Machine Learning Engineer Nanodegree - Capstone Project

Bastian Carvajal Yañez

February 2019

I. Definition

Project Overview

For many decades the algorithms used for Natural Language Processing, on tasks like machine translation and language understanding, were based on complex rules created manually; but today the state of the art was achieved with the use of Deep Neural Networks in many domains (Young et al. 2018). This change was driven by the increased availability of computing resources, but most importantly by the emergence of large annotated corpus of text, like OntoNotes (Weischedel et al. 2013). But, because training this kind of algorithms is a costly process, researchers are looking for ways to train them with small amounts of data.

In this project I explore the Active Learning strategy to train a Recurrent Neural Network for *sequence tagging*. The algorithm will be given batches of sentences, with the text tokenized, which means that the text will be separated into words. For example the phrase *"Jose trabaja para CNN en Sao Paulo, Brasil."* will be tokenized and converted to a list:

```
['Jose', 'trabaja', 'para', 'CNN', 'en', 'Sao', 'Paulo', ',', 'Brasil', '.']
```

Then the algorithm should predict the labels for each of the tokens, and output a list of same size:

```
['B-PER', 'O', 'O', 'B-ORG', 'O', 'B-LOC', 'I-LOC', 'O', 'B-LOC', 'O']
```

The resulting implementation, based on bidirectional LSTM + CRF, and an Active Learning traiing procedure, achieves state-of-the art performance for the NER task (Shen et al. 2017), on the popualar ConLL2002 Dataset (SIGNLL 2002). The key insight is that the model only uses 30% of the available training data...

REVISAR!!!!

Problem Statement

Neural Networks can achieve better accuracy for NLP tasks compared to traditional rule-based algorithms, or even other types of ML models. Yet, a major problem is that you need a lot of data to train them; as a general rule the more data you have the better. One potential solution to this problem is to make the algorithm learn from a small dataset by changing its internal architecture, or by changing the way it is trained.

To solve the above problem I explore the use of Active Learning for training an RNN model. Active Learning is a semi-supervised learning algorithm, in which the model is given a pool of unlabeled data from where to pick examples, it gives a prediction and ask for feedback to a human annotator. This is also called online learning. On each iteration the algorithm calculates the loss, between its predictions and the feedback, and chooses only the less confident prediction to ask for feedback again. This process is likely to converge faster than choosing at random an annotated example from the pool. And, because the model ask for specific examples, there is no need to annotate all the pool by humans.

The implementation is composed of the following steps:

- Build an RNN model for tagging variable length sentences. The architecture is based on a Bidirectional LSTM module as encoder, and a CRF module as decoder; this algorithm is based on the (Huang, Xu, and Yu 2015) paper.
- Build GloVe embeddings to encode words. These are precomputed once, before the training process, so we can reuse this embeddings on different experiments.
- For the Active Learning an automated pipeline is implemented, to simulate the process of interactively annotate the examples by a human.

The utility of the active learning process will be evaluated on multiple experiments. The goal is to achieve at least the same F1 score than with traditional supervised learning (using random sampling), but using only a fraction of the annotated dataset. Results comparable to the ones stated on (Shen et al. 2017) are expected.

Metrics

The F1 score (harmonic mean of precision and recall) will be used to measure the accuracy of the trained model. The formula is:

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{precision} \cdot \text{recall}}{(\beta^2 \cdot \text{precision}) + \text{recall}}$$

This is a standard metric for NER, and is used to compare results with (Huang, Xu, and Yu 2015) and (Shen et al. 2017) papers.

As the model will be trained many times, on an increasing amount of examples, the F1 score will be reported at each run. This way we can compare the runs and determine at which point the model converges. The scores will be plotted to a learning curve, to better visualise this metrics.

If the train takes to much time to complete the Active Learning becomes not practical, so the time it takes each train run will be measured as well. This will be useful to find the hyper parameters that make the algorithm more efficient.

