3.LSTM with RNN

November 4, 2020

[1]:
```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

[2]:
```
import torch
import torch.nn as nn

import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from datetime import datetime
from pathlib import Path
import pandas as pd

import torchtext.data as ttd

from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
```

[61]:
```
!pip install -q kaggle
```

[4]:
```
from google.colab import files
```

[5]:
```
files.upload()
```

```
<IPython.core.display.HTML object>


Saving kaggle.json to kaggle.json
```

[5]:
```
{'kaggle.json':
 b'{"username":"nabhsanjaymehtautd","key":"ee0e2e2e8b50d345f23e44404b090088"}'}
```

[6]:
```
!mkdir ~/.kaggle/
```

[7]:
```
!cp kaggle.json ~/.kaggle/
```

[8]:
```
!chmod 60 ~/.kaggle/kaggle.json
```

[9]:
```
!kaggle datasets list
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix
this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Warning: Looks like you're using an outdated API Version, please consider
updating (server 1.5.9 / client 1.5.4)
ref                                                         title
size  lastUpdated          downloadCount
----------------------------------------------------------
----------------------------------------------  -----  ------------------
-------------
terenceshin/covid19s-impact-on-airport-traffic              COVID-19's Impact on
Airport Traffic               106KB  2020-10-19 12:40:17           1644
sootersaalu/amazon-top-50-bestselling-books-2009-2019       Amazon Top 50
Bestselling Books 2009 - 2019         15KB  2020-10-13 09:39:21           1602
thomaskonstantin/highly-rated-children-books-and-stories    Highly Rated Children
Books And Stories             106KB  2020-10-24 12:09:59            340
tunguz/euro-parliament-proceedings-1996-2011                Euro Parliament
Proceedings 1996 - 2011              1GB  2020-10-26 17:48:29             22
rishidamarla/judicial-expenditures-across-all-50-states     Judicial Expenditures
across all 50 States           2KB  2020-10-25 00:07:45            214
docstein/brics-world-bank-indicators                        BRICS World Bank
Indicators                           4MB  2020-10-22 12:18:40            348
kanishk307/6000-indian-food-recipes-dataset                 6000+ Indian Food
Recipes Dataset                     9MB  2020-10-24 01:08:23            428
elvinagammed/chatbots-intent-recognition-dataset            Chatbots: Intent
Recognition Dataset                17KB  2020-10-23 07:44:59            184
omarhanyy/500-greatest-songs-of-all-time                    500 Greatest Songs of
All Time                       33KB  2020-10-26 13:36:09            420
balraj98/synthetic-objective-testing-set-sots-reside        Synthetic Objective
Testing Set (SOTS) [RESIDE]    415MB  2020-10-24 10:07:29             49
lunamcbride24/pokemon-type-matchup-data                     Pokemon Type Matchup
Data                            9KB  2020-10-14 18:56:23            256
gaurav2796/kaggle-competions-rankings-and-kernels           Kaggle Competions,
Rankings and Kernels            698KB  2020-10-15 04:05:15             31
balraj98/indoor-training-set-its-residestandard             Indoor Training Set
(ITS) [RESIDE-Standard]          5GB  2020-10-24 10:07:30             34
romazepa/moscow-schools-winners-of-educational-olympiads    Moscow schools -
winners of educational Olympiads    1MB  2020-10-12 21:45:01             55
sootersaalu/nigerian-songs-spotify                          Nigerian Songs Spotify
24KB  2020-10-25 19:10:23             58
salmaneunus/mechanical-tools-dataset                        Mechanical Tools
Classification Dataset            652MB  2020-11-01 11:28:22            169
```

```
thanatoz/hinglish-blogs                                     Hinglish blogs
2MB  2020-10-13 18:16:05             12
shivamb/netflix-shows                                       Netflix Movies and TV
Shows                          971KB  2020-01-20 07:33:56          52860
nehaprabhavalkar/indian-food-101                            Indian Food 101
7KB  2020-09-30 06:23:43           5941
heeraldedhia/groceries-dataset                              Groceries dataset
257KB  2020-09-17 04:36:08           6200
```

[10]:
```
!kaggle competitions download -c transferlearning-dl-spring2020
```

