Machine Learning for Scientists and Engineers

August 2024

Name	ID	Email
João Felipe Gueiros	941200164	joao.g@campus.technion.ac.il

1 Theory

1st Problem

Problem 2.1 To walk "downhill" on the loss function (equation 2.5), we measure its gradient with respect to the parameters ϕ_0 and ϕ_1. Calculate expressions for the slopes $\partial L/\partial\phi_0$ and $\partial L/\partial\phi_1$.

Solution:

$$\frac{\partial L}{\partial \phi_0} = \frac{\partial \sum_{i=1}^{I}(\phi_0 + \phi_1 x_i - y_i)^2}{\partial \phi_0} = \frac{\sum_{i=1}^{I} \partial(\phi_0 + \phi_1 x_i - y_i)^2}{\partial \phi_0} = \sum_{i=1}^{I}(2\phi_0 + 2\phi_1 x_i - 2y_i)$$

$$\frac{\partial L}{\partial \phi_1} = \frac{\partial \sum_{i=1}^{I}(\phi_0 + \phi_1 x_i - y_i)^2}{\partial \phi_1} = \frac{\sum_{i=1}^{I} \partial(\phi_0 + \phi_1 x_i - y_i)^2}{\partial \phi_1} = \sum_{i=1}^{I}(2\phi_0 x_i + 2\phi_1 x_i^2 - 2y_i x_i)$$

2nd Problem

Problem 2.2 Show that we can find the minimum of the loss function in closed form by setting the expression for the derivatives from problem 2.1 to zero and solving for ϕ_0 and ϕ_1. Note that this works for linear regression but not for more complex models; this is why we use iterative model fitting methods like gradient descent (figure 2.4).

Solution:

Let's set equate the expressions to 0:

$$\frac{\partial L}{\partial \phi_0} = \sum_{i=1}^{I}(2\phi_0 + 2\phi_1 x_i - 2y_i) = 0 \tag{1}$$

$$\frac{\partial L}{\partial \phi_1} = \sum_{i=1}^{I}(2\phi_0 x_i + 2\phi_1 x_i^2 - 2y_i x_i) = 0 \tag{2}$$

Remark: I'll just set the $\sum_{i=1}^{I}$ to $\sum$ to shorten my typing.
Back to equation (1)

$$\sum(y_i) = \sum \phi_0 + \sum \phi_1 x_i \Rightarrow \sum \phi_0 = \sum y_i - \phi_1 \sum x_i \Rightarrow I \cdot \phi_0 = \sum y_i - \phi_1 \sum x_i$$

$$\phi_0 = \frac{\sum y_i - \phi_1 \sum x_i}{I}$$

Now, we must find ϕ_1
Reorganizing equation (2), we get:

$$\phi_1 = \frac{\sum y_i x_i - \phi_0 \sum x_i}{\sum x_i^2}$$

Substituting ϕ_0 from before we get:

$$\phi_1 = \frac{\sum y_i x_i - \frac{\sum y_i - \phi_1 \sum x_i}{I} \sum x_i}{\sum x_i^2}$$

Reorganizing for ϕ_1

$$\phi_1 = \frac{I \sum y_i x_i - \sum x_i \sum y_i}{I \sum x_i^2 - (\sum x_i)^2}$$

□

2 Computation

Question 1.

HW2 > hw_2_question_1.py > ...

```python
import numpy as np
import matplotlib.pyplot as plt
# Getting data
x = np.array([1 , 2, 3, 4, 5, 6])
y = np.array([1 , 3, 2, 5, 4, 6])
data = np.concatenate((x[np.newaxis, :], y[np.newaxis, :]), axis=0)
## Calculating stuff
x_avg = np.mean(x)
y_avg = np.mean(y)
beta_1_num = np.sum((x - x_avg) * (y - y_avg))
beta_1_den = np.sum((x - x_avg) ** 2)
beta_1 = beta_1_num / beta_1_den
beta_0 = y_avg - beta_1*x_avg
x_line = np.linspace(1,6,100)

y_line = beta_1*x_line + beta_0
plt.plot(x_line,y_line,"-r",label = "Best line")
plt.plot()
plt.scatter(x, y, color='blue', label='Points')
plt.grid()
plt.xlabel('x')
plt.ylabel('y')
plt.title('Linear regression for blue points')
plt.show()
```

