Location Extraction from Spanish Text

Dmitry Kondratyev

dokondr@gmail.com

Table of Contents

Problem statement

Goal: Extract crime event locations from online news articles.

Location dictionaries

Using location dictionaries (gazetteers) looks like the most straightforward approach for location extraction from text. Yet analyzing Mexican location names illustrates cases when ambiguities in event location can not be resolved from text only without using additional external context.

Consider, for a example, a message:

```
'Pachuca, Hidalgo.- Provistos de sopletes y otras herramientas,
entre cinco y siete bandidos sorprendieron y sometieron a los
guardias de seguridad privada que cuidan las instalaciones de la
Torre de Rectoria,  de la Universidad Autonoma del Estado de
Hidalgo (UAEH), para saquear el cajero automatico, en la
madrugada.'
```

In the above text the following tokens need to be analyzed as possible location candidates:

```
tokens:  ['Pachuca', 'Hidalgo.-', 'Provistos', 'Torre',
'Rectoria', 'Universidad', 'Autonoma', 'Estado', 'Hidalgo',
'UAEH']
```

Using dictionary of locations we find matches for the given tokens both in states and municipalities! Here states are found from the assumption that our tokens represent municipalities. Element in the following list has state first and municipality second:

```
[('Hidalgo', 'Pachuca de Soto'),
 ('Veracruz', 'Martínez de la Torre'),
 ('Oaxaca', 'San Juan del Estado'),
 ('Estado de México', ''),
 ('Ciudad de México', 'Miguel Hidalgo'),
 ('Tamaulipas', 'Hidalgo'),
 ('Jalisco', 'San Martín Hidalgo'),
 ('Jalisco', 'Villa Hidalgo'),
 ('Durango', 'Hidalgo'),
 ('Puebla', 'Tepatlaxco de Hidalgo'),
 ('San Luis Potosí', 'Villa Hidalgo'),
 ('Hidalgo', 'Juárez Hidalgo'),
 ('Oaxaca', 'Mesones Hidalgo'),
 ('Oaxaca', 'Pluma Hidalgo'),
 ('Oaxaca', 'San Nicolás Hidalgo'),
 ('Oaxaca', 'Unión Hidalgo'),
 ('Oaxaca', 'Chalcatongo de Hidalgo'),
 ('Oaxaca', 'Villa Hidalgo'),
 ('Michoacán', 'Hidalgo'),
 ('Veracruz', 'Poza Rica de Hidalgo'),
 ('Veracruz', 'Zozocolco de Hidalgo'),
 ('Guanajuato', 'Dolores Hidalgo Cuna de la Independencia
Nacional'),
 ('Zacatecas', 'Villa Hidalgo'),
 ('Tlaxcala', 'Acuamanala de Miguel Hidalgo'),
 ('Chihuahua', 'Hidalgo del Parral'),
 ('Chiapas', 'Frontera Hidalgo'),
 ('Sonora', 'Villa Hidalgo'),
 ('Nuevo León', 'Sabinas Hidalgo'),
 ('Nuevo León', 'Hidalgo'),
 ('Coahuila', 'Hidalgo')]
```

We can further reduce possible candidates of states by matching state list with those tokens we have in the news article text. After this match and after removing duplicates in municipalities we have:

```
states:  {'Estado de México', 'Hidalgo'}
municipalities:  {'Mesones Hidalgo', 'Unión Hidalgo', 'Zozocolco
de Hidalgo', 'Dolores Hidalgo Cuna de la Independencia Nacional',
'Hidalgo del Parral', 'Sabinas Hidalgo', 'Pachuca de Soto', 'Villa
Hidalgo', 'Hidalgo', 'San Juan del Estado', 'Frontera Hidalgo',
'Pluma Hidalgo', 'Martínez de la Torre', 'Juárez Hidalgo',
'Chalcatongo de Hidalgo', 'San Martín Hidalgo', 'Poza Rica de
Hidalgo', 'Miguel Hidalgo', 'Tepatlaxco de Hidalgo', 'Acuamanala
de Miguel Hidalgo', 'San Nicolás Hidalgo'}
```

