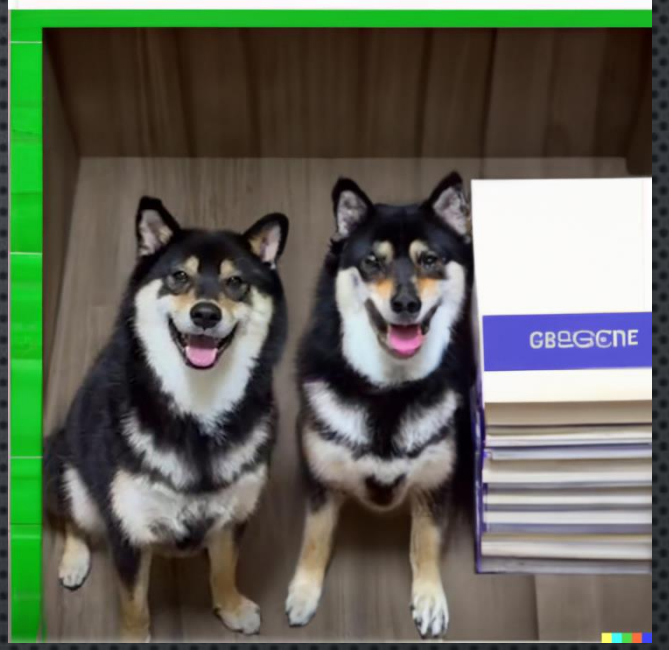
AI人工智慧—
訓練深度神經網路



繼續Keras神經網路模型
learning by doing

von anwendeng

Loss Function

- 機器學習大部分的演算法都有希望最大化/最小化一個函數/指標，這個函數被稱為「目標函數(Object function)」
- 人工智慧(大都深度學習)用到的目標函數是「損失函數(loss function)」，而模型的優劣部分的因素是損失函數的設計。

- 損失函數可以分成(分類和回歸)，基本上都是希望最小化損失函數。
- 回歸常用的損失函數: 均方誤差(Mean square error，MSE)
- 分類問題常用的損失函數: 交叉熵(cross-entropy)

y表示實際值，ŷ表示預測值=>Loss越小越好

- 均方誤差 (MSE) 損失函數
- $Loss = \frac{1}{n}\sum_{i=0}^{n-1}\left(y_i - \hat{y}_i\right)^2$

```
[1]  1  print('Loss function')
```

```
Loss function
```

```
     1  import numpy as np
     2  import matplotlib.pyplot as plt
```

```
[3]  1  def MSE(y, yH):
     2    n=y.shape[0]
     3    return sum((y-yH)**2)/n
```

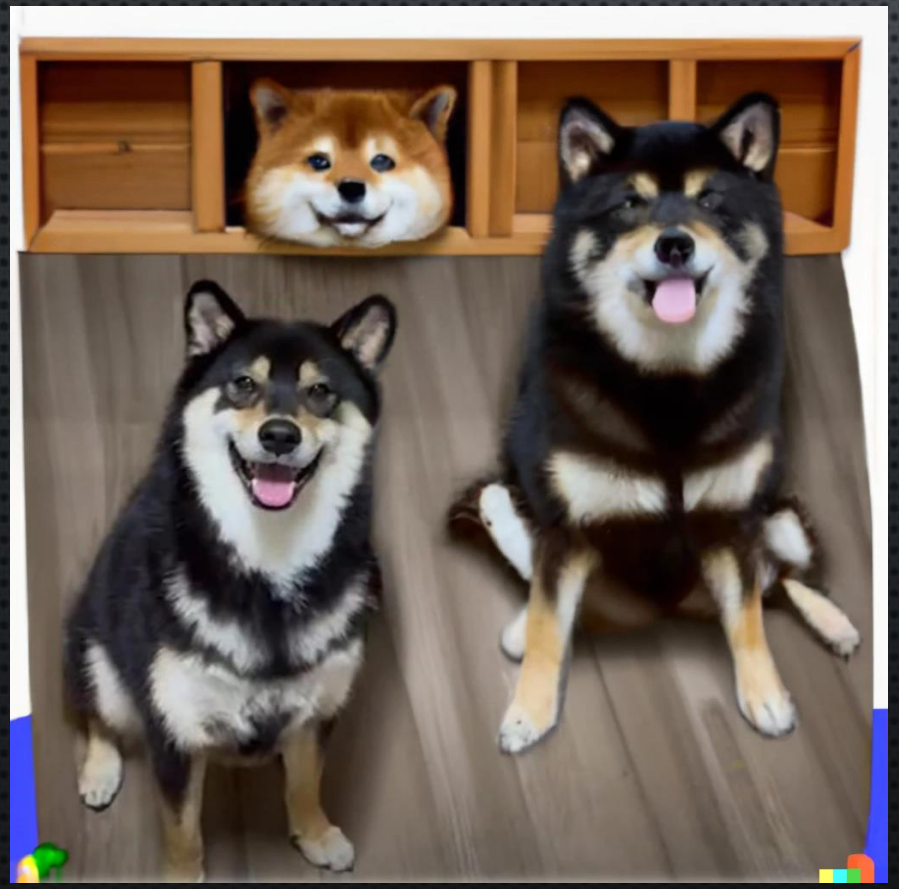
```
[4]  1  y=np.array([1.9, 2.8, 3.7])
     2  yH=np.array([2, 3, 4])
     3  y.shape[0]
```

