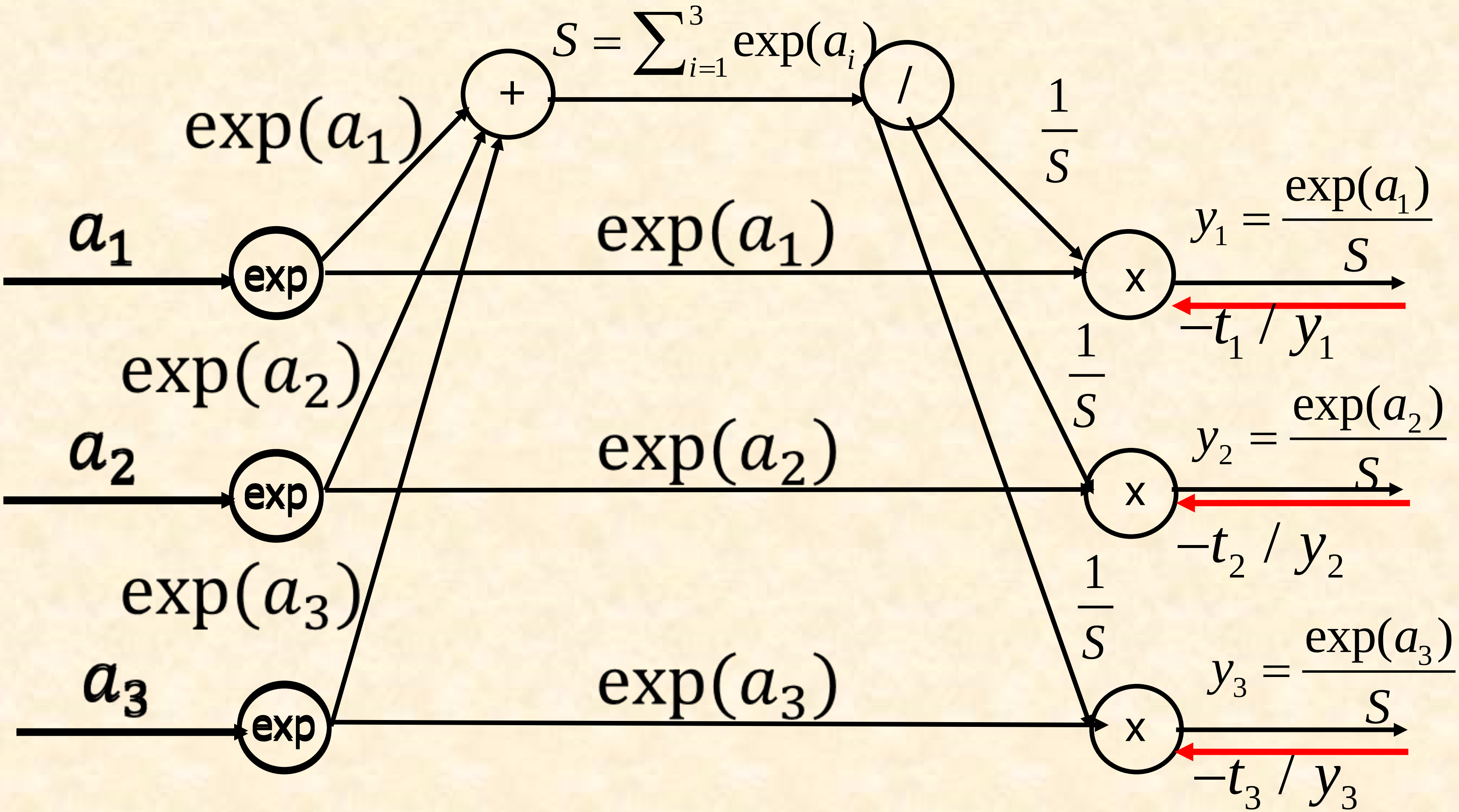
❖ Cross Entropy Error層的反向傳播



Softmax-with-Loss層

- Softmax層的反向傳播:步驟一

