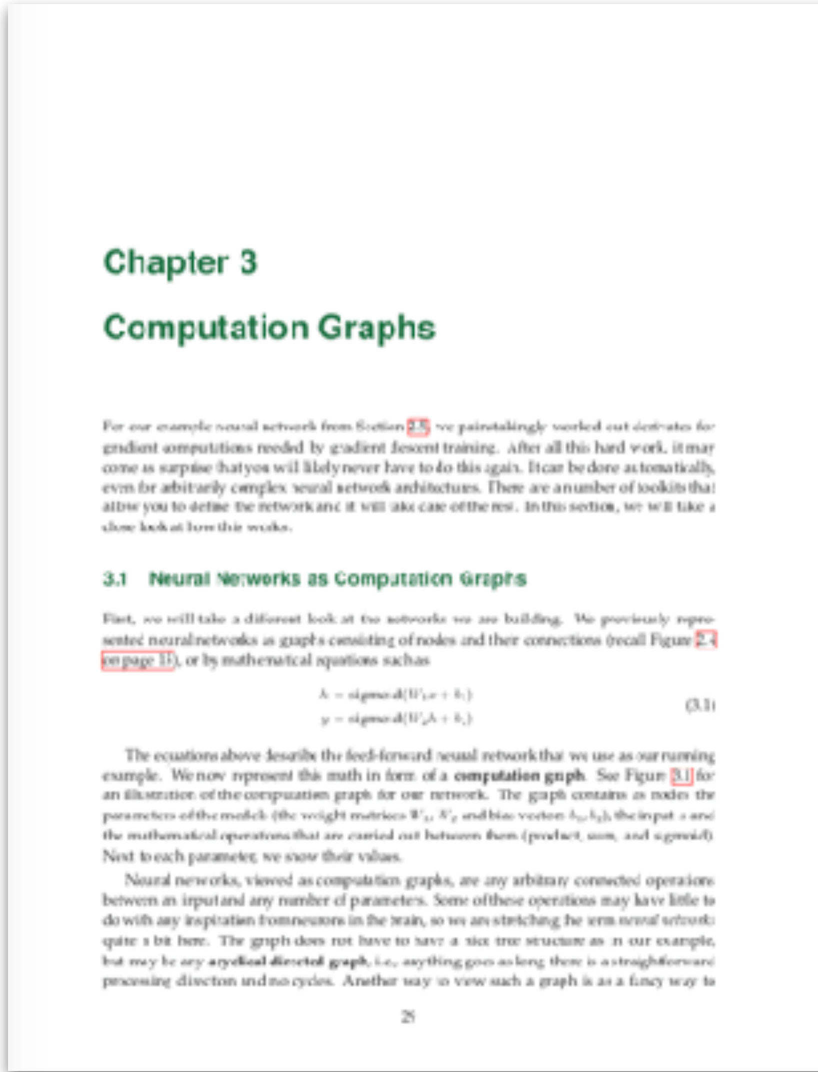
NMT 第 3 章

計算圖 Computation Graph

教科書與課程網站：mt-class.org/jhu/syllabus.html （草稿）

2018 0930

教科書相關章節



Chapter 3

Computation Graphs

For our example neural network from Section 2.5, we painstakingly worked out derivates for gradient computations needed by gradient descent training. After all this hard work, it may come as surprise that you will likely never have to do this again. It can be done automatically, even for arbitrarily complex neural network architectures. There are a number of toolkits that allow you to define the network and it will take care of the rest. In this section, we will take a close look at how this works.

3.1 Neural Networks as Computation Graphs

First, we will take a different look at the networks we are building. We previously represented neural networks as graphs consisting of nodes and their connections (recall Figure 2.4 on page 15), or by mathematical equations such as

$$
\begin{aligned}
h &= \text{sigmoid}(W_1 x + b_1) \\
y &= \text{sigmoid}(W_2 h + b_2)
\end{aligned}
\tag{3.1}
$$

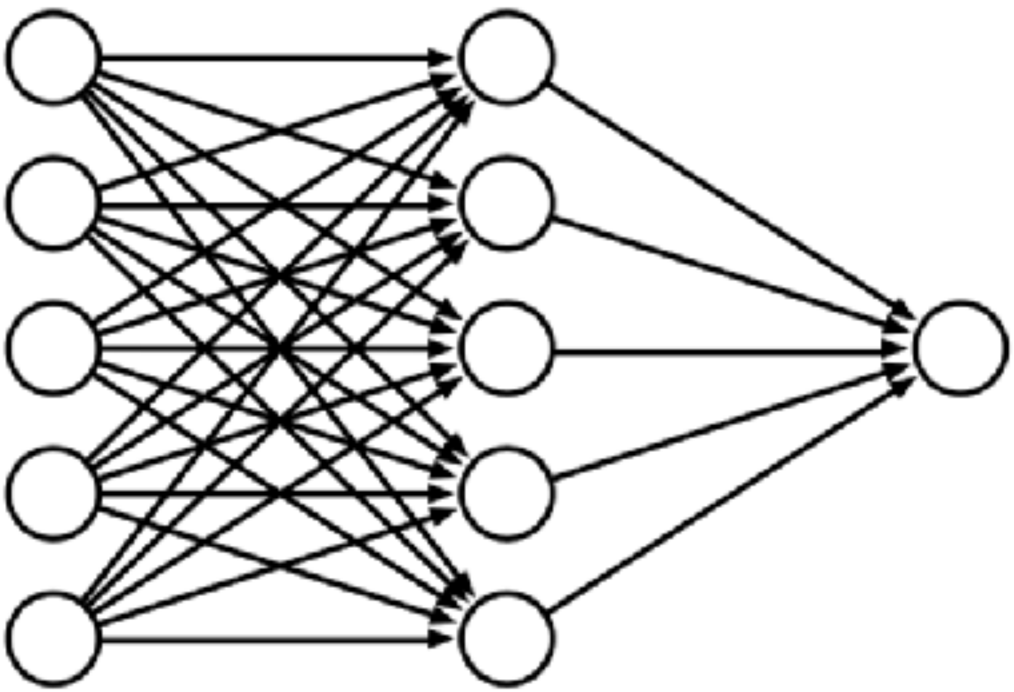
The equations above describe the feed-forward neural network that we use as our running example. We now represent this math in form of a **computation graph**. See Figure 3.1 for an illustration of the computation graph for our network. The graph contains as nodes the parameters of the models (the weight matrices W_1, W_2 and bias vectors b_1, b_2), the input x and the mathematical operations that are carried out between them (product, sum, and sigmoid). Next to each parameter, we show their values.

Neural networks, viewed as computation graphs, are any arbitrary connected operations between an input and any number of parameters. Some of these operations may have little to do with any inspiration from neurons in the brain, so we are stretching the term *neural networks* quite a bit here. The graph does not have to have a nice tree structure as in our example, but may be any **acyclical directed graph**, i.e., anything goes as long there is a straightforward processing direction and no cycles. Another way to view such a graph is as a fancy way to

來源：mt-class.org/jhu/assets/nmt-book.pdf (Draft of September 25, 2017)