```
Warning: Your Kaggle API key is readable by other users on this system! To fix
this, you can run 'chmod 600 /root/.kaggle/kaggle.json'
Warning: Looks like you're using an outdated API Version, please consider
updating (server 1.5.9 / client 1.5.4)
Downloading sample_submission.csv to /content
  0% 0.00/34.2k [00:00<?, ?B/s]
100% 34.2k/34.2k [00:00<00:00, 62.1MB/s]
Downloading test.csv to /content
  0% 0.00/506k [00:00<?, ?B/s]
100% 506k/506k [00:00<00:00, 71.6MB/s]
Downloading train.csv.zip to /content
  0% 0.00/527k [00:00<?, ?B/s]
100% 527k/527k [00:00<00:00, 74.1MB/s]
```

[11]:
```
!unzip train.csv.zip -d train
```

```
Archive:  train.csv.zip
  inflating: train/train.csv
```

[12]:
```
data = pd.read_csv('/content/train/train.csv', encoding = "ISO-8859-1")
testdata = pd.read_csv('/content/test.csv', encoding="ISO-8859-1")
```

[13]:
```
data.head()
```

[13]:

	id	text	target
0	86426	@USER She should ask a few native Americans wh…	1
1	16820	Amazon is investigating Chinese employees who …	0
2	62688	@USER Someone should'veTaken" this piece of sh…	1
3	43605	@USER @USER Obama wanted liberals & illega…	0
4	97670	@USER Liberals are all Kookoo !!!	1

[14]:
```
data = data.drop(["id"], axis=1)
```

[15]:
```
testdata = testdata.drop(["id"], axis = 1)
```

[16]:
```
data.head()
```

[16]:
```
                                                text  target
0  @USER She should ask a few native Americans wh…       1
1  Amazon is investigating Chinese employees who …       0
2  @USER Someone should'veTaken" this piece of sh…       1
3  @USER @USER Obama wanted liberals &amp; illega…       0
4                  @USER Liberals are all Kookoo !!!       1
```

[17]:
```
data['target'].value_counts()
```

[17]:
```
0    6220
1    3126
Name: target, dtype: int64
```

[18]:
```
data.columns = ['data','labels']
```

[19]:
```
testdata.columns = ['data']
```

[20]:
```
testdata.head()
```

[20]:
```
                                                data
0  @USER @USER Go home youâ re drunk!!! @USER #M…
1                   @USER @USER Oh noes! Tough shit.
2  @USER Canada doesnâ t need another CUCK! We a…
3  @USER @USER @USER It should scare every Americ…
4  @USER @USER @USER @USER LOL!!!   Throwing the …
```

[21]:
```
data.head()
```

[21]:
```
                                                data  labels
0  @USER She should ask a few native Americans wh…       1
1  Amazon is investigating Chinese employees who …       0
2  @USER Someone should'veTaken" this piece of sh…       1
3  @USER @USER Obama wanted liberals &amp; illega…       0
4                  @USER Liberals are all Kookoo !!!       1
```

[22]:
```
testdata = pd.Series(testdata['data'].to_numpy())
```

[23]:
```
X_train, X_test, y_train, y_test = train_test_split(data['data'],␣
↪data['labels'], test_size=0.3)
```

[24]:
```
type(X_test)
```

[24]:
```
pandas.core.series.Series
```

[25]:
```
print(X_train.shape,y_train.shape)
```

```
(6542,) (6542,)
```

[26]:
```
tokenizer = Tokenizer()
tokenizer.fit_on_texts(X_train)
X_train_tok = tokenizer.texts_to_sequences(X_train)
X_test_tok = tokenizer.texts_to_sequences(X_test)
```

[27]:
```
testdata_tok = tokenizer.texts_to_sequences(testdata)
```

[28]:
```
print(len(testdata_tok))
```

```
3894
```

[29]:
```
print(len(X_train_tok)) # it is a list , does not have shape attribute
```

```
6542
```

[30]:
```
# get words in each review
train_review_lengths = [len(x) for x in X_train_tok]
test_teview_lengths = [len(x) for x in X_test_tok]

testdata_review_lenghts = [len(x) for x in testdata_tok]
```

[31]:
```
# get word -> integer mapping
word2idx = tokenizer.word_index
V = len(word2idx)
print(f'Unique Tokens {V}')
```

```
Unique Tokens 14271
```

[32]:
```
# pad sequences so that we get a N x T matrix

# This function transforms a list (of length num_samples) of sequences (lists␣
 ↪of integers).
# into a 2D Numpy array of shape (num_samples, num_timesteps).
# num_timesteps is either the maxlen argument if provided,
# or the length of the longest sequence in the list.

