Regression from Scratch in Numpy vs. PyTorch

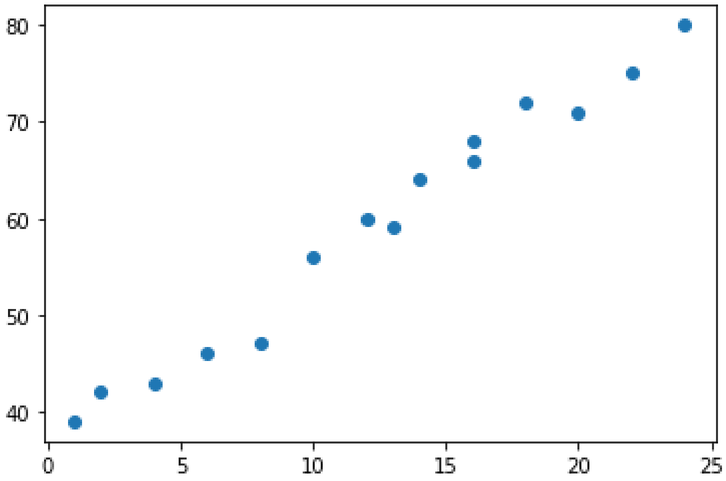
Regression in Numpy

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

```
X = np.array([1,2,4,6,8,10,12,13,14,16,16,18,20,22,24])
Y = np.array([39,42,43,46,47,56,60,59,64,66,68,72,71,75,80])
```

```
plt.scatter(X,Y)
```

```
<matplotlib.collections.PathCollection at 0x7fc3f3b97c18>
```



Saved successfully! ×

ip between X and Y.(We'll discuss more about correlation ir

```
'''Y = a*X+b is the equation of line/linear regression model.
Goal here is to find the values of a and b.
There are multiple techniques to achieve this:
1.Matrix calculations: Put all data into matrices to perform optimization.Used for small d
2.Gradient Descent : Try to minimize error/difference between actual and predicted values
3.Regularization: While minimizing error,also try to reduce impact of unnecessary features
4.Simple linear regression:If there are single input varaible and single output variable,u

More detailed explaination of above techniques is not in the scope here.
We'll implement method 2 i.e Gradient Descent here-more specific-Batch Gradient Descent.
Weights(a,b) are updated at end of complete batch/all rows as follow:
new a = old a - (learning_rate*gradient_a)
new b = old b - (learning_rate*gradient_b)

'''
```

```
"Y = a*X+b is the equation of line/linear regression model.\nGoal here is to find the
```

```
np.random.seed(2)
epochs=15
learning_rate = 0.001
w = np.random.randn()
b = np.random.randn()
y_pred = np.empty(len(Y))


for i in range(epochs):
    print("-----------epoch:{}--------".format(i))
    #prediction
    y_pred = w*X +b

    #Error/loss calculation is Mean Squared Error
    error = np.mean((Y - y_pred)**2)
    print('Total Error:{}'.format(error))

    #Gradient calculation
    gradient_a = np.mean(-2*X*(Y-y_pred))
    gradient_b = np.mean(-2*(Y-y_pred))

    #Update weights
    w -= learning_rate*gradient_a
    b -= learning_rate*gradient_b
```

Saved successfully! ✕

```
print(w,b)
```

```
4.042799282999869 0.4771951521774575
```

```
          epoch:2
```

```
'''Error is reducing with increment in epochs. Number of epochs and learning rate are hype
Let's not play around with it and jumpt to PyTorch'''
```

```
"Error is reducing with increment in epochs. Number of epochs and learning rate are h
```

```
-----------epoch:7--------
```

Regression in PyTorch

```
Total Error:317.6516893015321
```

```
import torch
```

```
-----------epoch:10--------
```

```
#initialise data/features and target
X_tensor = torch.from_numpy(X)
Y_tensor = torch.from_numpy(Y)
```

```
Total Error:315.47667816740153
```

```
#Initialise weights
'''Here unlike numpy we have to mention that these variables are trainable(need to calcula
This can be done using requires_grad:'''
```

```
'Here unlike numpy we have to mention that these variables are trainable(need to calc
```

```
torch.random.seed = 2
                    ad=True,dtype=torch.float)
                    ad=True,dtype=torch.float)
```

Saved successfully! ×

```
learning_rate = 0.001


w_tensor
```

```
tensor([-0.6845], requires_grad=True)
```

```
#Model without PyTorch in-built methods
for i in range(epochs):
    print("-----------epoch:{}--------".format(i))
    #prediction
    y_pred = w_tensor*X_tensor +b_tensor

    #Error/loss calculation is Mean Squared Error
    error = ((Y_tensor - y_pred)**2).mean()
    print('Total Error:{}'.format(error))

    '''Now no need to calculate gradients,PyTorch will do it if we tell which function/var
    error.backward()

    '''Actual values of gradients can be seen using grad attribute'''
    #print(w_tensor.grad,b_tensor.grad)

    '''We can not directly use gradients in normal calculation,so use no_grad() method to

    with torch.no_grad():
       w_tensor-= learning_rate*w_tensor.grad
       b_tensor-= learning_rate*b_tensor.grad


     #After each step,Reinitilaise gradients because PyTorch holds on to gradients
    w_tensor.grad.zero_()
    b_tensor.grad.zero_()
```

```
-----------epoch:0--------
Total Error:4754.9716796875
-----------epoch:1--------
Total Error:1877.7296142578125
-----------epoch:2--------
Total Error:858.9075317382812
-----------epoch:3--------
Total Error:498.0286560058594
-----------epoch:4--------
Total Error:370.0838623046875
-----------epoch:5--------
Total Error:324.6056823730469
-----------epoch:6--------
Total Error:308.32354736328125
-----------epoch:7--------
Total Error:302.3780822753906
-----------epoch:8--------
Total Error:300.0922546386719
-----------epoch:9--------
```

