CS564 : Foundations of Machine Learning
Assignment 4

By Priyanka Sachan (1901CS43)

Problem Statement

Design and implement a Feed Forward Neural Network (FFNN) and a Recurrent Neural Network (RNN) for the task of image classification on the CIFAR-10 dataset.

Installation

Install the following dependencies either using pip or through conda in a Python 3.5+ environment:

```
python3 -m pip install pandas matplotlib
```

Running the program

Use the following command to run the program :

```
python3 cifar-10-ffnn.py
```

Implementation

Code added in zip file or check [Notebook].

Import Packages & dataset

```
import torch
import torchvision
import numpy as np
import matplotlib.pyplot as plt
from torch.autograd import Variable
import torch.nn as nn
import torch.nn.functional as F
from torchvision.datasets import CIFAR10
from torchvision.transforms import ToTensor
from torchvision.utils import make_grid
from torch.utils.data.dataloader import DataLoader
from torch.utils.data import random_split
%matplotlib inline
```

```
dataset = CIFAR10(root='data/', download=True, transform=ToTensor())
test_dataset = CIFAR10(root='data/', train=False, transform=ToTensor())
```

Pre-processing data

```
dataset_size = len(dataset)
print('Dataset size:',dataset_size)
test_dataset_size = len(test_dataset)
print('Test Dataset size:',test_dataset_size)
```

```
Dataset size: 50000
Test Dataset size: 10000
```

```
classes = dataset.classes
num_classes = len(classes)
print('No. of classes: ',num_classes)
print('Dataset is classified as: ',classes)
```

```
No. of classes:  10
Dataset is classified as:  ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck']
```

```
# Dividing dataset into - train and validation datset
val_size = 5000
train_size = dataset_size-val_size
train_ds, val_ds = random_split(dataset, [train_size, val_size])

print('Train dataset size:', len(train_ds))
print('Validation dataset size:', len(val_ds))
```

```
Train dataset size: 45000
Validation dataset size: 5000
```

```
# Hyperparameters
batch_size=128
no_of_epochs=50
learning_rate=[0.01,0.033,0.067,0.1]
```

```
# Using stochastic gradient descent with batch_size=128
train_loader = DataLoader(train_ds, batch_size, shuffle=True, num_workers=2,
pin_memory=True)
val_loader = DataLoader(val_ds, batch_size*2, num_workers=2, pin_memory=True)
test_loader = DataLoader(test_dataset, batch_size*2, num_workers=2, pin_memory=True)
```

Model

Feed Forward Neural Network

```
class CIFAR10Model(nn.Module):
    def __init__(self):
        super().__init__()
        self.linear1 = nn.Linear(input_size,1024)
        self.linear2 = nn.Linear(1024,512)
        self.linear3 = nn.Linear(512,256)
        self.linear4 = nn.Linear(256,128)
        self.linear5 = nn.Linear(128,64)
        self.linear6 = nn.Linear(64,32)
        self.linear7 = nn.Linear(32,10)

    def forward(self, xb):
        # Flatten images into vectors
        out = xb.view(xb.size(0), -1)
        out = self.linear1(out)
        out = F.relu(out)
        out = self.linear2(out)
        out = F.relu(out)
        out = self.linear3(out)
        out = F.relu(out)
        out = self.linear4(out)
        out = F.relu(out)
        out = self.linear5(out)
        out = F.relu(out)
        out = self.linear6(out)
        out = F.relu(out)
        out = self.linear7(out)
        return out
```