Next, after finding from location dictionary what municipalities really exist in Estado de México and Hidalgo states and finding intersection of these municipalities with our tokens we get:

1. For state Hidalgo candidate municipalities are: `{'Juárez Hidalgo', 'Pachuca de Soto'}`

2. For state Estado de México we have empty set of candiate municipalities.

As a result we could reduce all possible state candidates to one: `Hidalgo` with two candidate municipalities that we need additional context to choose from. Also note that `Hidalgo` is a state and also a part of municipality name `Juárez Hidalgo`. And this is not the only case, there are other cases like this.

ML approach

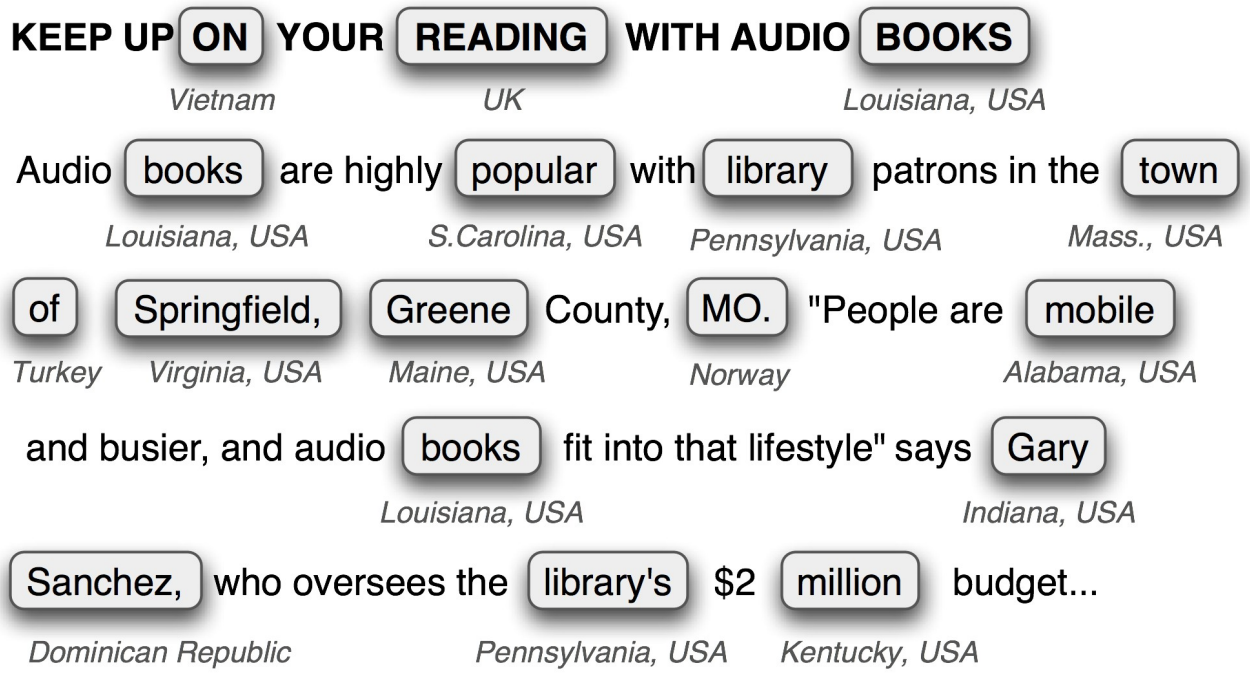
Location extraction from text is a NLP task that is a part of more general Named Entity Recognition task. From [1]: “Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates, and so on. “
Commonly used types of NEs are given inTable 1 [1]. Here "Facility" means human-made artifacts in the domains of architecture and civil engineering; and "GPE" - geo-political entities such as city, state/province, and country.

