Simple Neural Network in PyTorch

Why Google Colab??

- Google Colab provides a free cloud environment with a GPU (Graphic Processing Unit),
- It speeds up computations for deep learning tasks.
  - Runtime > Change runtime type > Hardware accelerator > GPU.
  - It will train the NN faster compared to using just the CPU.

Note:

- **!** tells the Colab environment to execute the command in the shell/terminal.
- Without the !, the command would be treated as a Python statement and would result in an error.

Step 1: install PyTorch library (torch)

```
!pip install torch torchvision
```

torchvision adds additional features that are important if you are working with image-based machine learning projects. **Torchvision** provides

- Pre-trained Models: ResNet (Residual network), VGG (Visual Geometry Group (VGG))
  - These models can be used for image classification, object detection, Datasets: MNIST, CIFAR-10, ImageNet
  - These datasets come with utilities for downloading, loading, and preprocessing.
- Transforms: Resizing, cropping, normalization and converting to tensors.
  - Required to prepare image data before feeding it into a neural network.

Step 2: Import the required libraries:

import torch   #For creating and managing the neural network.

import torch.nn as nn   #For defining the structure of the neural network.

import torch.optim as optim   #For optimization (like gradient descent).

import torchvision # For working with standard datasets like MNIST

import torchvision.transforms as transforms # is used for image preprocessing

import matplotlib.pyplot as plt   #For data visualization (create a variety of plots, such as line charts, bar charts, scatter plots, histograms)

Step 3: Preparing the Dataset

MNIST dataset is used that has handwritten digits from 0 to 9 and each image in the dataset represents a single digit.

- Convert images to tensors (PyTorch's data format)
- Normalize the image pixel values to make training easier.
  - helps make training faster and more stable

Step 3 Contd..

Transformations:

- ToTensor(): Converts the image into a PyTorch tensor
  - Scales pixel values from [0, 255] to [0.0, 1.0].
- Normalize((0.5,), (0.5,)):
  - Normalizes pixel values to the range [-1, 1]

    Normalized Pixel= (Pixel Value − Mean) / Std

    - Center the data around 0.
    - To train the model faster.
    - Symmetric Activation Functions

DataLoader:

- Organizes the dataset into batches for faster and more efficient training.
- train_loader is for training, and test_loader is for evaluation.

Step 4: Defining the Neural Network

A neural network is a series of layers of mathematical operations. The simplest form is a feedforward neural network, where data flows in one direction

input → hidden layers → output

self.fc1 = nn.Linear(28 * 28, 128) # Input to hidden

- It defines a fully connected layer **that maps 784 input features** (flattened MNIST pixels) to **128 output features** (hidden layer neurons).

self.fc2 = nn.Linear(128, 64) # Hidden to hidden

- It maps **128 features** and transforms it into another set of features **64 features**
  - ✓ It helps models to reduces computation

Step 4 Contd..

self.fc3 = nn.Linear(64, 10) **# Hidden to output**

- Output size = 10 (number of classes).
- In the MNIST dataset, there are 10 classes representing the digits 0 to 9.
- Each of the 10 output neurons corresponds to one class.

self.relu = nn.ReLU() **# Define the Relu Activation function,**

- ReLU stands for Rectified Linear Unit, i.e.; max(0, x)
- You can use ReLU multiple times in your network's forward pass without redefining it

Step 4 Contd..

self.softmax = nn.LogSoftmax(dim=1)

- It is used in the final output layer of classification models
- The Softmax function converts raw output values into probabilities for multi-class classification.
- dim=1 means Softmax is applied across column-wise (classes) for each row (sample).
    - For each row, the sum of probabilities is 1.

x = x.view(-1, 28 * 28) #Flatten input, reshapes x into a 2D tensor with shape because layer in NN expect a 2D input

Example: A batch of size 16, containing MNIST images ([16, 1, 28, 28]). Here 16 is the total image, 1 is the number of channel, (28, 28) is height and weight

For input ([16, 1, 28, 28]), output will be [16, 784]

Step 5: Define the Loss Function and Optimizer

model = SimpleNN() # Contains layers, activation functions, and the forward pass logic.

criterion = nn.NLLLoss() # **Loss function, Negative Log-Likelihood Loss**

- Measures how well the model's predictions match the ground truth (actual labels).
- Guides the optimizer on how to adjust the model's parameters, e.g. weight
- Calculates the negative logarithm of the probability of the correct class.
- **Lower NLLLoss** indicates **a better model**.

Step 5: Contd.

optimizer = optim.Adam(model.parameters(), lr=0.001)

- This sets up the optimizer, which **updates the model's parameters** during training to **minimize the loss.**
- Adam (Adaptive Moment Estimation) is an optimization algorithm that adjusts the learning rate for each parameter. It is widely used due to its adaptive learning rate capabilities.
- A smaller learning rate (**lr=0.001**) ensures stability and allows the network to gradually improve its predictions.
- By combining **NLLLoss()** and **Adam**, the neural network can learn from its mistakes and improve its performance over time.

Step 6: Train the Model

Step 7: Test the Model

Thank You

The statement, "The MNIST dataset has 28x28 pixel grayscale images, flattened into 784 values," means the following:
1. MNIST DatasetMNIST is a dataset of handwritten digits (0–9).
Each image in the dataset represents a single digit.

2. 28x28 Pixel Grayscale Images Each image in the dataset is a grayscale image (not colored, so it has a single intensity channel).

The image dimensions are 28x28 pixels, meaning the image has: 28 rows and 28 columns of pixels.
Each pixel holds an intensity value that ranges from 0 (black) to 255 (white), with shades of gray in between.