Machine Learning Exercise 1:

Stochastic Gradient Descent

Juuso-Petteri Kuisma 7322902
bay1538@studium.uni-hamburg.de

(2) Your final model (polynomial fuction) with optimal learned parameters.

```
# Show predicted weights by printing them
```

```
[ 0.63343005]]), array([[ 0.38245452],[ 1.00951722],[-0.59427346],[-1.94113935],[ 0.64517249]])]
```

(3) Your final α value.

After about 40 iterations, the error rate doesn't change much at all. The optimal value for learning rate is about 0.01, but a change to either direction didn't impact the learned function or the error per iterations significantly. Further down the document I attached screenshots with different parametres based on the generated data following a sine wave.

```
learning_rate = 0.01
iterations = 60
```

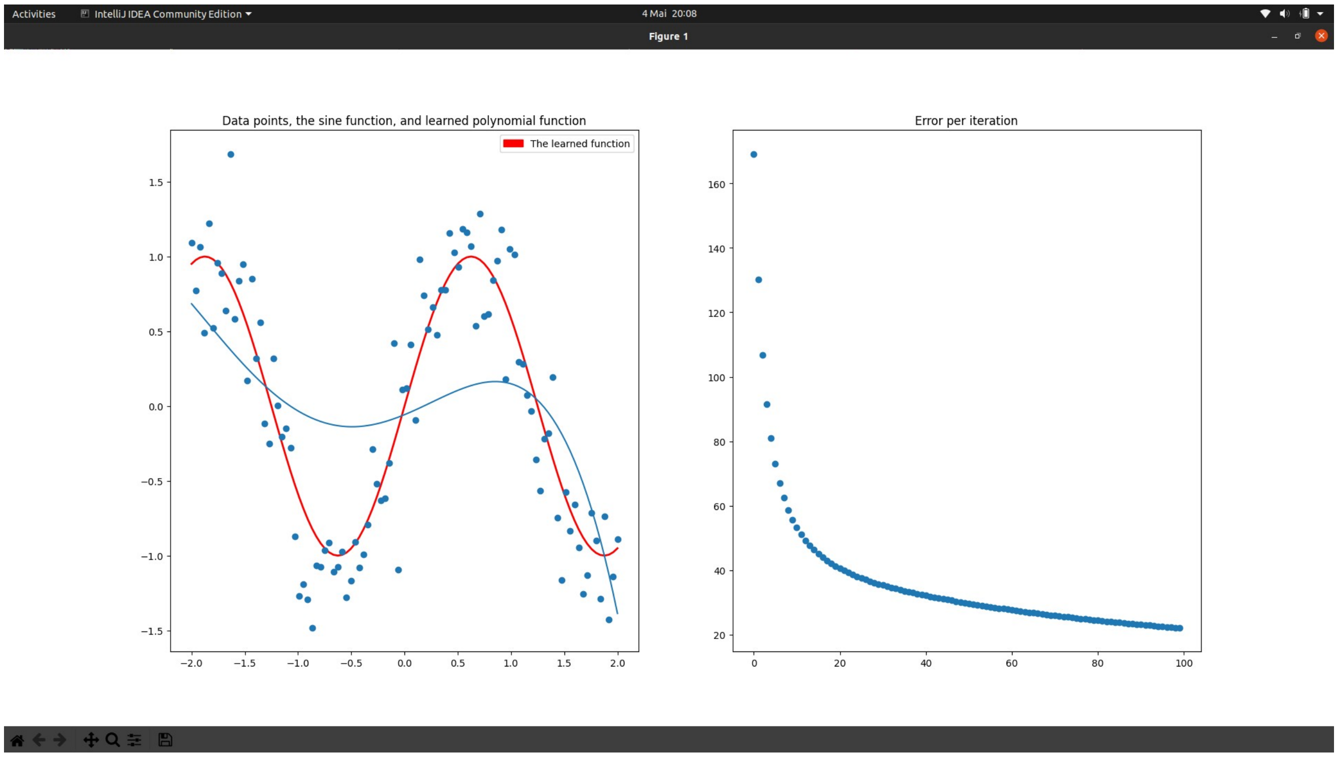
(4) One graph containing the cloud of the training data points, the sine function, and the learned polynomial function.

The first graph below shows the original data scattered around a sine wave with some noise, the original sine function fitted to the data, and the learned function highlighted in red.

