dl-lab-assignment-4

December 14, 2023

[ ]:
```
! pip install tensorflow
```

Q1. Implement RNN, LSTM and GRU for MNIST dataset.

1) RNN :-

[24]:
```
import torch
import torchvision
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import DataLoader
import torchvision.datasets as datasets
import torchvision.transforms as transforms
from tensorflow.keras.datasets import mnist
```

[26]:
```
#Setting up CPU or GPU
device=torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[27]:
```
#Setting hyperparametrs
input_size=28
sequence_length=28
num_layers=3
hidden_size=256
num_classes=10
learning_rate=0.001
batch_size=64
num_epochs=5
```

[28]:
```
#Defining the model-init and forward
class RNN(nn.Module):
  def __init__(self,input_size,hidden_size,num_layers,num_classes):
    super(RNN,self).__init__()
    self.hidden_size=hidden_size
    self.num_layers=num_layers
    self.rnn=nn.RNN(input_size,hidden_size,num_layers,batch_first=True)
    self.fc=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(device)
    out, _ =self.rnn(x,h0)
    out=out.reshape(out.shape[0],-1)
    out=self.fc(out)
    return out
```

[29]:

```
#Getting the dataset
train_dataset=datasets.MNIST(root='dataset/',train=True,transform=transforms.
 ↪ToTensor(),download=True)
test_dataset=datasets.MNIST(root='dataset/',train=False,transform=transforms.
 ↪ToTensor(),download=True)
```

[30]:

```
train_loader=DataLoader(dataset=train_dataset,batch_size=batch_size,shuffle=True)
test_loader=DataLoader(dataset=test_dataset,batch_size=batch_size,shuffle=True)
```

[15]:

```
#Call the model
model=RNN(input_size,hidden_size,num_layers,num_classes)
```

[16]:

```
#Define Loss function and Optimization Algorithm
criterion=nn.CrossEntropyLoss()
optimizer=optim.Adam(model.parameters(),lr=learning_rate)
```

[17]:

```
#Training the data
for epoch in range(num_epochs):
  for batch_idx, (data,targets) in enumerate(train_loader):
    data=data.to(device=device).squeeze(1)
    targets=targets.to(device=device)

    scores=model(data)
    loss=criterion(scores,targets)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

[19]:

```
model.eval()
```

[19]:

```
RNN(
  (rnn): RNN(28, 256, num_layers=3, batch_first=True)
  (fc): Linear(in_features=7168, out_features=10, bias=True)
)
```

[20]:

```
def check_aacuracy(loader,model):
  if loader.dataset.train:
    print("Accuracy of Training data")
  else:
    print("Accuracy of Testing data")
  num_correct=0
  num_samples=0
  model.eval()
  with torch.no_grad():
    for x,y in loader:
      x=x.to(device=device).squeeze(1)
      y=y.to(device=device)

      scores=model(x)
      _,predictions=scores.max(1)
      num_correct=num_correct+(predictions==y).sum()
      num_samples=num_samples+predictions.size(0)
    print((float(num_correct)/float(num_samples))*100)
    model.train()
```

[21]:

```
check_aacuracy(train_loader,model)
check_aacuracy(test_loader,model)
```

```
Accuracy of Training data
97.12833333333334
Accuracy of Testing data
96.66
```

2)LSTM:-

[31]:

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

[32]:

```
#Setting up CPU or GPU
device=torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[33]:

```
#Setting hyperparametrs
input_size=28
sequence_length=28
num_layers=2
hidden_size=256
num_classes=10
learning_rate=0.001
batch_size=64
num_epochs=2
```

[34]:

```
#Defining the model-init and forward
class LSTM(nn.Module):
  def __init__(self,input_size,hidden_size,num_layers,num_classes):
    super(LSTM,self).__init__()
    self.hidden_size=hidden_size
    self.num_layers=num_layers
    self.lstm=nn.LSTM(input_size,hidden_size,num_layers,batch_first=True)
    self.fc=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(device)
    c0=torch.zeros(self.num_layers,x.size(0),self.hidden_size).to(device)
    out, _ =self.lstm(x,(h0,c0))
    out=out.reshape(out.shape[0],-1)
    out=self.fc(out)
    return out
```

[35]:

```
#Getting the dataset
train_dataset=datasets.MNIST(root='dataset/',train=True,transform=transforms.
 ↪ToTensor(),download=True)
test_dataset=datasets.MNIST(root='dataset/',train=False,transform=transforms.
 ↪ToTensor(),download=True)
```

[36]:

