Final-Anuj-G

December 2, 2021

1 CP - Final - 2021

1.1 Instruction

- Modify this file to be Final-<Your FirstName-[First Letter of Last Name]>, e.g., Final-Chaklam-S.ipynb
- This exam is open-booked; open-internet.
- You ARE NOT allowed to use sklearn or pytorch libraries, unless stated.
- The completed exams shall be submitted at the Google Classroom
- All code should be **complemented with comments**, unless it's really obvious. **We reserve the privilege to give you zero for any part of the question where the benefit of doubt is not justified**

1.2 Examination Rules:

- You may leave the room temporarily with the approval and supervision of the proctors. No extra time will be added to the exam in such cases.
- You are required to turn on your webcam during the entire period of the exam time
- Students will be allowed to leave at the **earliest 45 minutes** after the exam has started
- **All work should belong to you**. A student should NOT engage in the following activities which proctors reserve the right to interpret any of such act as academic dishonesty without questioning:
  - Chatting with any human beings physically or via online methods
  - Plagiarism of any sort, i.e., copying from friends. **Both copee and copier shall be given a minimum penalty of zero mark for that particular question or the whole exam.**
- No make-up exams are allowed. Special considerations may be given upon a valid reason on unpredictable events such as accidents or serious sickness.

2 Coding

Summary In this section, you will be dealing with a classification problem. You will be generating your own data, training and evaluating them. You will be asked to create a neural network according to the instructions.

The generating data will be a 2d-data (28, 28) being arranged similar to the input shape of CNN2d with channel = 1, height = 28 and width = 28 (batch_size, channel, height, width)

You will then be asked to create a network consisting of 2 CNN2d layers followed by an fc layer.

After the fc layer, it exists lstm cells where num_layer and hidden_size can be defined. This hidden_size is set to be the same as num_classes. The output of lstm is then directly returned.

Import Libraries

[1]:
```
import numpy as np
import torch
from torch import nn
import sklearn
from sklearn.model_selection import train_test_split
from torch.utils.data import TensorDataset
from torch.utils.data import DataLoader
import matplotlib as mpl
import matplotlib.pyplot as plt
```

1. Generate a 4 class 2d-data with 70000 samples. (10 points)

- Class 0 data is sampled from a normal distribution with mean = 0, std = 1, size = (28,28)
- Class 1 data is sampled from a normal distribution with mean = 5, std = 1, size = (28,28)
- Class 2 data is sampled from a normal distribution with mean = 15, std = 1, size = (28,28)
- Class 3 data is sampled from a normal distribution with mean = 20, std = 1, size = (28,28)

The final shape of x should be (70000, 1, 28, 28) and y should be (70000,)

[2]:
```
X_0 = np.random.normal(0, 1, (17500, 28,28))
X_1 = np.random.normal(5, 1, (17500, 28,28))
X_2 = np.random.normal(15, 1, (17500, 28,28))
X_3 = np.random.normal(20, 1, (17500, 28,28))

y_0 = np.full((17500,), 0)
y_1 = np.full((17500,), 1)
y_2 = np.full((17500,), 2)
y_3 = np.full((17500,), 3)

X = np.concatenate((X_0,X_1,X_2,X_3))
y = np.concatenate((y_0,y_1,y_2,y_3))

X = X[:, np.newaxis, :, :]
```

[3]:
```
print(X.shape)
print(y.shape)
```

```
(70000, 1, 28, 28)
(70000,)
```

In case you are unable to do question 1, Use the following lines of code to generate your data and continue with question 3. But be aware that no marks will be given to question 1 and question 2

[ ]:
```
import torchvision
training_data = torchvision.datasets.MNIST('./data/', train=True, download=True,
                              transform=torchvision.transforms.Compose([
```

```
                                torchvision.transforms.ToTensor(),
                                torchvision.transforms.Normalize(
                                  (0.5,), (0.5,))
                              ]))

testing_data  = torchvision.datasets.MNIST('./data/', train=False,␣
 ↪download=True,
                              transform=torchvision.transforms.Compose([
                                torchvision.transforms.ToTensor(),
                                torchvision.transforms.Normalize(
                                  (0.5,), (0.5,))
                              ]))
```

2. Split your data into train and test with the split ratio of 1/7 (5 points)

[4]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=(1/7))
```

3. Check the size of your train and test sets (5 points)

[5]:
```
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(60000, 1, 28, 28)
(60000,)
(10000, 1, 28, 28)
(10000,)
```

4. Set up your train and test loader with a batch size of 32 and shuffle = True (10 points)

[6]:
```
X_train_tensor = torch.from_numpy(X_train)
y_train_tensor = torch.from_numpy(y_train)
X_test_tensor = torch.from_numpy(X_test)
y_test_tensor = torch.from_numpy(y_test)

train_ds = TensorDataset(X_train_tensor, y_train_tensor)
test_ds = TensorDataset(X_test_tensor, y_test_tensor)

batch_size = 32
train_dl = DataLoader(train_ds, batch_size, shuffle=True)
test_dl = DataLoader(test_ds, batch_size, shuffle=True)
```

[7]:
```
print(len(train_dl))
print(len(test_dl))
print(1875*32)
print(313*32)
```

