**FEDERAL FUNDS RATE PREDICTION: BERT SEQUENCE CLASSIFICATION ON FED CORPORA**

##### author: theo dimitrasopoulos\* **|** advisor: zachary feinstein\*

\* Department of Financial Engineering; Stevens Institute of Technology Babbio School of Business

Abstract — This paper focuses on extracting sentiment from Federal Reserve corpora in order to predict the federal funds rate. More specifically, it investigates minutes, statements, speeches and testimonies delivered by the Federal Reserve boards since 1980, which are preprocessed in short chunks that are then benchmarked against the Loughran-McDonald dictionary of financial terms for sentiment. Then, a base BERT model is trained on the preprocessed dataset and train/validation losses are recorded to estimate the accuracy of the model. The result highlights the importance of a wealth of data to train such a model. Additional finetuning or the use of a pre-trained BERT-model can provide insightful commentary on the prediction of the Federal Funds rate for use in trading strategies (mean-reversion, moving average etc.) and other applications of NLP.

Keywords — Federal Reserve, Federal Funds Rate, Interest Rate, Prediction, Sequence Classification, Bidirectional Encoders, Transformers,

# introduction

The Federal Open Market Committee (FOMC) meetings aim to discuss, implement and communicate monetary policy to the markets. The Federal Funds Rate, or the formal definition of the well-known Fed interest rate could be considered a latent feature in an NLP model which attempts to extract sentiment from the data and predict the direction of the interest rate at future dates.

# transformers

Transformers are a Deep Learning innovation that builds beyond recurrent neural networks with the ultimate goal of reducing processing times of even larger datasets, with equal or higher accuracy [1]. Gated RNN’s were the most sophisticated model before the introduction of transformers, require that the text tokens be processed sequentially, which greatly reduces the ability to parallelize the task. In the case of a transformer, a encoder-decoder architecture is utilized in order to enlarge the scope of data analysis and allow for bidirectional processing without the need to account for the beginning and end of a token [1].

The transformer model is structured as one large matrix calculation as follows,

where Q,K,V are the vectors the of the rows of the tokens fed into the model.

In the case of BERT in particular, the innovation is bidirectional training, or the encoder-decoder architecture mentioned above. Similarly to Next Sentence Classification, the classification task modeled in this research is performed by adding a classification layer on the transformer output for the [CLS] tokens [4].

# results & discussion

The data was sectioned in 200-word segments in order to ease processing and was grouped by speaker. The main speakers chosen were the chairpersons of the Federal Reserve, while all other speaker content was dropped from the data. Sentiment was added to each of the word segments using the Loughran-McDonald Dictionary of Financial Terms to identify the general stance towards interest rates (increase, decrease or no change) [3].



Figure Segmented data grouped by Speaker (Speakers other than the Federal Reserve Chairpersons were dropped).

BERT was then deployed on the preprocessed data to evaluate the model on the sourced data from the Federal Reserve Archives. Fig. 1 shows the training/validation loss result after 3 rounds of training.

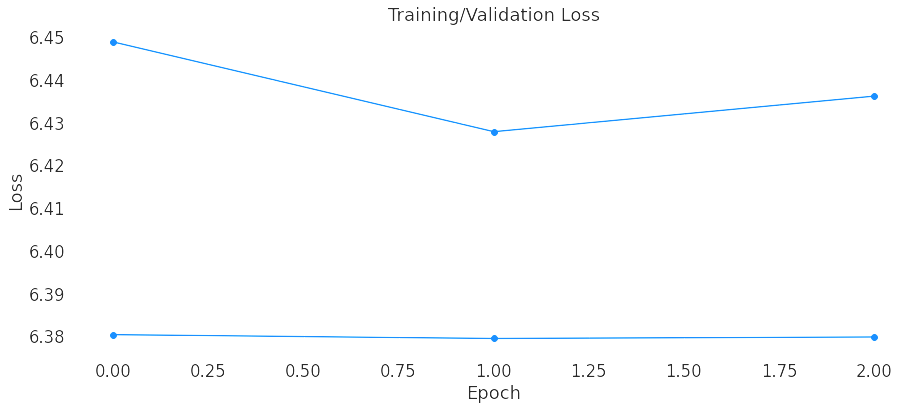


Figure Training/Validation during BERT's training on the Fed Data.