Training

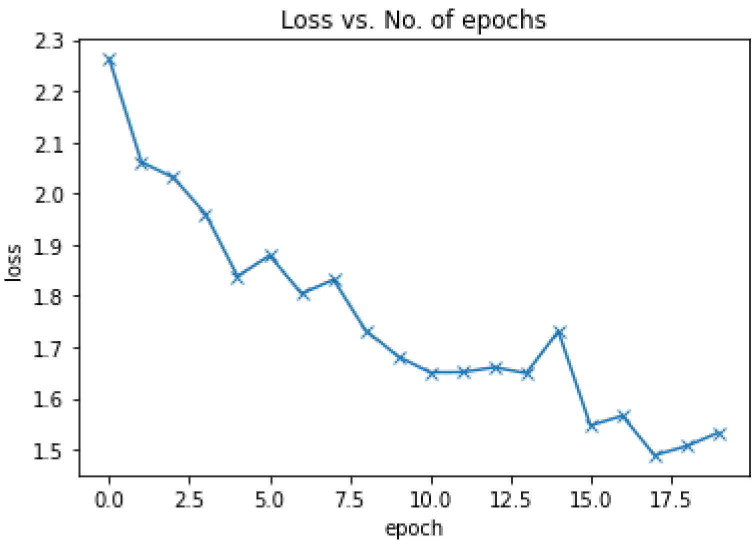
```
history={}
for i in range(len(learning_rate)):
 model = CIFAR10Model()
 print('\nTraining for learning rate: ',learning_rate[i])
 history['model_'+str(i)] = fit(no_of_epochs,learning_rate[i] , model, train_loader,
val_loader)
```

We get results from all specified learning rates, but the best one were from lr=0.1

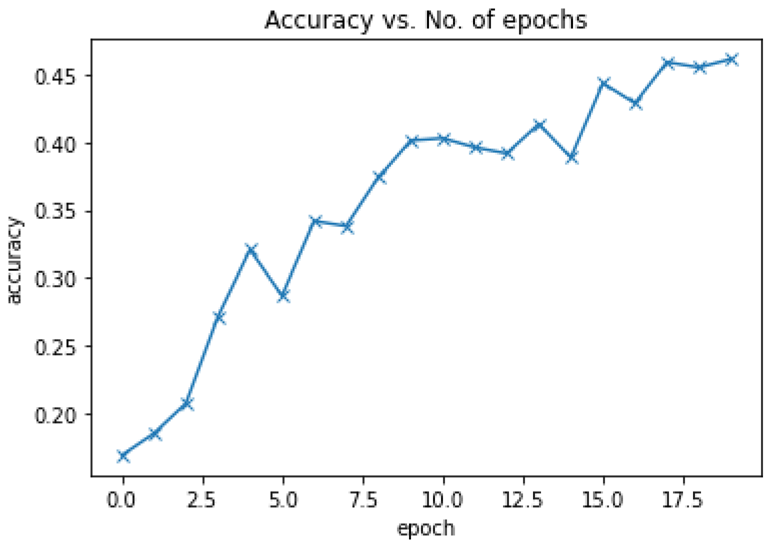
```
Training for learning rate:  0.1
Epoch [0], val_loss: 2.2620, val_acc: 0.1686
Epoch [1], val_loss: 2.0610, val_acc: 0.1848
Epoch [2], val_loss: 2.0308, val_acc: 0.2071
Epoch [3], val_loss: 1.9599, val_acc: 0.2714
Epoch [4], val_loss: 1.8381, val_acc: 0.3206
Epoch [5], val_loss: 1.8787, val_acc: 0.2867
Epoch [6], val_loss: 1.8047, val_acc: 0.3418
Epoch [7], val_loss: 1.8318, val_acc: 0.3381
Epoch [8], val_loss: 1.7327, val_acc: 0.3741
Epoch [9], val_loss: 1.6818, val_acc: 0.4014
Epoch [10], val_loss: 1.6508, val_acc: 0.4031
Epoch [11], val_loss: 1.6512, val_acc: 0.3964
Epoch [12], val_loss: 1.6607, val_acc: 0.3919
Epoch [13], val_loss: 1.6493, val_acc: 0.4137
Epoch [14], val_loss: 1.7313, val_acc: 0.3888
```

Evaluate

Validation Loss



Validation Accuracy



Result

Test loss & accuracy

```
{'val_loss': 1.4721524715423584, 'val_acc': 0.4813476502895355}
```

Recurrent Neural Network

Hyperparameters

```
#Hyperparameters
sequence_length=3*32
input_size=32
hidden_size=1024
no_of_epochs=50
batch_size=128
output_size=10
```