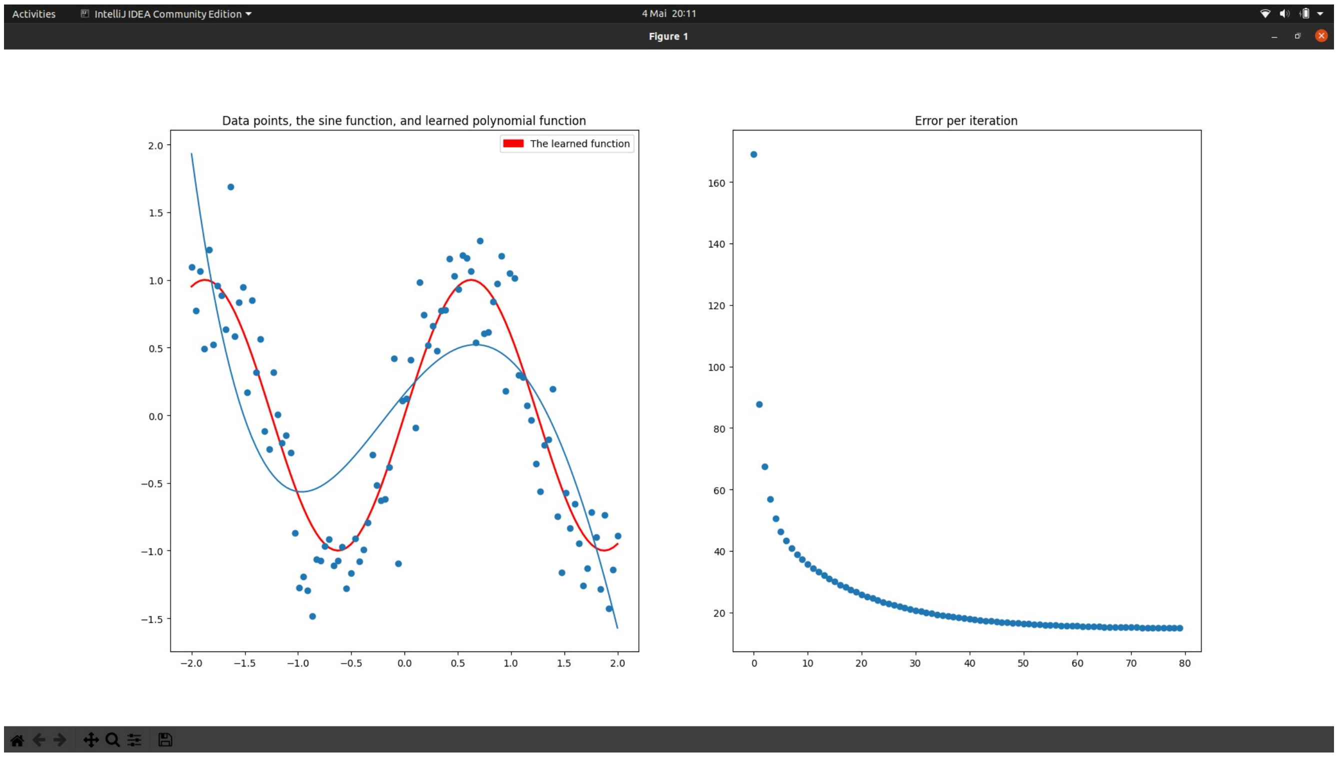
The graph on the right side shows how the number of iterations affect the errors. The error rate hits diminishing return fairly quickly, but seems to decline up until around 200 iterations. After that there aren't that discernible changes on how the learned function performs.

I also tried fairly high iterations, but evidently, the graph remains the same.