The model eventually became very cumbersome to the GPU within Google Collaboratory, leading to various runtime crashes past the 3rd fold validation. In cases where a TPU or more capable processing engine is available, the training/validation loss could be further decreased. The ultimate bottleneck however is the availability of data. This might make the selection of a pre-trained model more prudent.

# next steps

As seen in the analysis above, training a BERT model requires a wealth of data. In the next steps of this research, it is an imperative to source more data and perform more thorough preprocessing with various intervals in the sectioning for better parsing. There is also a lot of room for using pre-trained BERT models and fine-tuning the model’s hyperparameters. Lastly, it is worth exploring other models in parallel with BERT in order to identify the one with the highest accuracy before moving forward with various integrations with trading systems (one potential path forward here is to inform a mean-reversion strategy with the sentiment extracted from the NLP methodologies).

# acknowledgments

I would like to thank professor Feinstein for his invaluable guidance through his teachings in FE 690: Machine Learning in Finance, and his continued support in identifying the pain points and potential developments in the project proposal; The Stevens Institute of Technology Department of Financial Engineering for its provision of laptops and technological equipment to undertake this project; The Google Collaboratory for providing a streamlined platform for the testing and execution of the models; and lastly, my colleagues at the Nomura Research Institute for providing me with helpful tips and guidance on how to preprocess the data in a sound way and execute the model in this research.

# references

## [1] Jacob Devlin, Ming-Wei Chang. Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing. Accessed October 25th, 2020. [Online](http://ai.googleblog.com/2018/11/open-sourcing-bert-state-of-art-pre.html).

[2] Transformers. HuggingFace Documentation. Accessed October 20th, 2020. [Online](https://huggingface.co/transformers/).

[3] Takahashi, Yuki. Analyze Central Bank Announcements. Nomura Research Institute. Accessed October 20th, 2020. [Online](https://medium.com/@yuki678/fed-speak-nlp-ml-from-tfidf-to-bert-for-sentiment-classification-2350195baab2).

[4] Horev, Rani. BERT Explained: State-of-the-art language model for NLP. Accessed October 28th 2020. [Online](https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270).

# appendix

**Code for data preprocessing and BERT Sequence Classification Training:**

import sys

IN\_COLAB = 'google.colab' in sys.modules

IN\_COLAB

if IN\_COLAB:

from google.colab import drive

drive.mount('/content/drive', force\_remount=True)

!pip install numpy

!pip install pandas

!pip install tqdm

!pip install torch

!pip install scikit-plot

!pip install transformers

import pprint

import numpy as np

import pandas as pd

import datetime as dt

import os

import codecs

import io

from lxml import etree

from dateutil.relativedelta import \*

import seaborn as sns

import matplotlib.pyplot as plt

import matplotlib.ticker as ticker

import re

import pickle

from tqdm.notebook import tqdm

import nltk

from torch.utils.data import (DataLoader, RandomSampler, SequentialSampler, TensorDataset)

from transformers import BertTokenizer, BertForSequenceClassification, BertModel

def get\_word\_count(x):

x = x.replace("[SECTION]", "")

return len(re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', x))

def extract\_r\_change(x):

if type(x) is str:

try:

x = dt.datetime.strptime(x, '%Y-%m-%d')

except:

return None

if x in calendar.index:

return calendar.loc[x]['RateDecision']

else:

return None

def extract\_r(x):

if type(x) is str:

try:

x = dt.datetime.strptime(x, '%Y-%m-%d')

except:

return None

if x in calendar.index:

return calendar.loc[x]['Rate']

else:

return None

def meeting\_new(x):

if type(x) is str:

try:

x = dt.datetime.strptime(x, '%Y-%m-%d')

print(type(x))

except:

return None

x = x + dt.timedelta(days=2)

calendar.sort\_index(ascending=True, inplace=True)

if calendar['date'].iloc[0] > x:

return None

else:

for i in range(len(calendar)):

if x < calendar['date'].iloc[i]:

return calendar['date'].iloc[i]

return None

def chair(x):

if type(x) is str:

try:

x = dt.datetime.strftime(x, '%Y-%m-%d')