II. Analysis

Data Exploration

Only the CoNLL-2002 Ner dataset (SIGNLL 2002) will be used in this project.[1]

The data contains two languages: Spanish and Dutch, and has three files per language: one for training and two for testing. Each file consists of three columns separated by a single space. Each word has been put on a separate line and there is an empty line after each sentence. The first item on each line is the token, the second is the POS tag, and the third the Named Entity tag. The tags uses de BIO2 syntax: the prefix `B-` denotes the first item of a phrase, and `I-` the continuation of the previous tag. There is also the `O` tag that indicate the token is not part of any named entity. There are four categories for the named entities: person names (PER), organizations (ORG), locations (LOC) and miscellaneous names (MISC).

I gathered this statistics from the files:

file	phrases	words
dataset/conll2002/esp.testa	1915	52923
dataset/conll2002/esp.testb	1517	51533
dataset/conll2002/esp.train	8323	264715
dataset/conll2002/ned.testa	2895	37761
dataset/conll2002/ned.testb	5195	68994
dataset/conll2002/ned.train	15806	202931

And the class distribution is as follows:

file	O	B-ORG	I-ORG	B-PER	B-LOC	I-PER	I-MISC	B-MISC	I-LOC
esp.testa	45356	1700	1366	1222	984	859	654	445	337
esp.testb	45355	1400	1104	1084	735	634	557	339	325
esp.train	231920	7390	4992	4913	4321	3903	3212	2173	1891
ned.testa	34047	748	703	686	479	423	396	215	64
ned.testb	63236	1187	1098	882	807	774	551	410	49
ned.train	183633	4716	3338	3208	2883	2082	1405	1199	467

[1] Dataset is available at: https://www.kaggle.com/nltkdata/conll-corpora/version/1#conll2002.zip

Exploratory Visualization

To better visualize the contents of the dataset a couple of plots were generated. The first one below shows the frequency distribution for the top 50 words in the `dataset/conll2002/esp.train` file.

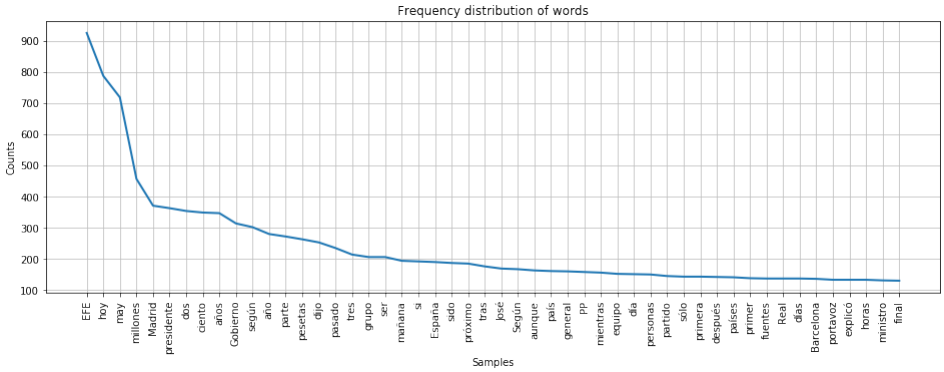

Figure 1: Frequency distribution of words in the "esp.train" file.

The second plot, shows the frequency distribution of the labels corresponding to the same file. Here we can see that the majority of words are tagged as "O", which means it is not part of any *Named Entity*. We can also see that there is slightly more samples with "B-ORG" than "I-ORG", this might be because there are more organizations as unigram than n-grams (the name only is one word, rather than multiple words).

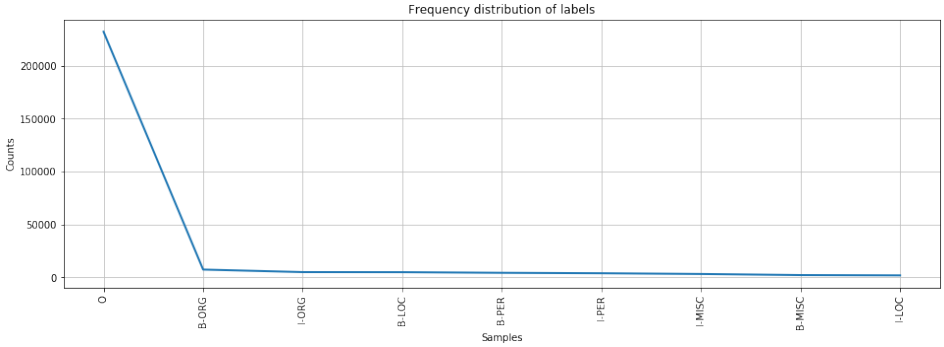

Figure 2: Frequency distribution of the categories in the "esp.train" file.