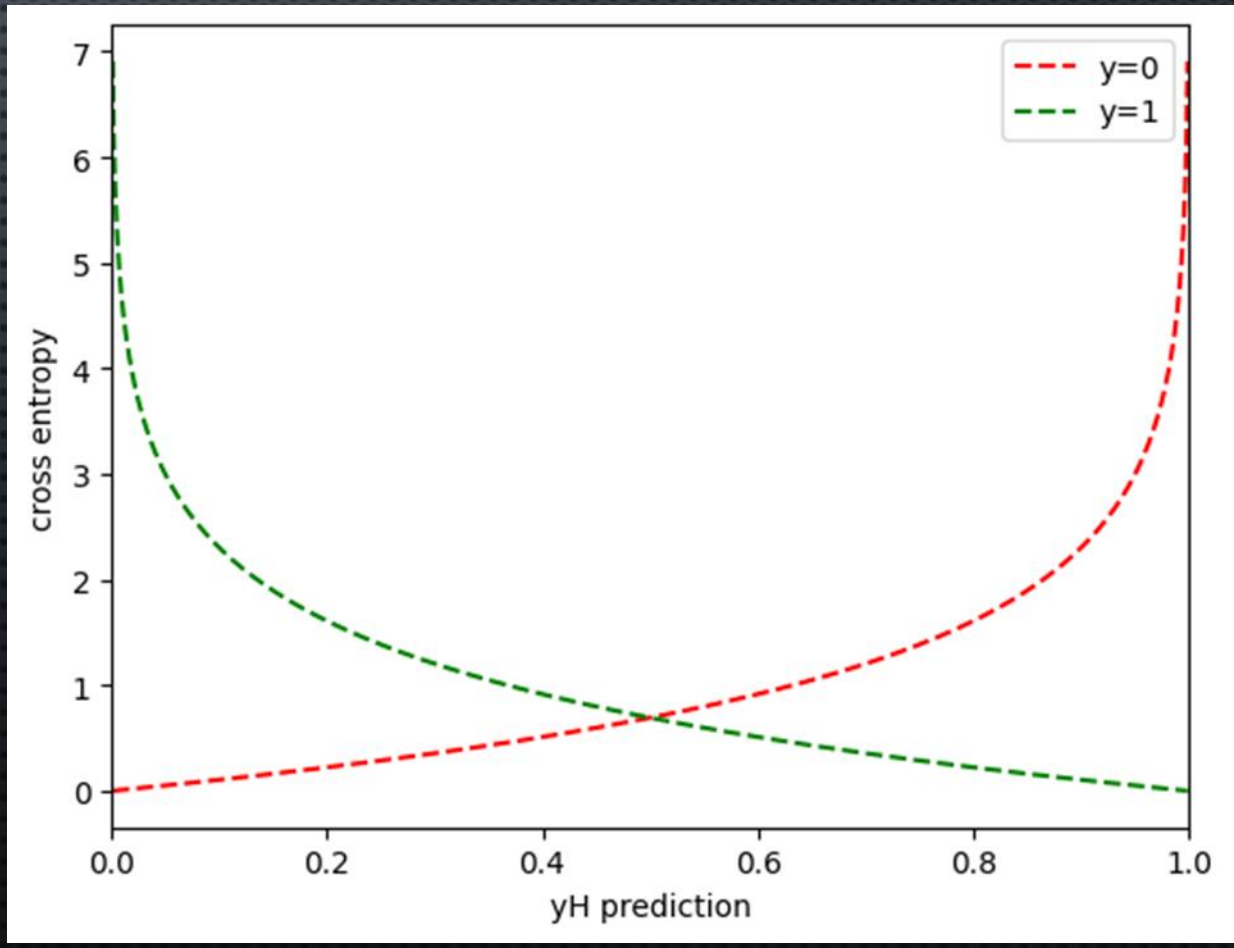
```
3
```

```
1  MSE(y, yH)
```

```
0.04666666666666666
```

交叉熵 (Cross Entropy)
損失函數





Entropy原本是物理上的概念，從Shannon導入此概念至Information Theory，Entropy是資料的亂度或是資料的不確定性。複習一下條件機率。

令 A 與 B 為事件,且 $p(B) > 0$, A 在條件 B 成立下之條件機率定義為

$$p(A|B) = \frac{p(A \cap B)}{p(B)}。$$

兩事件 A 與 B 稱為獨立事件(Independent Events)

$$\iff \quad p(A \cap B) = p(A)p(B)。$$

Bayes定理

若 A 與 B 為事件,且 $p(A) > 0$ 與 $p(B) > 0$,則

$$p(B)p(A|B) = p(A)p(B|A)。$$

熵,Entropy

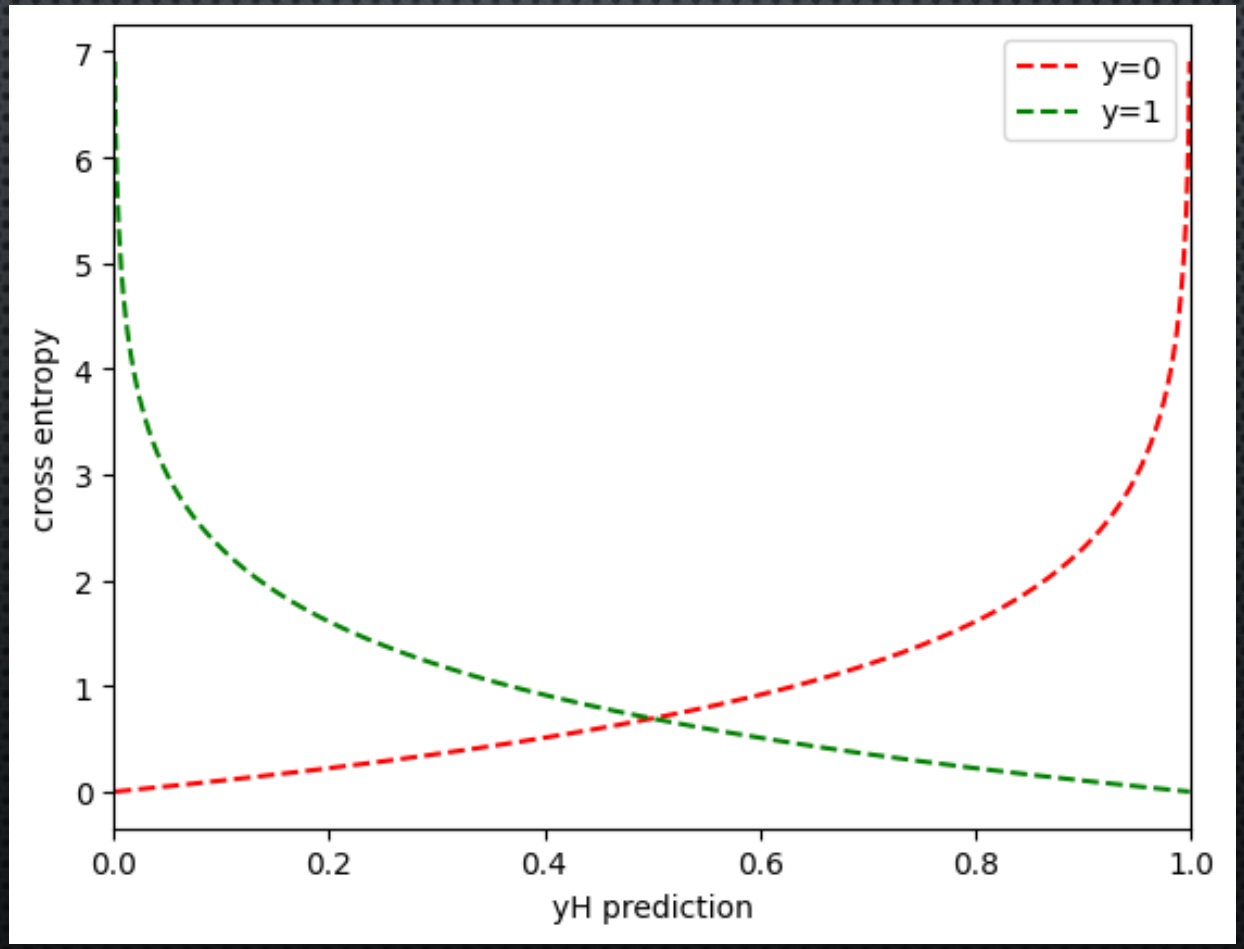
令 A 為樣本空間, X 為定義在樣本空間上之隨機變數,則隨機變數 X 之熵定義為

