Twitter Named Entity Recognition Case Study

About

Twitter is a microblogging and social networking service on which users post and interact with messages known as "tweets". Every second, on average, around 6,000 tweets are tweeted on Twitter, corresponding to over 350,000 tweets sent per minute, 500 million tweets per day.

Problem statement

Twitter wants to automatically tag and analyze tweets for better understanding of the trends and topics without being dependent on the hashtags that the users use. Many users do not use hashtags or sometimes use wrong or mis-spelled tags, so they want to completely remove this problem and create a system of recognizing important content of the tweets.

Objective

Named Entity Recognition (NER) is an important subtask of information extraction that seeks to locate and recognise named entities. We need to train models that will be able to identify the various named entities.

Data

Dataset is annotated with 10 fine-grained NER categories: person, geo-location, company, facility, product,music artist, movie, sports team, tv show and other. Dataset was extracted from tweets and is structured in CoNLL format., in English language. Containing in Text file format. The CoNLL format is a text file with one word per line with sentences separated by an empty line. The first word in a line should be the word and the last word should be the label.

In [1]:
```
# imports
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf

from warnings import filterwarnings
filterwarnings('ignore')
```

```
WARNING:tensorflow:From c:\Users\psyki\Downloads\Learning\github-repos\twitte
r-ner-case-study\venv\lib\site-packages\keras\src\losses.py:2976: The name t
f.losses.sparse_softmax_cross_entropy is deprecated. Please use tf.compat.v1.
losses.sparse_softmax_cross_entropy instead.
```

In [2]:
```
# setting path variables
import os
root_path = os.path.abspath(os.path.join(os.getcwd(),os.pardir))
data_path = os.path.join(root_path,'data')
train_data_path = os.path.join(data_path,'wnut 16.txt.conll')
test_data_path = os.path.join(data_path,'wnut 16test.txt.conll')
```

Getting the data

In [3]:
```
# reading the training file
with open(train_data_path,'r') as f:
    train_raw = f.read()
with open(test_data_path,'r') as f:
    test_raw = f.read()
```

In [4]:
```
# creating a function to format the data
def extract_ner_from_conll(conll_data):
    # Split the data into sentences based on empty lines
    sentences = [sentence.strip() for sentence in conll_data.strip().split
    ner_data = []

    for sentence in sentences:
        tokenised_sentence = []
        for token_entity in sentence.split('\n'):
            token, entity = token_entity.split('\t')
            tokenised_sentence.append((token,entity))
        ner_data.append(tokenised_sentence)

    return ner_data
```

In [5]:
```
# preprocessing the raw files
train_data = extract_ner_from_conll(train_raw)
test_data = extract_ner_from_conll(test_raw)
```

In [6]:
```
# checking sentences after preprocessing
print(train_data[0])
```

```
[('@SammieLynnsMom', 'O'), ('@tg10781', 'O'), ('they', 'O'), ('will', 'O'),
('be', 'O'), ('all', 'O'), ('done', 'O'), ('by', 'O'), ('Sunday', 'O'), ('tru
st', 'O'), ('me', 'O'), ('*wink*', 'O')]
```

EDA

In [7]:
```
# number of words in the vocabulary and lenght of sentences in the trainin

sentence_lenghts = list()
word_set = set()
for sentence in train_data:
    sentence_lenghts.append(len(sentence))
    for word in sentence:
        word_set.add(word[0])

NUM_WORDS = len(word_set)+2 # +2 to include padding and out of vocabulary
print(f"Number of unique words in training data (including padding and OOV
print(f"Maximum sentence length = {max(sentence_lenghts)}")
print(f"Minimum sentence length = {min(sentence_lenghts)}")
```

```
Number of unique words in training data (including padding and OOV token) = 1
0588
Maximum sentence length = 39
Minimum sentence length = 1
```

In [8]:
```
# since the max sentence length if 39, we will take a length of 45 in our
SENTENCE_LENGTH = 45
# we will keep the embedding dimensions to be 50 since the number of datap
EMBEDDING_DIMS = 300
```

