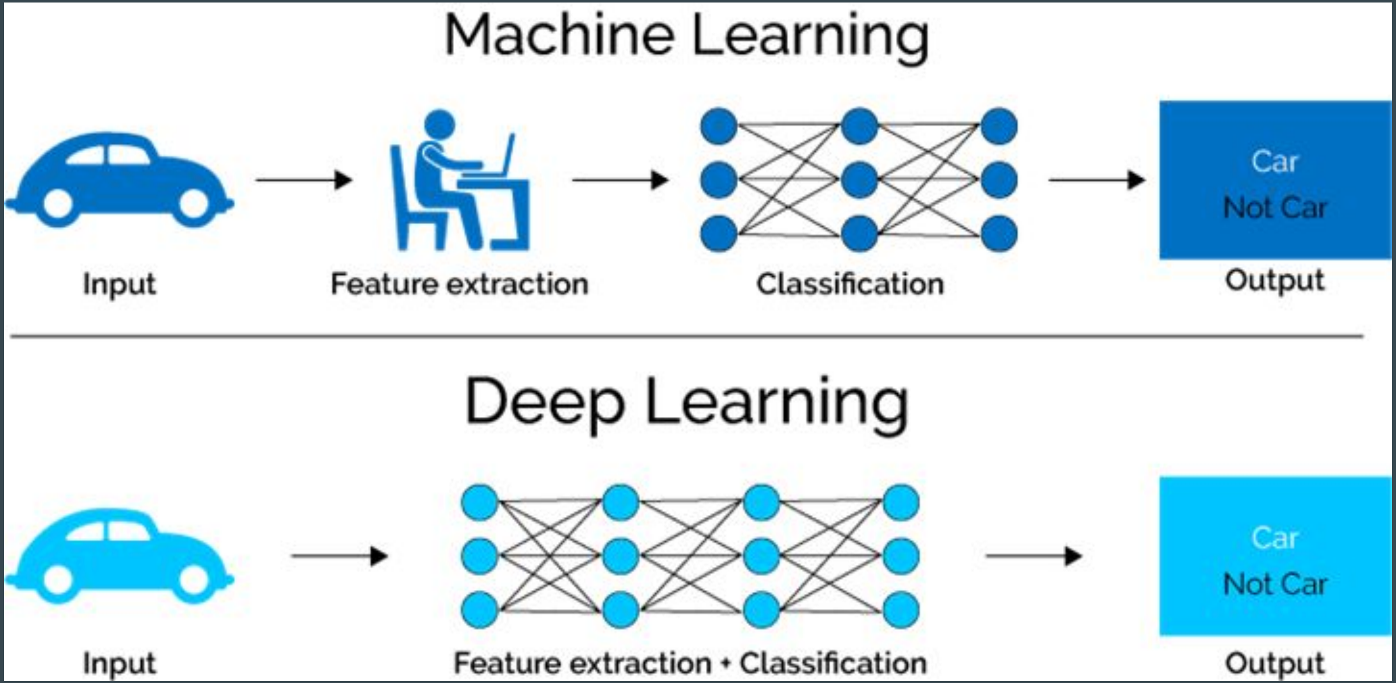
Ch04. Training Neural Networks

CK.D.Lee

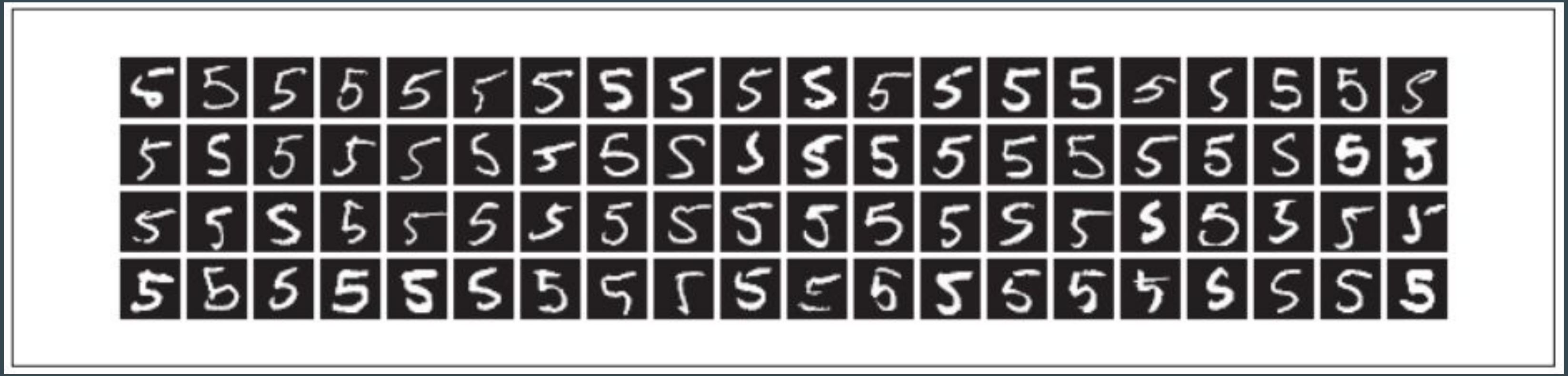
Learning from Data

Feature extraction is exhausting work. (Ch02. Perceptron only has three features.)

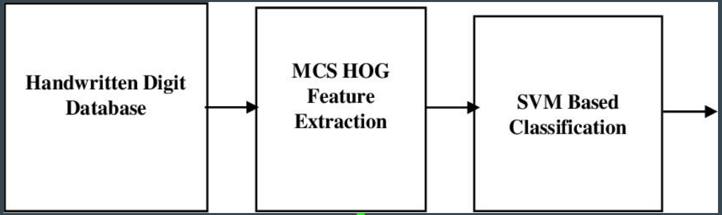


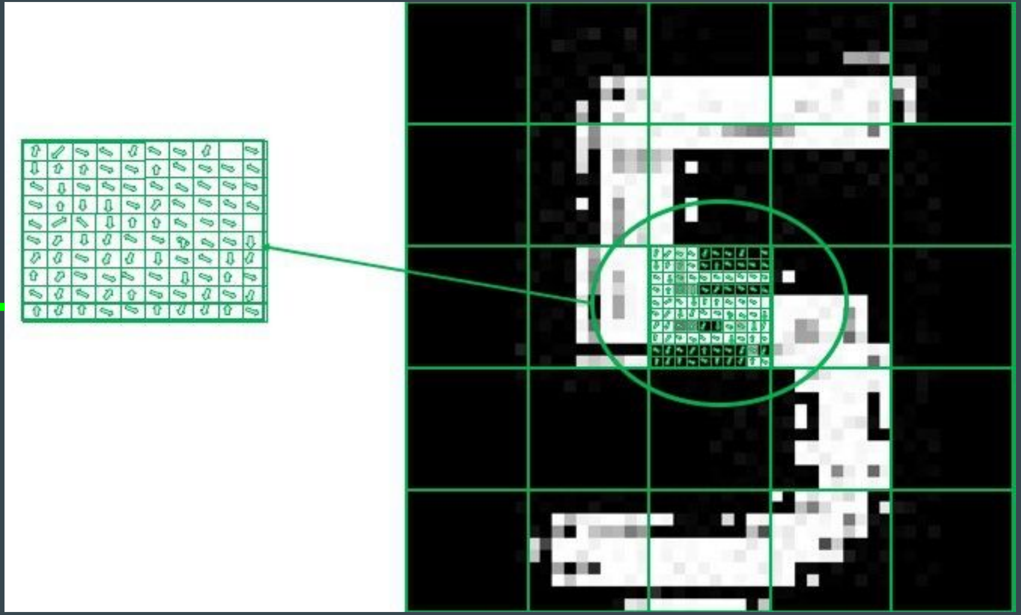
Data driven - How do we know this is "five"?

- Human
  - Instinct
- Machine
  - Feature
    - SIFT (Scale-invariant feature transform)
    - SURF (Speeded up robust features)
    - HOG (Histogram of oriented gradients)



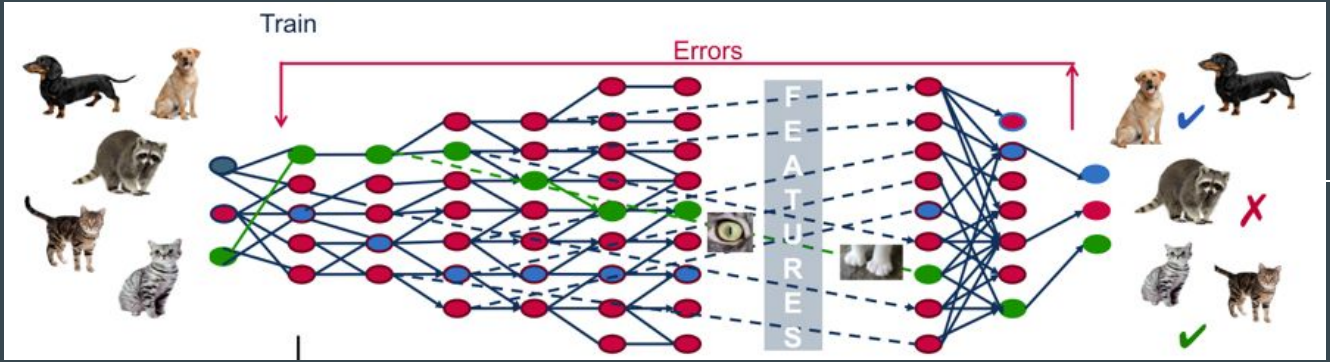
HOG based classification

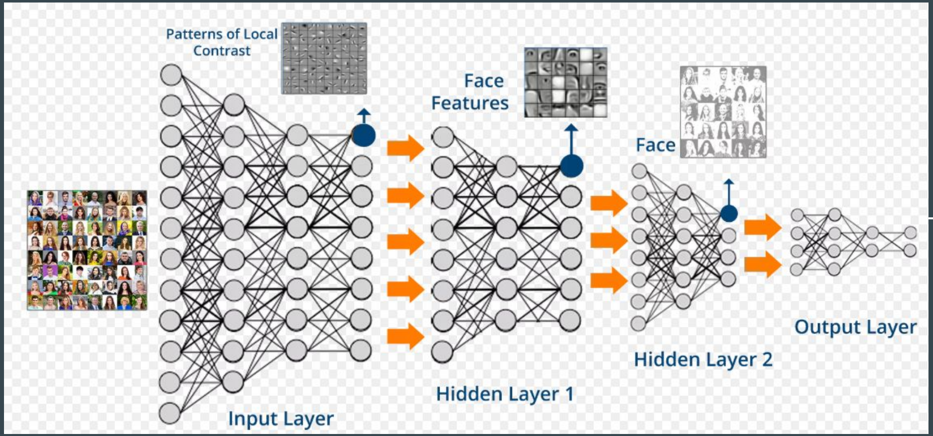




https://towardsdatascience.com/scanned-digits-recognition-using-k-nearest-neighbor-k-nn-d1a1528f0dea

Deep Learning





Animal v.s. Human
Similar.
Less work.
Good!

Training Data & Test Data

Why do we split data?

