Boosting Neural POS Tagger for Farsi Using Morphological Information

PEYMAN PASSBAN, QUN LIU, and ANDY WAY, ADAPT Centre,
School of Computing, Dublin City University, Ireland

Farsi (Persian) is a low-resource language that suffers from the data sparsity problem and a lack of efficient processing tools. Due to their broad application in natural language processing tasks, part-of-speech (POS) taggers are one of those important tools that should be considered in this respect. Despite recent work on Farsi tagging, there is still room for improvement. The best reported accuracy so far is 96%, which in special cases can rise to 96.9%. The main problem with existing taggers is their inefficiency in coping with out-of-vocabulary (OOV) words. Addressing both problems of accuracy and OOV words, we developed a neural network-based POS tagger (NPT) that performs efficiently on Farsi. Despite using less data, NPT provides better results in comparison to state-of-the-art systems. Our proposed tagger performs with an accuracy of 97.4%, with performance highly influenced by morphological features. We carry out a shallow morphological analysis and show considerable improvement over the baseline configuration.

CCS Concepts: • **Computing methodologies → Natural language processing**; **Neural networks**; *Artificial intelligence;*

Additional Key Words and Phrases: POS tagging, Farsi, morphological analysis

ACM Reference Format:
Peyman Passban, Qun Liu, and Andy Way. 2016. Boosting neural POS tagger for Farsi using morphological information. ACM Trans. Asian Low-Resour. Lang. Inf. Process. 16, 1, Article 4 (July 2016), 15 pages.
DOI: http://dx.doi.org/10.1145/2934676

1. INTRODUCTION

Part-of-speech (POS) tagging[1] can be viewed as a crucial step for many natural language processing (NLP) tasks. Tags are frequently used in many applications (Machine Translation, Information Retrieval, Semantic Parsing, etc.), so having a precise and robust tagger is essential. For some languages, such as English and German, tagging accuracy is about 98%, but tagging is still not a solved problem [Giesbrecht and Evert 2009]. Despite the existence of many solutions, there remain several unanswered questions. Manning [2011] tried to address some of those challenges. Size of training data, tagging mechanism, language type, and even implementation method are all parameters that can affect a tagger's performance.

HunPos [Halácsy et al. 2007] is a reimplementation of the TnT tagger [Brants 2000] that works more precisely and quickly just because of a more efficient implementation.

[1]Wherever we use tagging/tagger/tag in this article, it refers to POS tagging/tagger/tag.

This research was supported by Science Foundation Ireland through the CNGL Programme (Grant 12/CE/I2267) in the ADAPT Centre (http://www.adaptcentre.ie) at Dublin City University.
Authors' addresses: P. Passban, Q. Liu, and A. Way, ADAPT Centre, School of Computing, Dublin City University, Dublin, Ireland; emails: {ppassban, qliu, away}@computing.dcu.ie.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2016 ACM 2375-4699/2016/07-ART4 $15.00
DOI: http://dx.doi.org/10.1145/2934676

It tags English with an accuracy of 96.58%, whereas it works with an accuracy of 98.24% for Hungarian, which is a morphologically rich language. Mohseni and Minaei-Bidgoli [2010] used a dataset of 10M words to train a tagger, and although they benefited from morphological features of words to improve the tagging accuracy of Farsi, their results are not promising (detailed information in the next sections). A dataset of 10M words is large enough to provide quite precise results, but efficient use of that data is another issue altogether. In our case, we also use the same dataset as the state-of-the-art tagger, but with less data we obtain better results.

In this research, we work with neural networks (NNs), which usually depend on a set of parameters that may be very sensitive to implementation details. To obtain accurate results, the tagging parameters should be considered carefully, and the NN should be designed to be compatible with the tagging task. We discuss the impact of various factors in Section 4 and show how they can affect the whole pipeline.

Applying NN to the problem of POS tagging and to any classification task in general is not a new research topic. Many early works such as the POS tagger of Schmid [1994] and some recent works like Collobert et al. [2011] used NNs, but recently this field has become much more popular. In 2006, a breakthrough occurred when Hinton et al. [2006] introduced a new training algorithm for large-scale/deep NNs and attracted attention to NNs. Using those types of algorithms along with newly developed coding platforms such as Theano[2] [Bergstra et al. 2011], Torch[3] [Collobert et al. 2012], and Caffe[4] [Jia et al. 2014] made the construction and implementation of NNs much easier.

One work that is quite similar to ours was recently reported by Fonseca et al. [2015]. They used the NN as a POS tagger and obtained very good results for Portuguese. Prior to their work, the best reported accuracy for Portuguese was about 80%, but with their proposed NN-based tagger, they achieved an accuracy of 97%. With regard to Farsi, tagging performance was not very good prior to 2004. All efforts were made to develop rule-based taggers at that time, which were unsuccessful [Megerdoomian 2004]. In 2004, the BijanKhan corpus, a collection of 2.6M manually tagged words [Oroumchian et al. 2006], was developed. It persuaded researchers to apply data-driven models to Farsi, but thereafter a new problem appeared. The data problem was solved to a large extent, yet it was (and still is) difficult to fine tune existing taggers for Farsi owing to its rich morphology and inconsistent orthography. To demonstrate this difficulty, only Raja et al. [2007] and Seraji [2011] could achieve good results. They used TnT and PerTag, obtaining an accuracy of 96.64% and 96%, respectively. Accordingly, data representation is pivotal in Farsi tagging. Data should be normalized before tagging and morphological decomposition can help to boost tagging performance. Shamsfard et al. [2009] addressed these types of challenges with Farsi text normalization.