print(type(x))

except:

return None

chairr = chairs.loc[chairs['FromDate'] <= x].loc[x <= chairs['ToDate']]

return list(chairr.FirstName)[0] + " " + list(chairr.Surname)[0]

def preprocess(df, doc\_type):

if doc\_type in ('statement', 'minutes', 'press', 'meeting\_script'):

is\_meeting\_doc = True

elif doc\_type in ('speech', 'testimony'):

is\_meeting\_doc = False

else:

return None

dict = {

'type': doc\_type,

'date': df['date'],

'title': df['title'],

'speaker': df['speaker'],

'word\_count': df['contents'].map(get\_word\_count),

'decision': df['date'].map(lambda x: extract\_r\_change(x) if is\_meeting\_doc else None),

'rate': df['date'].map(lambda x: extract\_r(x) if is\_meeting\_doc else None),

'next\_meeting': df['date'].map(meeting\_new),

'decision\_n': df['date'].map(meeting\_new).map(extract\_r\_change),

'next\_rate': df['date'].map(meeting\_new).map(extract\_r),

'text': df['contents'].map(lambda x: x.replace('\n','').replace('\r','').strip()),

'text\_sections': df['contents'].map(lambda x: x.replace('\n','').replace('\r','').strip().split("[SECTION]")),

'processed': df['contents']

}

new\_df = pd.DataFrame(dict)

new\_df['decision'] = new\_df['decision'].astype('Int8')

new\_df['decision\_n'] = new\_df['decision\_n'].astype('Int8')

return new\_df

def split(text, split\_len=200, overlap=50):

l\_total = []

words = re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', text)

if len(words) < split\_len:

n = 1

else:

n = (len(words) - overlap) // (split\_len - overlap) + 1

for i in range(n):

l\_parcial = words[(split\_len - overlap) \* i: (split\_len - overlap) \* i + split\_len]

l\_total.append(" ".join(l\_parcial))

return l\_total

def split\_df(df, split\_len=200, overlap=50):

split\_data\_list = []

for i, row in tqdm(df.iterrows(), total=df.shape[0]):

text\_list = split(row["text"], split\_len, overlap)

for text in text\_list:

row['text'] = text

row['word\_count'] = len(re.findall(r'\b([a-zA-Z]+n\'t|[a-zA-Z]+\'s|[a-zA-Z]+)\b', text))

split\_data\_list.append(list(row))

split\_df = pd.DataFrame(split\_data\_list, columns=df.columns)

return split\_df

chairs = pd.DataFrame(

data=[["Volcker", "Paul", dt.datetime(1979,8,1), dt.datetime(1987,8,1)],["Greenspan", "Alan", dt.datetime(1987,8,1), dt.datetime(2006,1,31)],["Bernanke", "Ben", dt.datetime(2006,2,1), dt.datetime(2014,1,31)],["Yellen", "Janet", dt.datetime(2014,2,1), dt.datetime(2018,1,31)],["Powell", "Jerome", dt.datetime(2018,2,2), dt.datetime(2022,2,2)]],

columns=["Surname", "FirstName", "FromDate", "ToDate"])

chairs

file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/calendar.pickle', 'rb')

#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/calendar.pickle', 'rb')

calendar = pickle.load(file)

file.close()

calendar

file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/statement.pickle', 'rb')

#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/statement.pickle', 'rb')

statement\_df = pickle.load(file)

file.close()

statement\_df

file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/minutes.pickle', 'rb')

#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/minutes.pickle', 'rb')

minutes\_df = pickle.load(file)

file.close()

minutes\_df

file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/speech.pickle', 'rb')

#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/speech.pickle', 'rb')

speech\_df = pickle.load(file)

file.close()

speech\_df

file = open('/content/drive/My Drive/Colab Notebooks/proj2/data/FOMC/testimony.pickle', 'rb')

