# fake news notebook

September 27, 2024

[68]:
```
import time
import pandas as pd
import numpy as np
import re
import glob
import torch
import itertools
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
import pandas as pd
from torch.utils.data import Dataset, DataLoader
#from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from sklearn import preprocessing
from numpy import zeros
import tensorflow as tf
from tensorflow.keras.preprocessing.text import Tokenizer
```

[69]:
```
data2=pd.read_csv('news.csv')
data2.head()
```

```
C:\Users\Lakys\AppData\Local\Temp\ipykernel_2692\1704981109.py:1: DtypeWarning:
Columns (24,25,26,27,28,29,30,31,32,33,34,35,36,37,38,39,40,41,42,43,44,45,46,47
,48,49,50,51,52,53,54,55,56,57,58,59,60,61,62,63,64,65,66,67,68,69,70,71,72,73,7
4,75,76,77,78,79,80,81,82,83,84,85,86,87,88,89,90,91,92,93,94,95,96,97,98,99,100
,101,102,103,104,105,106,107,108,109,110,111,112,113,114,115,116,117,118,119,120
,121,122,123,124,125,126,127,128,129,130,131,132,133,134,135,136,137,138,139,140
) have mixed types. Specify dtype option on import or set low_memory=False.
  data2=pd.read_csv('news.csv')
```

[69]:
```
  Unnamed: 0                                              title  \
0       8476                           You Can Smell Hillary's Fear
1      10294  Watch The Exact Moment Paul Ryan Committed Pol…
2       3608        Kerry to go to Paris in gesture of sympathy
3      10142  Bernie supporters on Twitter erupt in anger ag…
4        875   The Battle of New York: Why This Primary Matters
```

```
                                                text label Unnamed: 4  \
0  Daniel Greenfield, a Shillman Journalism Fello…  FAKE        NaN
1  Google Pinterest Digg Linkedin Reddit Stumbleu…  FAKE        NaN
2  U.S. Secretary of State John F. Kerry said Mon…  REAL        NaN
3  — Kaydee King (@KaydeeKing) November 9, 2016 T…  FAKE        NaN
4  It's primary day in New York and front-runners…  REAL        NaN

  Unnamed: 5 Unnamed: 6 Unnamed: 7 Unnamed: 8 Unnamed: 9  … Unnamed: 131  \
0        NaN        NaN        NaN        NaN        NaN  …          NaN
1        NaN        NaN        NaN        NaN        NaN  …          NaN
2        NaN        NaN        NaN        NaN        NaN  …          NaN
3        NaN        NaN        NaN        NaN        NaN  …          NaN
4        NaN        NaN        NaN        NaN        NaN  …          NaN

  Unnamed: 132 Unnamed: 133 Unnamed: 134 Unnamed: 135 Unnamed: 136  \
0          NaN          NaN          NaN          NaN          NaN
1          NaN          NaN          NaN          NaN          NaN
2          NaN          NaN          NaN          NaN          NaN
3          NaN          NaN          NaN          NaN          NaN
4          NaN          NaN          NaN          NaN          NaN

  Unnamed: 137 Unnamed: 138 Unnamed: 139 Unnamed: 140
0          NaN          NaN          NaN          NaN
1          NaN          NaN          NaN          NaN
2          NaN          NaN          NaN          NaN
3          NaN          NaN          NaN          NaN
4          NaN          NaN          NaN          NaN

[5 rows x 141 columns]
```

```
[70]: data2=pd.concat([pd.Series(data2["text"], name="text"), pd.
       ↪Series(data2["label"], name="label")], axis=1)
      data2.head()
```

```
[70]:                                                 text label
      0  Daniel Greenfield, a Shillman Journalism Fello…  FAKE
      1  Google Pinterest Digg Linkedin Reddit Stumbleu…  FAKE
      2  U.S. Secretary of State John F. Kerry said Mon…  REAL
      3  — Kaydee King (@KaydeeKing) November 9, 2016 T…  FAKE
      4  It's primary day in New York and front-runners…  REAL
```

```
[71]: len(data2)
```

```
[71]: 7795
```

```
[72]: data3=data2[(data2["label"]=="FAKE") | (data2["label"]=="REAL")]
```