In this article, we describe a multilayer feed-forward perceptron (MLP), trained and used for the purpose of tagging. It provides an internal representation of input data by passing it through hidden layers and generates output by estimating the most probable tag class for the target word (the word that is going to be tagged). Our main contribution in this work is introducing an NN-based tagger that (a) is robust to out-of-vocabulary (OOV) words and tags them better than previous taggers, (b) provides better performance in comparison to the state-of-the-art tagger, and (c) efficiently uses data and benefits from morphological features.

The structure of the article is as follows. Section 2 discusses the training data and challenges with data preparation. We performed shallow morphological analysis to

[2]http://deeplearning.net/software/theano/.
[3]http://torch.ch/.
[4]http://caffe.berkeleyvision.org/.

boost the tagging accuracy, which is also discussed in Section 2. Section 3 explains our network. Section 4 includes experimental results. Section 5 briefly reviews other models together with a comparison of our tagger, whereas Section 6 concludes the article along with some avenues for future work.

2. FARSI LANGUAGE AND THE CORPUS

In terms of orthography and morphology, Farsi is one the most complicated languages for any processing purposes. We briefly address some of the main problems with this language, which can easily mislead any POS tagger. In addition to common problems, such as ambiguity and disjoint constituents, Farsi has its own challenges. Light verbs and compound verbs are quite common in this language. Farsi is written with Perso-Arabic scripts, which by nature are problematic for coding purposes and automatic text processing. Unlike other languages, some Farsi words include interword zero-width nonjoiner spaces or semispaces. Semispaces are nonprinting characters and usually are incorrectly written as regular space characters that may generate quite different results (U+0020 and U+200c are the Unicode values for space and semispace, respectively). As an example, the correct form of the word *greedy* in Farsi[5] is *āstyn•drāz*, with a semispace character (between *n* and *d* sounds). Clearly, the tag for this constituent is an adjective. If it is written with a space as in *styn○drāz*, it means “long sleeve,” which is a completely different meaning. When the semispace is wrongly substituted/written with a regular space character, the constituent is decomposed into two segments of *āstyn* (“sleeve”) and *drāz* (“long”), whose tags are a noun and an adjective in this case. This is a challenging issue in Farsi tagging.

Another problem is the presence of multiple writing forms for some characters. Farsi characters like *y* and *k* can be written in several forms. The diacritic problem is another issue. Words can appear both with and without diacritics. Diacritics are always pronounced but not written in most cases. This can be confusing for automatic tools like POS taggers that work at the word level, where multiple and inconsistent forms of words are encountered. Problems with the Farsi writing system are not limited to these cases only, but the cases mentioned are the most frequent. To solve these types of problems, we use an in-house Farsi text normalizer to performs tokenization. Using the DIN transliteration standard, it encodes Farsi letters to their corresponding Latin forms. The Latin form is easier for automatic text processing purposes. Then the normalizer corrects incorrect space usage and substitutes any instances with semispace forms.

Farsi is a morphologically rich language (for more details on Farsi morphology, see Perry and Kaye [2007]), and therefore is possible to make up different words by changing affixes. As an example, the word *āmd* (“(s)he came”) is a verb. If it comes with prefix *dr*, the new word is *dr+āmd*, which means “salary” with the *noun* tag, and if it comes with the prefix *kār*, the word is *kār+āmd*, meaning “efficient” with an *adjective* tag. Both derivational and inflectional affixes are quite common in Farsi, and as can be seen, there is valuable information that morphology provides for processing purposes. For this reason, we decided to perform some morphological analysis.

2.1. Shallow Morphological Analyzer

Unfortunately, there is no reliable and publicly available morphological analyzer for Farsi. Accordingly, we performed shallow morphological analysis. We developed an algorithm to separate the stem of each word from its affixes. Our analyzer works on

[5]We used the DIN transliteration standard to show the Farsi alphabets. Semispace and space are shown with the • and ○ characters, respectively; see http://en.wikipedia.org/wiki/Persian_alphabet.

top of a stemmer/lemmatizer. By use of Perstem,[6] a free Farsi stemmer, first words are stemmed and then we try to extract affixes and variations according to stems. By *affixes*, we mean any prefixes and suffixes that appear before and after the stem, and by *variations*, we mean any differences between the word after removing affixes and its corresponding stem. In some cases, after removing affixes, the remaining constituents are different from the stem form that we call a variation. To provide further clarification, we give an example in English, although as mentioned, we applied this method to Farsi. The stem for the word *prestudies* is *study*, where *pre* and *es* are the prefix and suffix, respectively. What remains after elimination of the affixes is *studi*, which is different from the stem form *study*. We call the difference between the word and its stem a variation, and based on the definition in this example, the variation is $y \rightarrow i$. What the shallow morphological analyzer generates as its output is a symbolic vector:[7]

$$[\textbf{prefix}: \textit{pre}, \textbf{variation}: \textit{stem} - y + i, \textbf{suffix}: \textit{es}].$$

All of these features (prefixes, suffixes, and variations) cause a considerable boost in tagging, and we use them all in our tagger. To implement the shallow morphological analyzer. the *Perstem* stemmer is used. After extracting the stem, we try to find a substring in the input word that best matches the stem. By the *best-matching substring,* we mean a portion of the string that shares the maximum number of characters with the stem. What follows that substring is considered the *suffix*, and what appears before is separated as the *prefix*. Then any variation is found. To extract the variation, we fine tuned the edit-distance algorithm [Masek and Paterson 1980] using three fundamental operations of delete (deleting the last character of a given string), shift (shifting one character to the left),[8] and insert (inserting a specified character to the end of a given string) to reach the substring from the stem. For the aforementioned example, the best substring that matches the stem is *studi*, as it has the same length as the stem and covers all letters of it. Applying the sequence of [$\textit{delete}()$, $\textit{insert}(i)$] operations, we can obtain *studi* from *study*.