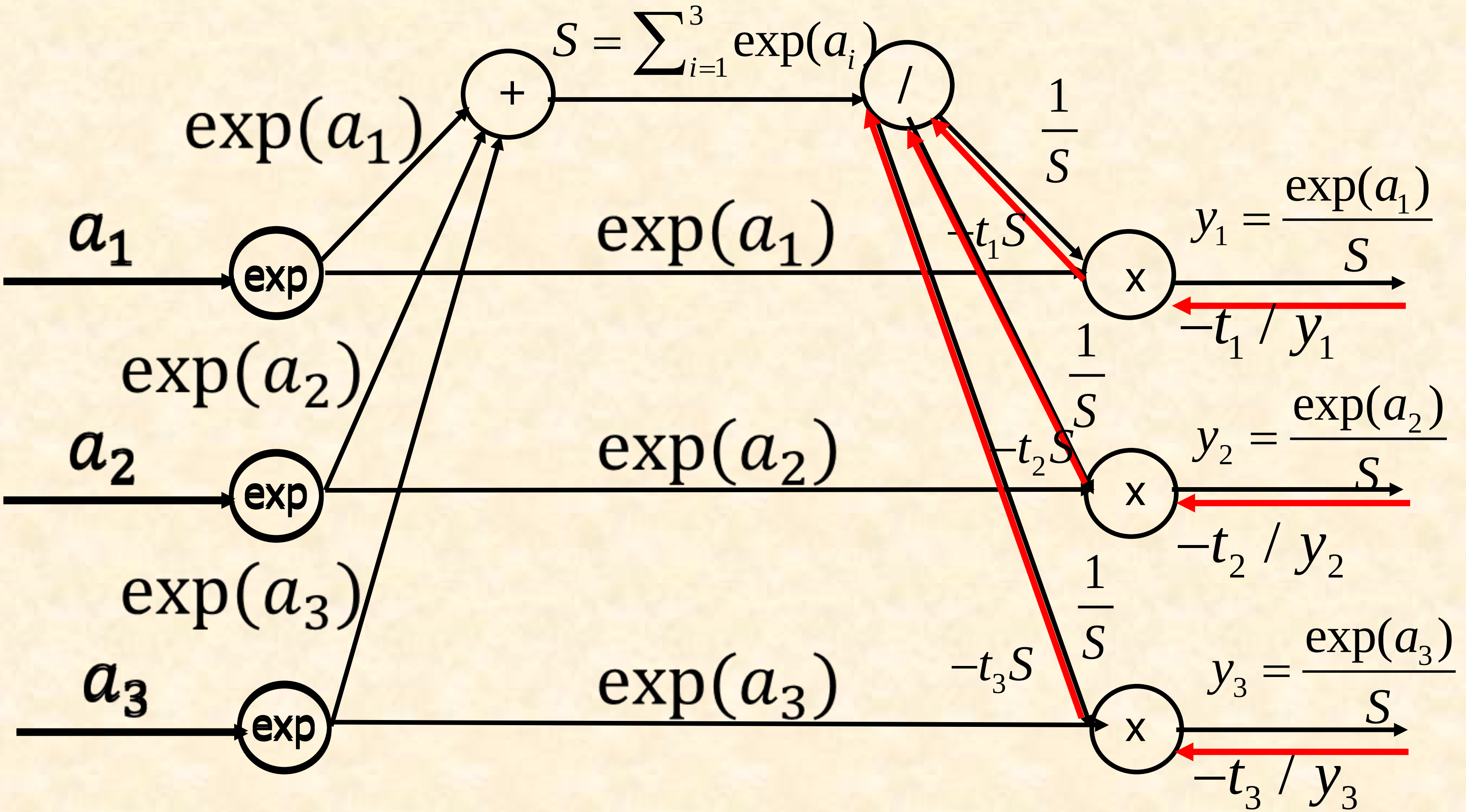
$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n}\exp(a_i)}$$