Figure 1: Python program to implement linear regression

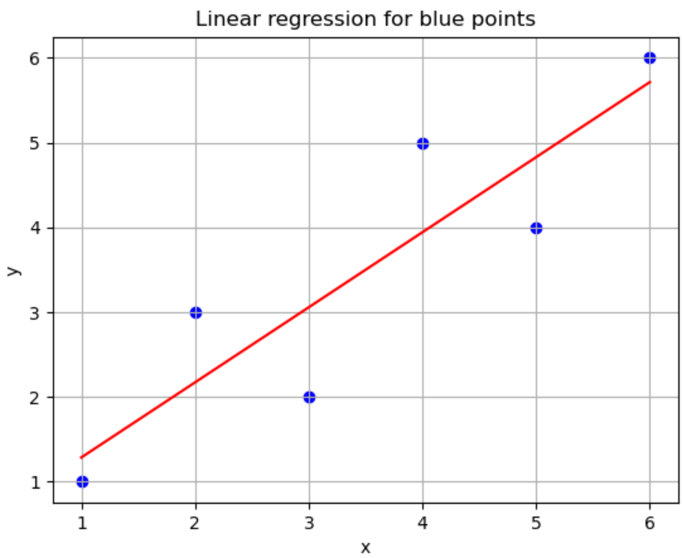


Figure 2: Plot of linear regression

Question 2.

```python
import numpy as np
import matplotlib.pyplot as plt
x = np.array([1 , 2, 3, 4, 5, 6])
y = np.array([1 , 3, 2, 5, 4, 6])

def cost_function(beta_0,beta_1,x,y):
    h = beta_0*1 + beta_1*x
    J = 1/(2*len(x)) * np.sum((h - y)**2)

    return J

def gradient_descent(alpha,iter,beta_0,beta_1,x,y,x0):
    cost = np.zeros(iter)
    for i in range(iter):
        h = beta_0*1 + beta_1*x
        gradient_0 = 1/(len(x)) * np.sum(h-y) * x0
        beta_0 = beta_0 - alpha*gradient_0
        gradient_1 = 1/(len(x)) * np.sum((h-y)*x)
        beta_1 = beta_1 - alpha*gradient_1
        cost[i] = cost_function(beta_0,beta_1,x,y)

    return [cost,beta_0,beta_1]

alpha = 0.01
iter = 1000
lins = np.linspace(1,iter,iter)
x0 = 1
beta_0 = 1
beta_1 = 1
[cost,beta_0,beta_1] = gradient_descent(alpha,iter,beta_0,beta_1,x,y,x0)
x_line = np.linspace(1,6,100)
y_line = beta_1*x_line + beta_0

# Figure 1: Plot of cost
plt.figure(1)  # Start a new figure
plt.plot(lins, cost, label="Cost")
plt.grid()
plt.xlabel('Iterations')
plt.ylabel('Cost')
plt.title('Cost Convergence')
plt.legend()
plt.show()

# Figure 2: Linear regression fit
plt.figure(2)  # Start a new figure
plt.plot(x_line, y_line, "-r", label="Best Line")  # Best-fit line
plt.scatter(x, y, color="blue", label="Data Points")  # Data points
plt.grid()
plt.xlabel("x")
plt.ylabel("y")
plt.title("Linear Regression for Blue Points")
plt.legend()
plt.show()
```

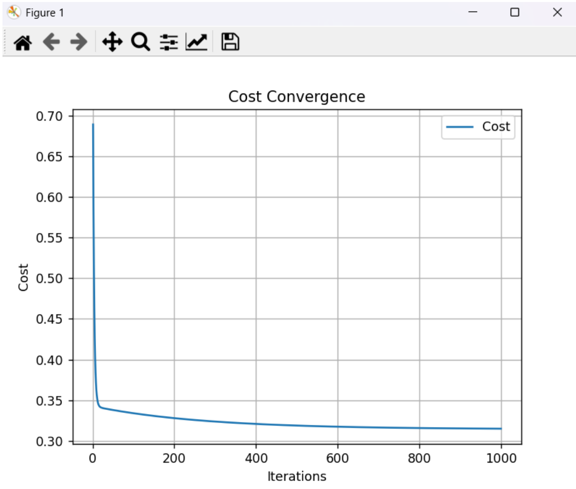


Figure 3: Cost convergence plot with $\alpha = 0.01$

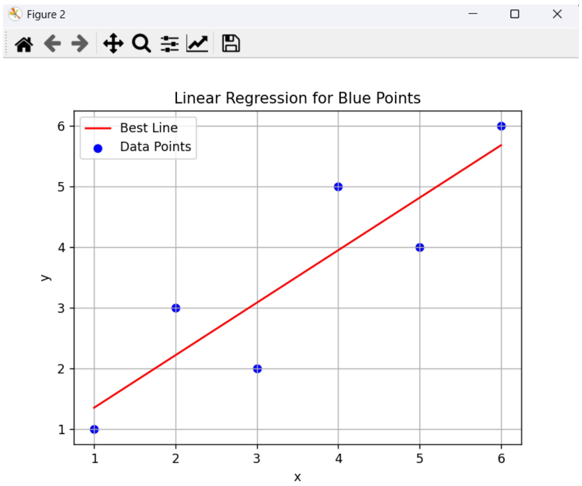


Figure 4: Linear regression with gradient descent plot

Question 4.

```python
import torch
import torch.nn as nn

# Define neural network
class Question4(nn.Module):
    def __init__(self, in_dim=2, HL1_dim=2, out_dim=1):
        super().__init__()
        self.output = nn.Sequential(
            nn.Linear(in_dim, HL1_dim), nn.Sigmoid(),
            nn.Linear(HL1_dim, out_dim), nn.Sigmoid()
        )