$$H(X) = -\sum_{x \in A} p(X = x) \log_2 (p(X = x))。$$

Binary cross entropy

$$C = -\frac{1}{n}\sum_{i=1}^{n}[\boldsymbol{y}_i \ln \hat{\boldsymbol{y}}_i + (1-\boldsymbol{y}_i)\ln(1-\hat{\boldsymbol{y}}_i)] \qquad \text{(公式 8.2)}$$

cross entropy適用於分類，是因為預測錯誤「懲罰」大，可參考下圖



```
[6]  1  print('cross entropy')
```

```
cross entropy
```

```
1  #請注意 log(0)數學上無定義，在計算上會分母為零的, yH=0 or 1會出問題
2  def x_entropy(y, yH):
3    return -1*(y*np.log(yH)+(1-y)*np.log(1-yH))
```

```
[8]  1  x_entropy(1, 0.99)
```

```
0.010050335853501451
```

```
1   y = np.linspace(0,1,12)
2   epsilon =1e-6
3   yH = np.asarray([0,1,0,0,1,1,1,0,1,1])
4   yH= np.abs(yH-epsilon)
5
6   log_loss = x_entropy(y[1:-1], yH)
7
8   print('y :', y[1:-1])
9   print('yH :', yH)
10
11  print("binary cross entropy:", log_loss)
```

ndarray: log_loss

ndarray with shape (10,)

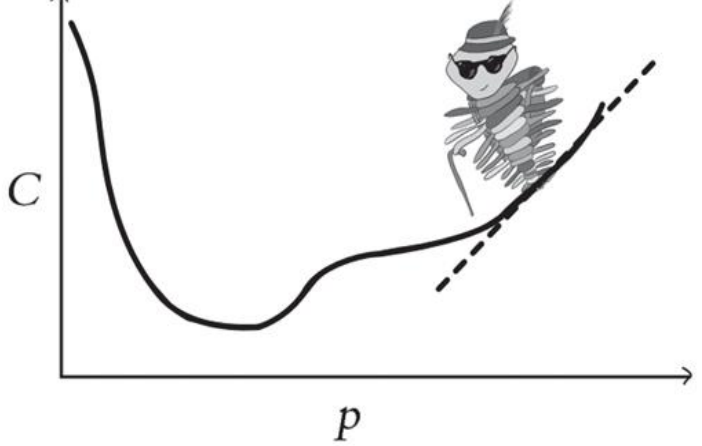
```
y : [0.09090909 0.18181818 0.27272727 0.36363636 0.45454545 0.54545455
 0.63636364 0.72727273 0.81818182 0.90909091]
yH : [1.00000e-06 9.99999e-01 1.00000e-06 1.00000e-06 9.99999e-01 9.99999e-01
 9.99999e-01 1.00000e-06 9.99999e-01 9.99999e-01]
binary cross entropy: [ 1.25595641 11.30359973  3.76786724  5.02382266  7.53573349  6.27977807
  5.02382266 10.04764431  2.51191183  1.25595641]
```

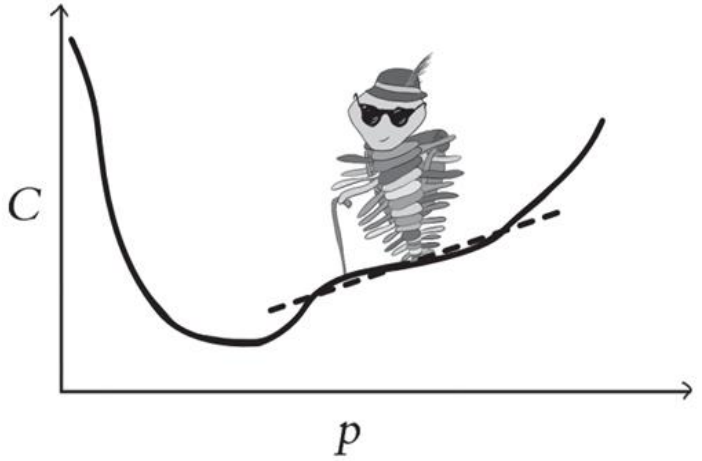
[10]
```
yH=np.linspace(0,1, 1002)
yH=yH[1:-1]
log_loss0=x_entropy(0, yH)
log_loss1=x_entropy(1, yH)
```