[73]:
```
set(list(data3["label"]))
```

[73]:
```
{'FAKE', 'REAL'}
```

[74]:
```
data3["label"].value_counts()
```

[74]:
```
label
REAL    3161
FAKE    3154
Name: count, dtype: int64
```

[75]:
```
data3
```

[75]:
```
                                                    text label
0     Daniel Greenfield, a Shillman Journalism Fello…  FAKE
1     Google Pinterest Digg Linkedin Reddit Stumbleu…  FAKE
2     U.S. Secretary of State John F. Kerry said Mon…  REAL
3     — Kaydee King (@KaydeeKing) November 9, 2016 T…  FAKE
4     It's primary day in New York and front-runners…  REAL
…                                                   …    …
7790  The State Department told the Republican Natio…  REAL
7791  The ‘P’ in PBS Should Stand for ‘Plutocratic’ …  FAKE
7792   Anti-Trump Protesters Are Tools of the Oligar…  FAKE
7793  ADDIS ABABA, Ethiopia —President Obama convene…  REAL
7794  Jeb Bush Is Suddenly Attacking Trump. Here's W…  REAL

[6315 rows x 2 columns]
```

[76]:
```
le = preprocessing.LabelEncoder()
labe=data3["label"]

labels_encoded=le.fit(labe)
labels_=le.transform(labe)
```

[77]:
```
list(zip(data3["label"][:5], labels_[:5]))
```

[77]:
```
[('FAKE', 0), ('FAKE', 0), ('REAL', 1), ('FAKE', 0), ('REAL', 1)]
```

[78]:
```
set(labels_)
```

[78]:
```
{0, 1}
```

[79]:
```
text_for_this=[]
for i in data3["text"]:
    text_for_this.append(' '.join(re.findall(r'[a-zA-Z]+', str(i))))
```

[80]:
```
#bytestring_=[i.encode() for i in text_for_this]
```

[81]:

```
datalist1=glob.glob('glove_file_*')
```

[82]:

```
combined_datalist=[pd.read_csv(i) for i in datalist1]
words_=list(itertools.chain.from_iterable([list(i.word) for i in␣
 ↪combined_datalist]))
values_=list(itertools.chain.from_iterable([list(i.values_) for i in␣
 ↪combined_datalist]))
values_=[np.array(re.findall(r'[\d\.]{1,8}',str(i)),dtype='float32') for i in␣
 ↪values_]
values_2=[i[:100] for i in values_]
embed_index=dict(zip(words_,values_2))
```

[83]:

```
word_tokenizer = Tokenizer()
```

[84]:

```
word_tokenizer.fit_on_texts(text_for_this)
```

[85]:

```
embedded_skill=word_tokenizer.texts_to_sequences(text_for_this)
```

[86]:

```
vocab_length = len(word_tokenizer.word_index) + 1

embedding_matrix = zeros((vocab_length, 100))
for word, index in word_tokenizer.word_index.items():
    embedding_vector = embed_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

len_sent=list([len(i) for i in embedded_skill])
length_long_sentence=max(len_sent)

padded_sentences = pad_sequences(embedded_skill, length_long_sentence,␣
 ↪padding='post')

validation_split=.1
indices=np.arange(np.array(padded_sentences,dtype=object).shape[0])
np.random.shuffle(indices)

data_rand=padded_sentences[indices]
```

[87]:

```
labels_rand=np.array(labels_)[indices]
```

[88]:

```
val_sample=int(validation_split * data3.shape[0])
```

[89]:

```
X_train=data_rand[:-val_sample]
y_train=labels_rand[:-val_sample]
X_test=data_rand[-val_sample:]
y_test=labels_rand[-val_sample:]
```

