demo10_b

April 7, 2020

[0]:

```
!pip install torchtext==0.4
```

```
Requirement already satisfied: torchtext==0.4 in /usr/local/lib/python3.6/dist-
packages (0.4.0)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-
packages (from torchtext==0.4) (2.21.0)
Requirement already satisfied: six in /usr/local/lib/python3.6/dist-packages
(from torchtext==0.4) (1.12.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages
(from torchtext==0.4) (1.18.2)
Requirement already satisfied: torch in /usr/local/lib/python3.6/dist-packages
(from torchtext==0.4) (1.4.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.6/dist-packages
(from torchtext==0.4) (4.38.0)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-
packages (from requests->torchtext==0.4) (2.8)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in
/usr/local/lib/python3.6/dist-packages (from requests->torchtext==0.4) (1.24.3)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in
/usr/local/lib/python3.6/dist-packages (from requests->torchtext==0.4) (3.0.4)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.6/dist-packages (from requests->torchtext==0.4)
(2019.11.28)
```

The AG_NEWS dataset consists of 4 types of news articles (World, sports, business, sci/tech). 120K train samples and 76K test samples.

[0]:

```
import torch
import torchtext
from torchtext.datasets import text_classification
import os
if not os.path.isdir('./.data'):
  os.mkdir('./.data')
train_dataset, test_dataset = text_classification.DATASETS['AG_NEWS'](root='./.
 ↪data',ngrams=2,vocab=None)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
BATCH_SIZE=16
```

```
120000lines [00:06, 18333.38lines/s]
```

```
120000lines [00:13, 8919.15lines/s]
7600lines [00:00, 9705.19lines/s]
```

The dataset is in a TorchText object. We can query properties using inbuilt routines.

[0]:
```
vocab_size = len(train_dataset.get_vocab())
num_class = len(train_dataset.get_labels())
print(vocab_size, num_class)
```

```
1308844 4
```

[0]:
```
ag_news_label = {0 : "World",
                 1 : "Sports",
                 2 : "Business",
                 3 : "Tech"}
```

We need to write a method to get batches of data from this object. We collect bunch of text sequences (of variable length). Since lengths are variable, we store a tensor of delimiters (‘offsets’) to denote beginning of each sequence.

[0]:
```
def generate_batch(batch):
  labels = torch.tensor([entry[0] for entry in batch])
  text = [entry[1] for entry in batch]
  offsets = [0] + [len(entry) for entry in text]
  offsets = torch.tensor(offsets[:-1]).cumsum(dim=0)
  text = torch.cat(text)
  return text, offsets, labels
```

Let’s train a simple language model to predict the type of news article. We will use an “embedding-bag” layer which converts words/2-grams to vectors, and then classify using a linear fully connected layer.

[0]:
```
import torch.nn as nn
import torch.nn.functional as F

class TextSentiment(nn.Module):
  def __init__(self,vocab_size,embed_dim,num_class):
    super().__init__()
    self.embedding = nn.EmbeddingBag(vocab_size,embed_dim,sparse=True)
    self.fc = nn.Linear(embed_dim,num_class)
    self.init_weights()

  def init_weights(self):
    initrange = 0.5
    self.embedding.weight.data.uniform_(-initrange,initrange)
    self.fc.weight.data.uniform_(-initrange,initrange)
    self.fc.bias.data.zero_()

  def forward(self,text,offsets):
    embedded = self.embedding(text,offsets)
    return self.fc(embedded)
```

OK, now let's write a simple data loader and routine to train/test the model.

[0]:

```
from torch.utils.data import DataLoader
embed_dim = 32
model = TextSentiment(vocab_size,embed_dim,num_class).to(device)

def train_func(sub_train):
  train_loss = 0
  train_acc = 0
  data =␣
 ↪DataLoader(sub_train,batch_size=BATCH_SIZE,shuffle=True,collate_fn=generate_batch)
  for i, (text,offsets,labels) in enumerate(data):
    optimizer.zero_grad()
    text,offsets,labels = text.to(device),offsets.to(device),labels.to(device)
    output = model(text,offsets)
    loss = criterion(output,labels)
    train_loss += loss.item()
    loss.backward()
    optimizer.step()
    train_acc += (output.argmax(1)==labels).sum().item()

  scheduler.step() #adjust learning rate

  return train_loss/len(sub_train), train_acc/len(sub_train)

def test_func(sub_test):
  test_loss = 0
  test_acc = 0
  data = DataLoader(sub_test,batch_size=BATCH_SIZE,collate_fn=generate_batch)
  for i, (text,offsets,labels) in enumerate(data):
    text,offsets,labels = text.to(device),offsets.to(device),labels.to(device)
    with torch.no_grad():
      output = model(text,offsets)
      loss = criterion(output,labels)
      test_loss += loss.item()
      test_acc += (output.argmax(1)==labels).sum().item()

  return test_loss/len(sub_test), test_acc/len(sub_test)
```

[0]:

```
num_epochs = 5
criterion = torch.nn.CrossEntropyLoss().to(device)
optimizer = torch.optim.SGD(model.parameters(),lr=4)
scheduler = torch.optim.lr_scheduler.StepLR(optimizer,1,gamma=0.9)
```

```
for epoch in range(num_epochs):
  %time train_loss, train_acc = train_func(train_dataset)

  print('Epoch = %d' %(epoch+1))
  print(f'\tLoss: {train_loss:.4f}(train)\t|\tAcc: {train_acc * 100:.
↪1f}%(train)')
```

```
CPU times: user 7.8 s, sys: 482 ms, total: 8.28 s
Wall time: 8.36 s
Epoch = 1
        Loss: 0.0259(train)     |       Acc: 84.8%(train)
CPU times: user 7.77 s, sys: 456 ms, total: 8.23 s
Wall time: 8.31 s
Epoch = 2
        Loss: 0.0118(train)     |       Acc: 93.7%(train)
CPU times: user 7.72 s, sys: 464 ms, total: 8.19 s
Wall time: 8.24 s
Epoch = 3
        Loss: 0.0070(train)     |       Acc: 96.3%(train)
CPU times: user 7.74 s, sys: 467 ms, total: 8.21 s
Wall time: 8.27 s
Epoch = 4
        Loss: 0.0040(train)     |       Acc: 98.0%(train)
CPU times: user 7.93 s, sys: 426 ms, total: 8.36 s
Wall time: 8.48 s
Epoch = 5
        Loss: 0.0023(train)     |       Acc: 99.0%(train)
```

[0]:

```
%time test_loss, test_acc = test_func(test_dataset)
print(f'\tLoss: {test_loss:.4f}(test)\t|\tAcc: {test_acc * 100:.1f}%(test)')
```

```
CPU times: user 201 ms, sys: 4.6 ms, total: 206 ms
Wall time: 207 ms
        Loss: 0.0238(test)      |       Acc: 90.3%(test)
```

Cool! Let us now see if we can test it on a real world example.

[0]:

```
from torchtext.data.utils import ngrams_iterator
from torchtext.data.utils import get_tokenizer

def predict(text,model,vocab,ngrams):
  tk = get_tokenizer('basic_english')
  with torch.no_grad():
    text = torch.tensor([vocab[token] for token in␣
↪ngrams_iterator(tk(text),ngrams)])
    output = model(text,torch.tensor([0]))
    return output.argmax(1).item()
```

```
ex_string = """The world’s airlines, no longer operating a globe-spanning␣
 ↪choreography of flights,
  are consumed with new work: navigating government bailout offers, negotiating␣
 ↪with unions,
  finding places to park idle planes and scrounging for business like flying␣
 ↪cargo
  and repatriating marooned travelers."""

vocab = train_dataset.get_vocab()
model = model.to('cpu')
print('%s' %ag_news_label[predict(ex_string,model,vocab,2)])
```

```
Business
```