Softmax-with-Loss層

- Softmax層的反向傳播:步驟二

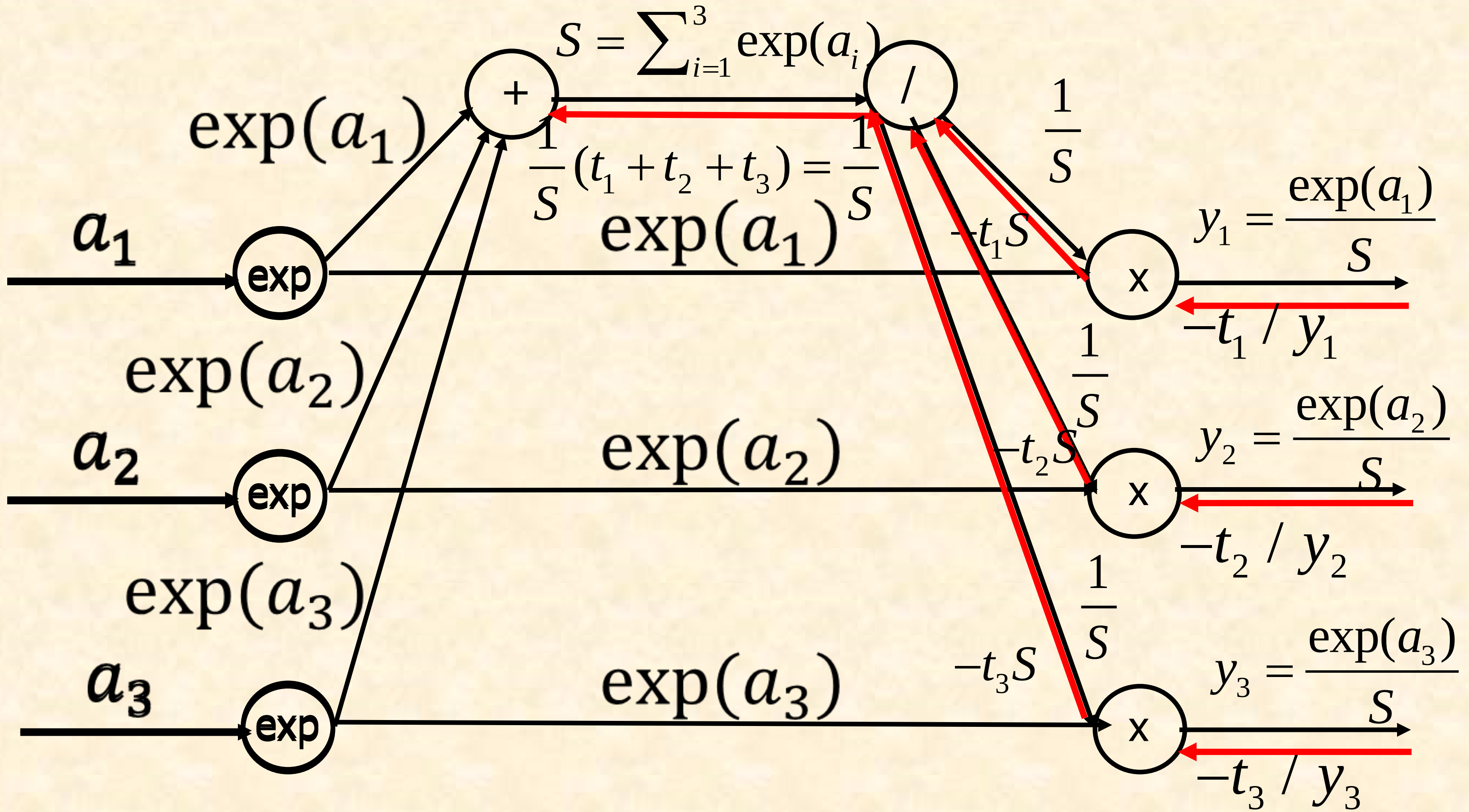
$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n} \exp(a_i)}$$