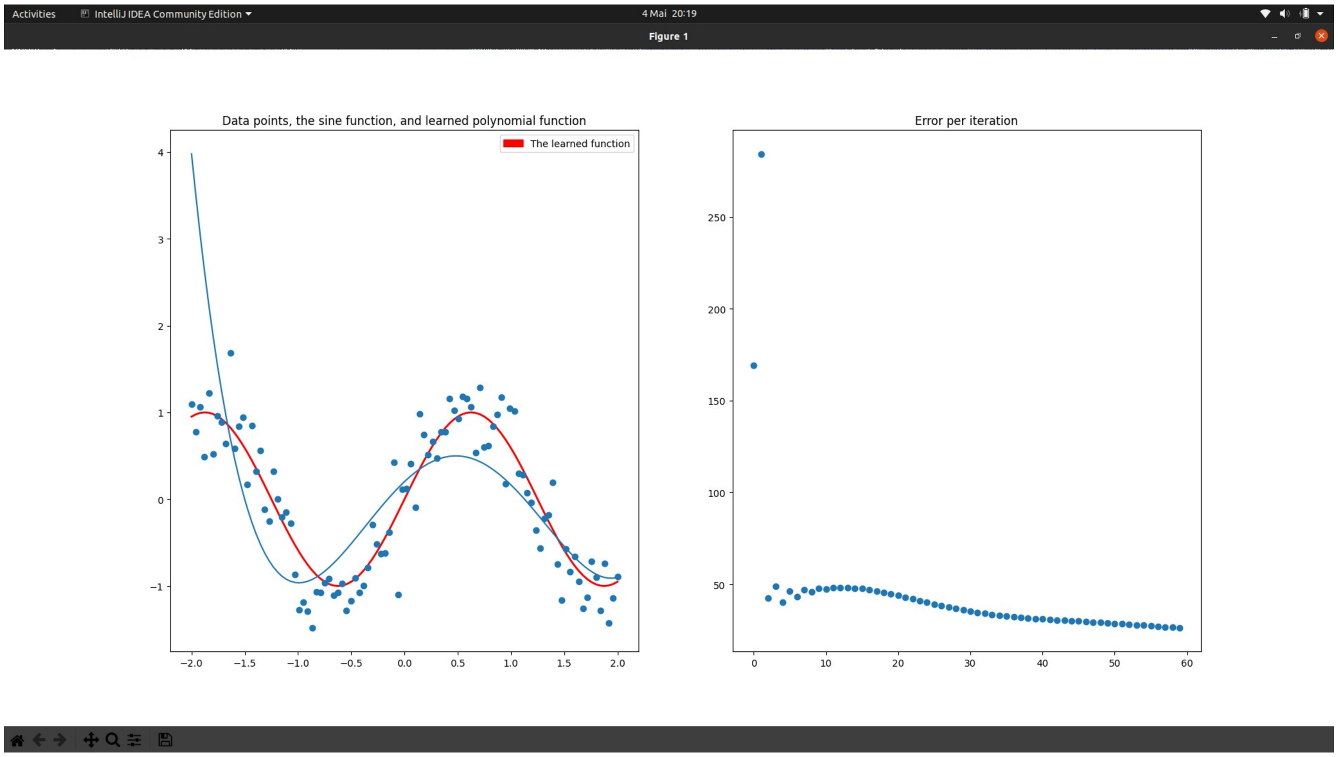
Learning rate 0.001 and 100 iterations


Activities
IntelliJ IDEA Community Edition
4 Mai 20:08
Figure 1
Data points, the sine function, and learned polynomial function
The learned function
Error per iteration