#file = open('C:/Users/theon/Desktop/proj2/data/FOMC/testimony.pickle', 'rb')

testimony\_df = pickle.load(file)

file.close()

testimony\_df

statement\_clean = preprocess(statement\_df, 'statement')

minutes\_clean = preprocess(minutes\_df, 'minutes')

speech\_clean = preprocess(speech\_df, 'speech')

testimony\_clean = preprocess(testimony\_df, 'testimony')

testimony\_sections = split\_df(statement\_clean)

minutes\_sections = split\_df(minutes\_clean)

testimony\_sections\_chair\_only = split\_df(testimony\_chair\_only\_raw)

tmp\_list = []

for i, row in speech\_clean.iterrows():

chairr = chair(row['date'])

if chairr.lower().split()[-1] in row['speaker'].lower():

row['speaker'] = chairr

tmp\_list.append(list(row))

col\_names = speech\_clean.columns

speech\_chair\_df = pd.DataFrame(data=tmp\_list, columns=col\_names)

speech\_sections = split\_df(speech\_chair\_df)

speech\_sections.reset\_index(drop=True, inplace=True)

speech\_chair\_df

tmp\_list = []

for i, row in testimony\_clean.iterrows():

chairr = chair(row['date'])

if chairr.lower().split()[-1] in row['speaker'].lower():

row['speaker'] = chairr

tmp\_list.append(list(row))

col\_names = testimony\_clean.columns

testimony\_chair\_only\_raw = pd.DataFrame(data=tmp\_list, columns=col\_names)

testimony\_chair\_only\_raw

data\_full = pd.concat([statement\_clean,

minutes\_clean,

speech\_chair\_df,

testimony\_chair\_only\_raw], sort=False)

data\_full.reset\_index(drop=True, inplace=True)

data\_sections = pd.concat([testimony\_sections,

minutes\_sections,

speech\_sections,

testimony\_sections\_chair\_only], sort=False)

data\_sections.reset\_index(drop=True, inplace=True)

#def save\_data(df, file\_name, dir\_name='C:/Users/theon/Desktop/proj2/data/preprocessed/'):

def save\_data(df, file\_name, dir\_name='/content/drive/My Drive/Colab Notebooks/proj2/data/'):

if not os.path.exists(dir\_name):

os.mkdir(dir\_name)

file = open(dir\_name + file\_name + '.pickle', 'wb')

pickle.dump(df, file)

file.close()

df.to\_csv(dir\_name + file\_name + '.csv', index=True)

save\_data(data\_full, 'data\_full')

save\_data(data\_sections, 'data\_sections')

# BERT

class InputFeature(object):

def \_\_init\_\_(self, id, input\_ids, masks, segments, meta, label=None):

self.id = id

self.features = {

'input\_ids': input\_ids,

'input\_mask': masks,

'segment\_ids': segments,

'meta': meta

}

self.label = label

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased', do\_lower\_case=True)

def bert\_encoder(text, max\_len=200):

tokens = tokenizer.tokenize(text)

tokens = tokens[:max\_len-2]

tokens = ["[CLS]"] + tokens + ["[SEP]"]

ids = tokenizer.convert\_tokens\_to\_ids(tokens)

ids += [0] \* (max\_len - len(tokens))

pad\_masks = [1] \* len(tokens) + [0] \* (max\_len - len(tokens))

segment\_ids = [0] \* len(tokens) + [0] \* (max\_len - len(tokens))

return ids, pad\_masks, segment\_ids

train\_set = []

max\_seq\_length = 200

meta\_size = 10

for index, row in tqdm(train\_data\_sections.iterrows(), total=train\_data\_sections.shape[0]):

input\_ids, masks, segments = bert\_encoder(row['text'], max\_seq\_length)

train\_set.append(InputFeature(row.index, input\_ids, masks, segments, row[nontext\_columns + ['tone']], int(row['target'])))

labels = train\_data\_sections['target'].astype(int).values

ids\_in = np.array([data.features['input\_ids'] for data in train\_set])

masks\_in = np.array([data.features['input\_mask'] for data in train\_set])

segids\_in =np.array([data.features['segment\_ids'] for data in train\_set])

metadata\_in =np.array([data.features['meta'] for data in train\_set], dtype=np.float64)

labels\_in = np.array([data.label for data in train\_set])

train\_dataset = np.zeros((len(train\_data\_sections), 3), dtype=np.float32)

print(metadata\_in[0])

print(metadata\_in[1])

class BertSeq(nn.Module):

def \_\_init\_\_(self, hsize, dsize, meta\_size, osize, dop=0.1):

"""

Initialize the model

"""

super().\_\_init\_\_()

self.osize = osize

self.dop = dop

self.bert = BertModel.from\_pretrained('bert-base-uncased',output\_hidden\_states=True,output\_attentions=True)

for param in self.bert.parameters():

param.requires\_grad = True

self.weights = nn.Parameter(torch.rand(13, 1))

self.dop = nn.dop(dop)

self.fc1 = nn.Linear(hsize, dsize)

self.fc2 = nn.Linear(dsize + meta\_size, osize)

self.softmax = nn.LogSoftmax(dim=1)

def forward(self, input\_ids, nn\_input\_meta):

hidden\_states, attt = self.bert(input\_ids)[-2:]

batch\_size = input\_ids.shape[0]

ht\_cls = torch.cat(hidden\_states)[:, :1, :].view(13, batch\_size, 1, 768)

att = torch.sum(ht\_cls \* self.weights.view(13, 1, 1, 1), dim=[1, 3])

att = F.softmax(att.view(-1), dim=0)

feature = torch.sum(ht\_cls \* att.view(13, 1, 1, 1), dim=[0, 2])

dense\_out = self.fc1(self.dop(feature))

concat\_layer = torch.cat((dense\_out, nn\_input\_meta.float()), 1)

out = self.fc2(concat\_layer)

return out

bert\_seq = BertSeq(768, 128, meta\_size, 3, dop=0.1)

learning\_rate = 1e-5

num\_runtime\_0s = 3

batch\_size = 32

patience =2

file\_name = 'model'

use\_skf = True

bert\_hsize = 768

bert\_dsize =128

def train\_bert(fold, tind, vind):

logger.info('layer{}'.format(fold))

tids\_in = torch.tensor(ids\_in[tind], dtype=torch.long)

tmask\_in = torch.tensor(masks\_in[tind], dtype=torch.long)

tseg\_in = torch.tensor(segids\_in[tind], dtype=torch.long)

tlabel\_in = torch.tensor(labels\_in[tind], dtype=torch.long)

tmeta\_in = torch.tensor(metadata\_in[tind], dtype=torch.long)

vids\_in = torch.tensor(ids\_in[vind], dtype=torch.long)

vmask\_in = torch.tensor(masks\_in[vind], dtype=torch.long)

vseg\_in = torch.tensor(segids\_in[vind], dtype=torch.long)

vlabel\_in = torch.tensor(labels\_in[vind], dtype=torch.long)

vmeta\_in = torch.tensor(metadata\_in[vind], dtype=torch.long)

train = torch.utils.data.TensorDataset(tids\_in, tmask\_in, tseg\_in, tmeta\_in, tlabel\_in)

valid = torch.utils.data.TensorDataset(vids\_in, vmask\_in, vseg\_in, vmeta\_in, vlabel\_in)

tload = torch.utils.data.DataLoader(train, batch\_size=batch\_size, shuffle=True)

vload = torch.utils.data.DataLoader(valid, batch\_size=batch\_size, shuffle=False)

bert\_seq = BertSeq(bert\_hsize, bert\_dsize, meta\_size, 3, dop=0.1)

device = 'cuda:0' if torch.cuda.is\_available() else 'cpu'

bert\_seq = bert\_seq.to(device)

loss\_fn = torch.nn.CrossEntropyLoss()

param\_opt = list(model.named\_parameters())

no\_decay = ['bias', 'LayerNorm.bias', 'LayerNorm.weight']

adam\_params = [{'params': [p for n, p in param\_opt if not any(nd in n for nd in no\_decay)], 'weight\_decay': 0.01},{'params': [p for n, p in param\_opt if any(nd in n for nd in no\_decay)], 'weight\_decay': 0.0}]

opt = AdamW(adam\_params, lr=learning\_rate, eps=1e-6)

bert\_seq.train()

best\_f1 = 0.

vchoose = np.zeros((vlabel\_in.size(0), 2))

segfault = 0

tlosss = []

vlosss = []

for runtime\_0 in range(num\_runtime\_0s):

logger.info('batch{}'.format(runtime\_0+1))

train\_loss = 0.