Clearly, this type of processing would not be precise. In the last example, from a linguistic point of view, *s* should be the suffix (not *es*). In a case where the difference between substring and the stem is considerable, the algorithm may fail to find the correct variation. Furthermore, many other problems may occur. Because of these challenges, we call this process *shallow analysis*. As far as Farsi is concerned, it should be mentioned that most of the affixes are inflectional, and also in almost all cases, the stem is the same as the subword after removing affixes. Accordingly, this type of processing can provide us with useful and approximately correct information. In Section 4, we will see that this type of information enhances tagging accuracy quite dramatically.

2.2. Training Corpus

There are only three tagged corpora for training any data-driven POS tagger in Farsi. The Bijankhan corpus[9] is a freely available corpus of 2.6M manually tagged words and is used in most research work. The Bijankhan corpus has several problems in its tagging scheme. It uses a very fine-grain tagset of 550 tags. Obviously, this makes the tagging task difficult, but fortunately there are some smaller tagsets for the Bijankhan corpus and some instructions on how to reduce the tagset size. In our experiments, we

[6]http://sourceforge.net/projects/perstem/.
[7]This vector is different from numerical feature vectors of NN, so we call this a *symbolic* vector.
[8]The Farsi writing system is left to right, and this command shifts characters to the left-hand side.
[9]https://en.wikipedia.org/wiki/Bijankhan_Corpus.

used a standard tagset for Bijankhan with 40 different tags, which was proposed by the data development team.

UPC is another Farsi corpus developed by Seraji et al. [2012] and is a modified version of Bijankhan with additional sentences including 2.7M words annotated with 32 POS tags.[10] Another corpus that includes 10M tagged words is the Peykareh corpus [Bijankhan et al. 2011], which unfortunately is not publicly available. In Farsi NLP tasks, the Bijankhan corpus is frequently used, and to make our work comparable, we also used this dataset.

3. NETWORK ARCHITECTURE

For the purposes of time and structural complexity, we tried to design a simple architecture for our network. There are many neural taggers [Prezortiz and Forcada 2001; Zheng et al. 2013] that use deep architectures or complex models like recurrent networks; however, in our experiments, we used an MLP. As a general-purpose function approximator, theoretically MLP is able to estimate any type of function or equivalently able to find any type of mapping [Hornik et al. 1989]. Tagging obviously is a classification task in which an input vector is mapped to a specific class. We used MLP as our classifier. Our MLP architecture includes three main modules, which we discuss next.

First is the representation module. Words are symbolic constituents that should be represented numerically and efficiently. This is the most important parameter that directly affects the network performance. There are several ways of representing words. For this purpose, we used *Word2Vec* [Mikolov et al. 2013] and *GloVe* [Pennington et al. 2014]. We first convert our vocabulary into vectors. The dimension of word vectors also has a considerable effect on the final accuracy.

The second module includes hidden layers. In our experiments, we used a different number of hidden layers with various sizes. The number of hidden layers and their size help us to control the accuracy and the convergence speed. In our hidden layers, we used *tanh* units. Nonlinearity is a key feature in NNs, and there are several nonlinear functions (*sigmoid*, *hardtanh*, *tanh*, *qubic*, etc.), of which the most compatible nonlinear unit to our architecture was *tanh*. The value of the *tanh* function for any given input a is computed as in (1):

$$tanh_i^{(h)}(a) = \frac{e^a - e^{-a}}{e^a + e^{-a}}. \tag{1}$$

Equation (1) describes the behavior of the *i-th tanh*(.) unit in the hidden layer h. In an NN, what is taken by a hidden layer is a vector of $W^{(h)}\mathbf{x} + b_i^{(h)}$, where W is a weight matrix, b is the bias, and $\mathbf{x}$ indicates the input vector from the previous/input layer. Figure 1 illustrates the values and computations.

The third module of our network is the prediction layer. The output of each hidden layer is passed to each successive hidden layer, and finally the last one passes its data to the prediction layer, which is a set of *Softmax* units. We calculate the probability of an input vector $\mathbf{x}$, a sample of the dataset $\mathbf{D}$ is a member of class $\boldsymbol{i}$, a value of stochastic variable $\mathbf{Y}$, at the final stage. We used *Softmax* because we are calculating probabilities, and it ensures that we always have a value between zero and one as an output. Based on the definition in Equation (2),

$$Softmax(\alpha)_i = \frac{e^{\alpha_i}}{\sum_j e^{\alpha_j}}, \tag{2}$$

[10]http://stp.lingfil.uu.se/∼mojgan/UPC.html

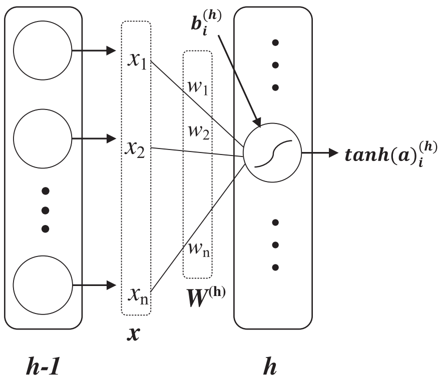

Fig. 1. $tanh(.)$ function illustration.

and what the *i-th* unit of the prediction layer calculates is shown in Equation (3),

$$P(Y = i|x, W, b) = Softmax_i(Wx + b), \tag{3}$$

where b again is the bias value for the *i-th* unit, x is a vector produced by the *tanh* units of the last hidden layer, and W is a matrix of corresponding weights between the last hidden layer and the prediction layer. The network prediction for the tag of input vector is the class whose probability is maximum, as in Equation (4):

$$prediction = \arg\max_i P(Y = i|x, W, b). \tag{4}$$

To train the network and optimize the parameter set $\theta = [W, b]$, we used stochastic gradient descent with mini-batches where loss function is defined by negative log likelihood. The loss function is shown in Equation (5):

$$\ell(\theta, D) = -\sum_{i=0}^{|D|} \log P(Y = y^{(i)}|x^{(i)}, \theta) + \eta R. \tag{5}$$

To prevent overfitting, we use the $L2$ regulizer, which is referred to by R in (5). As can be seen, the network has many hyperparameters, which are addressed in more detail in the next section. The network was coded in Python using Theano [Bergstra et al. 2011].

