**ROHAN NYATI**

**500075940**

**R177219148**

**BATCH – 5 (Ai & Ml )**

**Experiment -2**

|  |
| --- |
| # Import torch |
|  |
| **import** torch |
| **import** torch.nn **as** nn |
| **import** torchvision |
| **import** matplotlib.pyplot **as** plt |
|  |
| # Device configuration |
| device = torch.device('cuda' **if** torch.cuda.is\_available() **else** 'cpu') |
| **print**(**f**"Running on '{device}'") |

**Steps to Implement in PyTorch -**

* Import PyTorch
* Initialize Hyper-parameters
* Download MNIST Dataset
* Load the Dataset
* Build the Feedforward Neural Network
* Instantiate the FNN
* Enable GPU
* Choose the Loss Function and Optimizer
* Training the FNN Model
* Testing the FNN Model
* Save the trained FNN Model for future use

|  |
| --- |
|  |
| # Downloading dataset |
|  |
| **from** torchvision **import** datasets |
|  |
| train\_data = datasets.MNIST( |
| *root* = 'data', |
| *train* = True, |
| *download* = True, |
| ) |
|  |
| test\_data = datasets.MNIST( |
| *root* = 'data', |
| *train* = False, |
| *download* = True, |
| ) |
|  |
| # Loading dataset |
|  |
| train\_labels = train\_data.targets |
| test\_labels = test\_data.targets |
| train\_data = train\_data.data |
| test\_data = test\_data.data |
|  |
| **print**(**f**"train\_data.shape:{train\_data.shape} test\_data.shape:{test\_data.shape}") |
|  |
| # Exploring data |
|  |
| plt.imshow(train\_data[0], *cmap*='gray') |
| plt.show() |
|  |
| # Set hyperparameters |
|  |
| n\_train\_sample = train\_data.shape[0] |
| n\_test\_samples = test\_data.shape[0] |
|  |
| image\_size = train\_data.shape[-1] |
| input\_size = image\_size \* image\_size |
| hidden\_layer\_size = 512 |
| n\_classes = 10 |
|  |
| n\_iters = 50 |
| batch\_size = 2000 |
| lr = 0.001 |

|  |
| --- |
|  |
| # Transform data |
|  |
| train\_data = train\_data.to(torch.float32)/255. |
| test\_data = test\_data.to(torch.float32)/255. |
| train\_data = train\_data.reshape(-1, input\_size) |
| test\_data = test\_data.reshape(-1, input\_size) |
|  |
| # Store all data on gpu |
|  |
| train\_data = train\_data.to(device) |
| test\_data = test\_data.to(device) |
| train\_labels = train\_labels.to(device) |
| test\_labels = test\_labels.to(device) |
|  |
| # Making model |
|  |
| **class FFNN**(***nn***.***Module***): |
|  |
| **def \_\_init\_\_**(*self*, *input\_size*, *hidden\_layer\_size*, *n\_classes*): |
| ***super***(FFNN, self).**\_\_init\_\_**() |
| self.input\_size = input\_size |
| self.hidden\_layer\_size = hidden\_layer\_size |
| self.n\_classes = n\_classes |
| # Adding layers |
| self.l1 = nn.Linear(input\_size, hidden\_layer\_size) |
| self.l1\_activation = nn.ReLU() |
| self.l2 = nn.Linear(hidden\_layer\_size, n\_classes) |
|  |
| **def forward**(*self*, *x*): |
| z = self.l1(x) |
| a = self.l1\_activation(z) |
| z2 = self.l2(a) |
| **return** z2 |
|  |
| # Making model and training |
|  |
| ffnn = FFNN(input\_size, hidden\_layer\_size, n\_classes) |
| ffnn.to(device) |
|  |
| # Loss and Optimizer |
|  |
| criterion = nn.CrossEntropyLoss() |
| optimizer = torch.optim.Adam(ffnn.parameters(), *lr*=lr) |

|  |
| --- |
|  |
| # Training |
|  |
| losses = [] |
| test\_losses = [] |
|  |
| **for** epoch **in range**(n\_iters): |
| avg, c = 0, 0 |
| **for** idx **in range**(0, n\_train\_sample, batch\_size): |
| # Get images and labels for current epoch |
| batch\_x = train\_data[idx:idx+batch\_size] |
| batch\_labels = train\_labels[idx:idx+batch\_size] |
| # Forward pass |
| predictions = ffnn.forward(batch\_x) |
| # Compute Loss |
| loss = criterion(predictions, batch\_labels) |
| losses.append(***float***(loss)) |
| avg, c = avg+loss, c+1 |
| # Backprop |
| optimizer.zero\_grad() # Reset gradients |
| loss.backward() # Recompute gradients |
| optimizer.step() # Update weights |
| **print**(**f**"Epoch:{epoch+1}\tBatch:{idx}\tLoss:{loss}") |
| # Output loss every 100 epochs |
| **if** ((epoch + 1) % 5) == 0: |
| **print**(**f**"Epoch:{epoch+1}\tAverageLoss:{(avg/c)}") |
| # Compute loss on test set |
| predictions = ffnn.forward(test\_data) |
| test\_loss = criterion(predictions, test\_labels) |
| test\_losses.append(***float***(test\_loss)) |
|  |
| # Plot losses |
|  |
| plt.plot(losses) |
| plt.title("Training Loss") |
| plt.show() |
|  |
| # Plot test losses |
|  |
| plt.plot(test\_losses) |
| plt.title("Test Losses") |
| plt.show() |
|  |
| # Finally compute loss on both sets once more |
|  |
| predictions = ffnn.forward(train\_data) |
| train\_loss = criterion(predictions, train\_labels) |
| **print**(**f**"Loss on train set: {train\_loss}") |
|  |
| predictions = ffnn.forward(test\_data) |
| test\_loss = criterion(predictions, test\_labels) |
| **print**(**f**"Loss on test set: {test\_loss}") |
|  |
| # Save model for future use |
|  |
| torch.save(ffnn.state\_dict(), "./ffnn\_mnist.torch") |

# OUTPUT AND SCREENSHOTS







