Problem statement

Context: 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. 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: You need to train a model that will be able to identify the various named entities.

Downloading data

```
!gdown 14_VHffl1qBUEnZ1IWFHnh6B9M5_A-Wf8
!gdown 1cnrGjppPOU_NtHNpGu0RJGg1CUNNsse_
```

```
Downloading...
From: https://drive.google.com/uc?id=14_VHffl1qBUEnZ1IWFHnh6B9M5_A-Wf8
To: /content/wnut 16.txt.conll
100% 403k/403k [00:00<00:00, 118MB/s]
Downloading...
From: https://drive.google.com/uc?id=1cnrGjppPOU_NtHNpGu0RJGg1CUNNsse_
To: /content/wnut 16test.txt.conll
100% 635k/635k [00:00<00:00, 132MB/s]
```

Installing libraries

```
%pip install datasets transformers
%pip install tensorflow-addons
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: datasets in /usr/local/lib/python3.7/dist-packages (2.7.0)
Requirement already satisfied: transformers in /usr/local/lib/python3.7/dist-packages (4.24.0)
Requirement already satisfied: requests>=2.19.0 in /usr/local/lib/python3.7/dist-packages (from datasets) (2.23.0)
Requirement already satisfied: huggingface-hub<1.0.0,>=0.2.0 in /usr/local/lib/python3.7/dist-packages (from datasets) (0.11.0)
Requirement already satisfied: multiprocess in /usr/local/lib/python3.7/dist-packages (from datasets) (0.70.14)
Requirement already satisfied: xxhash in /usr/local/lib/python3.7/dist-packages (from datasets) (3.1.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from datasets) (21.3)
Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from datasets) (4.13.0)
Requirement already satisfied: fsspec[http]>=2021.11.1 in /usr/local/lib/python3.7/dist-packages (from datasets) (2022.10.0)
Requirement already satisfied: pyarrow>=6.0.0 in /usr/local/lib/python3.7/dist-packages (from datasets) (6.0.1)
Requirement already satisfied: tqdm>=4.62.1 in /usr/local/lib/python3.7/dist-packages (from datasets) (4.64.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from datasets) (1.21.6)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.7/dist-packages (from datasets) (3.8.3)
Requirement already satisfied: responses<0.19 in /usr/local/lib/python3.7/dist-packages (from datasets) (0.18.0)
Requirement already satisfied: dill<0.3.7 in /usr/local/lib/python3.7/dist-packages (from datasets) (0.3.6)
Requirement already satisfied: pandas in /usr/local/lib/python3.7/dist-packages (from datasets) (1.3.5)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.7/dist-packages (from datasets) (6.0)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (1.8.1)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (1.3.3)
Requirement already satisfied: async-timeout<5.0,>=4.0.0a3 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (4.0.
Requirement already satisfied: typing-extensions>=3.7.4 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (4.1.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (22.1.0)
Requirement already satisfied: asynctest==0.13.0 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (0.13.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: charset-normalizer<3.0,>=2.0 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (2.1
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.7/dist-packages (from aiohttp->datasets) (6.0.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from huggingface-hub<1.0.0,>=0.2.0->datasets) (3
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->datasets) (3.0.9
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests>=2.19.0->datasets) (3.0.4
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests>=2.19.0->datasets) (2.10)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in /usr/local/lib/python3.7/dist-packages (from requests>=2.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests>=2.19.0->datasets) (2022
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (2022.6.2)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.7/dist-packages (from transformers) (0.1
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->datasets) (3.10.0)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas->datasets) (2022.6)
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.7.3->pandas->datasets) (
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: tensorflow-addons in /usr/local/lib/python3.7/dist-packages (0.18.0)
Requirement already satisfied: typeguard>=2.7 in /usr/local/lib/python3.7/dist-packages (from tensorflow-addons) (2.7.1)
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from tensorflow-addons) (21.3)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging->tensorflow-addon
```

```
import pandas as pd
import tensorflow as tf
```

Loading data from the files

```
def load_data(filename: str):
  # Conll file is stored as (token, tag) pairs, one per line
  # Extracting data from conll files
  with open(filename, 'r') as file:
    lines = [line[:-1].split() for line in file] # Skipping last line as it will be a blank space
  samples, start = [], 0
  for end, parts in enumerate(lines):
      if not parts:
          sample = [(token, tag)
                      for token, tag in lines[start:end]]
          samples.append(sample)
          start = end + 1
  if start < end:
    samples.append(lines[start:end])
  return samples

train_samples = load_data('wnut 16.txt.conll')
test_samples = load_data('wnut 16test.txt.conll')
samples = train_samples + test_samples
schema = ['_'] + sorted({tag for sentence in samples
                             for _, tag in sentence}) # '_' is used to indicate a null (blank) token.
```

Structure of data

```
train_samples[1]
```

```
[('Made', 'O'),
 ('it', 'O'),
 ('back', 'O'),
 ('home', 'O'),
 ('to', 'O'),
 ('GA', 'B-geo-loc'),
 ('.', 'O'),
 ('It', 'O'),
 ('sucks', 'O'),
 ('not', 'O'),
 ('to', 'O'),
 ('be', 'O'),
 ('at', 'O'),
 ('Disney', 'B-facility'),
 ('world', 'I-facility'),
 (',', 'O'),
 ('but', 'O'),
 ('its', 'O'),
 ('good', 'O'),
 ('to', 'O'),
 ('be', 'O'),
 ('home', 'O'),
 ('.', 'O'),
 ('Time', 'O'),
 ('to', 'O'),
 ('start', 'O'),
 ('planning', 'O'),
 ('the', 'O'),
 ('next', 'O'),
 ('Disney', 'B-facility'),
 ('World', 'I-facility'),
 ('trip', 'O'),
 ('.', 'O')]
```

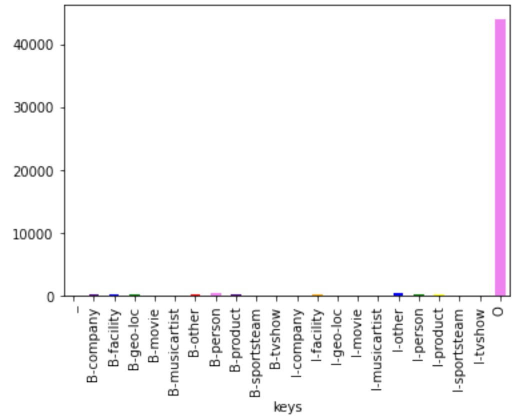
EDA: Let's have a look at the distribution of tags on data

```
import seaborn as sns
colors = ['violet', 'indigo', 'blue', 'green', 'yellow', 'orange', 'red']
counts = {}

# Calculateing the number of data points having a given label
for tag in schema:
  counts[tag] = 0
  for sample in train_samples:
    for label in sample:
      if label[1] == tag:
        counts[tag]+=1

counts_df = pd.DataFrame({'keys': list(counts.keys()), 'values': list(counts.values())})
counts_df.plot.bar(x='keys', y='values', legend=False, color=colors)
```

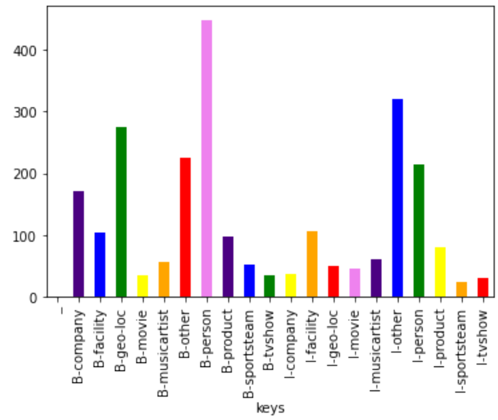
<matplotlib.axes._subplots.AxesSubplot at 0x7f18c55d1750>



- We have too many "other" fields, which is natural as only few annotations exist per sentence
- let's remove `O` tag and see tag distribution

```
counts.pop('O')
counts_df = pd.DataFrame({'keys': list(counts.keys()), 'values': list(counts.values())})
counts_df.plot.bar(x='keys', y='values', legend=False, color=colors)
```

<matplotlib.axes._subplots.AxesSubplot at 0x7f18c5622690>



Tag information

- B-* Start token for a tag
- I-* Continuation tokens for a tag

Available Entities

- Company
- Facility
- Geo-loc: geolocation
- Musicartist
- Person
- Product
- Sportsteam
- TV Show

- Other

More preprocessing

- let's get vocab & sequence lengths

```
from collections import defaultdict
all_samples = train_samples
all_samples.extend(test_samples)

word_counts = defaultdict(int) # Calculate vocab size
max_len = 0 # Calculate max length of a sentence

for sample in all_samples:
  for word in sample:
    word_counts[word[0]]+=1

  max_len = max(max_len, len(sample))

n_words = len(word_counts.items())


print("*"*30)
print("Max Length: ", max_len)
print("Vocab Size: ", n_words)
```

```
******************************
Max Length:  39
Vocab Size:  25382
```

Our approach

- Train a simple LSTM + CRF model to get a baseline
- Look at the results of transformer based architectures

Training LSTM + CRF model:

- Let's using glove to initialize embeddings

```
import gensim.downloader as api
word2vec = api.load("glove-twitter-200") # Loading word2vec gensim model
embedding_dim = 200
```

Training a tokenizer for LSTM input embeddings

```
all_sentences = [] # Concating test, train sentences. To train a tokenizer
for sample in all_samples:
  sentence = [tag[0] for tag in sample]
  all_sentences.append(sentence)

crf_tokenizer = tf.keras.preprocessing.text.Tokenizer(num_words=n_words, lower=True)
crf_tokenizer.fit_on_texts(all_sentences)
```

Prepare embedding matrix

```
import numpy as np
num_tokens = len(crf_tokenizer.word_index) + 1
hits = 0
misses = 0
missed_words = []


# Prepare embedding matrix
embedding_matrix = np.zeros((num_tokens, embedding_dim))
for word, i in crf_tokenizer.word_index.items():
  embedding_vector = None
  try:
    embedding_vector = word2vec[word]
  except Exception :
    pass

  if embedding_vector is not None:
    # Words not found in embedding index will be all-zeros.
    # This includes the representation for "padding" and "OOV"
    embedding_matrix[i] = embedding_vector
    hits += 1
  else:
    missed_words.append(word)
    misses += 1
print("Converted %d words (%d misses)" % (hits, misses))
```

```
Converted 11495 words (10438 misses)
```

LSTM + CRF Model training

Creating a training dataset

```
tag2id = {} # Label to indicies mapping
id2tag = {} # Index to label mapping
for i, tag in enumerate(schema):
  tag2id[tag] = i
  id2tag[i] = tag
```

- We will encode our labels as OHE vectors. This is to keep it compatible with SigmoidFocalCrossEntropy loss

```
def get_dataset(samples, max_len, tag2id, tokenizer):
  '''Prepares the input dataset

  Args:
    `samples`: List[List[Tuple[word, tag]]], input data
    `max_len`: Maximum input length
    `tag2id`: Mapping[tag: integer]
    `tokenizer`: Tensorflow tokenizer, for tokenizing input sequence

  Returns:
    Tuple[np.ndarray, np.ndarray]: sentences and it's labels
  '''
  dataset = {'samples':[], 'labels': []}

  for sample in samples:
    # Extracting inputs and labels
    inputs = [x[0] for x in sample]
    outputs = [x[1] for x in sample]

    # Tokenizing inputs
    inputs = tokenizer.texts_to_sequences([inputs])[0]

    # padding labels
    padded_inputs = [inputs[i] if i < len(inputs) else 0 for i in range(max_len)]

    # Initializing labels as One Hot Encoded Vectors
    padded_labels = [[0 for i in range(len(tag2id))] for j in range(max_len)]
    for i in range(len(outputs)):
      padded_labels[i][tag2id[outputs[i]]] = 1

    # Adding padded inputs & labels to dataset
    dataset['samples'].append(padded_inputs)
    dataset['labels'].append(padded_labels)

  return np.array(dataset['samples']), np.array(dataset['labels'])

train_sentences, train_labels = get_dataset(train_samples, max_len, tag2id, crf_tokenizer)
test_sentences, test_labels = get_dataset(test_samples, max_len, tag2id, crf_tokenizer)
```

Training Model

- using sigmoid focal cross entropy loss. It performs better than sparse categorical cross entropy for highly imbalanced data.

```
from keras.models import Model
from tensorflow.keras.layers import Input
from tensorflow_addons.utils.types import FloatTensorLike, TensorLike

# LSTM components
from keras.layers import LSTM, Embedding, Dense, TimeDistributed, Dropout, Bidirectional

# CRF layer
from tensorflow_addons.layers import CRF

# Sigmoid focal cross entropy loss. works well with highly unbalanced input data
from tensorflow_addons.losses import SigmoidFocalCrossEntropy
from tensorflow_addons.optimizers import AdamW


def build_model():
  # Model definition
  input = Input(shape=(max_len,))

  # Get embeddings
  embeddings = Embedding(input_dim=embedding_matrix.shape[0],
                      output_dim=embedding_dim,
                      input_length=max_len, mask_zero=True,
                      embeddings_initializer=tf.keras.initializers.Constant(embedding_matrix)
                    )(input)

  # variational biLSTM
  output_sequences = Bidirectional(LSTM(units=50, return_sequences=True))(embeddings)

  # Stacking
  output_sequences = Bidirectional(LSTM(units=50, return_sequences=True))(output_sequences)

  # Adding more non-linearity
  dense_out = TimeDistributed(Dense(25, activation="relu"))(output_sequences)

  # CRF layer
  crf = CRF(len(schema), name='crf')
  predicted_sequence, potentials, sequence_length, crf_kernel = crf(dense_out)

  model = Model(input, potentials)
  model.compile(
      optimizer=AdamW(weight_decay=0.001),
      loss= SigmoidFocalCrossEntropy()) # Sigmoid focal cross entropy loss

  return model

model = build_model()

# Checkpointing
save_model = tf.keras.callbacks.ModelCheckpoint(filepath='twitter_ner_crf.h5',
  monitor='val_loss',
  save_weights_only=True,
  save_best_only=True,
  verbose=1
)

# Early stopping
es = tf.keras.callbacks.EarlyStopping(monitor='val_loss', verbose=1, patience=10)

callbacks = [save_model, es]

model.summary()
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 39)]              0

 embedding (Embedding)       (None, 39, 200)           4386800

 bidirectional (Bidirectiona  (None, 39, 100)          100400
 l)

 bidirectional_1 (Bidirectio  (None, 39, 100)          60400
 nal)

 time_distributed (TimeDistr  (None, 39, 25)           2525
 ibuted)

 crf (CRF)                   [(None, 39),              1100
                              (None, 39, 22),
                              (None,),
                              (22, 22)]
```

```
=================================================================
Total params: 4,551,225
Trainable params: 4,551,225
Non-trainable params: 0
_________________________________________________________________
```

Training our model

```
model.fit(train_sentences, train_labels,
          validation_data = (test_sentences, test_labels),
          epochs = 300,
          callbacks  = callbacks,
          shuffle=True)
```

```
Epoch 1/300
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:0'] when minimizing the loss. If you're using `model.compi
WARNING:tensorflow:Gradients do not exist for variables ['chain_kernel:0'] when minimizing the loss. If you're using `model.compi
195/196 [============================>.] - ETA: 0s - loss: 0.0846
Epoch 1: val_loss improved from inf to 0.04645, saving model to twitter_ner_crf.h5
196/196 [==============================] - 29s 65ms/step - loss: 0.0846 - val_loss: 0.0464
Epoch 2/300
194/196 [============================>.] - ETA: 0s - loss: 0.0380
Epoch 2: val_loss improved from 0.04645 to 0.03888, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 32ms/step - loss: 0.0381 - val_loss: 0.0389
Epoch 3/300
195/196 [============================>.] - ETA: 0s - loss: 0.0342
Epoch 3: val_loss improved from 0.03888 to 0.03484, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 33ms/step - loss: 0.0342 - val_loss: 0.0348
Epoch 4/300
195/196 [============================>.] - ETA: 0s - loss: 0.0300
Epoch 4: val_loss improved from 0.03484 to 0.02962, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 33ms/step - loss: 0.0300 - val_loss: 0.0296
Epoch 5/300
195/196 [============================>.] - ETA: 0s - loss: 0.0249
Epoch 5: val_loss improved from 0.02962 to 0.02404, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 32ms/step - loss: 0.0249 - val_loss: 0.0240
Epoch 6/300
196/196 [==============================] - ETA: 0s - loss: 0.0207
Epoch 6: val_loss improved from 0.02404 to 0.02076, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 33ms/step - loss: 0.0207 - val_loss: 0.0208
Epoch 7/300
195/196 [============================>.] - ETA: 0s - loss: 0.0178
Epoch 7: val_loss improved from 0.02076 to 0.01858, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 32ms/step - loss: 0.0178 - val_loss: 0.0186
Epoch 8/300
195/196 [============================>.] - ETA: 0s - loss: 0.0161
Epoch 8: val_loss improved from 0.01858 to 0.01733, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 33ms/step - loss: 0.0161 - val_loss: 0.0173
Epoch 9/300
195/196 [============================>.] - ETA: 0s - loss: 0.0151
Epoch 9: val_loss improved from 0.01733 to 0.01596, saving model to twitter_ner_crf.h5
196/196 [==============================] - 7s 34ms/step - loss: 0.0151 - val_loss: 0.0160
Epoch 10/300
196/196 [==============================] - ETA: 0s - loss: 0.0142
Epoch 10: val_loss improved from 0.01596 to 0.01520, saving model to twitter_ner_crf.h5
196/196 [==============================] - 8s 38ms/step - loss: 0.0142 - val_loss: 0.0152
Epoch 11/300
195/196 [============================>.] - ETA: 0s - loss: 0.0134
Epoch 11: val_loss improved from 0.01520 to 0.01443, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 33ms/step - loss: 0.0134 - val_loss: 0.0144
Epoch 12/300
195/196 [============================>.] - ETA: 0s - loss: 0.0127
Epoch 12: val_loss improved from 0.01443 to 0.01376, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 32ms/step - loss: 0.0127 - val_loss: 0.0138
Epoch 13/300
195/196 [============================>.] - ETA: 0s - loss: 0.0121
Epoch 13: val_loss improved from 0.01376 to 0.01312, saving model to twitter_ner_crf.h5
196/196 [==============================] - 6s 32ms/step - loss: 0.0121 - val_loss: 0.0131
Epoch 14/300
194/196 [============================>.] - ETA: 0s - loss: 0.0115
Epoch 14: val_loss improved from 0.01312 to 0.01228, saving model to twitter_ner_crf.h5
```

Let's load the best model

```
model.load_weights('twitter_ner_crf.h5')
```

```
crf_model = tf.keras.Model(inputs=model.input, outputs=[model.output, model.get_layer('crf').output, model.input])
```

Let's calculate average accuracy of the model on test set

```
def calculate_accuracy(y_true, y_pred):
  '''Convert categorical one hot encodings to indices and compute accuracy

  Args:
    `y_true`: true values
    `y_pred`: model predictions

  Returns:
    Integer, accuracy of prediction
  '''
  acc_metric = tf.keras.metrics.Accuracy()
  y_true = tf.argmax(y_true, axis=-1)
  return acc_metric(y_true, y_pred).numpy().item()

def calculate_model_accuracy(crf_model, test_sentences, test_labels):
  '''Calculates average validation accuracy of model'''

  # Batch the dataset
  batched_validation_set = tf.data.Dataset.from_tensor_slices((test_sentences, test_labels)).batch(32)

  average_acc = 0
  # Iterate through batches
  for batch_test_sentences, batch_test_labels in batched_validation_set:
    predicted_labels, _, _, _ = crf_model(batch_test_sentences)[1]
    average_acc += calculate_accuracy(batch_test_labels, predicted_labels)

  average_acc/=len(batched_validation_set)
  return average_acc

average_acc = calculate_model_accuracy(crf_model, test_sentences, test_labels)

print("*"*32)
print(f"Average accuracy of model on test set: {average_acc:.3f}")
```

```
********************************
Average accuracy of model on test set: 0.986
```

BERT Model

Getting the bert model

```
from transformers import AutoConfig, TFAutoModelForTokenClassification

MODEL_NAME = 'bert-base-uncased'
```

Loading the tokenizer

```
from transformers import AutoTokenizer
tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME) # Load bert-base-uncased tokenizer
```

Downloading: 100%	28.0/28.0 [00:00<00:00, 956B/s]
Downloading: 100%	570/570 [00:00<00:00, 18.4kB/s]
Downloading: 100%	232k/232k [00:00<00:00, 6.52MB/s]
Downloading: 100%	466k/466k [00:00<00:00, 8.00MB/s]

- tokenizer adds 101 and 102 token id at the start and end of the tokens
- using[1:-1] to eliminate the extra 101, 102 that tokenizer adds
- Let us have a peak at tokenization of a training sample

```
sample=train_samples[10] # Random tokenized sample
for token, tag in sample:
  for subtoken in tokenizer(token)['input_ids'][1:-1]:
    print(token,subtoken)
```

```
RT 19387
@Hatshepsutely 1030
```

```
@Hatshepsutely 16717
@Hatshepsutely 5369
@Hatshepsutely 4523
@Hatshepsutely 10421
@Hatshepsutely 2135
: 1024
@adamlambert 1030
@adamlambert 4205
@adamlambert 10278
@adamlambert 8296
please 3531
, 1010
oh 2821
please 3531
wear 4929
the 1996
infamous 14429
beach 3509
hat 6045
tonight 3892
during 2076
your 2115
encore 19493
( 1006
in 1999
lieu 22470
of 1997
a 1037
rasta 20710
rasta 2696
wig) 24405
wig) 1007
. 1012
&lt; 1004
&lt; 8318
&lt; 1025
3333 21211
3333 2509
```

Get Datasets

```
import numpy as np
import tqdm

def tokenize_sample(sample):
  # Expand label to all subtokens and add 'O' label to start and end tokens
  seq = [
    (subtoken, tag)
    for token, tag in sample
    for subtoken in tokenizer(token.lower())['input_ids'][1:-1]
  ]
  return [(3, 'O')] + seq + [(4, 'O')]

def preprocess(samples, tag2id):
  tokenized_samples = list((map(tokenize_sample, samples)))
  max_len = max(map(len, tokenized_samples))

  # Subtokens
  X_input_ids = np.zeros((len(samples), max_len), dtype=np.int32)

  # Masks
  X_input_masks = np.zeros((len(samples), max_len), dtype=np.int32)

  # labels
  y = np.zeros((len(samples), max_len), dtype=np.int32)

  for i, sentence in enumerate(tokenized_samples):
    for j in range(len(sentence)):
      X_input_masks[i, j] = 1
    for j, (subtoken_id, tag) in enumerate(sentence):
      X_input_ids[i, j] = subtoken_id
      y[i, j] = tag2id[tag]
  return (X_input_ids, X_input_masks), y

X_train, y_train = preprocess(train_samples, tag2id)
X_test, y_test = preprocess(test_samples, tag2id)
```

Loading model

```
config = AutoConfig.from_pretrained(MODEL_NAME, num_labels=len(schema),
                                    id2tag=id2tag, tag2id=tag2id) # Bert config

model = TFAutoModelForTokenClassification.from_pretrained(MODEL_NAME,
                                                          config=config) # Loading Bert model
model.summary()
```

```
Downloading: 100%                                    536M/536M [00:11<00:00, 44.3MB/s]
All model checkpoint layers were used when initializing TFBertForTokenClassification.

Some layers of TFBertForTokenClassification were not initialized from the model check
You should probably TRAIN this model on a down-stream task to be able to use it for p
Model: "tf_bert_for_token_classification"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bert (TFBertMainLayer)      multiple                  108891648

 dropout_37 (Dropout)        multiple                  0

 classifier (Dense)          multiple                  16918

=================================================================
Total params: 108,908,566
Trainable params: 108,908,566
Non-trainable params: 0
```

Fit model on training data

```
BATCH_SIZE=32

optimizer = tf.keras.optimizers.Adam(learning_rate=0.0001) # Creating optimizer

loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

model.compile(optimizer=optimizer, loss=loss, metrics=metric)

history = model.fit(X_train, y_train,
                  validation_split=0.2, epochs=10,
                  batch_size=BATCH_SIZE)
```

```
Epoch 1/10
157/157 [==============================] - 188s 1s/step - loss: 0.1666 - accuracy: 0.9630 - val_loss: 0.0701 - val_accuracy: 0.9845
Epoch 2/10
157/157 [==============================] - 171s 1s/step - loss: 0.0495 - accuracy: 0.9888 - val_loss: 0.0497 - val_accuracy: 0.9879
Epoch 3/10
157/157 [==============================] - 172s 1s/step - loss: 0.0311 - accuracy: 0.9919 - val_loss: 0.0432 - val_accuracy: 0.9891
Epoch 4/10
157/157 [==============================] - 172s 1s/step - loss: 0.0211 - accuracy: 0.9942 - val_loss: 0.0434 - val_accuracy: 0.9900
Epoch 5/10
157/157 [==============================] - 172s 1s/step - loss: 0.0142 - accuracy: 0.9963 - val_loss: 0.0416 - val_accuracy: 0.9911
Epoch 6/10
157/157 [==============================] - 172s 1s/step - loss: 0.0096 - accuracy: 0.9975 - val_loss: 0.0408 - val_accuracy: 0.9914
Epoch 7/10
157/157 [==============================] - 172s 1s/step - loss: 0.0068 - accuracy: 0.9984 - val_loss: 0.0448 - val_accuracy: 0.9913
Epoch 8/10
157/157 [==============================] - 172s 1s/step - loss: 0.0054 - accuracy: 0.9987 - val_loss: 0.0437 - val_accuracy: 0.9907
Epoch 9/10
157/157 [==============================] - 172s 1s/step - loss: 0.0042 - accuracy: 0.9990 - val_loss: 0.0465 - val_accuracy: 0.9913
Epoch 10/10
157/157 [==============================] - 172s 1s/step - loss: 0.0042 - accuracy: 0.9989 - val_loss: 0.0489 - val_accuracy: 0.9911
```

- Lets have a side by side view of true labels and model predictions
- Arranged as an array of Tuple(token, true label, model prediction)

```
def aggregate(sample, predictions):
  results = []
  i = 1
  for token, y_true in sample:
      nr_subtoken = len(tokenizer(token.lower())['input_ids']) - 2 # Extracting word tokens
      pred = predictions[i:i+nr_subtoken] # Extracting predictions
      i += nr_subtoken
      y_pred = schema[np.argmax(np.sum(pred, axis=0))] # Get label of prediction
      results.append((token, y_true, y_pred))
```

```
for i in range(10,15):
  print(predictions[i])
```

```
[('I', 'O', 'O'), ('drive', 'O', 'O'), ('by', 'O', 'O'), ('that', 'O', 'O'), ('motel', 'O', 'O'), ('almost', 'O', 'O'), ('every', '(
[('Apple', 'B-product', 'B-product'), ('MacBook', 'I-product', 'I-product'), ('Pro', 'I-product', 'I-product'), ('A1278', 'I-product
[('Tuff', 'B-musicartist', 'B-musicartist'), ('Culture', 'I-musicartist', 'I-musicartist'), ('-', 'O', 'O'), ('Destiny', 'B-product
[('December', 'O', 'O'), ('23', 'O', 'O'), (',', 'O', 'O'), ('2015', 'O', 'O'), ('at', 'O', 'O'), ('03:44', 'O', 'O'), ('PM', 'O',
[('RT', 'O', 'O'), ('@YahooDrSaturday', 'O', 'O'), (':', 'O', 'O'), ('This', 'O', 'O'), ('is', 'O', 'O'), ('how', 'O', 'O'), ('Arkar
```

```
model.save_pretrained("output/NER_pretrained")
```

Could not connect to the reCAPTCHA service. Please check your internet connection and reload to get a reCAPTCHA challenge.