```
#!/usr/bin/env python
# coding: utf-8



# Import Numpy & PyTorch
import torch
import numpy as np
import matplotlib.pyplot as plt
from IPython import get_ipython
get_ipython().magic('reset -sf')


# Define the data
T = np.array([1,1])
T = T.reshape(2,1)
n = 100
X = 10 * np.random.rand(n,1)
b = np.ones_like(X)
X1 = np.hstack((b,X))
X1_d = X1.astype(np.float32)
F1 = np.dot(X1, T)
eps = np.random.randn(n,1)
Y1 = F1 + eps
Y1_d = Y1.astype(np.float32)

# Closed form
h = X1.transpose() @ X1
h_inv = np.linalg.inv(h)
p = np.dot(X1.transpose(), Y1)
theta_closed =  h_inv @ p
y1_closed = np.dot(X1, theta_closed)

error = np.sqrt(1/n*np.dot((F1-y1_closed).transpose(),(F1-y1_closed)))
print('Matrix inverion solution error = ',error)

# Plot data points
fig, ax = plt.subplots()
ax.scatter(X,Y1)
ax.plot(X,y1_closed, color = 'k')
ax.plot(X,F1, color = 'r')
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Linear Regression")
plt.show

# # Define the data
# a = torch.ones(1)
# x = 10*torch.rand((100,1))
# a0 = torch.randn(100,1)
```

```
# y = torch.matmul(x,a) + a0

# Define PyTorch tensors
X_tens = torch.tensor(X)
X1_tens = torch.tensor(X1)
#X1_tens = x
X1_d_tens = torch.from_numpy(X1_d)
F1_tens = torch.from_numpy(F1)
Y1_tens = torch.from_numpy(Y1)
Y2_d_tens = torch.from_numpy(Y1_d)
#Y1_tens = y


# Define model inputs and targets and initalize weights and bias
#inputs_d = X1_d_tens
inputs = X1_d_tens
inputs2 = X1_tens
targets = Y1_tens
targets2 = Y2_d_tens
w = torch.randn(1,2, requires_grad=True)




# Define the model
def model(x):
    return x@w.t()

# Define loss function
def mse(t1, t2):
    diff = t1 - t2
    return torch.sum(diff * diff) / diff.numel()


# Iterate and modify via gradient decent
for i in range(100):
    preds = model(inputs)
    loss = mse(preds, targets)
    loss.backward()
    with torch.no_grad():
        w -= w.grad * .01
        w.grad.zero_()

y1_grad = preds

error =
np.sqrt(1/n*np.dot((F1-y1_grad.detach().numpy()).transpose(),(F1-y1_grad.detach().nu
mpy())))
print('Manual model solution error = ',error)
```

```
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader


train_ds = TensorDataset(inputs, targets2)


# Define data loader
batch_size = 5
train_dl = DataLoader(train_ds, batch_size, shuffle=True)



# Define model
model2 = nn.Linear(2, 1)



# Define optimizer
opt = torch.optim.SGD(model2.parameters(), lr=.001)



# Import nn.functional
import torch.nn.functional as F



# Define loss function
loss_fn = F.mse_loss
loss = loss_fn(model2(inputs), targets)


# Define a utility function to train the model
def fit(num_epochs, model, loss_fn, opt, inputs):
    for epoch in range(num_epochs):
        for xb,yb in train_dl:
            # Generate predictions
            pred = model(xb)
            loss = loss_fn(pred, yb)
            # Perform gradient descent
            loss.backward()
            opt.step()
            opt.zero_grad()
    return model(inputs)


# Train the model for 100 epochs
y2_grad = fit(100, model2, loss_fn, opt, inputs)
```

```
error =
np.sqrt(1/n*np.dot((F1-y2_grad.detach().numpy()).transpose(),(F1-y2_grad.detach().nu
mpy())))
print('Pytorch model solution error = ',error)

class SimpleNet(nn.Module):
    # Initialize the layers
    def __init__(self):
        super().__init__()
        self.linear1 = nn.Linear(2, 2)
        self.act1 = nn.ReLU() # Activation function
        self.linear2 = nn.Linear(2, 1)

    # Perform the computation
    def forward(self, x):
        x = self.linear1(x)
        x = self.act1(x)
        x = self.linear2(x)
        return x



model3 = SimpleNet()
opt = torch.optim.SGD(model3.parameters(), .001)
loss_fn = F.mse_loss


y3_network = fit(100, model3, loss_fn, opt, inputs)

error =
np.sqrt(1/n*np.dot((F1-y3_network.detach().numpy()).transpose(),(F1-y3_network.detac
h().numpy())))
print('Single layer neural network solution error = ',error)


# Plot data points
fig, bx = plt.subplots()
bx.scatter(X,Y1)
bx.plot(X,y1_closed, color='k')
bx.plot(X,y1_grad.detach().numpy(), color='g')
bx.plot(X,y2_grad.detach().numpy(), color='c')
bx.plot(X,y3_network.detach().numpy(), color='m')
bx.plot(X,F1, color='r')
plt.xlabel("X")
plt.ylabel("Y")
plt.title("Linear Regression")
bx.legend(['Matrix Inversion', 'Manual Model', 'PyTorch Model', 'Single Layer Neural
Net', 'True','Data'])
plt.show
```