X_train_tok_pad = pad_sequences(X_train_tok)
print(f'Shape of X_train_tok_pad : {X_train_tok_pad.shape}')
```

```
Shape of X_train_tok_pad : (6542, 102)
```

[33]:
```
testdata_tok_pad = pad_sequences(testdata_tok)
```

[34]:
```
# get sequence length
T = X_train_tok_pad.shape[1]
T
```

[34]:
```
102
```

[35]:
```
Test_T = testdata_tok_pad.shape[1]
```

[36]:
```
Test_T
```

[36]:
```
98
```

[36]:
```

```

[37]:
```
X_test_tok_pad = pad_sequences(X_test_tok, maxlen=T)
print(f'Shape of X_test_tok_pad : {X_test_tok_pad.shape}')
```

```
Shape of X_test_tok_pad : (2804, 102)
```

[38]:
```
X_train_tensor = torch.from_numpy(X_train_tok_pad).long() #int64
y_train_tensor = torch.from_numpy(y_train.to_numpy()) # pandas series to numpy␣
↪, numpy to tensor
X_test_tensor = torch.from_numpy(X_test_tok_pad).long() # int64
y_test_tensor = torch.from_numpy(y_test.to_numpy())
```

[39]:
```
testdata_tensor = torch.from_numpy(testdata_tok_pad).long()
```

[40]:
```
# create dataset objects
train_dataset = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = torch.utils.data.TensorDataset(X_test_tensor,y_test_tensor)
```

[41]:
```
testdata_dataset = torch.utils.data.TensorDataset(testdata_tensor)
```

[42]:
```
# Data loaders
train_iter = torch.utils.data.DataLoader(dataset=train_dataset,
                                         batch_size=32,
                                         shuffle=True)

test_iter = torch.utils.data.DataLoader(dataset=test_dataset,
                                        batch_size=256,
                                        shuffle=False)
```

[43]:
```
testdata_iter = torch.utils.data.DataLoader(dataset=testdata_dataset,
                                            batch_size = 64,
                                            shuffle = False)
```

[44]:
```
for i in testdata_iter:
  print(i)
  break
```

```
[tensor([[    0,     0,     0,  …,    38,  1171,    16],
        [    0,     0,     0,  …,   212,  1676,    89],
        [    0,     0,     0,  …,   153,   131, 12513],
```

```
        …,
        [    0,     0,     0,  …,     5, 12609,  1845],
        [    0,     0,     0,  …,  1468,    53,   814],
        [    0,     0,     0,  …, 11248,    39,    16]])]
```

[45]:
```
for inputs, targets in train_iter:
  print("inputs:", inputs, "shape:", inputs.shape)
  print("targets:", targets, "shape:", targets.shape)
  break
```

```
inputs: tensor([[   0,    0,    0,  …,   97, 1017,  260],
        [   0,    0,    0,  …,  336,  143, 1673],
        [   0,    0,    0,  …,   54,  238,  372],
        …,
        [   0,    0,    0,  …,  305,   17,  127],
        [   0,    0,    0,  …,    5,  153,  919],
        [   0,    0,    0,  …,   38, 4739,   16]]) shape: torch.Size([32,
102])
targets: tensor([1, 0, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 1,
        0, 1, 0, 1, 0, 0, 0, 0]) shape: torch.Size([32])
```

[46]:
```
for inputs, targets in test_iter:
  print("inputs:", inputs)
  print("targets:", targets)
  break
```