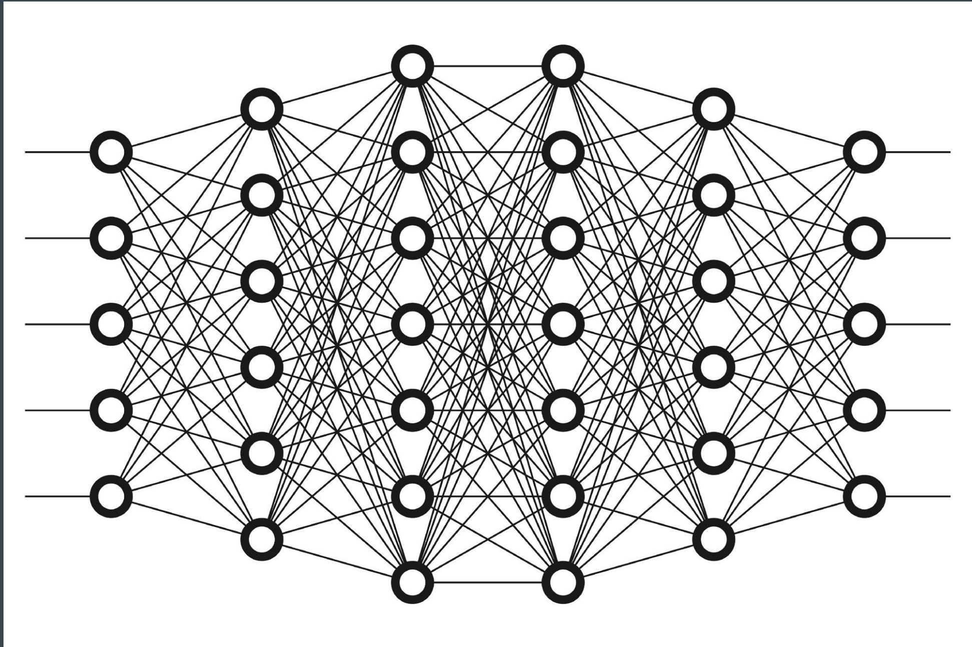
- To evaluate model's generalization capability

넌 얼마나 좋으니?

Loss function

Compare training class to output class to find the best model.

a loss function or cost function is a function that maps an event or values of one or more variables onto a real number intuitively representing some "cost" associated with the event.



Loss function cont.

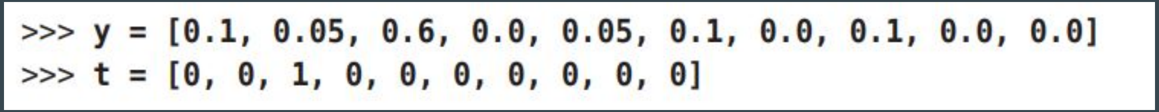
- Mean Squared error

$$E = \frac{1}{2}\sum_k (y_k - t_k)^2$$

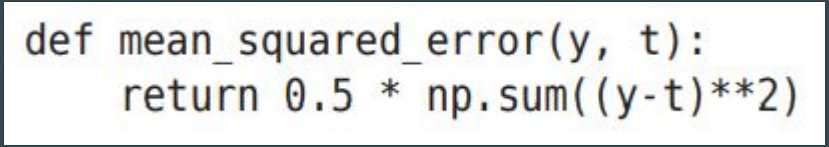
- Cross entropy error

$$E = -\sum_k t_k \log y_k$$

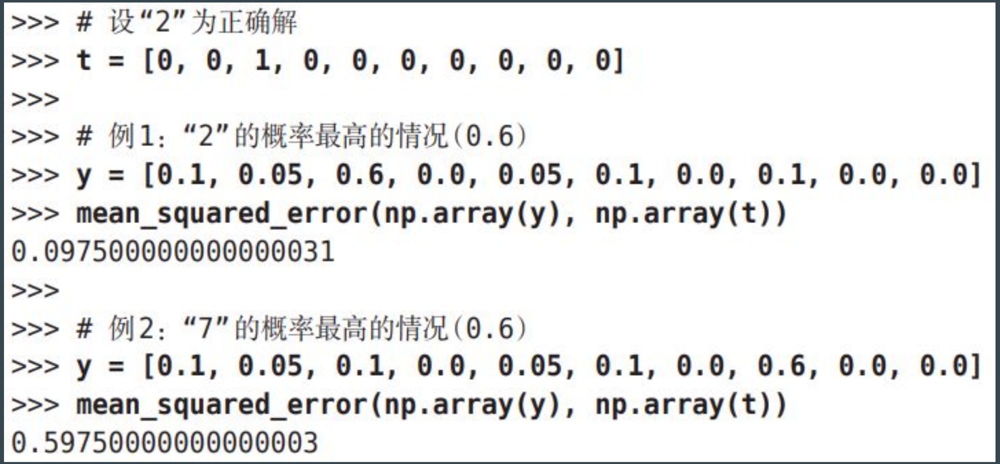
Loss function - Mean Squared error



```
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
```

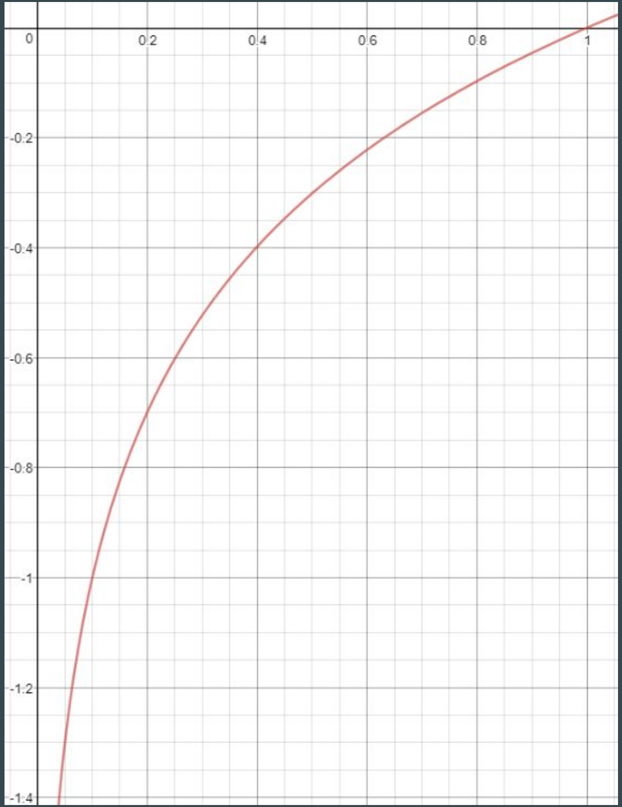


```
def mean_squared_error(y, t):
    return 0.5 * np.sum((y-t)**2)
```



```
>>> # 设“2”为正确解
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
>>>
>>> # 例1：“2”的概率最高的情况(0.6)
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
>>> mean_squared_error(np.array(y), np.array(t))
0.097500000000000031
>>>
>>> # 例2：“7”的概率最高的情况(0.6)
>>> y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
>>> mean_squared_error(np.array(y), np.array(t))
0.59750000000000003
```

Loss function - Cross Entropy error

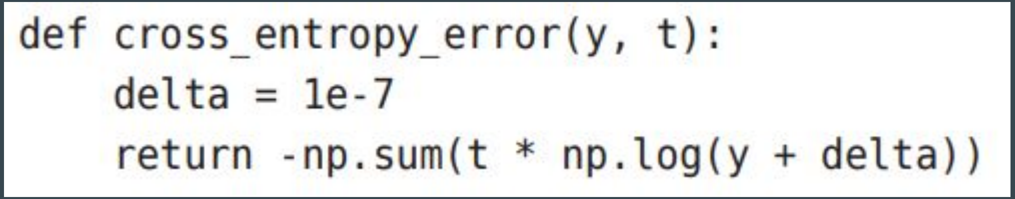


y = log(x)

$X \rightarrow 1$ --------> $Y \rightarrow 0$

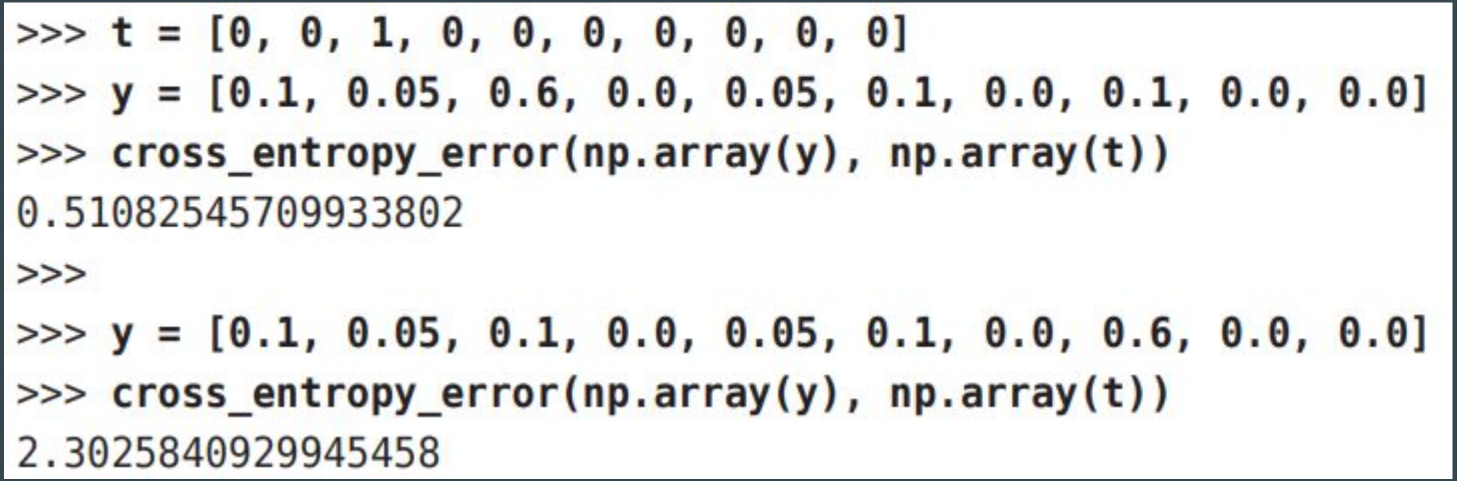
https://www.desmos.com/calculator

Loss function - Cross Entropy error



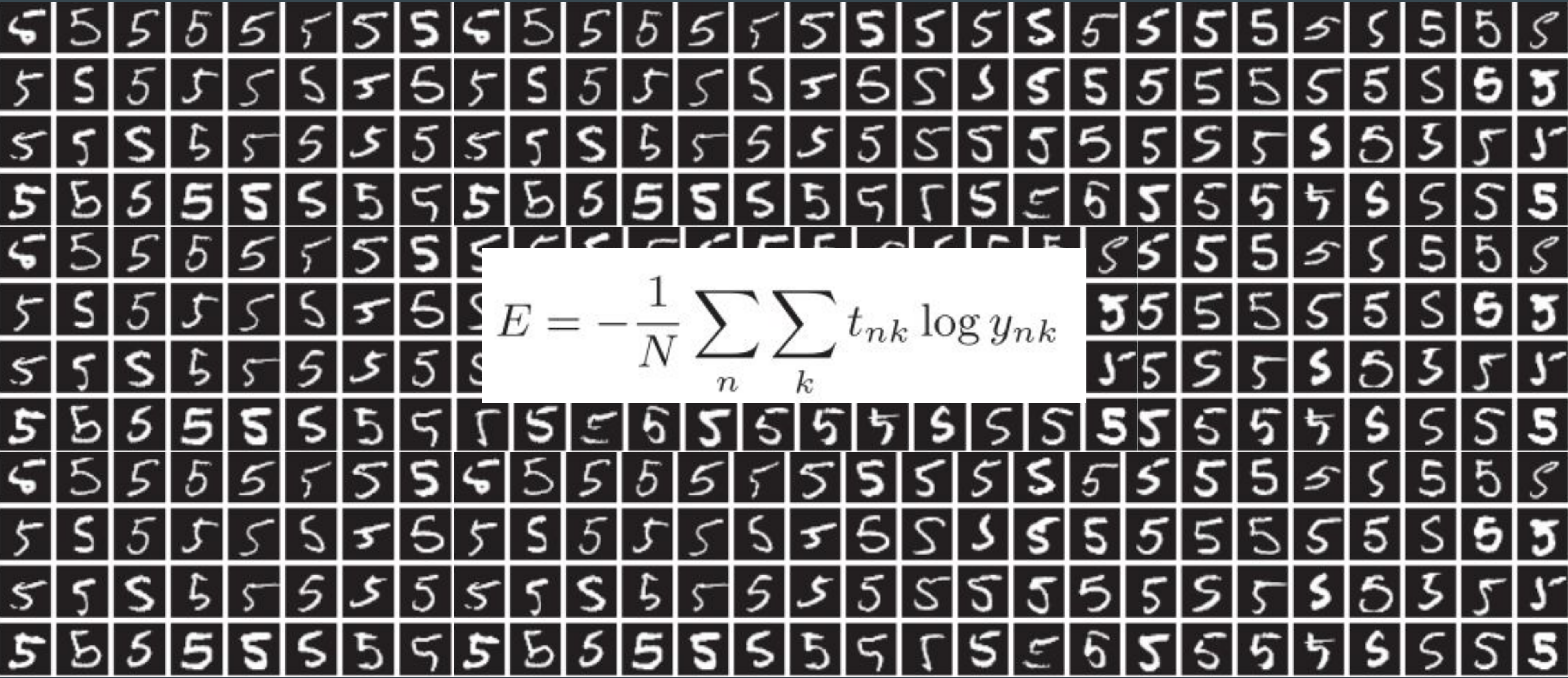
```
def cross_entropy_error(y, t):
    delta = 1e-7
    return -np.sum(t * np.log(y + delta))
```

delta is added to avoid -inf.



