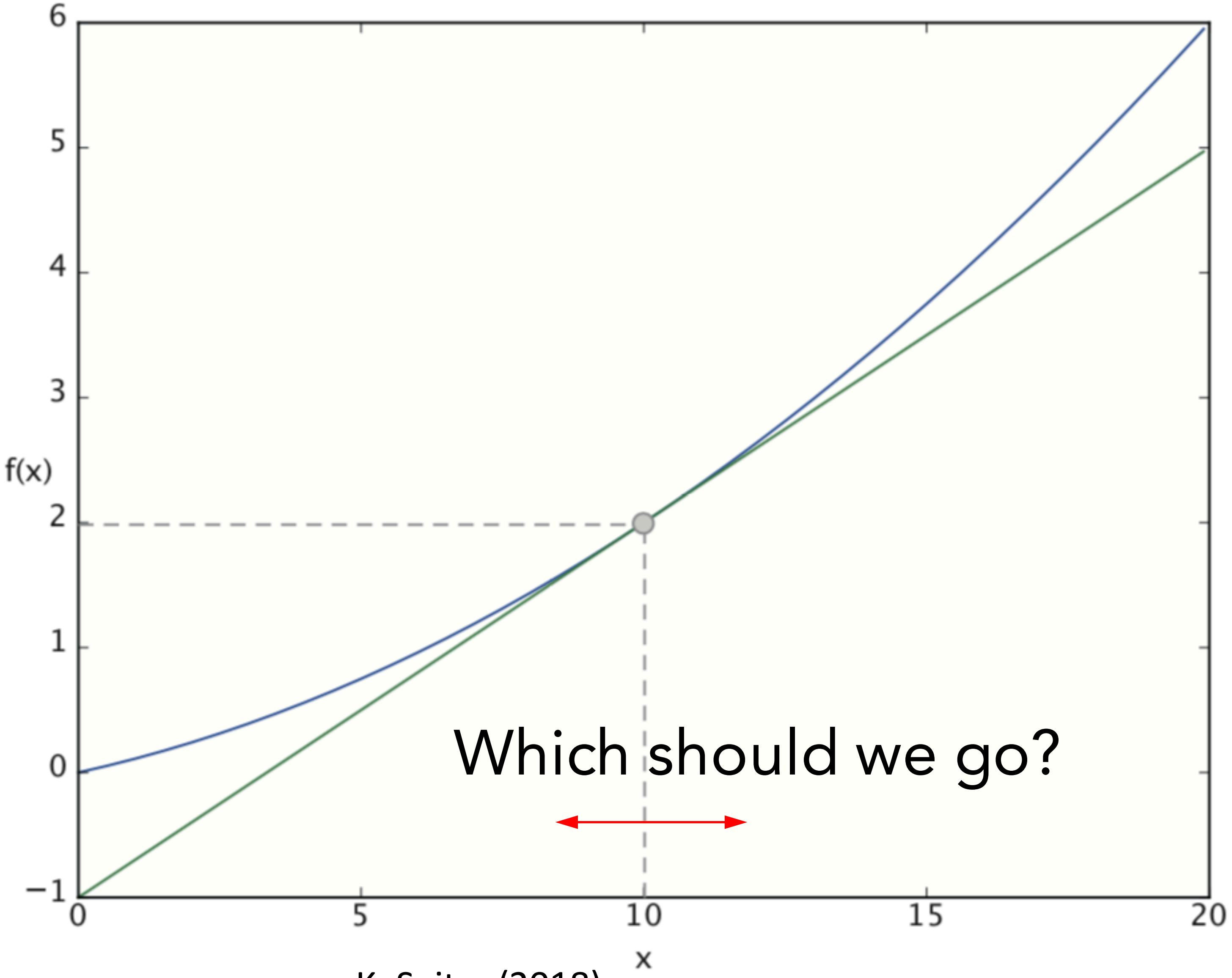
Python learning
chapter 4

section 4.3: Numerical differentiation
section 4.4: Gradient

M1 Sano Chihiro

Objective



K. Saito, (2018)

Differentiation

The definition of differentiation is…

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

Numerical differentiation

Try to express differentiation on programing

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

Numerical differentiation, point

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

Not to become too small
Use central difference

Partial differentiation

For example: $f(x_0,x_1)=x_0^2 + x_1^2$

When $(x_0,x_1) = (3, 4)$, calculate $\frac{\partial f}{\partial x_0}$

```
def function_2(x):
        return x[0]**2 + x[1]**2


def function_tmp1(x0):
        return x0*x0 + 4.0**2

Numerical_diff(function_tmp1, 3.0)
```