Table 1:

Commonly Used Types of Named Entity

NE Type	Examples
ORGANIZATION	Georgia-Pacific Corp., WHO
PERSON	Eddy Bonte, President Obama
LOCATION	Murray River, Mount Everest
DATE	June, 2008-06-29
TIME	two fifty a m, 1:30 p.m.
MONEY	175 million Canadian Dollars, GBP 10.40
PERCENT	twenty pct, 18.75 %
FACILITY	Washington Monument, Stonehenge
GPE	South East Asia, Midlothian

“The goal of a named entity recognition (NER) system is to identify all textual mentions of the named entities. “[1]
Looking up named entities in dictionaries (gazetteers) does not always work. A good example when gazetteer provides controversial information is given in [1]:
“For example, in the case of locations, we could use a gazetteer, or geographical dictionary, such as the Alexandria Gazetteer or the Getty Gazetteer. However, doing this blindly runs into problems:



Location Detection by Simple Lookup for a News Story: Looking up every word in a gazetteer is error-prone; case distinctions may help, but these are not always present."

What makes extraction from text non-trivial is that for a lot of entities, like location or organizations names, we just don't have any prior knowledge about them. Yet this does not present any problem for a human, who when reading unknown words understands from surrounding context what these words may stand for. Thus we need an algorithm that can use contextual information from the sentence to predict entity types as humans do.
Recurrent Neuron Networks (RNN) [2] can do just that: predict entity type taking into account context that surrounds it.
In this work we use Bi-LSTM [3], which is a variation of RNN, with surrounding words and text casing as a context.

Data preprocessing

To train Bi-LSTM we use Spanish data from CONLL 2002 corpus [4]. According to CONLL 2002 creators, the data consists of three datasets: one training file and two test files 'testa' and 'testb'. We use all three files to train network and a separate data set of Spanish news articles to test it.

```
# Read NLTK Spanish corpus
nltk.corpus.conll2002.fileids()
```

```
['esp.testa', 'esp.testb', 'esp.train', 'ned.testa', 'ned.testb', 'ned.train']
```

```
%%time
train_sents = list(nltk.corpus.conll2002.iob_sents('esp.train'))
test_sents = list(nltk.corpus.conll2002.iob_sents('esp.testb'))
dev_sents = list(nltk.corpus.conll2002.iob_sents('esp.testa'))
```

```
CPU times: user 6.17 s, sys: 288 ms, total: 6.46 s
Wall time: 6.76 s
```

```
data = train_sents + test_sents + dev_sents
len(data)
```

```
11755
```

All in all 11755 entries to train network. Entries in CONLL 2002 are POS (part of speech) and IOB tagged. IOB stands for 'Inside', 'Outside' and 'Beginning' tags.
The **IOB Tagging** system contains tags of the form:

1. **B-{CHUNK_TYPE}** – for the word in the **B**eginning chunk
2. **I-{CHUNK_TYPE}** – for words **I**nside the chunk
3. **O** – **O**utside any chunk

For example, sentence:

“Por su parte , el Abogado General de Victoria , Rob Hulls , indicó que no hay nadie que controle que las informaciones contenidas en CrimeNet son veraces . “

in CONLL 2002 will be represented as:

```
[('Por', 'SP', 'O'),
 ('su', 'DP', 'O'),
 ('parte', 'NC', 'O'),
 (',', 'Fc', 'O'),
 ('el', 'DA', 'O'),
 ('Abogado', 'NC', 'B-PER'),
 ('General', 'AQ', 'I-PER'),
 ('de', 'SP', 'O'),
 ('Victoria', 'NC', 'B-LOC'),
 (',', 'Fc', 'O'),
 ('Rob', 'NC', 'B-PER'),
 ('Hulls', 'AQ', 'I-PER'),
 (',', 'Fc', 'O'),
 ('indicó', 'VMI', 'O'),
 ('que', 'CS', 'O'),
 ('no', 'RN', 'O'),
 ('hay', 'VAI', 'O'),
 ('nadie', 'PI', 'O'),
 ('que', 'PR', 'O'),
 ('controle', 'VMS', 'O'),
 ('que', 'CS', 'O'),
 ('las', 'DA', 'O'),
 ('informaciones', 'NC', 'O'),
 ('contenidas', 'AQ', 'O'),
 ('en', 'SP', 'O'),
 ('CrimeNet', 'NC', 'B-MISC'),
 ('son', 'VSI', 'O'),
 ('veraces', 'AQ', 'O'),
 ('.', 'Fp', 'O')]
```