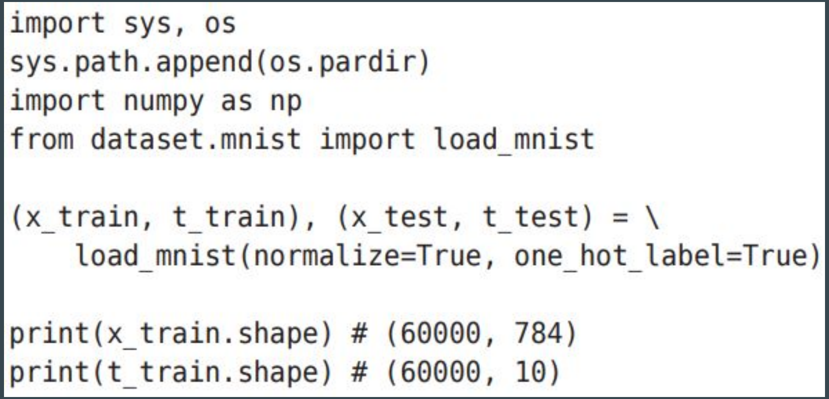
```
>>> t = [0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
>>> y = [0.1, 0.05, 0.6, 0.0, 0.05, 0.1, 0.0, 0.1, 0.0, 0.0]
>>> cross_entropy_error(np.array(y), np.array(t))
0.51082545709933802
>>>
>>> y = [0.1, 0.05, 0.1, 0.0, 0.05, 0.1, 0.0, 0.6, 0.0, 0.0]
>>> cross_entropy_error(np.array(y), np.array(t))
2.3025840929945458
```

mini-batch



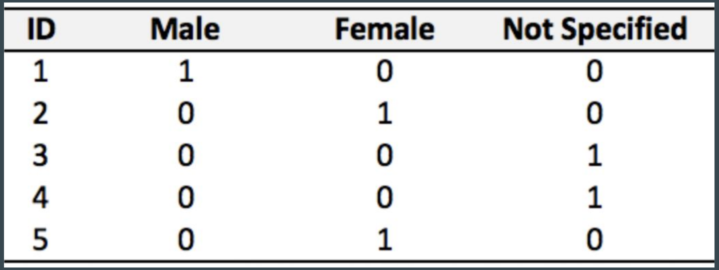
$$E = -\frac{1}{N}\sum_{n}\sum_{k} t_{nk} \log y_{nk}$$

Mini-batch cont.



```
import sys, os
sys.path.append(os.pardir)
import numpy as np
from dataset.mnist import load_mnist

(x_train, t_train), (x_test, t_test) = \
    load_mnist(normalize=True, one_hot_label=True)

print(x_train.shape) # (60000, 784)
print(t_train.shape) # (60000, 10)
```

Load MNIST data

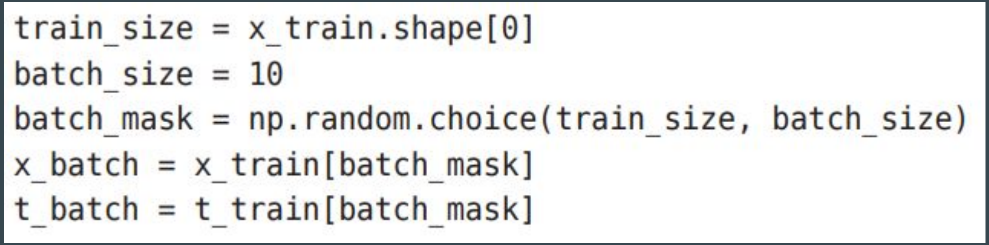


ID	Male	Female	Not Specified
1	1	0	0
2	0	1	0
3	0	0	1
4	0	0	1
5	0	1	0

One-hot label

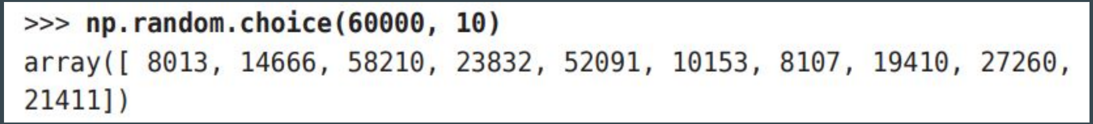
*（28 × 28）image data
x_train、t_ train -> (60000, 784) and (60000, 10)

Mini-batch cont.



```
train_size = x_train.shape[0]
batch_size = 10
batch_mask = np.random.choice(train_size, batch_size)
x_batch = x_train[batch_mask]
t_batch = t_train[batch_mask]
```

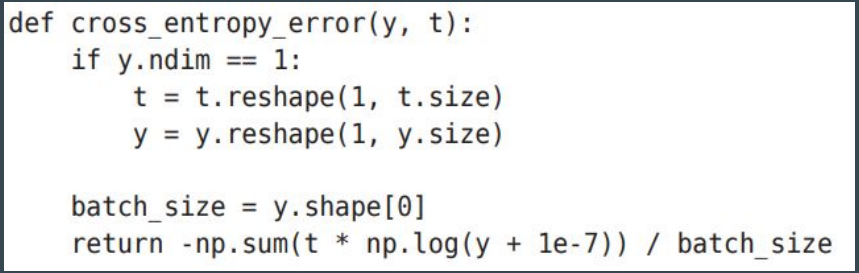
Choose 10 random data



```
>>> np.random.choice(60000, 10)
array([ 8013, 14666, 58210, 23832, 52091, 10153, 8107, 19410, 27260,
21411])
```

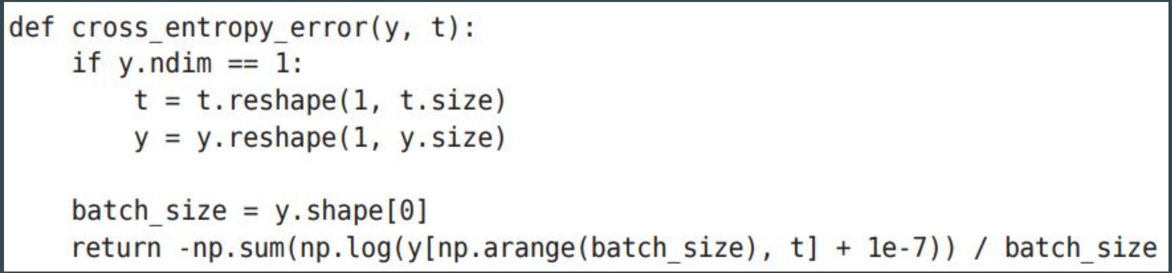
np.random.choice()

Mini-batch ver. Cross-Entropy error



```
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    batch_size = y.shape[0]
    return -np.sum(t * np.log(y + 1e-7)) / batch_size
```

One-hot labeled

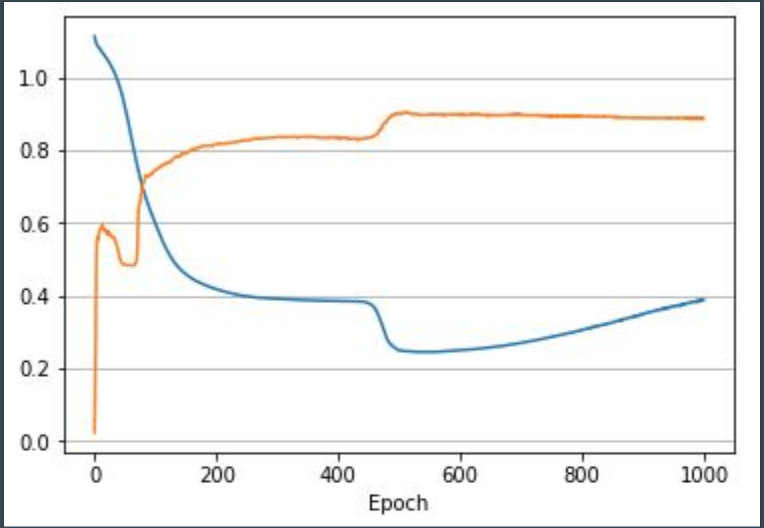


```
def cross_entropy_error(y, t):
    if y.ndim == 1:
        t = t.reshape(1, t.size)
        y = y.reshape(1, y.size)

    batch_size = y.shape[0]
    return -np.sum(np.log(y[np.arange(batch_size), t] + 1e-7)) / batch_size
```

None one-hot labeled

WHY loss-function not accuracy?

A loss function is used to optimize a machine learning algorithm. An accuracy metric is used to measure the algorithm's performance (accuracy) in an interpretable way.

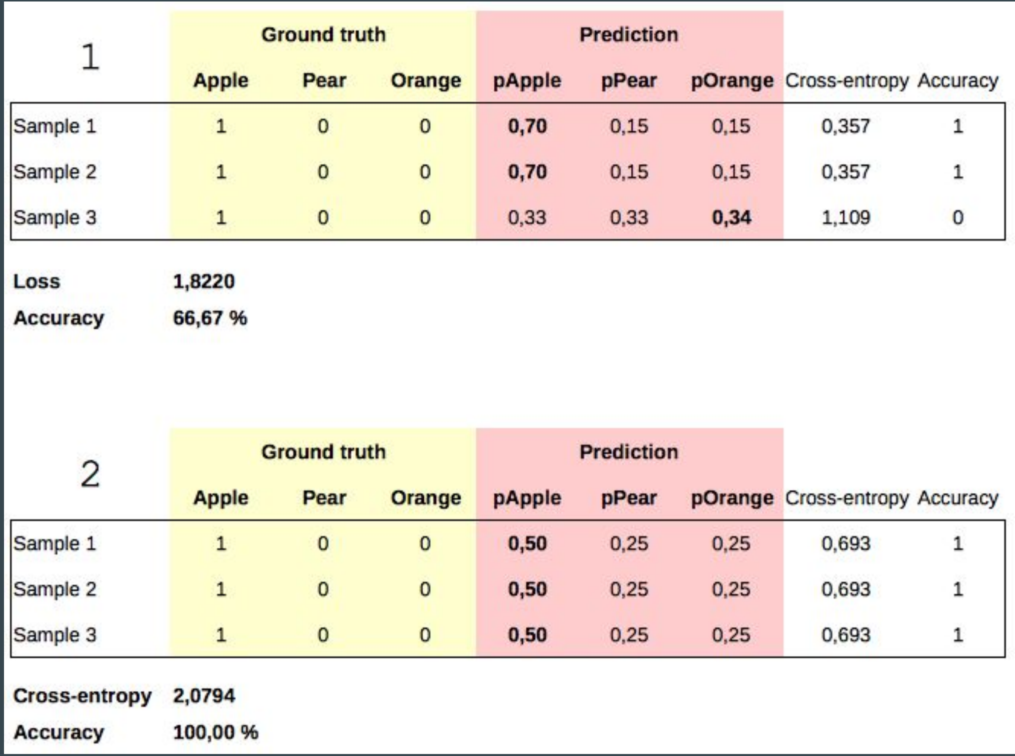


http://www.jussihuotari.com/2018/01/17/why-loss-and-accuracy-metrics-conflict/

WHY loss-function not accuracy? cont.

loss function here is categorical cross-entropy that is used to predict class probabilities.