通常以點及邊所構成的圖來顯示類神經網路

- 過去的線性模型用特徵質的線性組合



類神經網路的數學式

- 隱藏層

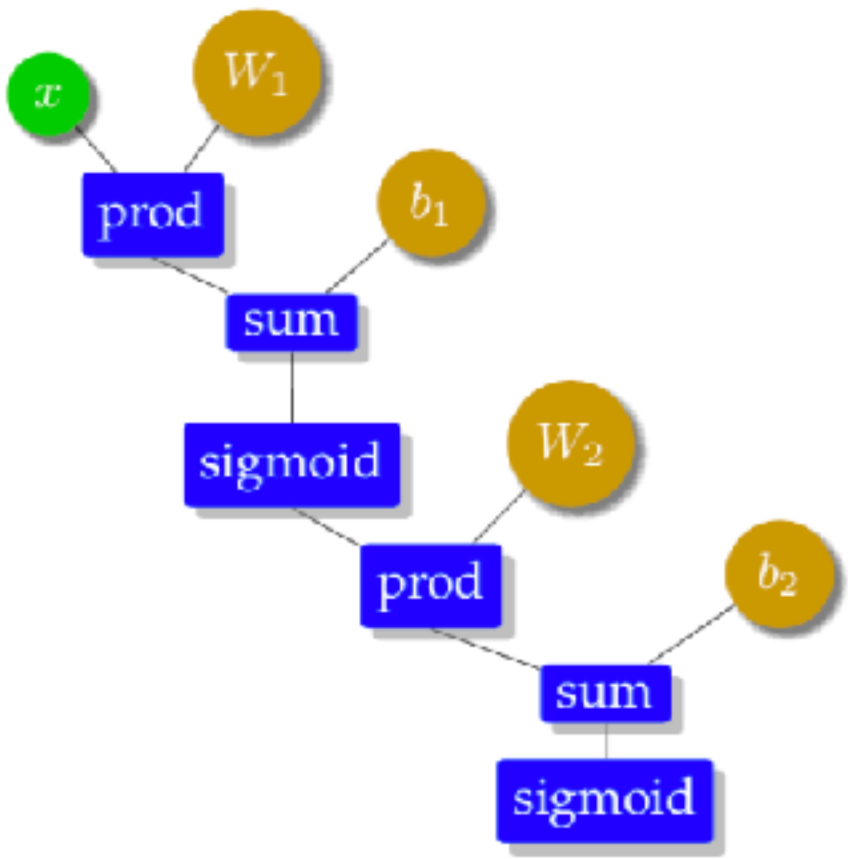
$$h = \mathrm{sigmoid}(W_1 x + b_1)$$

- 最後輸出層

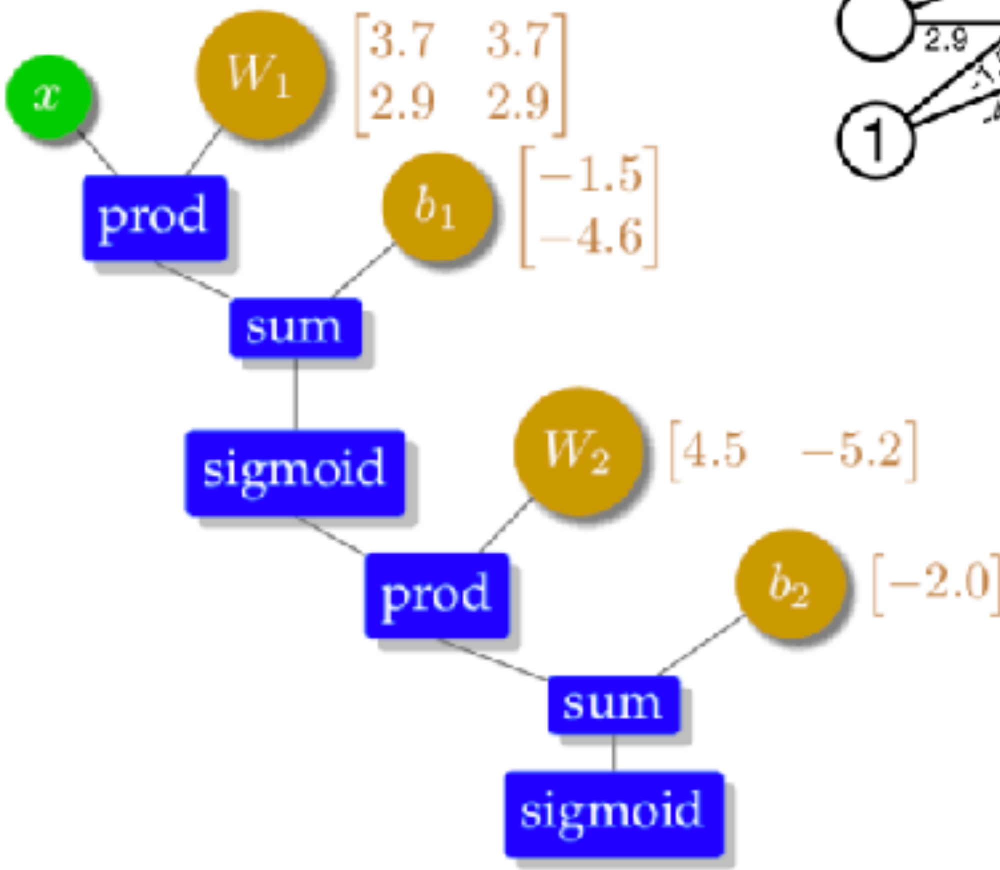
$$y = \mathrm{sigmoid}(W_2 h + b_2)$$

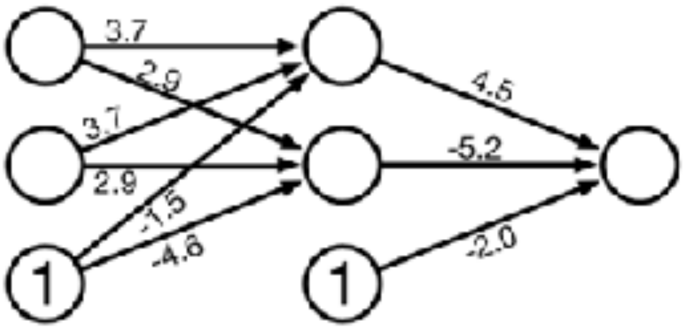
表達類神經網路為計算圖

- 類神經網路轉化為向量、矩陣的計算圖

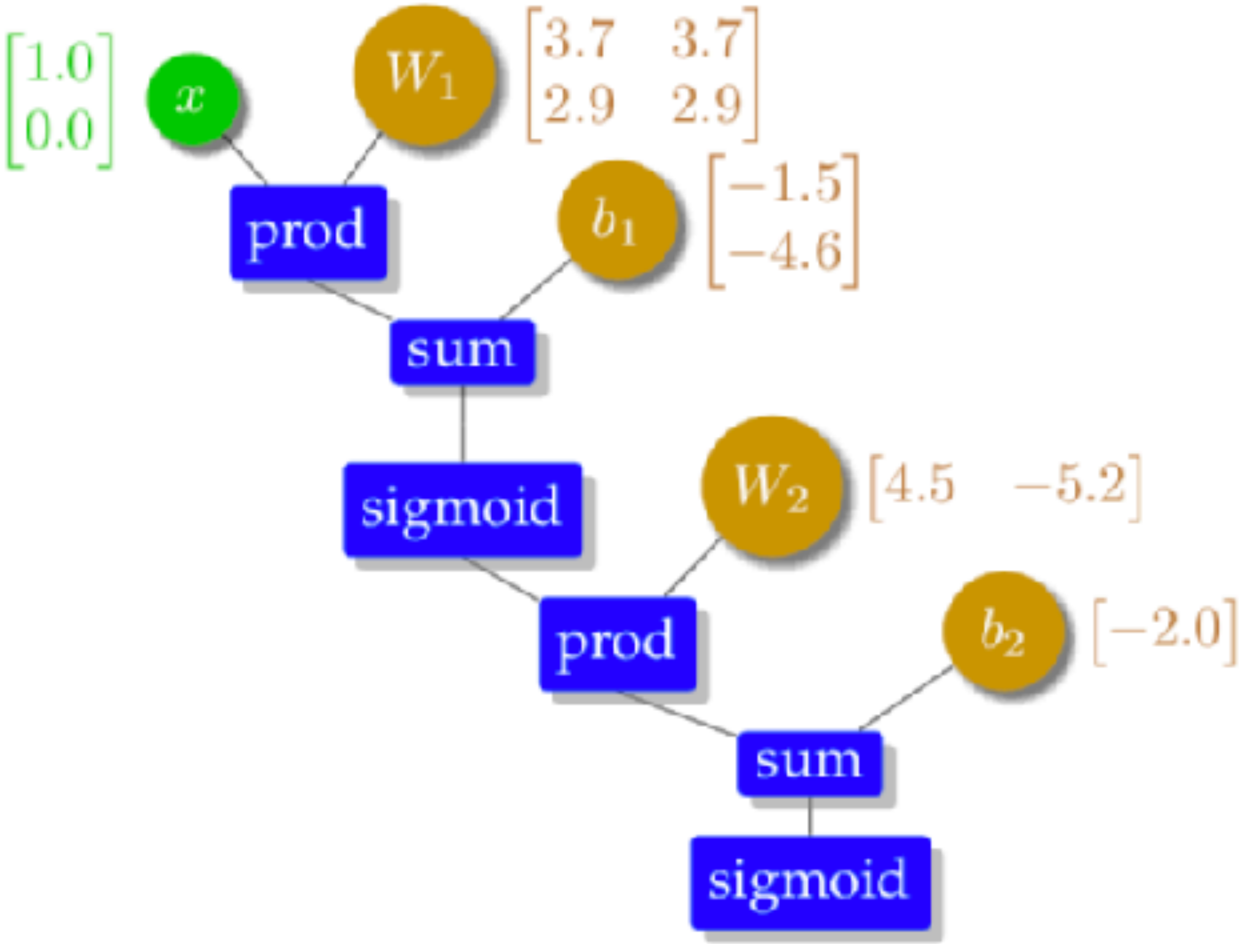


對應的計算圖


x
W1
[3.7 3.7; 2.9 2.9]
prod
b1
[−1.5; −4.6]
sum
sigmoid
W2
[4.5 −5.2]
prod
b2
[−2.0]
sum
sigmoid

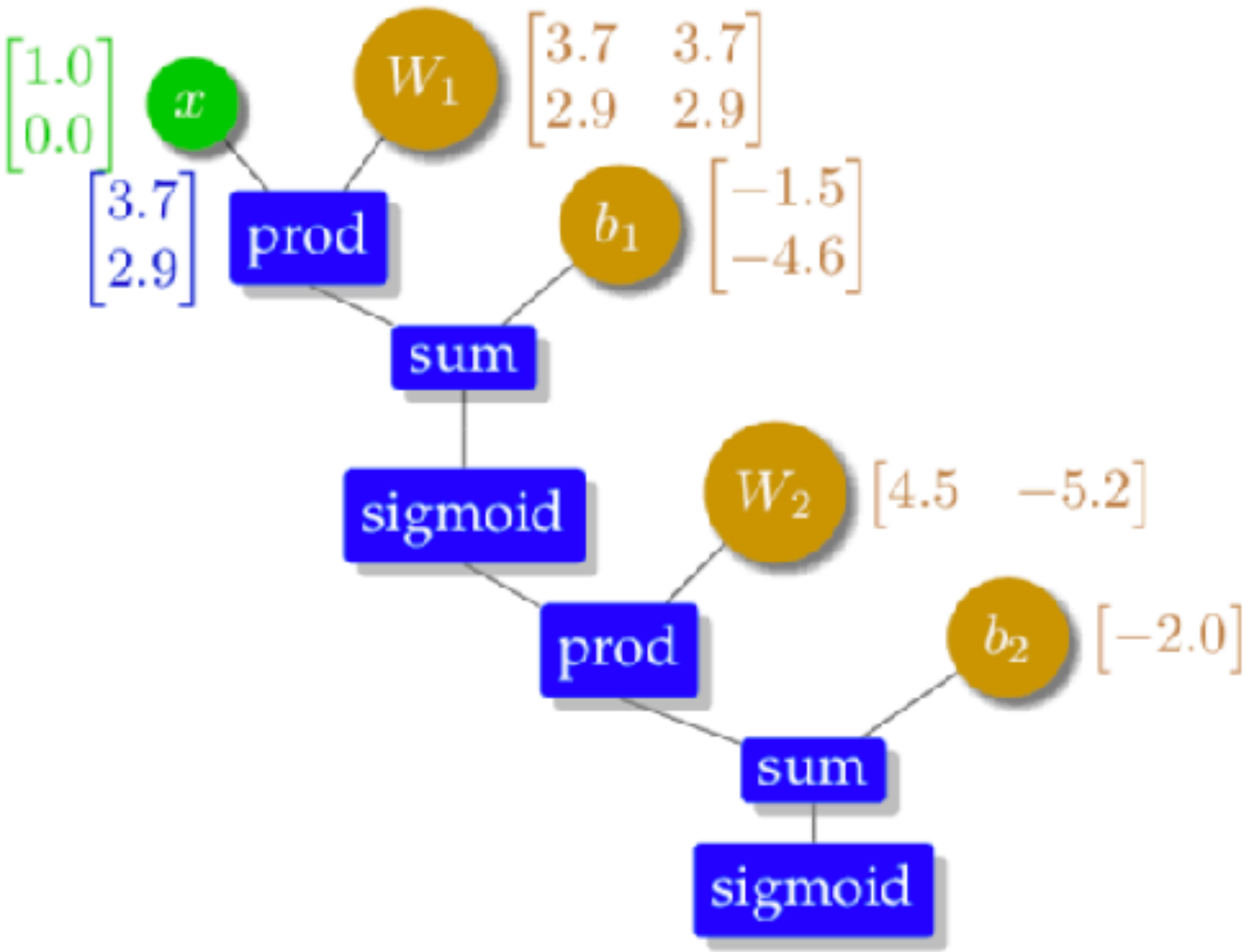

3.7
2.9
3.7
2.9
-1.5
-4.6
1
1
4.5
-5.2
-2.0

輸入資料進行計算 (1)



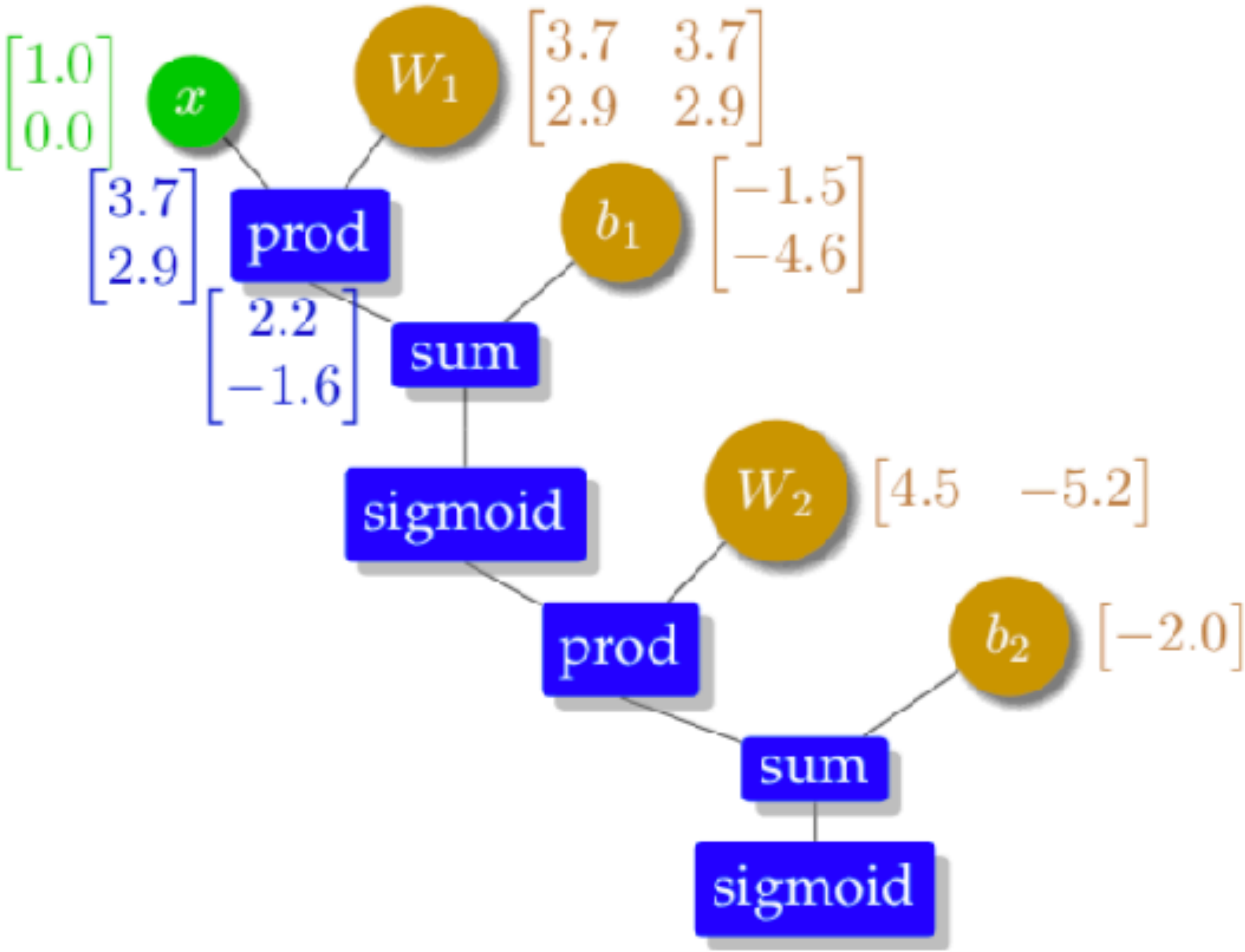
$\begin{bmatrix}1.0\\0.0\end{bmatrix}$
x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix}$
prod
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix}$
sum
sigmoid
W_2
$\begin{bmatrix}4.5 & -5.2\end{bmatrix}$
prod
b_2
$\begin{bmatrix}-2.0\end{bmatrix}$
sum
sigmoid

輸入資料進行計算 (2)



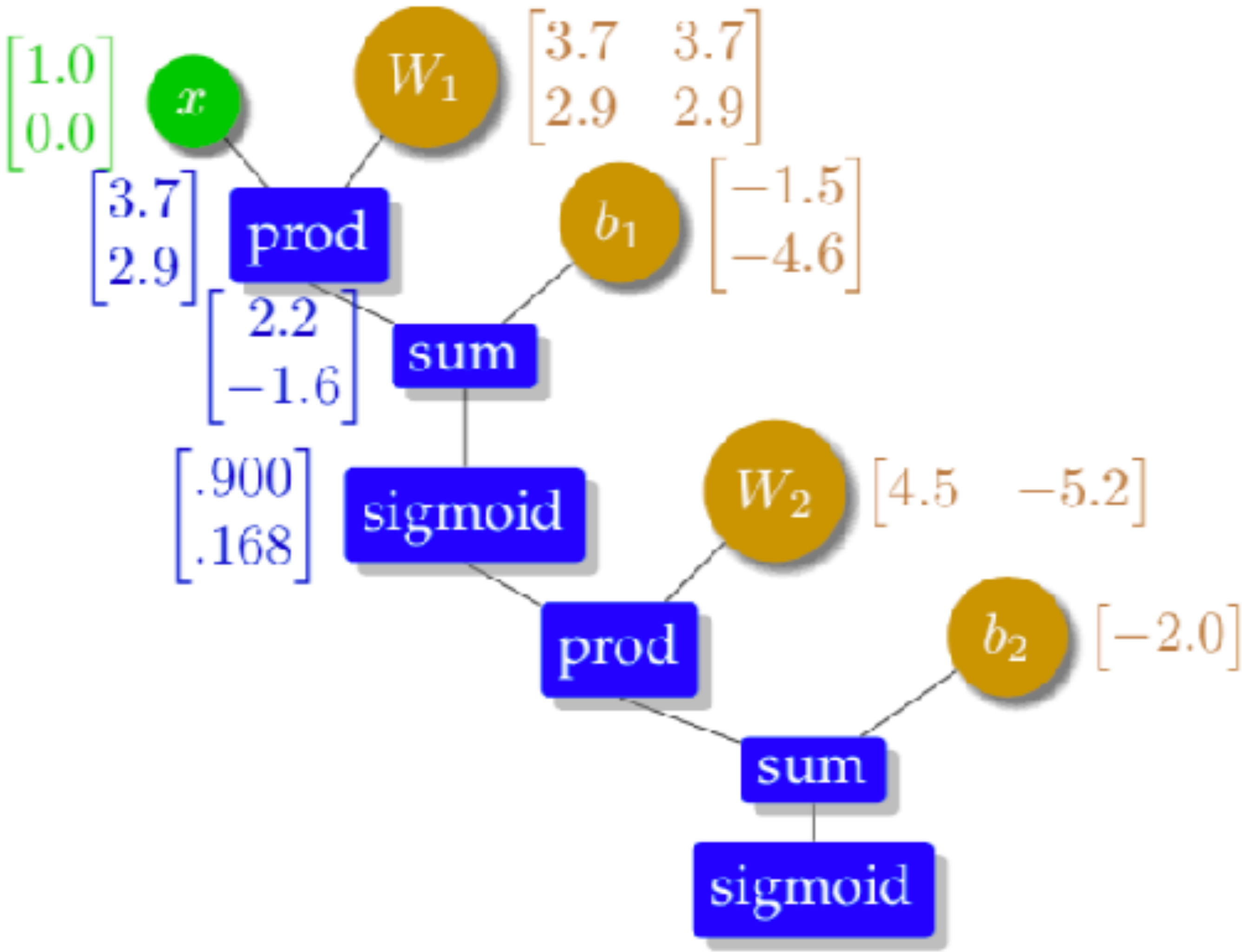
$\begin{bmatrix}1.0\\0.0\end{bmatrix}$
x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix}$
$\begin{bmatrix}3.7\\2.9\end{bmatrix}$
prod
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix}$
sum
sigmoid
W_2
$\begin{bmatrix}4.5 & -5.2\end{bmatrix}$
prod
b_2
$\begin{bmatrix}-2.0\end{bmatrix}$
sum
sigmoid

