UNIVERSITÄT HAMBURG

MASHINE LEARNING

EXERCISE1

Stochastic Gradient Descent

Benjamin Ostendorf
Waijd Ghafoor

Prof. Dr. Victor Emanuel
de Atocha Uc Cetina
Dr. Timo Baumann

April 15, 2019

1 Stochastic Gradient Descent

1.1 Implement in your favorite programming language the Stochastic Gradient Descent

We decided to use Python version 3 for this exercise.

1.2 Generate 100 artificial data points (x_i,y_i) where each x_i is randomly generated from the interval [0,1] and $y_i = sin(2pix_i) + e$. Here, e is a random noise value in the interval [-0.3,0.3]

In [104]:

```
import numpy as np
import math as m
import matplotlib.pyplot as plt
```

In [126]:

```
y_ones = np.ones(100)
x = np.random.uniform(size=100)
epsilon = np.random.uniform(low=-0.3,high=0.3,size=100)
alpha = 0.1
```

In [127]:

```
y = np.multiply(np.add(np.sin((2*np.pi*x)),epsilon),y_ones)
```

In [128]:

```
plt.plot(x, y, 'ro')
plt.axis([0, 1, -1.5, 1.5])
plt.show()

```

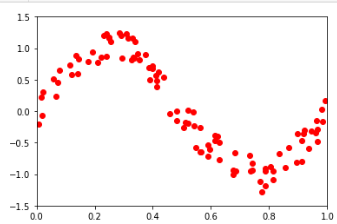

Figure 1: Plot of the sine function.

In 1 it is shown how x_i and y_i are generated. We used the framework Numpy to generate the 100 random datapoints x_i and y_i. After that we plotet y_i and interesting is the Sinus shape. This was also our imagination, because we used the sin function with respect to the noise. Here we have to add that the random.uniform range includes the first intervall parameter (low), but excludes the second one (high). To create the original range, we decided to set high to the last possible float represention after 1.0.

1.3 Make youre initial learning rate constant $\alpha = 0.1$, and train a polynomial model using your artificially created data. A polynomial model has the form $y = \Theta + \Theta \cdot x + \Theta \cdot x^2 + \cdots + \Theta_M \cdot x^M$

We implemenmted our polynomial Model and than we updated the Θ_i by our cost function:

$$J(\Theta) = \frac{1}{2}\sum_{i=1}^{m}[h_\theta(x^{(i)}) - y^{(i)}]^2$$

```
def init_func(degree):
    return np.random.uniform(low=-0.5, high=0.5, size=degree)
```

```
def eval_polynom(polynom, x_i):
    return sum([polynom[theta]*pow(x_i,theta) for theta in range(len(polynom))])

```

```
def learning_step(theta_index, polynom, seed_index):
    polynom[theta_index] = polynom[theta_index] + alpha *(y[seed_index]-
                                        eval_polynom(polynom,x[seed_index]))*(x[seed_index]**theta_index)
    return polynom
```

```
def error_function(polynom):
    return 0.5*sum([pow(eval_polynom(polynom,x[seed_index])- y[seed_index],2) for seed_index in range(len(x))])
```

```
def Stochastic_Gradient_descent(x,y,polynom,iteration):
    error_array = []
    for it in range(iteration):
        for seed_index in range(len(x)):
            for theta_index in range(len(polynom)):
                error_array.append(error_function(polynom))
                polynom = learning_step(theta_index, polynom, seed_index)
    return (polynom, error_array)



```

```
a = Stochastic_Gradient_descent(x,y,init_func(10),100)
```

Figure 2: Plot of the implemented Stochastic greedy and descent

1.4 All initial Θ_i parameters are randomly generetad in the interval [-0.5,0.5]

```
def init_func(degree):
    return np.random.uniform(low=-0.5, high=0.5, size=degree)
```

Figure 3: Function to decide the degree of the polynom

In the Figure it is shown that the Thetas are generated randomly in the interval [-0.5,0.5]. Also the parameter of init-func returns the degree of the polynom.

1.5 Try different α values to speed up the learning process

High α returned a bad approximation of the sine function. Also values above 0.1 showed good values, but it is important to find the best iteration step.

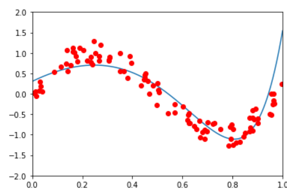

Figure 4: Plot for $alpha = 0.2$.

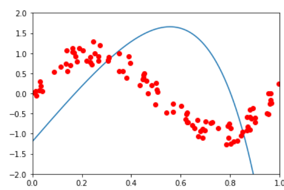

Figure 5: Plot for higher $alpha$.

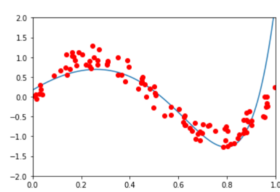

Figure 6: Plot for $alpha = 0.3$.

1.6 Once you have found the best model, plot the graph containing the data points, the sine function, and the learned function.

The following plots shows the results of our algorithmn. The higher the degree of the polynom the better results we achieved. But unfortunatly the runtime gets worse by higher degrees. If we remember correctly the degree of 3 by Prof. Dr. Victor Emanuel de Atocha Uc Cetina is in our example not sufficient to approximate the function closely enough. We have to add that we just iterated 100 times.

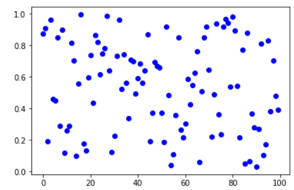

Figure 7: Plot of the random generated x values for the sin function

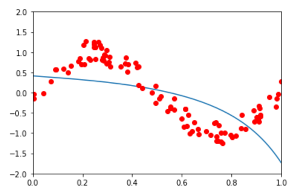

Figure 8: Plot of the initial function

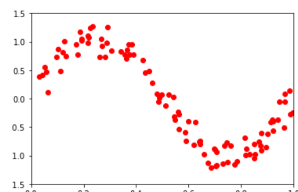

Figure 9: Plot of the sine function.

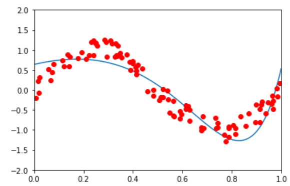

Figure 10: Plot of the learned function.

1.7 plot also the error curve.

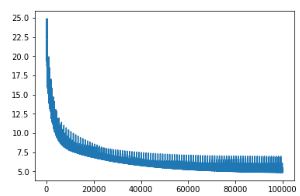

Figure 11: Plot of the error curve.

1.8 Prepare a report containing your final model, your final α value, and ure graph

Final Model where the Array[i] describes the θ_i

```
array([ 0.7045875 ,  1.21308849, -3.27177334, -2.82788017, -0.91965745,
        0.55362515,  0.86540899,  1.302885  ,  1.71905906,  1.39077429])
```

Figure 12: Plot of the Model with degree 9

$alpha = 0.1$ The final graph is given:

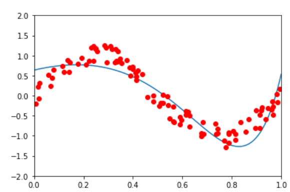

Figure 13: Plot of the final learned function.