The target values are one-hot encoded so the loss is the best when the model's output is very close to 1 for the right category and very close to 0 for other categories. The loss is a continuous variable.



1	Ground truth			Prediction				
	Apple	**Pear**	**Orange**	**pApple**	**pPear**	**pOrange**	Cross-entropy	Accuracy
Sample 1	1	0	0	**0,70**	0,15	0,15	0,357	1
Sample 2	1	0	0	**0,70**	0,15	0,15	0,357	1
Sample 3	1	0	0	0,33	0,33	**0,34**	1,109	0

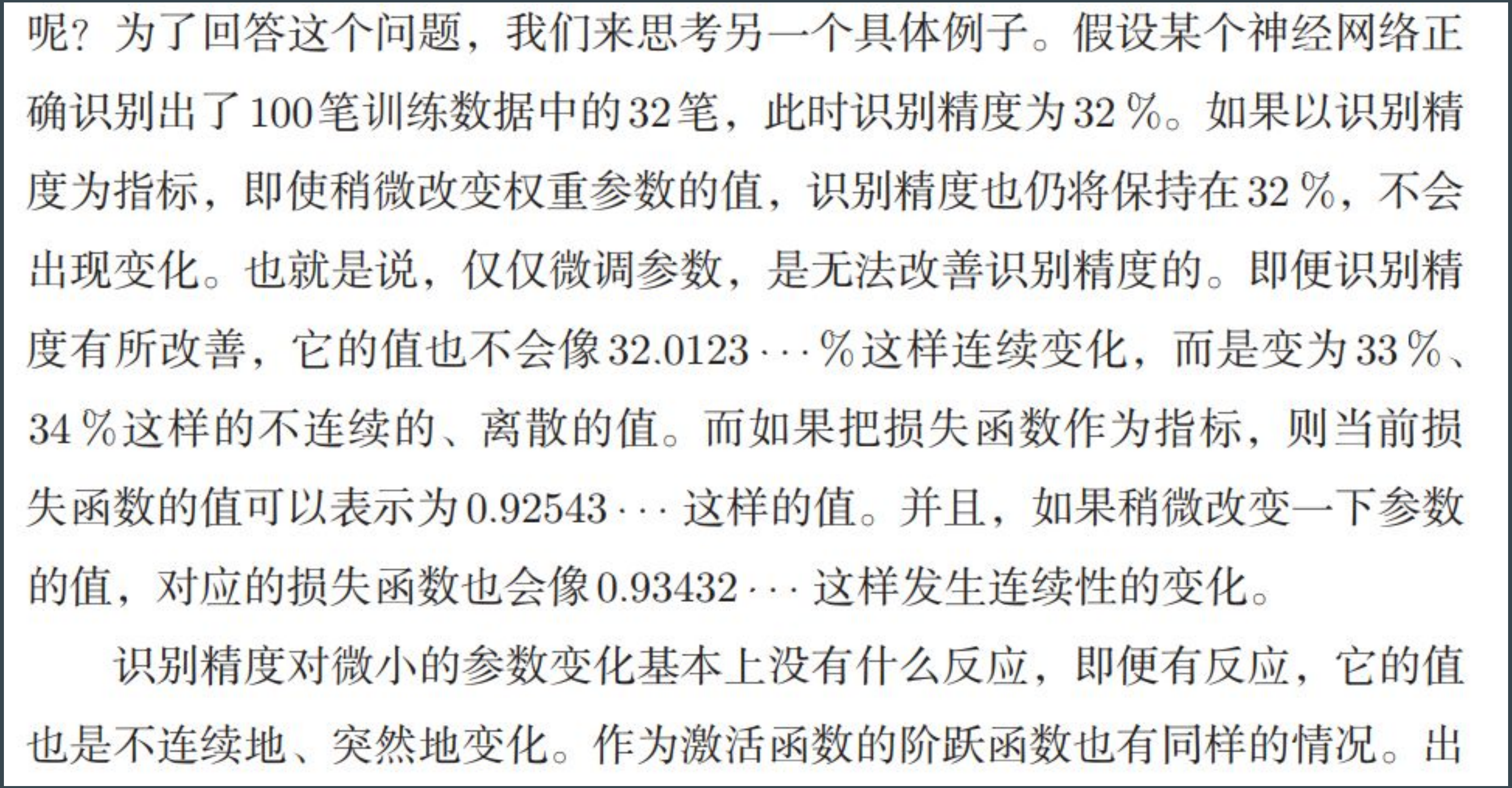
Loss	**1,8220**
Accuracy	**66,67 %**

2	Ground truth			Prediction				
	Apple	**Pear**	**Orange**	**pApple**	**pPear**	**pOrange**	Cross-entropy	Accuracy
Sample 1	1	0	0	**0,50**	0,25	0,25	0,693	1
Sample 2	1	0	0	**0,50**	0,25	0,25	0,693	1
Sample 3	1	0	0	**0,50**	0,25	0,25	0,693	1

Cross-entropy	**2,0794**
Accuracy	**100,00 %**

http://www.jussihuotari.com/2018/01/17/why-loss-and-accuracy-metrics-conflict/

WHY loss-function not accuracy? cont.



呢？为了回答这个问题，我们来思考另一个具体例子。假设某个神经网络正确识别出了100笔训练数据中的32笔，此时识别精度为32 %。如果以识别精度为指标，即使稍微改变权重参数的值，识别精度也仍将保持在32 %，不会出现变化。也就是说，仅仅微调参数，是无法改善识别精度的。即便识别精度有所改善，它的值也不会像32.0123 · · · %这样连续变化，而是变为33 %、34 %这样的不连续的、离散的值。而如果把损失函数作为指标，则当前损失函数的值可以表示为0.92543 · · · 这样的值。并且，如果稍微改变一下参数的值，对应的损失函数也会像0.93432 · · · 这样发生连续性的变化。

识别精度对微小的参数变化基本上没有什么反应，即便有反应，它的值也是不连续地、突然地变化。作为激活函数的阶跃函数也有同样的情况。出

Step function v.s. Sigmoid function

Sigmoid function will never have derivative function with 0

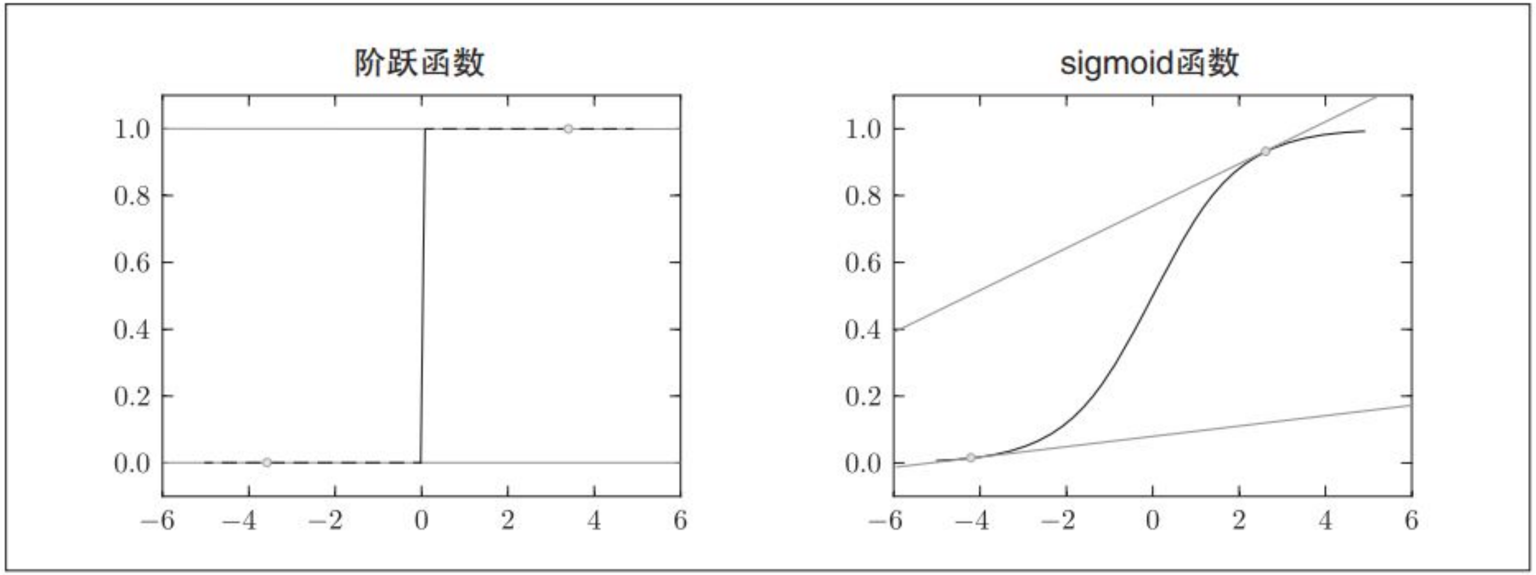
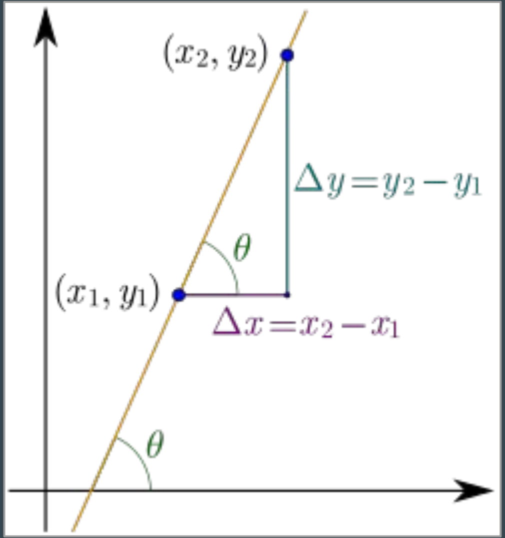


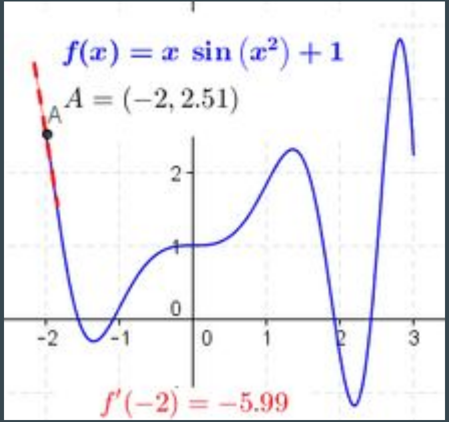
图4-4　阶跃函数和sigmoid函数：阶跃函数的斜率在绝大多数地方都为0，而sigmoid函数的斜率（切线）不会为0

Derivative (导数)

The derivative of a function of a real variable measures the sensitivity to change of the function value (output value) with respect to a change in its argument (input value)

$$m = \frac{\text{change in } y}{\text{change in } x} = \frac{\Delta y}{\Delta x} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

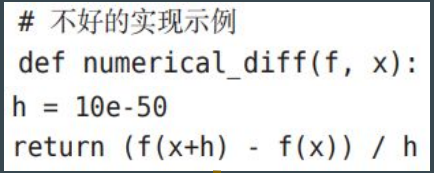




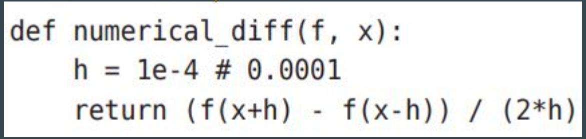
The derivative at different points of a differentiable function. In this case, the derivative is equal to: $\sin(x^2) + 2x^2 \cos(x^2)$

https://en.wikipedia.org/wiki/Derivative

Derivative cont.



```
# 不好的实现示例
def numerical_diff(f, x):
h = 10e-50
return (f(x+h) - f(x)) / h
```



```
def numerical_diff(f, x):
    h = 1e-4 # 0.0001
    return (f(x+h) - f(x-h)) / (2*h)
```