輸入資料進行計算 (3)



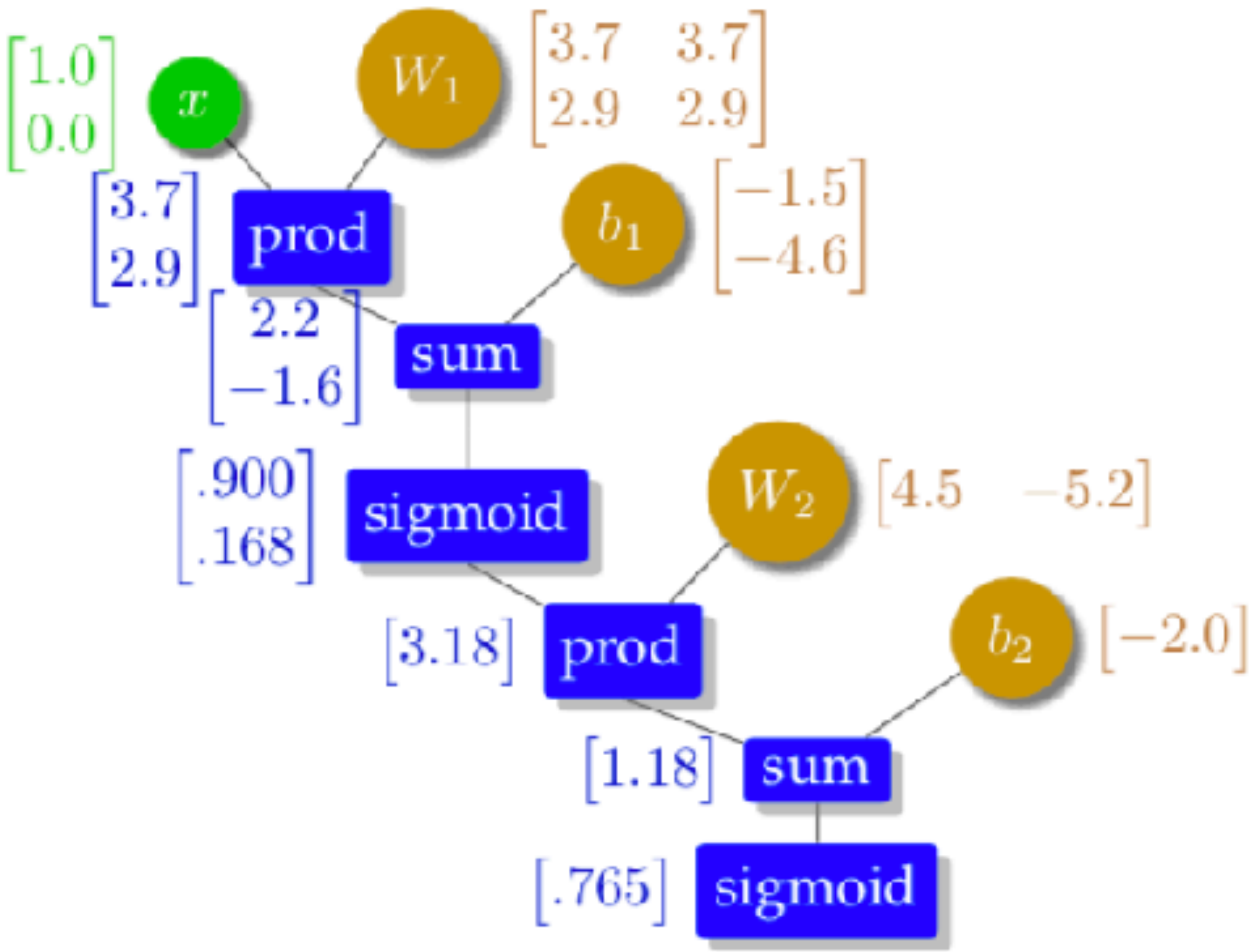
$\begin{bmatrix}1.0\\0.0\end{bmatrix}$
x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix}$
$\begin{bmatrix}3.7\\2.9\end{bmatrix}$
prod
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix}$
$\begin{bmatrix}2.2\\-1.6\end{bmatrix}$
sum
sigmoid
W_2
$\begin{bmatrix}4.5 & -5.2\end{bmatrix}$
prod
b_2
$\begin{bmatrix}-2.0\end{bmatrix}$
sum
sigmoid

輸入資料進行計算 (4)



$\begin{bmatrix}1.0\\0.0\end{bmatrix}$
x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix}$
$\begin{bmatrix}3.7\\2.9\end{bmatrix}$
prod
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix}$
$\begin{bmatrix}2.2\\-1.6\end{bmatrix}$
sum
$\begin{bmatrix}.900\\.168\end{bmatrix}$
sigmoid
W_2
$\begin{bmatrix}4.5 & -5.2\end{bmatrix}$
prod
b_2
$\begin{bmatrix}-2.0\end{bmatrix}$
sum
sigmoid

輸入資料進行計算 (5)



1.0
0.0
x
W1
3.7 3.7
2.9 2.9
3.7
2.9
prod
b1
−1.5
−4.6
2.2
−1.6
sum
.900
.168
sigmoid
W2
4.5 −5.2
[3.18]
prod
b2
[−2.0]
[1.18]
sum
[.765]
sigmoid

誤差函數

- For training, we need a measure how well we do ⇒ Cost function
- also known as objective function, loss, gain, cost, ...
- For instance L2 norm
-

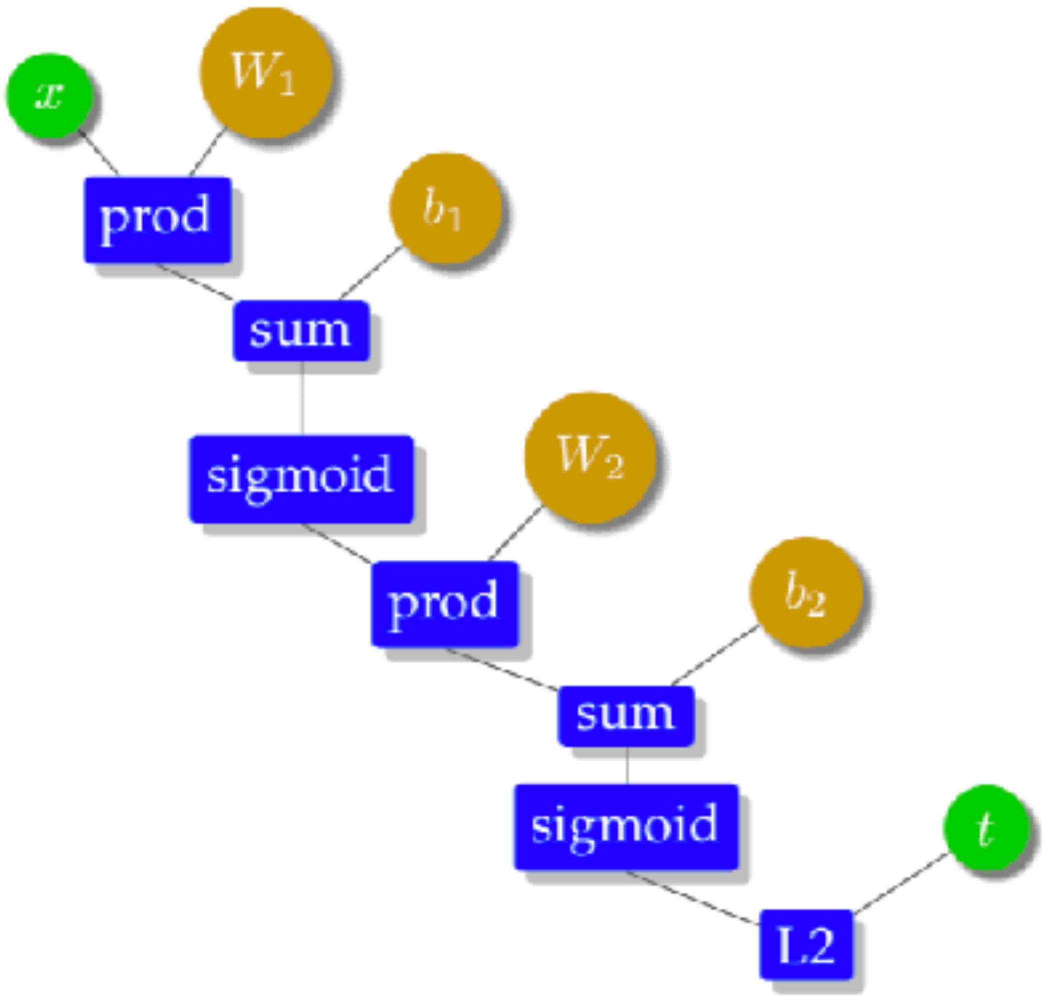
$$\text{error} = \frac{1}{2}(t - y)^2$$

最後層的權重更新

- 最後一層權重的線性組合 $s = \sum_k w_k h_k$
- 激化函數 $y = \text{sigmoid}(s)$
- 誤差函數 $E = \frac{1}{2}(t - y)^2$
- 對其中一個權重 w_k 取微分

$$\frac{dE}{dw_k} = \frac{dE}{dy}\frac{dy}{ds}\frac{ds}{dw_k}$$

在計算圖上的誤差計算



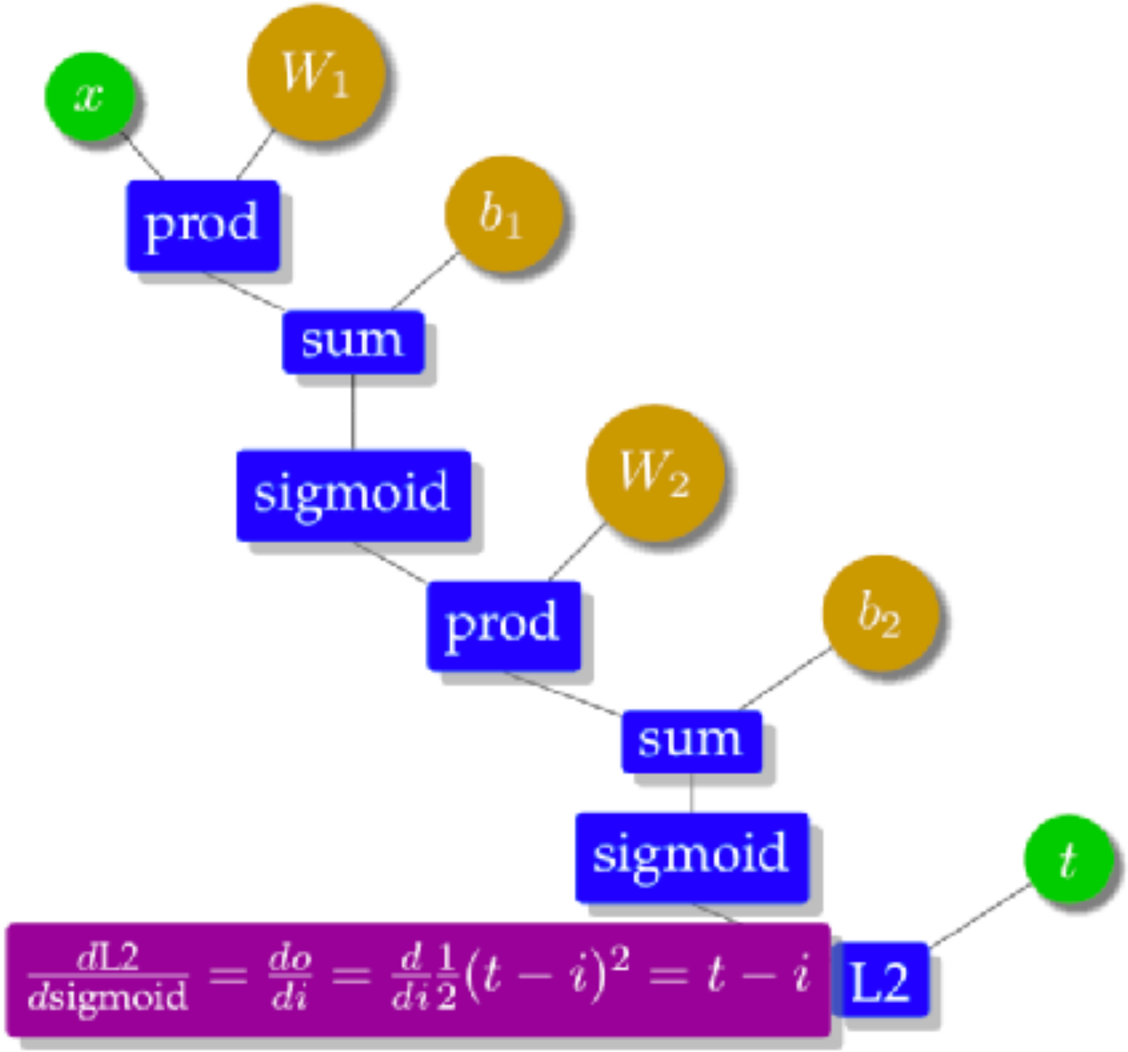
在計算圖上的誤差計算

- 在節點 A 上計算誤差 E 的微分值
- 假設我們已經計算好 E 對 B 的微分 $\frac{dE}{dB}$
- 所以現在我們只要計算B 對 A 的微分
- 假設 $B = A^2$ 則