4. EXPERIMENTAL RESULTS

In this section, we first report the performance of our tagger and then address the impact of different factors, such as network parameters and word representations. Several taggers have been developed for Farsi. Raja et al. [2007] trained several open-source taggers and compared them. The best performance belongs to TnT (a Markov model), with an accuracy of 96.86% on the Bijankhan corpus with 80.09% for unseen words that occurred 1.94% of the time. They used 85% of the corpus for training (≈2,200,000 tokens) and the remaining 15% as a test set (400,000). There is another model trained using HunPos on the Bijankhan corpus [Seraji 2011]. When the dataset is divided into three parts of training, validation and test sets including the 80%, 10%, and 10% of the Bijankhan corpus, respectively, the accuracy of the tagger is 96%; however, when the training set is enlarged by adding the validation set, the accuracy rises to 96.9%. In the last case, the training set includes 95% of the corpus and covers almost all of the test set, and thus it is natural to have a higher accuracy. However,

Table I. NPT Performance

	Training	Test
Number of tokens	2M	200,000
Number of unique tokens	60,144	14,700
Type-token ratio	0.03	0.07
OOV percentage	5.4%	
Seen word accuracy	97.85%	
Unseen word accuracy	89.01%	
Overall accuracy	97.376%	

Table II. NPT Accuracy Over Tagset

Tag	Description	Percentage (%)	Accuracy	Tag	Description	Percentage (%)	Accuracy
ADJ_CM	Adj, Comparative	0.0028	97.90	MQUA	QuantifierModifier	0.0001	97.00
ADJ_IN	Past Participle	0.0104	48.49	MS	Math. Symbol	0.0001	97.16
ADJ_OR	Adj, Ordinal	0.0025	97.90	NN	Number	2.07e-05	96.41
ADJ_SI	Adj, Simple	0.0889	96.99	N_PL	Noun, Plural	0.06175	97.72
ADJ_SU	Adj, Superlative	0.0028	97.48	N_SING	Noun, Sing.	0.3723	97.86
ADV	Adv, General	0.0005	92.31	OH	Oh Interjection	0.0001	97.39
ADV_EX	Adv, Exemplar	0.0012	97.90	P	Preposition	0.1231	99.13
ADV_I	Adv, Question	0.0008	86.44	PP	PrepositionalPhrase	0.0003	97.29
ADV_NG	Adv, Negation	0.0006	89.20	PRO	Pronoun	0.0238	98.03
ADV_NI	Adv, NotQuestion	0.0084	97.13	PS	Psedo-Sentence	0.0001	97.90
ADV_TM	Adv, Time	0.0032	91.41	QUA	Quantifier	0.0059	97.90
AR	Arabic Word	0.0013	96.00	SPEC	Specifier	0.0014	100
CON	Conjunction	0.0809	98.91	V_AUX	V, Auxiliary	0.0061	98.23
DEF	Default	7.39e-05	98.99	V_IMP	V, Imperative	0.0004	88.49
DELM	Delimiter	0.0987	99.89	V_PA	V, Past Tense	0.0310	95.95
DET	Determiner	0.0176	93.75	V_PRE	V, Predicative	0.0163	97.54
IF	Conditional	0.0012	96.84	V_PRS	V, Present Tense	0.0199	93.35
INT	Interjection	4.35e-05	85.57	V_SUB	V, Subjunctive	0.0130	95.64
MORP	Morpheme	0.0011	97.91				

HunPos is an open-source tagger,[11] and as there is no detailed information about the experimental setup (an exact number of tokens has been used for training or the percentage of OOV words), we were not able to repeat that work.

Since the highest number was reported from HunPos, we consider that setting as the state of the art and use it as a comparison. The performance of our tagger is summarized in Table I, with detailed information provided in Table II. For the given training and test sets in Table I, HunPos obtained 96%, where a network-based POS tagger (NPT) works with an accuracy of 97.37%. For HunPos, unfortunately unseen word accuracy was not reported, but the unseen accuracy of TnT on the same data set for the 2.09% of OOV words is 77.69% [Raja et al. 2007], whereas we obtained 89.01% accuracy for the 5.4% of OOV words.

In our experiments, we selected the first 2M words of the corpus as our training data, the next 200,000 for the validation set, and the next 200,000 for the test set. We also neglected the last 200,000 of the corpus, as it did not change the experimental results when added to the training set. To make our experiments more reliable, we also performed 10-fold cross validation on the Bijankhan corpus, on which the highest result obtained was 97.37%, the lowest was 96.95%, and average accuracy was 97.16%. In Table II, tags are in the first column and their brief explanations are in the second

[11]https://code.google.com/p/hunpos/downloads/list.

column.[12] The third column shows the probability of each tag (with only four floating points) in the corpus, and the fourth column shows the related accuracy.

4.1. Impact of Network Parameters

Our tagger basically is an NN, and the parameter setting is the most challenging part of working with NNs. In this section, we discuss network hyperparameters and the process of word representation. We empirically discovered that the best architecture for our tagger is a network with four layers. One input layer simply reflects word vectors, two hidden layers of *tanh* units provide internal representations, and one output layer of *Softmax* units performs the prediction. A smaller network was not able to provide precise results, and a network with more than two hidden layers did not act very differently and only delayed the convergence time.