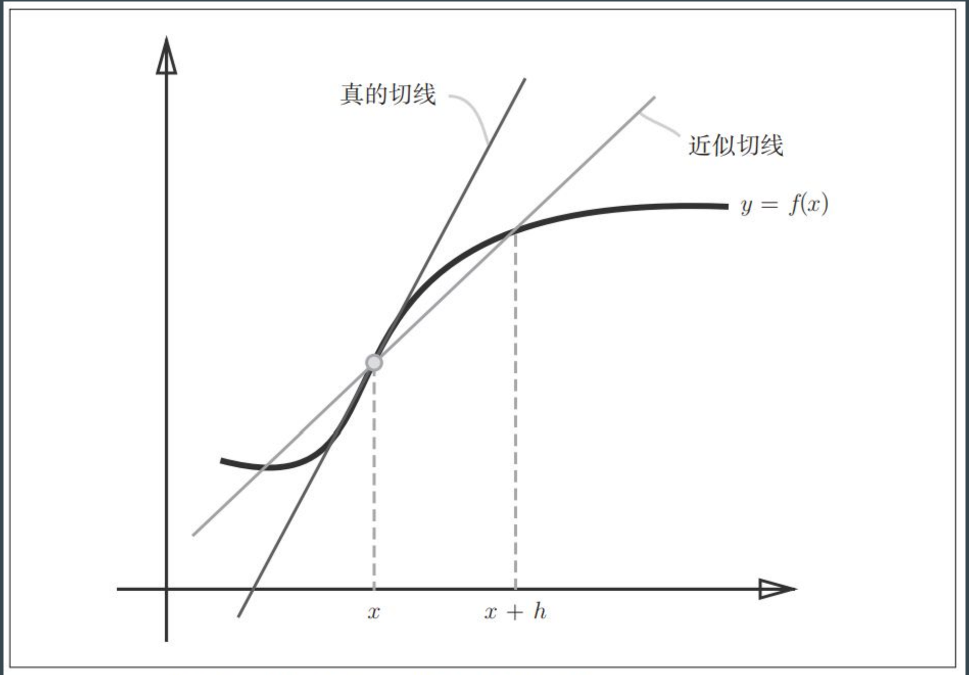
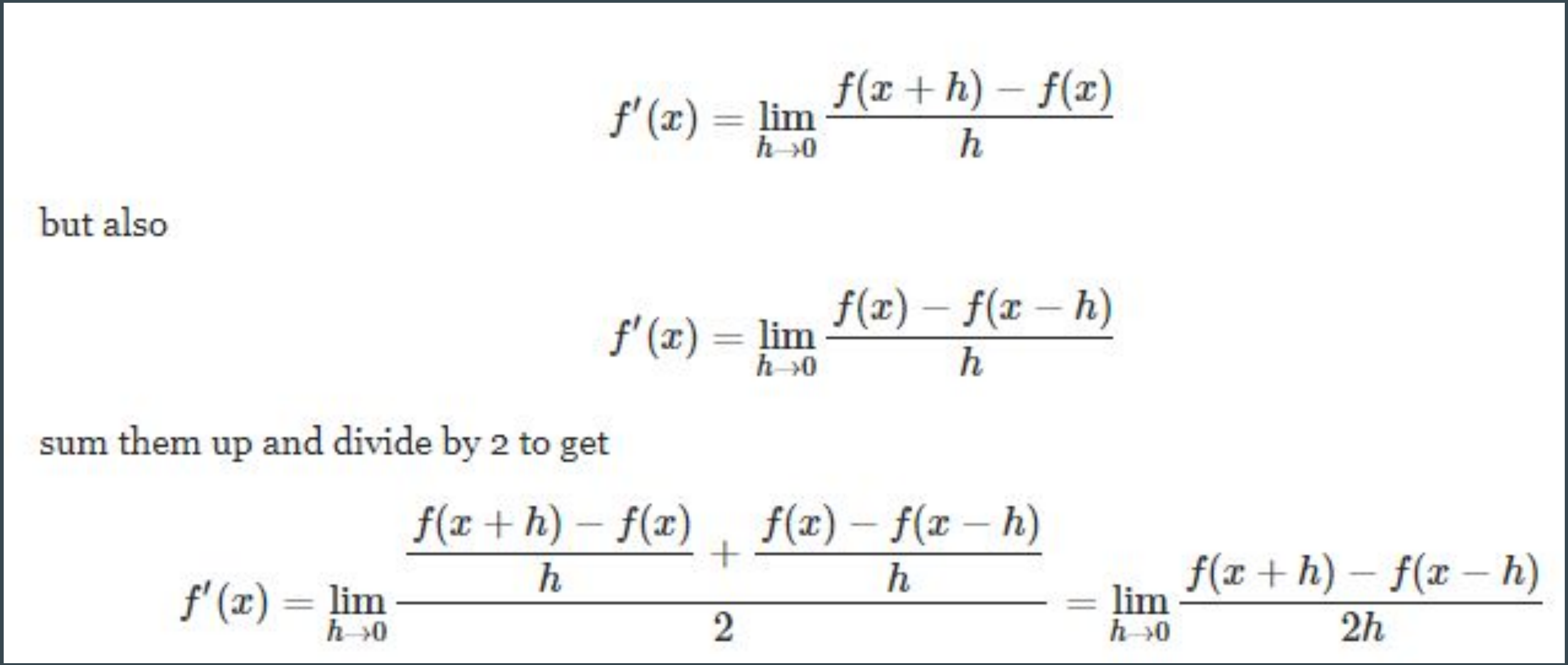


图 4-5　真的导数（真的切线）和数值微分（近似切线）的值不同

Derivative cont.



$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h}$$

but also

$$f'(x) = \lim_{h \to 0} \frac{f(x) - f(x-h)}{h}$$

sum them up and divide by 2 to get

$$f'(x) = \lim_{h \to 0} \frac{\frac{f(x+h) - f(x)}{h} + \frac{f(x) - f(x-h)}{h}}{2} = \lim_{h \to 0} \frac{f(x+h) - f(x-h)}{2h}$$

https://math.stackexchange.com/questions/1500892/show-that-fx-lim-limits-h-rightarrow-0-dfracfxh-fx-h2h

Numerical differentiation (微分)

$$y = 0.01x^2 + 0.1x$$

```
def function_1(x):
    return 0.01*x**2 + 0.1*x
```

```
import numpy as np
import matplotlib.pylab as plt

x = np.arange(0.0, 20.0, 0.1) # 以0.1为单位，从0到20的数组x
y = function_1(x)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.plot(x, y)
plt.show()
```

```
>>> numerical_diff(function_1, 5)
0.1999999999990898
>>> numerical_diff(function_1, 10)
0.2999999999986347
```

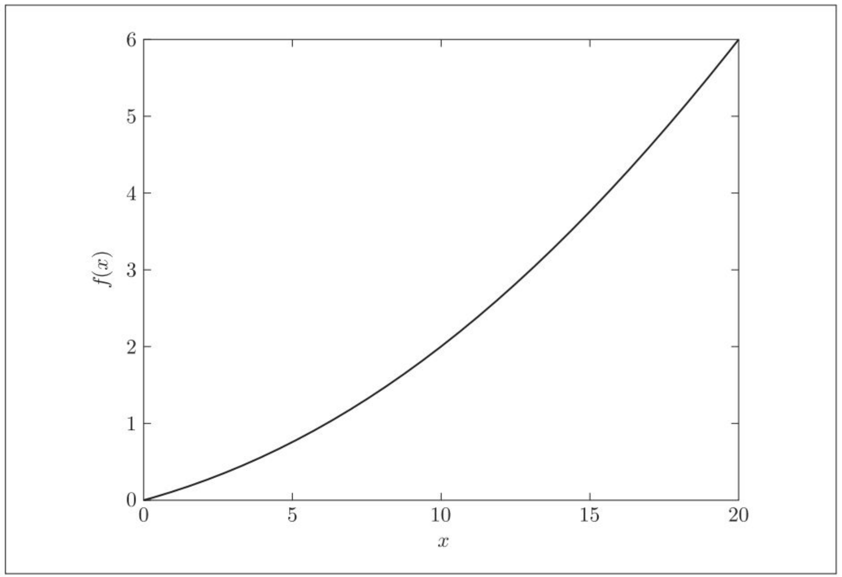


图4-6　$f(x) = 0.01x^2 + 0.1x$的图像

$$\frac{\mathrm{d}f(x)}{\mathrm{d}x} = 0.02x + 0.1$$

Numerical differentiation cont.

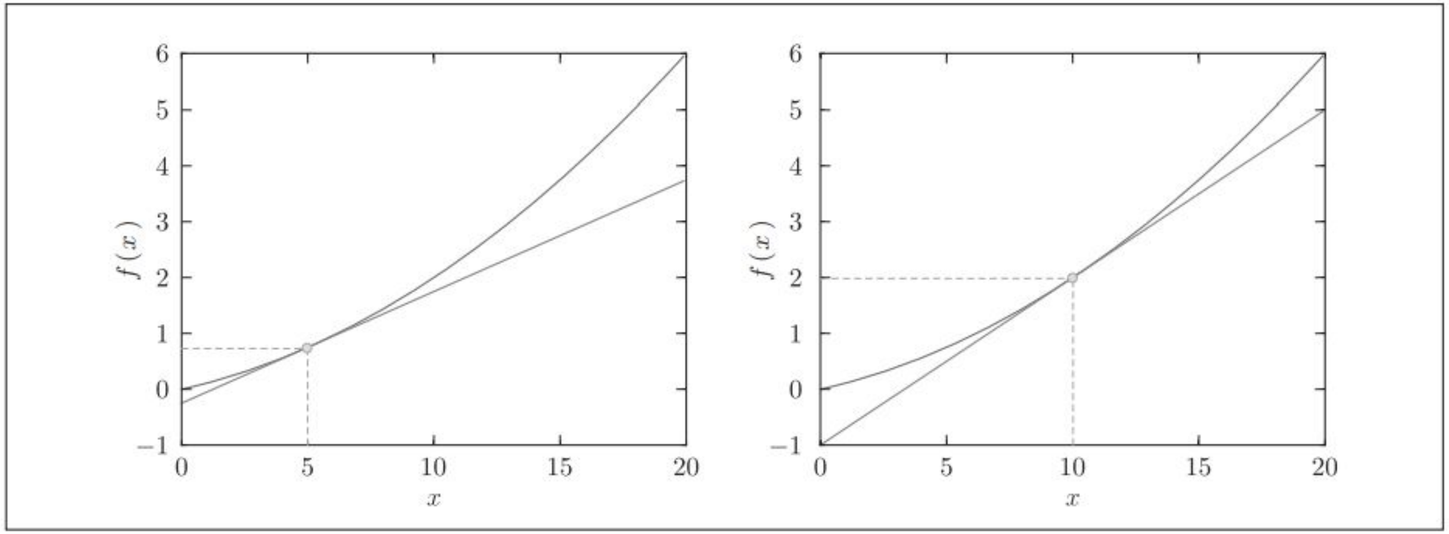


图 4-7　$x = 5$、$x = 10$ 处的切线：直线的斜率使用数值微分的值

Partial derivative (偏导数)

In mathematics, a partial derivative of a function of several variables is its derivative with respect to one of those variables, with the others held constant (as opposed to the total derivative, in which all variables are allowed to vary).