$$A:\ \frac{dE}{dA} = \frac{dE}{dB}\frac{dB}{dA}$$

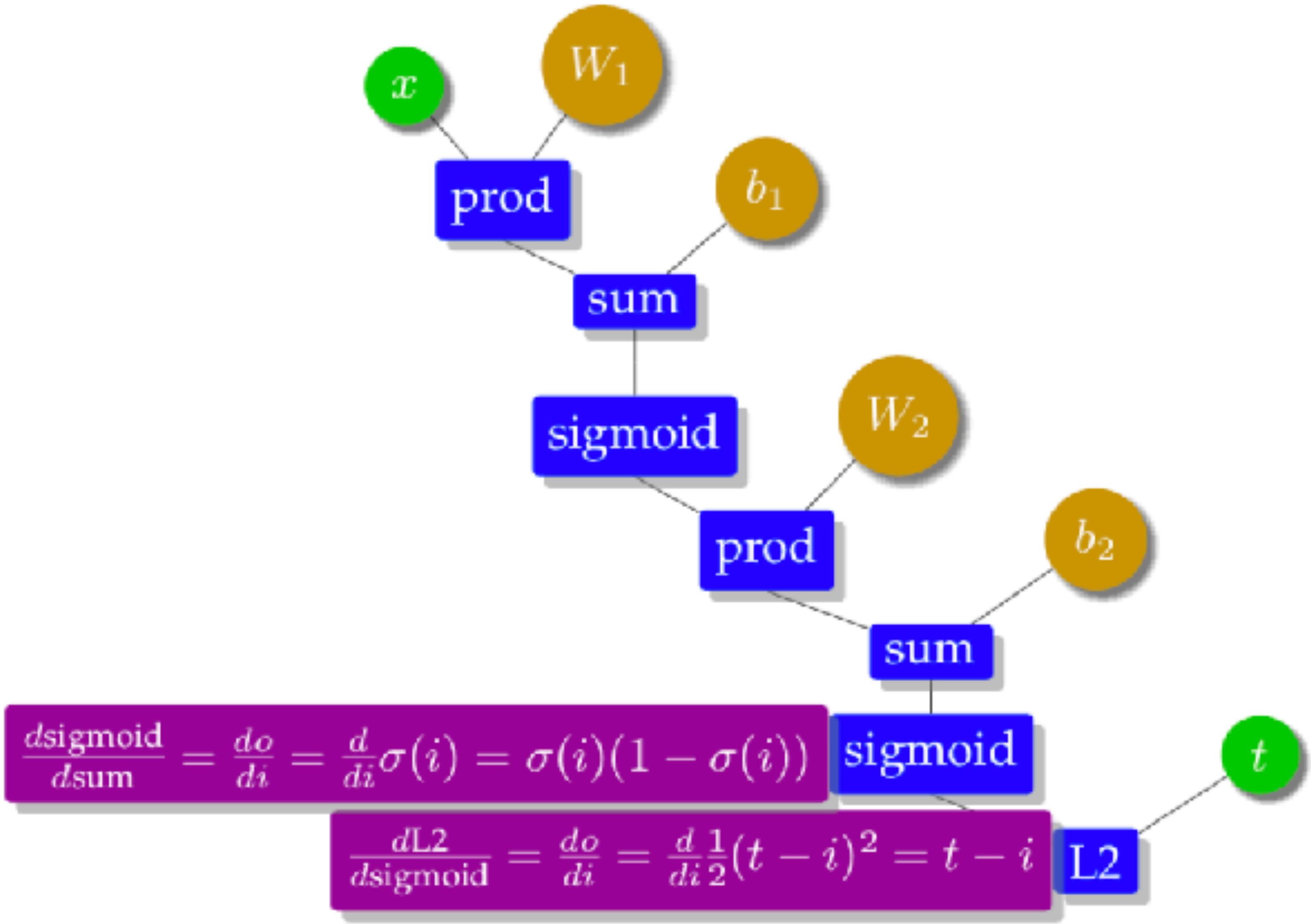
$$\frac{dB}{dA} = \frac{dA^2}{dA} = 2A$$

每個節點的微分 (1)



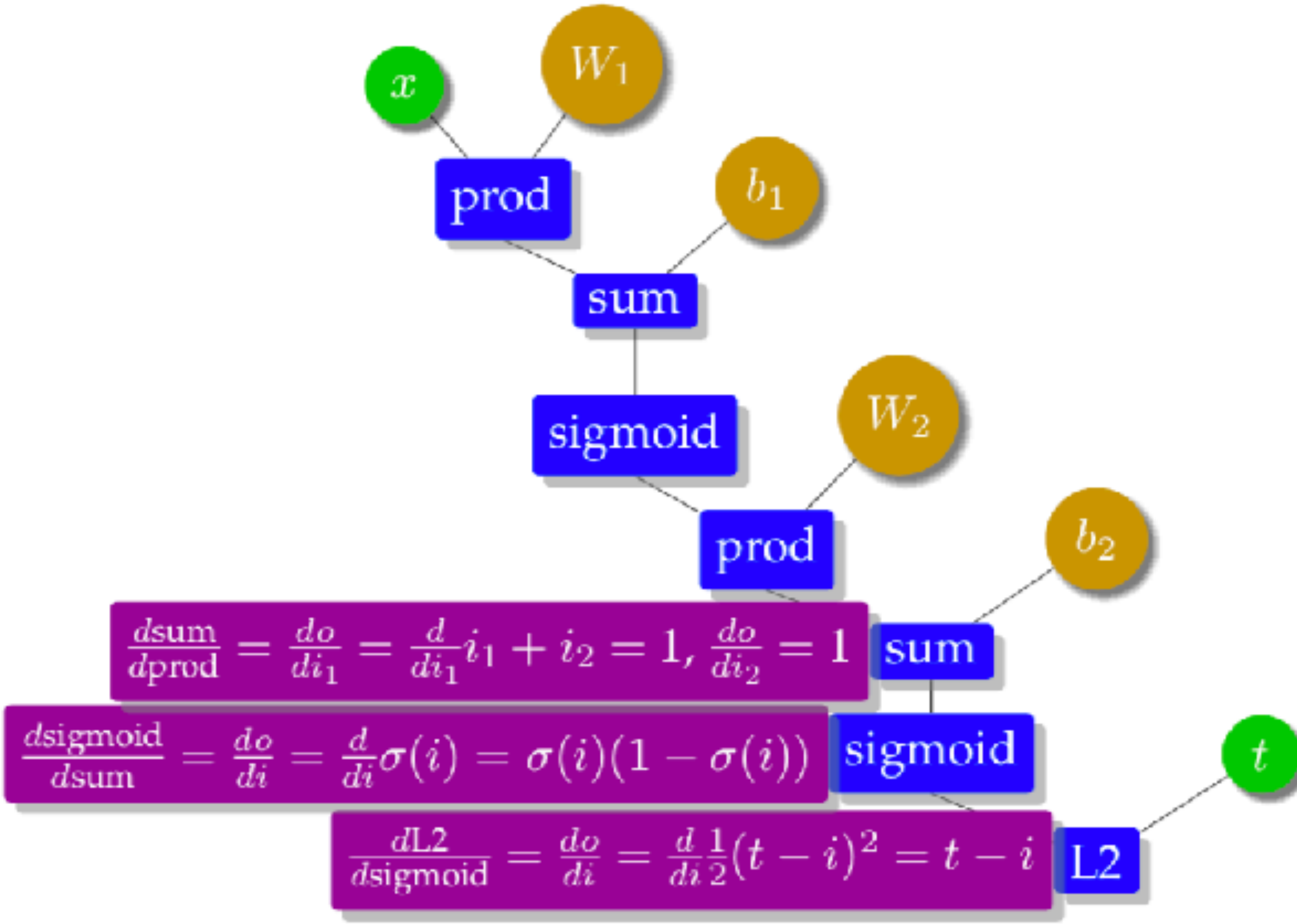
x
W_1
prod
b_1
sum
sigmoid
W_2
prod
b_2
sum
sigmoid
t
$\frac{dL2}{d\mathrm{sigmoid}} = \frac{do}{di} = \frac{d}{di}\frac{1}{2}(t-i)^2 = t - i$
L2

每個節點的微分 (2)



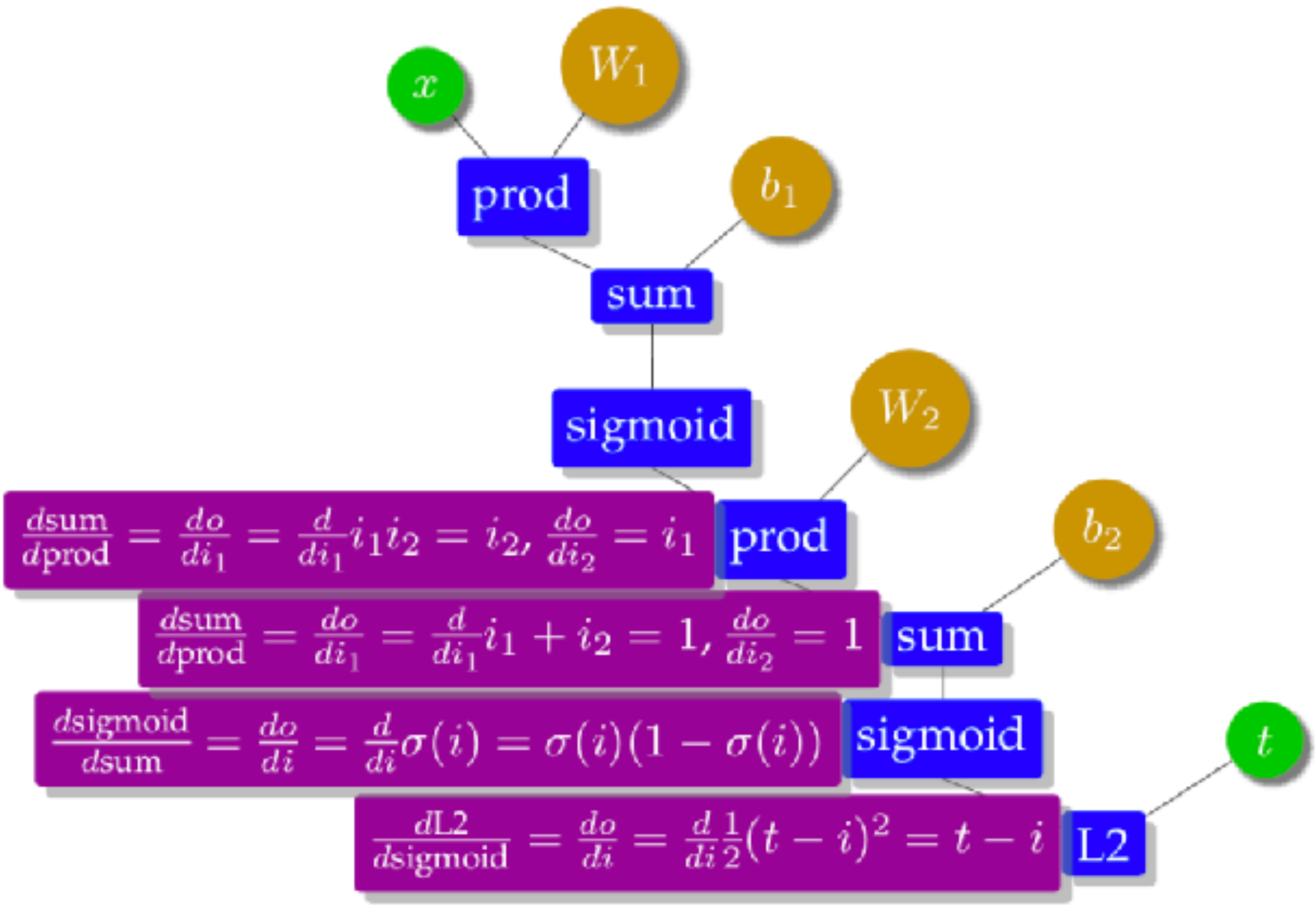
x
W_1
prod
b_1
sum
sigmoid
W_2
prod
b_2
sum
sigmoid
t
L2
$\frac{d\mathrm{sigmoid}}{d\mathrm{sum}} = \frac{do}{di} = \frac{d}{di}\sigma(i) = \sigma(i)(1-\sigma(i))$
$\frac{d\mathrm{L2}}{d\mathrm{sigmoid}} = \frac{do}{di} = \frac{d}{di}\frac{1}{2}(t-i)^2 = t-i$

每個節點的微分 (3)



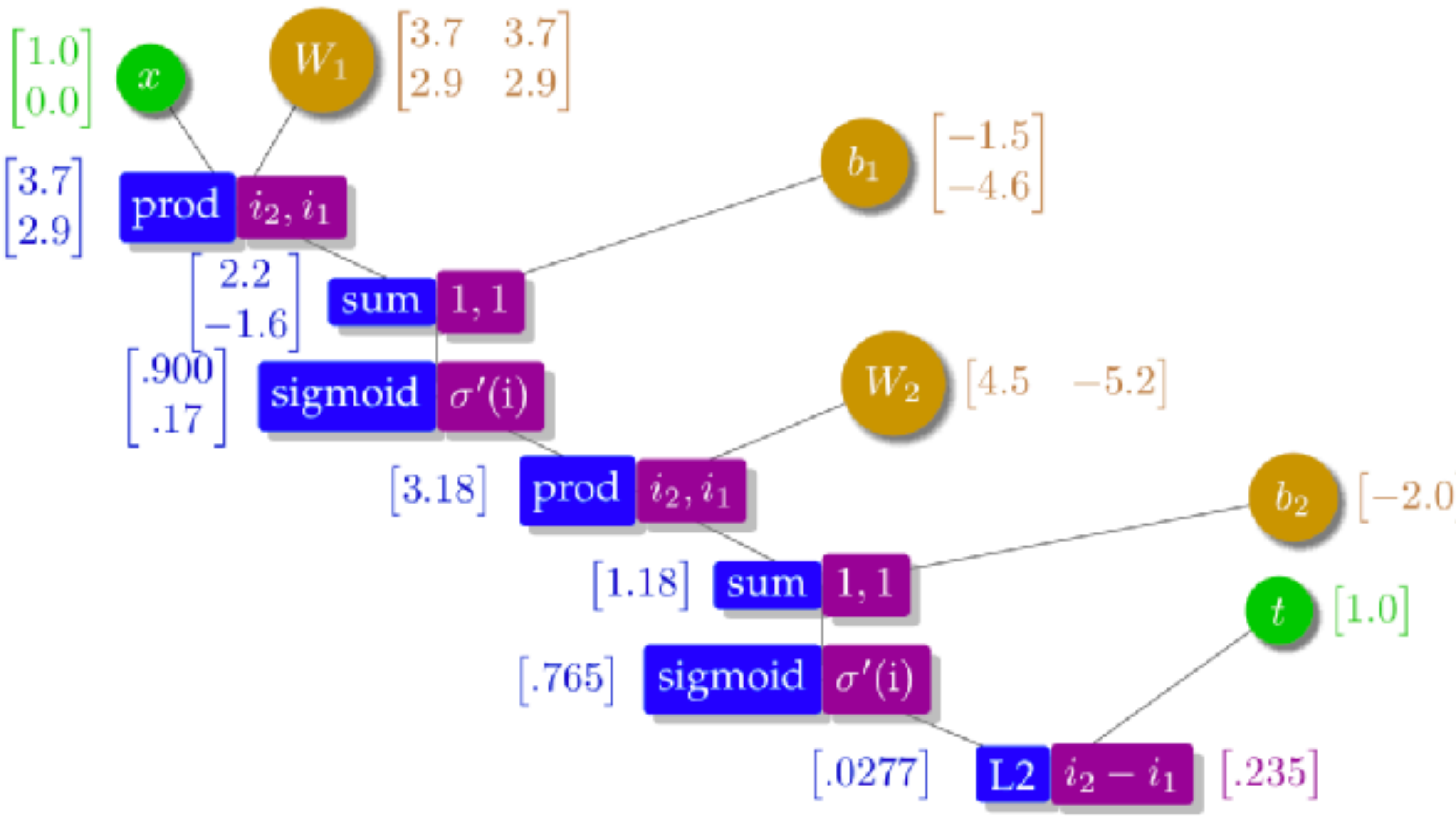
x
W_1
prod
b_1
sum
sigmoid
W_2
prod
b_2
$\frac{d\mathrm{sum}}{d\mathrm{prod}} = \frac{do}{di_1} = \frac{d}{di_1} i_1 + i_2 = 1, \frac{do}{di_2} = 1$
sum
$\frac{d\mathrm{sigmoid}}{d\mathrm{sum}} = \frac{do}{di} = \frac{d}{di}\sigma(i) = \sigma(i)(1 - \sigma(i))$
sigmoid
t
$\frac{d\mathrm{L2}}{d\mathrm{sigmoid}} = \frac{do}{di} = \frac{d}{di}\frac{1}{2}(t - i)^2 = t - i$
L2

