Agenda

- Introduction and expectations
- Explain Linear Regression with an example. (30 - 45 Mins)
- Set up. (15 Minutes)
- Participants to implement a programme on Linear Regression with a different dataset which they will create. (60 + Minutes)
- Refactor code with Optimizer. (15-20 Minutes), Optional
- Participants to experiment with Learning rate, epochs (15 mins) , Optional
- Discussion (Industrial Use cases - How to solve - Challenges) 5 minutes
- Roadmap for you.
- Feedback / Q & A

Introduction and expectations

- Introductions of Participants, understand their expectation.
- Facilitator Expectations - Code heavy session. Do more, talk less.
- Please do not ask for code to be given to paste and run. This session is purposely designed for participants to understand and write every line of code.

Some word of advice

- Some concepts may not be clear immediately, with time, things normally fall into place.
- They will become clear when you practice more.
- We will start with a Simple Example.
- After this session, you should go back and practice other type of problems.

What is Data Science

Finding Insights from given data. This is not only predictive modelling.

Some Real world examples - Device Remaining Useful life, Image Identification.

Various Algorithms can be used for various use cases.

What is a model - (algo + data)

Linear Regression is one of the most popular algorithms in Predictive Learning.

Logistic Regression is used for Binary Classification problems.

What is Linear Regression

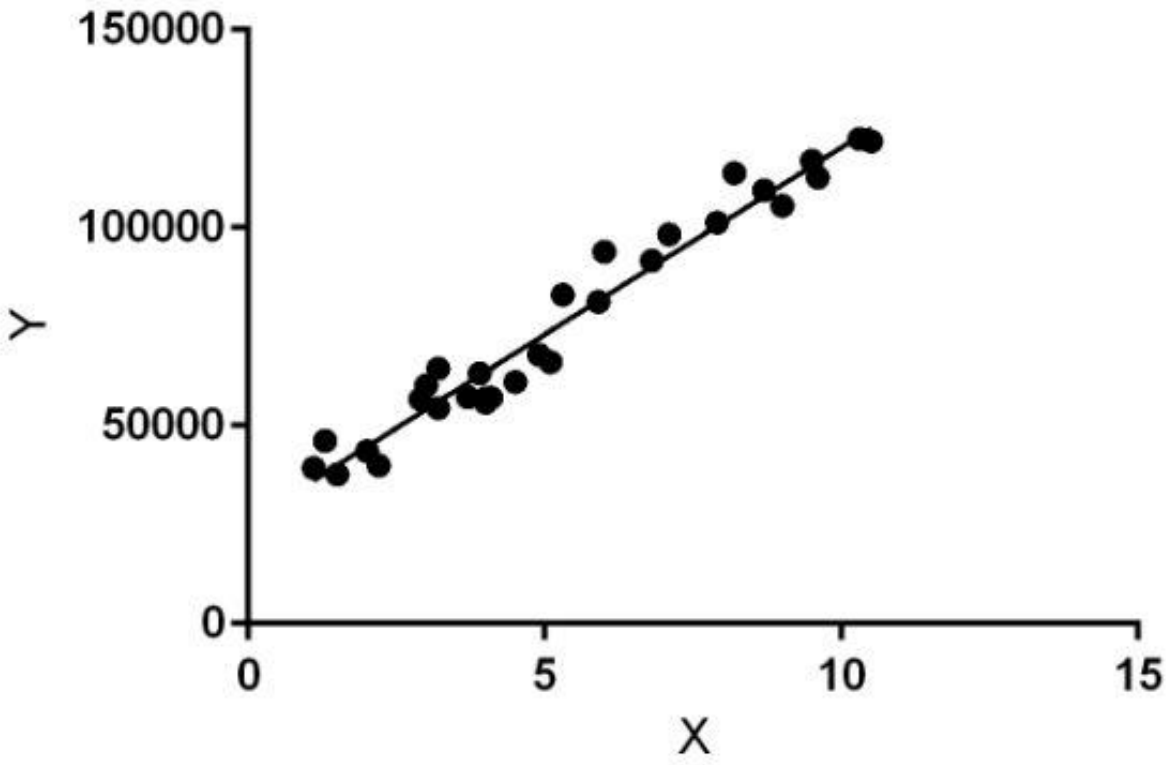
Explains linear relationship between two or variables.

These are called as Response and Predictor, Dependent and Independent.

Two parameters, (w and b) define this relationship.

In a machine learning scenario, a model is something which has these two parameters defined and they have those values, which explains the model the best.