```

```python
        # Initialize weights and biases for the first layer
        self.output[0].weight.data = torch.tensor([[0.1, 0.3], [0.2, 0.4]])  # W1
        self.output[0].bias.data = torch.tensor([0.5, 0.6])  # B1

        # Initialize weights and biases for the second layer
        self.output[2].weight.data = torch.tensor([[0.7, 0.8]])  # W2
        self.output[2].bias.data = torch.tensor([0.9])  # B2

    def forward(self, x):
        return self.output(x)

# Input tensor
x = torch.tensor([[0.3, 0.7]])

# Target value
target = torch.tensor([[1.0]])  # Expected output

# Define model
model = Question4()

# Perform forward pass
output = model(x)

# Compute error using the given formula
error = 0.5 * ((output - target) ** 2).sum()  # Squared error

# Print output and error
print(f"Output: {output.item()}")
print(f"Error: {error.item()}")
```

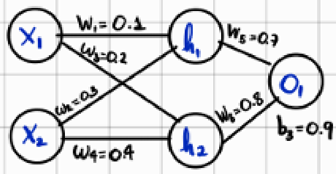
```
PS C:\Users\Joao Felipe\OneDrive - Technion\Documents\Winter 2024 - 2025\Machine Learning> & C:/anaconda3/python.exe "c:/Users/Joao Felipe/OneDrive - Technion/Documents/Winter 2024 - 2025/Machine Learning/HW2/hw_2_question_4.py"
Output: 0.8753568530082703
Error: 0.007767957169562578
PS C:\Users\Joao Felipe\OneDrive - Technion\Documents\Winter 2024 - 2025\Machine Learning>
```

Figure 5: Result of error

I obtained a very similar result to the hand calculation, probably this slight difference is due to my approximations in the hand calculation.

Question 3.

Written solution in the following 2 pages



$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^T = [0.3 \;\; 0.7]$$

$$W_1 = \begin{bmatrix} w_1 & w_2 \\ w_3 & w_4 \end{bmatrix} = \begin{bmatrix} 0.1 & 0.2 \\ 0.3 & 0.4 \end{bmatrix}$$

$$B_1 = [b_1 \;\; b_2] = [0.5 \;\; 0.6]$$

$$W_2 = \begin{bmatrix} w_5 \\ w_6 \end{bmatrix} = \begin{bmatrix} 0.7 \\ 0.8 \end{bmatrix}$$

$$B_2 = [b_3] = [0.9]$$

θ_{ij}, i is the # of the hidden unit and j is # of the input

h_i, i is the # of the hidden unit

$$h_d = a\left[\theta_{d0} + \sum_{i=1}^{D_i (\#\text{ inputs})} \theta_{di} x_i\right], \quad y_j = \phi_{j0} + \sum_{d=1}^{D} \phi_{jd} h_d$$

$$h_1 = a[\theta_{10} + \theta_{11} x_1 + \theta_{21} x_2]$$

$$h_2 = a[\theta_{20} + \theta_{21} x_1 + \theta_{22} x_2]$$

$$y_0 = \phi_{10} + \phi_{01} h_1 + \phi_{02} h_2$$

Figure 6: Question 3 solution part 1

So, $h_1 = a[b_1 + w_1 x_1 + w_2 x_2]$

$h_2 = a[b_2 + w_3 x_1 + w_4 x_2]$

$$\Rightarrow \begin{array}{l} h_1 = a[0.5 + 0.1 \cdot 0.3 + 0.3 \cdot 0.7] = a[0.74] \\ h_2 = a[0.6 + 0.2 \cdot 0.3 + 0.4 \cdot 0.7] = a[0.94] \end{array}$$

$$\Rightarrow h_1 = \frac{1}{1+e^{-0.74}} \simeq 0.677$$

$$h_2 = \frac{1}{1+e^{-0.94}} \approx 0.719$$

So,

$$O_0 = a[y_0] = a[\phi_{10} + \phi_{01} h_1 + \phi_{02} h_2]$$

$$\Rightarrow y_0 = b_3 + w_5 h_1 + w_6 \cdot h_2 = 0.9 + 0.7 \cdot (0.677) + 0.8(0.719)$$

$$\Rightarrow y_0 = 1.941$$

$$O_0 = a[1.941] = \frac{1}{1+e^{-1.941}} \approx 0.874$$

$$\text{error} = \frac{1}{2}(0.874 - 1)^2 \approx 0.00788$$

Figure 7: Question 3 solution part 2