每個節點的微分 (4)



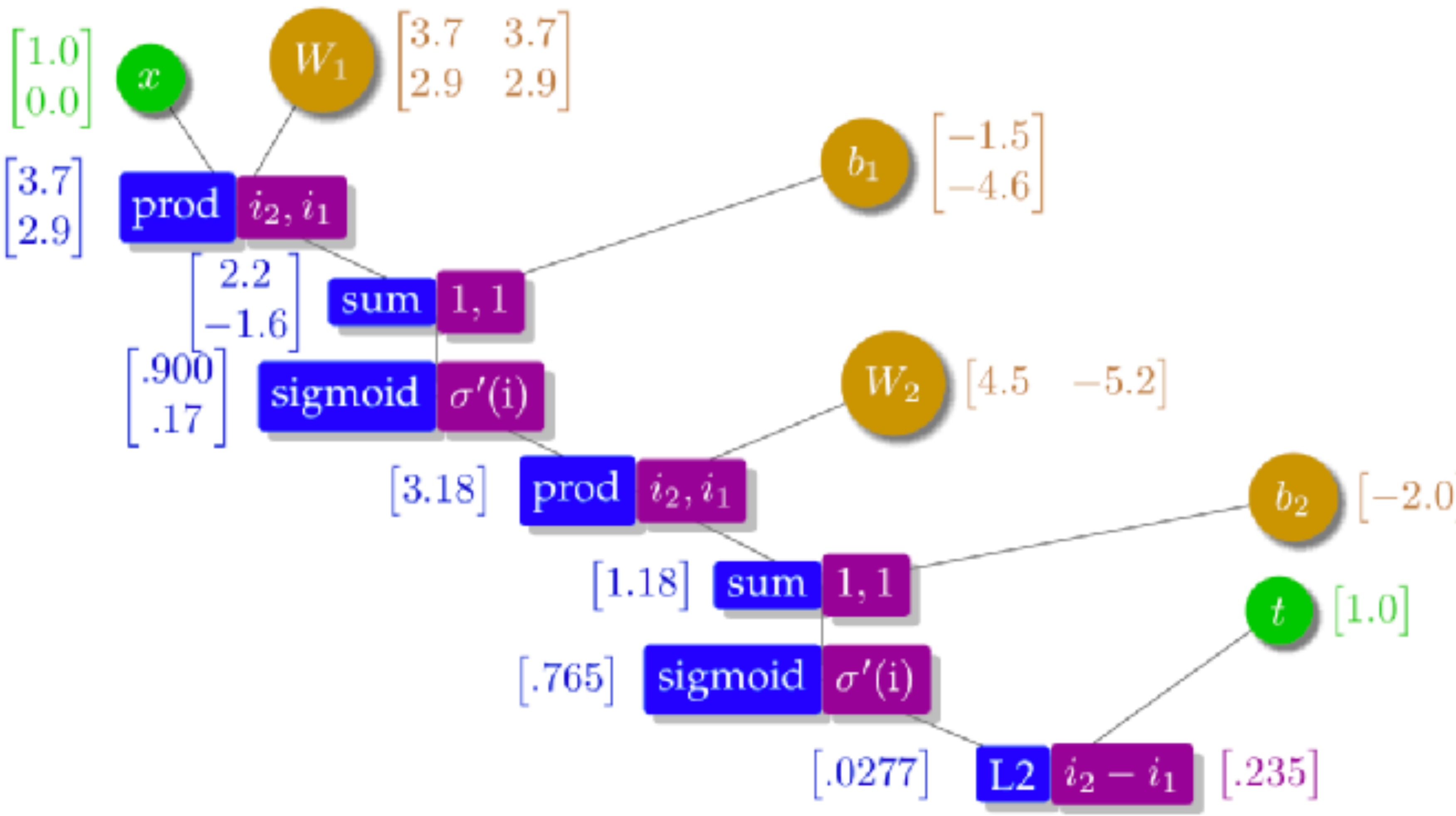
x
W_1
prod
b_1
sum
sigmoid
W_2
$\frac{d\text{sum}}{d\text{prod}} = \frac{do}{di_1} = \frac{d}{di_1} i_1 i_2 = i_2, \frac{do}{di_2} = i_1$
prod
b_2
$\frac{d\text{sum}}{d\text{prod}} = \frac{do}{di_1} = \frac{d}{di_1} i_1 + i_2 = 1, \frac{do}{di_2} = 1$
sum
$\frac{d\text{sigmoid}}{d\text{sum}} = \frac{do}{di} = \frac{d}{di}\sigma(i) = \sigma(i)(1-\sigma(i))$
sigmoid
t
$\frac{d\text{L2}}{d\text{sigmoid}} = \frac{do}{di} = \frac{d}{di}\frac{1}{2}(t-i)^2 = t-i$
L2

向後傳遞：微分計算 (1)



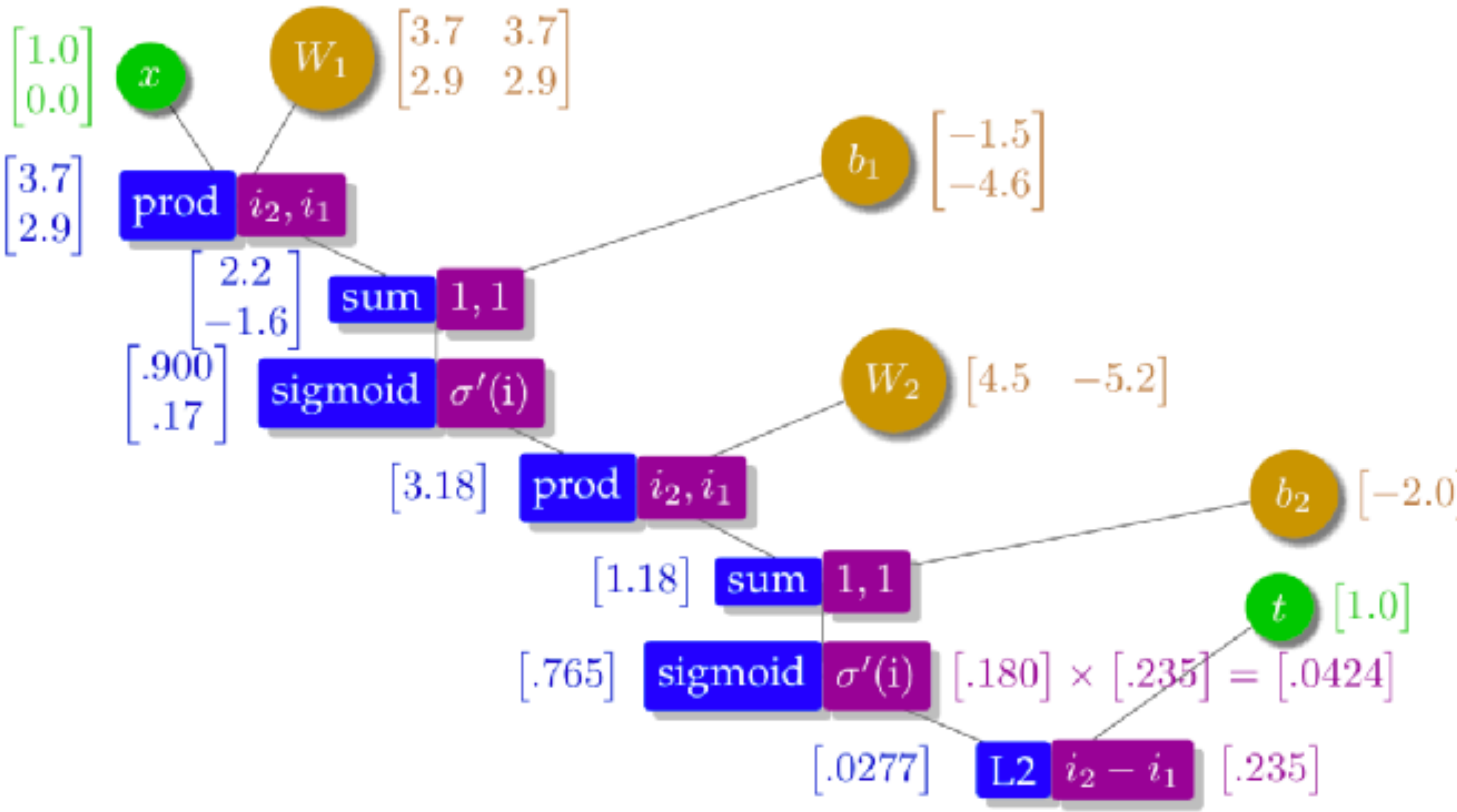
$\begin{bmatrix}1.0\\0.0\end{bmatrix}$
x
W_1
$\begin{bmatrix}3.7 & 3.7\\2.9 & 2.9\end{bmatrix}$
b_1
$\begin{bmatrix}-1.5\\-4.6\end{bmatrix}$
$\begin{bmatrix}3.7\\2.9\end{bmatrix}$
prod
i_2, i_1
$\begin{bmatrix}2.2\\-1.6\end{bmatrix}$
sum
$1, 1$
$\begin{bmatrix}.900\\.17\end{bmatrix}$
sigmoid
$\sigma'(i)$
W_2
$\begin{bmatrix}4.5 & -5.2\end{bmatrix}$
$[3.18]$
prod
i_2, i_1
b_2
$[-2.0]$
$[1.18]$
sum
$1, 1$
t
$[1.0]$
$[.765]$
sigmoid
$\sigma'(i)$
$[.0277]$
L2
$i_2 - i_1$
$[.235]$

向後傳遞：微分計算 (1)



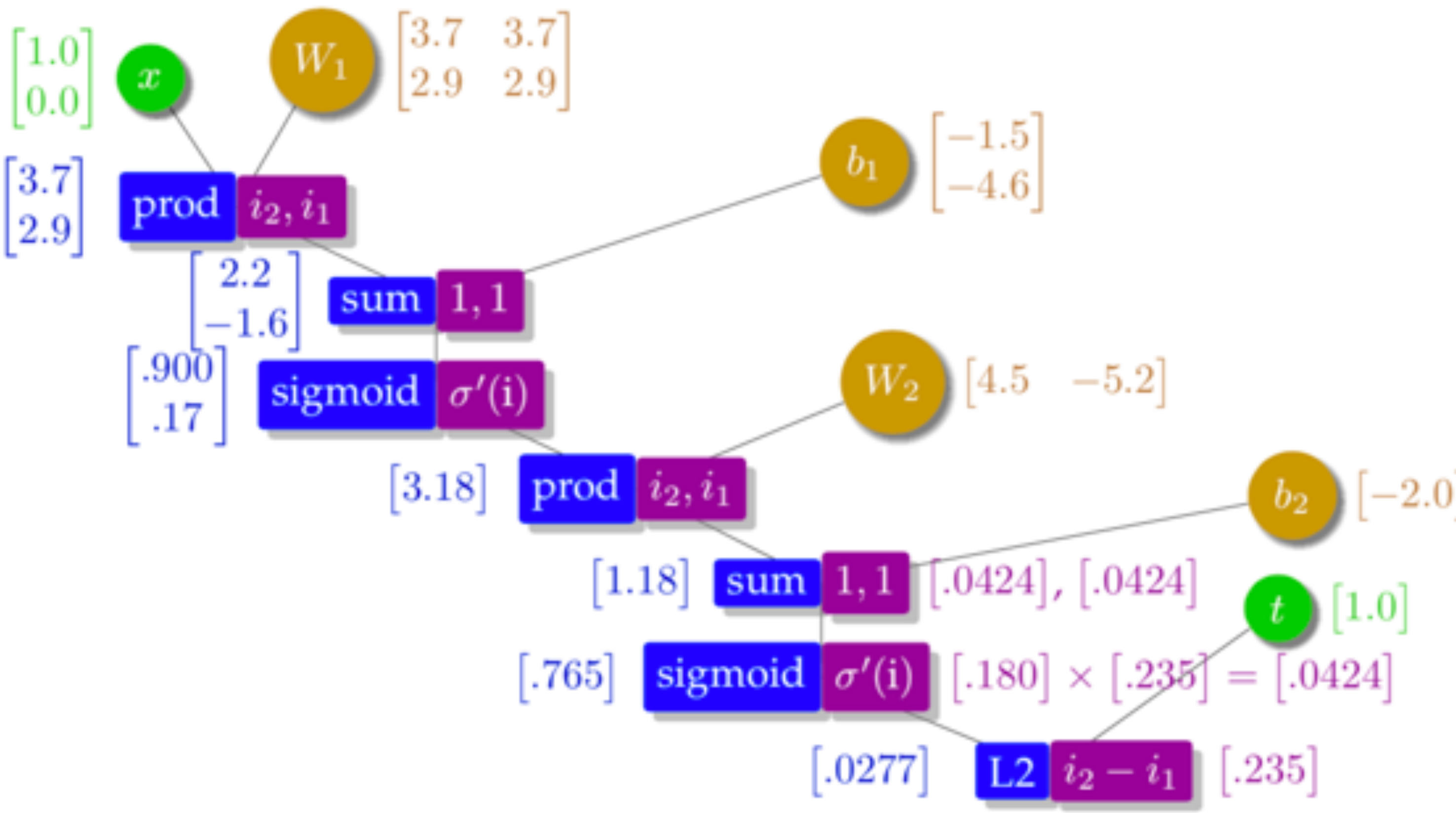
[1.0 0.0]
x
W1
[3.7 3.7; 2.9 2.9]
[3.7 2.9]
prod
i2, i1
b1
[−1.5 −4.6]
[2.2 −1.6]
sum
1, 1
[.900 .17]
sigmoid
σ′(i)
W2
[4.5 −5.2]
[3.18]
prod
i2, i1
b2
[−2.0]
[1.18]
sum
1, 1
t
[1.0]
[.765]
sigmoid
σ′(i)
[.0277]
L2
i2 − i1
[.235]

