Implementation of entity extraction systems with convolutional neural nets and conditional random fields

Juan Felipe Carrasquilla Alvarez

June 7, 2016

Abstract

This is a brief report about implementations of an entity extraction system using convolutional neural nets (CNN) and conditional random fields (CRF). For the CNN implementation I relied on TensorFlow [1] while for the CRF implementation I used the python-crfsuite [2, 3]. I find that both the CRF and the CNN systems perform well and attain an accuracy of about %98 on a test set based on 20% of the data provided by Maluuba.

1 Brief description of the problem and approaches to its solution

As a part of the interview process at Maluuba, candidates are asked to do programming questions. This problem tests the candidates' ability to build an entity extraction system. The data provided is a file that contains sentences annotated with inside outside beginning (IOB) tags for different entities. The problem consists in building an extraction system that can take a sentence and extract out the entities using two different approaches, one based on deep learning (CNN, recurrent neural nets, etc) and another one based on CRF.

1.1 Deep learning approach

In this subsection I briefly describe my approach to the problem above using a CNN. CNN's are heavily used in computer vision but they can also be applied to natural language processing (NLP) [4]. Instead of image pixels, the input to NLP tasks are words, sentences or documents, which can also often be represented as a matrix. In my approach each row of the matrix corresponds to a word. That is, each row is vector that represents a word. In my solution, these vectors are word embeddings from the provided word2vec file whenever the word under consideration belongs to the provided dictionary. When the word is not present, then I assign a unique random vector and label the word as 'UNK' (similar to what is done in this tutorial [5]). If the word is a digit, then I assign another unique random vector and label the word as 'DIGIT'.

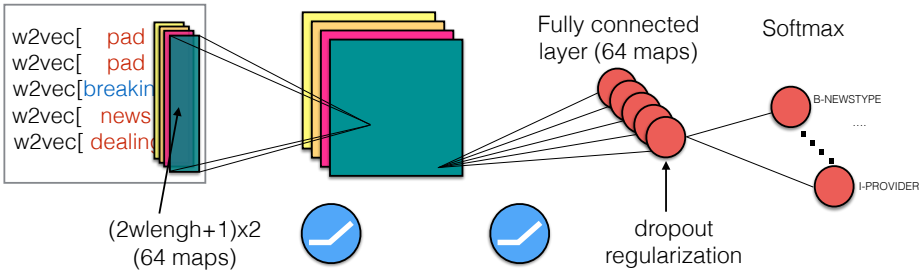

Figure 1: Illustrating the convolutional neural network. The first hidden layer typically convolves 64 $(2wl+1)\times 2$ filters applied to the feature matrix, followed by rectified linear units (ReLu). The outcome is followed by fully-connected layer with 64 units and a softmax output layer with 9 units.

Features

The features that I use in the training and testing for each word are built from the embedding of the word itself and from the embeddings of a few neighboring words within a given sentence preceding and succeeding the word under consideration. I define a variable called wl (short for window length) which means exactly how many preceding and subsequent words are considered in the definition of the feature matrix. If a particular word is at or near the beginning or the end of the sentence, then there may not be enough words preceding or succeeding. In such cases, a pad is introduced in the empty space in the feature matrix with the vector $\vec{\mathrm{P}} = [-1 - 1 \cdots - 1 - 1]$. Thus the dimensions of the feature matrix will be $[(2wl + 1) \times k]$, where k is the dimension of the word embedding ($k = 300$ in the provided word2vec dictionary).

Details of the convolutional neural net

The details of the convolutional neural network, schematically described in Figure 1, is as follows. The input layer is the feature matrix with $(2wl+1)\times k$ values. The first hidden layer typically convolves 64 $(2wl+1)\times 2$ filters horizontally on the feature matrix with a unit stride and no padding, followed by rectified linear unit (ReLu). The final hidden layer is a fully-connected layer with 64 ReLu units, while the output is a softmax layer with 9 outputs corresponding the 9 different word annotations. Here is a dictionary with the labels found in the data: labeldict={'I-SECTION': 0, 'B-SECTION': 1, 'O': 2, 'I-KEYWORDS': 3, 'B-KEYWORDS': 4, 'B-PROVIDER': 5, 'B-NEWSTYPE': 6, 'I-NEWSTYPE': 7, 'I-PROVIDER': 8}. To prevent overfitting, we apply a dropout regularization in the fully-connected layer [6]. This model has been implemented using TensorFlow [1]. Other packages I made use of in this solution include numpy and os.

I typically train the model with a batch size of 100 and for a maximum of 25K iterations with an Adam Optimizer [7].

