PyTorch – freeCodeCamp – Basics of Pytorch – tensors and gradients

Difference between classical programming and machine learning

In classical programming, you set the rules and when data arrives the rules give you answers. E.g – if you need to find the shortest path, you must work out bfs or dfs and come up with answers.

In case of machine learning, you are given data and answers and the task it to figure out the rules. E.g – you are given two images of cat and dog and say that which is which. Now form a set of images of cats and dogs your algorithm must be able to discern cats and dogs.

What is tensor?

Tensor can be anything, from a number, to an array, to a vector, to a vector of vector of vectors aka n-dimensional array.

Matrix multiplication along with other operations such as addition, division, etc. can be done easily using tensors.

Tensor maintains conformity. If in the 4th dimension a row has 12 columns, all the rows will have equal number of columns. If one of those columns have 7 rows than all 12 columns will have 7 rows.

The term gradients and derivatives are same. It mathematical expression is dy/dx and it is the slope of a curve. Gradient or derivative is for 1 unit of movement along with x-axis, how much movement happens in y-axis.

Derivatives are used when you are dealing with numbers and gradients are used when you are dealing with matrices.

PyTorch Functions

# **Linear Regression**

import torch.nn as nn

torch.from\_numpy() [takes in numpy matrix and turns into a tensor. Why? Numpy works best for CPUs; whereas, tensors are built for parallel computation using GPUs.]

from torch.utils.data import TensorDataset [Helps to create batch from datasets]

train\_ds = TensorDataset(inputs, targets)

from torch.utils.data import DataLoader [splits data into pre-defined size while training and helps to shuffle, random sampling of data]

batch\_size = 5

train\_dl = DataLoader(train\_ds, batch\_size, shuffle=True)

model = nn.Linear(3,2) [#of inputs and outputs]

import torch.nn.functional as F

loss\_func = F.mse\_loss

loss = loss\_func(model(inputs), targets)

if you put “**?F.mse\_loss”** inside a cell, it will show details about the Mean Square Function. Similar goes for everything.

opt = torch.optim.SGD(model.parameters(), lr=le-5) // lr stands for learning rate

def fit(num\_epochs, model, loss\_func, opt):

for epoch in range(num\_epochs):

for xb, yb in train\_dl:

pred = model(xb) // generating predictions

loss = loss\_func(pred, yb) // comparing predictions with outputs

loss.backward() // computing the gradients

opt.step() // NEW :- update parameters with gradients

opt.zero\_grad() // resetting the gradients to zero

if (epoch+1) % 10 == 0

print(‘Epoch [{}/{}], Loss: {: .4f}’ .format(epoch+1, num\_epochs, loss.item()))

now to train the model:

fit(100, model, loss\_func, opt)