向後傳遞：微分計算 (2)



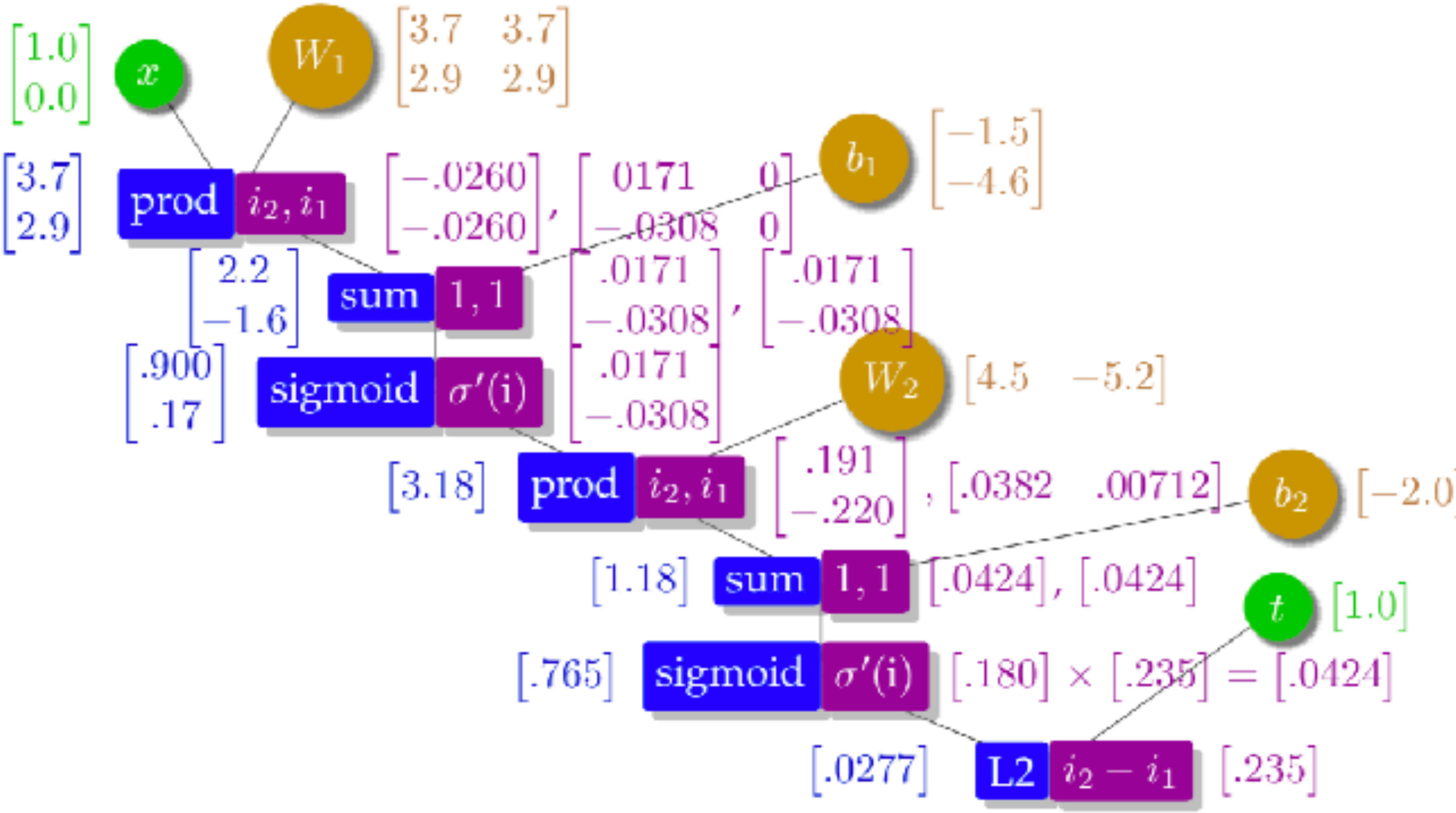
[1.0 0.0]
x
W1
[3.7 3.7; 2.9 2.9]
[3.7 2.9]
prod
i2, i1
b1
[−1.5 −4.6]
[2.2 −1.6]
sum
1, 1
[.900 .17]
sigmoid
σ′(i)
W2
[4.5 −5.2]
[3.18]
prod
i2, i1
b2
[−2.0]
[1.18]
sum
1, 1
t
[1.0]
[.765]
sigmoid
σ′(i)
[.180] × [.235] = [.0424]
[.0277]
L2
i2 − i1
[.235]

向後傳遞：微分計算 (3)



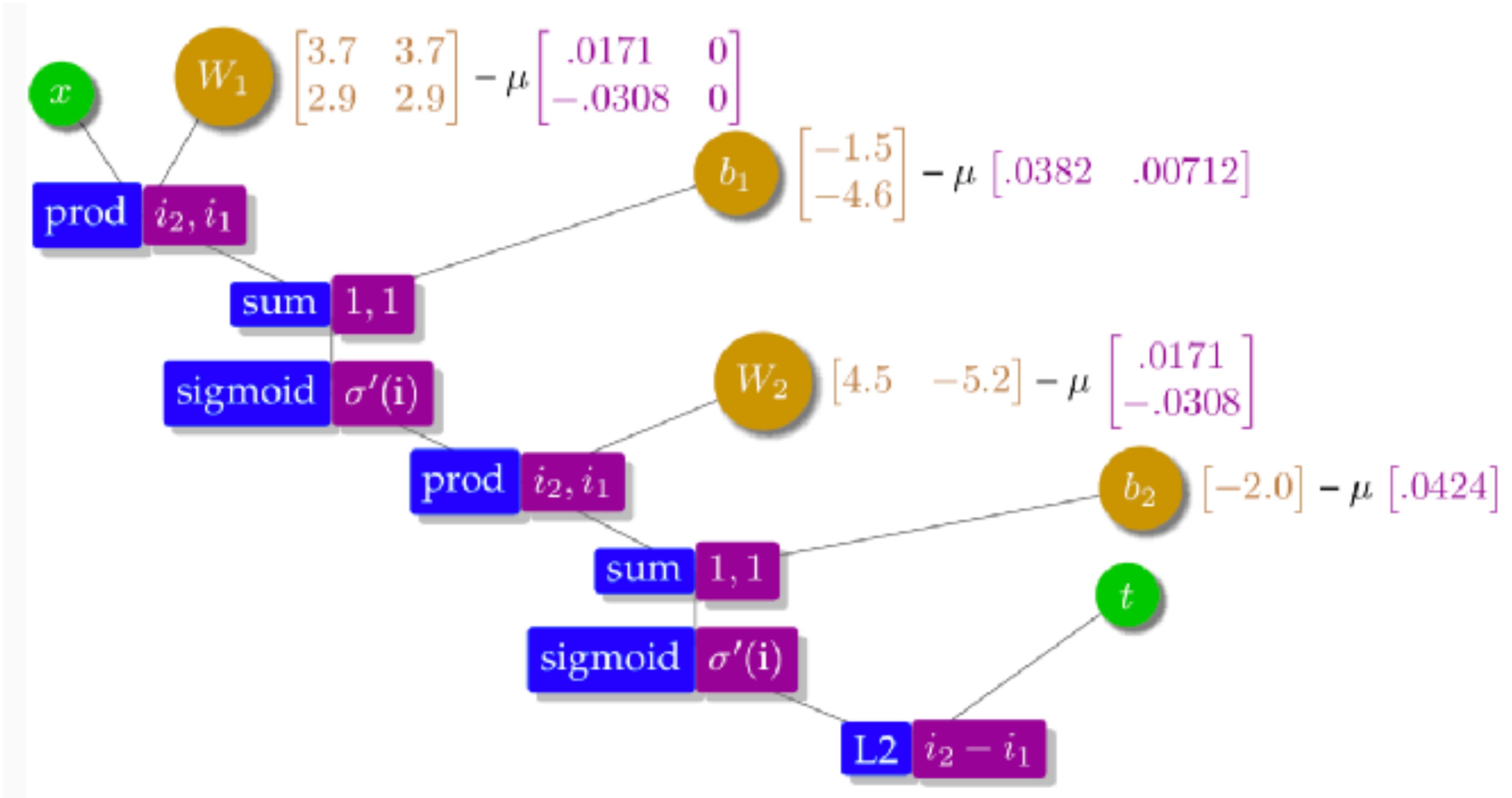
[1.0 0.0]
x
W1
[3.7 3.7; 2.9 2.9]
[3.7 2.9]
prod
i2, i1
b1
[−1.5 −4.6]
[2.2 −1.6]
sum
1, 1
[.900 .17]
sigmoid
σ′(i)
W2
[4.5 −5.2]
[3.18]
prod
i2, i1
b2
[−2.0]
[1.18]
sum
1, 1
[.0424], [.0424]
t
[1.0]
[.765]
sigmoid
σ′(i)
[.180] × [.235] = [.0424]
[.0277]
L2
i2 − i1
[.235]

向後傳遞：微分計算 (4)



x
W_1
prod
i_2, i_1
sum
1, 1
sigmoid
$\sigma'(i)$
b_1
W_2
b_2
t
L2
$i_2 - i_1$
[1.0, 0.0]
[3.7 3.7; 2.9 2.9]
[3.7, 2.9]
[−.0260, −.0260], [0171 0; −.0308 0]
[−1.5, −4.6]
[2.2, −1.6]
[.0171, −.0308], [.0171, −.0308]
[.900, .17]
[.0171, −.0308]
[4.5 −5.2]
[3.18]
[.191, −.220], [.0382 .00712]
[−2.0]
[1.18]
[.0424], [.0424]
[1.0]
[.765]
[.180] × [.235] = [.0424]
[.0277]
[.235]

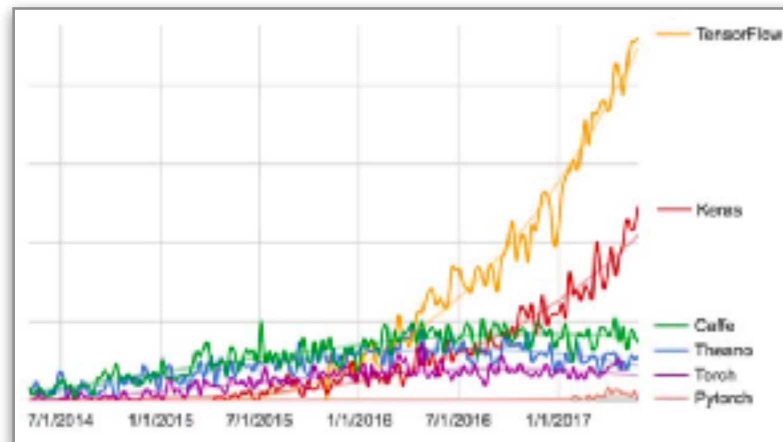
參數更新的梯度

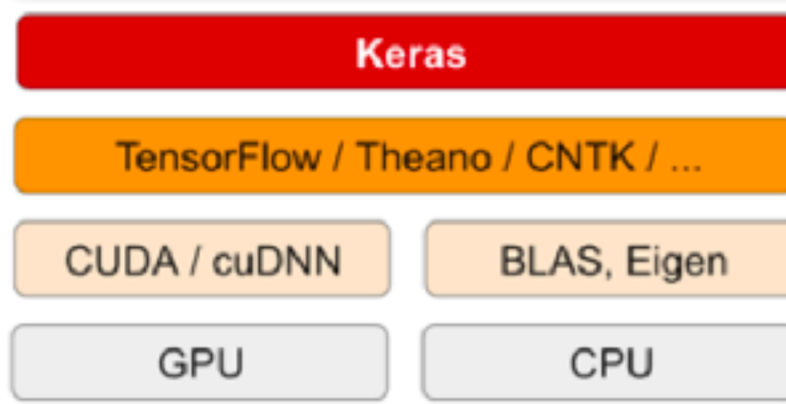


x
W_1
$\begin{bmatrix} 3.7 & 3.7 \\ 2.9 & 2.9 \end{bmatrix} - \mu \begin{bmatrix} .0171 & 0 \\ -.0308 & 0 \end{bmatrix}$
b_1
$\begin{bmatrix} -1.5 \\ -4.6 \end{bmatrix} - \mu \begin{bmatrix} .0382 & .00712 \end{bmatrix}$
prod
i_2, i_1
sum
$1, 1$
sigmoid
$\sigma'(\mathrm{i})$
W_2
$\begin{bmatrix} 4.5 & -5.2 \end{bmatrix} - \mu \begin{bmatrix} .0171 \\ -.0308 \end{bmatrix}$
prod
i_2, i_1
b_2
$\begin{bmatrix} -2.0 \end{bmatrix} - \mu \begin{bmatrix} .0424 \end{bmatrix}$
sum
$1, 1$
t
sigmoid
$\sigma'(\mathrm{i})$
L2
$i_2 - i_1$

暴增的各種工具

- 基本功能
  - Tensorflow (Google) www.tensorflow.org, eigen.tuxfamily.org
  - Theano (MILA lab/U. Montreal) deeplearning.net/software/theano
  - CogNitive Toolkit: CNTK (Microsoft) github.com/Microsoft/CNTK
  - Dynet (CMU)
  - MX-Net (Amazon)
  - Marian (AMU/Edinburgh)
- Library 程式館
  - Keras (Chollet/Google)
  - Torch, pyTorch (Facebook)
- 特定任務模型
  - openNMT (Harvard)
- 深度學習工具搜尋度變化時間圖





