Named Entity Recognition
Prof. Babaali
July 30, 2015

Named Entity Recognition for Persian

Nima Mohammadi <nima.mohammadi@ut.ac.ir>
Masoud Molayem <masoudmolayem@ut.ac.ir>

First, we define the problem and give some background information. Having presented the dataset, then we move on to preprocessing the data and how we present the data to the proposed model whose architecture will be defined.

The task, entitled Named Entity Recognition (NER for short), is to learn a function which maps proper nouns to its type, which is essentially some category, if you may, such as *Person*, *Location*, or *Organization*. For example, given the sentence below,

“Hamid is the director of a trade company called HamidFirm which is located in Tehran.”

an NER system would tag words of interest as

“**Hamid** [Person] is the director of a trade company called **HamidFirm** [Organization] which is located in **Tehran** [Location].”

The algorithm is supposed to make use of both spelling and contextual rules. That is it could just simply look up proper nouns (e.g Tehran) in a lookup table, provided that it somehow knows that what part of speech each word is. Though we can not possibly trust a lookup table to be expanded enough, and the sublime goal is to design a system that can overcome this problem without having an exhaustive lookup table. Basing decisions on contextual rules, the word surroundings in a sentence, is one of the highly contributing factors in how generalizable an NER system is. For instance, you may say that any word following “Mr.” is of type *Person*.

In this report, we describe our work on a neural network-based entity extractor, which identifies some useful entities from a dataset in Persian language.

A challenge in NLP problems that use neural networks, is the presentation of data to the network. We decided to use the famous word2vec algorithm of Google Inc. for data representation. Word2vec is a two-layer neural network which takes a rather big corpus of text and turns the words into numerical vectors of same size. Given enough amount of data, this unsupervised algorithm can make rather surprisingly well assumptions over the meaning of the words. These assumptions, which are solely based on the past appearances of words, can establish the words’ association graph. This have a range of applications ranging from document clustering and sentiment analysis to our problem of interest, that is NER.

Word2vec gives us a vocabulary which associates each word with its corresponding vector. The proximity of words to each other can be measured by their cosine similarity, which is roughly translated to how distant two words are. A perfect right angle represent identity, so “سـاری”, a city in northern Iran, gives distance 1 to itself. In our model, “سـاری” has a cosine distance of 0.84615 to “آمـل” which is another city in the same province. Word2vec is a successful example of a “shallow” learning. There are two distinct methods for learning word2vec, namely, CBOW which predicts the current word w0 given only C and is faster and more suitable for larger corpora, and skip-gram, which predicts words from C given w0 and produces better word vectors for infrequent words.

Word2vec is the name of a family of related algorithms, and to be more exact, we used DBOW (i.e. distributed bag of words) in this study. We used an implementation of word2vec from Gensim software package[1]. The algorithm is relatively very sufficient and we are aiming to have it run on larger dataset later.

To train word2vec, beside the sentences provided in our Persian dataset, we used the latest dump of Persian Wikipedia. The Wikipedia dump is rather enormous, and due to our limited time, we could only use ~150000 sentences, yielding three million words in total. Needless to say that the quality of Persian Wikipedia is not on par with English Wikipedia. For example, visual inspection of the dump, reveals a lot of spam and to our utmost astonishment, unacceptable amount of profanity! Therefore, the reader should consider that a better corpus of text, for example one that is extracted from news feeds, may be a better choice.

The dataset is courtesy of Prof. Bahrani of Sharif University of Technology. It is comprised of 72 files on different subjects ranging from sport to political texts. Other than the ‘zero’ class which denotes the not-entity-named words, there are 6 classes:

Class	# of words
ZERO	146828
B_ORG	3841
I_ORG	5485
B_LOC	4134
I_LOC	1142
B_PERS	3016
I_PERS	1730

The whole dataset is 9876 sentences, consisting 166176 words. 5106 of these sentences include at least one non-zero word.

But what makes neural networks a suitable choice for our problem. One of the encouraging reasons behind using NN for this problem is its extraordinary ability to derive meaning from complex or imprecise data, which is the case in this problem.

Fortunately the dataset provides a list of tags, including part-of-speech taggings and some syntactical ones, for each word. Overall, they amount to 87 kind of tags. To better assess the performance of our proposed neural network, first we provide the result of classification via Support Vector Machines:

Support Vector Machines

Confusion Matrix	ZERO	72641	212	4	65	161	379	25
	B_LOC	322	1749	0	0	27	26	1
	I_LOC	226	226	79	0	27	35	12
	B_ORG	1289	127	0	333	15	160	0
	I_ORG	1472	465	30	15	642	110	3
	B_PERS	152	5	0	1	1	1312	51
	I_PERS	89	1	0	2	0	391	436

Correctly-classified non-zeroes:	**4551**
Incorrectly-classified non-zeroes:	**7128**
Accuracy:	**92.6%**

The accuracy in this problem gives a false sense of satisfaction. Most of the words are in fact of class ZERO and therefore the classifiers learns to associate most of samples also with class ZERO, hence we decided to include the number of correctly/incorrectly classified non-zero samples.

For feature vector, we used the word2vec-generated 100-length vector of each word. As our classifier is a sequential deep neural network (i.e. not recurrent), we decided to also put the preceding and following words of each word into the feature vector. Notice that this is a common practice to have a window of some length. We also tested with window of length five (two preceding and two following word), but failed to observe a noticeable improvement. Moreover, we considered the so-called border effect, that is, the feature window not being well-defined for words beginning and ending the sentence[2]. To circumvent this problem, the sentence is augmented with null vectors (i.e. vectors of same length filled by zeroes). This could also give the model a sense of "start" and "stop" symbolism in the sequence of samples. It's common in NER for Latin languages to have features as indication of capitalization or whether the spelling contains any characters other than upper or lower case letters. Obviously, these features are very relevant for Latin corpora, this does not apply to languages written in Arabic script, such as Persian.

As for the proposed method, we have a four-layer "deep" neural network [5]. It has 387 input neurons in the input layer, that is 300 for the three adjacent words, plus 87 neurons corresponding to the tags of the word in question. The two next layers each have 200 and 80 neurons, respectively. These numbers were found empirically by trial and error. The output layers consist of seven Softmax neurons. The other layers have hyperbolic tangent activation function. The initial weights are drawn randomly from a uniform distribution function.

We also set the dropout fraction of the two middle layers to 0.75 and 0.3, respectively. Deep neural networks contain multiple non-linear hidden layers which makes them very expressive models, capable of learning complicated relationships. However, overfitting is a serious problem in such networks. Dropout sets a fraction of units to zero at each update during the training time, which is considered a rather straightforward technique to prevent overfitting, to some extent. This random dropping of units also makes training of DNNs faster, which can be highly appreciated considering how relatively slow they are. A general formalism of this concept can be found in [3].

For implementation of our neural network we used Keras library[4], which makes use of Theano for fast tensor manipulation and CUDA-based GPU acceleration. The model is trained on a server of UT CHPC which has a Tesla K40c GPU.

We use Minibatch SGD for our model. As our dataset is relatively enormous, setting up mini batch size helps to minimize the number of updates. This can significantly increase the speed of training phase. The model optimization is done with Stochastic Gradient Descent (SGD). The use of SGD is motivated by the high cost of running back propagation over the full training set. SGD can overcome this cost and still lead to fast convergence. Generally each parameter update in SGD is calculated considering a few training examples or a mini-batch as opposed to a single example. In SGD the learning rate *a* is typically much smaller than a corresponding learning rate in batch gradient descent because there is much more variance in the update. Moreover, when we use momentum, *a* may need to be smaller since the magnitude of the gradient will be larger. In our model we set learning rate and momentum to 0.1 and 0.97, respectively. By smoothing the weight updates across iterations, momentum tends to make deep learning with SGD both stabler and faster.

The unbalanced number of samples from class ZERO compared to other classes is an inherent problem of NER and many other NLP problems. This problem which can also be found under other ML problem is called class-imbalanced problem. The usual technique to circumvent this problem is undersampling or oversampling. Our train and test datasets are built by alternatively putting sentences in one of them, effectively putting roughly half of the dataset for testing phase. Then we specified the sentences of train set that contain no non-zero-class words. We progressively decrease these sentences and observed increase in performance. At last, we omitted all those sentences altogether in our train set. We may be better off by further decreasing the number of ZERO samples, though we have to omit some of the ZERO words in sentences that contain non-zero words. This may harm the contextual integrity of the text as the neighboring words whose vector we put in the feature space are no longer valid.

To asses the effect of including tags in our feature vector, we run our model twice, with and without the tags, with each having 300 and 387 input neurons, respectively. The tables below show how they performed:

DEEP NN (w/o tags)								
Confusion Matrix	ZERO	71755	379	94	396	505	141	226
	B_LOC	418	1597	28	0	62	6	14
	I_LOC	161	94	268	10	52	8	12
	B_ORG	897	123	2	862	31	4	5
	I_ORG	952	276	56	77	1349	13	14
	B_PERS	698	7	1	1	9	702	104
	I_PERS	209	2	1	0	1	31	675

Correctly-classified non-zeroes:	**5453**
Incorrectly-classified non-zeroes:	**4379**
Accuracy:	**92.6%**

Although the accuracy of DNN without tags is on par with SVM, you can see how the non-zero classes are better classified in DNN. The reason why the accuracy has not increased is due to the number of ZERO samples that are falsely reported as non-zero classes. SVM classified 7128 samples as non-zero while DNN without tags classified 9679 of them as so.

Then we include the tags, which are supposedly going to improve the performance, as giving more information features.

As the final table depicts, this model outperforms the previous models. While the accuracy is higher, the higher number of correctly classified non-zeroes and lower number of incorrectly classified non-zeroes firmly confirm the effectiveness of tags.

DEEP NN (with tags)

Confusion Matrix	ZERO	72153	200	64	475	317	216	62
	B_LOC	176	1759	83	5	81	15	6
	I_LOC	141	74	348	2	12	11	17
	B_ORG	580	111	9	1109	35	69	11
	I_ORG	943	232	121	77	1315	33	16
	B_PERS	109	8	4	11	14	1233	143
	I_PERS	69	0	3	9	4	47	787

Correctly-classified non-zeroes:	**6561**
Incorrectly-classified non-zeroes:	**3281**
Accuracy:	**94.4%**

References:

[1] Řehůřek, R., and P. Sojka. “Gensim–Python Framework for Vector Space Modelling.” NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic (2011).

[2] Collobert, Ronan, et al. “Natural language processing (almost) from scratch.” The Journal of Machine Learning Research 12 (2011): 2493-2537.

[3] Baldi, Pierre, and Peter J. Sadowski. “Understanding dropout.” Advances in Neural Information Processing Systems. 2013.

[4] F. Chollet. “Keras: Theano-based Deep Learning library”. Code: https://github.com/fchollet . Documentation: http://keras.io/

[5] Collobert, Ronan, and Jason Weston. "A unified architecture for natural language processing: Deep neural networks with multitask learning." Proceedings of the 25th international conference on Machine learning. ACM, 2008.