```
1875
313
```

```
60000
10016
```

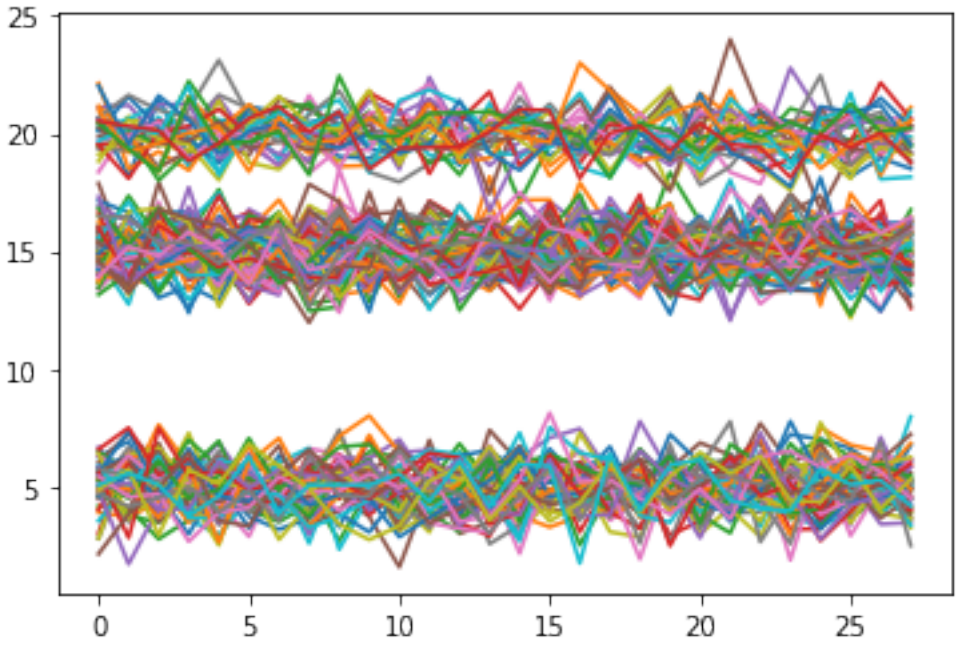
5. Check the shape of your batch. It should be [batchsize = 32, channel = 1, height = 28, width = 28] (5 points)

[8]:
```
for x, y in train_dl:
    print(x.shape)
    print(y.shape)
    break
```

```
torch.Size([32, 1, 28, 28])
torch.Size([32])
```

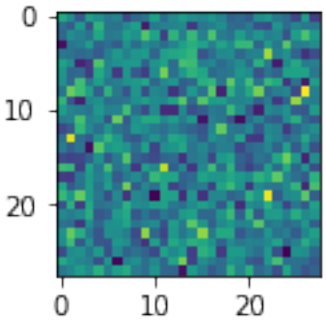
6. Plot 6 samples of any classes (10 points)

[9]:
```
sample1 = X_train_tensor[1]
sample2 = X_train_tensor[2]
sample3 = X_train_tensor[3]
sample4 = X_train_tensor[4]
sample5 = X_train_tensor[5]
sample6 = X_train_tensor[6]

samples = torch.cat((sample1,sample2,sample3,sample4,sample5,sample6))

for sample in samples:
    plt.plot(sample)
```



[10]:
```
plt.subplot(221)

plt.imshow(np.reshape(X_train[0,:],(28,28)))
plt.imshow(np.reshape(X_train[1,:],(28,28)))
```

[10]:
```
<matplotlib.image.AxesImage at 0x1fdf0938fd0>
```



If you use MNIST

[ ]:
```

```

7. Configure your device. (5 points)

[11]:
```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
device
```

[11]:
```
device(type='cpu')
```

8. Define your class called Net with the following layers (40 points)

- cnn2d layer 1 with in_channel = 1, out_channel = 10, kernel size of 5, dropout of p = 0.5 maxpool and relu as its activation function
- cnn2d layer 2 with in_channel = 10, out_channel = 20, kernel size of 5, dropout of p = 0.5 maxpool and relu as its activation function
- linear layer with output of 25
- lstm with num_layer = 2 and set hidden size to be num_classes

[12]:
```
class Net(nn.Module):
    def __init__(self, num_classes=4):
        super(Net, self).__init__()
        self.layer1 = nn.Sequential(
            #in_channel = 1
            #out_channel = 10
            #padding = (kernel_size - 1) / 2 = 2
            nn.Conv2d(1, 10, kernel_size=5),
            nn.ReLU(),
```

