**Setting Up Loss Function and Optimizer:**

Optimizer is one of the ways that can nudge the parameters of our models. In our case, weights or bias towards the values. To Lower the loss function.

Here we used ---- nn.L1Loss

Regression --- L1 loss, MSE loss we use it.

Criterions are helpful to train a neural network. Given an input and a target, they compute a gradient according to a given loss function.

Optimizer algorithm --- SGD (Stochastic Gradient Descent)

What is happening here, we are using random numbers, and randomly adjusting the values given here, (to ensure 1 random number is generated, we are using seed here)

Then just randomly adjusting these values to minimize the loss function.

We are fixing lr = learning rate. ( most important hyperparameter)

Smaller the learning rate, smaller the change in parameter. Larger the learning rate, larger the change in parameter.

**PyTorch training loop:**

Forward pass – means data moving from input to output layer.

A couple of things we need in a training loop:

1. Loop Through the data.
2. Forward pass (this involves data moving through our model's forward() functions to make predictions on data - also called forward propagation.)
3. Calculate the loss (compare forward pass predictions to ground truth labels)
4. Optimizer zero grad.
5. Loss backward - move backwards through the network to calculate the gradients of each of the parameters of our model with respect to the loss. (**backpropagation**)
6. Optimizer Step - use the optimizer to adjust our model's parameters to try and improve the loss (**gradient descent**) (means minimize the gradient)

We are going to set it to train mode.

Loss function --- input first and target next

Five major steps of training a data.

Learning rate scheduling: -- Start with the big steps, as we get closer and closer, then minimizes the steps and make it small. (finding coin under sofa analogy)

torch.inference\_mode()

**Writing Testing Code:**

model\_0.eval() # turns off different settings in the model not needed for evaluation/testing (drop out/ batch norm layers)

torch.inference\_mode()

--- turns off gradient tracking

And couple more things behind the scene.

with torch.no\_grad():

base change kora holo(started a new instance of our model.

I tried learning rate .1 first, end result was not good, again used .01 it was giving really good result

One error problem. All tensor values are derived. That is why errors are given.

# Plot the loss curves

plt.plot(epoch\_count, loss\_values, label="Train loss")

plt.plot(epoch\_count, test\_loss\_values, label="Test Loss")

plt.title("Training and test loss curves")

plt.ylabel("Loss")

plt.xlabel("Epochs")

loss values are still on pytorch, the reason we need to convert them right now.

import numpy as np

epoch\_count,  test\_loss\_values

np.array(torch.tensor(loss\_values).cpu().numpy())

Train and test match up at some distance, train the model more longer we will get more accurate value.

**#Saving and Loading a model. :**

Have to know about python pickle.

The pickle module implements binary protocols for serializing and de-serializing a Python object structure. “Pickling” is the process whereby a Python object hierarchy is converted into a byte stream, and “unpickling” is the inverse operation, whereby a byte stream (from a binary file or bytes-like object) is converted back into an object hierarchy

1. Serializing – saving.
2. De Serializing – loading.

<https://pytorch.org/tutorials/beginner/saving_loading_models.html>

--- saving and loading state dict.

--- Another, saving and loading entire model.

pytorch objects usually has extension of pth. ( either .pt or .pth)

beauty of pytorch is it’s flexibility.

**Putting it all together:**

state\_dict()

clear out the role of state\_dict()

device type is converted to GPU (current\_device) [ Previously model was called on CPU devices)

Model used here is linear layers.