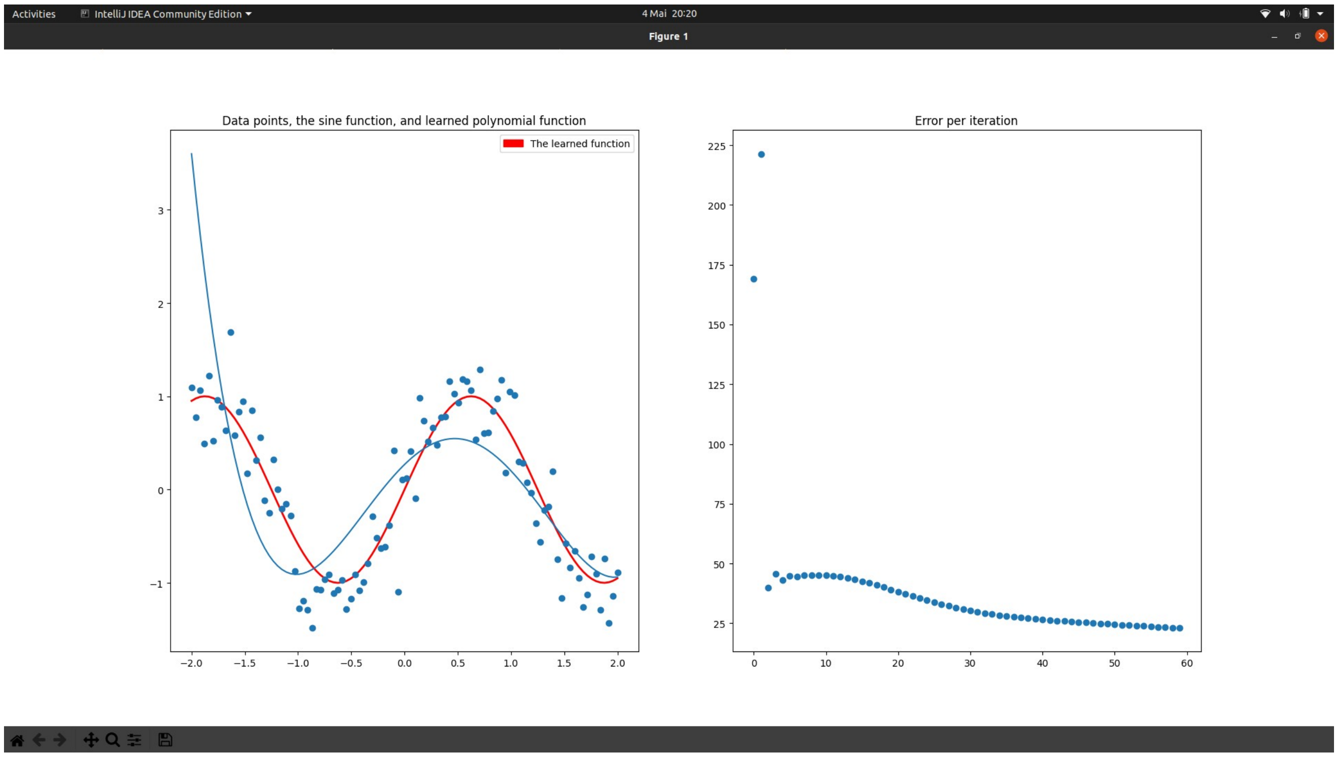
Learning rate 0,005 and 80 iterations


Activities
IntelliJ IDEA Community Edition
4 Mai 20:11
Figure 1
Data points, the sine function, and learned polynomial function
The learned function
Error per iteration