Softmax-with-Loss層

- Softmax層的反向傳播:步驟三

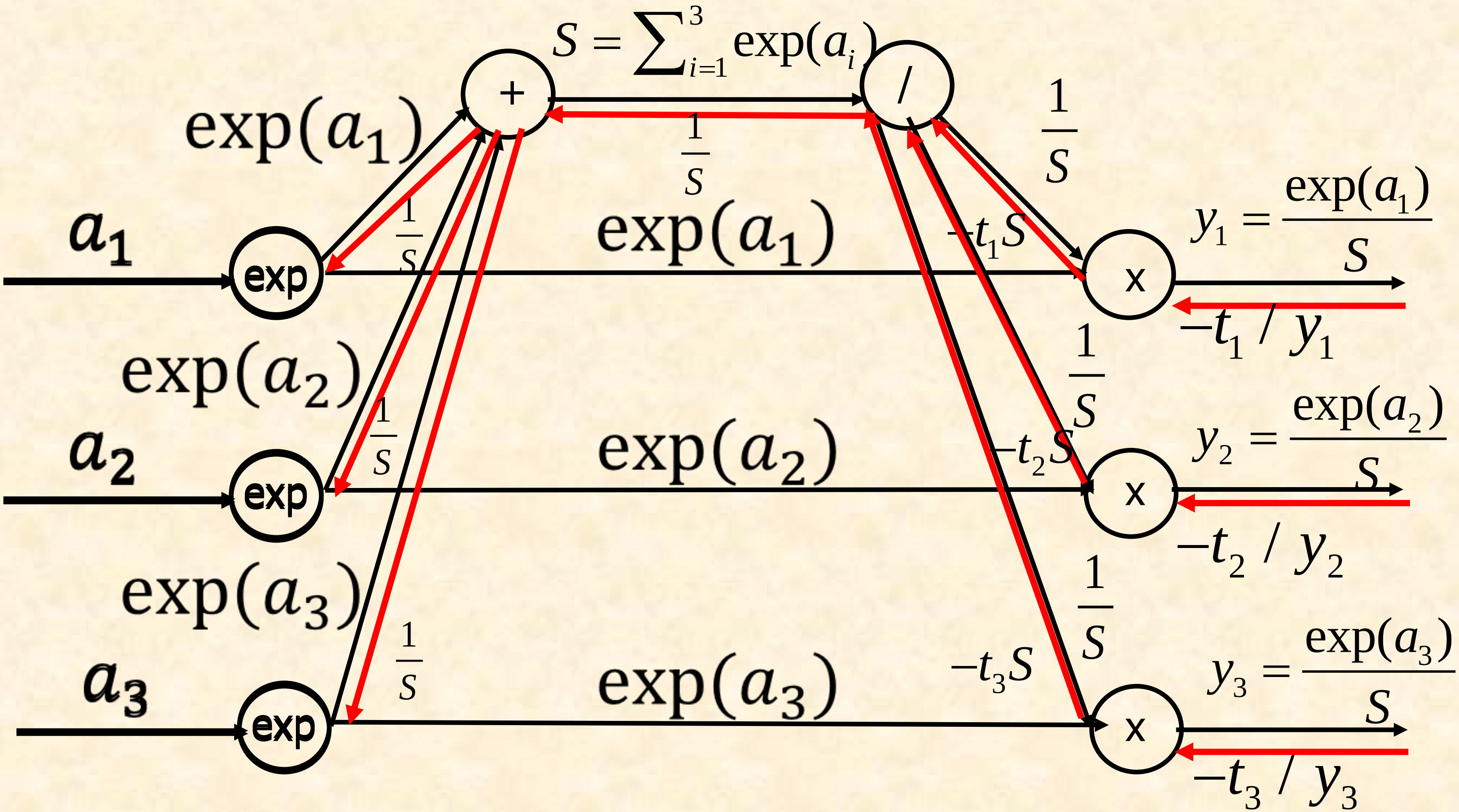
$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n}\exp(a_i)}$$