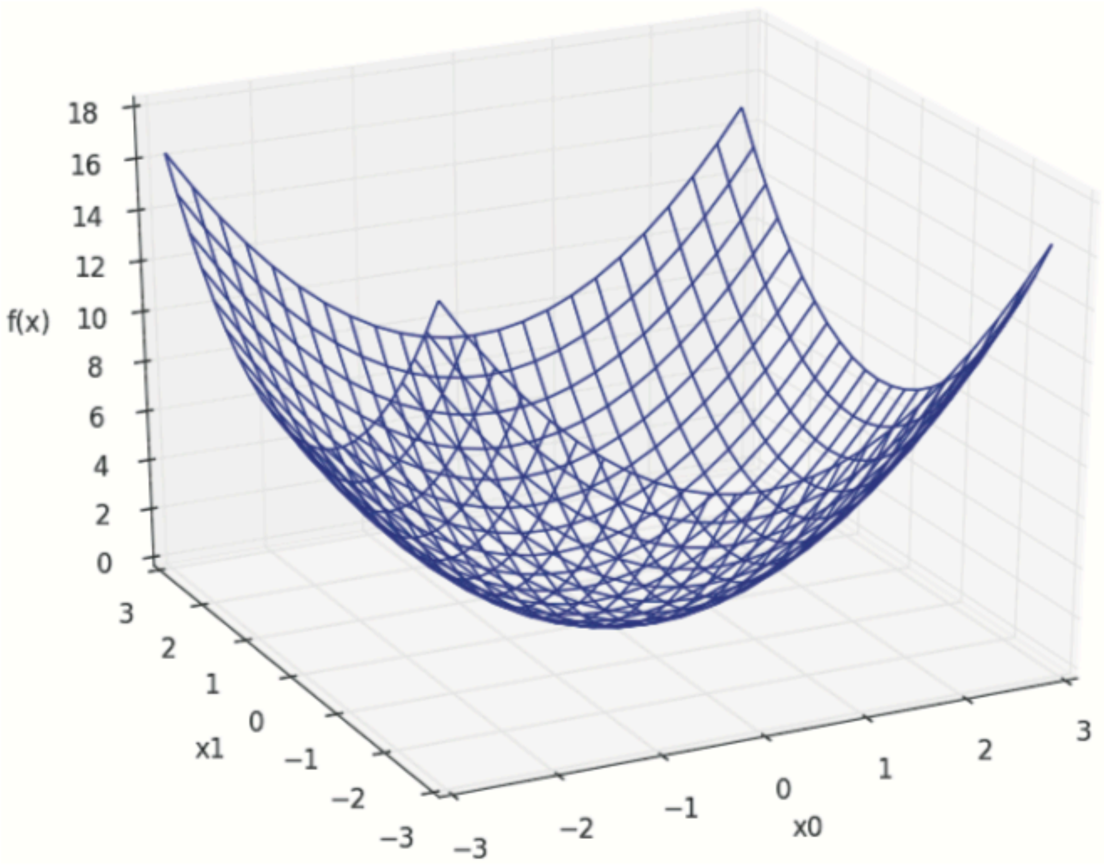
Gradient

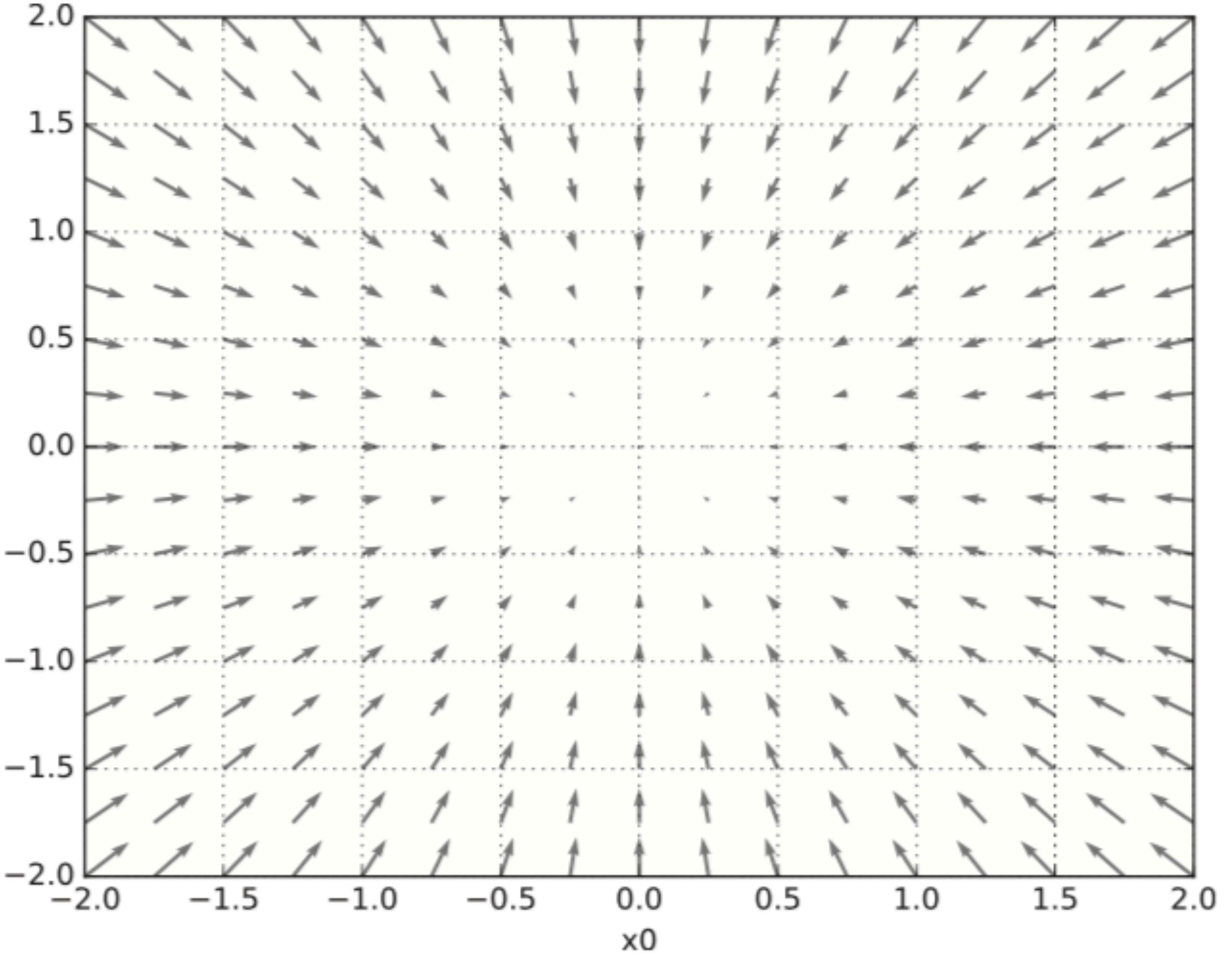
We can calculate gradient like this:

$$\nabla f = \left(\frac{\partial f}{\partial x_0}, \frac{\partial f}{\partial x_1}\right)$$

```
def numerical_gradient(f, x):
        h = 1e-4
        grad = np.zeros_like(x)

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

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

                grad[idx] = (fxh1 – fxh2) / (2*h)
                x[idx] = tmp_val

        return grad
```