for i, batch in tqdm(enumerate(tload), total=len(tload), desc='Training'):

batch = tuple(t.to(device) for t in batch)

x\_ids, x\_mask, x\_sids, x\_meta, y\_truth = batch

y\_pred = bert\_seq(x\_ids, x\_meta)

loss = loss\_fn(y\_pred, y\_truth)

opt.zero\_grad()

loss.backward()

opt.step()

train\_loss += loss.item() / len(tload)

logger.debug('train batch: %d, train\_loss: %8f\n' % (i, train\_loss))

tlosss.append(train\_loss)

model.eval()

vloss = 0.

vpred = np.zeros((vlabel\_in.size(0), 3))

with torch.no\_grad():

for i, batch in tqdm(enumerate(vload), total=len(vload), desc='Validation'):

batch = tuple(t.to(device) for t in batch)

x\_ids, x\_mask, x\_sids, x\_meta, y\_truth = batch

y\_pred = bert\_seq(x\_ids, x\_meta).detach()

loss = loss\_fn(y\_pred, y\_truth)

vloss += loss.item() / len(vload)

vpred[i \* batch\_size:(i + 1) \* batch\_size] = F.softmax(y\_pred, dim=1).cpu().numpy()

logger.debug('validation batch: {}, vloss: {}, vpred: {}'.format(i, vloss, vpred[i \* batch\_size:(i + 1) \* batch\_size]))

vlosss.append(vloss)

acc, f1 = metric(labels\_in[vind], np.argmax(vpred, axis=1))

if best\_f1 < f1:

segfault = 0

best\_f1 = f1

vchoose = vpred

torch.save(bert\_seq.state\_dict(), output\_dir + 'model\_fold\_{}.dict'.format(fold))

else:

segfault += 1

logger.info('runtime\_0: %d, train loss: %.8f, valid loss: %.8f, acc: %.8f, f1: %.8f, best\_f1: %.8f\n' %

(runtime\_0, train\_loss, vloss, acc, f1, best\_f1))

if device == 'cuda:0':

torch.cuda.empty\_cache()

if segfault >= patience:

break

model.train()

vpred = np.zeros((vlabel\_in.size(0), 3))

sns.set(font\_scale=1.5)

plt.rcParams["figure.figsize"] = (15,6)

plt.plot(tlosss, 'b-o')

plt.plot(vlosss, 'b-o')

plt.title("Training/Validation Loss")

plt.xlabel("Runtime\_0")

plt.ylabel("Loss")

plt.show()

bert\_seq.load\_state\_dict(torch.load(output\_dir + 'model\_fold\_{}.dict'.format(fold)))

bert\_seq.eval()

with torch.no\_grad():

for i, batch in tqdm(enumerate(vload), total=len(vload)):

batch = tuple(t.to(device) for t in batch)

x\_ids, x\_mask, x\_sids, x\_meta, y\_truth = batch

y\_pred = bert\_seq(x\_ids, x\_meta).detach()

vpred[i \* batch\_size:(i + 1) \* batch\_size] = F.softmax(y\_pred, dim=1).cpu().numpy()

vchoose = vpred

train\_dataset[vind] = vchoose

acc, f1 = metric(labels\_in[vind], np.argmax(vchoose, axis=1))

logger.info('runtime\_0: best, acc: %.8f, f1: %.8f, best\_f1: %.8f\n' % (acc, f1, best\_f1))

if use\_skf:

skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

for fold, (tind, vind) in enumerate(skf.split(labels\_in, labels\_in)):

train\_bert(fold, tind, vind)

else:

tbal = 0.7

tind = np.arange(0, int(len(labels\_in)\*tbal))

vind = np.arange(int(len(labels\_in)\*tbal), len(labels\_in))

train\_bert(0, tind, vind)

def save\_data(df, file\_name, dir\_name=train\_dir):

if not os.path.exists(dir\_name):

os.mkdir(dir\_name)

file = open(dir\_name + file\_name + '.pickle', 'wb')

pickle.dump(df, file)

file.close()

df.to\_csv(dir\_name + file\_name + '.csv', index=True)

save\_data(train\_data, 'train\_data')

save\_data(txt\_data, 'txt\_data')

save\_data(train\_data, 'train\_data\_sections')