Softmax-with-Loss層

❖ Softmax層的反向傳播:步驟4

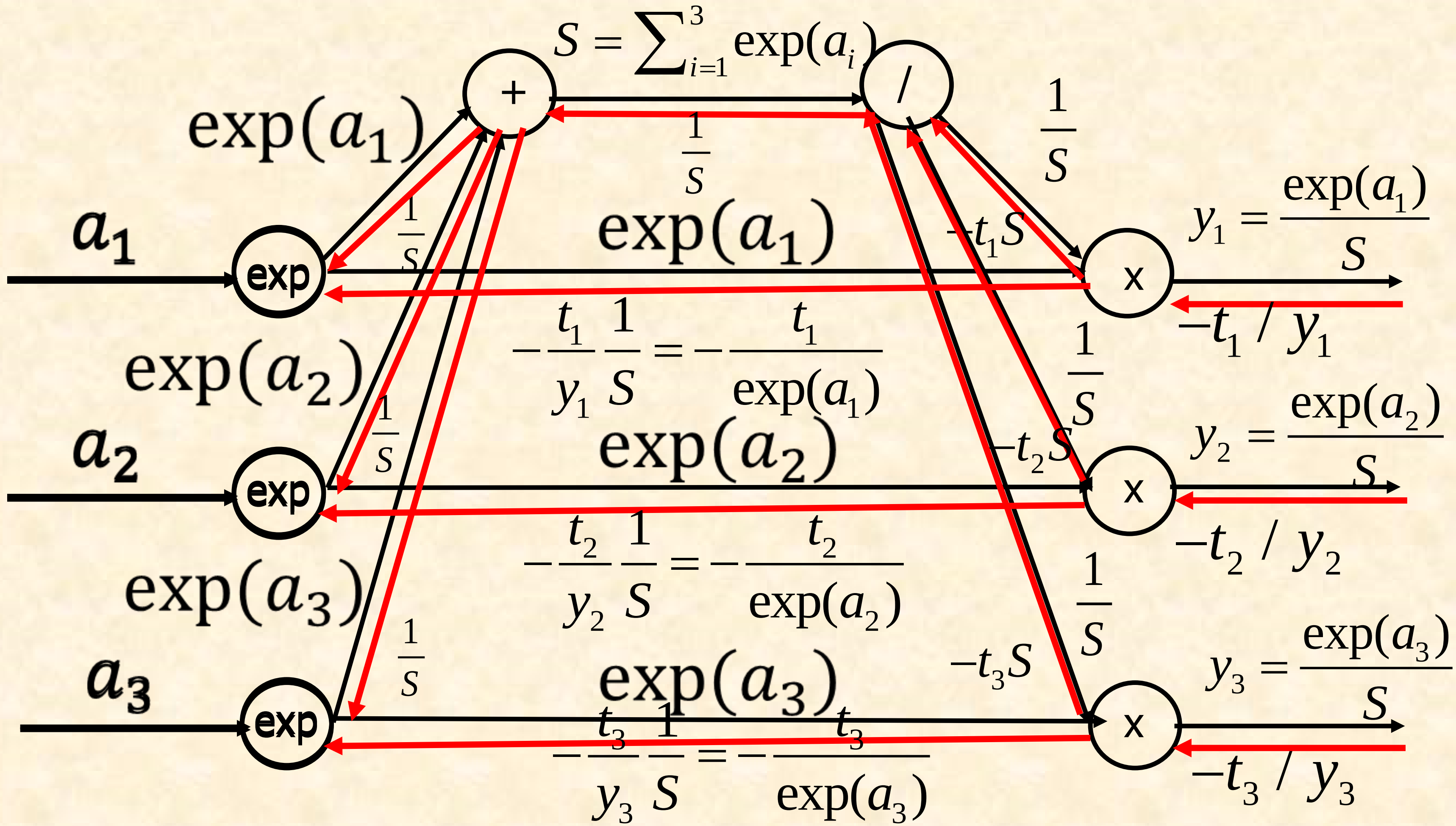
$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n}\exp(a_i)}$$