In our current work POS tags are not used to train network, so for all train data we keep only IOB tags:

```
[['Por', 'O'],
 ['su', 'O'],
 ['parte', 'O'],
 [',', 'O'],
 ['el', 'O'],
 ['Abogado', 'B-PER'],
 ['General', 'I-PER'],
 ['de', 'O'],
 ['Victoria', 'B-LOC'],
 [',', 'O'],
 ['Rob', 'B-PER'],
 ['Hulls', 'I-PER'],
 [',', 'O'],
 ['indicó', 'O'],
 ['que', 'O'],
 ['no', 'O'],
 ['hay', 'O'],
 ['nadie', 'O'],
 ['que', 'O'],
 ['controle', 'O'],
 ['que', 'O'],
 ['las', 'O'],
 ['informaciones', 'O'],
 ['contenidas', 'O'],
 ['en', 'O'],
 ['CrimeNet', 'B-MISC'],
 ['son', 'O'],
 ['veraces', 'O'],
 ['.', 'O']]
```

In the CoNLL2002 task [4], the entities are `LOC, PER, ORG and MISC` denote *locations, persons, organizations* and *miscellaneous*. The no-entity tag is `O`. Because some entities (like city `San Luis Potosí`) have multiple words, we use a *tagging scheme* to distinguish between the beginning (tag `B-...`), or the inside of an entity (tag `I-...`).

Contextual Word Representation

For each word, we build a vector w a R^n that captures the meaning and relevant features for NER. We build this vector as a concatenation of two vectors. First we create word embeddings $w_{Word2Vec}$ a R^{d1} vector using Word2Vec[5]. Then, to address the issue that lot of words don't have a pre-trained word vector, we concatenate word embeddings vector with vector containing features extracted from word characters w_{chars} a R^{d2} . For character level features we use cases of characters that constitute a word:

```
{'PADDING_TOKEN': 7,
 'allLower': 1,
 'allUpper': 2,
 'contains_digit': 6,
 'initialUpper': 3,
 'mainly_numeric': 5,
 'numeric': 0,
 'other': 4}
```

where:

`allLower` – all characters in a word are lower case

`allUpper` – all characters in a word are upper case
`initialUpper` – word is a title, initial char upper, then all lower
`numeric` – word is a digit
`mainly_numeric` – 50% or more characters of a word are digits
`contains_digit` – less then 50% characters of a word are digits
`other` – other character combinations in a word

For word embeddings we use Spanish Billion Word Corpus and Embeddings[6] - corpus of the Spanish language of nearly 1.5 billion words, compiled from different corpora and resources from the web. We use pre-trained word vectors (or embeddings) created in this corpus with the word2vec algorithm, provided by the Gensim [11] package.
We build Bi-LSTM model and then predict named entity classes with Keras[7] - a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.
As we already said, every word vector in our case is in fact a concatenation of two vectors: word embeddings vector and word casing vector. Keras[7] requires word vectors to be indexed with integers. For this reason, at the data preprocessing step we create array of indexes for all words, used both for training and prediction, and array of corresponding word vectors.
Example of word indices:

```
word2Idx
```

```
{'Tablero': 30019,
 'Latorre': 20798,
 'FABIOLA': 54594,
 'TANQUES': 49856,
 'Insumo': 48346,
 'asesoran': 26180,
 'Volarán': 50049,
 'PLANTAS': 32767,
 'Balear': 24539,
 'pamplona': 48180,
 'Cálculos': 24870,
 'Henares': 13449,
 'equipazo': 33234,
 'PATAS': 46603,
 'AMOR': 28355,
```