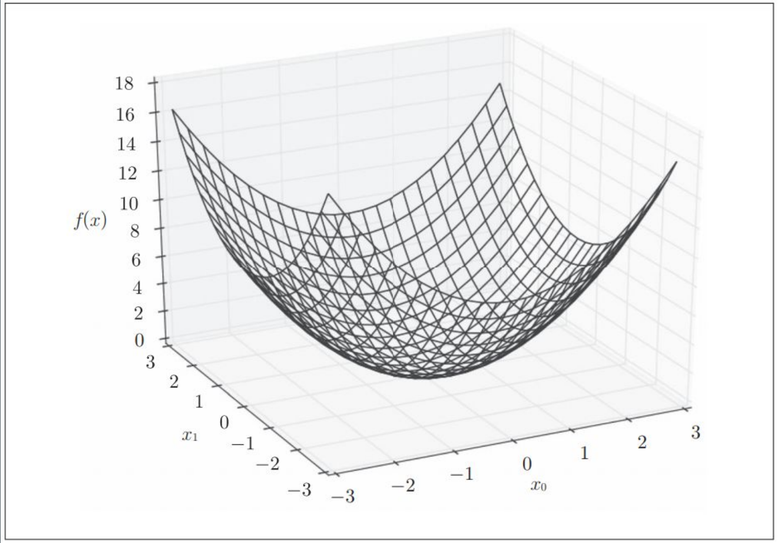
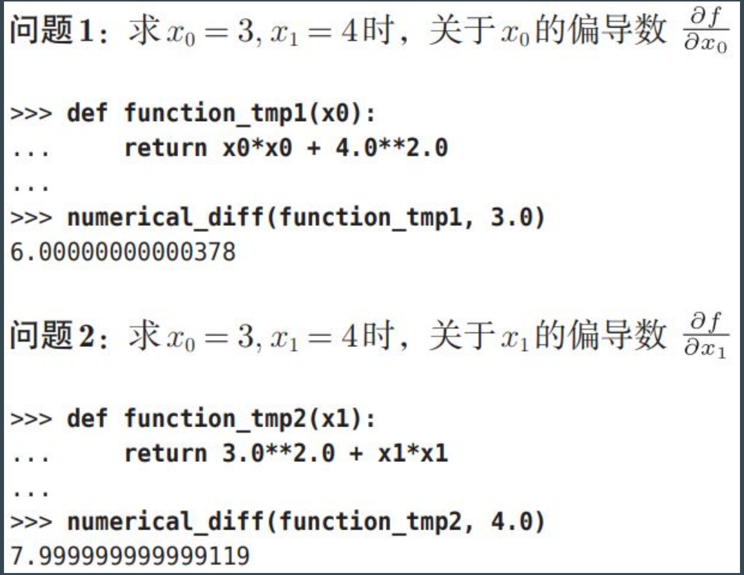


图4-8 $f(x_0, x_1) = x_0^2 + x_1^2$ 的图像

$\frac{\partial f}{\partial x_0}$、$\frac{\partial f}{\partial x_1}$



问题1：求 $x_0 = 3, x_1 = 4$ 时，关于 x_0 的偏导数 $\frac{\partial f}{\partial x_0}$

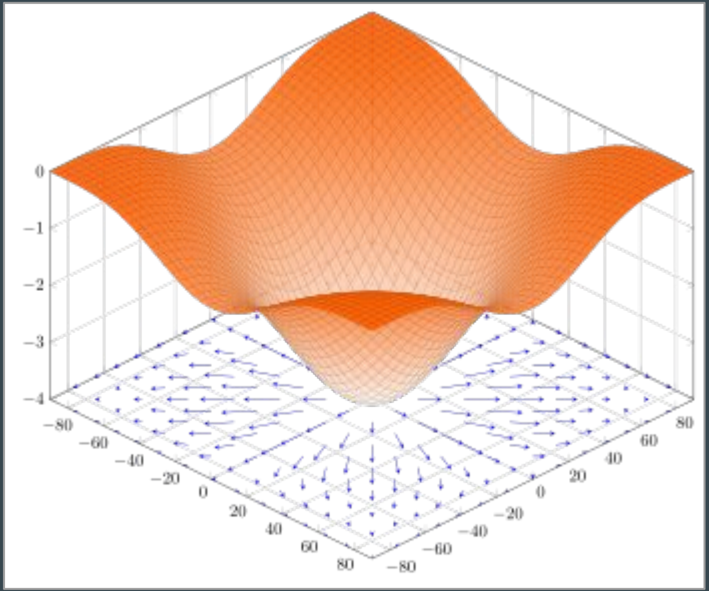
```
>>> def function_tmp1(x0):
...     return x0*x0 + 4.0**2.0
...
>>> numerical_diff(function_tmp1, 3.0)
6.00000000000378
```

问题2：求 $x_0 = 3, x_1 = 4$ 时，关于 x_1 的偏导数 $\frac{\partial f}{\partial x_1}$

```
>>> def function_tmp2(x1):
...     return 3.0**2.0 + x1*x1
...
>>> numerical_diff(function_tmp2, 4.0)
7.999999999999119
```

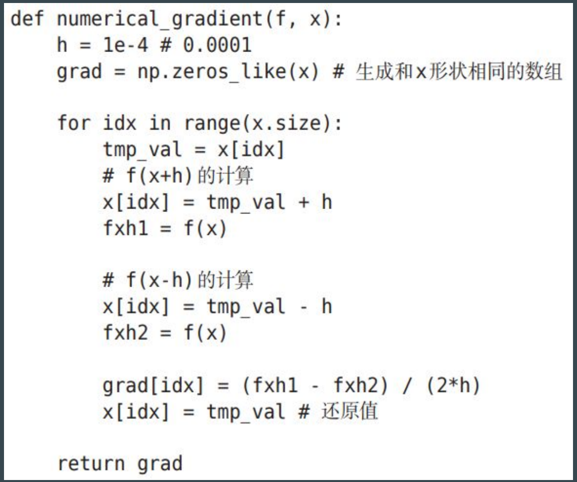
Gradient

In mathematics, the gradient is a multi-variable generalization of the derivative.

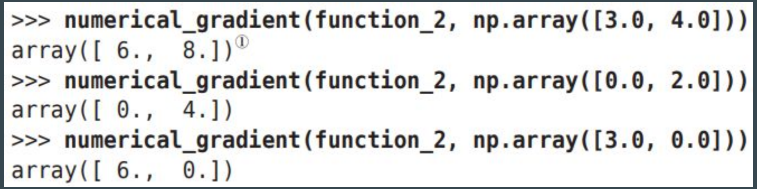


https://en.wikipedia.org/wiki/Gradient

Gradient cont.



```
def numerical_gradient(f, x):
    h = 1e-4 # 0.0001
    grad = np.zeros_like(x) # 生成和x形状相同的数组

    for idx in range(x.size):
        tmp_val = x[idx]
        # f(x+h)的计算
        x[idx] = tmp_val + h
        fxh1 = f(x)

        # f(x-h)的计算
        x[idx] = tmp_val - h
        fxh2 = f(x)

        grad[idx] = (fxh1 - fxh2) / (2*h)
        x[idx] = tmp_val # 还原值

    return grad
```



```
>>> numerical_gradient(function_2, np.array([3.0, 4.0]))
array([ 6.,  8.])[1]
>>> numerical_gradient(function_2, np.array([0.0, 2.0]))
array([ 0.,  4.])
>>> numerical_gradient(function_2, np.array([3.0, 0.0]))
array([ 6.,  0.])
```

?

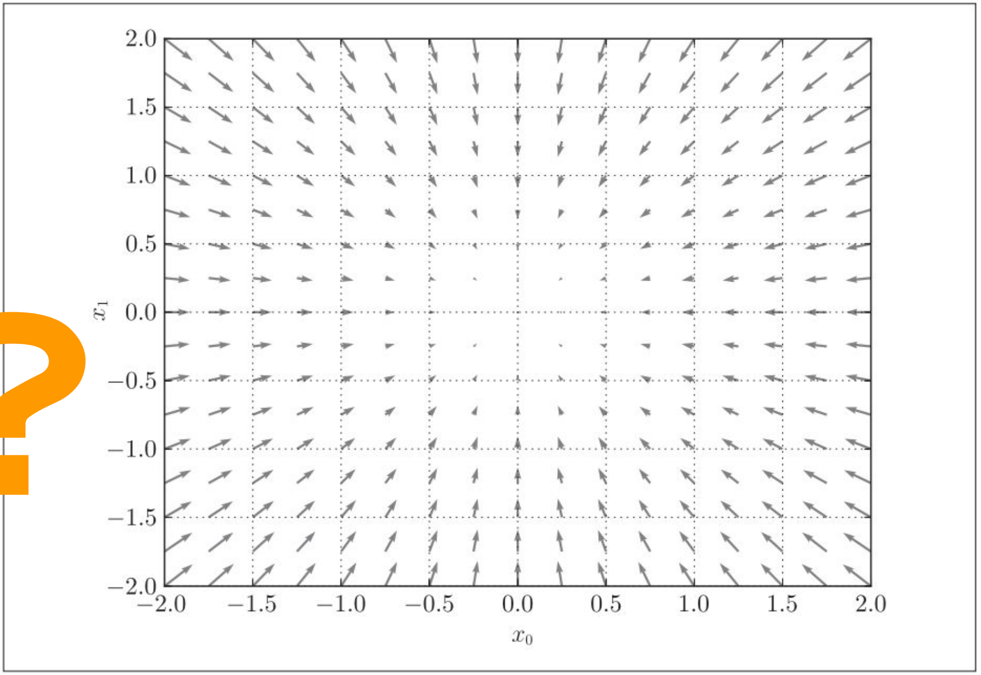


图4-9 $f(x_0, x_1) = x_0^2 + x_1^2$ 的梯度

Gradient method

In optimization, gradient method is an algorithm to solve problems of the form

$$\min_{x \in \mathbb{R}^n} f(x)$$

with the search directions defined by the gradient of the function at the current point. Examples of gradient method are the gradient descent and the conjugate gradient.

- **Gradient descent**
    - Gradient descent is a first-order iterative optimization algorithm for finding the minimum of a function.
- **Gradient ascent**
    - Vice versa

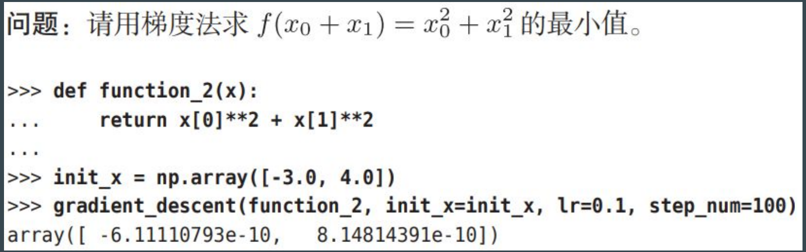
Gradient method cont.

$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$
$$x_1 = x_1 - \eta \frac{\partial f}{\partial x_1}$$

η : Learning rate

```
def gradient_descent(f, init_x, lr=0.01, step_num=100):
    x = init_x

    for i in range(step_num):
        grad = numerical_gradient(f, x)
        x -= lr * grad

    return x
```

Gradient method cont.