Algorithms and Techniques

As stated before, the model architecture consist of a Bidirectional LSTM module as the *encoder*, and a CRF module as *decoder*. This architecture is based on the (Huang, Xu, and Yu 2015) paper, and has the following stages that will be explained in detail:

- Word Representation
- Contextual Word Representation
- Decoding

Word Representation

For each word, we want to build a vector $w \in \mathbb{R}^n$ that will capture the meaning and relevant features for the NER task. We're gonna build this vector by extracting the word embeddings $w_{glove} \in \mathbb{R}^d$ from GloVe. We will use pre-trained GloVe vectors[2] on *Common Crawl*, which is a dataset with around 840 Billion words, in many languages, all scrapped from the web.

Contextual Word Representation

Once we have our word representation w we simply run a LSTM (or bi-LSTM) over the sequence of word vectors and obtain another sequence of vectors (the hidden states of the LSTM or the concatenation of the two hidden states in the case of a bi-LSTM), $h \in \mathbb{R}^k$.

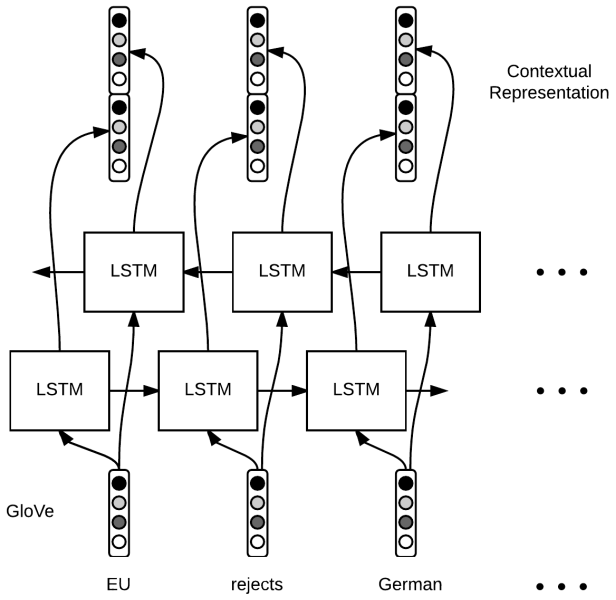

Figure 3: Bidirectional LSTM on top of word representation to extract contextual representation of each word.

Decoding

At this stage, each word w is associated to a vector h that captures information from the meaning of the word, and its context. We can use a fully connected neural network to get

[2]GloVe: Global Vectors for Word Representation

a vector where each entry corresponds to a score for each tag. With this, we want to be able to compute the probability $\mathbb{P}(y_1, \ldots, y_m)$ of a tagging sequence y_t and find the sequence with the highest probability. Here, y_t is the id of the tag for the t-th word.

Using *dynamic programing* we can compute efficiently an optimal sequence. The (Huang, Xu, and Yu 2015) paper describes thoroughly an implementation for this.

Active Learning

The active learning method is based on *Uncertainty Based Sampling*, more specifically "Least Confidence" which is described in (Shen et al. 2017). The paper proposed to sort examples in ascending order according to the probability assigned by the model to the most likely sequence of tags.

$$1 - \text{argmax}_{y_1,\ldots,y_n} \mathbb{P}[y_1, \ldots, y_n | \{x_{ij}\}]$$

Here, an automated pipeline is implemented, to simulate the process of interactively annotate the examples by a human. The heuristic could be as follow:

- First, the model is initialized randomly with 1% of the train dataset from Conll2002, $\mathbb{X}_0 \in \mathbb{Z}$.
- Before each train run t the model will make predictions $\mathbb{Y}_{t-1}$ on the full unlabeled pool (here the model is using the state from the previous train).
- Based on this predictions it will choose the next batch of samples to annotate, $\mathbb{S}_t \in \mathbb{Z}^U$.
- Then, it looks the labels for this samples on the original dataset $\mathbb{S}_t \in \mathbb{Z}^L$ (this is the simulation of a human annotating the examples).
- Finally, it adds this labeled samples to his current train dataset $\mathbb{X}_t = \mathbb{X}_{t-1} + \mathbb{S}_t$. And the model is trained again on the updated dataset $\mathbb{X}_t$.