Test results

I have tested a few different sets hyperparameters in order to see which setting achieves the best performance while keeping the computational resources to a minimum.

Here is a table with accuracies on the single word prediction on a test set for the different hyperparameters I considered. In particular, I considered varying the window size wl and the number of maps in the convolutional layer Nm and number of neurons in the fully connected layer Nfc, with the constraint that $Nm = Nfc$.

window size wl	Nm, Nfc	overall precision
1	32	0.958
1	64	0.965
3	32	0.975
3	64	0.982
3	128	0.983
4	64	0.980

In the attached programs and scripts you will find the parameters of the trained model (file name is model.ckpt) corresponding to the $wl = 3$ and $Nm = Nfc = 64$ whose performance is comparable to the more numerically expensive cases ($wl = 3, 4$ with $Nm = 64, 128$).

1.2 Conditional random fields approach

In this subsection I briefly describe the approach to the problem using a CRF. Imagine that we have a *sequential* data (e.g. images or words in a sentence) and want to label each image/word according to what each image/word means. One way would be to ignore the sequential nature of the data, and build a classifier. By ignoring the sequential aspect, however, you lose a lot of information. One way to potentially increase the power of the classifier is to incorporate the information of nearby images/words in time, and this is precisely what a conditional random field does. Notice that this is very much like the approach we have followed to construct the features in our convolutional neural net approach.

In my implementation of the CRF I have used the python-crfsuite suite and have followed the example described in Ref. [2].

Features

The features I used in this case are closely related to the features used in the CNN case, namely the words preceding and succeeding the word under consideration. More precisely, pseudo code for what the features look like given the sentence stored in the variable sent is presented in Figure 2. In the pseudo code the variable i points to the word under consideration. The features include a bias string, the word itself, a boolean variable that tells whether the word is a digit or not, the preceding word sent[i-1] if there is one, a boolean variable that tells whether the preceding word is a digit or not, and similar for the succeeding word. Also included are strings indicating whether the word is at the beginning (BOS) or at the end of the sentence (EOS).

```
word = sent[i][0]
features = [
    'bias',
    'word=' + word,
    'word.isdigit=%s' % word.isdigit()
]

if i > 0:
    word1 = sent[i-1][0]
    features.extend([
        '-1:word=' + word1,
        '-1:word.isdigit=%s' % word1.isdigit()
    ])
else:
    features.append('BOS')

if i < len(sent)-1:
    word1 = sent[i+1][0]
    features.extend([
        '+1:word=' + word1,
        '+1:word.isdigit=%s' % word1.isdigit()
    ])
else:
    features.append('EOS')
```

Figure 2: Pseudo code illustrating the features used in the CRF implementation.

Details of the CRF

I have used the L-BFGS training algorithm with Elastic Net (L1+ L2) regularization. The regularization parameters that I have used are $c_1 = 0.5$ for the L1 regularization while the $c_2 = 0.0001$ for the L2 regularization. I set the maximum number of iterations to 25K.

Test results

We train the model and perform tests on a data set with 20% of the provided data. Here is a summary of the results

	precision	recall	f1-score
B-KEYWORDS	0.98	0.97	0.97
I-KEYWORDS	0.98	0.99	0.98
B-NEWSTYPE	0.99	0.99	0.99
I-NEWSTYPE	1.00	1.00	1.00
B-PROVIDER	0.93	0.96	0.95
I-PROVIDER	0.97	0.97	0.97
B-SECTION	1.00	0.91	0.95
I-SECTION	0.88	0.96	0.92
0	0.99	0.99	0.99
avg / total	0.98	0.98	0.98

So, the overall accuracy is 98%, similar to the best results obtained with the CNN approach. It is worth mentioning that the setup for the CRS is way simpler than the setup for the CNN and the training is much faster (it takes about 14 seconds to train the CRF model, while the CNN can take up to 2593 seconds in the most numerically expensive setup). All the experiments have been performed on a 2.4 GHz Intel Core i7 with 8 GB 1600 MHz DDR3 memory laptop running OSX.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. `http://tensorflow.org/`.

[2] A python binding to crfsuite. `https://python-crfsuite.readthedocs.io/en/latest/index.html` (visited: 2016-06-06).

[3] Naoaki Okazaki. CRFsuite: a fast implementation of Conditional Random Fields (CRFs), 2007. `http://www.chokkan.org/software/crfsuite/` (visited: 2016-06-06).

[4] Understanding convolutional neural networks for nlp. http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/ (visited: 2016-06-06).

[5] Recurrent neural networks with word embeddings. http://deeplearning.net/tutorial/rnnslu.html (visited: 2016-06-06).

[6] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[7] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2014.