Saved successfully! ×

```
Total Error:298.5710754394531
-----------epoch:11--------
Total Error:298.20263671875
-----------epoch:12--------
Total Error:297.8919372558594
-----------epoch:13--------
Total Error:297.6018371582031
-----------epoch:14--------
Total Error:297.3192443847656
```

```
#Model with PyTorch in-built methods
optimizer = torch.optim.SGD([w_tensor, b_tensor], lr=learning_rate)
loss = torch.nn.MSELoss(reduction='mean')
for i in range(epochs):
    print("-----------epoch:{}--------".format(i))
    #prediction
    y_pred = w_tensor*X_tensor +b_tensor

    #Error/loss calculation is Mean Squared Error
    error = loss(Y_tensor, y_pred)
    print('Total Error:{}'.format(error))
    '''Now no need to calculate gradients,PyTorch will do it if we tell which function/var
    error.backward()

    #Update weights using Optimizer
    optimizer.step()

    #After each step,Reinitilaise gradients because PyTorch holds on to gradients
    #Reinitilaise gradients using Optimizer
    optimizer.zero_grad()
```

```
-----------epoch:0--------
Total Error:297.0393981933594
-----------epoch:1--------
Total Error:296.7607727050781
-----------epoch:2--------
Total Error:296.48272705078125
-----------epoch:3--------
Total Error:296.2050476074219
-----------epoch:4--------
Total Error:295.927734375
-----------epoch:5--------
Total Error:295.6506652832031
-----------epoch:6--------
Total Error:295.3738098144531
-----------epoch:7--------
Total Error:295.0972595214844
-----------epoch:8--------
Total Error:294.8209533691406
-----------epoch:9--------
Total Error:294.5448913574219
-----------epoch:10--------
Total Error:294.2691345214844
-----------epoch:11--------
Total Error:293.99365234375
```

Saved successfully!

```
           epoch:13
Total Error:293.4433898925781
-----------epoch:14--------
Total Error:293.16864013671875
```

```
'''Till now,we've explored loss calculation and Optimizers.
The only manual step remaining is prediction step. Let's remove that also'''
```

```
"Till now,we've explored loss calculation and Optimizers.\nThe only manual step remai
```

```
#Create Network by extending parent nn.Module.
'''We have to implement __init__ and forward methods '''
class Network(torch.nn.Module):
  def __init__(self):
    super().__init__()
    #Intialise parameters whcih should be trained. Note that parameters need to be wrapped
    self.w_tensor = torch.nn.Parameter(torch.randn(1,requires_grad=True,dtype=torch.float)
    self.b_tensor = torch.nn.Parameter(torch.randn(1,requires_grad=True,dtype=torch.float)
  def forward(self,x):
    #Output prediction calculation
    return  w_tensor*x +b_tensor


#Model with PyTorch in-built methods
model = Network()
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
loss = torch.nn.MSELoss(reduction='mean')
for i in range(epochs):
    print("-----------epoch:{}--------".format(i))
    #This will not do actual training but will set model in training mode.
    model.train()

    #prediction
    y_pred = model(X_tensor)

    #Error/loss calculation is Mean Squared Error
    error = loss(Y_tensor, y_pred)
    print('Total Error:{}'.format(error))

    '''Now no need to calculate gradients,PyTorch will do it if we tell which function/var
    error.backward()

    #Update weights using Optimizer
    optimizer.step()

    #After each step,Reinitilaise gradients because PyTorch holds on to gradients
    #Reinitilaise gradients using Optimizer
    optimizer.zero_grad()
```

Saved successfully! ×

```
-----------epoch:0--------
Total Error:288.8082275390625
-----------epoch:1--------
Total Error:288.8082275390625
-----------epoch:2--------
Total Error:288.8082275390625
-----------epoch:3--------
Total Error:288.8082275390625
-----------epoch:4--------
Total Error:288.8082275390625
-----------epoch:5--------
Total Error:288.8082275390625
-----------epoch:6--------
Total Error:288.8082275390625
-----------epoch:7--------
Total Error:288.8082275390625
-----------epoch:8--------
Total Error:288.8082275390625
-----------epoch:9--------
Total Error:288.8082275390625
-----------epoch:10--------
Total Error:288.8082275390625
-----------epoch:11--------
Total Error:288.8082275390625
-----------epoch:12--------
Total Error:288.8082275390625
-----------epoch:13--------
Total Error:288.8082275390625
-----------epoch:14--------
Total Error:288.8082275390625
```

```
'''To summarize,following are steps for model creation PyTorch:
1.Create Model class in which   init__() method contains trainable parameter and forward m
                                   tion

    model.train()--- Set model in training mode
    pred = model(X)-- Prediction
    loss = LossFunction(pred,actual)-- Loss calculation
    loss.backward()-- Gradient calculation
    optimizer.step()-- Update weights/parameters
    optimizer.zero_grad()-- Reset gradients'''
```

Saved successfully!

```
'To summarize,following are steps for model creation PyTorch:\n1.Create Model class i
```

Saved successfully!