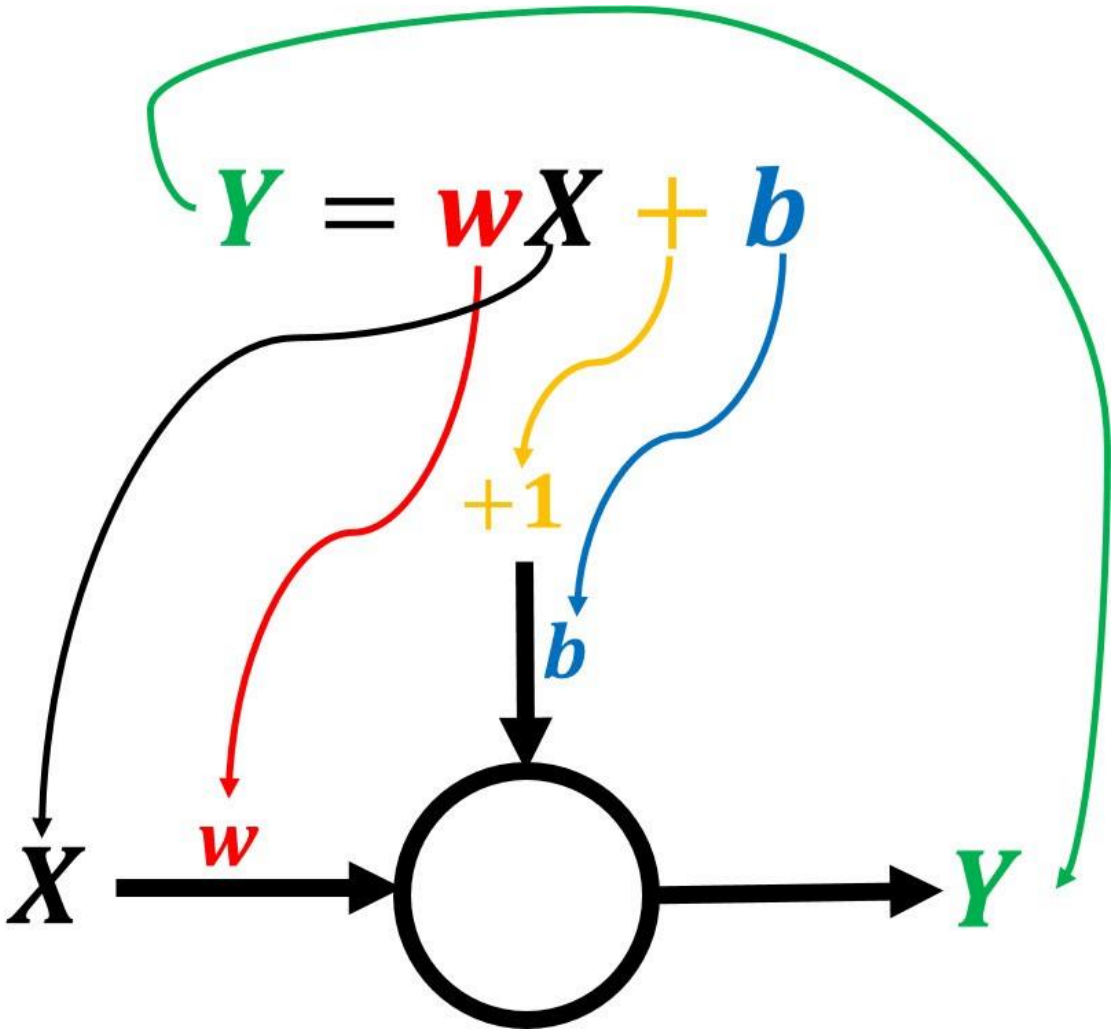
Above is for one input, for multiple inputs, parameters increase.



Some real world use cases of Linear Regression

- Output is a continuous variable.
- Remaining Useful Life of an equipment.
- Age of a person from an image
- No of visitors in a restaurant
- Sales of an Item
- No of warranty claims invoked in a repair shop.