Benchmark

The algorithm to be implemented will not be design to get state of the art performance on this corpus (F1 score +90.0). But, as the architecture chosen is based on the bi-LSTM-CRF algorithm from (Huang, Xu, and Yu 2015), the results in terms of accuracy should be similar of those reported in that paper (F1 score 84.26).

The benchmark will be focus on the use of Active Learning during training. With the same amount of training data, the model should get at least the same accuracy than trained with random sampling.

III. Methodology

Data Preprocessing

In this case, the only preprocessing of the dataset is to reformat the files to make it easy to read the sentences and labels. Instead of having a token per line, we save one sentence

per line in a new file `train.words.txt`, and separate the tokens by a space; in another file `train.tags.txt` we save the labels, the same way as before. This has the advantage that we can lookup a sentence, and its correspondent labels, by its line index on the file.

With those files we generate a vocabulary `vocab.words.txt`. Then we precompute the word embeddings with pre-trained GloVe vectors from `glove.840B.300d.txt`.[3] For each word in the vocabulary we extract the vector from that file, saving them in the file `glove.npz`, which is a numpy array of shape (*vocab_length* x *embedding_dimensions*).

Implementation

In the beginning I tried to implement the algorithm using the *Tensorflow* low level APIs, and soon I realized that you need a lot of boilerplate to make it work. In particular, to train an RNN on variable length text, you have to create your own logic to batch sentences, taking into account you have to pad the shorter ones. Also, configure the model checkpoints and metrics gathering was a real challenge.

Luckily, I found great tutorials[4][5] about the `tf.estimator` and `tf.data` APIs. Those APIs simplify significantly the implementation, as they abstract the training, evaluation, and metrics serialization (for *Tensorboard*[6]) processes.

bi-LSTM + CRF with TensorFlow Estimators

First, for the ingestion pipeline I used `tf.data.Dataset.from_generator`, which allows you to do the data loading (from file or elsewhere) and some preprocessing in python before feeding it into the graph. As a convenience, we can define the function `input_fn` and pass it to the *Estimator* later on; this function creates the dataset from the generator (which reads the files `train.words.txt` and `train.tags.txt` as explained in previous section), and can be parameterized depending on your needs.

Second, the `model_fn` function is were you define the computation graph. With both functions we can create our model, or estimator, and pass it to the `tf.estimator.train_and_evaluate` method, which runs the training, evaluation, etc.

Here I want to explain a little bit about the architecture:

```
# Word Embeddings
word_ids = vocab_words.lookup(words)
glove = np.load(params['glove'])['embeddings']  # np.array
variable = np.vstack([glove, [[0.] * params['dim']]])
variable = tf.Variable(variable, dtype=tf.float32, trainable=False)
embeddings = tf.nn.embedding_lookup(variable, word_ids)
embeddings = tf.layers.dropout(embeddings, rate=dropout, training=training)
```

[3] Availablle at: http://nlp.stanford.edu/data/glove.840B.300d.zip
[4] Introduction to Estimators - Tensorflow docs
[5] Intro to tf.estimator and tf.data, by Guillaume Genthial
[6] This is a tool for visualize graphs and data from Tensorflow.

The first step in the graph is getting the word representation. Using a vocabulary lookup table we can map token strings to ids. Then, we load a `np.array` containing the pre-trained *GloVe* vectors (as explained in previous section), where the row index corresponds to a token id. Here we set the `trainable` parameter to false because we do not want to change the embeddings. Next, the model will perform a lookup in this array to get the embedding of every token. And finally, we apply dropout to the dense representation to prevent overfitting.

```
# LSTM
t = tf.transpose(embeddings, perm=[1, 0, 2])
lstm_cell_fw = tf.contrib.rnn.LSTMBlockFusedCell(params['lstm_size'])
lstm_cell_bw = tf.contrib.rnn.LSTMBlockFusedCell(params['lstm_size'])
lstm_cell_bw = tf.contrib.rnn.TimeReversedFusedRNN(lstm_cell_bw)
output_fw, _ = lstm_cell_fw(t, dtype=tf.float32, sequence_length=nwords)
output_bw, _ = lstm_cell_bw(t, dtype=tf.float32, sequence_length=nwords)
output = tf.concat([output_fw, output_bw], axis=-1)
output = tf.transpose(output, perm=[1, 0, 2])
output = tf.layers.dropout(output, rate=dropout, training=training)
```

