Python Programming

Jian Zhang

NumPy is a Python library used for working with arrays.

It also has functions for working in domain of linear algebra, fourier transform, and matrices.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

NumPy stands for Numerical Python.

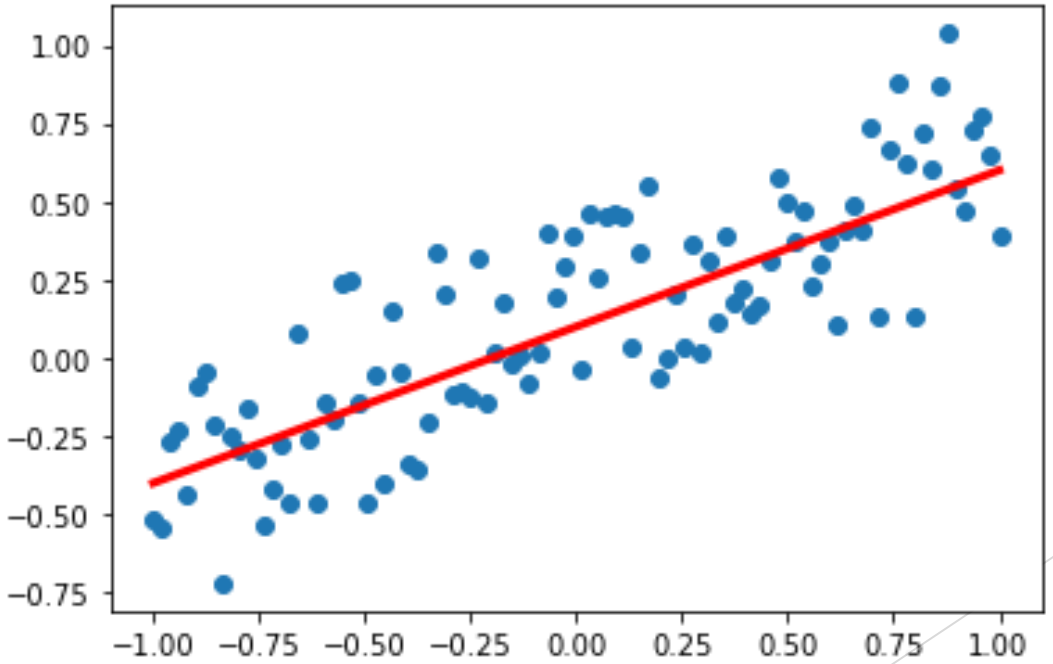
!pip install numpy

Array Operations/Math

```
x = np.array([[1, 2, 3],
              [4, 5, 6]])

print(np.sum(x))

print(np.sum(x, axis=0))

print(np.sum(x, axis=1))

print(np.max(x, axis=1))

x = np.array([[1, 2, 3],
              [3, 5, 7]])
print(x * 2)
```

Regression Problem

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1, 1, 100)
y = 0.5*x + 0.1 + + 0.2*np.random.randn(100)
```



Regression Problem

```
learning_rate = 0.01
initial_b = 0
initial_w = 0
num_iterations = 101

b = initial_b
w = initial_w

for i in range(num_iterations):
    b, w = step_gradient(b, w, points, learning_rate)

print(b, w)
```

Regression Problem

```
def step_gradient(b_current, w_current, points, learningRate):
    b_gradient = 0
    w_gradient = 0
    N = float(len(points))
    for i in range(0, len(points)):
        x = points[i, 0]
        y = points[i, 1]
        b_gradient += -(2/N) * (y - ((w_current * x) + b_current))
        w_gradient += -(2/N) * x * (y - ((w_current * x) + b_current))
    new_b = b_current - (learningRate * b_gradient)
    new_w = w_current - (learningRate * w_gradient)
    return [new_b, new_w]
```

Regression Problem

```
def step_gradient_fast(b_current, w_current, points, learningRate):
    N = float(len(points))
    b_gradient = np.sum(-(2/N)*(points[:,1] - ((w_current * x) + b_current)))
    w_gradient = np.sum(-(2/N)*x * (y - ((w_current * x) + b_current)))

    new_b = b_current - (learningRate * b_gradient)
    new_w = w_current - (learningRate * w_gradient)
    return [new_b, new_w]
```

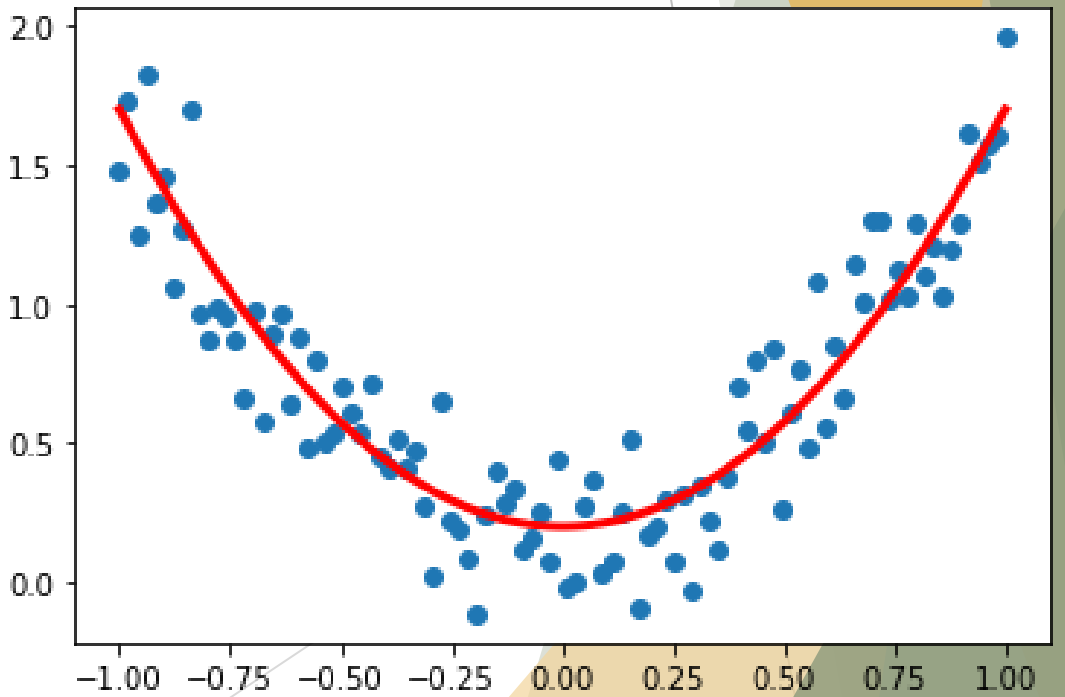
Regression Problem

```
learning_rate = 0.01
b = 0
w = 0
num_iterations = 1001
points = np.zeros((100, 2))
points[:,0] = x
points[:,1] = y

for i in range(num_iterations):
    if i % 100 == 0:
        print('iteration %d, error is %.4f' % (i, np.mean((y - (w * x + b)) ** 2)))
        plt.scatter(x, y)
        plt.plot(x, 0.5*x + 0.1 , 'r-', lw=2)
        plt.show()
    b, w = step_gradient_fast(b, w, points, learning_rate)

print("b = %.4f, w = %.4f" % (b, w))
```

Homework

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt

x = np.linspace(-1, 1, 100)
y = 1.5*x*x + 0.2 + + 0.2*np.random.randn(100)
plt.scatter(x, y)
plt.plot(x, 1.5*x*x + 0.2, 'r-', lw=3)
plt.show()
```

Given a set of blue points, calculate the parameters of the fitted quadratic curve using the gradient descent method.



Questions?