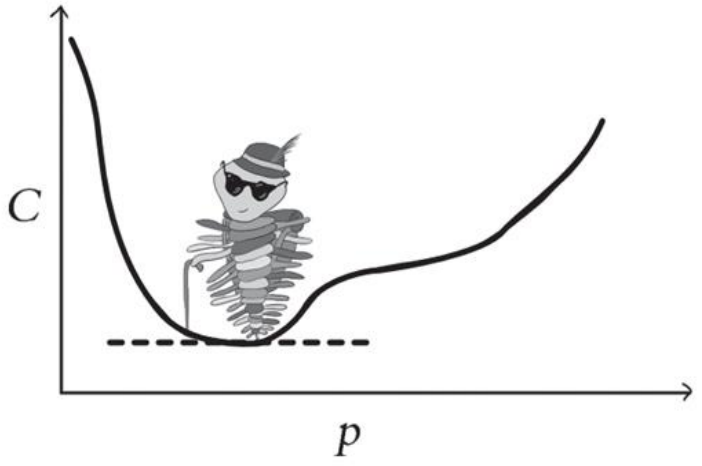
```
plt.plot(yH, log_loss0, 'r--',label="y=0" )
plt.plot(yH, log_loss1, 'g--',label="y=1" )
plt.xlim(0, 1)
plt.ylabel("cross entropy")
plt.xlabel("yH prediction")
plt.legend(loc="best")
plt.savefig('x_entropy.jpg')
plt.show()
```