```
            nn.MaxPool2d(kernel_size=2, stride=2))
        #after layer 1 will be of shape [32, 10, 12, 12]
        self.layer2 = nn.Sequential(
            nn.Conv2d(10, 20, kernel_size=5),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2))
        #after layer 2 will be of shape [32, 20, 4, 4]
        self.fc = nn.Linear(320, 25)
        self.drop_out = nn.Dropout(p=0.5)  #zeroed 0.2% data
        #after fc will be of shape [32, 25]
        self.lstm = nn.LSTM(25, 4, num_layers=2, batch_first=True)

    def forward(self, x):
        #x shape: [32, 1, 28, 28]
#         print('input shape:', x.shape)
        out = self.layer1(x)
        out = self.drop_out(out)
        #after layer 1: shape: [32, 10, 12, 12]
#         print('after layer 1:', out.shape)
        out = self.layer2(out)
        out = self.drop_out(out)
        #after layer 2: shape: [32, 20, 4, 4]
#         print('after layer 2:', out.shape)
        out = out.reshape(out.size(0), -1)
#         print('after reshape:', out.shape)
#         after squeezing: shape: [32, 320]
        #we squeeze so that it can be inputted into the fc layer
        out = self.fc(out)
#         print('after fc:', out.shape)
        #after fc layer: shape: [32, 25]


        #LSTM layer didnt work with training so it's commented out here

#         out = self.lstm(out)
        return out
```

9. Create an model object with num_layers of lstm = 2, hidden_size and num_classes = 4 (10 points)

If you use MNIST hidden_size and num_classes = 10

[13]:
```
model = Net()
model = model.double()
model
```

[13]:
```
Net(
  (layer1): Sequential(
    (0): Conv2d(1, 10, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (layer2): Sequential(
    (0): Conv2d(10, 20, kernel_size=(5, 5), stride=(1, 1))
    (1): ReLU()
    (2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1,
ceil_mode=False)
  )
  (fc): Linear(in_features=320, out_features=25, bias=True)
  (drop_out): Dropout(p=0.5, inplace=False)
  (lstm): LSTM(25, 4, num_layers=2, batch_first=True)
)
```

10. Define an appropriate loss function for classification of this dataset (2.5 points)

[14]:
```
criterion = nn.CrossEntropyLoss()
```

11. Define your optimizer as Adam with learning rate of 0.001 (5 points)

[15]:
```
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)
```

12. Define your train function and train your model with n_epoch = 3 (15 points)

[16]:
```
import sys

# Train the model
n_epoch=3

losses = []
accs = []
all_labels = []
all_predicted = []

total_step = len(train_dl)

for epoch in range(n_epoch):
    correct = 0
    total = 0
    for i, (images, labels) in enumerate(train_dl):
```

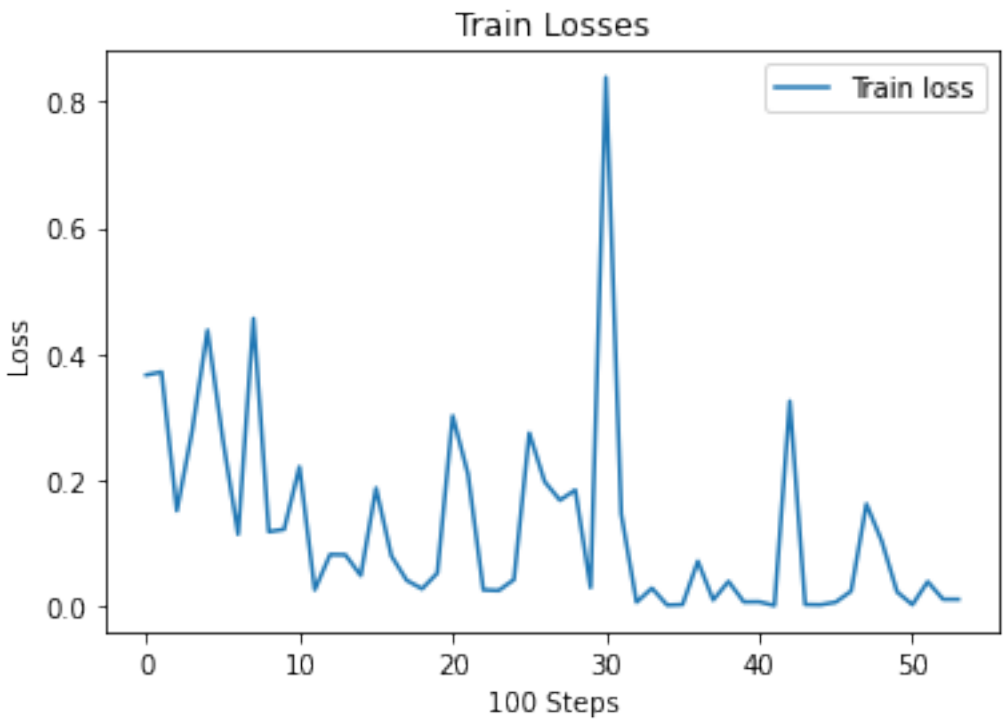
```
        images = images.double()
        labels = labels.long()