Example of word vectors:

```
wordEmbeddings[7]
```

```
array([ -2.68900000e-03,  -6.34560000e-02,   9.61900000e-03,
        -7.59120000e-02,   2.57900000e-03,   2.52870000e-02,
        -5.17740000e-02,  -1.58100000e-02,   1.61642000e-01,
        -6.28080000e-02,  -1.38640000e-02,  -9.85600000e-03,
         4.57040000e-02,  -7.46800000e-02,   5.11350000e-02,
        -1.71510000e-02,  -3.56100000e-03,  -1.27600000e-03,
        -7.71270000e-02,   1.66030000e-02,   1.17330000e-02,
        -4.87390000e-02,   3.97480000e-02,   5.42350000e-02,
        -3.64680000e-02,  -5.29450000e-02,  -6.03100000e-03,
         9.96510000e-02,  -2.87180000e-02,   7.59660000e-02,
         5.43790000e-02,  -3.94120000e-02,  -2.74340000e-02,
        -1.07070000e-02,   2.90610000e-02,  -1.19580000e-02,
        -2.70640000e-02,  -3.31640000e-02,  -9.17940000e-02,
        -3.97890000e-02,   4.75860000e-02,   3.72410000e-02,
```

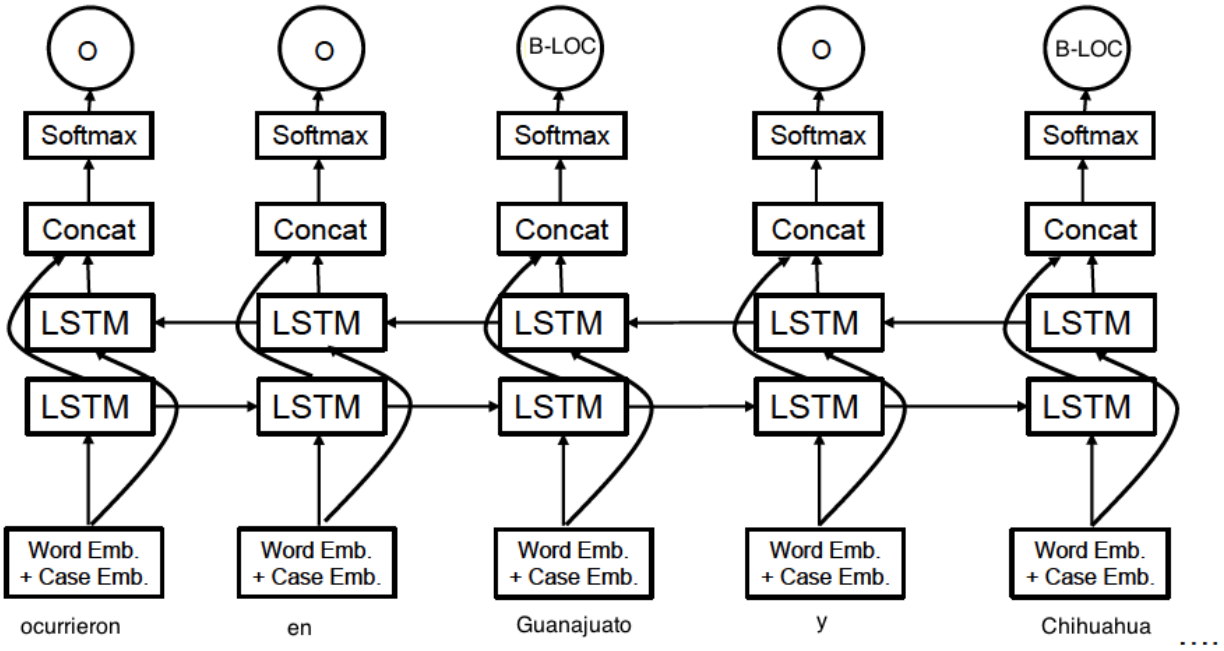
Vectors with uniformly distributed component values are created for words not found in pre-trained embeddings.
As a result, apart from a set of word embeddings a train and test data sets are created. Each entry in a train set includes three arrays: word embedding indexes, word case indexes and labels:

```
train_set[0]
```

```
[[12558, 1, 1445, 1, 1, 1, 18468, 1, 6566, 1, 1],
 [3, 4, 3, 4, 4, 0, 1, 4, 2, 4, 4],
 [0, 3, 0, 3, 3, 3, 3, 3, 1, 3, 3]]
```