Weight and Bias



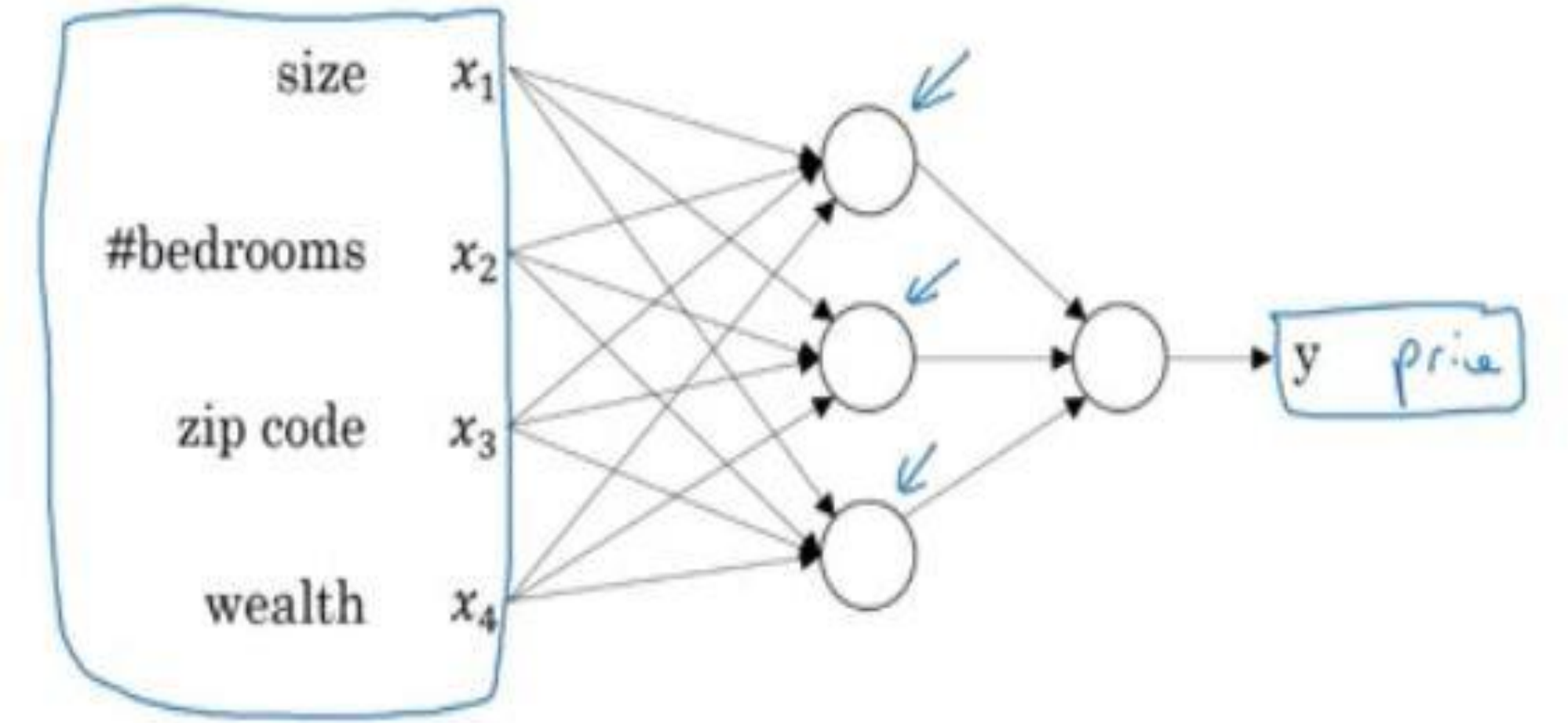
Explain Linear Regression with code

Linear Regression Demo with code

https://colab.research.google.com/drive/1j8U35VjzFHs3u4sKuogVws_fpngSUTgr?usp=sharing

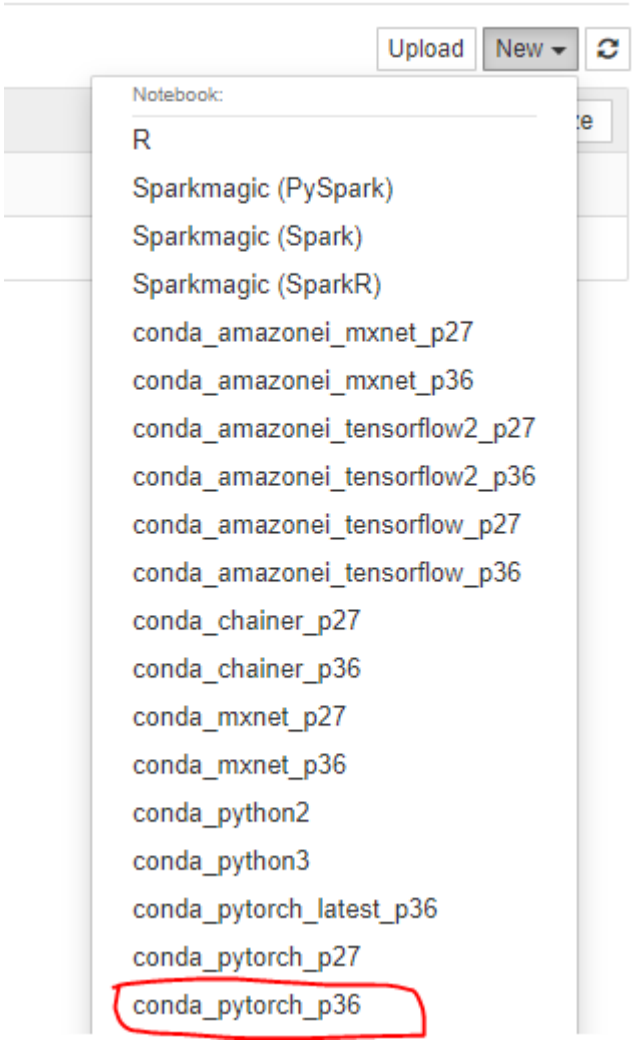
https://docs.google.com/spreadsheets/d/1wp-H47co8i-Yh35xPNi5OrFftkLBoTCBOtfm7vsj6mc/edit?usp=sharing

A Neural Network for multiple inputs



Set up and access AWS (Sagemaker)

Create a notebook and name it as <YourName>Example 1



Linear Regression

Participants to Write Now

Concepts to cover

- Learning Rate (LR and Alpha)
- Gradient Descent
- Epochs
- Weights and Bias

Importing Libraries

import torch

import torch.nn as nn

import matplotlib.pyplot as plt

import numpy as np

Generating Data

```
X = torch.randn(100, 1)*10

y = X + 3*torch.randn(100, 1)

plt.plot(X.numpy(), y.numpy(), 'o')

plt.ylabel('y')

plt.xlabel('x')
```