In [9]:
```
# number of entities
entity_set = set()
for sentence in train_data:
    for word in sentence:
        entity_set.add(word[1])

NUM_ENTITIES = len(entity_set)
print(f"Number of unique entities in training data = {NUM_ENTITIES}")
```

```
Number of unique entities in training data = 21
```

Data preparation

In [10]:
```
import re
import string
punctuations = string.punctuation
```

In [11]:
```
# create a function to prepare the data to be fed into the model

def clean_text(text):
    return re.sub("[^A-Za-z0-9]+",'',str(text).lower())

def prepare_data(text_data):

    # initialize empty lists for sentences and entities
    sentences = []
    entities = []

    for sentence in text_data:

        # initialize empty lists for sentence text and corresponding entit
        word_list = []
        entity_list = []

        for token in sentence:
            word = token[0]
            entity = token[1]
            if word in punctuations:
                continue
            else:
                word_list.append(clean_text(word))
                entity_list.append(entity)

        sentences.append(word_list)
        entities.append(entity_list)

    # create a single string for each sentence and entity by joining eleme
    sentences = np.array([' '.join(sentence) for sentence in sentences])
    entities = np.array([' '.join(entity) for entity in entities])

    return (sentences,entities)
```

In [12]:
```
# since the datapoints in train file is very low, we will merge the datase
data = []
for i in train_data:
    data.append(i)
for i in test_data:
    data.append(i)

xdata, ydata = prepare_data(data)

from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest = train_test_split(xdata, ydata, test_size=0.
```

In [13]:
```
# checking sentences are conversion
print(f"Before conversion\n{' '.join([word[0] for word in data[0]])}\n")
print(f"After conversion\n{xdata[0]}\n")
print(f"Entities\n{ydata[0]}")
```

```
Before conversion
@SammieLynnsMom @tg10781 they will be all done by Sunday trust me *wink*

After conversion
sammielynnsmom tg10781 they will be all done by sunday trust me wink

Entities
O O O O O O O O O O O O
```

Bidirectional LSTM + CRF

In [14]:
```
from tensorflow.keras.layers import Input, TextVectorization, Embedding, B
from tensorflow.keras.models import Model
from tensorflow_addons.layers import CRF
from tensorflow_addons.losses import SigmoidFocalCrossEntropy
```

In [15]:
```
# adapt the vectorizer layer before modeling
sentence_tokenizer = TextVectorization(max_tokens=NUM_WORDS, output_sequen
sentence_tokenizer.adapt(xtrain)

# indexing the tokens in train and test
train_lstm_sentence_indexed = sentence_tokenizer(xtrain)
test_lstm_sentence_indexed = sentence_tokenizer(xtest)
```

```
WARNING:tensorflow:From c:\Users\psyki\Downloads\Learning\github-repos\twitte
r-ner-case-study\venv\lib\site-packages\keras\src\backend.py:873: The name t
f.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph
instead.

WARNING:tensorflow:From c:\Users\psyki\Downloads\Learning\github-repos\twitte
r-ner-case-study\venv\lib\site-packages\keras\src\utils\tf_utils.py:492: The
name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragge
d.RaggedTensorValue instead.
```

In [16]:
```
# initializing the embedding vector
import gensim.downloader as api
word2vec = api.load('word2vec-google-news-300')
embedding_matrix = np.zeros(shape=(NUM_WORDS,EMBEDDING_DIMS), dtype=np.flo

for i,word in enumerate(sentence_tokenizer.get_vocabulary()):
    try:
        embedding_matrix[i] = word2vec[word]
    except:
        pass
```