问题：请用梯度法求 $f(x_0+x_1)=x_0^2+x_1^2$ 的最小值。

```
>>> def function_2(x):
...     return x[0]**2 + x[1]**2
...
>>> init_x = np.array([-3.0, 4.0])
>>> gradient_descent(function_2, init_x=init_x, lr=0.1, step_num=100)
array([ -6.11110793e-10,   8.14814391e-10])
```

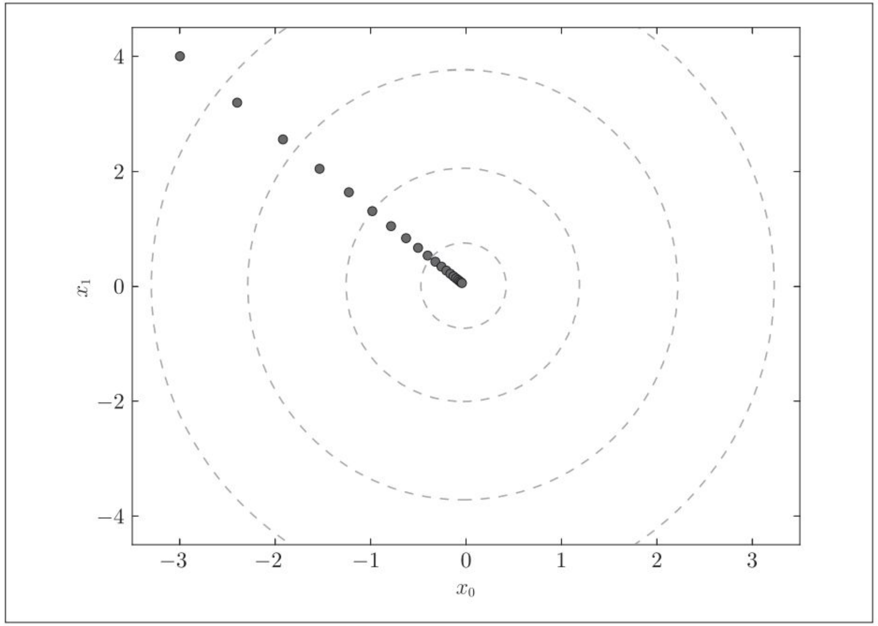
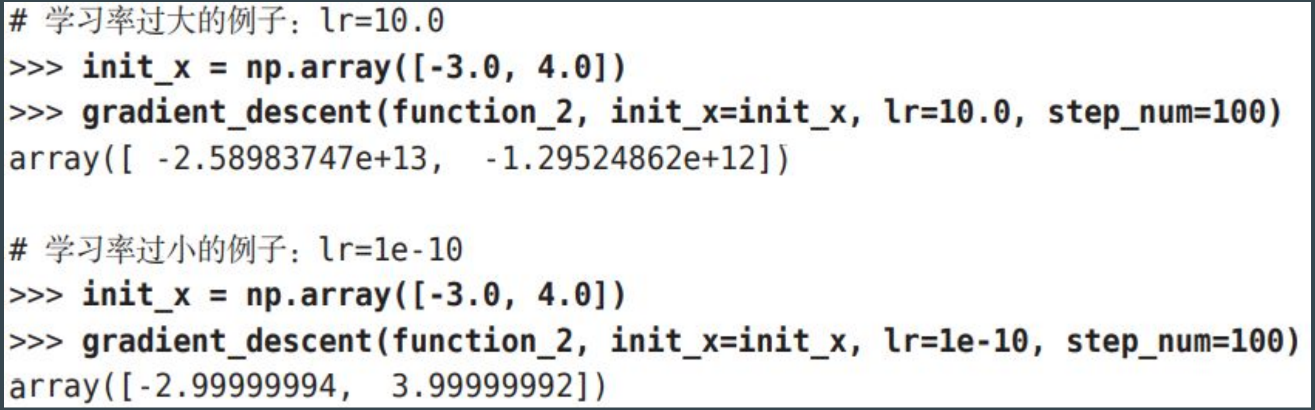


图4-10　$f(x_0, x_1) = x_0^2 + x_1^2$ 的梯度法的更新过程：虚线是函数的等高线

Gradient method - learning rate

Learning rate need to be set correctly.
Too big or too small will never be able to find a right point.



```
# 学习率过大的例子：lr=10.0
>>> init_x = np.array([-3.0, 4.0])
>>> gradient_descent(function_2, init_x=init_x, lr=10.0, step_num=100)
array([ -2.58983747e+13,  -1.29524862e+12])

# 学习率过小的例子：lr=1e-10
>>> init_x = np.array([-3.0, 4.0])
>>> gradient_descent(function_2, init_x=init_x, lr=1e-10, step_num=100)
array([-2.99999994,  3.99999992])
```

Gradient in Deep Learning

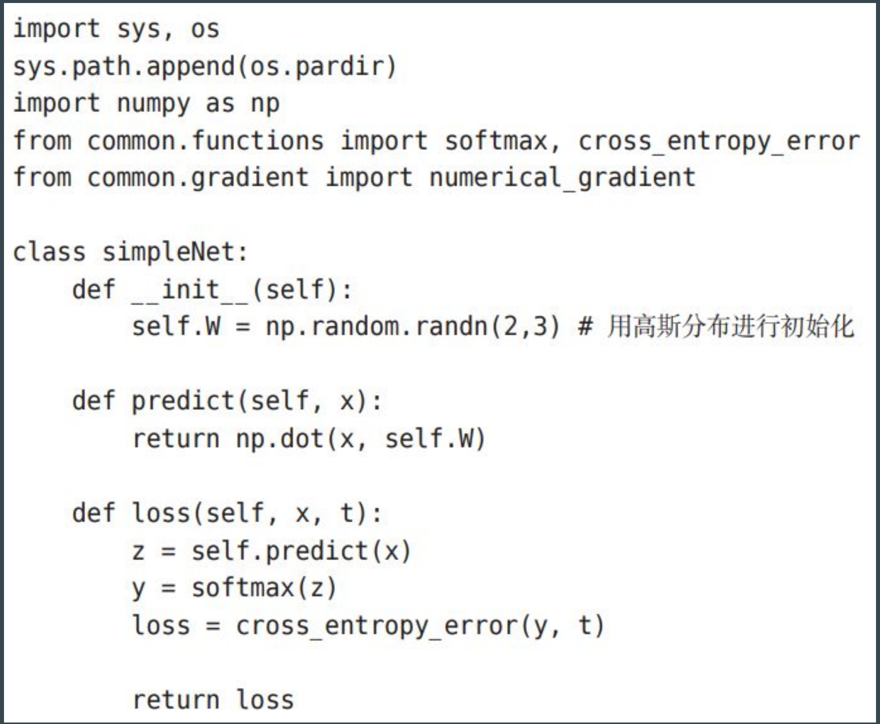
Weights and loss function:

$$\boldsymbol{W} = \begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \end{pmatrix}$$

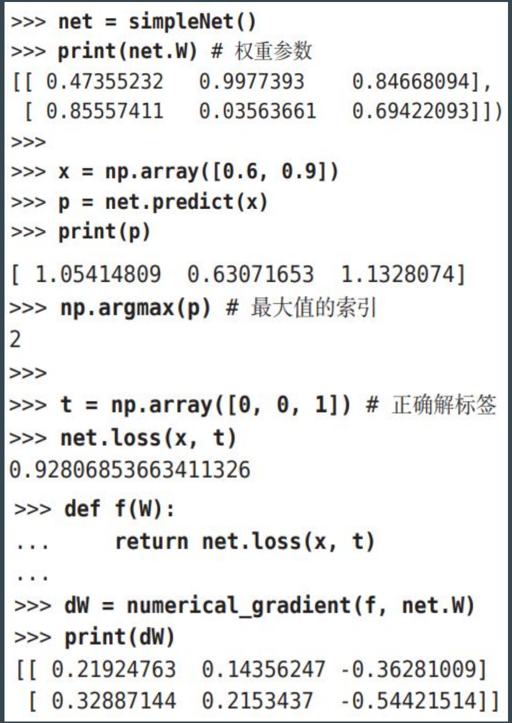
$$\frac{\partial L}{\partial \boldsymbol{W}} = \begin{pmatrix} \frac{\partial L}{\partial w_{11}} & \frac{\partial L}{\partial w_{12}} & \frac{\partial L}{\partial w_{13}} \\ \frac{\partial L}{\partial w_{21}} & \frac{\partial L}{\partial w_{22}} & \frac{\partial L}{\partial w_{23}} \end{pmatrix}$$

$\frac{\partial L}{\partial w_{11}}$ 表示当 w_{11} 稍微变化时，损失函数 L 会发生多大变化

Calculate gradient with sample



```python
import sys, os
sys.path.append(os.pardir)
import numpy as np
from common.functions import softmax, cross_entropy_error
from common.gradient import numerical_gradient

class simpleNet:
    def __init__(self):
        self.W = np.random.randn(2,3) # 用高斯分布进行初始化

    def predict(self, x):
        return np.dot(x, self.W)

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

        return loss
```

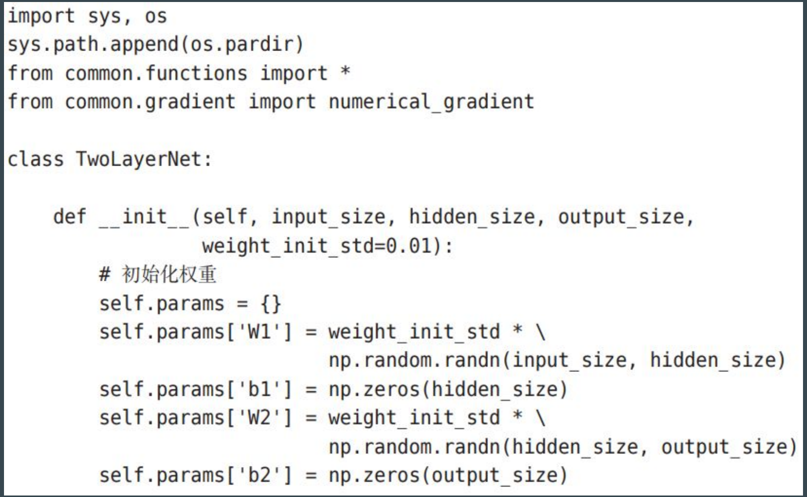


```
>>> net = simpleNet()
>>> print(net.W) # 权重参数
[[ 0.47355232   0.9977393    0.84668094],
 [ 0.85557411   0.03563661   0.69422093]])
>>>
>>> x = np.array([0.6, 0.9])
>>> p = net.predict(x)
>>> print(p)
[ 1.05414809  0.63071653  1.1328074]
>>> np.argmax(p) # 最大值的索引
2
>>>
>>> t = np.array([0, 0, 1]) # 正确解标签
>>> net.loss(x, t)
0.92806853663411326
>>> def f(W):
...     return net.loss(x, t)
...
>>> dW = numerical_gradient(f, net.W)
>>> print(dW)
[[ 0.21924763  0.14356247 -0.36281009]
 [ 0.32887144  0.2153437  -0.54421514]]
```

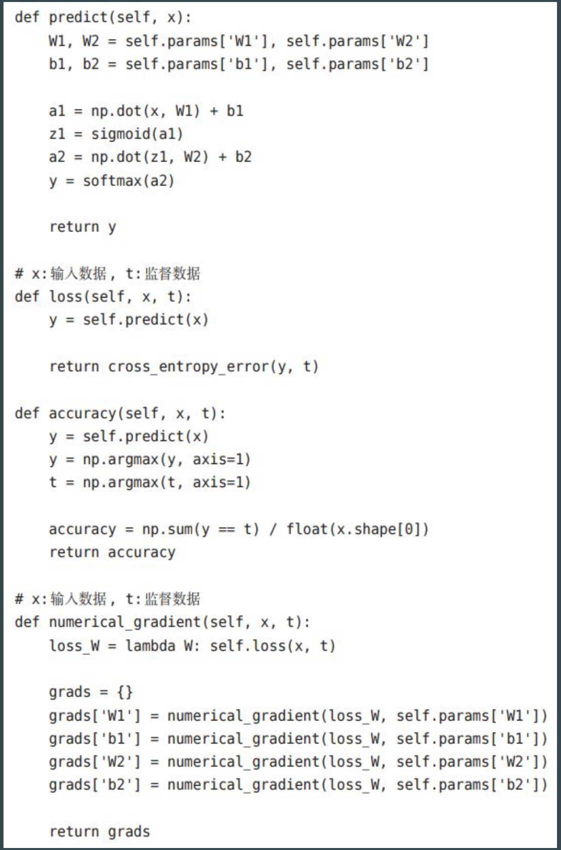
Learning steps of Deep learning (stochastic gradient descent)

1. Mini-batch
   a. To minimize loss-function for mini-batch data
2. Calculate gradient
   a. Get all the gradients for all the weights
3. Update weights
   a. Update weights in gradients' direction
4. Repeat