```
inputs: tensor([[   0,    0,    0,  …, 6087,   91,  752],
        [   0,    0,    0,  …,   70,   61,  857],
        [   0,    0,    0,  …,  527,   27,  333],
        …,
        [   0,    0,    0,  …,    2, 2787, 1392],
        [   0,    0,    0,  …,  934,   13,  278],
        [   0,    0,    0,  …,    6,  275, 7291]])
targets: tensor([0, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0,
0, 1, 0,
        0, 0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 0, 1, 0, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0,
        1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 1,
        1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 1,
        0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 1,
        1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 0, 1, 1,
        0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1, 0, 0, 1,
        1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1,
        0, 1, 0, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0,
        1, 1, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 1, 0])
```

[47]:
```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

[48]:
```
# Define the model
class RNN(nn.Module):
  def __init__(self, n_vocab, embed_dim, n_hidden, n_rnnlayers, n_outputs):
    super(RNN, self).__init__()
    self.V = n_vocab
    self.D = embed_dim
    self.M = n_hidden
    self.K = n_outputs
    self.L = n_rnnlayers

    self.embed = nn.Embedding(self.V, self.D)
    self.rnn = nn.LSTM(
        input_size=self.D,
        hidden_size=self.M,
        num_layers=self.L,
        batch_first=True)
    self.fc = nn.Linear(self.M, self.K)

  def forward(self, X):
    # initial hidden states
    h0 = torch.zeros(self.L, X.size(0), self.M).to(device)
    c0 = torch.zeros(self.L, X.size(0), self.M).to(device)

    # embedding layer
    # turns word indexes into word vectors
    out = self.embed(X)

    # get RNN unit output
    out, _ = self.rnn(out, (h0, c0))

    # max pool
    out, _ = torch.max(out, 1)

    # we only want h(T) at the final time step
    out = self.fc(out)
    return out
```

[49]:
```
model = RNN(V+1, 300, 100, 1, 3) # V= len(vocab) + 1 for (padding)
model.to(device)
```

[49]:
```
RNN(
  (embed): Embedding(14272, 300)
```

```
  (rnn): LSTM(300, 100, batch_first=True)
  (fc): Linear(in_features=100, out_features=3, bias=True)
)
```

[50]:
```
print(model)
```

```
RNN(
  (embed): Embedding(14272, 300)
  (rnn): LSTM(300, 100, batch_first=True)
  (fc): Linear(in_features=100, out_features=3, bias=True)
)
```

[51]:
```
for name, param in model.named_parameters():
  print(name, param.shape)
```

```
embed.weight torch.Size([14272, 300])
rnn.weight_ih_l0 torch.Size([400, 300])
rnn.weight_hh_l0 torch.Size([400, 100])
rnn.bias_ih_l0 torch.Size([400])
rnn.bias_hh_l0 torch.Size([400])
fc.weight torch.Size([3, 100])
fc.bias torch.Size([3])
```

