HW3: Gradient descent

2082014 김은심

Problem 1:

Your goal is to minimize the following loss function

$L(\beta_0, \beta_1) = (\beta_0 - 3)^2 + 2.4(\beta_1 - 5)^2 + 10$

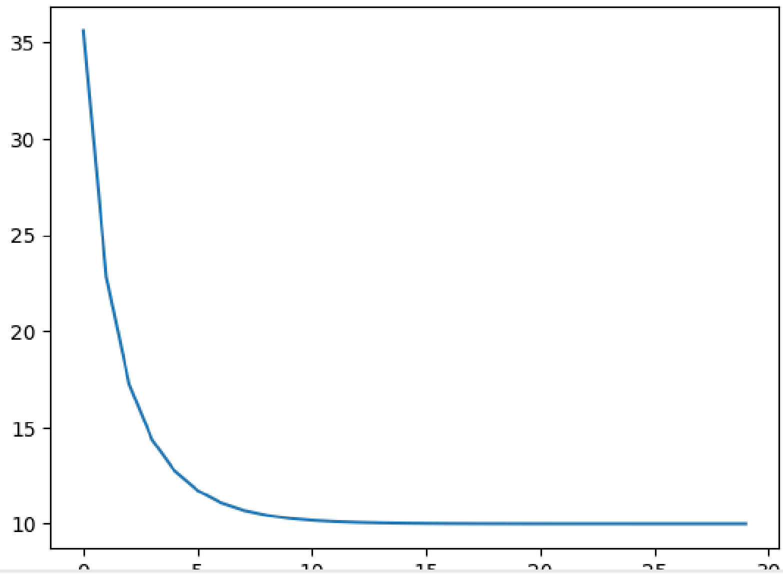
Using the following specification, find β_0, β_1 which minimize the loss function.

Instruction:

- Start with initial values $\beta_0 = 7.0$ and $\beta_1 = 3.0$.
- Run the gradient descent algorithm for proper amount of iterations and track the loss values at each step. Plot the learning curve to see whether the learning is enoughly done.

```
import torch
import numpy as np
import matplotlib.pyplot as plt

beta=torch.tensor([7.0, 3.0], requires_grad=True)



epochs=30
lr=0.1
history=[]

for i in range(epochs):
  beta.grad=None
  y = (beta[0]-3)**2+2.4*(beta[1]-5)**2+10

  y.backward()
  beta.data=beta.data-lr*beta.grad
  history.append(y.item())

plt.plot(history)
beta.data
```

```
tensor([3.0050, 5.0000])
```



Problem 2:

You are given a dataset with two input features X_1 and X_2, and an output Y. Your task is to fit a linear regression model using gradient descent to predict Y based on X_1 and X_2.

The model can be represented as:

$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2$.

The goal is to minimize the mean squared error between the predicted values $\widehat{Y}$ and the true values Y.

Your dataset is given by

```
X1 = torch.tensor([0.83, -0.18,  0.27,  0.32,  0.25, -0.73,  0.19, -0.08, -0.47, -0.28, -0.23,  0.25, 0.15,  0.92,  1.27, -0
X2 = torch.tensor([0.63, -1.43, -1.35,  0.70, -1.48,  0.06,  1.14,  0.96, -1.65,  0.44,  0.62,  0.47, -0.08,  1.03,  1.24, -
Y = torch.tensor([0.50, 1.34, 1.39, 0.40, 1.20, 0.11, -0.21, -0.26, 1.18, 0.59, 0.13, 0.17, 0.55, 0.32, 0.19, 0.46, 0.32, -0
```