In [17]:
```
del word2vec
```

In [18]:
```
# define a function to build and compile model
def build_lstm_model(name='bi-lstm+crf'):

    # input layer for getting sentences
    sentence_input = Input(shape=(SENTENCE_LENGTH,), dtype=tf.float32, nam

    # creating embeddings for each token in sentence
    embeddings = Embedding(
        input_dim = NUM_WORDS,
        output_dim = EMBEDDING_DIMS,
        mask_zero = True,
        name = 'word_embedding',
        embeddings_initializer = tf.keras.initializers.constant(embedding_
    )(sentence_input)

    # stacking two bidirectional LSTMs
    output_sequence = Bidirectional(LSTM(50, return_sequences=True), name=
    output_sequence = Bidirectional(LSTM(50, return_sequences=True), name=

    # passing the sequence through dense layer to compress the information
    dense_sequence = TimeDistributed(Dense(25, activation='relu'), name='d

    # passing the dense sequences through crf layer
    predicted_sequence, potentials, sequence_length, crf_kernel = CRF(NUM_

    # define the train model
    training_model = Model(sentence_input, potentials)

    # compile the model
    training_model.compile(
        loss=SigmoidFocalCrossEntropy(),
        optimizer='adam'
    )

    # create an inference model
    inference_model = Model(sentence_input, predicted_sequence)

    return training_model, inference_model
```

In [19]:
```
# creating the traing and inferencing model
lstm_training_model, lstm_inference_model = build_lstm_model()
```

```
WARNING:tensorflow:From c:\Users\psyki\Downloads\Learning\github-repos\twitte
r-ner-case-study\venv\lib\site-packages\keras\src\optimizers\__init__.py:309:
The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Opti
mizer instead.
```

In [20]:
```
lstm_training_model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sentence_input (InputLayer  [(None, 45)]              0
 )

 word_embedding (Embedding)  (None, 45, 300)           3176400

 lstm_1 (Bidirectional)      (None, 45, 100)           140400

 lstm_2 (Bidirectional)      (None, 45, 100)           60400

 dense (TimeDistributed)     (None, 45, 25)            2525

 crf (CRF)                   [(None, 45),              1029
                              (None, 45, 21),
                              (None,),
                              (21, 21)]

=================================================================
Total params: 3380754 (12.90 MB)
Trainable params: 3380754 (12.90 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

In [21]:
```
# prepare the entity data

# tokenizing the entities
entity_tokenizer = TextVectorization(max_tokens=NUM_ENTITIES, output_seque
entity_tokenizer.adapt(ytrain)
ytrain_tokenized = entity_tokenizer(ytrain)
ytest_tokenized = entity_tokenizer(ytest)

# one hot encoding the entitiy tokens
entity_ohe = Lambda(lambda x: tf.one_hot(x, NUM_ENTITIES))
ytrain_tokenized = entity_ohe(ytrain_tokenized)
ytest_tokenized = entity_ohe(ytest_tokenized)
```

In [22]:

```
# defining callbacks
mc = tf.keras.callbacks.ModelCheckpoint(
    os.path.join(root_path,'models','lstm.ckpt'),
    monitor="val_loss",
    save_best_only=True,
    save_weights_only=True
)
es = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=5,
    start_from_epoch=5,
)
callbacks=[mc,es]
```

In [23]:

```
# fitting the lstm+crf model
history_lstm = lstm_training_model.fit(
    train_lstm_sentence_indexed, ytrain_tokenized,
    validation_data=(test_lstm_sentence_indexed, ytest_tokenized),
    epochs=50,
    batch_size=16,
    callbacks=callbacks
)
```

```
Epoch 1/50
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:0'] wh
en minimizing the loss. If you're using `model.compile()`, did you forget to
provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:0'] wh
en minimizing the loss. If you're using `model.compile()`, did you forget to
provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:0'] wh
en minimizing the loss. If you're using `model.compile()`, did you forget to
provide a `loss` argument?
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:0'] wh
en minimizing the loss. If you're using `model.compile()`, did you forget to
provide a `loss` argument?
313/313 [==============================] - 36s 73ms/step - loss: 0.3820 - val
_loss: 0.1575
Epoch 2/50
313/313 [==============================] - 18s 57ms/step - loss: 0.0876 - val
_loss: 0.0535
Epoch 3/50
313/313 [==============================] - 17s 54ms/step - loss: 0.0367 - val
_loss: 0.0436
Epoch 4/50
313/313 [==============================] - 17s 55ms/step - loss: 0.0269 - val
_loss: 0.0449
Epoch 5/50
313/313 [==============================] - 19s 59ms/step - loss: 0.0235 - val
_loss: 0.0645
Epoch 6/50
313/313 [==============================] - 17s 55ms/step - loss: 0.0216 - val
_loss: 0.0738
Epoch 7/50
313/313 [==============================] - 17s 56ms/step - loss: 0.0206 - val
_loss: 0.1044
Epoch 8/50
313/313 [==============================] - 17s 55ms/step - loss: 0.0188 - val
_loss: 0.1294
Epoch 9/50
313/313 [==============================] - 18s 57ms/step - loss: 0.0212 - val
_loss: 0.1290
Epoch 10/50
313/313 [==============================] - 18s 56ms/step - loss: 0.0188 - val
_loss: 0.1456
Epoch 11/50
313/313 [==============================] - 18s 56ms/step - loss: 0.0180 - val
_loss: 0.1766
```

In [24]:
```
# loading the best model
lstm_training_model.load_weights(os.path.join(root_path,'models','lstm.ckp
```

Out[24]:
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x26c55d15e7
0>
```

In [25]:
```
# evaluating on validation data
lstm_training_model.evaluate(test_lstm_sentence_indexed,ytest_tokenized)
```

```
40/40 [==============================] - 1s 12ms/step - loss: 0.0436
```

Out[25]:
```
0.04360351338982582
```

Inferencing the Bi-LSTM+CRF model

In [32]:
```
# function to infer prediction from a text in xtest randomly
def infer_lstm():
    idx = np.random.choice(range(len(xtest)),1,replace=False)
    text = xtest[idx]
    text_tokenized = sentence_tokenizer(text)
    pred_labels = lstm_inference_model.predict(text_tokenized)
    act_labels = ytest[idx]
    text_len = min(len(text[0].split()),SENTENCE_LENGTH)

    pred_labels = np.asarray(entity_tokenizer.get_vocabulary())[list(pred_
    result = pd.DataFrame(
        {
            'text':text[0].split(),
            'actual_labels':act_labels[0].split(),
            'predicted_labels':pred_labels[:text_len]
        }
    )
    display(result)
```

In [64]:
```
# infer random text from the validation data
infer_lstm()
```

```
1/1 [==============================] - 0s 26ms/step
```

	text	actual_labels	predicted_labels
0	rt	O	o
1	miriamsaying	O	o
2	may	O	o
3	mga	O	o
4	patama	O	o
5	talagang	O	o
6	di	O	o
7	naman	O	o
8	para	O	o
9	sayo	O	o
10	sadyang	O	o
11	assumera	O	o
12	ka	O	o
13	lang	O	o
14	kaya	O	o
15	inaangkin	O	o
16	mo	O	o

BERT

In [65]:
```
# imports
from transformers import TFBertForTokenClassification
from transformers import BertTokenizer
```

In [66]:
```
# initializing the bert tokenizer (Sub word tokenizer)
bert_tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_low
```

In [67]:
```
# preprocessing data for bert

def preprocess_bert(text,labels):

    text_list = []
    label_list = []

    for i in range(len(text)):
        tokenized_words = []
        tokenized_labels = []
        words = text[i].split()
        entities = labels[i].split()
        sentence_len = len(words)
        for j in range(sentence_len):
            tokenized_word = bert_tokenizer.tokenize(words[j])
            tokenized_words.extend(tokenized_word)
            tokenized_label = [entities[j]]*len(tokenized_word)
            tokenized_labels.extend(tokenized_label)
        text_list.append(' '.join(tokenized_words))
        label_list.append(' '.join(tokenized_labels))
    return text_list,label_list
```

In [68]:
```
# applying preprocessing to the sentences
train_bert_tokenized_sentences, train_bert_tokenized_labels = preprocess_b
test_bert_tokenized_sentences, test_bert_tokenized_labels = preprocess_ber
```

In [69]:
```
# checking the sentences after tokenization
print(train_bert_tokenized_sentences[0])
print(train_bert_tokenized_labels[0])
```

```
new tee ##shi ##rts ordered with all new designs i hope people will like octo
ber 9th 2010 remington s annual october ##fest
O O O O O O O O O O O O O O O O O B-other I-other I-other
```

In [70]:
```
# adapt the vectorizer layer before modeling
sentence_indexer_bert = TextVectorization(max_tokens=NUM_WORDS, output_seq
sentence_indexer_bert.adapt(train_bert_tokenized_sentences)
train_bert_sentence_index = sentence_indexer_bert(train_bert_tokenized_sen
test_bert_sentence_index = sentence_indexer_bert(test_bert_tokenized_sente
```

In [71]:
```
# initializing the token type ids for train and test sentences
train_token_type_ids = np.zeros(shape=(len(train_bert_tokenized_sentences)
test_token_type_ids = np.zeros(shape=(len(test_bert_tokenized_sentences),S
```

In [72]:
```
# initialiazing the attention masks for train and test sentences
train_attn_mask = np.zeros(shape=(len(train_bert_tokenized_sentences),SENT
test_attn_mask = np.zeros(shape=(len(test_bert_tokenized_sentences),SENTEN
for i in range(len(train_bert_tokenized_sentences)):
    length = min(len(train_bert_tokenized_sentences[i].split()),SENTENCE_L
    train_attn_mask[i,:length] = 1
for i in range(len(test_bert_tokenized_sentences)):
    length = min(len(test_bert_tokenized_sentences[i].split()),SENTENCE_LE
    test_attn_mask[i,:length] = 1
```

In [73]:
```
# prepare the entity data

# tokenizing the entities
entity_indexer_bert = TextVectorization(max_tokens=NUM_ENTITIES, output_se
entity_indexer_bert.adapt(ytrain)
train_bert_label_index = entity_indexer_bert(train_bert_tokenized_labels)
test_bert_label_index = entity_indexer_bert(test_bert_tokenized_labels)
```

In [74]:
```
# imports for modeling
from keras.losses import SparseCategoricalCrossentropy
from keras.optimizers import Adam
```

In [87]:

```
# build bert model
encoder = TFBertForTokenClassification.from_pretrained('bert-base-uncased'
def build_bert_model():

    # getting inputs to the model
    input_sentence_ids = Input(shape=(SENTENCE_LENGTH,), dtype=tf.int32)
    input_token_type_ids = Input(shape=(SENTENCE_LENGTH,), dtype=tf.int32)
    input_attn_ids = Input(shape=(SENTENCE_LENGTH,), dtype=tf.int32)

    # sending inputs through the bert model
    embeddings = encoder(
        input_ids = input_sentence_ids,
        token_type_ids = input_token_type_ids,
        attention_mask = input_attn_ids,
    )[0]

    # sending the context vectors through dense layer with linear activati
    output_logits = Dense(NUM_ENTITIES,activation='linear')(embeddings)

    # defining the model
    model = Model(
        inputs = [input_sentence_ids, input_token_type_ids, input_attn_ids
        outputs = [output_logits]
    )

    # compiling the model
    model.compile(
        loss = SparseCategoricalCrossentropy(from_logits=True),
        optimizer = Adam(),
        run_eagerly=True
    )

    # returning the built model
    return model
```

```
All PyTorch model weights were used when initializing TFBertForTokenClassific
ation.

Some weights or buffers of the TF 2.0 model TFBertForTokenClassification were
not initialized from the PyTorch model and are newly initialized: ['classifie
r.weight', 'classifier.bias']
You should probably TRAIN this model on a down-stream task to be able to use
it for predictions and inference.
```