[52]:
```
learning_rate = 0.01
epochs=65
# STEP 5: INSTANTIATE LOSS CLASS
criterion = nn.CrossEntropyLoss()

# STEP 6: INSTANTIATE OPTIMIZER CLASS

optimizer = torch.optim.SGD(model.parameters(), lr=learning_rate)

# STEP 7: TRAIN THE MODEL

train_losses= np.zeros(epochs)
test_losses= np.zeros(epochs)


for epoch in range(epochs):

  t0= datetime.now()
  train_loss=[]

  model.train()
  for input,targets in train_iter:
    # load input and output to GPU
    input = input.to(device)
```

```
    targets= targets.to(device)

    # forward pass
    output= model(input)
    loss=criterion(output,targets)

    # set gradients to zero
    optimizer.zero_grad()

    # backward pass
    loss.backward()
    optimizer.step()
    train_loss.append(loss.item())

  train_loss=np.mean(train_loss)

  test_loss=[]
  model.eval()
  with torch.no_grad():
    for input,targets in test_iter:
      # load input and output to GPU
      input = input.to(device)
      targets= targets.to(device)


      # forward pass
      output= model(input)
      loss=criterion(output,targets)
      test_loss.append(loss.item())

    test_loss=np.mean(test_loss)

  # save Losses
  train_losses[epoch]= train_loss
  test_losses[epoch]= test_loss
  dt= datetime.now()-t0
  print(f'Epoch {epoch+1}/{epochs}, Train Loss: {train_loss:.4f}    Test Loss:␣
↪{test_loss:.4f}, Duration: {dt}')
```

```
Epoch 1/65, Train Loss: 0.7063    Test Loss: 0.6634, Duration: 0:00:01.490683
Epoch 2/65, Train Loss: 0.6409    Test Loss: 0.6506, Duration: 0:00:01.279411
Epoch 3/65, Train Loss: 0.6327    Test Loss: 0.6427, Duration: 0:00:01.264833
Epoch 4/65, Train Loss: 0.6277    Test Loss: 0.6407, Duration: 0:00:01.277945
Epoch 5/65, Train Loss: 0.6247    Test Loss: 0.6371, Duration: 0:00:01.263438
Epoch 6/65, Train Loss: 0.6206    Test Loss: 0.6351, Duration: 0:00:01.262783
Epoch 7/65, Train Loss: 0.6179    Test Loss: 0.6330, Duration: 0:00:01.276571
Epoch 8/65, Train Loss: 0.6139    Test Loss: 0.6349, Duration: 0:00:01.376868
```

```
Epoch 9/65, Train Loss: 0.6096    Test Loss: 0.6329, Duration: 0:00:01.273635
Epoch 10/65, Train Loss: 0.6049    Test Loss: 0.6292, Duration: 0:00:01.292015
Epoch 11/65, Train Loss: 0.6004    Test Loss: 0.6242, Duration: 0:00:01.270034
Epoch 12/65, Train Loss: 0.5956    Test Loss: 0.6234, Duration: 0:00:01.270217
Epoch 13/65, Train Loss: 0.5905    Test Loss: 0.6188, Duration: 0:00:01.261470
Epoch 14/65, Train Loss: 0.5850    Test Loss: 0.6175, Duration: 0:00:01.281212
Epoch 15/65, Train Loss: 0.5780    Test Loss: 0.6135, Duration: 0:00:01.269371
Epoch 16/65, Train Loss: 0.5718    Test Loss: 0.6135, Duration: 0:00:01.272129
Epoch 17/65, Train Loss: 0.5650    Test Loss: 0.6125, Duration: 0:00:01.266158
Epoch 18/65, Train Loss: 0.5567    Test Loss: 0.6045, Duration: 0:00:01.279906
Epoch 19/65, Train Loss: 0.5509    Test Loss: 0.6016, Duration: 0:00:01.270779
Epoch 20/65, Train Loss: 0.5435    Test Loss: 0.6050, Duration: 0:00:01.268913
Epoch 21/65, Train Loss: 0.5357    Test Loss: 0.5966, Duration: 0:00:01.266107
Epoch 22/65, Train Loss: 0.5281    Test Loss: 0.5949, Duration: 0:00:01.260385
Epoch 23/65, Train Loss: 0.5212    Test Loss: 0.5959, Duration: 0:00:01.279710
Epoch 24/65, Train Loss: 0.5136    Test Loss: 0.5911, Duration: 0:00:01.279293