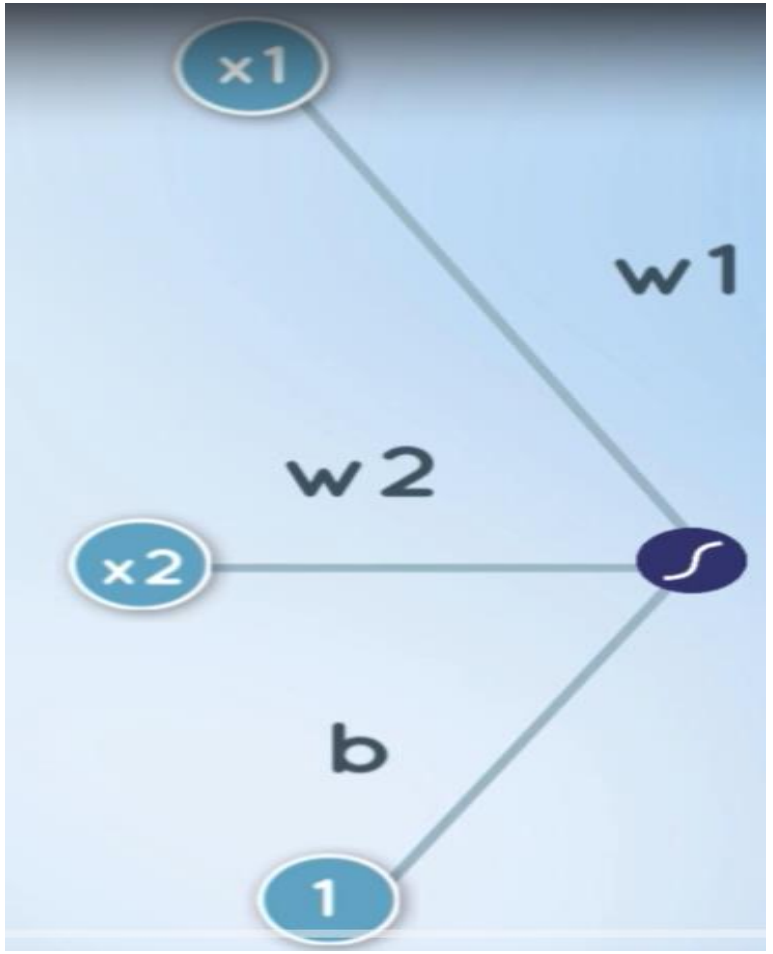
Defining Model

```
class Linear_Regression(nn.Module):
    def __init__(self, input_size, output_size):
        super().__init__()
        self.linear = nn.Linear(input_size, output_size)
    def forward(self, x):
        pred = self.linear(x)
        return pred
```

Check Model Parameters

```
torch.manual_seed(100)
model = LR(1,1)   use Linear_Regression
for param in model.parameters():
    print(param)
```

```
Parameter containing:
tensor([[-0.7767]], requires_grad=True)
Parameter containing:
tensor([0.6317], requires_grad=True)
```



Prediction from untrained model (random weights)

```
2 * -0.7767 + 0.6317
```

```
-0.9216999999999999
```

```
model(torch.tensor(2.0).unsqueeze(-1))
```

```
tensor([-0.9217], grad_fn=<AddBackward0>)
```

Loss function /Cost

used in Model Training.

We need a variable to track if the model is getting better.

Idea is to keep this to minimum when training a neural model.

Gradient Descent is used for training.

```
def mse(y_hat, y): return ((y_hat-y)**2).mean()
```

Gradient Descent

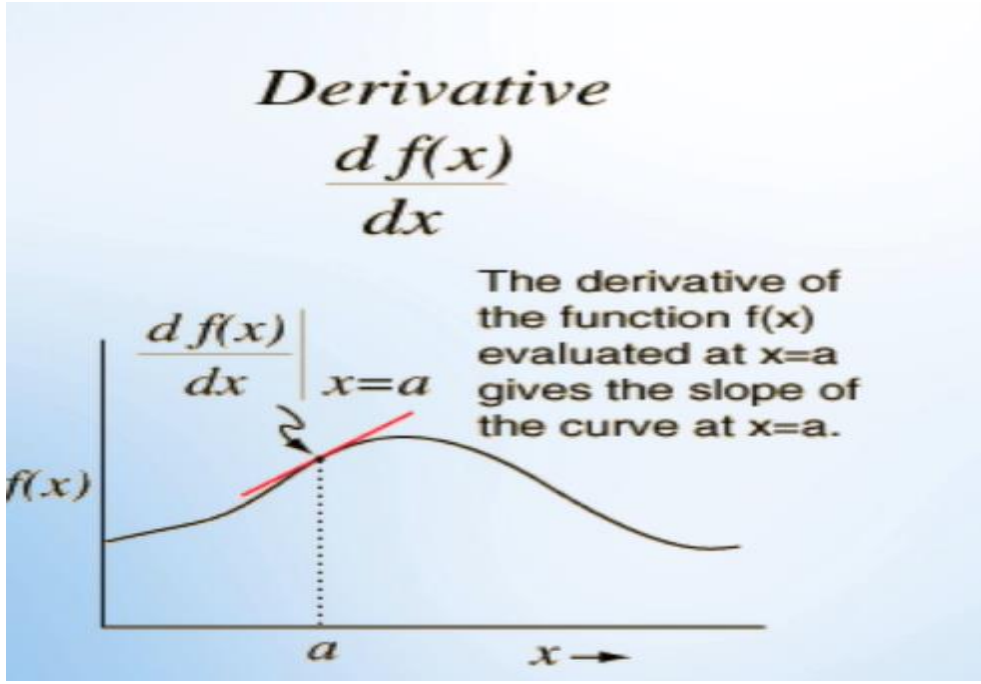
The derivative at a point gives the slope.