Bi-LSTM

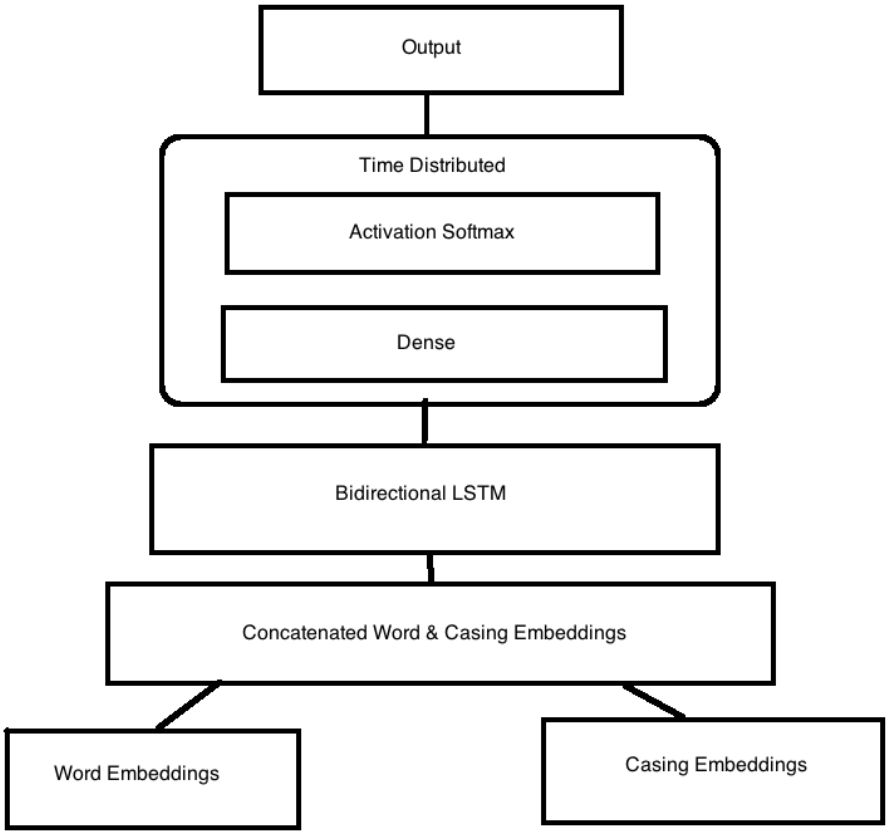
Recurrent neural networks (RNNs) operate on sequential data. They take as input a sequence of vectors $(x_1, x_2, \ldots, x_n)$ and return another sequence $(h_1, h_2, \ldots, h_n)$ that represents some information about the sequence at every step in the input.
In theory RNNs can learn long dependencies, but in practice they tend to be biased towards their most recent inputs in the sequence [8].
Long Short-term Memory Networks (LSTMs) solve this problem and can capture long-range dependencies by incorporating a memory-cell. LSTMs use several gates that control the proportion of the input to give to the memory cell, and the proportion from the previous state to forget [9].
For a given sentence $(x_1, x_2, \ldots, x_n)$ containing n words, each represented as a d -dimensional vector, a forward LSTM computes a representation h_{TL} of the left context of the sentence at every word t . Second, backward LSTM that reads the same sequence in reverse, generates a representation h_{TR} of the right context. Such pair of forward and backward LSTMs, each with different parameters, is called a bidirectional LSTM [10].
A representation of a word in this model is created by concatenating its left and right context representations $h_T = [h_{TL}; h_{TR}]$, to be used in different tagging applications.
The following example shows operation of Bi-LSTM on a part of a sentence containing words "ocurrieron en Guanajuato y Chihuahua".