Epoch 25/65, Train Loss: 0.5057    Test Loss: 0.5890, Duration: 0:00:01.273019
Epoch 26/65, Train Loss: 0.4976    Test Loss: 0.5910, Duration: 0:00:01.285706
Epoch 27/65, Train Loss: 0.4908    Test Loss: 0.5839, Duration: 0:00:01.273856
Epoch 28/65, Train Loss: 0.4829    Test Loss: 0.5831, Duration: 0:00:01.275602
Epoch 29/65, Train Loss: 0.4751    Test Loss: 0.5814, Duration: 0:00:01.266347
Epoch 30/65, Train Loss: 0.4673    Test Loss: 0.5892, Duration: 0:00:01.275406
Epoch 31/65, Train Loss: 0.4591    Test Loss: 0.5913, Duration: 0:00:01.271679
Epoch 32/65, Train Loss: 0.4517    Test Loss: 0.5798, Duration: 0:00:01.287203
Epoch 33/65, Train Loss: 0.4433    Test Loss: 0.5784, Duration: 0:00:01.270036
Epoch 34/65, Train Loss: 0.4353    Test Loss: 0.5775, Duration: 0:00:01.293369
Epoch 35/65, Train Loss: 0.4275    Test Loss: 0.5769, Duration: 0:00:01.266690
Epoch 36/65, Train Loss: 0.4196    Test Loss: 0.5755, Duration: 0:00:01.272767
Epoch 37/65, Train Loss: 0.4108    Test Loss: 0.5750, Duration: 0:00:01.284099
Epoch 38/65, Train Loss: 0.4030    Test Loss: 0.5748, Duration: 0:00:01.296037
Epoch 39/65, Train Loss: 0.3944    Test Loss: 0.5868, Duration: 0:00:01.273962
Epoch 40/65, Train Loss: 0.3865    Test Loss: 0.5791, Duration: 0:00:01.263577
Epoch 41/65, Train Loss: 0.3785    Test Loss: 0.5748, Duration: 0:00:01.281850
Epoch 42/65, Train Loss: 0.3704    Test Loss: 0.5739, Duration: 0:00:01.295523
Epoch 43/65, Train Loss: 0.3617    Test Loss: 0.5751, Duration: 0:00:01.277871
Epoch 44/65, Train Loss: 0.3537    Test Loss: 0.5756, Duration: 0:00:01.276830
Epoch 45/65, Train Loss: 0.3448    Test Loss: 0.5807, Duration: 0:00:01.281492
Epoch 46/65, Train Loss: 0.3364    Test Loss: 0.5831, Duration: 0:00:01.288589
Epoch 47/65, Train Loss: 0.3287    Test Loss: 0.5778, Duration: 0:00:01.285233
Epoch 48/65, Train Loss: 0.3202    Test Loss: 0.5856, Duration: 0:00:01.265179
Epoch 49/65, Train Loss: 0.3122    Test Loss: 0.5801, Duration: 0:00:01.275257
Epoch 50/65, Train Loss: 0.3048    Test Loss: 0.5809, Duration: 0:00:01.298449
Epoch 51/65, Train Loss: 0.2965    Test Loss: 0.5850, Duration: 0:00:01.278326
Epoch 52/65, Train Loss: 0.2882    Test Loss: 0.5896, Duration: 0:00:01.301009
Epoch 53/65, Train Loss: 0.2806    Test Loss: 0.5853, Duration: 0:00:01.309506
Epoch 54/65, Train Loss: 0.2727    Test Loss: 0.5959, Duration: 0:00:01.300037
Epoch 55/65, Train Loss: 0.2648    Test Loss: 0.6012, Duration: 0:00:01.303709
Epoch 56/65, Train Loss: 0.2581    Test Loss: 0.5901, Duration: 0:00:01.289537
```

```
Epoch 57/65, Train Loss: 0.2504    Test Loss: 0.5931, Duration: 0:00:01.289044
Epoch 58/65, Train Loss: 0.2434    Test Loss: 0.6019, Duration: 0:00:01.292184
Epoch 59/65, Train Loss: 0.2357    Test Loss: 0.5955, Duration: 0:00:01.291926
Epoch 60/65, Train Loss: 0.2289    Test Loss: 0.5997, Duration: 0:00:01.288225
Epoch 61/65, Train Loss: 0.2225    Test Loss: 0.5999, Duration: 0:00:01.285225
Epoch 62/65, Train Loss: 0.2153    Test Loss: 0.6019, Duration: 0:00:01.271857
Epoch 63/65, Train Loss: 0.2086    Test Loss: 0.6011, Duration: 0:00:01.269726
Epoch 64/65, Train Loss: 0.2025    Test Loss: 0.6129, Duration: 0:00:01.270997
Epoch 65/65, Train Loss: 0.1962    Test Loss: 0.6175, Duration: 0:00:01.280889
```

