written5

November 20, 2024

1 STP598 Machine Learning & Deep Learning

1.1 Written Assignment 5

1.1.1 Due 11:59pm Saturday Nov. 30, 2024 on Canvas

1.1.2 name, id

2 LSTM

In this practice, we build an LSTM to classify MNIST handwritten digits.

We can follow this receipt to build the Long-Short Term Memory (LSTM) **Steps of LSTM:** 1. Import Libraries 1. Prepare Dataset 1. Create LSTM Model 1. hidden layer dimension is 100 1. number of hidden layer is 1 1. Instantiate Model 1. Instantiate Loss 1. Cross entropy loss 1. It also has softmax(logistic function) in it. 1. Instantiate Optimizer 1. SGD Optimizer 1. Traning the Model 1. Prediction

[ ]:
```
# Import Libraries
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt

import torch
import torch.nn as nn
from torch.autograd import Variable
from torch.utils.data import random_split, DataLoader, TensorDataset
```

Now download the dataset *digit-recognizer.zip* from the course website and unzip it. Load the dataset using `pd.read_csv` and split data into features(pixels) and labels(numbers from 0 to 9).

2.1 Question 1

Creat **featuresTrain**, **featuresTest** (both float tesor `torch.FloatTensor`) and **targetsTrain**, **targetsTest** (both long tensor `torch.LongTensor`) by splitting the train/test based on 4:1 of the dataset.

[ ]:
```
# load data
train = pd.read_csv(r"./digit-recognizer/train.csv",dtype = np.float32)

# split data into features(pixels) and labels(numbers from 0 to 9)
```

```
targets_numpy = train.label.values
features_numpy = train.loc[:,train.columns != "label"].values/255 #␣
 ↪normalization

# Your code goes here:
```

We can take a look at some digit.

[ ]:
```
# batch_size, epoch and iteration
batch_size = 128
n_iters = 5000
num_epochs = n_iters / (len(train_idx) / batch_size)
num_epochs = int(num_epochs)
print("Epoch Number: ",num_epochs)

# Pytorch train and test sets
train = TensorDataset(featuresTrain,targetsTrain)
test = TensorDataset(featuresTest,targetsTest)

# data loader
train_loader = DataLoader(train, batch_size = batch_size, shuffle = False)
test_loader = DataLoader(test, batch_size = batch_size, shuffle = False)

# visualize one of the images in data set
plt.imshow(features_numpy[598].reshape(28,28))
plt.axis("off")
plt.title(str(targets_numpy[598]))
plt.show()

print(len(train_loader.dataset))
print(len(test_loader.dataset))
```

We view each image (28x28) as a time series of input size 28 unrolled 28 steps.

2.2 Question 2

Define `LSTM` layer with specified `input_size` (28) and `hidden_size` (e.g. 100). Set `batch_first` to **True** to make sure the output has `batch_size` in its first dimension. Then after `LSTM`, add a `Linear` layer to output 10 logits for 10 digits (`output_size` 10).

[ ]:
```
class LSTMModel(nn.Module):
    def __init__(self, input_dim, hidden_dim, layer_dim, output_dim):
        super(LSTMModel, self).__init__()

        # Hidden dimensions
        self.hidden_dim = hidden_dim

        # Number of hidden layers
```

```
        self.layer_dim = layer_dim

        # LSTM
        # define your LSTM layer here

        # Readout layer
        # define your Linear output layer here

    def forward(self, x):
        # Initialize hidden state with zeros
        h0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).
↪requires_grad_()

        # Initialize cell state
        c0 = torch.zeros(self.layer_dim, x.size(0), self.hidden_dim).
↪requires_grad_()

        # 28 time steps
        # We need to detach as we are doing truncated backpropagation through␣
↪time (BPTT)
        # If we don't, we'll backprop all the way to the start even after going␣
↪through another batch
        out, (hn, cn) = self.lstm(x, (h0.detach(), c0.detach()))

        # Index hidden state of last time step
        # out.size() --> 128, 28, 100
        # out[:, -1, :] --> 128, 100 --> just want last time step hidden states!
↪
        out = self.fc(out[:, -1, :])
        # out.size() --> 128, 10
        return out

input_dim = 28
hidden_dim = 100
layer_dim = 1
output_dim = 10
model = LSTMModel(input_dim, hidden_dim, layer_dim, output_dim)
```

Now we choose `CrossEntropyLoss` and `SGD` as the optimizer. Run the followig code to train the LSTM you defined.

[ ]:
```
# loss
error = nn.CrossEntropyLoss()
# optimizer
learning_rate = 0.1
optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)
```

```
# Number of steps to unroll
seq_dim = 28
loss_list = []
iteration_list = []
accuracy_list = []
count = 0
for epoch in range(num_epochs):
    for i, (images, labels) in enumerate(train_loader):
        # Load images as a torch tensor with gradient accumulation abilities
        images = images.view(-1, seq_dim, input_dim).requires_grad_()

        # Clear gradients w.r.t. parameters
        optimizer.zero_grad()

        # Forward pass to get output/logits
        # outputs.size 100, 10
        outputs = model(images)

        # Calculate Loss: softmax --> cross entropy loss
        loss = error(outputs, labels)

        # Getting gradients
        loss.backward()

        # Updating parameters
        optimizer.step()

        count += 1

        if count % 500 == 0:
            # Calculate Accuracy
            correct = 0
            total = 0
            for images, labels in test_loader:

                images = images.view(-1, seq_dim, input_dim)

                # Forward pass only to get logits/output
                outputs = model(images)

                # Get predictions from the maximum value
                _, predicted = torch.max(outputs.data, 1)

                # Total number of labels
                total += labels.size(0)

                # Total correct predictions
```

```
                correct += (predicted == labels).sum()

            accuracy = 100 * correct / total

            loss_list.append(loss.data.item())
            iteration_list.append(count)
            accuracy_list.append(accuracy)

            # Print Loss
            print('Iteration: {}. Loss: {}. Accuracy: {}'.format(count, loss.
↪data.item(), accuracy))
```

2.3 Question 3

Print two graphs to visulize the paths of loss and accuracy in the training process * Loss vs Number of iteration: `loss_list` vs `iteration_list` * Accuracy vs Number of iteration: `accuracy_list` vs `iteration_list`

[ ]:
```
# plot graphs here:
```

2.4 Extra Credit

Try one of the pytroch-optimizer that is not standard optimizers adopted by `PyTorch`, e.g. `AdaBound`. Redo **Question 3**.