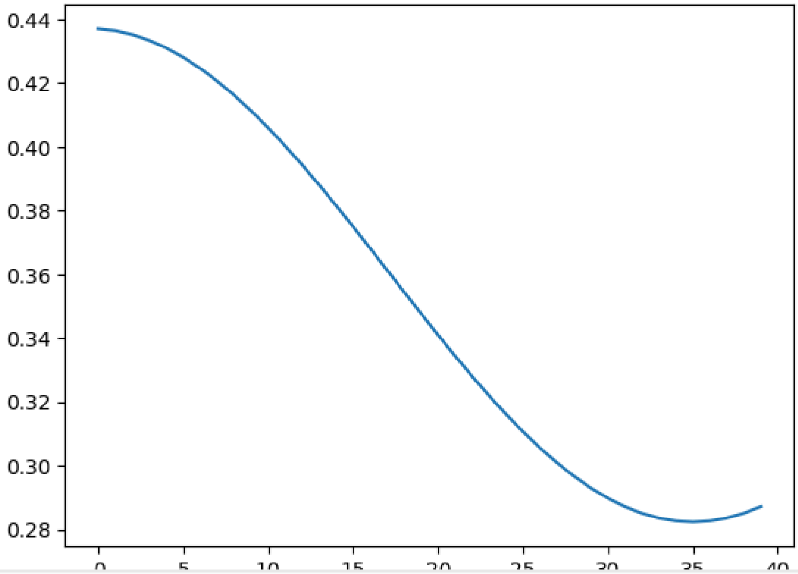
The following is the step-by-step instruction. Complete the code.

```
X1 = torch.tensor([0.83, -0.18,  0.27,  0.32,  0.25, -0.73,  0.19, -0.08, -0.47,
X2 = torch.tensor([0.63, -1.43, -1.35,  0.70, -1.48,  0.06,  1.14,  0.96, -1.65,
Y = torch.tensor([0.50, 1.34, 1.39, 0.40, 1.20, 0.11, -0.21, -0.26, 1.18, 0.59,
```

```
#Step 1: Reshape the Data

ones = torch.ones([20])
X=torch.stack([ones,X1,X2],axis=1)
X.shape

#step2: Initialize the Parameters
beta = torch.tensor([[0.1],[-0.035],[0.12]],requires_grad=True)
beta.shape

#Step 3: Define the Prediction Function
def predict(X, beta):
  yhat=X@beta
  return yhat

#Step 4: Define the Loss Function
```

```
def mse_loss(Y, yhat):
   temp = torch.mean((Y-yhat)**2)
   return temp

#Step 5: Implement Gradient Descent

lr=0.001
history=[]
epochs=40

for i in range(epochs):
  yhat=X@beta
  loss=mse_loss(Y,yhat)
  loss.backward()
  beta.data=beta.data-lr*beta.grad
  history.append(loss.item())
plt.plot(history)
beta.data

```

```
tensor([[ 0.5680],
        [-0.0054],
        [-0.0033]])
```



1 코딩을 시작하거나 AI로 코드를 생성하세요.

Problem 3:

You are given the cubic equation:

$x^3 - 6x^2 + 11x - 6 = 0.$

Use gradient descent to find all the real solutions to this equation.

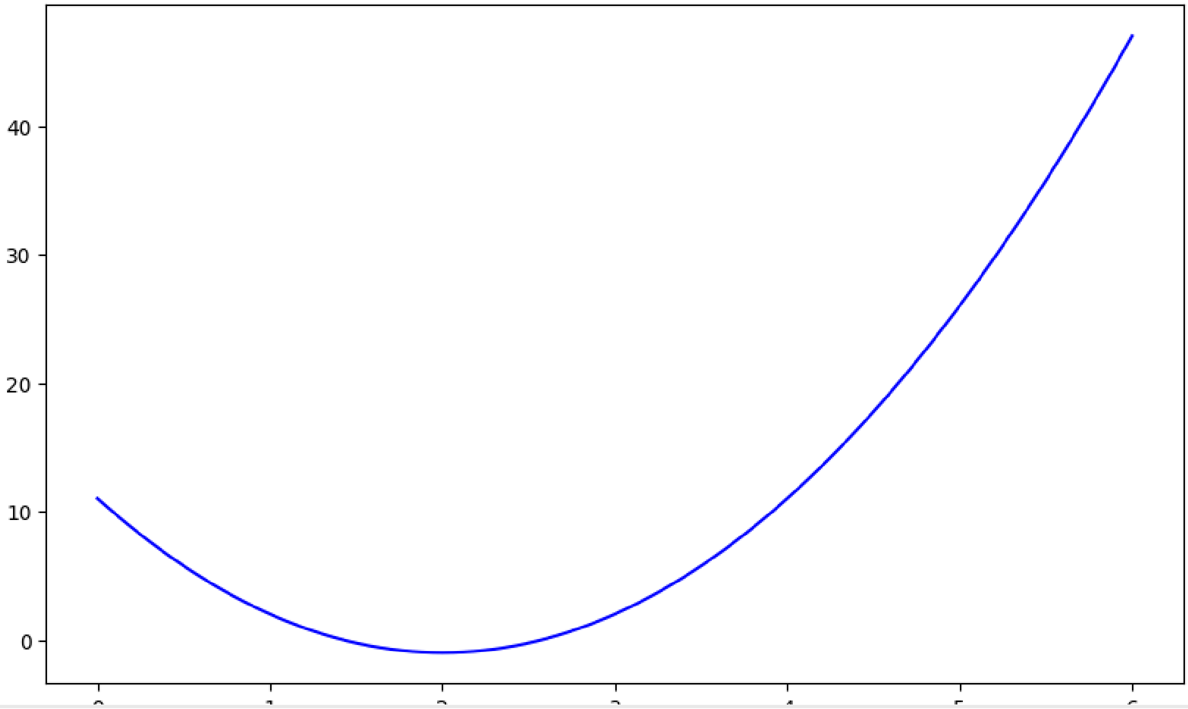
Hint:

