```
import torch
import torchvision.datasets as datasets
import torchvision.transforms as transforms
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
from torch.utils.data import Dataset
from tqdm import tqdm


input_size=28
sequence_length =28
num_layers=2
hidden_size=256

learning_rate = 0.001
num_epochs = 5

num_classes =10
batch_size = 64


class SimpleRNN(nn.Module):
    def __init__(self, input_size, num_layers, hidden_size, sequence_length, nu
        super(SimpleRNN, self).__init__()
        self.num_layers = num_layers
        self.hidden_size= hidden_size

        self.rnn = nn.RNN(input_size, hidden_size, num_layers, batch_first=True
        self.fc1 = nn.Linear(hidden_size * sequence_length, num_classes)

    def forward(self, x):

        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(devic

        out, _ = self.rnn(x, h0)
#         print(out.shape)
        out = out.reshape(out.shape[0], -1)
        out = self.fc1(out)
        return out


class SimpleGRU(nn.Module):
    def __init__(self, input_size=input_size, hidden_size=hidden_size, num_laye
        super(SimpleGRU, self).__init__()
        self.hidden_size  = hidden_size
        self.num_layers = num_layers

        self.gru = nn.GRU(input_size, hidden_size, num_layers, batch_first=True
        self.fc1 = nn.Linear(hidden_size * sequence_length, num_classes)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(devic
```

```
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(devic

        out,_ = self.gru(x, h0)
        out = out.reshape(out.shape[0], -1)
        out = self.fc1(out)
        return out



class SimpleLSTM(nn.Module):
    def __init__(self, input_size=input_size, hidden_size=hidden_size, num_laye
        super(SimpleLSTM, self).__init__()

        self.hidden_size = hidden_size
        self.num_layers = num_layers

        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=Tr
        self.fc1 = nn.Linear(hidden_size * sequence_length, num_classes)

    def forward(self, x):
        h0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(devic
        c0 = torch.zeros(self.num_layers, x.size(0), self.hidden_size).to(devic

        out, _ = self.lstm(x,(h0, c0))
        out = out.reshape(out.size(0), -1)
        out = self.fc1(out)
        return out


device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(device)
```

```
cuda
```

```
models = [SimpleRNN( input_size=input_size, hidden_size=hidden_size, num_layers


for model in models:
  print(model._get_name())
  x = torch.randn(64,28,28).to(device=device)
  y = model(x)
  print(y.shape)
```

```
SimpleRNN
torch.Size([64, 10])
SimpleGRU
torch.Size([64, 10])
SimpleLSTM
torch.Size([64, 10])
```

```
import pandas as pd
import numpy as np

class MnistDataset(Dataset):
    def __init__(self, datapath):
```

```
        __    __
        super(MnistDataset).__init__()
        df = pd.read_csv(datapath, dtype=np.float)

        self.x = torch.from_numpy(df.iloc[:, 1:].values)
        self.x = self.x.reshape(self.x.size(0), 1, 28, 28).squeeze(1) # GRU and
        self.x = self.x.float()

        self.y = torch.from_numpy(df.iloc[:, 0].values)
        self.y = self.y.long()

        self.n_samples = df.shape[0]

    def __getitem__(self, index):
        return self.x[index], self.y[index]

    def __len__(self):
        return self.n_samples


transform = transforms.Compose([
    transforms.Resize((28, 28)),
    transforms.ToTensor(),
])



train_dataset = datasets.MNIST(root='./', train=True, transform=transform, downl
test_dataset = datasets.MNIST(root='./', train=False, transform=transform, downl


x, y = train_dataset[0]
print(x.shape, y)
```

```
 torch.Size([1, 28, 28]) 5
```

```
train_dataloader = DataLoader(dataset=train_dataset, batch_size=batch_size, shu
test_dataloader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuff


loss_criterion  = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr = learning_rate)


for model in models:
  print(model._get_name())
  current_loss = 0
  for epoch in range(num_epochs):
      for data, target in tqdm(train_dataloader):
          data = data.to(device=device)
          target = target.to(device=device)
          data.squeeze_(1)
          score = model(data)
          loss = loss_criterion(score, target)
          current_loss = loss
```

```
            optimizer.zero_grad()
            loss.backward()

            optimizer.step()
          print(f"At epoch: {epoch}, loss: {current_loss}")
```

```
SimpleRNN
100%|██████████| 938/938 [00:11<00:00, 79.41it/s]
At epoch: 0, loss: 2.286947250366211
100%|██████████| 938/938 [00:08<00:00, 109.71it/s]
At epoch: 1, loss: 2.3030507564544678
100%|██████████| 938/938 [00:08<00:00, 114.91it/s]
At epoch: 2, loss: 2.311488151550293
100%|██████████| 938/938 [00:08<00:00, 112.32it/s]
At epoch: 3, loss: 2.3215227127075195
100%|██████████| 938/938 [00:08<00:00, 110.08it/s]
At epoch: 4, loss: 2.295093536376953
SimpleGRU
100%|██████████| 938/938 [00:10<00:00, 91.93it/s]
At epoch: 0, loss: 2.295081615447998
100%|██████████| 938/938 [00:10<00:00, 92.14it/s]
At epoch: 1, loss: 2.2987771034240723
100%|██████████| 938/938 [00:09<00:00, 98.23it/s]
At epoch: 2, loss: 2.3044912815093994
100%|██████████| 938/938 [00:10<00:00, 91.39it/s]
At epoch: 3, loss: 2.306396007537842
100%|██████████| 938/938 [00:10<00:00, 85.50it/s]
At epoch: 4, loss: 2.2920422554016113
SimpleLSTM
100%|██████████| 938/938 [00:10<00:00, 86.17it/s]
At epoch: 0, loss: 0.01781553402543068
100%|██████████| 938/938 [00:10<00:00, 86.94it/s]
At epoch: 1, loss: 0.028112657368183136
100%|██████████| 938/938 [00:10<00:00, 87.14it/s]
At epoch: 2, loss: 0.005351718980818987
100%|██████████| 938/938 [00:10<00:00, 87.57it/s]
At epoch: 3, loss: 0.0028403163887560368
100%|██████████| 938/938 [00:10<00:00, 85.37it/s]At epoch: 4, loss: 0.00018
```

```
def check_accuracy(dlr,model):

    total_correct = 0
    total_samples = 0

    model.eval()

    with torch.no_grad():
        for x, y in dlr:
            x = x.to(device=device)
            y = y.to(device=device)
            x.squeeze_(1)
            score = model(x)
            _,predictions = score.max(1)

            total_correct += (y==predictions).sum()
```

```
            total_samples += predictions.size(0)

    model.train()
    print(f"total samples: {total_samples} total_correct: {total_correct} accura


for model in models:
  print(model._get_name())
  check_accuracy(train_dataloader, model)
  check_accuracy(test_dataloader, model)
```

```
SimpleRNN
total samples: 60000 total_correct: 6153 accuracy : 10.254999995231628
total samples: 10000 total_correct: 1058 accuracy : 10.579999536275864
SimpleGRU
total samples: 60000 total_correct: 7308 accuracy : 12.1799997985363
total samples: 10000 total_correct: 1159 accuracy : 11.589999496936798
SimpleLSTM
total samples: 60000 total_correct: 59470 accuracy : 99.11666512489319
total samples: 10000 total_correct: 9872 accuracy : 98.71999621391296
```