The size of the layers is also a very effective feature. The size of the first layer is almost fixed, as it depends on the dimensionality of the input vectors, and the size of the last layer has the same condition where its size is controlled by the number of output classes. Hidden layers with a small number of units cannot represent the data efficiently, and very large layers exponentially increase the computational complexity and provide no gain. The number of classes in our experiments is the number of tags in Table II, namely 37. According to experiments, when the size of the second hidden layer is set to 100, the network provides better results. The size of 50 caused some errors and was not representative enough, whereas a layer with more than 200 units did not provide any further value. In the range of 50 to 200, we experimentally established 100 to be our optimal number. For the first hidden layer, the best condition is when the layer size is set according to Equation (6):

$$|input| \times 1.2 < |h^{(1)}| < |input| \times 2. \tag{6}$$

To initialize the network parameters, bias values (b) were set to zero. We know that weight values at each layer should be sampled uniformly from a symmetric interval. By use of results from Glorot and Bengio [2010], the best interval for *tanh* units is shown in Equation (7):

$$\left[-\sqrt{\frac{6}{U_{in}+U_{out}}}, \sqrt{\frac{6}{U_{in}+U_{out}}}\right], \tag{7}$$

where U_{in} and and U_{out} are the number of *tanh* units in the *(i-1)-th* and *(i)-th* layers, respectively. We initialized the weight matrix based on this criterion. The learning rate was set to 0.01, and the regulizer coefficient is $\eta = 0.001$. As mentioned previously, SGD with mini-batches was used to train the network, where the optimal batch size was between 500 and 1,000. All experiments are reported with the batch size of 1,000.

So far, we have explained the architecture and parameter setting of the network, but we have not yet described what the network takes as its input. The input is a vector of training features. In our setting, we used a vector of words and morphological features. A corresponding input vector for each target word is f_i: [w_i^{-2}, w_i^{-1}, w_i, $mf(w_i)$]. Given the vector f_i that is a concatenation of four feature vectors, the goal is to find the most probable tag for the target word w_i, where it is followed by the two prior words w_i^{-2} and w_i^{-1}. $mf(w_i)$ is a fixed size vector of morphological features of the word w_i. Words and morphological information are symbolic concepts that need to be transferred into numerical/vector forms.

[12]More information about the tagset is available at http://ece.ut.ac.ir/dbrg/bijankhan.

To numerically represent words, we trained *GloVe* and *Word2Vec* on our training corpus. We tried to train both of them in the same training conditions. We set the context window size to 10 and the initial learning rate to 0.025. We ran the models for 10 iterations. In *Word2Vec*, we used the continuous bag-of-words (CBOW) configuration. In our case, CBOW uses the mean of context-word vectors. In our setting, if we are confronted with an OOV word, we find the closest match for it and use the match's embedding. The closest match is found based on the edit-distance criterion. The closest match is a word that has the minimum distance to the OOV word.

To generate vectors/embeddings of morphological features, we performed almost the same experiment. As mentioned in Section 2, the shallow morphological analyzer generates a symbolic vector for each word that reflects its morphological features and has considerable impact in detecting the word's tag. Each symbolic vector can be assumed as a morphological annotation of the corresponding word. For the Bijankhan corpus, our analyzer generated 8,818 unique morphological annotations (with 153 unique prefixes, 8,100 unique suffixes, and 6,554 unique variations). To obtain morphological embeddings, we made a corpus of morphological annotations by replacing each word with its annotation and trained *Word2Vec*/*GloVe* on it. The size of vectors directly influences the final results.

The key point in the proposed architecture is the way we treat word and morphology embeddings. During training, we keep word embeddings unchanged and only update morphology embeddings. This means that morphology embeddings are part of the network parameters that are updated at each iteration. They were initialized by *Word2Vec*/*GloVe* and tuned during training. The feature set in the input layer has four subvectors (the target word itself, two words before the target word, and the morphological annotation of the target word). To show the impact of each of them, we designed different experiments, the results of which are illustrated in the following figures. Based on our experiments, the selected four features are the most impactful elements in Farsi tagging. We tried other elements, such as w_i^{+1} and w_i^{+2}, or used stems and lemmas instead of words (surface forms) but did not obtain better results. To train *Word2Vec*/*GloVe*, we used the Bijankhan corpus. We explored different results with the embedding sizes of 10, 50, 100, and 200 for words. For morphology embeddings, the dimension was always set to 50. Figures 2 and 3 show the experimental results.

As Figure 2 shows, the size of the embeddings directly affects the network accuracy. As the dimensionality increases, the accuracy increases correspondingly. If the word itself is used as a feature, tagging results are not very high; however, as the other features are added, the accuracy increases. The most effective feature apart from the word itself is $mf(w_i)$, the morphology vector of w_i. The second most effective feature is the adjacent word just before the target word. The figures show the performance results for 10D, 50D, and 200D word vectors, as well as the comparison between *Word2Vec* and *GloVe*. To exhibit another comparison of *Word2Vec* and *Glove* and to better show the impact of the word dimensionality, we depict the learning curve and its fluctuations in the last 100 iterations in Figure 3. The results are based on all four feature sets. The results confirm that *Word2Vec* works better than *GloVe* on our corpus and that word embeddings with dimensionality of 200D are more efficient. The runtime of our system for tagging 200,000 words with 200D vectors is 244 seconds, which is an acceptable time for a tagger, considering that most of the tagging is a once-off operation. HunPos tags this much data in 8 seconds, much faster than us, but as previously explained, its results are considerably worse.

We performed another experiment to show the impact of each morphological subunit (prefix, suffix, and variation) separately. As Figure 2 shows, the best configuration is when the target word w is accompanied by w^{-2}, w^{-1}, and $mf(w)$. We used different

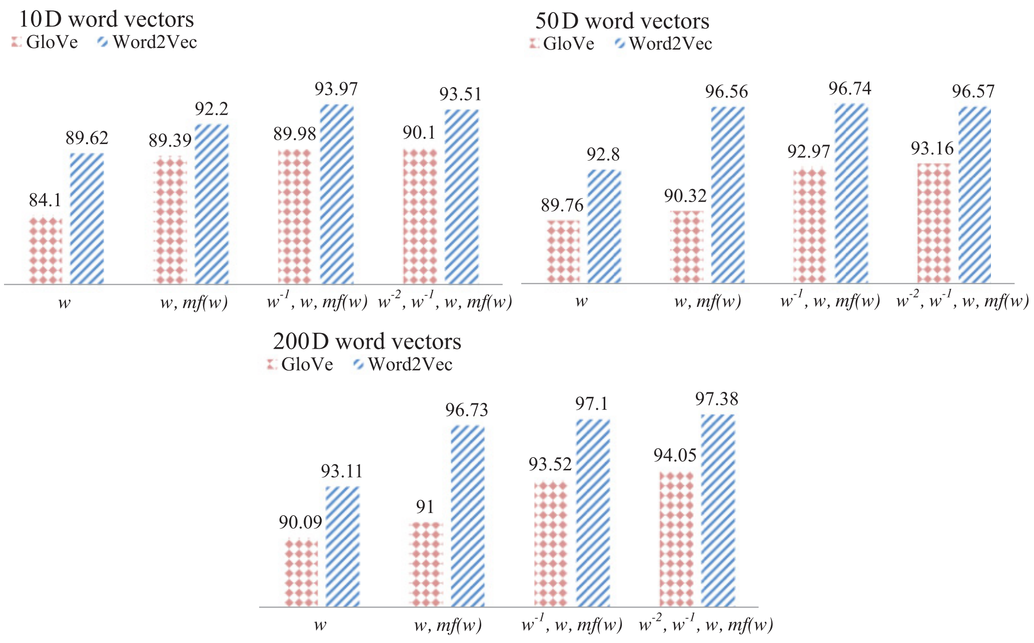

Fig. 2. NPT accuracy with 10D, 50D, and 200D word embeddings.

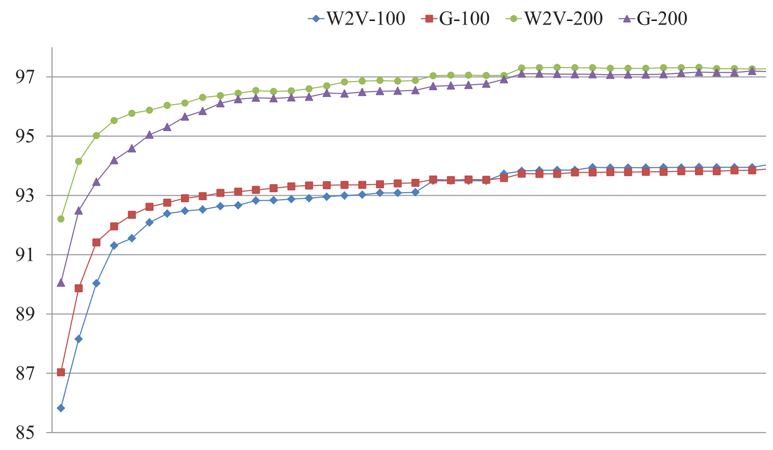

Fig. 3. GloVe versus Word2Vec.

feature sets based on the $mf(w)$ vector. Table III shows the impact of each morphological unit in the tagging process.

5. RELATED WORK AND DISCUSSION

Similar to Tseng et al. [2005], we showed that morphological features help in tagging of OOV words and tagging in general. We performed tagging using an NN. Neural POS tagging and neural NLP in general have become a very hot research topic. Neural taggers generate comparable results to state-of-the-art systems and even surpass them.

Table III. Accuracy of NPT With Different Morphological Features

Feature Set	Accuracy	Improvement
w^{-2}, w^{-1}, w	94.65	—
w^{-2}, w^{-1}, w, *prefix*	95.16	+0.51
w^{-2}, w^{-1}, w, *suffix*	96.48	+1.83
w^{-2}, w^{-1}, w, *variation*	95.79	+1.14
w^{-2}, w^{-1}, w, *prefix+suffix*	96.61	+1.96
w^{-2}, w^{-1}, w, *prefix+suffix+variation*	97.38	+2.73

Table IV. Accuracy of Different POS Taggers on the Bijankhan Corpus

System	Knowns	Unknowns	Known Accuracy	Unknowns Accuracy	Overall
Markov	98.06	1.94	97.18	80.09	96.86
MbT	98.06	1.94	96.72	91.80	96.62
MLE	98.06	1.94	96.78	16	94.91
HunPos	—	—	—	—	96.90
NPT	94.60	5.40	97.85	89.01	**97.37**

The works by Ma et al. [2014], Wang et al. [2015], Zennaki et al. [2015], and Jagadeesh et al. [2016] are some of the most recent in this field.

We use the Bijankhan corpus in our experiments. The corpus is a standard collection, and other similar approaches use it to evaluate their systems. Table IV summarizes reported numbers on the Bijankhan corpus.

The first column in the table indicates the system name. The first system is based on a Markov model (TnT). The second one is a memory-based tagger, and the third system uses maximum likelihood estimation [Raja et al. 2007]. For each system, we selected the best reported number. The second and third columns show the percentage of known and unknown words in the test set, respectively. Accordingly, the last two columns show the accuracy of each tagger for known and unknown words. Based on the reported numbers, NPT performs better than the other approaches.

We compared conventional tagging models to NPT, but to the best of our knowledge, there is no other neural tagger for Farsi. The most related models to ours that similarly use NNs for tagging are those of Collobert et al. [2011] and Santos and Zadrozny [2014]. The model of Santos and Zadrozny [2014] is a character-based model in which embeddings for word characters are trained. Embeddings are combined through a convolution function to construct words. We do not have access to their network, so we could not apply the character-based model to our case. The other model mentioned is one of the most referred works in the field of neural tagging. Due to its importance, we decided to reimplement the model and study its behavior in our setting. The proposed model by Collobert et al. [2011] has a similar architecture to ours. We tried to exactly follow their work and reimplement the model as close as possible to the original network. The best accuracy that we could achieve by the reimplemented model was 95.13%.