In [88]:
```
# calling the build model function and checking it's structure
bert_model = build_bert_model()
bert_model.summary()
```

```
Model: "model_3"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected
to
==================================================================================================
 input_4 (InputLayer)        [(None, 45)]                 0         []

 input_6 (InputLayer)        [(None, 45)]                 0         []

 input_5 (InputLayer)        [(None, 45)]                 0         []

 bert_layer (TFBertForToken  TFTokenClassifierOutput(lo   1088931   ['input_4
[0][0]',
 Classification)             ss=None, logits=(None, 45,   86         'input_6
[0][0]',
                              2),                                   'input_5
[0][0]']
                              hidden_states=None, atten
                             tions=None)

 dense_3 (Dense)             (None, 45, 21)               63        ['bert_la
yer[0][0]']

==================================================================================================
Total params: 108893249 (415.39 MB)
Trainable params: 108893249 (415.39 MB)
Non-trainable params: 0 (0.00 Byte)
__________________________________________________________________________________________________
```

In [89]:
```
# defining callbacks
mc = tf.keras.callbacks.ModelCheckpoint(
    os.path.join(root_path,'models','bert.ckpt'),
    monitor="val_loss",
    save_best_only=True,
)
es = tf.keras.callbacks.EarlyStopping(
    monitor="val_loss",
    min_delta=0,
    patience=5,
    start_from_epoch=5,
)
callbacks=[mc,es]
```

In [92]:
```
# fitting the bert model
history_bert = bert_model.fit(
    [train_bert_sentence_index, train_token_type_ids, train_attn_mask], tr
    validation_data = ([test_bert_sentence_index, test_token_type_ids, tes
    epochs=3,
    batch_size=16,
    callbacks=callbacks
)
```

```
Epoch 1/3
313/313 [==============================] - ETA: 0s - loss: 0.9673INFO:tensorf
low:Assets written to: c:\Users\psyki\Downloads\Learning\github-repos\twitter
-ner-case-study\models\bert.ckpt\assets

INFO:tensorflow:Assets written to: c:\Users\psyki\Downloads\Learning\github-r
epos\twitter-ner-case-study\models\bert.ckpt\assets

313/313 [==============================] - 1137s 4s/step - loss: 0.9673 - val
_loss: 0.9566
Epoch 2/3
313/313 [==============================] - ETA: 0s - loss: 0.9585INFO:tensorf
low:Assets written to: c:\Users\psyki\Downloads\Learning\github-repos\twitter
-ner-case-study\models\bert.ckpt\assets

INFO:tensorflow:Assets written to: c:\Users\psyki\Downloads\Learning\github-r
epos\twitter-ner-case-study\models\bert.ckpt\assets

313/313 [==============================] - 1240s 4s/step - loss: 0.9585 - val
_loss: 0.9535
Epoch 3/3
313/313 [==============================] - 1303s 4s/step - loss: 0.9572 - val
_loss: 0.9539
```

In [93]:
```
# loading the best model
bert_model.load_weights(os.path.join(root_path,'models','bert.ckpt'))
```

Out[93]:
```
<tensorflow.python.checkpoint.checkpoint.CheckpointLoadStatus at 0x26cbe72a65
0>
```

BERT Inference

In [94]:
```
# getting the predictions
predictions = bert_model.predict([test_bert_sentence_index,test_token_type
```

```
40/40 [==============================] - 56s 1s/step
```

In [95]:
```python
# defining a function to join back the tokenized text
def get_joined_labels(tokenized_text,tokenized_labels):
    joined_labels = []
    begin = 0
    end = 0

    for i in range(len(tokenized_text.split())):

        if i==len(tokenized_text.split())-1:
            if begin==end:
                joined_labels.append(tokenized_labels.split()[i])
            else:
                label_segment = tokenized_labels.split()[begin:end+1]
                added = None
                for j in label_segment:
                    if j!='o':
                        added = j
                        break
                if added == None:
                    added = 'o'
                joined_labels.append(added)

        elif str(tokenized_text.split()[i+1]).startswith("##"):
            end = i+1

        else:
            if begin==end:
                begin = i+1
                end = i+1
                joined_labels.append(tokenized_labels.split()[i])
            else:
                label_segment = tokenized_labels.split()[begin:end+1]
                added = None
                for j in label_segment:
                    if j!='o':
                        added = j
                        break
                if added == None:
                    added = 'o'
                joined_labels.append(added)
                begin = i+1
                end = i+1
    return ' '.join(joined_labels)
```