[90]:

```
x_train = torch.tensor(X_train, dtype=torch.long)
y_train = torch.tensor(y_train, dtype=torch.long)
x_cv = torch.tensor(X_test, dtype=torch.long)
y_cv = torch.tensor(y_test, dtype=torch.long)

train = torch.utils.data.TensorDataset(x_train, y_train)
valid = torch.utils.data.TensorDataset(x_cv, y_cv)

train_loader = torch.utils.data.DataLoader(train, batch_size=32, shuffle=True)
valid_loader = torch.utils.data.DataLoader(valid, batch_size=32, shuffle=False)
```

[91]:

```
class BiLSTM(nn.Module):
  def __init__(self):
    super(BiLSTM, self).__init__()
    self.hidden_size = 5
    drp = 0.4
    n_classes = len(le.classes_)
    self.embedding = nn.Embedding(max_features, embed_size)
    self.embedding.weight = nn.Parameter(torch.tensor(embedding_matrix, dtype=torch.float32))
    self.embedding.weight.requires_grad = True
    self.lstm = nn.LSTM(embed_size, self.hidden_size, bidirectional=True, batch_first=True)
    self.linear = nn.Linear(self.hidden_size*4 , 1, bias=False)
    self.relu = nn.ReLU()
    self.out = nn.Linear(1, n_classes, bias=False)


  def forward(self, x):
    h_embedding = self.embedding(x)
    h_lstm, _ = self.lstm(h_embedding)
    avg_pool = torch.mean(h_lstm, 1)
    max_pool, _ = torch.max(h_lstm, 1)
    conc = torch.cat(( avg_pool, max_pool), 1)
    conc = self.relu(self.linear(conc))
    out = self.out(conc)
    return out
```

[92]:

```
embed_size=100
max_features=vocab_length
n_epochs = 4
model = BiLSTM()
loss_fn = nn.CrossEntropyLoss(reduction='mean')
optimizer = torch.optim.AdamW(filter(lambda p: p.requires_grad, model.parameters()), lr=0.001, weight_decay=.0001)
model.cpu()
```

[92]:
```
BiLSTM(
  (embedding): Embedding(63464, 100)
  (lstm): LSTM(100, 5, batch_first=True, bidirectional=True)
  (linear): Linear(in_features=20, out_features=1, bias=False)
  (relu): ReLU()
  (out): Linear(in_features=1, out_features=2, bias=False)
)
```

[93]:
```
output=[]
val_preds_=[]
for epoch in range(n_epochs):
    start_time = time.time()

    model.train()

    avg_loss = 0.
    for i, (x_batch, y_batch) in enumerate(train_loader):
      y_pred = model(x_batch)
      loss = loss_fn(y_pred, y_batch)
      optimizer.zero_grad()
      loss.backward()
      optimizer.step()
      avg_loss += loss.item() / len(train_loader)


    avg_val_loss = 0.
    val_preds=[]
    val_preds_.append(val_preds)
    for i, (x_batch, y_batch) in enumerate(valid_loader):

      y_pred = model(x_batch).detach()
      avg_val_loss += loss_fn(y_pred, y_batch).item() / len(valid_loader)
      val_preds.append(F.sigmoid(y_pred).cpu().numpy())

    elapsed_time = time.time() - start_time

    print(f'epoch:{epoch}, average loss: {avg_loss}, average validation loss:
 ↪{avg_val_loss}, elapsed time:{elapsed_time}')
    output.append('Epoch {}/{} \t loss={:.4f} \t val_loss={:.4f}  \t time={:.
 ↪2f}s'.format(
                epoch + 1, n_epochs, avg_loss, avg_val_loss, elapsed_time))
```

```
epoch:0, average loss: 0.6915454519598674, average validation
loss:0.6828594446182251, elapsed time:76.54161190986633
epoch:1, average loss: 0.6293323612614966, average validation
loss:0.5336180493235588, elapsed time:77.69974327087402
epoch:2, average loss: 0.5089707277464061, average validation
loss:0.47241965532302865, elapsed time:75.15815925598145
```

```
epoch:3, average loss: 0.4312521821997139, average validation
loss:0.4586188316345214, elapsed time:75.19009327888489
```