[53]:
```python
# Accuracy- write a function to get accuracy
# use this function to get train/test accuracy and print accuracy
def get_accuracy(train_iter, test_iter, model):
  model.eval()
  with torch.no_grad():
    correct_train=correct_test=0
    total_train=total_test=0

    for input, targets in train_iter:
      input= input.to(device)
      targets= targets.to(device)
      #input = input.view(-1, 784)
      output=model(input)
      _,indices = torch.max(output,dim=1)
      correct_train+= (targets==indices).sum().item()
      total_train += targets.shape[0]

    train_acc= correct_train/total_train

    for input, targets in test_iter:
      input= input.to(device)
      targets= targets.to(device)
      #input = input.view(-1, 784)
      output=model(input)
      _,indices = torch.max(output,dim=1)
      correct_test+= (targets==indices).sum().item()
      total_test += targets.shape[0]

    test_acc= correct_test/total_test
    return train_acc, test_acc
```

[54]:
```python
train_acc, test_acc = get_accuracy(train_iter, test_iter, model)
print(f'Train acc: {train_acc:.4f},\t Test acc: {test_acc:.4f}')
```

```
Train acc: 0.9633,       Test acc: 0.7204
```

[59]:
```
# Write a function to get predictions

def get_predictions(test_iter, model):
  model.eval()
  with torch.no_grad():
    predictions= np.array([])
    y_test= np.array([])

    for batch in test_iter:

      output=model(batch.data)
      _,indices = torch.max(output,dim=1)
      predictions=np.concatenate((predictions,indices.cpu().numpy()))
      y_test = np.concatenate((y_test,batch.label.cpu().numpy()))

  return y_test, predictions
```

[60]:
```
predictions = get_predictions(testdata_iter, model)
```

```
␣
↪---------------------------------------------------------------------------

       AttributeError                            Traceback (most recent call␣
↪last)

       <ipython-input-60-97df6b2fb9a6> in <module>()
   ----> 1 predictions = get_predictions(testdata_iter, model)


       <ipython-input-59-9a95447eeb0d> in get_predictions(test_iter, model)
         9     for batch in test_iter:
        10
   ---> 11       output=model(batch.data)
        12       _,indices = torch.max(output,dim=1)
        13       predictions=np.concatenate((predictions,indices.cpu().numpy()))


       AttributeError: 'list' object has no attribute 'data'
```

[ ]:
```
y_test, predictions=get_predictions(test_iter, model)
```

[ ]:
```
predictions.max()
```

[ ]:
```
# We are using confusion metrics from sklearn
# we are done with model building and predictions
```

```
# let us convert test data set to numpy arrays now

#y_test=y_test.numpy()

cm=confusion_matrix(y_test,predictions)
cm
```

[ ]:
```
# Write a function to print confusion matrix
# plot confusion matrix
# need to import confusion_matrix from sklearn for this function to work
# need to import seaborn as sns
# import seaborn as sns
# import matplotlib.pyplot as plt
# from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(y_true,y_pred,normalize=None):
  cm=confusion_matrix(y_true,y_pred,normalize=normalize)
  fig, ax = plt.subplots(figsize=(6,5))
  if normalize == None:
    fmt='d'
    fig.suptitle('Confusion matrix without Normalization', fontsize=12)

  else :
    fmt='0.2f'
    fig.suptitle('Normalized confusion matrix', fontsize=12)

  ax=sns.heatmap(cm,cmap=plt.cm.Blues,annot=True,fmt=fmt)
  ax.axhline(y=0, color='k',linewidth=1)
  ax.axhline(y=cm.shape[1], color='k',linewidth=2)
  ax.axvline(x=0, color='k',linewidth=1)
  ax.axvline(x=cm.shape[0], color='k',linewidth=2)

  ax.set_xlabel('Predicted label', fontsize=12)
  ax.set_ylabel('True label', fontsize=12)
```

[ ]:
```
plot_confusion_matrix(y_test,predictions)
```

[ ]:
```
predictions.shape
```

[ ]:
```
test_data = pd.read_csv('/content/test.csv', encoding = "ISO-8859-1")
```

[ ]:
```
data.shape
```

[ ]:
```
pd.DataFrame({'Id': test_data.id[:2804], 'target': predictions}).
↪to_csv('solution_base.csv', index =False)
```

[ ]:
```
pwd
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: