Import Libraries

In [1]:
```
import nltk
import tensorflow as tf
import keras
from gensim.models import Word2Vec
import multiprocessing
import os
from keras.initializers import Constant
import matplotlib.pyplot as plt
import keras.backend as K
from keras.utils import plot_model
from keras.callbacks import ModelCheckpoint,EarlyStopping
```

```
Using TensorFlow backend.
C:\Program Files\Python36\lib\site-packages\gensim\utils.py:1212: UserWarning: detected Windows; aliasing chunkize to ch
unkize_serial
  warnings.warn("detected Windows; aliasing chunkize to chunkize_serial")
```

Custom Method for F1 Score

In [2]:
```
def f1(y_true, y_pred):
    y_pred = K.round(y_pred)
    tp = K.sum(K.cast(y_true*y_pred, 'float'), axis=0)
    fp = K.sum(K.cast((1-y_true)*y_pred, 'float'), axis=0)
    fn = K.sum(K.cast(y_true*(1-y_pred), 'float'), axis=0)

    p = tp / (tp + fp )
    r = tp / (tp + fn )

    f1 = 2*p*r / (p+r)
    f1 = tf.where(tf.is_nan(f1), tf.zeros_like(f1), f1)
    return K.mean(f1)
```

Importing Data from Penn Treebank Dataset

In [3]:
```
tagged_sentences = nltk.corpus.treebank.tagged_sents()

print(tagged_sentences[0])
print("Tagged sentences: ", len(tagged_sentences))
print("Tagged words:", len(nltk.corpus.treebank.tagged_words()))
```

```
[('Pierre', 'NNP'), ('Vinken', 'NNP'), (',', ','), ('61', 'CD'), ('years', 'NNS'), ('old', 'JJ'), (',', ','), ('will',
'MD'), ('join', 'VB'), ('the', 'DT'), ('board', 'NN'), ('as', 'IN'), ('a', 'DT'), ('nonexecutive', 'JJ'), ('director',
'NN'), ('Nov.', 'NNP'), ('29', 'CD'), ('.', '.')]
Tagged sentences:  3914
Tagged words: 100676
```

Pre-processing of Data

In [4]:
```
import numpy as np

sentences, sentence_tags =[], []
for tagged_sentence in tagged_sentences:
    sentence, tags = zip(*tagged_sentence)
    sentences.append(sentence)
    sentence_tags.append(tags)
print(sentences[0])
print(sentence_tags[0])
# ['Lorillard' 'Inc.' ',' 'the' 'unit' 'of' 'New' 'York-based' 'Loews'
#  'Corp.' 'that' '*T*-2' 'makes' 'Kent' 'cigarettes' ',' 'stopped' 'using'
#  'crocidolite' 'in' 'its' 'Micronite' 'cigarette' 'filters' 'in' '1956'
# '.']
# ['NNP' 'NNP' ',' 'DT' 'NN' 'IN' 'JJ' 'JJ' 'NNP' 'NNP' 'WDT' '-NONE-' 'VBZ'
#  'NNP' 'NNS' ',' 'VBD' 'VBG' 'NN' 'IN' 'PRP$' 'NN' 'NN' 'NNS' 'IN' 'CD'
#  '.']]
```

```
('Pierre', 'Vinken', ',', '61', 'years', 'old', ',', 'will', 'join', 'the', 'board', 'as', 'a', 'nonexecutive', 'directo
r', 'Nov.', '29', '.')
('NNP', 'NNP', ',', 'CD', 'NNS', 'JJ', ',', 'MD', 'VB', 'DT', 'NN', 'IN', 'DT', 'JJ', 'NN', 'NNP', 'CD', '.')
```

Building Dictionary, Adding Padding and Out of Vocabulary

In [5]:
```
from sklearn.model_selection import train_test_split

train_sentences, test_sentences, train_tags, test_tags = train_test_split(sentences, sentence_tags, test_size=0.2)

words, tags = set([]), set([])

for s in train_sentences:
    for w in s:
        words.add(w.lower())

for ts in train_tags:
    for t in ts:
        tags.add(t)

word2index = {w: i + 2 for i, w in enumerate(list(words))}
word2index['-PAD-'] = 0  # The special value used for padding
word2index['-OOV-'] = 1  # The special value used for OOVs

tag2index = {t: i + 1 for i, t in enumerate(list(tags))}
tag2index['-PAD-'] = 0  # The special value used to padding
```

If you want to use Word2Vec

Model Parameters for Word2Vec

In [6]:
```
#Declare Model Parameters
cbow = 0
skipgram = 1
EMB_DIM = 300 #more dimensions, more computationally expensive to train
min_word_count = 1
workers = multiprocessing.cpu_count() #based on computer cpu count
context_size = 7
downsampling = 1e-3
learning_rate = 0.025 #initial learning rate
min_learning_rate = 0.025 #fixated learning rate
num_epoch = 15
```

Initialize and Train Word2Vec

In [7]:
```
w2v = Word2Vec(
    sg = skipgram,
    hs = 1, #hierarchical softmax
    size = EMB_DIM,
    min_count = min_word_count,
    workers = workers,
    window = context_size,
    sample = downsampling,
    alpha = learning_rate,
    min_alpha = min_learning_rate
)
print('Vocabulary size: %d' % len(words))
w2v.build_vocab(train_sentences)
w2v.train(train_sentences,epochs=10,total_examples=w2v.corpus_count)
words = list(w2v.wv.vocab)
# save model in ASCII (word2vec) format
filename = 'embedding_word2vec.txt'
w2v.wv.save_word2vec_format(filename, binary=False)
```

```
Vocabulary size: 10118
```

Create Embedding Matrix

In [8]:
```
embeddings_index={}
f=open(os.path.join('','embedding_word2vec.txt '),encoding="utf-8")
for line in f:
    values=line.split()
    word=values[0]
    coefs=np.asarray(values[1:])
    embeddings_index[word]=coefs
f.close()
```

In [9]:
```
train_sentences_X, test_sentences_X, train_tags_y, test_tags_y = [], [], [], []

num_words=len(word2index)+1
embedding_matrix=np.zeros((num_words,EMB_DIM))
print(word2index)
for word,i in word2index.items():
    if i>num_words:
        continue
    embedding_vector=embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i]=embedding_vector
```

```
'ian': 474, 'credentials': 475, 'promised': 476, 'appearing': 477, 'exchange': 478, 'psychiatric': 479, 'reckless': 48
0, 'rulings': 481, 'simply': 482, 'cents-a-unit': 483, 'triggered': 484, 'theory': 485, 'red-flag': 486, 'referring':
487, '500,004': 488, 'enough': 489, 'aspersions': 490, 'speaks': 491, '143.80': 492, 'rowe': 493, 'caller': 494, 'cole
man': 495, 'averted': 496, 'onus': 497, '1,500': 498, 'relied': 499, '8300': 500, 'included': 501, '50-state': 502, 'd
omestic': 503, 'andersson': 504, 'pa': 505, 'academically': 506, 'davenport': 507, 'commonwealth': 508, 'columbus': 50

9, 'superiors': 510, 'influenced': 511, 'surely': 512, '*-52': 513, 'ensembles': 514, '13.90': 515, 'flag': 516, 'rela
tions': 517, '160': 518, 'decades': 519, 'that': 520, 'smartly': 521, 'climbed': 522, 'oversight': 523, 'easily': 524,
'prices': 525, '59': 526, 'acted': 527, 'projector': 528, 'difficult': 529, 'callers': 530, 'possess': 531, 'were': 53
2, 'einhorn': 533, 'along': 534, 'ingredients': 535, 'bullets': 536, 'case': 537, 'simultaneous': 538, 'inventories':
539, 'seat': 540, '492': 541, 'fast-food': 542, 'nationale': 543, 'women': 544, 'puerto': 545, 'assisted': 546, 'rhon
e': 547, 'upstate': 548, 'finnish': 549, '1.9': 550, 'flourish': 551, 'geography': 552, '6.84': 553, "'re": 554, 'comp
ete': 555, 'pharmaceutical': 556, 'bottom-line': 557, 'priced': 558, 'speculator': 559, 'regenerate': 560, 'u.s.a.': 5
61, '100,980': 562, 'safe-deposit': 563, 'certainly': 564, 'pepperdine': 565, 'leveraging': 566, '3.61': 567, 'wtd': 5
68, 'whelen': 569, 'federal': 570, 'continued': 571, 'demonstrators': 572, 'grants': 573, 'broadcasts': 574, 'lsi': 57
5, 'labeling': 576, 'quarter': 577, 'lesser': 578, 'assert': 579, 'rotie': 580, 'please': 581, 'recently': 582, 'withs
tand': 583, '*-54': 584, 'anything': 585, 'prospects': 586, 'sensation': 587, 'warnings': 588, 'scrupulously': 589, '4
2': 590, 'shareholders': 591, 'bloody': 592, 'forced': 593, 'debate': 594, 'toast': 595, 'massachusetts': 596, 'beddin
g': 597, 'midwesco': 598, 'being': 599, 'operators': 600, 'n.c.': 601, 'trading-company': 602, 'stevenson': 603, 'tryi
ng': 604, 'wickliffe': 605, '4.9': 606, 'adjusting': 607, 'rebound': 608, 'outcry': 609, 'heated': 610, 'turn': 611,
```

Convert Sentences and Tags to Indexes

In [10]:
```
for s in train_sentences:
    s_int = []
    for w in s:
        try:
            s_int.append(word2index[w.lower()])
        except KeyError:
            s_int.append(word2index['-OOV-'])

    train_sentences_X.append(s_int)

for s in test_sentences:
    s_int = []
    for w in s:
        try:
            s_int.append(word2index[w.lower()])
        except KeyError:
            s_int.append(word2index['-OOV-'])

    test_sentences_X.append(s_int)

for s in train_tags:
    train_tags_y.append([tag2index[t] for t in s])

for s in test_tags:
    test_tags_y.append([tag2index[t] for t in s])

print(train_sentences_X[0])
print(test_sentences_X[0])
print(train_tags_y[0])
print(test_tags_y[0])
```

```
[4843, 233, 4189, 4506, 520, 6287, 1663, 7866, 9205, 5432, 7375, 5847, 6253, 7790, 2830, 9077, 9090, 8575, 4929, 9, 765
5, 9205, 9621, 7423, 8372]
[9205, 5188, 233, 8641, 6577, 7980, 7953, 7392, 9920, 8739, 1826, 6443, 31, 8642, 9205, 4664, 233, 4823, 1135, 7501, 657
7, 304, 7754, 5431, 1392, 9376, 6443, 6816, 9889, 9205, 4664, 8739, 2047, 6253, 7314, 736, 2357, 9653, 8739, 5833, 8935,
7312, 9889, 2151, 8372]
[7, 4, 10, 10, 25, 32, 5, 4, 45, 7, 26, 10, 43, 7, 4, 40, 40, 17, 8, 15, 4, 45, 15, 7, 24]
[45, 7, 4, 40, 34, 4, 32, 3, 32, 31, 8, 45, 7, 4, 45, 7, 4, 15, 7, 10, 34, 21, 32, 38, 17, 8, 45, 7, 4, 45, 7, 31, 7, 4
3, 8, 15, 10, 32, 31, 8, 44, 7, 4, 7, 24]
```

Paddings

In [11]:
```python
MAX_LENGTH = len(max(train_sentences_X, key=len))
print(MAX_LENGTH)  # 271

from keras.preprocessing.sequence import pad_sequences

train_sentences_X = pad_sequences(train_sentences_X, maxlen=MAX_LENGTH, padding='post')
test_sentences_X = pad_sequences(test_sentences_X, maxlen=MAX_LENGTH, padding='post')
train_tags_y = pad_sequences(train_tags_y, maxlen=MAX_LENGTH, padding='post')
test_tags_y = pad_sequences(test_tags_y, maxlen=MAX_LENGTH, padding='post')

print(train_sentences_X[0])
print(train_tags_y[0])
```

```
271
[4843  233 4189 4506  520 6287 1663 7866 9205 5432 7375 5847 6253 7790
 2830 9077 9090 8575 4929    9 7655 9205 9621 7423 8372    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0    0    0    0    0    0    0    0    0    0
    0    0    0    0    0]
[ 7  4 10 10 25 32  5  4 45  7 26 10 43  7  4 40 40 17  8 15  4 45 15  7
 24  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
  0  0  0  0  0  0  0]
```

Create your Keras Model

In [12]:
```
from keras.models import Sequential
from keras.layers import Dense, CuDNNLSTM,LSTM, InputLayer, Bidirectional, TimeDistributed, Embedding, Activation,Dropout
from keras.optimizers import Adam
from keras.models import load_model

model = Sequential()
#embedding_layer=Embedding(num_words,EMB_DIM,embeddings_initializer=Constant(embedding_matrix),input_length=MAX_LENGTH,trai
embedding_layer=Embedding(num_words, 300,mask_zero=True)
model.add(InputLayer(input_shape=(MAX_LENGTH, )))
model.add(embedding_layer)
model.add(Bidirectional(LSTM(128, return_sequences=True,activation="tanh")))
model.add(Dropout(0.5))
model.add(Bidirectional(LSTM(128, return_sequences=True,activation="tanh")))
model.add(Dropout(0.5))
model.add(TimeDistributed(Dense(len(tag2index),activation="relu")))
model.add(Activation('softmax'))

model.compile(loss='categorical_crossentropy',
              optimizer=Adam(0.01),
              metrics=["accuracy",f1])
#plot_model(model, to_file='model.png')
model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 271, 300)          3036300
_________________________________________________________________
bidirectional_1 (Bidirection (None, 271, 256)          439296
_________________________________________________________________
dropout_1 (Dropout)          (None, 271, 256)          0
_________________________________________________________________
bidirectional_2 (Bidirection (None, 271, 256)          394240
_________________________________________________________________
dropout_2 (Dropout)          (None, 271, 256)          0
_________________________________________________________________
time_distributed_1 (TimeDist (None, 271, 47)           12079
_________________________________________________________________
activation_1 (Activation)    (None, 271, 47)           0
=================================================================
Total params: 3,881,915
Trainable params: 3,881,915
Non-trainable params: 0
_________________________________________________________________
```

One-Hot Encoding

In [13]:
```
def to_categorical(sequences, categories):
    cat_sequences = []
    for s in sequences:
        cats = []
        for item in s:
            cats.append(np.zeros(categories))
            cats[-1][item] = 1.0
        cat_sequences.append(cats)
    return np.array(cat_sequences)


cat_train_tags_y = to_categorical(train_tags_y, len(tag2index))
print(cat_train_tags_y[0])
```

```
[[0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 [0. 0. 0. ... 0. 0. 0.]
 ...
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]
 [1. 0. 0. ... 0. 0. 0.]]
```

Training the Model

In [14]:
```
es = EarlyStopping(monitor='val_acc', mode='max',verbose=1,patience=3,min_delta=0)
history=model.fit(train_sentences_X, to_categorical(train_tags_y, len(tag2index)),batch_size=256, callbacks=[es], epochs=30
```

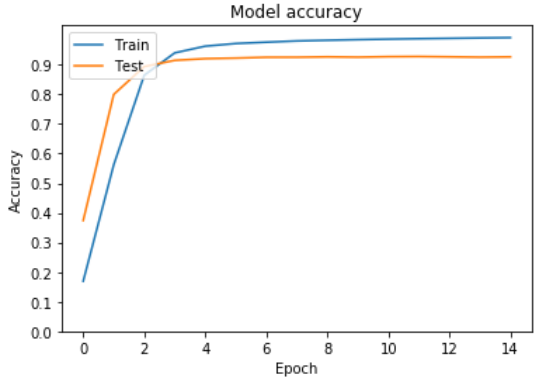
```
Train on 3131 samples, validate on 783 samples
Epoch 1/30
3131/3131 [==============================] - 689s 220ms/step - loss: 3.1568 - acc: 0.1693 - f1: 4.1037e-04 - val_loss:
2.2888 - val_acc: 0.3741 - val_f1: 0.0059
Epoch 2/30
3131/3131 [==============================] - 69s 22ms/step - loss: 1.6797 - acc: 0.5620 - f1: 0.0212 - val_loss: 0.8794
- val_acc: 0.7996 - val_f1: 0.0474
Epoch 3/30
3131/3131 [==============================] - 65s 21ms/step - loss: 0.5743 - acc: 0.8646 - f1: 0.0602 - val_loss: 0.4333
- val_acc: 0.8938 - val_f1: 0.0684
Epoch 4/30
3131/3131 [==============================] - 68s 22ms/step - loss: 0.2451 - acc: 0.9401 - f1: 0.0745 - val_loss: 0.3471
- val_acc: 0.9148 - val_f1: 0.0752
Epoch 5/30
3131/3131 [==============================] - 68s 22ms/step - loss: 0.1537 - acc: 0.9627 - f1: 0.0807 - val_loss: 0.3265
- val_acc: 0.9206 - val_f1: 0.0771
Epoch 6/30
3131/3131 [==============================] - 71s 23ms/step - loss: 0.1096 - acc: 0.9717 - f1: 0.0827 - val_loss: 0.3167
- val_acc: 0.9225 - val_f1: 0.0775
Epoch 7/30
3131/3131 [==============================] - 79s 25ms/step - loss: 0.0884 - acc: 0.9762 - f1: 0.0835 - val_loss: 0.3130
- val_acc: 0.9252 - val_f1: 0.0782
Epoch 8/30
3131/3131 [==============================] - 71s 23ms/step - loss: 0.0720 - acc: 0.9803 - f1: 0.0845 - val_loss: 0.3240
- val_acc: 0.9253 - val_f1: 0.0784
Epoch 9/30
3131/3131 [==============================] - 74s 24ms/step - loss: 0.0615 - acc: 0.9826 - f1: 0.0843 - val_loss: 0.3292
- val_acc: 0.9266 - val_f1: 0.0787
Epoch 10/30
3131/3131 [==============================] - 81s 26ms/step - loss: 0.0537 - acc: 0.9847 - f1: 0.0849 - val_loss: 0.3446
- val_acc: 0.9256 - val_f1: 0.0786
Epoch 11/30
3131/3131 [==============================] - 74s 24ms/step - loss: 0.0475 - acc: 0.9865 - f1: 0.0851 - val_loss: 0.3425
- val_acc: 0.9273 - val_f1: 0.0787
Epoch 12/30
3131/3131 [==============================] - 81s 26ms/step - loss: 0.0438 - acc: 0.9880 - f1: 0.0853 - val_loss: 0.3443
- val_acc: 0.9277 - val_f1: 0.0788
Epoch 13/30
3131/3131 [==============================] - 74s 24ms/step - loss: 0.0377 - acc: 0.9892 - f1: 0.0859 - val_loss: 0.3666
- val_acc: 0.9267 - val_f1: 0.0787
Epoch 14/30
3131/3131 [==============================] - 78s 25ms/step - loss: 0.0338 - acc: 0.9905 - f1: 0.0859 - val_loss: 0.3720
- val_acc: 0.9255 - val_f1: 0.0784
Epoch 15/30
3131/3131 [==============================] - 77s 25ms/step - loss: 0.0303 - acc: 0.9915 - f1: 0.0859 - val_loss: 0.3821
- val_acc: 0.9265 - val_f1: 0.0785
Epoch 00015: early stopping
```

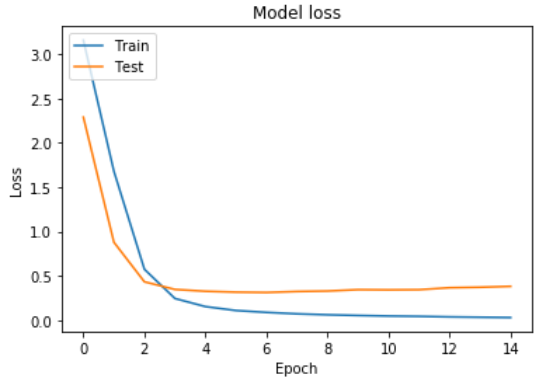
Accompanying Visualizations

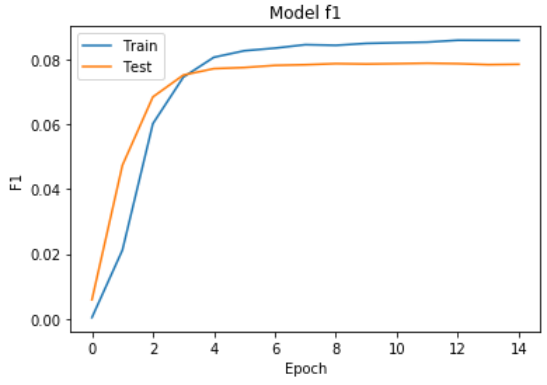
In [15]:
```
# Plot training & validation accuracy values
plt.figure()
plt.plot(history.history['acc'])
plt.plot(history.history['val_acc'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.yticks(np.arange(0,1,step=0.1))
plt.savefig("fake_acc.png")

plt.figure( )
# Plot training & validation loss values
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.savefig("loss.png")

# Plot training & validation loss values
plt.figure()
plt.plot(history.history['f1'])
plt.plot(history.history['val_f1'])
plt.title('Model f1')
plt.ylabel('F1')
plt.xlabel('Epoch')
plt.legend(['Train', 'Test'], loc='upper left')
plt.savefig("f1.png")
```







Model Evaluation

In [16]:
```
scores = model.evaluate(test_sentences_X, to_categorical(test_tags_y, len(tag2index)))
print(model.metrics_names)   # acc: 99.09751977804825
print(scores)   # acc: 99.09751977804825
model.save("model.h5")
```

```
783/783 [==============================] - 22s 28ms/step
['loss', 'acc', 'f1']
[0.38278851955938764, 0.9263724499735339, 0.032061531792464994]
```

Testing the Model

In [17]:
```
test_samples = [
    "Mr. Roxas will run for President".split(),
    "There is a criminal on the run".split()
]

# [['running', 'is', 'very', 'important', 'for', 'me', '.'], ['I', 'was', 'running', 'every', 'day', 'for', 'a', 'month',

test_samples_X = []
for s in test_samples:
    s_int = []
    for w in s:
        try:
            s_int.append(word2index[w.lower()])
        except KeyError:
            s_int.append(word2index['-OOV-'])
    test_samples_X.append(s_int)

test_samples_X = pad_sequences(test_samples_X, maxlen=MAX_LENGTH, padding='post')


predictions = model.predict(test_samples_X)
def logits_to_tokens(sequences, index):
    token_sequences = []
    for categorical_sequence in sequences:
        token_sequence = []
        for categorical in categorical_sequence:
            token_sequence.append(index[np.argmax(categorical)])

        token_sequences.append(token_sequence)

    return token_sequences


final_pred=logits_to_tokens(predictions, {i: t for t, i in tag2index.items()})
for x in range(len(test_samples)):
    print(test_samples[x])
    print(final_pred[x][0:len(test_samples[x])])
```

```
['Mr.', 'Roxas', 'will', 'run', 'for', 'President']
['NNP', 'NNP', 'MD', 'VB', 'IN', 'NN']
['There', 'is', 'a', 'criminal', 'on', 'the', 'run']
['EX', 'VBZ', 'DT', 'JJ', 'IN', 'DT', 'NN']
```

In [ ]:
```

```