Gradient





K. Saito, (2018)

Gradient method

Moving to gradient direction and
Decreasing the function value

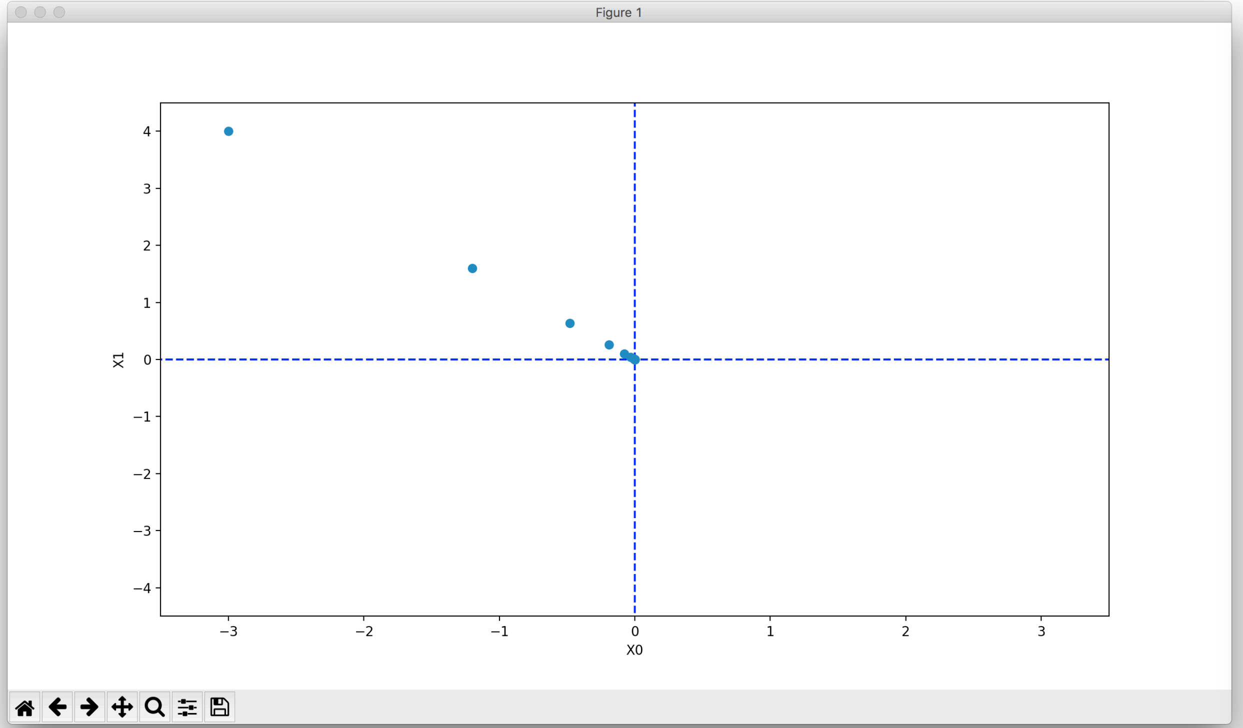
$$x_0 = x_0 - \eta \frac{\partial f}{\partial x_0}$$

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

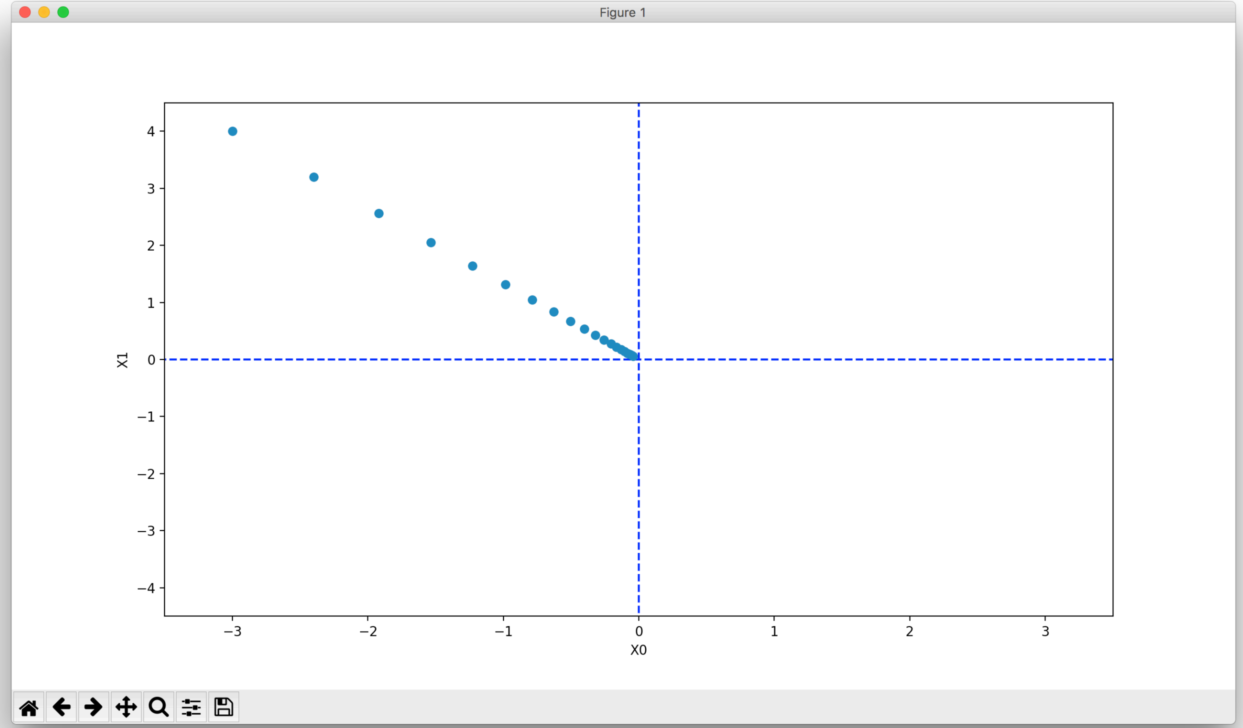
η: learning rate

```
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
```