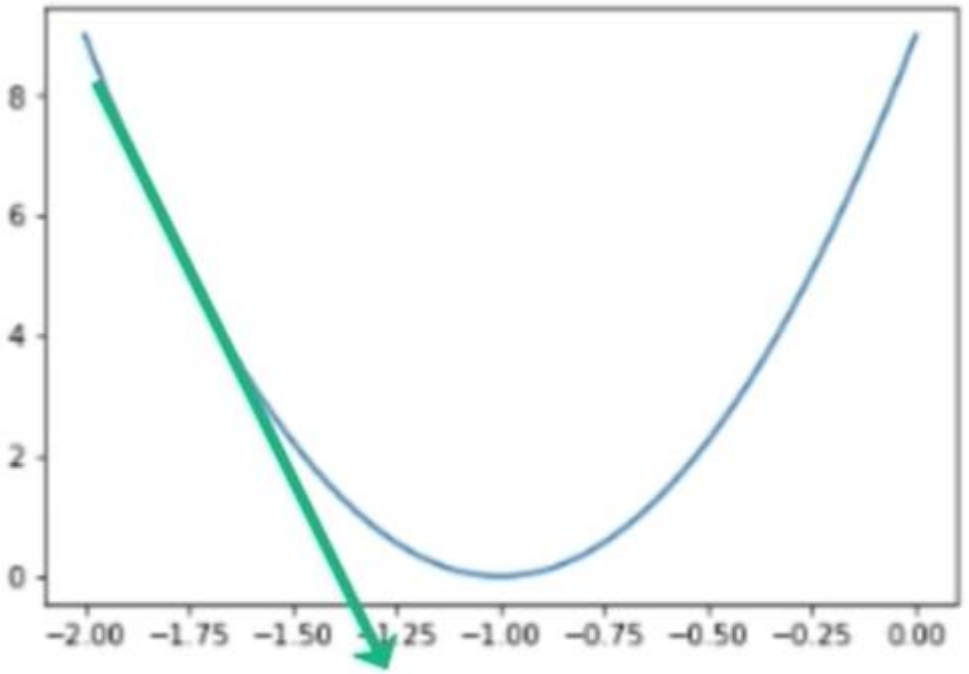
This points to the direction of maximum error. (Upwards).

Since our objective is to go to a direction where loss decreases (towards minimum), we reverse the direction using negative sign.

Parameters are updated with a learning rate

When a model is trained, this process is done a number of times, till convergence is reached.

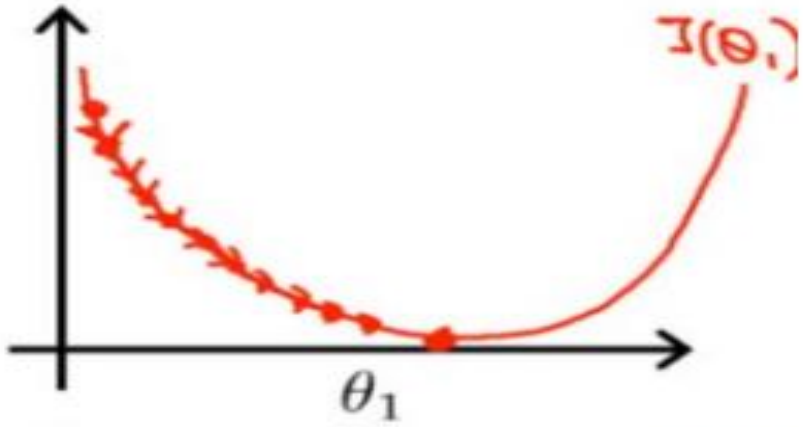




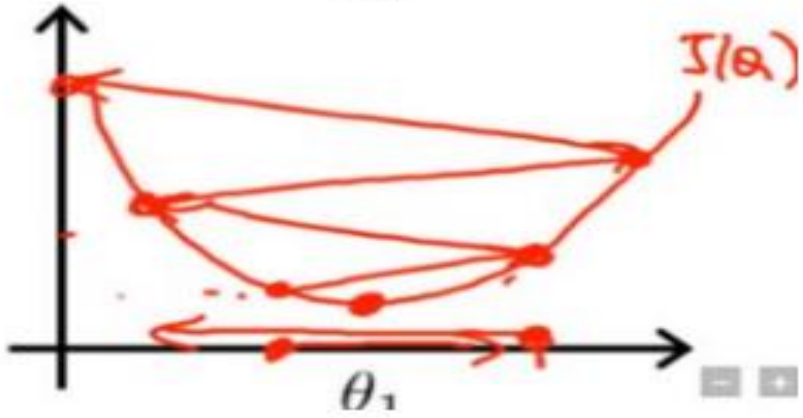
Gradient Descent

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)$$

If α is too small, gradient descent can be slow.



If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.



Gradient descent with small (top) and large (bottom) learning rates. Source: Andrew Ng's Machine Learning course on Coursera

Gradient in Pytorch, (instead of x,y use a,b)

Pytorch makes the Gradients available through the calls variable.grad.

They need to be declared with the parameter requires_grad = True.

z = a **3 + 2 * (b**2)

```
x = torch.tensor(2., requires_grad = True)
y = torch.tensor(4., requires_grad = True)
z = x**3 + 2*(y**2)
z.backward()
print(x.grad)
print(y.grad)
# derivateive wrt x will be 3x**2
# at x =2, derivative = 12
```

```
tensor(12.)
tensor(16.)
```

Updating Weights of the model

DELETE the gradient code as it as overwritten X & y variables, which we don’t want.

Steps:

Define the model - Look at LR class definition.

Define the loss function- mse.

get predicted values with this model. (model(X))

Calculate loss using y and y_hat.

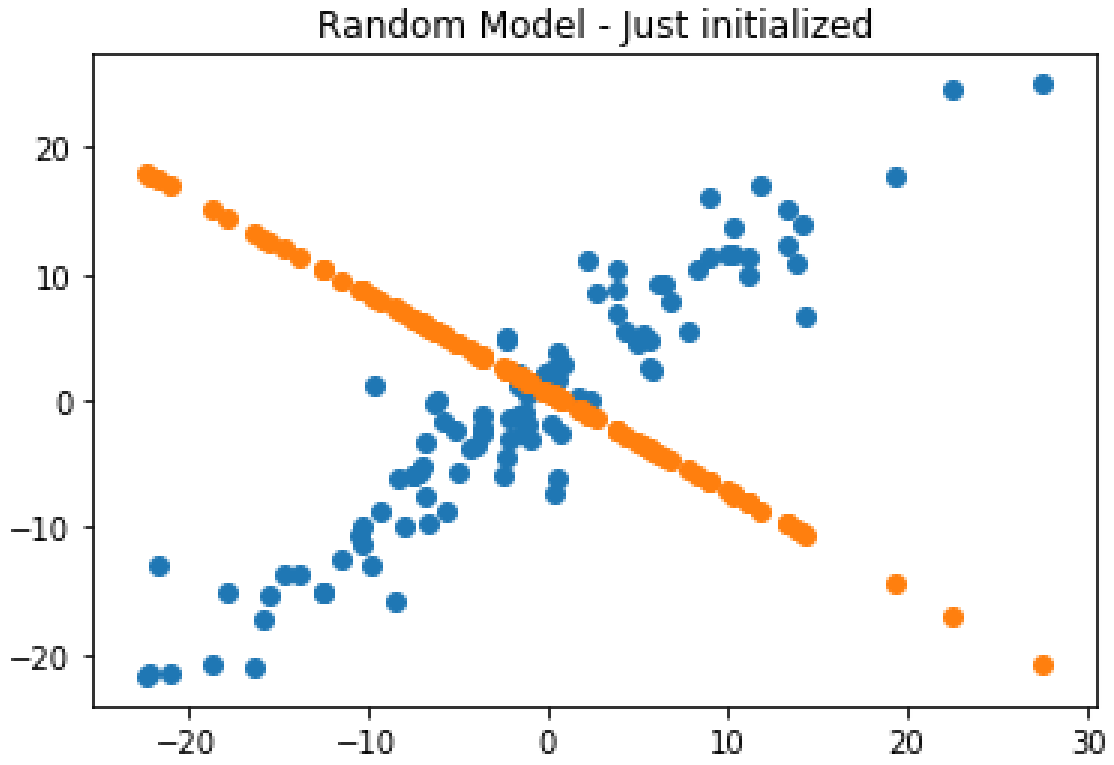
call loss.backward()

update weights by subtracting the gradient from the current weight.

call param.grad.zero_() # important

Random Model

```
plt.scatter(X,y)
plt.scatter(X, model(X).detach().numpy())
_ = plt.title("Random Model - Just initialized")
```



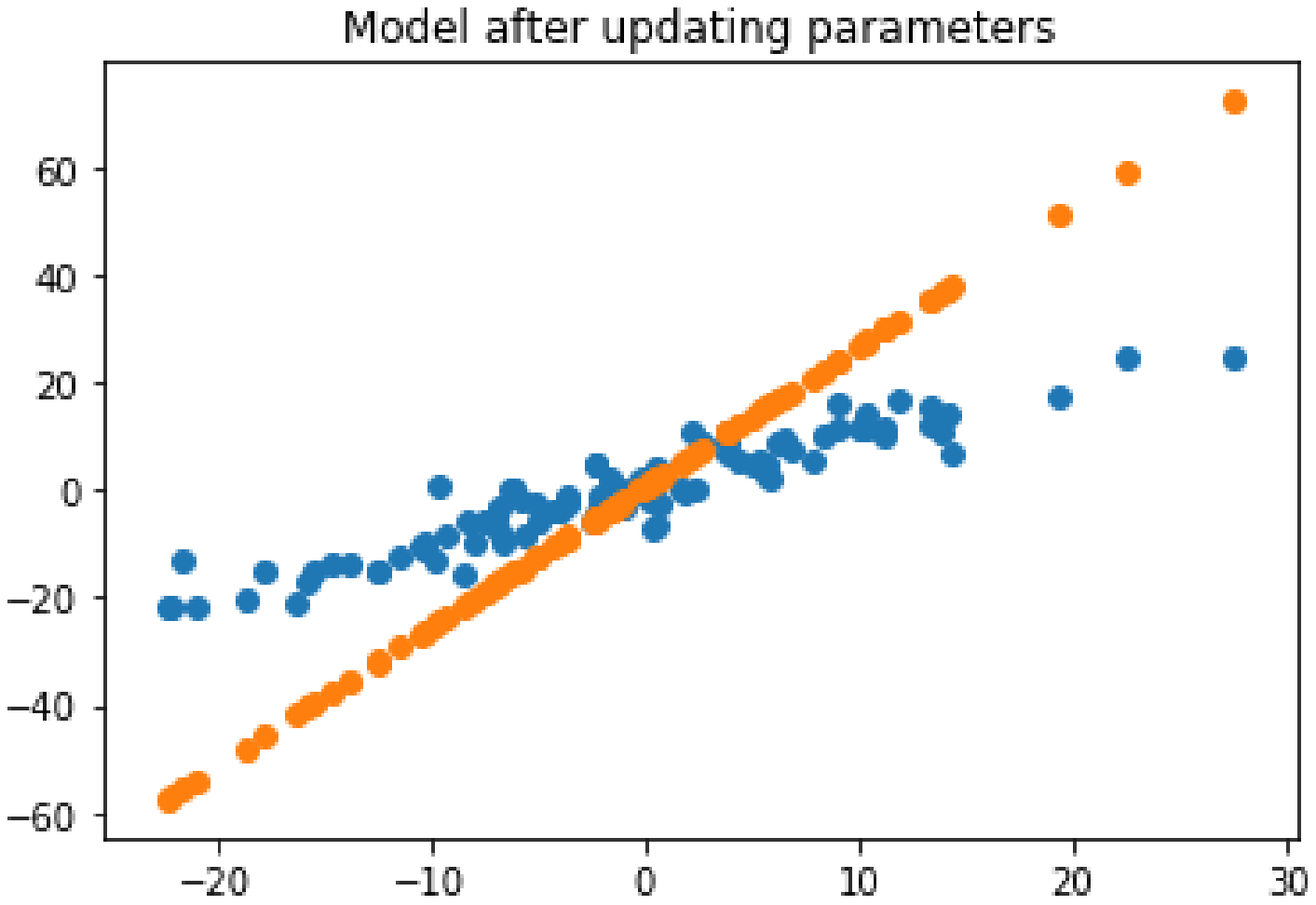
Looking at parameters (weights and bias) values before updating the weights

```
y_hat = model(X)
loss = mse(y_hat, y)
loss.backward()
lr = .01
print('before updating parameters')
for param in model.parameters():
    print(param)
```

Update Parameters - use param.grad.zero_()

```
with torch.no_grad():
    for param in model.parameters():
        param.sub_(lr * param.grad)
        param.grad.zero_()
print('after')
for param in model.parameters():
    print(param)
```

Visualize the model after weights are updated once



Training almost done

The last step of this training is to wrap this in a loop.

In Deep Learning community, this is referred as an epoch.

An epoch is said to be completed when the model has seen all the samples once during backpropagation.

Exercise to you

Implement the whole stuff in a loop.

Once implemented, you should be able to run the training loop for a number of times, say 100.

during the loop, print the loss value on every tenth iteration.

Use: *if i % 10 == 0: print(i, loss)*

Also, for every epoch, capture loss in a list variable called losses. This will be used to print losses over epochs.

Hint: declare losses = [], inside loop losses.append(loss).

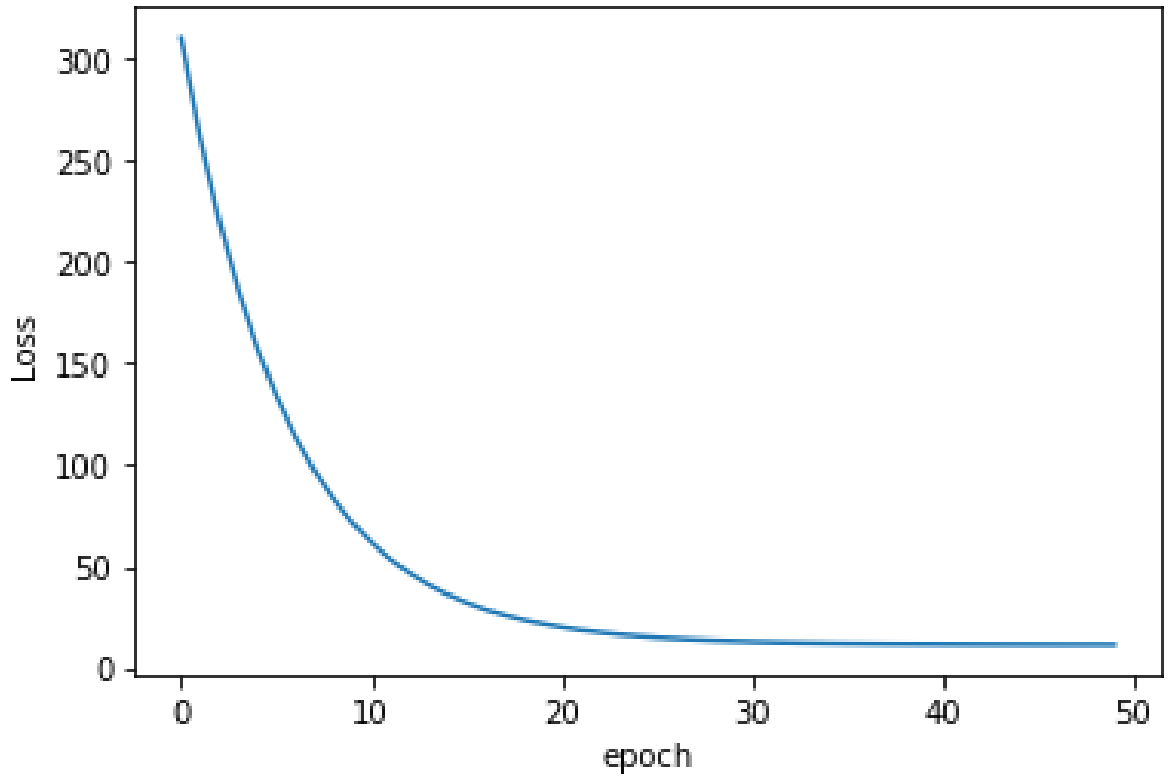
Putting it all together - Structure

```
num_epochs = 10

# losses = [] capture losses

for i in range(num_epochs):

    # calculate y_hat

   # calculate loss

   # call loss.backward

  # update parameters

 # losses.append(loss)

 # every tenth iteration print loss Use: if i % 10 == 0: print(i, loss)
```

Exercise to you

Plot the final model and losses.

```
plt.plot(range(epochs), losses)
plt.ylabel('Loss')
_ = plt.xlabel('epoch')
```



Solution

Solution

```
def do_training(model, num_epochs = 100):
    losses = []
    for i in range(num_epochs):
        y_hat = model(X)
        loss = mse(y_hat, y)
        losses.append(loss)
        if i % 20 == 0: print(i, loss)
        with torch.no_grad():
            loss.backward()
            for param in model.parameters():
                param.sub_(lr * param.grad)
                param.grad.zero_()
    return(model, losses)
```

Congrats

You have successfully built Neural Networks for Linear Regression from basics.

Exercise 1A - Optimizer and Loss Functions

Optimizers and Loss functions

The job of updating parameters will be done by the Optimizers.

A lot of optimizers have evolved in recent years,which help in training neural networks efficiently and make them more accurate.

Instead of *param.sub_(lr * param.grad)* during backprop, *optimizer.step()* is called. This is where frameworks are becoming useful, as they do a lot of background work for us.

As in Optimizers, Loss functions are also provided by the framework. Let us replace our hand coded function by the one provided by pytorch and let optimizers do the weight updates.

Exercise - Refactoring Code for Optimizer

```
torch.manual_seed(100)
criterion = nn.MSELoss()
optimizer = torch.optim.SGD(model.parameters(), lr = 0.01)
lm = Linear_Model(1,1)
```

```
epochs = 50
losses = []
for i in range(epochs):
    y_pred = model.forward(X)
    loss = criterion(y_pred, y)
    if i % 20 == 0: print(i, loss)
    #print("epoch:", i, "loss:", loss.item())
    losses.append(loss)
    optimizer.zero_grad() # Note that torch.no_grad
                          #and param.zero_grad is taken away
    loss.backward()
    optimizer.step()
```

Some exercises

- Change the learning rate and see the impact on training.
- Play with number of epochs and observe.
- Find out the best **learning rate** for your model, using (.001, .01 and .1)

Take the backup of your work.

- Your AWS login will be deleted after this weekend.

Roadmap for You

Two Tracks - Data Science or/and Data Engineer

What suits you is the question?

Usability of Models - Customer have few requirements like how to use a model, how to train models for zillion devices :-)

Why and Where you can be a good fit?

Good architect skills always come in handy, ability to research and deliver is what counts in today’s scenario.

Complete solution givers are in demand, bits and pieces player with one speciality could be helpful.

Further Suggestions

Try to implement for your own use cases.

Classification problems are aplenty, spam detection, sentiment mining, faulty sensors, disease conditions, they are all candidates of applying this technique.

if you see an opportunity, please discuss.

You can try to host this model. Some common ways are Flask, AWS Lambda, AWS Sagemaker. These are requirements of the market nowadays and this skill is not found easily.

All the Best and THANK YOU!