Softmax-with-Loss層

❖ Softmax層的反向傳播:步驟5

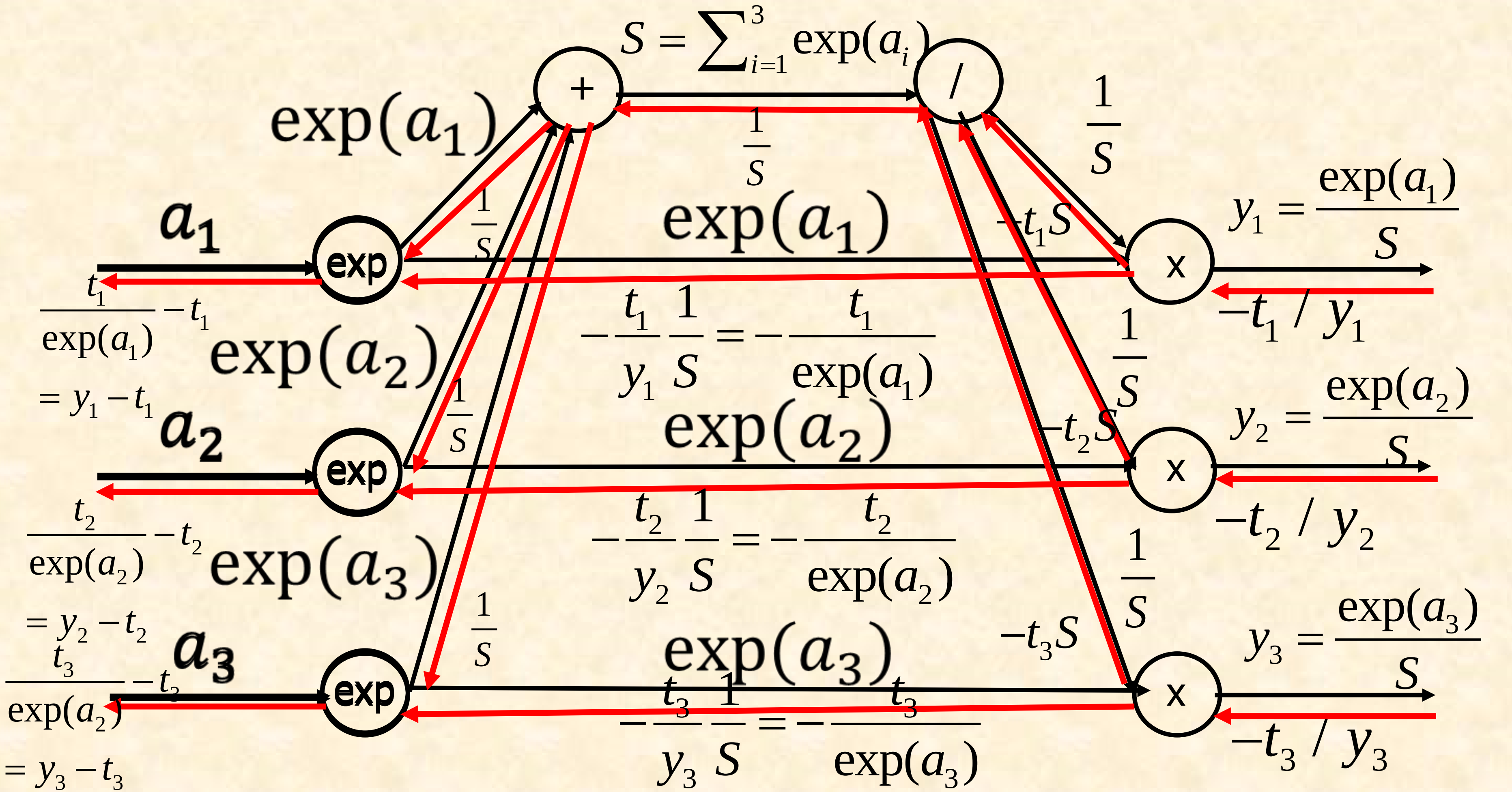
$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n}\exp(a_i)}$$



Softmax-with-Loss層

- Softmax層的反向傳播:步驟6

$$y_k = \frac{\exp(a_k)}{\sum_{i=1}^{n}\exp(a_i)}$$



執行誤差反向傳播法

- 步驟1(小批次)
  - 從訓練資料中，隨機取出部分資料。
- 步驟2(計算梯度)
  - 計算與各權重參數有關的損失函數梯度。
- 步驟3(更新參數)
  - 往梯度方向微量更新權重參數。
- 步驟4(重複)
  - 重複步驟1、步驟2、步驟3。

執行對應誤差反向學習:以雙層神經網路為例

- 請參考如下Python程式碼
  - TwoLayerNet.py
  - Grdient_check.py
  - Train_neuralnet.py

Any Questions?