機器學習工具的功能很強

- 研發者只需要
  - 定義計算圖
  - 提供資料
  - 決定訓練的策略 (e.g., 批次訓練)
- 工具就處理其餘的計算細節

例子：Theano (1)

- 載入工具模組

```
>>> import numb
>>> import theano
>>> import theano.tensor as T
```

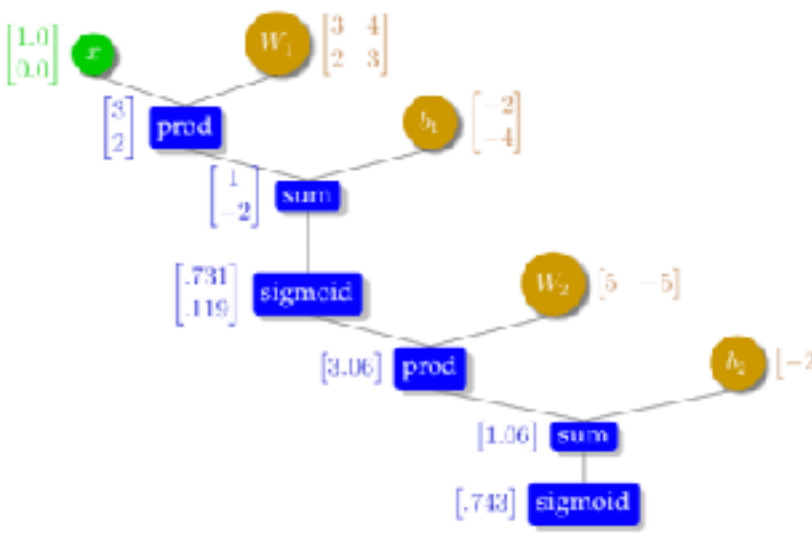
- Definition of parameters

```
>>> x = T.dmatrix()
>>> W = theano.shared(value=numpy.array([[3.0, 2.0],[4.0, 3.0]]))
>>> b = theano.shared(value=numpy.array([-2.0, -4.0]))
```

- Definition of feed-forward layer

```
>>> h = T.nnet.sigmoid(T.dot(x,W)+b)
```

  [note: x is matrix → several training examples]



- Define as callable function

```
>>> h_function = theano.function([x], h)
```

- Apply to data

```
>>> h_function([[1,0]])
>>> array([[ 0.73105858, 0.11920292]])
```

例子：Theano (2)

- 以同樣方式，設定隱藏層到輸出層的計算

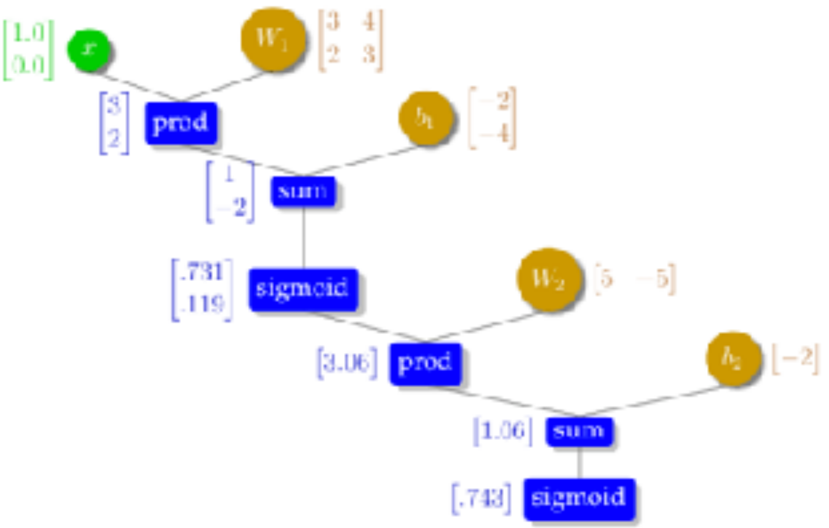
```
>>> W2 = theano.shared( value=numpy.array[5.0, -5.0] ))
>>> b2 =  theano.shared( -2.0)
>>> y_pred = T.nnet.sigmoid(T.dot(h, W2)+b2)
```

- 定義成可以呼叫的 predict 函數callable function 測試整個網路模型

```
>>> predict = theano.function([x], y_pred)
```

- 運作到資料上

```
>>> predict([[1, 0]])
array([[ 0.743 ]])
```



訓練模型 (1)

- 首先定義輸出的標準答案 y

```
>>> y = T.dvector()
```

- 接著定義誤差函數 (用 L2 norm)

```
>>> l2 = (y-y_pred)**2
>>> cost = l2.mean()
```

- 梯度下降訓練：計算微分

```
>>> gW, gb, gW2, gb2  = T.grad(cost, [W, b, W2, b2])
```

- 更新規則 (實用學習速率 0.1)

```
>>> train = theano.function(inputs=[x, y], outputs=[y_pred, cost],
                    updates=(  (W, W-0.1*gW), (b, b-0.1*gb),
                               (W2, W2-0.1*gW2), (b2, b2-0.1*gb2) ))
```

訓練模型 (2)

- 準備訓練資料
  >>> DATA_X = numpy.array([[0,0], [0,1], [1, 0], [1,1]   ])

- 用起始模型預測訓練資料的答案
  >>> predict(DATA_X)
  array([0.18, 0.74, 0.74, 0.33])

訓練模型 (3)

- 用訓練資料進行訓練
  >>> train(DATA_X, DATA_Y)
  [array([0.1833, 0.7426, 0.7426, 0.3343], array(0.06948)]

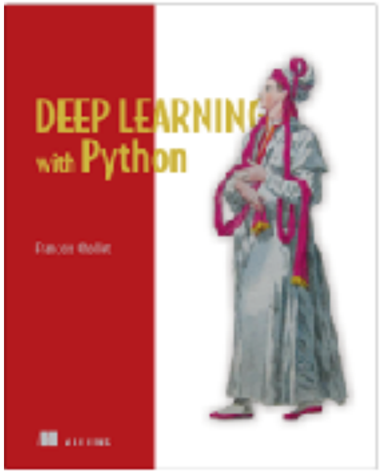
  [註：回傳訓練前的預測、誤差]

- 第二次呼叫訓練函數
  >>> train(DATA_X, DATA_Y)
  [array([0.1835, 0.7426, 0.7432 , 0.3322], array(0.06923)]

  [註：預測、誤差都小幅變好]

程式範例 github.com/fchollet/deep-learning-with-python-notebooks

- 2.1-a-first-look-at-a-neural-network.ipynb
- 3.5-classifying-movie-reviews.ipynb
- 3.6-classifying-newswires.ipynb
- 3.7-predicting-house-prices.ipynb
- 4.4-overfitting-and-underfitting.ipynb
- 5.1-introduction-to-convnets.ipynb
- 5.2-using-convnets-with-small-datasets.ipynb
- 5.3-using-a-pretrained-convnet.ipynb
- 5.4-visualizing-what-convnets-learn.ipynb
- 6.1-one-hot-encoding-of-words-or-characters.ipynb
- 6.1-using-word-embeddings.ipynb
- 6.2-understanding-recurrent-neural-networks.ipynb
- 6.3-advanced-usage-of-recurrent-neural-networks.ipynb
- 6.4-sequence-processing-with-convnets.ipynb
- 8.1-text-generation-with-lstm.ipynb
- 8.2-deep-dream.ipynb
- 8.3-neural-style-transfer.ipynb
- 8.4-generating-images-with-vaes.ipynb
- 8.5-introduction-to-gans.ipynb



7.5.3.1 電影評論二元分類器：Keras (1)

- 用 Keras 預設 IMDB 資料訓練與測試資料各 25,000筆正負面各佔 50% (80MB)
- 限制詞彙為最高頻的10,000 詞，其他視為 UNK，詞由字串轉為整數 [1, 9999]
-

Listing 3.1 Loading the IMDB dataset

```
from keras.datasets import imdb

(train_data, train_labels), (test_data, test_labels) = imdb.load_data(
    num_words=10000)
```

```
>>> train_data[0]
[1, 14, 22, 16, ... 178, 32]
>>> train_labels[0]
1
>>> max([max(sequence) for sequence in train_data])
9999

>>> word_index = imdb.get_word_index()
>>> reverse_index = dict([(value, key) for (key, value)
...     in word_index.items()])
>>> ' '.join([reverse_word_index.get (i - 3, '?')
...     for i in train_data[0]])

"? this film was just brilliant casting location scenery
   ...   ...   ...   ...   …
was someone's life after all that was shared with us all"
```

資料來源：www.manning.com/books/deep-learning-with-python
github.com/fchollet/deep-learning-with-python-notebooks

電影評論二元分類器的例子：Keras (2)

Listing 3.2 Encoding the integer sequences into a binary matrix

```
import numpy as np

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)
```

產生全0 矩陣
大小為 |seq| x 1000

在第 i 筆資料 result[i] 的
詞 j 的對應位置設為 1.0

向量化 25000 筆訓練資料

向量化 25000 筆測試資料

```
>>> test = np.zeros((7, 7))
>>> test[0, [3,5]]= 1.
>>> test[0]
array([0., 0., 0., 1., 0., 1., 0.])

>>> x_train = vectorize_sequences(train_data)
>>> x_train[0]
array([0., 1., 1., ..., 0., 0., 0.])
>>> len(x_train[0])
10000
>>> len(x_train)
25000
```

電影評論二元分類器的例子：Keras (3)

Listing 3.3 The model definition

```
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))
```

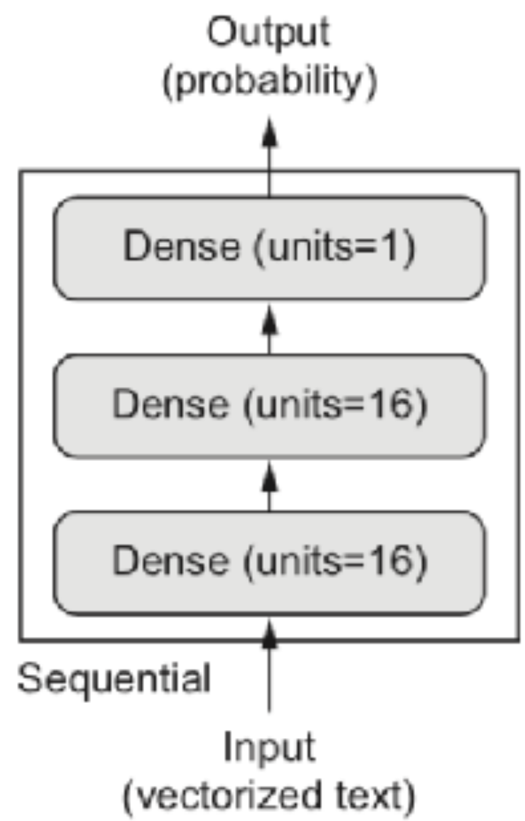
relu: 斜坡整流函數
sigmoid: S 函數

Listing 3.4 Compiling the model

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])
```

Listing 3.5 Configuring the optimizer

```
from keras import optimizers

model.compile(optimizer=optimizers.RMSprop(lr=0.001),
              loss='binary_crossentropy',
              metrics=['accuracy'])
```



- 定義模型、編譯模型、配置最佳化程式

訓練模型、繪製損失、精確率圖

Listing 3.8 Training your model

```
model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['acc'])

history = model.fit(partial_x_train,
                    partial_y_train,
                    epochs=20,
                    batch_size=512,
                    validation_data=(x_val, y_val))
```

總共執行 20 次疊代
每批次 512 筆資料
用驗證資料 val 檢查過適

Listing 3.9 Plotting the training and validation loss

```
import matplotlib.pyplot as plt

history_dict = history.history
loss_values = history_dict['loss']
val_loss_values = history_dict['val_loss']

epcchs = range(1, len(acc) + 1)

plt.plot(epochs, loss_values, 'bo', label='Training loss')
plt.plot(epochs, val_loss_values, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

bo
代表藍點

b
代表藍線

繪圖觀察訓練過程

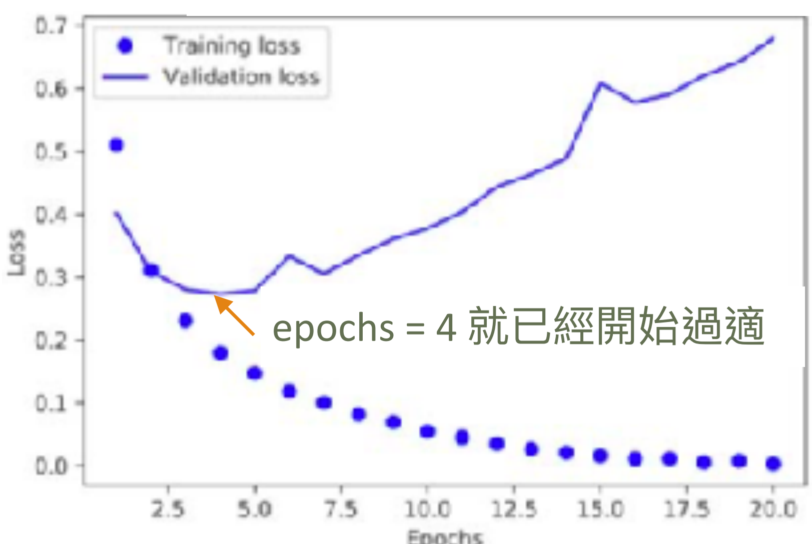
Listing 3.10 Plotting the training and validation accuracy

```
plt.clf()
acc_values = history_dict['acc']
val_acc_values = history_dict['val_acc']

plt.plot(epochs, acc, 'bo', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.show()
```