In this example word vectors are given to a Bi-LSTM. Forward LSTM represents the word and its left context, backward LSTM represents the word and its right context. Concatenating vectors from these two LSTMs creates a representation of the word in its complete context.

Keras model

Keras provides API to build various NN architectures including RNN and LSTM. The following figure shows the main building blocks of Keras model with Bi-LSTM:



Neural networks are defined in Keras as a sequence of layers. We use Keras functional API to create our model. Model has word embeddings and word casing input layers, which outputs are concatenated together. Bi-LSTM is implemented with Bidirectional layer.
The first hidden LSTM layer has 50 memory units and the output layer is a fully connected layer that outputs one value per timestep. A softmax activation function is used on the output to predict classes (labels) of words.
The 'dropout' and 'recurrent_dropout' LSTM parameters specified in this layer are fractions of the units to drop for the linear transformation of the inputs and recurrent state respectively. Dropout is used to prevent model overfitting while training.
A TimeDistributed wrapper layer is used around the output Dense layer so, that one value per timestep can be predicted given the full sequence provided as input. In this case Bi-LSTM hidden Dense layer returns a sequence of values (one per timestep) rather than a single value for the whole input sequence.
The following illustration shows creation of Keras model:

```
wordEmbeddings.shape, caseEmbeddings.shape
```

```
((54819, 300), (8, 8))
```

```
n_out = len(label2Idx)

words_input = Input(shape=(None,), dtype='int32', name='words_input')
words = Embedding(input_dim=wordEmbeddings.shape[0], output_dim=wordEmbeddings.shape[1],  weights=[wordEmbeddings], tra

casing_input = Input(shape=(None,), dtype='int32', name='casing_input')
casing = Embedding(output_dim=caseEmbeddings.shape[1], input_dim=caseEmbeddings.shape[0], weights=[caseEmbeddings], tra

output = concatenate([words, casing])
output = Bidirectional(LSTM(50, return_sequences=True, dropout=0.25, recurrent_dropout=0.25))(output)
output = TimeDistributed(Dense(n_out, activation='softmax'))(output)

#Create our model and compile it using Nadam optimizer with categorical cross-entropy for sparse y-labels
model = Model(inputs=[words_input, casing_input], outputs=[output])
model.compile(loss='sparse_categorical_crossentropy', optimizer='nadam')
model.summary()
```

Training

To train Bi-LSTM we randomly split CONLL 2002 [4] corpus into training and test sets.
For each epoch sentences are randomly selected from train set for the network to be fit on. This ensures that the model does not memorize a single sequence and instead can generalize a solution to solve all possible random input sequences for this problem.
Every time after network is trained, we check the quality of the model using CONLL 2002 [4] test set. We calculate precision, recall and F1 score for this set. As a final result we have:
Precision = 0.891, Recall = 0.889, F1 = 0.890

```
--------- Epoch 11 -----------
418.44 sec for training
--------- Epoch 12 -----------
396.06 sec for training
--------- Epoch 13 -----------
409.45 sec for training
--------- Epoch 14 -----------
429.79 sec for training
--------- Epoch 15 -----------
399.60 sec for training
--------- Epoch 16 -----------
401.84 sec for training
--------- Epoch 17 -----------
415.87 sec for training
--------- Epoch 18 -----------
417.51 sec for training
--------- Epoch 19 -----------
440.48 sec for training
Test-Data: Prec: 0.891, Rec: 0.889, F1: 0.890
459.99 sec for evaluation
```

Prediction

After training is complete, Keras model is saved to a file to be later used for predictions.
We predict word labels for Alethea Spanish Corpus fields 'Titulo', 'Sintesis' and 'Texto'. During data preparation step text in all these fields was already tokenized, indexed , cased and word embeddings were found. Text in 'Titulo', 'Sintesis' and 'Texto' fields, as well as word and case indexes, and predicted word labels are all then saved in a single CSV file.