- It might be helpful to draw the graph of $y = x^3 - 6x^2 + 11x - 6$.

```
def f(x):
    return 3*x**2 - 12*x + 11

# Generate x values
x = np.linspace(0, 6, 100)  # Range of x values
```

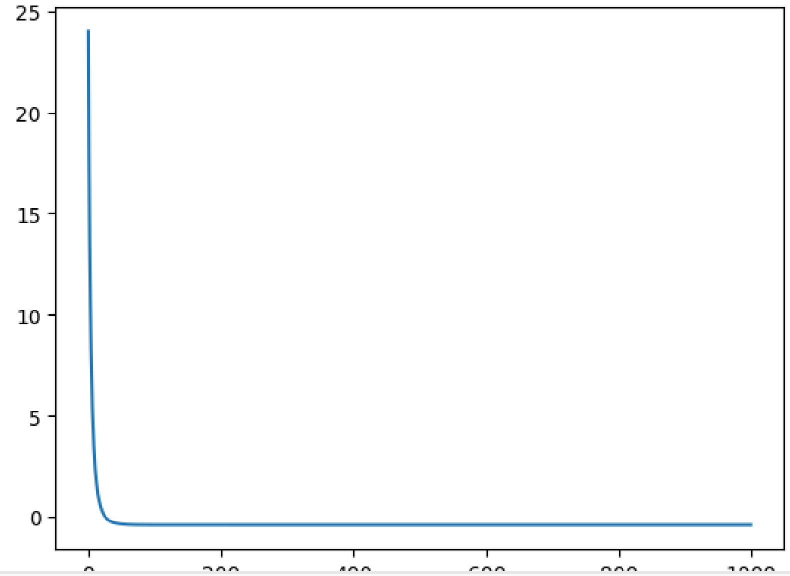
```
y = f(x)  # Calculate corresponding y values

# Create the plot
plt.figure(figsize=(10, 6))
plt.plot(x, y, label='y = x^3 - 6x^2 + 11x - 6', color='blue')
```

```
[<matplotlib.lines.Line2D at 0x7bd0be11df60>]
```




```
#optimization

def loss_fns(x):
  y=x**3-6*x**2+11*x-6
  return y

x=torch.tensor(5.0,requires_grad=True)

lr=0.01
epochs=1000
history=[]
for i in range(epochs):
  y=loss_fns(x)
  x.grad=None
  y.backward()
  x.data=x.data-lr*x.grad
  history.append(y.item())


print(x.data)

plt.plot(history)

```

```
tensor(2.5774)
[<matplotlib.lines.Line2D at 0x7bd0bd9dff70>]
```



Problem 4:

You are given $X \sim \text{Gamma(shape = 2.0, rate = 1.3)}$.

Answer the following questions.

1. Find $x_0 = \hat{x}_0$ such that $P(X < x_0) = 0.95$. Note that x_0 is called as the 95-th quantile of $\text{Gamma(shape = 2.0, rate = 1.3)}$.
2. Confirm that your answer in part 1 is correct by calculating $P(X < \hat{x}_0)$.
3. Calculate $E[X^3]$ by simulating 10000 samples of X.