Although the model by Collobert et al. [2011] performs well for English, NPT still performs better for Farsi. There could be several possible reasons for this, but we believe that the main reason is because of the network architecture. We proposed and tuned NPT specifically for the Farsi POS tagging task, whereas their model is a general-purpose network for different NLP tasks. They proposed two different implementations of the word approach network (WAN) and sentence approach network (SAN).[13] They have two hidden layers in the WAN model. The first layer includes a lookup table. Basically, the lookup table is a repository to store word vectors/embeddings whose

[13]Since SAN uses a convolutional architecture that is quite different from NPT, we only focus on WAN.

values are updated during the error back-propagation phase. This architecture is very close to NPT, but ours is more optimal in the following ways:

—Similar to WAN, we have a lookup table in the first layer. We initialize the lookup table with *Word2Vec / GloVe* embeddings. They have acceptable quality and preserve information about their context. In WAN, embeddings are updated during training; however, in our case, word embeddings are kept unchanged and we only update morphology embeddings. With this technique, we try to find the most optimal configuration. By using pretrained embeddings, we already have contextual information, and thus instead of tuning that part, we only focus on specific morphological information, which is more impactful for the POS tagging task. We empirically discovered this point. We obtained the best performance when we only changed morphological embeddings.
—NPT benefits from the morphological information, but WAN only works with surface form embeddings.
—We optimized the size of our hidden layers with respect to the problem, but WAN uses fixed-size hidden layers. We also selected the best nonlinear unit for our experiments. Moreover, hidden layers with a large number of nonlinear units can perform better [Mikolov et al. 2013; Zheng et al. 2013]. In our case, the size of the first hidden layer is 650, whereas this number is 300 for that of Collobert et al. [2011].
—The size of word embeddings is one of the most important features that directly affects tagging accuracy. Our embedding size is 200, but their model uses 50D embeddings.
—WAN receives five words, with the target word in the middle as its input. This setting might be suitable for English, yet in Farsi, words following the target word have no effect on the final accuracy, and therefore we do not use them. WAN also pads its input, with the padding size equal to half of the context window's size. It adds paddings both to the beginning and end of inputs. We believe that the way we prepare our input is more representative and useful. We exclude redundant following words. In contrast, the padding technique incorporates some extra amount of input information that is not necessarily helpful and can even degrade the accuracy. It should be also considered that we train word embeddings with the context window of 10 words, and thus the information that we provide for NPT is richer than that of WAN.

There could be several reasons for the difference between WAN and NPT, but we believe that the issues mentioned are the most likely.

Another important issue about the network is its architecture. Normally, POS tagging task is treated as a sequence labeling task. In such cases, recurrent networks are preferred; however, in our case, we were able to easily solve the same problem with a simpler model. The main advantage of recurrent models is that they can model long(er)-distance dependencies better than feed-forward models in their input, which yields a richer input/context. Additionally, due to the recurrency feature, they can provide information about tags of preceding words. These two features are not quite essential for our case. With regard to the first feature, in Farsi, the target word's tag is strongly influenced by the target word itself and its morphological features (see Figure 2, which clearly indicates this point). The second most impactful parameter is the preceding word just before the target word (w^{-1}), and the third most important factor is w^{-2}. Obviously, other context words have impact on the final accuracy, but their impact is negligible. Thus, we do not need a rich context in Farsi. This property is quite language depended, and with a very similar neural POS tagger to NPT and WAN, Zheng et al. [2013] reported that the optimal context window size for Chinese POS tagging was 3, but as reported in WAN, more than three words are required for English to produce acceptable results.

With regard to the second feature, to address the impact of the POS tags of the preceding words, we directly presented the POS tags to the network (as another input feature), but it did not provide any extra gain.

Our network choice was a trade-off between training time, accuracy, and structural complexity, and we tried to find the best configuration for Farsi POS tagging. Usually, POS taggers are used as modules that serve other NLP tasks, so they should perform fast. Our in-house recurrent POS tagger performs slightly better than NPT, but its training/running time is not comparable to NPT, so we prefer to use NPT in our NLP experiments.

6. CONCLUSION

In this article, we discussed an NN-based POS tagger for Farsi. The exclusive feature of the tagger is its accuracy. The main reason we preferred a neural model to other conventional models is its ability to tag unseen words. Its performance on OOV words is 89.01%, which is considerably better than other existing taggers. NPT is also fast in terms of runtime.

In addition to NPT, we developed a shallow analyzer to extract morphological features of Farsi words. In our corpus, we extracted 8,818 unique morphological annotations, which is a clear indication of the rich morphology of the language. Therefore, incorporating morphological information would appear to be essential for languages such as Farsi. We also studied the distributional representation of words. We use word embeddings in our tagger that provide distributed representations. Due to this feature, neural taggers generally perform better, especially in the presence of OOVs. Distributed representations preserve the semantic relatedness of words, as well as contextual information, which is crucial to POS tagging.

Since NNs have a good generalization capability, we hope to try and apply NN-based methods to other NLP tasks in low-resource languages to surpass the existing/conventional methods. There is no semantic role labeler or name-entity recognizer for Farsi, so NNs might be used efficiently for these purposes. These applications are examples of word-level computations with dependencies on context words. Apart from these models, we are also interested in sequence-level computations such as parsing or translation via NNs.

ACKNOWLEDGMENTS

We would like to thank the three anonymous reviewers for their valuable comments.

REFERENCES

James Bergstra, Frédéric Bastien, Olivier Breuleux, Pascal Lamblin, Razvan Pascanu, Olivier Delalleau, Guillaume Desjardins, et al. 2011. Theano: Deep learning on GPUs with Python. In *Proceedings of Advances in Neural Information Processing Systems 24 (NIPS'11)*.

Mahmood Bijankhan, Javad Sheykhzadegan, Mohammad Bahrani, and Masood Ghayoomi. 2011. Lessons from building a Persian written corpus: Peykare. *Language Resources and Evaluation* 45, 2, 143–164.

Thorsten Brants. 2000. TnT: A statistical part-of-speech tagger. In *Proceedings of the 6th Conference on Applied Natural Language Processing*. 224–231.

Ronan Collobert, Koray Kavukcuoglu, and Clément Farabet. 2012. Implementing neural networks efficiently. In *Neural Networks: Tricks of the Trade*. Springer, 537–557.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of Machine Learning Research* 12, 2493–2537.

Erick R. Fonseca, João Luís G. Rosa, and Sandra Maria Aluísio. 2015. Evaluating word embeddings and a revised corpus for part-of-speech tagging in Portuguese. *Journal of the Brazilian Computer Society* 21, 1, 1–14.

Eugenie Giesbrecht and Stefan Evert. 2009. Is part-of-speech tagging a solved task? An evaluation of POS taggers for the German Web as corpus. In *Proceedings of the 5th Web as Corpus Workshop*. 27–35.

Xavier Glorot and Yoshua Bengio. 2010. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*. 249–256.

Péter Halácsy, András Kornai, and Csaba Oravecz. 2007. HunPos: An open source trigram tagger. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*. 209–212.

Geoffrey E. Hinton, Simon Osindero, and Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Computation* 18, 7, 1527–1554.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. 1989. Multilayer feedforward networks are universal approximators. *Neural Networks* 2, 5, 359–366.

M. Jagadeesh, M. Anand Kumar, and K. P. Soman. 2016. Deep belief network based part-of-speech tagger for Telugu language. In *Proceedings of the 2nd International Conference on Computer and Communication Technologies*. 75–84.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the ACM International Conference on Multimedia (MM'14)*. ACM, New York, NY, 675–678.

Ji Ma, Yue Zhang, and Jingbo Zhu. 2014. Tagging the Web: Building a robust Web tagger with neural network. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, Vol. 1. 144–154.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In *Proceedings of the 12th International Conference on Computational Linguistics and Intelligent Text Processing, Part I (CICLing'11)*. 171–189.

William J. Masek and Michael S. Paterson. 1980. A faster algorithm computing string edit distances. *Journal of Computer and System Sciences* 20, 1, 18–31.

Karine Megerdoomian. 2004. Developing a Persian part of speech tagger. In *Proceedings of the 1st Workshop on Persian Language and Computer*. 99–105.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. arXiv:1301.3781.

Mahdi Mohseni and Behrouz Minaei-Bidgoli. 2010. A Persian part-of-speech tagger based on morphological analysis. In *Proceedings of the 7th International Conference on Language Resources and Evaluation (LREC'10)*. 1253–1257.

Farhad Oroumchian, Samira Tasharofi, Hadi Amiri, Hossein Hojjat, and Fahime Raja. 2006. *Creating a Feasible Corpus for Persian POS Tagging*. Technical Report No. TR3/06. University of Wollongong, New South Wales, Australia.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP'14)*. 1532–1543. http://www.aclweb.org/anthology/D14-1162.

John R. Perry and Alan S. Kaye. 2007. Persian morphology. *Morphologies of Asia and Africa* 2, 975–1019.

Juan Antonio Prezortiz and Mikel L. Forcada. 2001. Part-of-speech tagging with recurrent neural networks. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN'01)*.

Fahimeh Raja, Hadi Amiri, Samira Tasharofi, Mehdi Sarmadi, Hossein Hojjat, and Farhad Oroumchian. 2007. Evaluation of part of speech tagging on Persian text. In *Proceedings of the 2nd Workshop on Computational Approaches to Arabic Script-Based Languages*.

Cicero D. Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st International Conference on Machine Learning (ICML'14)*. 1818–1826.

Helmut Schmid. 1994. Part-of-speech tagging with neural networks. In *Proceedings of the 15th Conference on Computational Linguistics, Volume 1 (COLING'94)*. 172–176.

Mojgan Seraji. 2011. A statistical part-of-speech tagger for Persian. In *Proceedings of the 18th Nordic Conference of Computational Linguistics (NODALIDA'11)*. 340–343.

Mojgan Seraji, Beáta Megyesi, and Joakim Nivre. 2012. A basic language resource kit for Persian. In *Proceedings of the 8th International Conference on Language Resources and Evaluation (LREC'12)*. 2245–2252.

Mehrnoush Shamsfard, Soheila Kiani, and Yaseer Shahedi. 2009. STeP-1: Standard text preparation for Persian language. In *Proceedings of the 3rd Workshop on Computational Approaches to Arabic Script-Based Languages*.

Huihsin Tseng, Daniel Jurafsky, and Christopher Manning. 2005. Morphological features help POS tagging of unknown words across language varieties. In *Proceedings of the 4th SIGHAN Workshop on Chinese Language Processing*. 32–39.

Peilu Wang, Yao Qian, Frank K. Soong, Lei He, and Hai Zhao. 2015. Part-of-speech tagging with bidirectional long short-term memory recurrent neural network. arXiv:1510.06168.

Othman Zennaki, Nasredine Semmar, and Laurent Besacier. 2015. *Unsupervised and Lightly Supervised Part-of-Speech Tagging Using Recurrent Neural Networks*. Retrieved June 30, 2016, from https://aclweb.org/anthology/Y/Y15/Y15-1016.pdf.

Xiaoqing Zheng, Hanyang Chen, and Tianyu Xu. 2013. Deep learning for Chinese word segmentation and POS tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP'13)*. 647–657.

Received January 2016; revised March 2016; accepted April 2016