Two-layered net source code

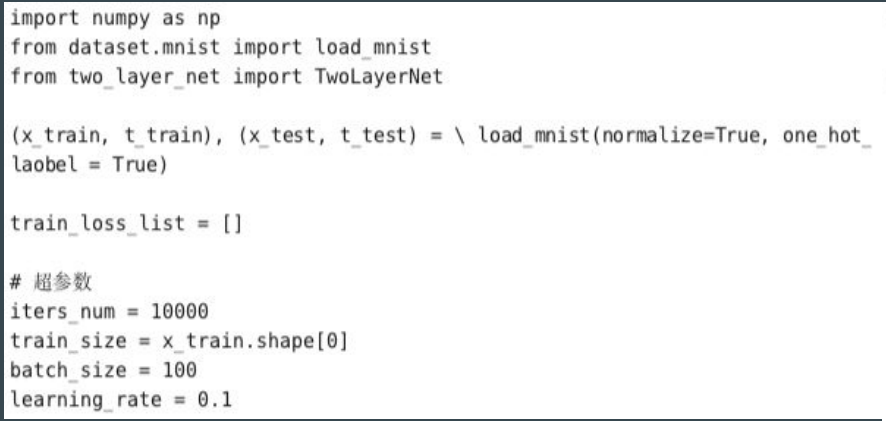


```python
import sys, os
sys.path.append(os.pardir)
from common.functions import *
from common.gradient import numerical_gradient

class TwoLayerNet:

    def __init__(self, input_size, hidden_size, output_size,
                 weight_init_std=0.01):
        # 初始化权重
        self.params = {}
        self.params['W1'] = weight_init_std * \
                            np.random.randn(input_size, hidden_size)
        self.params['b1'] = np.zeros(hidden_size)
        self.params['W2'] = weight_init_std * \
                            np.random.randn(hidden_size, output_size)
        self.params['b2'] = np.zeros(output_size)
```

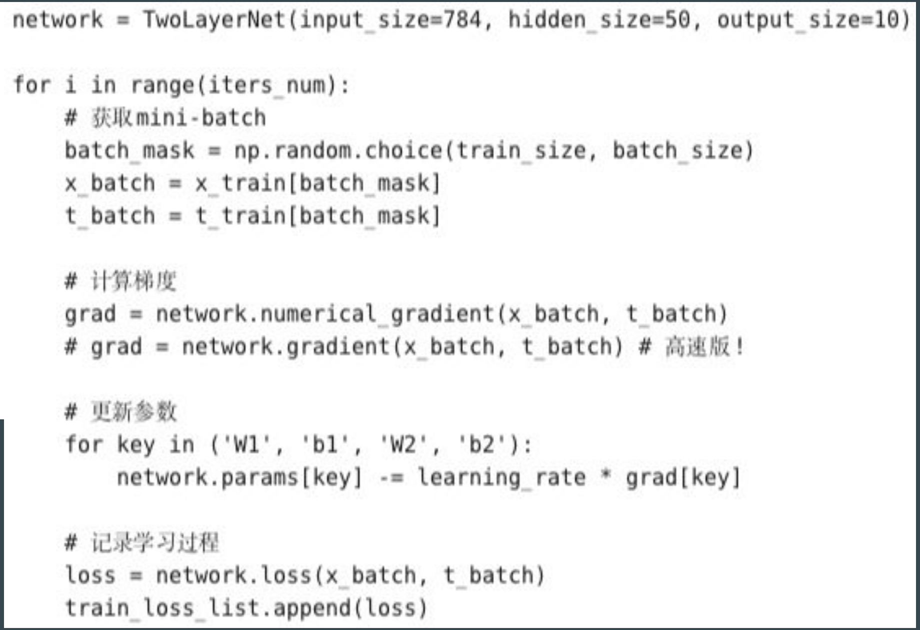


```python
def predict(self, x):
    W1, W2 = self.params['W1'], self.params['W2']
    b1, b2 = self.params['b1'], self.params['b2']

    a1 = np.dot(x, W1) + b1
    z1 = sigmoid(a1)
    a2 = np.dot(z1, W2) + b2
    y = softmax(a2)

    return y

# x:输入数据, t:监督数据
def loss(self, x, t):
    y = self.predict(x)

    return cross_entropy_error(y, t)

def accuracy(self, x, t):
    y = self.predict(x)
    y = np.argmax(y, axis=1)
    t = np.argmax(t, axis=1)

    accuracy = np.sum(y == t) / float(x.shape[0])
    return accuracy

# x:输入数据, t:监督数据
def numerical_gradient(self, x, t):
    loss_W = lambda W: self.loss(x, t)

    grads = {}
    grads['W1'] = numerical_gradient(loss_W, self.params['W1'])
    grads['b1'] = numerical_gradient(loss_W, self.params['b1'])
    grads['W2'] = numerical_gradient(loss_W, self.params['W2'])
    grads['b2'] = numerical_gradient(loss_W, self.params['b2'])

    return grads
```

Mini-batch source code



```
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

(x_train, t_train), (x_test, t_test) = \ load_mnist(normalize=True, one_hot_
laobel = True)

train_loss_list = []

# 超参数
iters_num = 10000
train_size = x_train.shape[0]
batch_size = 100
learning_rate = 0.1
```



```
network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

for i in range(iters_num):
    # 获取mini-batch
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 计算梯度
    grad = network.numerical_gradient(x_batch, t_batch)
    # grad = network.gradient(x_batch, t_batch) # 高速版!

    # 更新参数
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    # 记录学习过程
    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)
```

Change in loss function

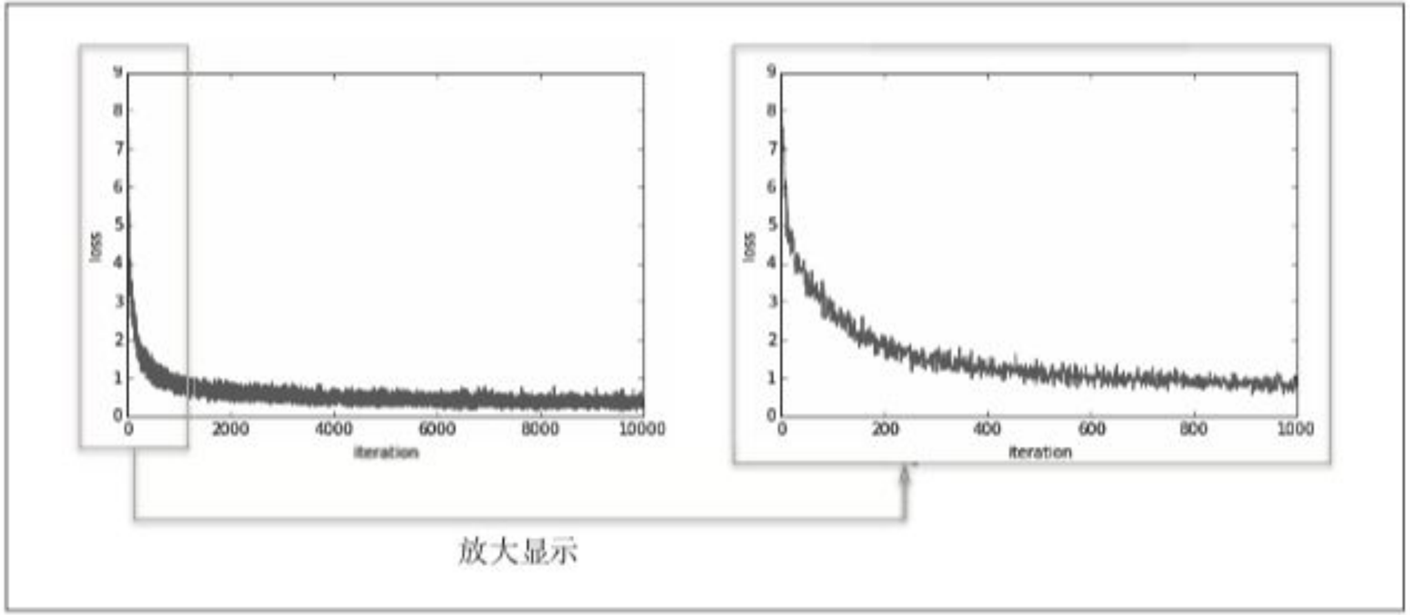


图 4-11　损失函数的推移：左图是 10000 次循环的推移，右图是 1000 次循环的推移

```
import numpy as np
from dataset.mnist import load_mnist
from two_layer_net import TwoLayerNet

(x_train, t_train), (x_test, t_test) = \ load_mnist(normalize=True, one_hot_
laobel = True)

train_loss_list = []
train_acc_list = []
test_acc_list = []
# 平均每个epoch的重复次数
iter_per_epoch = max(train_size / batch_size, 1)

# 超参数
iters_num = 10000
batch_size = 100
learning_rate = 0.1

network = TwoLayerNet(input_size=784, hidden_size=50, output_size=10)

for i in range(iters_num):
    # 获取mini-batch
    batch_mask = np.random.choice(train_size, batch_size)
    x_batch = x_train[batch_mask]
    t_batch = t_train[batch_mask]

    # 计算梯度
    grad = network.numerical_gradient(x_batch, t_batch)
    # grad = network.gradient(x_batch, t_batch) # 高速版!

    # 更新参数
    for key in ('W1', 'b1', 'W2', 'b2'):
        network.params[key] -= learning_rate * grad[key]

    loss = network.loss(x_batch, t_batch)
    train_loss_list.append(loss)
    # 计算每个epoch的识别精度
    if i % iter_per_epoch == 0:
        train_acc = network.accuracy(x_train, t_train)
        test_acc = network.accuracy(x_test, t_test)
        train_acc_list.append(train_acc)
        test_acc_list.append(test_acc)
        print("train acc, test acc | " + str(train_acc) + ", " + str(test_acc))
```

Accuracy change based no epochs

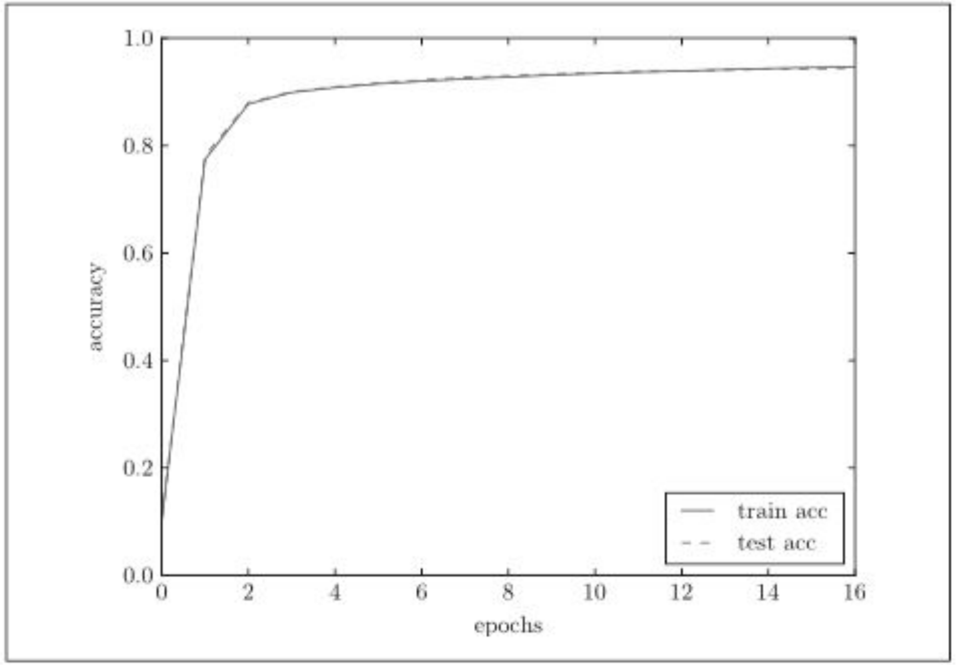


图 4-12　训练数据和测试数据的识别精度的推移（横轴的单位是 epoch）