[94]:
```
val_accuracy=[]
for i in range(len(val_preds_)):
    for p in range(len(val_preds_[i])):
        val_cov_=[]
        length=len(val_cov_)
        for s in val_preds_[i][p]:
            val_cov=[]
            for l in s:
                if l>.5:
                    val_cov.append(int(0))
                else:
                    val_cov.append(int(1))
            val_cov_.append(val_cov[np.array(val_cov).argmax()])
        accuracy=len([i for i in val_cov_ if i==1])/len(val_cov_)
    val_accuracy.append(accuracy)
```

[95]:
```
output_df=pd.DataFrame([i.split('\t') for i in output])
output_df["validation_accuracy"]=[round(i,4) for i in val_accuracy]
output_df.
 ↪columns=['epoch','loss','validation_loss','time','validation_accuracy']
output_df
```

[95]:
```
         epoch          loss     validation_loss          time  \
0  Epoch 1/4    loss=0.6915    val_loss=0.6829     time=76.54s
1  Epoch 2/4    loss=0.6293    val_loss=0.5336     time=77.70s
2  Epoch 3/4    loss=0.5090    val_loss=0.4724     time=75.16s
3  Epoch 4/4    loss=0.4313    val_loss=0.4586     time=75.19s

   validation_accuracy
0                0.087
1                1.000
2                1.000
3                1.000
```

[96]:
```
f'Average Accuracy:  {round(np.mean(np.array(val_accuracy)),2)*100}%'
```

[96]:
```
'Average Accuracy:  77.0%'
```

[97]:
```
test_data=pd.read_csv("test.csv", encoding="latin1")
```

[98]:
```
text_for_test=[]
for i in test_data["text"]:
    text_for_test.append(' '.join(re.findall(r'[a-zA-Z]+', str(i))))
word_tokenizer = Tokenizer()
word_tokenizer.fit_on_texts(text_for_test)
```

```
embedded_skill=word_tokenizer.texts_to_sequences(text_for_test)

vocab_length = len(word_tokenizer.word_index) + 1

embedding_matrix = zeros((vocab_length, 100))
for word, index in word_tokenizer.word_index.items():
    embedding_vector = embed_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector

len_sent=list([len(i) for i in embedded_skill])
length_long_sentence=max(len_sent)

padded_sentences_ = pad_sequences(embedded_skill, length_long_sentence,␣
 ↪padding='post')
```

[99]:

```
test_data1 = torch.tensor(padded_sentences_, dtype=torch.long)
test_data_ = torch.utils.data.TensorDataset(test_data1)
test_data_loader = torch.utils.data.DataLoader(test_data_, batch_size=32,␣
 ↪shuffle=True)
```

[100]:

```
pred = []
model.eval()

for inputs in test_data_loader:
    y_pred = model(inputs[0]).detach()
    pred.append(y_pred)
```

[101]:

```
prediction_class=[]
for i in range(len(pred)):

    for p in range(len(pred[i])):
        val_cov_=[]
        length=len(val_cov_)
        for s in val_preds_[i][p]:
            val_cov=[]

            for l in s:
                if l>.5:
                    val_cov.append(int(0))
                else:
                    val_cov.append(int(1))
        val_cov_.append(val_cov[np.array(val_cov).argmax()])
        prediction_class.append(val_cov_)
```

[102]:

```
classification_dictionary={1:"REAL", 0:"FAKE"}
```

```
[103]: prediction_class=[classification_dictionary[i[0]] for i in prediction_class]
```

```
[104]: test_data["Prediction Class"]=prediction_class
```

```
[105]: test_data
```

```
[105]:                                                  text Prediction Class
        0  A roundup of some of the most popular but comp…             FAKE
        1  The Biden administration is working on a new d…             FAKE
        2  Pope Francis, in the first-ever papal address …             FAKE
```

```
[ ]:
```