The second step is getting the context representation by applying a *bidirectional LSTM* on top of the token representation. This is composed of two layers, one that reads the sequence forward in time, and the other that reads it backwards. The oputput of both layers are concatenated together, to produce a vector of shape (*time_len* x *batch_size* x *input_size*). This is useful as we can get more context by reading the sentences in both directions; as stated in (Huang, Xu, and Yu 2015) "*we can efficiently make use of past features (via forward states) and future features (via backward states) for a specific time frame*".

We use `tf.contrib.rnn.LSTMBlockFusedCell` cells because this implementation is very efficient on GPUs,[7] and this is critical for the active learning process. At the end, we add a dropout layer that we activate only when training the model.

```
# CRF
logits = tf.layers.dense(output, num_tags)
crf_params = tf.get_variable("crf", [num_tags, num_tags], dtype=tf.float32)
pred_ids, best_score = tf.contrib.crf.crf_decode(logits, crf_params, nwords)
```

The third step is decoding the sentence using a CRF. As explained in previous sections, we use a fully connected layer to get a vector where each entry corresponds to a score for each tag. Then, trough dynamic programing we can calculate the sequence of tags with the higher score. This is already implemented in Tensorflow, so we only add one line here.

[7] As stated in the docs: https://www.tensorflow.org/api_docs/python/tf/contrib/rnn/LSTMBlockFusedCell

```
# Loss
vocab_tags = tf.contrib.lookup.index_table_from_file(params['tags'])
tags = vocab_tags.lookup(labels)
log_likelihood, _ = tf.contrib.crf.crf_log_likelihood(logits, tags, nwords, crf_params)
loss = tf.reduce_mean(-log_likelihood)
if mode == tf.estimator.ModeKeys.TRAIN:
    train_op = tf.train.AdamOptimizer().minimize(loss)
```

We calculate the loss by computing the log likelihood of tag sequences, using the transition params from the CRF. Then, we use AdamOptimizer to minimize the loss.

Active learning

Last but not least, the active learning process has its own challenges.

COMPLETAR!!!!

Refinement

Todo...

COMPLETAR!!!!

IV. Results

Model Evaluation and Validation

Todo...

COMPLETAR!!!!

Justification

Todo...

COMPLETAR!!!!

V. Conclusion

Free-Form Visualization

Todo...

COMPLETAR!!!!

Reflection

Todo...

COMPLETAR!!!!

Improvement

Todo...

One way to improve this algorithm is to use character level representation, with these the model could potentialy learn any variant of the words, by deriving the new tokens from the stem word...

On the other hand, we could improbe the Active Learning by usign MNLP, as proposed by [@]. The problem with the Least Confidence sampling is that desproportionaly chooses longer sentences, as it assumes those are harder to annotate...

COMPLETAR!!!!

References

Huang, Zhiheng, Wei Xu, and Kai Yu. 2015. "Bidirectional Lstm-Crf Models for Sequence Tagging." *arXiv Preprint arXiv:1508.01991*.

Shen, Yanyao, Hyokun Yun, Zachary C Lipton, Yakov Kronrod, and Animashree Anandkumar. 2017. "Deep Active Learning for Named Entity Recognition." *arXiv Preprint arXiv:1707.05928*.

SIGNLL. 2002. "Language-Independent Named Entity Recognition (I)." https://www.clips.uantwerpen.be/conll2002/ner/.

Weischedel, Ralph, Martha Palmer, Mitchell Marcus, Eduard Hovy, Sameer Pradhan, Lance Ramshaw, Nianwen Xue, et al. 2013. "Ontonotes Release 5.0 Ldc2013t19." *Linguistic Data Consortium, Philadelphia, PA*.

Young, Tom, Devamanyu Hazarika, Soujanya Poria, and Erik Cambria. 2018. "Recent Trends in Deep Learning Based Natural Language Processing." *Ieee Computational intelligenCe Magazine* 13 (3). IEEE:55–75.