In [98]:
```python
# joining back the tokenized text
pred_labels = np.argmax(predictions, axis=-1)
bert_joined_labels = []
for i in range(len(xtest)):
    text = test_bert_tokenized_sentences[i]
    lbls = ' '.join(np.asarray(entity_indexer_bert.get_vocabulary())[list(
    bert_joined_labels.append(get_joined_labels(text,lbls))
```

In [ ]:
```
# function to infer prediction from a text in xtest randomly
def infer_bert():
    idx = np.random.choice(range(len(test_bert_sentence_index)),1,replace=

    text = xtest[idx]
    labels_act = ytest[idx]
    pred_labels = bert_joined_labels[idx]
    result = pd.DataFrame(
        {
            'text':text.split(),
            'actual_labels':labels_act.split(),
            'predicted_labels':pred_labels.split()
        }
    )
    display(result)
```

In [ ]:
```
# infer random text from the validation data
infer_bert()
```

```
1/1 [==============================] - 0s 26ms/step
```

	text	actual_labels	predicted_labels
0	rt	O	o
1	miriamsaying	O	o
2	may	O	o
3	mga	O	o
4	patama	O	o
5	talagang	O	o
6	di	O	o
7	naman	O	o
8	para	O	o
9	sayo	O	o
10	sadyang	O	o
11	assumera	O	o
12	ka	O	o
13	lang	O	o
14	kaya	O	o
15	inaangkin	O	o
16	mo	O	o

Conclusion

- The BI-LSTM+CRF model tends to overfit to the data
- The prediction from the CRF layer is not accurate

- Since BERT is larger model with model weights and more layers, it takes a lot of time to train
- The loss in BERT is larger compared to that in LSTM
- BERT tends to perform better than the LSRM model even though the loss is high
- Early stoping and Model checkpoint callbacks assists in training the model

Questions

Q. Defining the problem statements and where can this and modifications of this be used?
Ans. Similar model can be used in Part-of-Speech tagging (POS tagging) or any other problem where we need a prediction at a token level. Meaning that the input length and the output length is the same.

Q. Explain the data format (conll bio format)
Ans. Conll format is a text storing format where each word in the text is separated by a line (\n), and annotation of that word is separated by tab (\t) and each text is separated by two lines (\n\n).

Q. What other ner data annotation formats are available and how are they different
Ans. Other formats can be BIO, IOB, JSON and XML

Q. Why do we need tokenization of the data in our case
Ans. We need tokenization in our case to divide the sentence into smaller substituents in order to capture the sequence, context and attention mechanism. If we do not tokenize our text, the input to the model will be one big string. Another problem will be to convert the string to vectors.

Q. What other models can you use for this task
Ans. We can replace LSTM with RNN or GRU and couple them with CRF layer. Or in case of BERT, we can use transformers (encoder-decoder architecture) or GPT architecture.

Q. Did early stopping have any effect on the training and results.
Ans. Yes. Since the model started to overfit as the epochs progresses, early stopping caused the training to stop since the performance was not improving.

Q. How does the BERT model expect a pair of sentences to be processed?
Ans. The BERT model expects a token_type_id tensor, which represents which sentence does each of the token belongs to. The pair of sentences are merged into a single vector and token_type_id is used to distinguish between them.

Q. Why choose Attention based models over Recurrent based ones?
Ans. Recurrent based models have problems capturing long term dependencies of the words. Also, each word can be connected to multiple other words which can be capture by the attention heads. Hence, attention based models are preferred if long term dependencies or connection of words is required.

Q. Differentiate BERT and simple transformers
Ans. BERT can be considered as the encoder only part of the transformers. In transformer architecture, first the input is passed through the encoder, then through the decoder and then the final prediction is made. But in BERT, the encoder only architecture is responsible for prediction.

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```