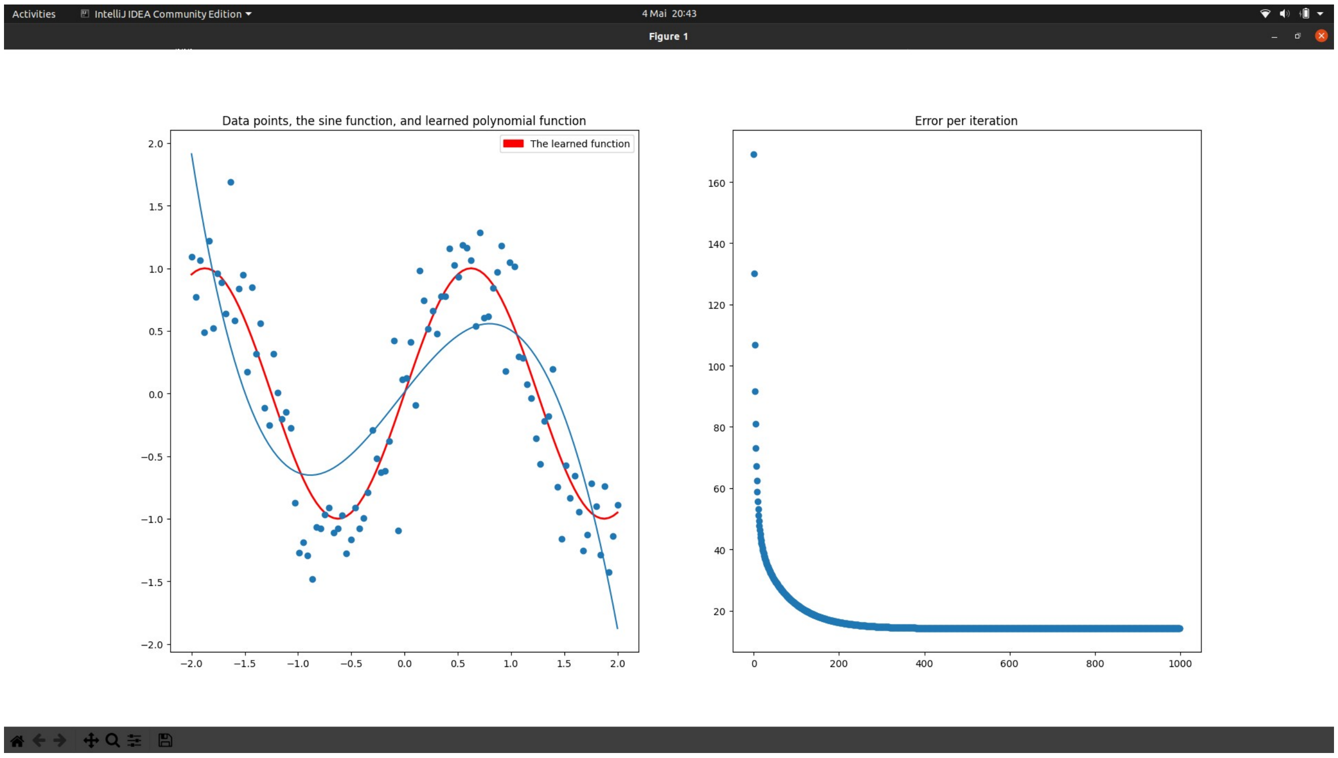
Learning rate 0.1 and 60 iterations


Activities
IntelliJ IDEA Community Edition
4 Mai 20:19
Figure 1
Data points, the sine function, and learned polynomial function
The learned function
Error per iteration

Learning rate 0.05 and 60 iterations


Activities
IntelliJ IDEA Community Edition
4 Mai 20:20
Figure 1
Data points, the sine function, and learned polynomial function
The learned function
Error per iteration

Learning rate 0.001 and 1000 iterations


Activities
IntelliJ IDEA Community Edition
4 Mai 20:43
Figure 1
Data points, the sine function, and learned polynomial function
The learned function
Error per iteration

Source code

```
# Stochastic gradient descent
# ML Exercise 1
# Juuso Kuisma
import numpy as np
import matplotlib.pyplot as plt
```

```python
import matplotlib.patches as mpatches

np.random.seed(seed=12)

def draw_plot(x, y, w, Error, w_s):
    '''
    A method for drawing the plots of the function.
    '''
    fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(28, 10))
    ax[0].plot(x[:, 1], [np.sin(0.8 * np.pi * x_i) for x_i in x[:, 1]], c='red', linewidth=2, zorder=0)
    red_patch = mpatches.Patch(color='red', label='The learned function')
    ax[0].legend(handles=[red_patch])
    ax[0].scatter(x[:, 1], y)
    ax[0].plot(x[:, 1], np.dot(x, w))
    ax[0].set_title('Data points, the sine function, and learned polynomial function')
    ax[1].scatter(range(iterations), Error)
    ax[1].set_title('Error per iteration')

    plt.show()

def generate_data():
    '''
    A method to generate and initialize the data used where the 100 data points take the
    form of a sine wave (4 degrees) and have a slight noise to the distribution.
    '''
    x = PolynomialFeatures(degree=4).fit_transform(np.linspace(-2, 2, 100).reshape(100, -1))
    x[:, 1:] = MinMaxScaler(feature_range=(-2, 2), copy=False).fit_transform(x[:, 1:])
    l = lambda x_i: np.sin(0.8 * np.pi * x_i)
    data = l(x[:, 1])
    # The noise of the distribution from [-0.3, 0-3]
    noise = np.random.normal(0, 0.3, size=np.shape(data))
    y = data + noise
    y = y.reshape(100, 1)
    # Normalize the data and return the generated data points
    return {'x': x, 'y': y}

def stochastic_gradient_descent(x, y, w, learning_rate):
    '''
    The Stochastic Gradient Descent method for solving the regression problem
    '''
    derivative = -(y - np.dot(w.T, x)) * x.reshape(np.shape(w))
    return learning_rate * derivative

# Initialize all of the variables for the function
data = generate_data()
# Tables for both x and y
x = data['x']
y = data['y']
# Initializing the function for the sine wave
w = np.random.normal(size=(np.shape(x)[1], 1))
# Learning rate and number of iterations
learning_rate = 0.001
iterations = 1000

# Update w based on the Stochastic Gradient Descent function
w_s = []
```

```
Error = []
for i in range(iterations):

    # Calculation for the error using the root-mean-square error
    error = (1 / 2) * np.sum([(y[i] - np.dot(w.T, x[i, :])) ** 2 for i in range(len(x))])
    # Append the error to the array
    Error.append(error)

    # Stochastic Gradient Descent function call where the given parametres are used
    for d in range(len(x)):
        w -= stochastic_gradient_descent(x[d, :], y[d], w, learning_rate)
        w_s.append(w.copy())

# Show predicted weights by printing them
print(w_s)

# Plot the predicted function and the error and the final learned function
draw_plot(x, y, w, Error, w_s)
```