```
train_loader=DataLoader(dataset=train_dataset,batch_size=batch_size,shuffle=True)
test_loader=DataLoader(dataset=test_dataset,batch_size=batch_size,shuffle=True)
```

[37]:

```
#Call the model
model=LSTM(input_size,hidden_size,num_layers,num_classes)
```

[38]:

```
#Define Loss function and Optimization Algorithm
criterion=nn.CrossEntropyLoss()
optimizer=optim.Adam(model.parameters(),lr=learning_rate)
```

[39]:

```
#Training the data
for epoch in range(num_epochs):
  for batch_idx, (data,targets) in enumerate(train_loader):
    data=data.to(device=device).squeeze(1)
    targets=targets.to(device=device)

    scores=model(data)
    loss=criterion(scores,targets)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

[40]:

```
model.eval()
```

[40]:

```
LSTM(
  (lstm): LSTM(28, 256, num_layers=2, batch_first=True)
  (fc): Linear(in_features=7168, out_features=10, bias=True)
)
```

[41]:

```
def check_aacuracy(loader,model):
  if loader.dataset.train:
    print("Accuracy of Training data")
  else:
    print("Accuracy of Testing data")
  num_correct=0
  num_samples=0
  model.eval()
  with torch.no_grad():
    for x,y in loader:
      x=x.to(device=device).squeeze(1)
      y=y.to(device=device)

      scores=model(x)
      _,predictions=scores.max(1)
      num_correct=num_correct+(predictions==y).sum()
      num_samples=num_samples+predictions.size(0)
    print((float(num_correct)/float(num_samples))*100)
    model.train()
```

[42]:

```
check_aacuracy(train_loader,model)
check_aacuracy(test_loader,model)
```

```
Accuracy of Training data
98.20166666666667
Accuracy of Testing data
97.92999999999999
```

3)GRU:-

[43]:

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

[44]:

```
#Setting up CPU or GPU
device=torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

[45]:

```
#Setting hyperparametrs
input_size=28
sequence_length=28
num_layers=2
hidden_size=256
num_classes=10
learning_rate=0.001
batch_size=64
num_epochs=2
```

[46]:

```
#Defining the model-init and forward
class GRU(nn.Module):
  def __init__(self,input_size,hidden_size,num_layers,num_classes):
    super(GRU,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.fc=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(device)
    out, _ =self.gru(x,h0)
    out=out.reshape(out.shape[0],-1)
    out=self.fc(out)
    return out
```

[47]:

```
#Getting the dataset
train_dataset=datasets.MNIST(root='dataset/',train=True,transform=transforms.ToTensor(),download=True)
test_dataset=datasets.MNIST(root='dataset/',train=False,transform=transforms.ToTensor(),download=True)
```

[48]:

```
train_loader=DataLoader(dataset=train_dataset,batch_size=batch_size,shuffle=True)
test_loader=DataLoader(dataset=test_dataset,batch_size=batch_size,shuffle=True)
```

[49]:

```
#Call the model
model=GRU(input_size,hidden_size,num_layers,num_classes)
```

[50]:

```
#Define Loss function and Optimization Algorithm
criterion=nn.CrossEntropyLoss()
optimizer=optim.Adam(model.parameters(),lr=learning_rate)
```

[51]:

```
#Training the data
for epoch in range(num_epochs):
  for batch_idx, (data,targets) in enumerate(train_loader):
    data=data.to(device=device).squeeze(1)
    targets=targets.to(device=device)
    scores=model(data)
    loss=criterion(scores,targets)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

[52]:

```
model.eval()
```

[52]:

```
GRU(
  (gru): GRU(28, 256, num_layers=2, batch_first=True)
  (fc): Linear(in_features=7168, out_features=10, bias=True)
)
```

[53]:

```
def check_aacuracy(loader,model):
  if loader.dataset.train:
    print("Accuracy of Training data")
  else:
    print("Accuracy of Testing data")
  num_correct=0
  num_samples=0
  model.eval()
  with torch.no_grad():
    for x,y in loader:
      x=x.to(device=device).squeeze(1)
      y=y.to(device=device)

      scores=model(x)
      _,predictions=scores.max(1)
      num_correct=num_correct+(predictions==y).sum()
      num_samples=num_samples+predictions.size(0)
    print((float(num_correct)/float(num_samples))*100)
    model.train()
```

[54]:

```
check_aacuracy(train_loader,model)
check_aacuracy(test_loader,model)
```

```
Accuracy of Training data
98.45166666666667
Accuracy of Testing data
98.1
```