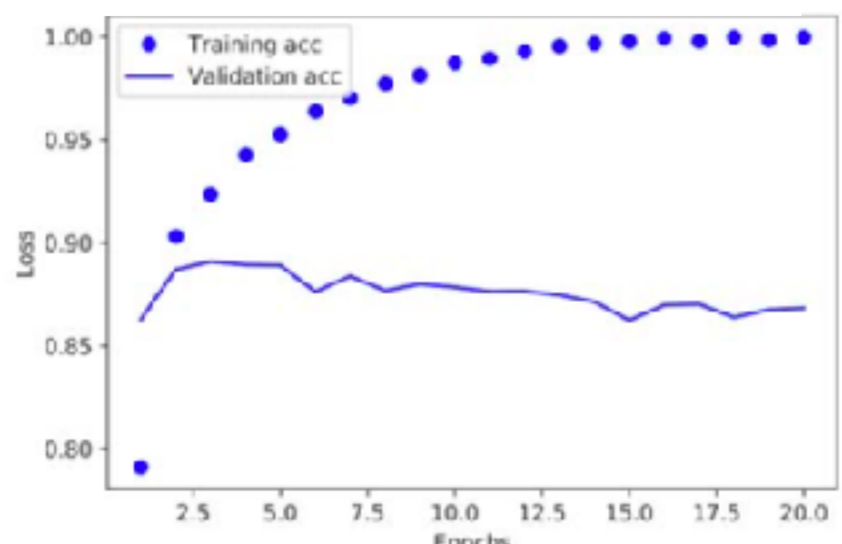
清空圖形

訓練資料用藍點 bo 表示
驗證資料用藍線 b 表示

損失函數 (訓練 vs. 驗證)



精確率 (訓練 vs. 驗證)



重新訓練一個模型

Listing 3.11 Retraining a model from scratch

```
model = models.Sequential()
model.add(layers.Dense(16, activation='relu', input_shape=(10000,)))
model.add(layers.Dense(16, activation='relu'))
model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='rmsprop',
              loss='binary_crossentropy',
              metrics=['accuracy'])

model.fit(x_train, y_train, epochs=4, batch_size=512)
results = model.evaluate(x_test, y_test)
```

指定 epochs = 4 避免過適

```
>>> results
[0.29, 0.88]
>>> model.predict(x_test)
array([[ 0.98006207]
       [ 0.99758697]
       [ 0.99975556]
       ...,
       [ 0.82167041]
       [ 0.02885115]
       [ 0.65371346]], dtype=float32)
```

損失函數 = .29 精確度 = 88% (目前最高可達 95%)

這筆資料分類信心高 (0.99, 0.01)

這筆資料分類信心低 (0.65, 0.35)

做更多的實驗

- 試試看一個隱藏層或三個隱藏層，而不是兩層
- 試試看每層多幾個或少幾個隱藏單元 32, 64 等等
- 試試看改變錯誤損失函數 mse 而不是binary_crossentropy.
- 試試看用 tanh 激化函數而不用 relu

小結

- 資料需要很多預處理，變成 tensors 才能饋入 NN
  - 詞可以用 0 , 1 的二元向量，但是最好用低維度高密度的詞內嵌向量
- 完全連階層 + relu 激化韓式可以用來解決很多問題

  二元分類問題
  - 用完全連階層結束到一個單元
  - 用 sigmoid 激化函數 產生 0 到 1 的機率值
  - 損失函數應該就用 binary_crossentropy
  - 也可以用 rmsprop 最佳化通常是很有效的選擇
  - is generally a good enough choice,
- 模型對訓練資料愈來愈有效，就開始過度適合，對新資料反而效果不佳
- 所以，必須觀察訓練資料以外的資料

7.5.3.2 類神經語言模型 (1)

- 用 big.txt 作為訓練資料
- 輸入 2 個詞，預測下一個詞

Listing

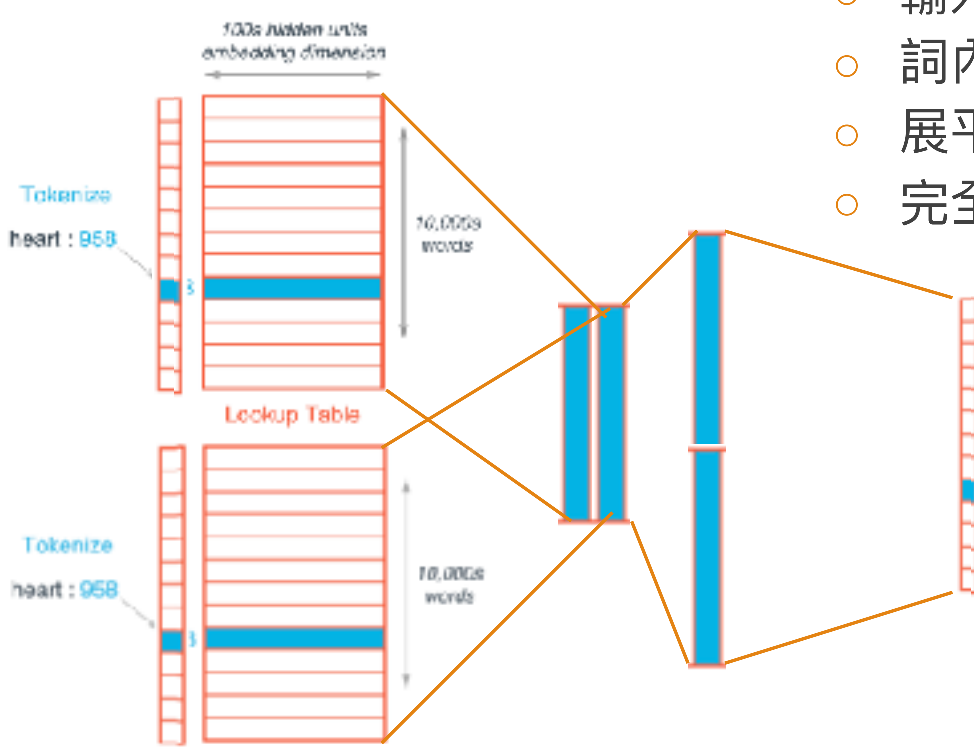
```
import os; os.environ['KERAS_BACKEND'] = 'tensorflow'
from keras.preprocessing.text import Tokenizer
from keras.utils import to_categorical
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Flatten, Embedding

data = open('big.txt').read()[:100000]
tokenizer = Tokenizer()
tokenizer.fit_on_texts([data])
encoded = tokenizer.texts_to_sequences([data])[0]
vocab_size = len(tokenizer.word_index) + 1

sequences = [ encoded[i-2:i+1] for i in range(2, len(encoded)) ] # trigrams
sequences = array(sequences)
X, y = sequences[:,:-1], sequences[:,-1] # first 2 as input, last as output
y = to_categorical(y, num_classes=vocab_size)
```

資料來源：www.manning.com/books/deep-learning-with-python
github.com/fchollet/deep-learning-with-python-notebooks

7.5.3.2 類神經語言模型 (2)

Listing

```
from keras import models
from keras import layers

model = models Sequential()
model.add(layers.Embedding(vocab_size, 10, input_length=max_length-1))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(vocab_size, activation='softmax'))
```



- 輸入：兩個 1-hot 的詞
- 詞內嵌編碼層
- 展平層：二維變一維
- 完全連階層：

7.5.3.2 類神經語言模型 (3)

Listing

```
from keras import models
from keras import layers

model = models Sequential()
model.add(layers.Embedding(vocab_size, 10, input_length=max_length-1))
model.add(layers.Flatten())
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(vocab_size, activation='softmax'))
print(model.summary())

model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 2, 10)             129010
_________________________________________________________________
flatten_1 (Flatten)          (None, 20)                0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               2100
_________________________________________________________________
dense_2 (Dense)              (None, 12901)             1303001
=================================================================
Total params: 1,434,111
```

資料來源：www.manning.com/books/deep-learning-with-python
github.com/fchollet/deep-learning-with-python-notebooks

7.5.3.2 類神經語言模型 (4)

Listing

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=10, verbose=2)
```

```
Epoch 1/20 - 105s - loss: 6.6446 - acc: 0.0953
Epoch 2/20 - 101s - loss: 5.9738 - acc: 0.1284
Epoch 3/20 - 101s - loss: 5.6269 - acc: 0.1418
Epoch 4/20 - 103s - loss: 5.3645 - acc: 0.1518
Epoch 5/20 - 103s - loss: 5.1544 - acc: 0.1591
Epoch 6/20 - 102s - loss: 4.9761 - acc: 0.1658
Epoch 7/20 - 102s - loss: 4.8187 - acc: 0.1714
Epoch 8/20 - 103s - loss: 4.6782 - acc: 0.1765
Epoch 9/20 - 102s - loss: 4.5495 - acc: 0.1816
Epoch 10/20 - 102s - loss: 4.4331 - acc: 0.1866
Epoch 11/20 - 104s - loss: 4.3300 - acc: 0.1916
Epoch 12/20 - 104s - loss: 4.2339 - acc: 0.1979
Epoch 13/20 - 103s - loss: 4.1505 - acc: 0.2049
Epoch 14/20 - 103s - loss: 4.0767 - acc: 0.2136
Epoch 15/20 - 103s - loss: 4.0106 - acc: 0.2215
Epoch 16/20 - 103s - loss: 3.9539 - acc: 0.2289
Epoch 17/20 - 104s - loss: 3.9042 - acc: 0.2356
Epoch 18/20 - 103s - loss: 3.8599 - acc: 0.2406
Epoch 19/20 - 103s - loss: 3.8205 - acc: 0.2454
Epoch 20/20 - 104s - loss: 3.7867 - acc: 0.2499
```

7.5.3.2 類神經語言模型 (5)

Listing

```
def generate(model, x, n_words):
    res = x
    for _ in range(n_words):
        padded = pad_sequences([res], maxlen=2, padding='pre')
        res += [ model.predict_classes(padded)[0] ]
    return res

revIndex = dict( [ (value, key)
                   for key, value in tokenizer.word_index.items() ])

def view(index): return ' '.join( [revIndex.get(i, '?') for i in index])

print( 'I want:', view( generate(model,
                              tokenizer.texts_to_sequences(['I want'])[0], 5) ))
```

```
# evaluate model
I want: i want to be a little too
real     34m18.256s
user     232m59.932s
sys 15m23.629s
```

資料來源：www.manning.com/books/deep-learning-with-python
github.com/fchollet/deep-learning-with-python-notebooks

做更多實驗：用 pre-trained 詞向量 (1)

- 下載 GloVe 詞向量 nlp.stanford.edu/projects/glove/
- 先放到矩陣 embedding_matrix

Listing

```
# Download glove.6B.100d.txt from https://nlp.stanford.edu/projects/glove/

embeddings_index = {}
f = open(os.path.join('./', 'glove.6B.100d.txt'))
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
print('Found %s word vectors.' % len(embeddings_index))

max_words = vocab_size
word_index = tokenizer.word_index
embedding_dim = 100
embedding_matrix = np.zeros((max_words, embedding_dim))
for word, i in word_index.items():
    if i < max_words:
        embedding_vector = embeddings_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector
```

資料來源：www.manning.com/books/deep-learning-with-python
github.com/fchollet/deep-learning-with-python-notebooks

做更多實驗：用 pre-trained 詞向量 (2)

- 定義模型後，將詞向量載入詞內嵌層 model.layers[0].set_weights([embedding_matrix])
- 設定詞內嵌層為不訓練 model.layers[0].trainable = False

Listing 3

```
model = Sequential()
model.add(Embedding(vocab_size, embedding_dim, input_length=max_length-1))
model.add(Flatten())
model.add(Dense(embedding_dim, activation='relu'))
model.add(Dense(vocab_size, activation='softmax'))
print(model.summary())

model.layers[0].set_weights([embedding_matrix])
model.layers[0].trainable = False
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 2, 100)            777400
_________________________________________________________________
flatten_1 (Flatten)          (None, 200)               0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               20100
_________________________________________________________________
dense_2 (Dense)              (None, 7774)              785174
=================================================================
Total params: 1,582,674
Trainable params: 805,274
Non-trainable params: 777,400
```

資料來源：www.manning.com/books/deep-learning-with-python
github.com/fchollet/deep-learning-with-python-notebooks

7.5.3.2 類神經語言模型 (3)

- 編輯模型
- 訓練模型

Listing

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=10, verbose=2)
```

```
Epoch 1/20 - 3s - loss: 6.3520 - acc: 0.0695
Epoch 2/20 - 2s - loss: 5.5241 - acc: 0.0942
Epoch 3/20 - 3s - loss: 5.1070 - acc: 0.1102
Epoch 4/20 - 3s - loss: 4.7313 - acc: 0.1258
Epoch 5/20 - 3s - loss: 4.3445 - acc: 0.1394
Epoch 6/20 - 3s - loss: 3.9495 - acc: 0.1649
Epoch 7/20 - 3s - loss: 3.5903 - acc: 0.2100
Epoch 8/20 - 3s - loss: 3.3156 - acc: 0.2559
Epoch 9/20 - 3s - loss: 3.1262 - acc: 0.2854
Epoch 10/20 - 3s - loss: 2.9870 - acc: 0.3048
Epoch 11/20 - 3s - loss: 2.8823 - acc: 0.3158
Epoch 12/20 - 3s - loss: 2.7952 - acc: 0.3269
Epoch 13/20 - 3s - loss: 2.7243 - acc: 0.3373
Epoch 14/20 - 3s - loss: 2.6579 - acc: 0.3490
Epoch 15/20 - 3s - loss: 2.6045 - acc: 0.3532
Epoch 16/20 - 3s - loss: 2.5588 - acc: 0.3606
Epoch 17/20 - 3s - loss: 2.5105 - acc: 0.3695
Epoch 18/20 - 3s - loss: 2.4710 - acc: 0.3727
Epoch 19/20 - 3s - loss: 2.4318 - acc: 0.3792
Epoch 20/20 - 3s - loss: 2.3971 - acc: 0.3842
```

資料來源：www.manning.com/books/deep-learning-with-python
github.com/fchollet/deep-learning-with-python-notebooks

7.5.3.2 類神經語言模型 (4)

- 執行預測

Listing

```
def generate(model, x, n_words):
    res = x
    for _ in range(n_words):
        padded = pad_sequences([res], maxlen=2, padding='pre')
        res += [ model.predict_classes(padded)[0] ]
    return res

revIndex = dict( [ (value, key)
                   for key, value in tokenizer.word_index.items() ])

def view(index): return ' '.join( [revIndex.get(i, '?') for i in index])

print('# evaluate model\n’, ‘I want:', view( generate(model,
                              tokenizer.texts_to_sequences(['I want'])[0], 5) ))
```

```
# evaluate model
I want: i want to be the first of
```

資料來源：www.manning.com/books/deep-learning-with-python
github.com/fchollet/deep-learning-with-python-notebooks

小結

- 使用預先訓練好的詞內嵌的優點
  - 減少可訓練模型參數
  - 降低資料不足的衝擊
  - 降低訓練時間
  - 保有相似詞的推廣作用效果
- 缺點
  - 有時候預先訓練的詞內嵌會和任務的資料脫節
- 改進方法
  - 先凍結不訓練
  - 之後，再用任務相關的訓練資料小調整