藉由訓練讓誤差值最小化

---gradient descent

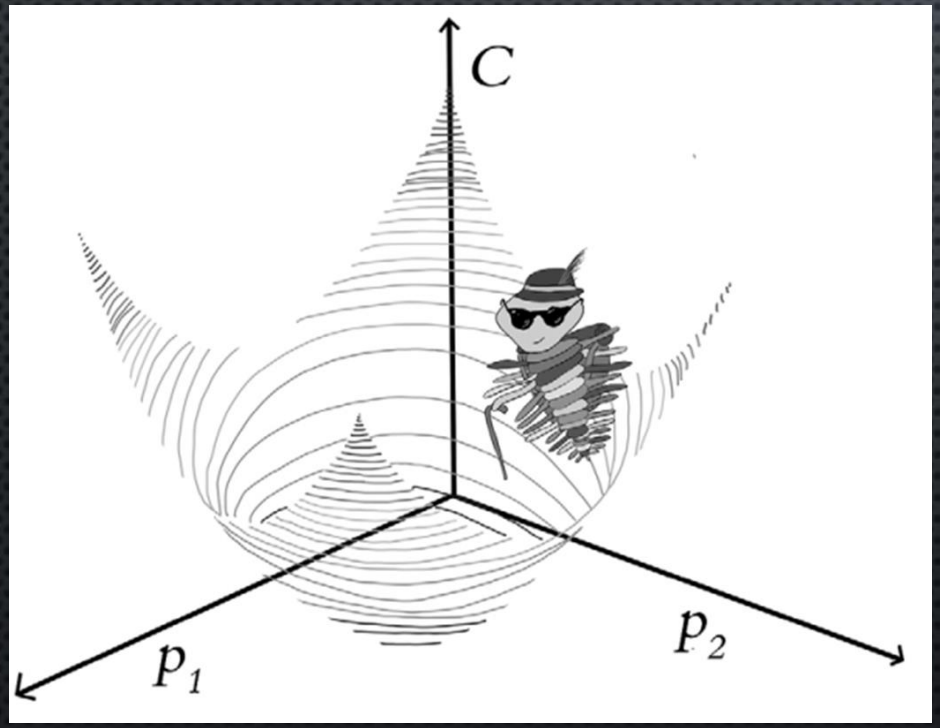


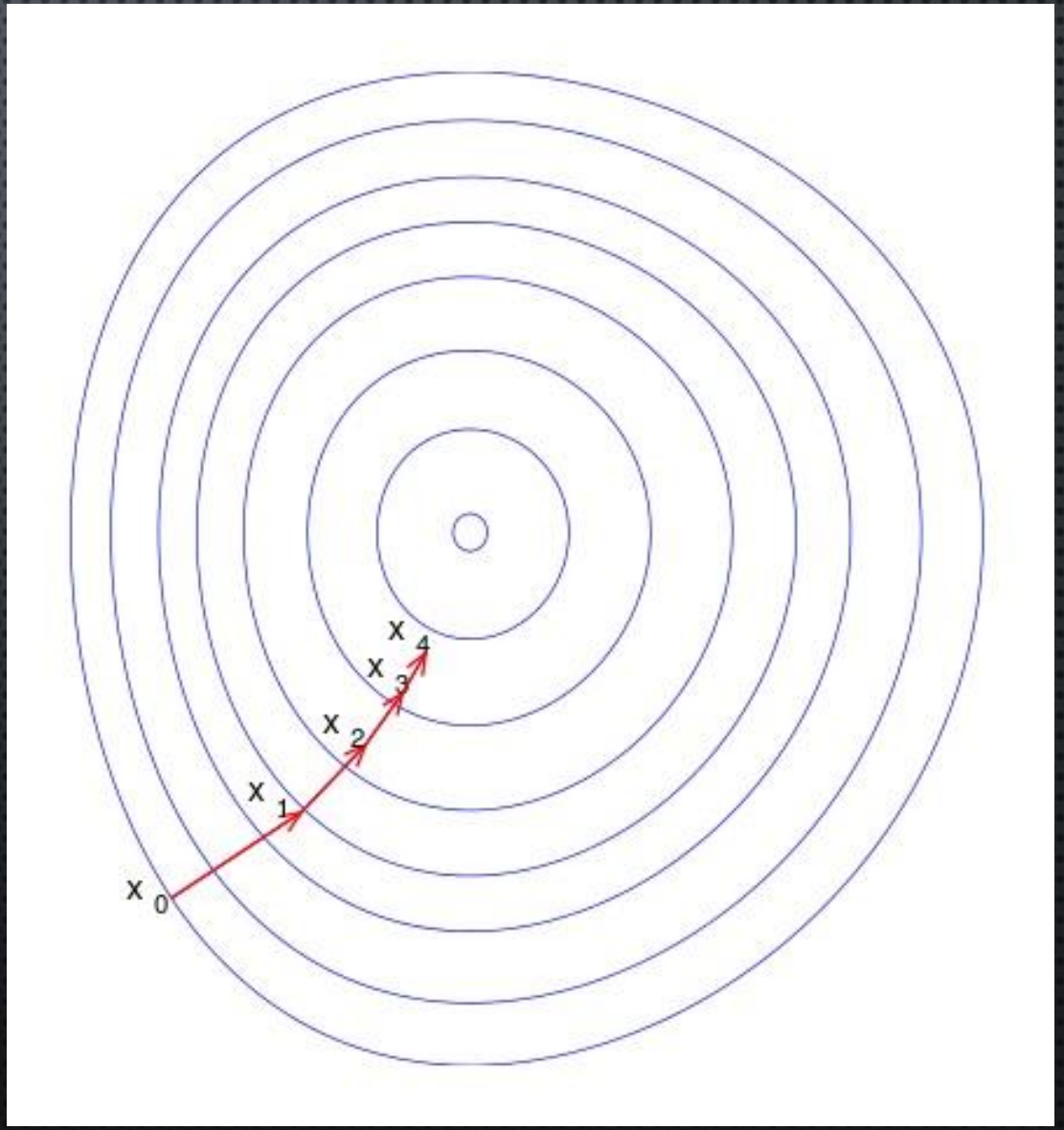




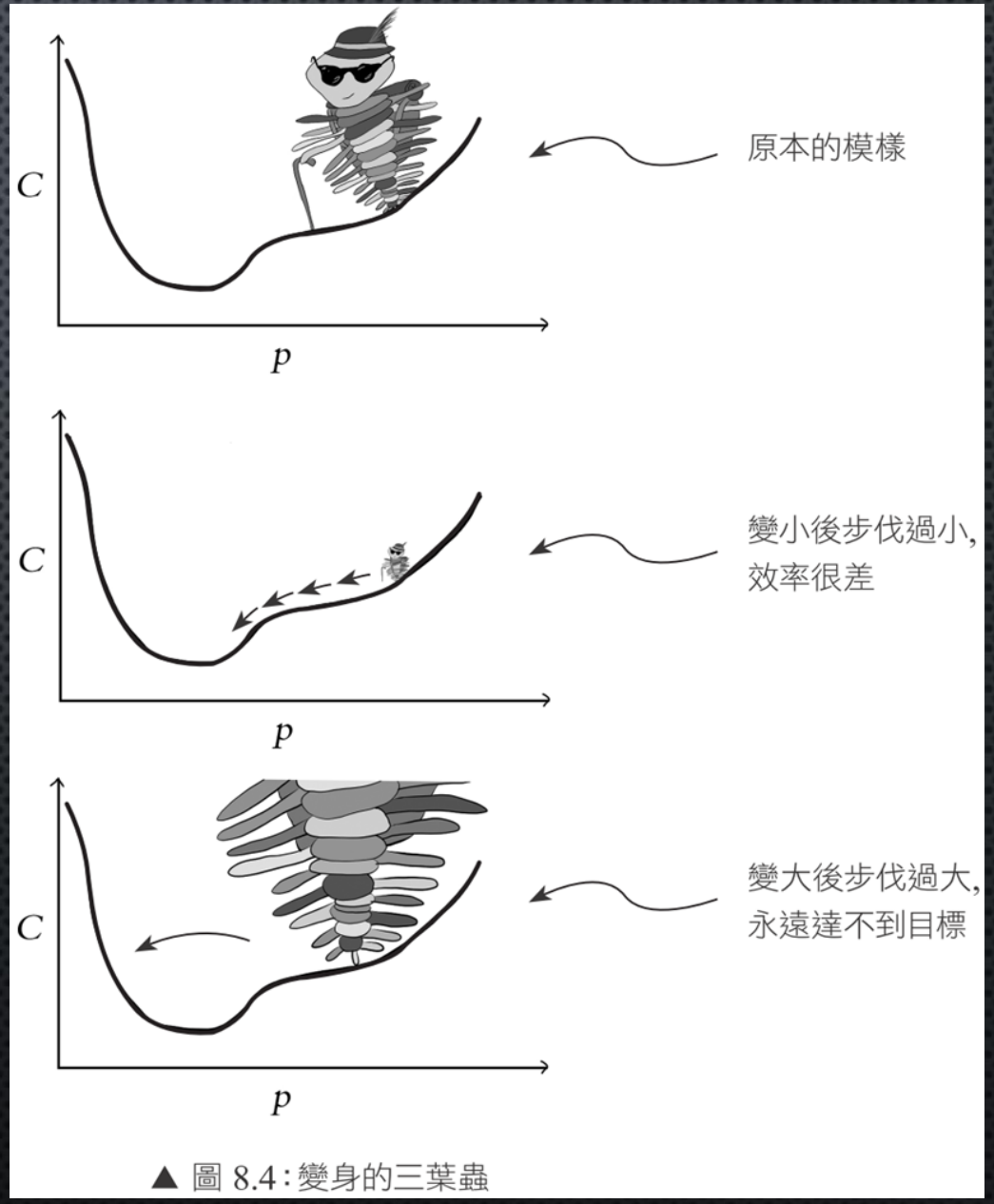
$$x_n = x_{n-1} - \eta \nabla L(x_{n-1})$$