From all words in text fields we select only those, which labels denote locations (B-LOC, I-LOC). For example, for a 'Sintesis' field in the first row of our data set, containing text:

```
Los casos ocurrieron en Guanajuato y Chihuahua. Mas de 177.000
personas han sido asesinadas y unas 30.000 estan desaparecidas en
desde que el gobierno federal lanzo un operativo militar
antidrogas en 2006
```

The following location predictions were found:

```
[('Guanajuato', 'B-LOC'), ('Chihuahua', 'B-LOC')]
```

Not all news articles in Alethea Spanish Corpus mention locations. So from 2251 news articles in this corpus locations were found in 48% , 59% and 91% of 'Titulo', 'Sintesis' and 'Texto' fields respectively.

As a base line algorithm to detect final event location we use a simple 'best match' function that selects location predicted most of the times in all three fields 'Titulo', 'Sintesis' and 'Texto'. For example:

Titulo:

```
'Ejecutan a cuatro personas en Chihuahua'
```

Sintesis:

```
'Cuatro personas fueron ejecutadas esta noche en la Ciudad de
Chihuahua, informaron autoridades.'
```

Texto:

```
'El homicidio se registro en una vivienda de la calle Articulo 39
en la Colonia Insurgentes II, en el norte, donde irrumpieron
sujetos con armas largas y dispararon en multiples ocasiones.  De
acuerdo con autoridades, los agresores huyeron en una camioneta
negra.  Hasta esta noche las victimas no habian sido
identificadas.  '
```

Predicted locations and 'best match':

```
titulo_locations:  [('Chihuahua', 'B-LOC')]

sintesis_locations:  [('Ciudad', 'B-LOC'), ('de', 'I-LOC'), ('Chihuahua', 'I-
LOC')]

texto_locations:  [('calle', 'B-LOC'), ('Articulo', 'I-LOC'), ('Colonia', 'B-
LOC'), ('Insurgentes', 'I-LOC'), ('II', 'I-LOC')]


*** Best Match:  Chihuahua
```

Results and future work

Location prediction classifier was implemented with Keras Bi-LSTM network. We use CONLL 2002 [4] Spanish corpus to train our network. After evaluating classifier on annotated test set from the same corpus, we have: precision = 0.891, recall = 0.889 and F1 score = 0.890.

To analyze quality of location extraction from Mexican news articles, true event locations must be known. At the moment of this writing true locations are not specified in Alethea Spanish Corpus, that we plan to use for quality analysis. As long as Alethea Spanish Corpus has more specific to Mexico locations then general, with locations from all around the world, CONLL 2002 [4] corpus does, we hope that annotating and using Alethea corpus will improve location prediction in Mexican news.

For cases when ambiguities in event location can not be resolved from text only without using additional external context, such as when the same name can be used both for states and municipalities, additional algorithms should be researched and developed.

References

1. Steven Bird, Ewan Klein, and Edward Loper, “Natural Language Processing with Python – Analyzing Text with the Natural Language Toolkit”.
2. Andrej Karpathy, “The Unreasonable Effectiveness of Recurrent Neural Networks”
3. Christopher Olah, “Understanding LSTM Networks”
4. Conference on Computational Natural Language Learning (CoNLL-2002)
5. Word2Vec https://code.google.com/archive/p/word2vec/
6. SBWCE - Spanish Billion Word Corpus and Embeddings: http://crscardellino.me/SBWCE/
7. Keras: https://keras.io/
8. Yoshua Bengio, Patrice Simard, and Paolo Frasconi. 1994. Learning long-term dependencies with gradient descent is difficult. Neural Networks, IEEE Transactions on, 5(2):157–166.
9. [Hochreiter and Schmidhuber1997] Sepp Hochreiter and Jurgen Schmidhuber. 1997. Long short-term memory. Neural Computation, 9(8):1735–1780.
10. Alex Graves and Jurgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM networks. In Proc.IJCNN
11. Gensim Python library: https://radimrehurek.com/gensim/