Hint 1:

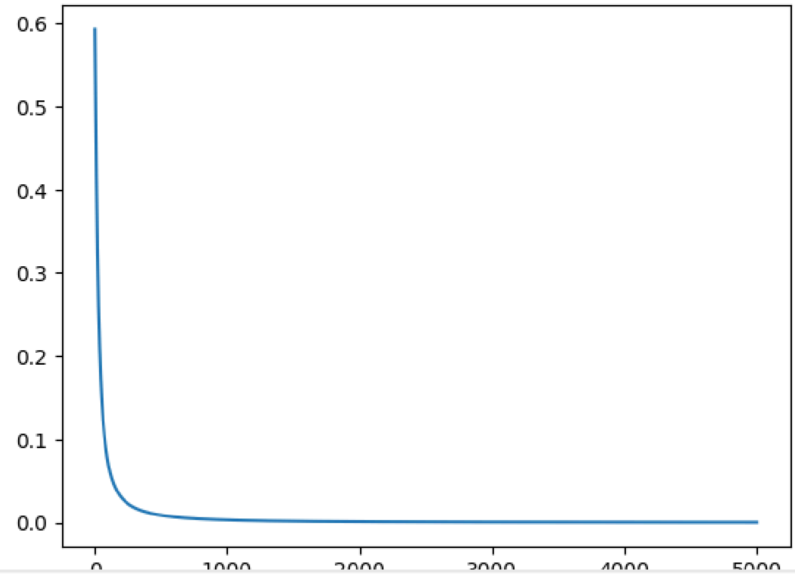
You may use the following loss function

$L(x_0) = (P(X < x_0) - 0.95)^2$.

```
import torch

# Define the parameters for the Gamma distribution
shape = 2.0
rate = 1.3

# Create a Gamma distribution object
gamma_dist = torch.distributions.Gamma(torch.tensor([shape]), torch.tensor([1/ra

# Print the Gamma distribution
print(gamma_dist)



#b. Confirm that your answer in part 1 is correct by calculating  P(X<x^0) .
def loss_fns(x):
  y=(gamma_dist.cdf(x)-0.95)**2
  return y

x=torch.tensor([1.0],requires_grad=True)
history=[]
lr=0.1
epochs=5000
```

```

for i in range(epochs):
  y=loss_fns(x)
  x.grad=None
  y.backward()
  x.data=x.data-lr*x.grad
  history.append(y.item())


print(x.data)

plt.plot(history)
x.data
#C. Calculate  E[X3]  by simulating 10000 samples of  X .
# 3)
x=gamma_dist.sample([10000])
torch.mean(x**3)
```

```
Gamma(concentration: tensor([2.]), rate: tensor([0.7692]))
tensor([5.8759])
tensor(54.0150)
```



Step 2: Calculating the CDF

The **Cumulative Distribution Function (CDF)** gives the probability that a random variable is less than or equal to a certain value. In PyTorch, you can calculate the CDF of the Gamma distribution using the `cdf()` method.

Let's calculate the CDF of the Gamma distribution at `θ_0 = 2.0`:

```
#2)
# Define the value at which to compute the CDF
theta_0 = torch.tensor([2.0])

# Calculate the CDF of the Gamma distribution at theta_0
cdf_value = gamma_dist.cdf(theta_0)

# Print the result
print(f"CDF at theta_0 = 2.0: {cdf_value.item()}")

```

```
CDF at theta_0 = 2.0: 0.45496392250061035
```

Problem 5:

You are given $X \sim$ Exp(mean = 2.0).

Answer the following questions.

1. Find $x_0 = \hat{x}_0$ such that $P(X < x_0) = 0.90$.
2. Confirm that your answer in part 1 is correct by calculating $P(X < \hat{x}_0)$.
3. Calculate $E[X]$ by simulating 10000 samples of X.

```
#1.

# Step 1: Define parameters for the Exponential distribution
mean = 2.0
lambda_ = 1 / mean  # Rate parameter
x_0 = -2 * torch.log(torch.tensor(0.10))  # Using ln(0.10)
print(x_0)

#2.
# Calculate x0 such that P(X < x0) = 0.90

#define Exp distribution
Exp=torch.distributions.Exponential(rate=lambda_)

def loss_fns(x):
  y=(Exp.cdf(x)-0.95)**2
  return y
x=torch.tensor([3.0],requires_grad=True)
history=[]
lr=0.0112
epochs=10900
for i in range(epochs):
  y=loss_fns(x)
  x.grad=None
  y.backward()
  x.data=x.data-lr*x.grad
  history.append(y.item())


print(x.data)

plt.plot(history)


#3. Simulate 10,000 samples and calculate E[X]
num_samples = 10000
samples = torch.distributions.Exponential(rate=lambda_).sample((num_samples,))
E_X = torch.mean(samples)
print(E_X)
```

```
tensor(4.6052)
tensor([4.5601])
tensor(2.0101)
```