- Gradient Descent，就是對loss function做偏微分（切線斜率）就是找極大極小值的概念，找一組參數讓loss function越小越好


C
p_1
p_2

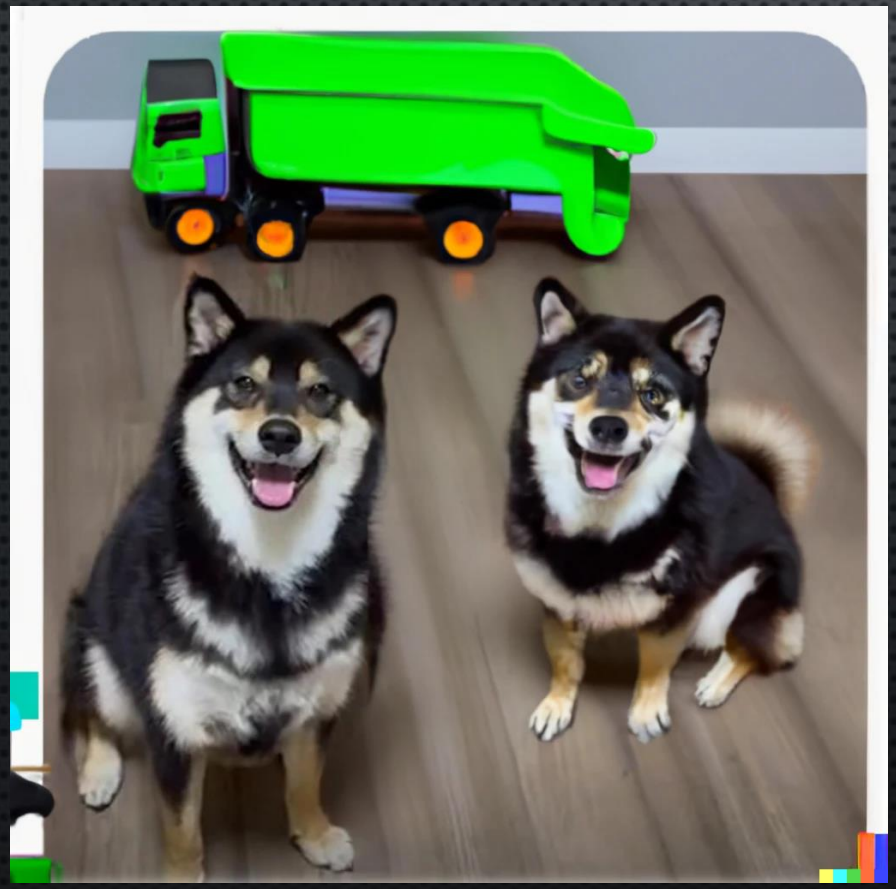

x 0
x 1
x 2
x 3
x 4

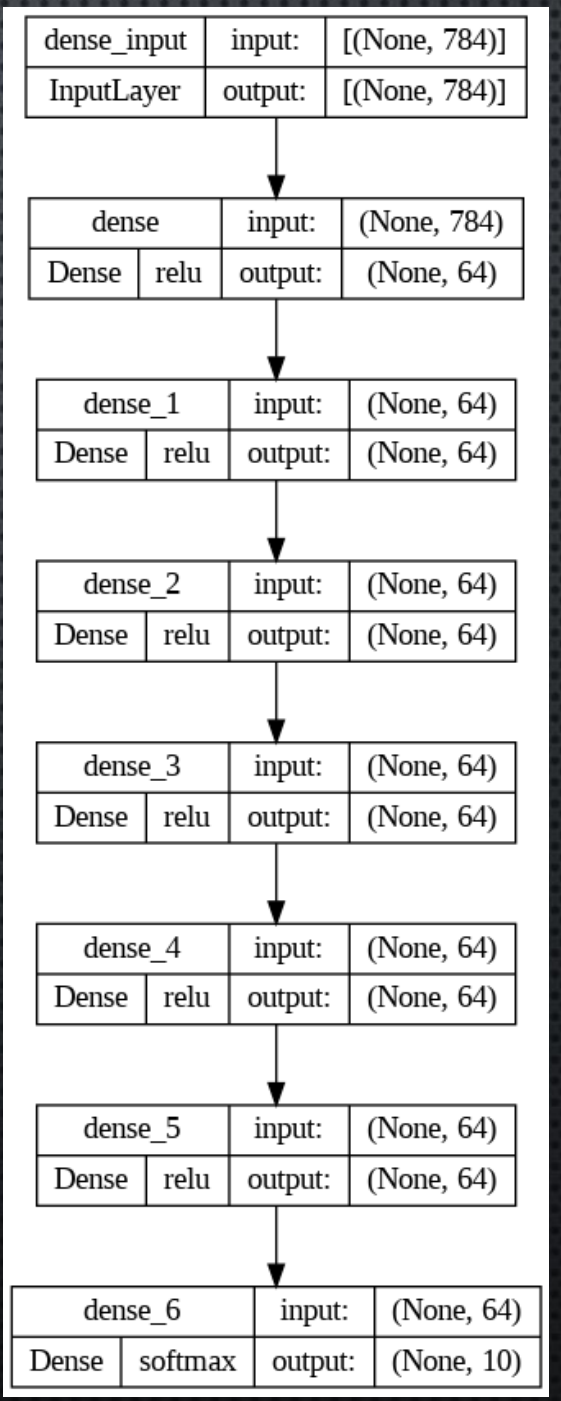
η=學習率
Learning rate



▲ 圖 8.4：變身的三葉蟲

SGD，反向傳播，建更複雜的神經網路





隨機梯度下降法 (SGD)

```
print('Loss  function修改')
model.compile(
        loss='categorical_crossentropy',
        optimizer=optimizers.SGD(learning_rate=0.01),   #隨機梯度下降法
        metrics=['accuracy']
)
```

SGD 的概念就是將訓練資料分成小批次量 (minibatch) 分批處理

- 隨機順序處理小批次量
- 每次小批次量修正Weight Matrix
-

```
history=model.fit(
        X_train,  y_train,  batch_size=128,  epochs=30,
        verbose=1,
        validation_data=(X_test,  y_test)
)
```

SGD pseudo code

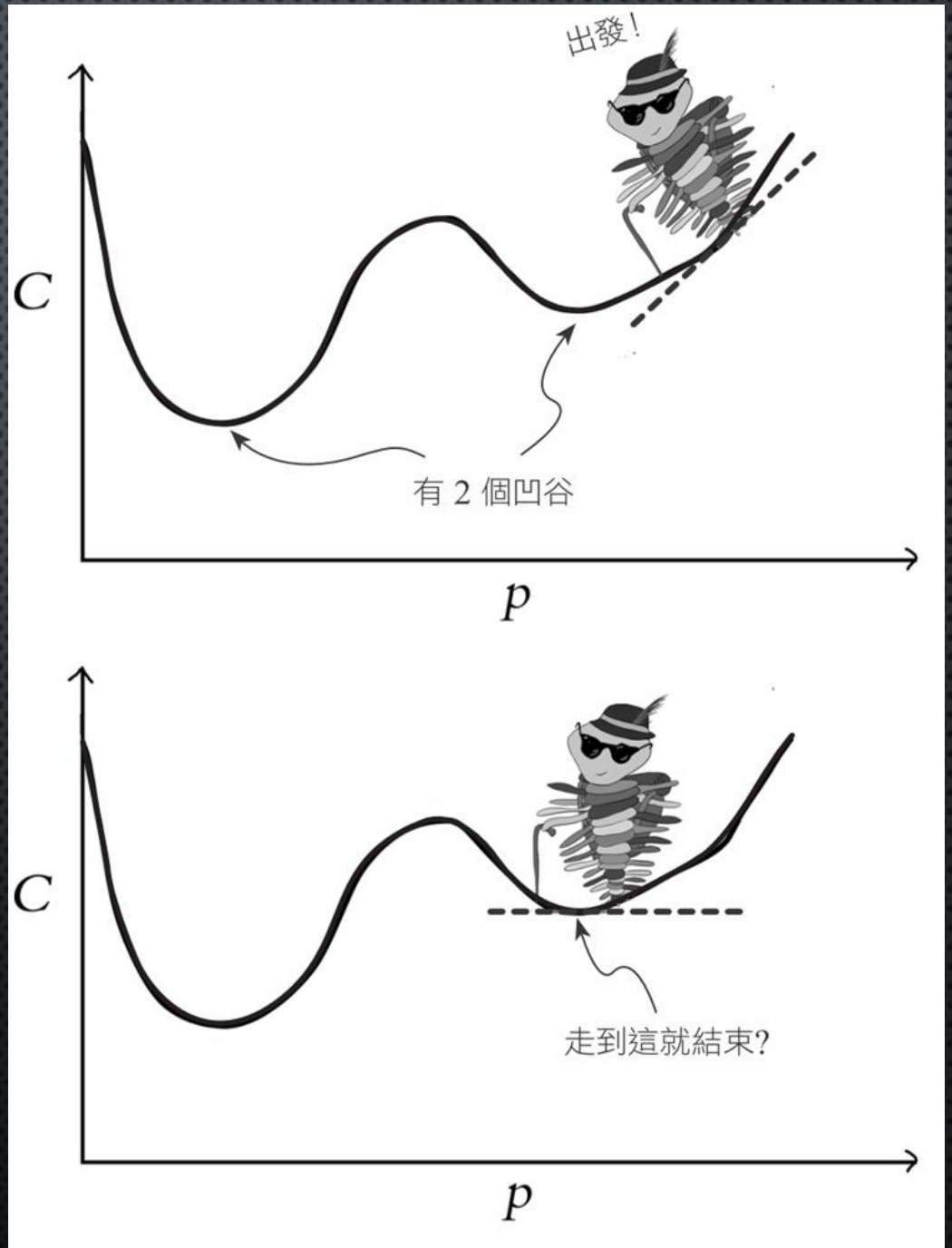
```
stochastic_gradient_descent(X, y, learning_rate=0.01, epochs=30, batch_size=128):
    initialize weights (including bias)
    for epoch in range(epochs):
        shuffle data (X, y)
        for i in range(0, len(X), batch_size):
            X_batch, y_batch = get_next_batch(X, y, i, batch_size)
            predictions = predict(X_batch, weights)
            errors = predictions - y_batch
            gradient = compute_gradient(X_batch, errors)
            update weights using gradient and learning_rate
    return weights
```

batch-size 設定

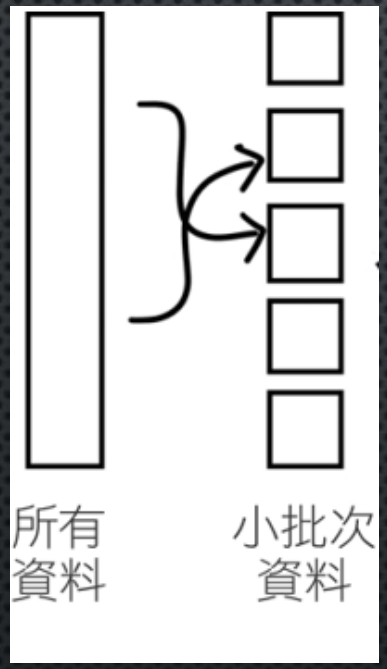
- $\left\lceil \frac{60000}{128} \right\rceil = \lceil 468.75 \rceil = 469$

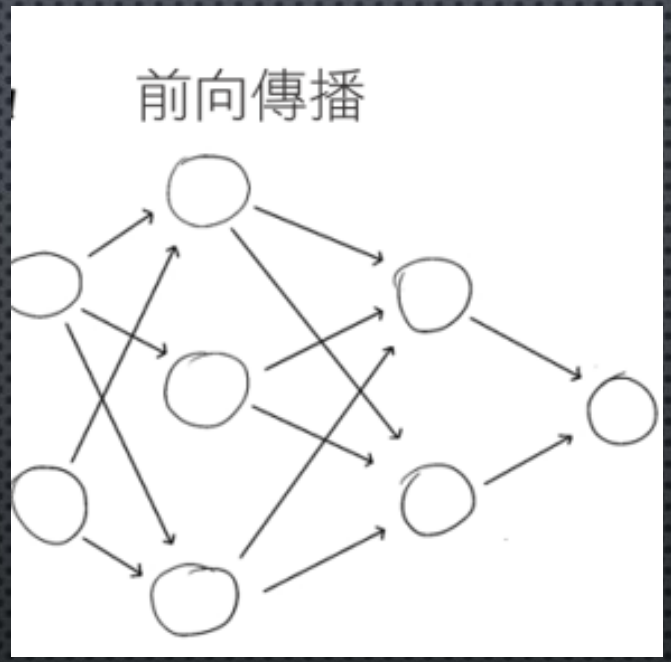
脫離局部最小值 (local minimum)

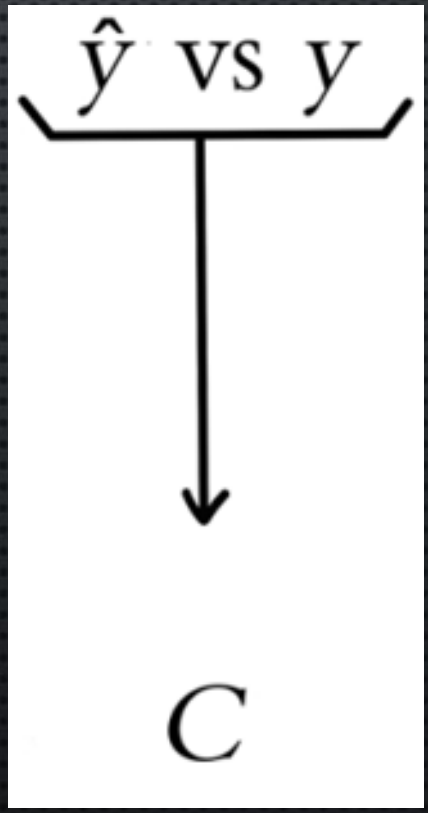
Adagrad、RMSprop、Momentum and Adam



back propagation







while (!condition):

$$x_n = x_{n-1} - \eta \nabla C(x_{n-1})$$

- Gradient Descent，就是對loss function C做偏微分（切線斜率）

Chain Rule & Backpropagation

取自 https://towardsdatascience.com/understanding-backpropagation-algorithm-7bb3aa2f95fd

For a single weight *(w_jk)^l,* the gradient is:

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}\frac{\partial z_j^l}{\partial w_{jk}^l} \qquad chain\ rule$$

$$z_j^l = \sum_{k=1}^{m} w_{jk}^l a_k^{l-1} + b_j^l \qquad by\ definition$$

$$m\ -\ number\ of\ neurons\ in\ l-1\ layer$$

$$\frac{\partial z_j^l}{\partial w_{jk}^l} = a_k^{l-1} \qquad by\ differentiation\ (calculating\ derivative)$$

$$\frac{\partial C}{\partial w_{jk}^l} = \frac{\partial C}{\partial z_j^l}a_k^{l-1} \qquad final\ value$$

$$\frac{\partial C}{\partial x} = [\frac{\partial C}{\partial x_1}, \frac{\partial C}{\partial x_2}, \cdots, \frac{\partial C}{\partial x_m}]$$

隱藏層要設幾層？2~4

- 梯度消失問題（Vanishing gradient problem）出現在以梯度下降法和反向傳播訓練AI神經網路的時候。在每次訓練的迭代中，神經網路權重的更新值與誤差函數的偏導數成比例，然而在某些情況下，梯度值會幾乎消失，使得權重無法得到有效更新，甚至神經網路可能完全無法繼續訓練。

多層神經網路模型V2

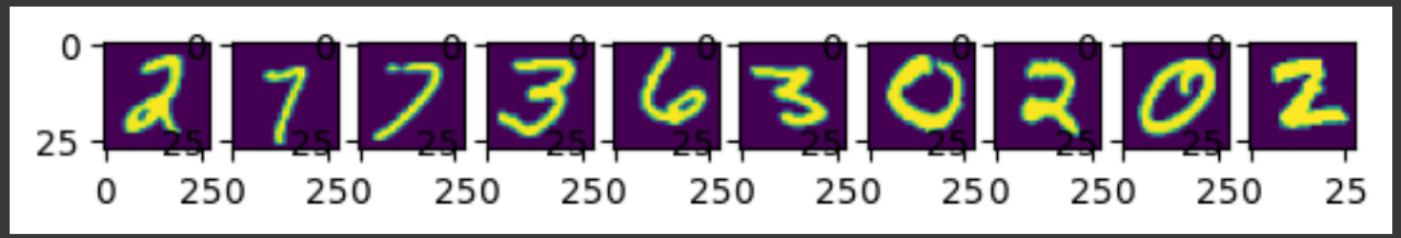
```
print('h層Dense')
h=int(input('多層Dense,  h='))
def  construct_model(activation='relu',  hidden=h):
    model=Sequential()
    model.add(Dense(64,  activation='relu',  input_shape=(784,  )))
    for  _  in  range(h-1):
        model.add(Dense(64,  activation='relu'))
    model.add(Dense(10,  activation='softmax'))
    return  model
model=construct_model(hidden=h)  #  numbers  of  hidden  layers=h
```

```
1  model.evaluate(X_test,  y_test)
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.1185 - accuracy: 0.9638
[0.11852539330720901, 0.9638000130653381]
```

```
[27]  1  from  datetime  import  datetime
```

```
1  np.random.seed(int(datetime.now().timestamp()))
2  I=np.random.choice(y_test.shape[0],  10,  replace=False)
3  X_c=X_test[I]
4  y_c=y_test[I]
5  for  i  in  range(10):
6      plt.subplot(1,  10,  i+1)
7      plt.imshow(X_c[i].reshape((28,  28)))
8  plt.show()
9  prediction=np.argmax(model.predict(X_c),  axis=1)
10 print("predict:      ",prediction)
11 print("true  value:",np.argmax(y_c,  axis=1))
```



各層神經元的數量要設多少？2^n

- 若輸入資料的低階特徵（顯而易見的特徵）較多，模型前幾層放更多神經元可能會管用。
- 若特徵不明顯，則後面幾層多設點神經元會較好

- 損失函數、隨機梯度下降法（SGD）與反向傳播等訓練神經網路的關鍵概念,找出最佳的權重參數配置。
- 學習率、批次量、訓練週期等神經網路超參數，建構一個架構更複雜的神經網路

軟體實作

von anwendeng

感謝觀賞

Herzlichen Dank für die Aufmerksamkeit

von anwendeng