Model

```
class ImageRNN(nn.Module):
   def __init__(self, batch_size,num_layers, sequence_length, input_size, hidden_size,
output_size):
       super(ImageRNN, self).__init__()

       self.hidden_size = hidden_size
       self.batch_size = batch_size
       self.sequence_length = sequence_length
       self.input_size = input_size
       self.output_size = output_size
       self.num_layers=num_layers
       self.basic_rnn = nn.RNN(self.input_size,
self.hidden_size,self.num_layers,batch_first=True)
       self.FC = nn.Linear(self.hidden_size, self.output_size)

   def init_hidden(self,):
       # (num_layers, batch_size, hidden_size)
       return (torch.zeros(self.num_layers, self.batch_size, self.hidden_size))

   def forward(self, X):
       # self.batch_size = X.size(1)
       self.batch_size = X.size(0)
       self.hidden = self.init_hidden()
       # lstm_out=seq_len x batch_size x hidden_size
       # self.hidden =num_layers x batch_size x hidden_size
       lstm_out, self.hidden = self.basic_rnn(X, self.hidden)

       lstm_out = self.FC(lstm_out[:, -1, :])
       return lstm_out
```

Training

```
for epoch in range(no_of_epochs):  # loop over the dataset multiple times
   train_running_loss = 0.0
   train_acc = 0.0
   model.train()

   # TRAINING ROUND
   for i, data in enumerate(train_loader):
        # zero the parameter gradients
       optimizer.zero_grad()

       # reset hidden states
       model.hidden = model.init_hidden()

       # get the inputs
       inputs, labels = data
       inputs = inputs.view(-1, 3*32,32)

       # forward + backward + optimize
       outputs = model(inputs)

       loss = criterion(outputs, labels)
       loss.backward()
       optimizer.step()

       train_running_loss += loss.detach().item()
       train_acc += get_accuracy(outputs, labels, batch_size)

   model.eval()
   print('Epoch:  %d | Loss: %.4f | Train Accuracy: %.2f'
         %(epoch, train_running_loss / i, train_acc/i))
```

```
Epoch:  19 | Loss: 1.6619 | Train Accuracy: 40.03
Epoch:  20 | Loss: 1.6557 | Train Accuracy: 40.18
Epoch:  21 | Loss: 1.6459 | Train Accuracy: 40.60
Epoch:  22 | Loss: 1.6416 | Train Accuracy: 40.52
Epoch:  23 | Loss: 1.6321 | Train Accuracy: 40.88
Epoch:  24 | Loss: 1.6286 | Train Accuracy: 41.44
Epoch:  25 | Loss: 1.6323 | Train Accuracy: 41.15
Epoch:  26 | Loss: 1.6203 | Train Accuracy: 41.66
Epoch:  27 | Loss: 1.6205 | Train Accuracy: 41.41
Epoch:  28 | Loss: 1.7870 | Train Accuracy: 34.75
Epoch:  29 | Loss: 1.7049 | Train Accuracy: 37.77
Epoch:  30 | Loss: 1.6594 | Train Accuracy: 39.90
Epoch:  31 | Loss: 1.6407 | Train Accuracy: 40.66
Epoch:  32 | Loss: 1.6261 | Train Accuracy: 41.03
Epoch:  33 | Loss: 1.6037 | Train Accuracy: 42.02
Epoch:  34 | Loss: 1.6687 | Train Accuracy: 39.66
Epoch:  35 | Loss: 1.6025 | Train Accuracy: 42.11
Epoch:  36 | Loss: 1.5888 | Train Accuracy: 42.69
Epoch:  37 | Loss: 1.5848 | Train Accuracy: 42.82
Epoch:  38 | Loss: 1.5907 | Train Accuracy: 42.65
Epoch:  39 | Loss: 1.5848 | Train Accuracy: 43.06
Epoch:  40 | Loss: 1.5684 | Train Accuracy: 43.54
```

Result

Test Loss & Accuracy

Test Loss: 1.7802
Accuracy: 42.01