Learning rate (1)

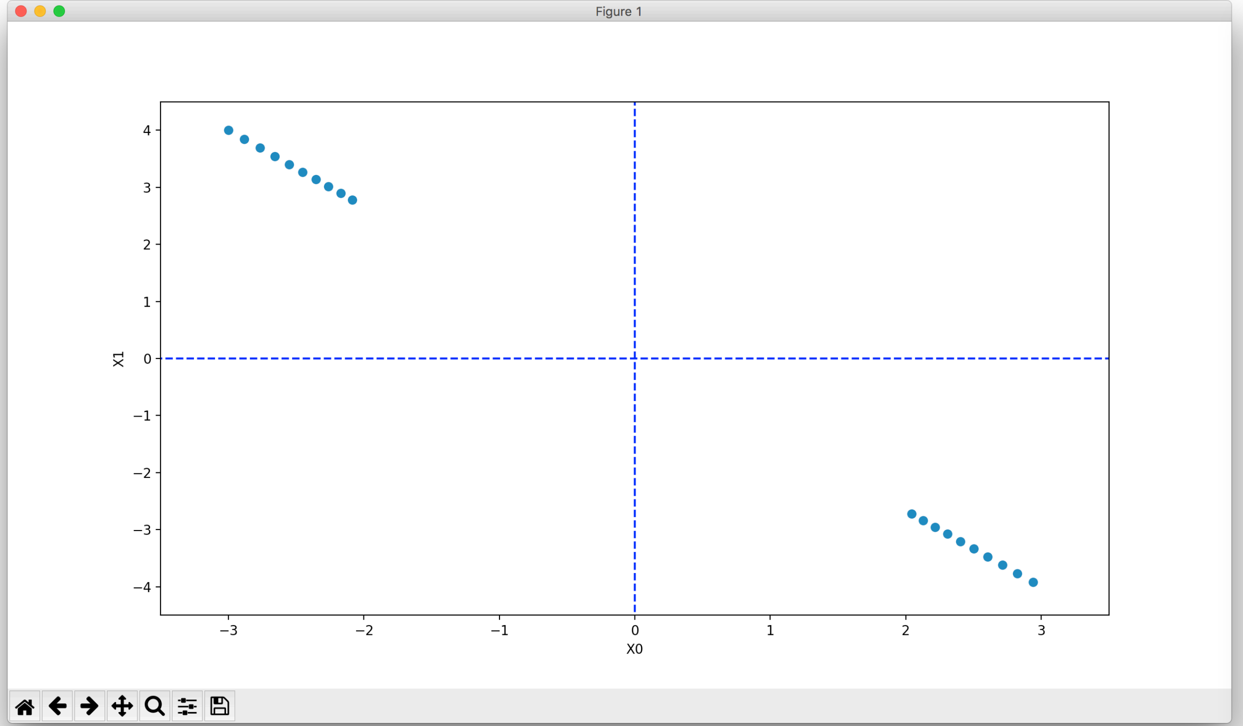


Learning rate: 0.3

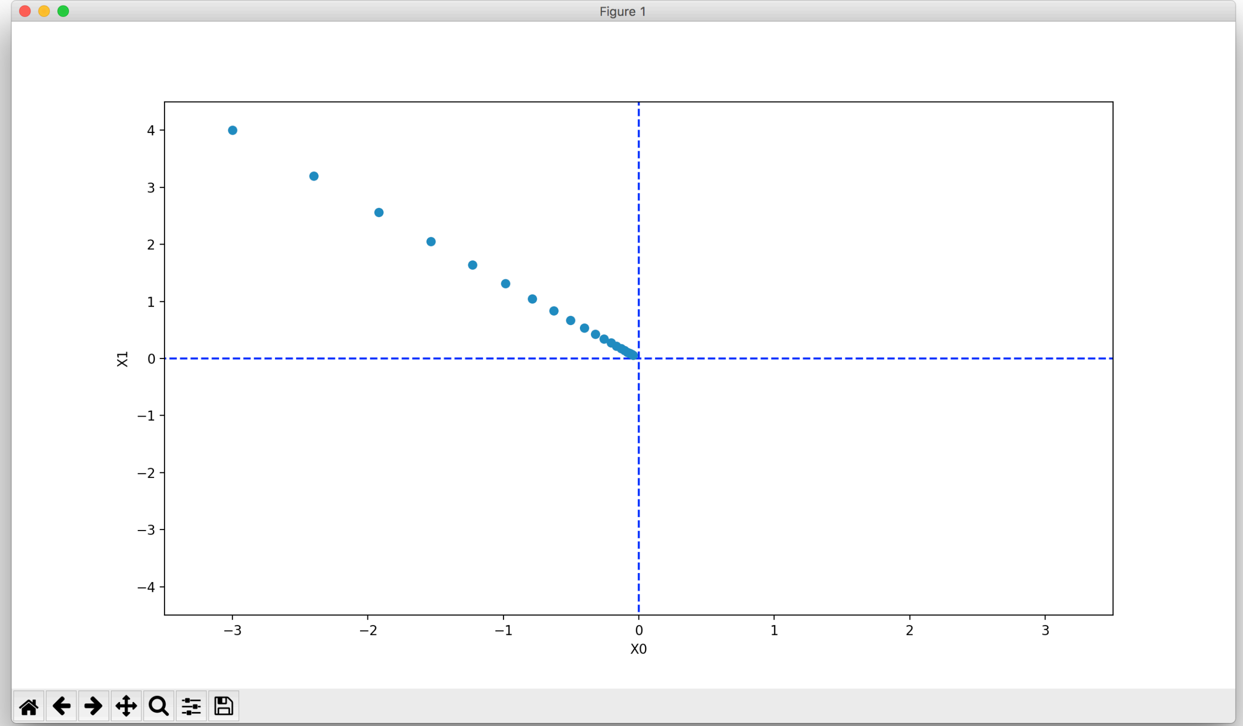


Learning rate: 0.1

Learning rate (2)



Learning rate: 0.99



Learning rate: 0.1

Learning rate (3)

- We have to choose “good learning rate”
  it is difficult

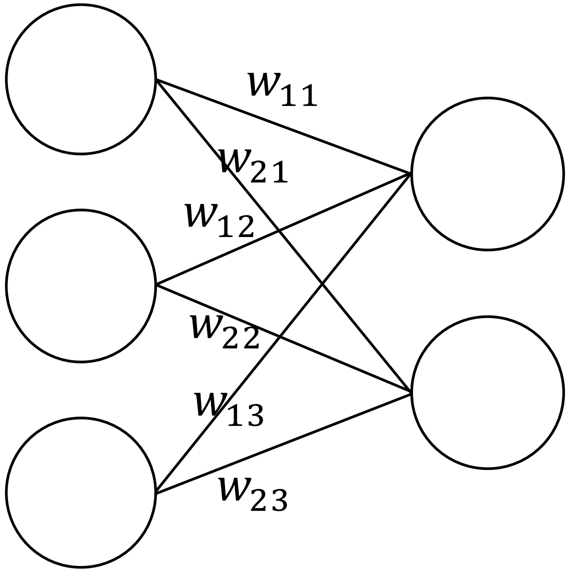
=>Ada Grad, RMSProp, Adam

Gradient for NN

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

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

L : Loss function



Gradient for NN

```
import sys, ossys.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)
    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
x = np.array([0.6, 0.9])
t = np.array([0, 0, 1])

net = simpleNet()

f = lambda w: net.loss(x, t)
dW = numerical_gradient(f, net.W)

print(dW)
```

Result

```
[[ 0.22195011  0.21065484 -0.43260494]
 [ 0.33292516  0.31598226 -0.64890742]]
```

Update to “-”

Update to “+”

Summery

- Decrease the value of loss function with gradient method
- When you use gradient method, the learning rate has important mean.