#         print('images.shape:',images.shape)
#         print('labels.shape:',labels.shape)

        # Forward pass
        outputs = model(images)
        predictions = model(images)
        _, predicted = torch.max(predictions.data, 1)  #returns max value,␣
↪indices
        total += labels.size(0)  #keep track of total
        correct += (predicted == labels).sum().item()  #.item() give the raw␣
↪number

        acc = 100 * (correct / total)
        accs.append(acc)


        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

        if (i+1) % 100 == 0:
            losses.append(loss.item())
            sys.stdout.write ('\rEpoch [{}/{}], Step [{}/{}], Loss: {:.4f}'
                   .format(epoch+1, n_epoch, i+1, total_step, loss.item()))
```

```
Epoch [3/3], Step [1800/1875], Loss: 0.0104
```

13. Plot your train losses and accuracies (10 points)

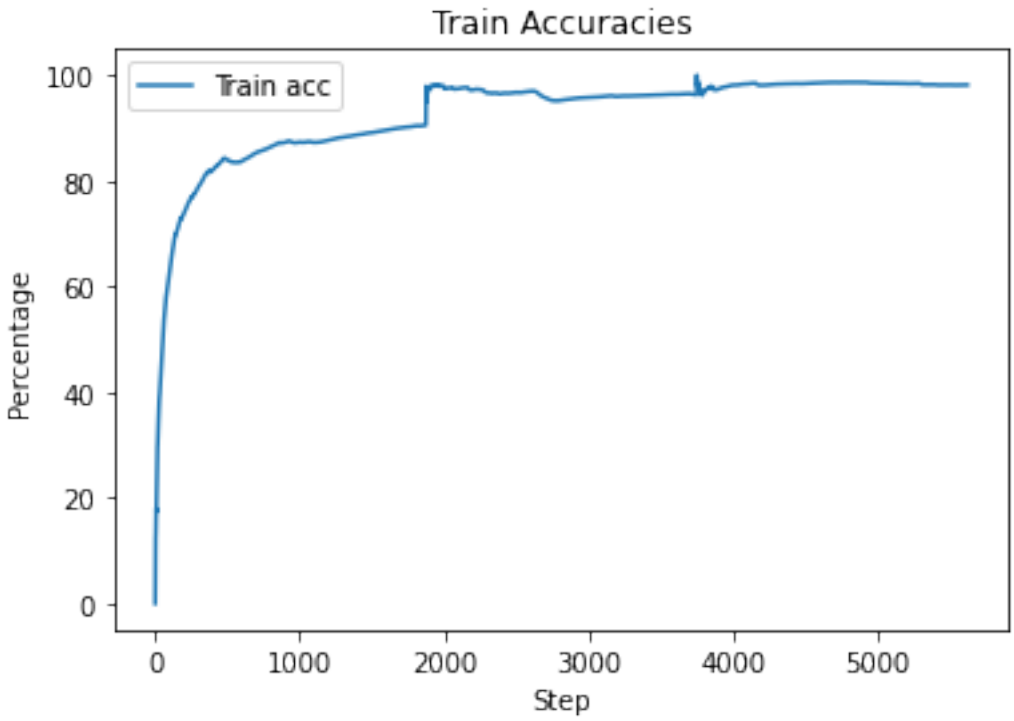
[17]:
```
plt.plot(losses, label='Train loss')
plt.legend()
plt.title("Train Losses")
plt.xlabel("100 Steps")
plt.ylabel("Loss")
```

[17]:
```
Text(0, 0.5, 'Loss')
```



[18]:
```
plt.plot(accs, label='Train acc')
plt.legend()
plt.title("Train Accuracies")
plt.xlabel("Step")
plt.ylabel("Percentage")
```

[18]:
```
Text(0, 0.5, 'Percentage')
```



14. Evaluate your model with your test set (10 points)

[19]:
```
model.eval()  # eval mode will turn off the dropout; good to explicitly call␣
 ↪when you test
all_labels = []
all_predicted = []
with torch.no_grad():
    correct = 0
    total = 0
    for images, labels in test_dl:

        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()
        all_labels.append(labels.numpy())
        all_predicted.append(predicted.numpy())

    print('Test Accuracy of the model on the 10000 test samples: {} %'.
 ↪format(100 * correct / total))
```

```
Test Accuracy of the model on the 10000 test samples: 99.79 %
```

[ ]: