Evaluation of Word Embeddings for Sequence Labelling Tasks

A Anonymous **B Anonymous**

Abstract

XXX

1 Introduction

In the last years, distributed word representations have been applied to several NLP tasks. Inspired by distributional semantics models, distributed word representation methods represent each word as a continuous vector, where similar words have a similar vector representation, therefore, capturing the similarity between words.

The resulting vectors can be used as features in many NLP applications and it has been shown that they outperform methods that treats words as atomic units (). Their attractiveness relies in the ability to learn word representations in an unsupervised way, thus directly providing lexical features from big amounts of unlabelled data and, therefore, alleviating the cost of human annotations. It has been also claimed that word embeddings have the ability to connect out of vocabulary words to known ones, as well known as vocabulary expansion hypothesis. Hence, suggesting that word embeddings are a good resource for applications that need to be adapted to a certain domain, different from the one the application have been tuned for. For example,... Another property attribute to word embeddings is their capacity to encouraging common behaviour among related in-vocabulary words, for instance...

Something about the architectures

As with other learning methods, it is well known that the performance of machine learning algorithms heavily depends on parameter optimization, the size of the training data used and the applications they target. For example, (**?**) shows that the optimal word embedding dimensions are task specific. Moreover, there are several word embeddings methods, which used different algorithms and resources. Some methods involve feedback from the end task when learning (or fine-tuning) the word representations and others do not. Learning algorithm that involves fine-tuning are supposed to perform better since word representations become task-specific, at the cost of performing worst for out of vocabulary words. But still, there is not systematic comparison between these two methods.

In this paper, we perform an extensive evaluation of five word embedding approaches under fixed experiment conditions, and evaluate them over different sequence labelling tasks: POS-tagging, chunking, NER and MWE (Multi Word Expression Identification), within the following aims: (i) perform a fair comparison of different word embeddings algorithms. This includes running different word embeddings algorithms under controlled conditions, for example, use the same training set, the same preprocessing, etc. (ii) measure the influence of word embeddings in sequence labeling tasks in semi-supervised settings (fine-tuning) (iii) use word embeddings for MWE. To the best of our knowledge, word embeddings have not been used for this task before. (iv) systematically compared the usefulness of word embedding versus unigram features for sequence tagging.

static share hypothesis (in vocabulary words, e.g., individual first names are also rare in the treebank, but tend to cluster together in distributional representations); embedding structure hypothesis (e.g., group words by definiteness, like each, this, every, few, most, etc.).

2 Related Work

Word embedding learning methods are the new generation of distributional semantics models and have been applied to several NLP tasks that we summaries in this section.

?) proposed a neuronal network architecture that learn word embeddings and use them in POS-

tagging, chunking, NER and SRL. Without specializing their architecture for the mentioned tasks, they achieve close state-of-the-art performance. After including specialized features (e.g., word suffixes for POS-tagging; Gazzeters for NER, etc.) and other tricks like cascading and ensambling classifiers, achieve competitive state-of-the-art performance. Similarly, **?**) explored the impact of using word features learned from cluster-based and word embeddings representations for NER and chunking. They conclude that unsupervised word representation improve NER and chunking, and that combining different word representations can further improve the performance. Word representation from Brown clusters have been also shown to enhance Twitter POS tagging **?**).

?) presented a MWE analyser that, among other features, used unsupervised word clusters. They observed that the clusters were useful for identifying words that usually belong to proper names, which are considered MWE in the data set used. Nevertheless, they mentioned that it is difficult to measure the impact of the word embeddings features, since other features may capture the same information.

Word embeddings have been also used as features for syntactic dependency parsing and constituent parsing. **?**) used word embeddings as features for dependency parsing, which used the syntactic dependency context instead of the linear context in raw text. They found that simple attempts based on discretization of individual word vector dimensions do not improve parsing. Only when performing hierarchical clustering of the continuous word vectors then using features based on the hierarchy, they gain performance. They also pointed out that ensemble of different word embeddings representations improved performance. Within the same aim, (**?**) explores the used of word embeddings for constituency parsing and conclude that the information they provide might be redundant with the one acquire by a syntactic parser trained with a small amount of data. Others that boost the performance when including word embeddings representations for syntactic parsing includes (**?**; **?**; **?**; **?**).

Word embedding have also been applied to other (non-sequential NLP) tasks such as supersense tagging (**?**); grammar induction (**?**) and semantic task such as semantic relatedness, synonymy detection, concept categorization selection preferences and analogy (**?**)

3 Word Representations

Distributed word representation methods represent each word as a continuous vector, where similar words have a similar vector representation, therefore, capturing the similarity between words.

Example

The estimation of the word vectors can be carry out with different models architectures. We evaluate five different methods, which are the following:

3.1 Word Embedding Learning Algorithms

- Glove (**?**)
- Skip-gram (**?**)
- CBOW (**?**)
- Neural language model (**?**)
- Brown cluster (**?**)

The first three methods were chosen because they are recent state-of-the-art word embedding methods and because their software is available. The final method (Brown clusters) was selected as a benchmark word representation, which makes use of hard word clusters rather than a distributed representation.

Discuss here about fundamental differences between the above methods

4 Experimental Setup

We evaluate different learning approaches of word embeddings in four different sequence tagging tasks: POS tagging, chunking, NER and MWE identification.

4.1 Materials

It is well known that the choice of a corpora have an important effect in the final accuracy of machine learning algorithms. Therefore, each word embedding method was trained with the same corpora (Table **??**). The main reason of choosing the corpora is that they are publicly available.

4.2 Preprocessing

In order to make the comparison of different word embedding approaches across different applications, we applied the same preprocessing to the data sets used. The preprocessing pipeline consist

Data set	Size	Words
UMBC	48.1GB	3 billions
One Billion	4.1GB	0.8 billions
Latest wikipedia dump	49.6GB	3 billions

Table 1: Corpus used to learn word embeddings

of a sentence splitter, a tokenizer, a POS-tagger and a lemmatizer. The pipeline is built with the UIMA architecture and the DKPro NLP tools.

4.3 Hyperparameters tuning

The performance of learning methods heavily depends on their parameters optimization. In order to search for the best hyper-parameters, we look for the shared parameters across methods (methods specific parameters were set to their optimal reported values). The parameters we tuned are the followings:

- **Word vector size**: [25, 50, 100, 200, 400, 800].
- **Context window size**: [5, 10, 15].
- **Number of clusters**: [250, 500, 1000, 2000, 4000].

For each task, we split the data into a training set, validation set, and a test set. The hyper parameters are tuned on the validation set with random search (**?**). To be fair, for each model, we randomly choose 100 hyper parameter combinations and pick up the best one based on its performance on the validation set. Then each model is evaluated in a semi-supervised setting. We start with training models on 10% of the training data, and evaluate them on the test dataset. Then we incrementally add another 10% of the training data and evaluate them until all training data is used. We adopt per-word F1 scores as the evaluation metric for all tasks except POS tagging, for which we used per-word accuracy. In order to evaluate model performance on unknown words, we report also the average F1 scores for the words that do not occur in the training set.

4.4 Training

For each task, we feed learned embeddings into the graph transformer trained with sentence tag criterion (**?**). The graph transformer is equivalent to CRF, if we do not update word embeddings. For all tasks, we train graph transformer with pre-trained word embeddings in the following settings:

- CRF with conventional features.
- Graph transformer *does not* fine tune embeddings during training.
- Graph transformer fine tunes the embeddings during training.

We also set up experiments to verify that CRF/graph transformer requires different feature design for different kinds of pre-trained word embeddings. One kind of word embeddings is represented by (**?**). It maximises the word sequence likelihood with a model based on a weighted linear combination of word embedding features. Another kind of word embeddings is skip-gram and its variants, which is based on the dot product of two word embeddings. Therefore, we compare at least two ways of representing context word embedding features for each token:

i Concatenate word embeddings of context words within a fixed window as context features;

ii Concatenate the result of element-wise multiplication of current token embedding and each context word embedding as context features.

5 Experimental Results and Discussion

Because we can either update pre-trained word embeddings during training or not, through the evaluation, we want to answer the following questions:

- How well do different word embeddings perform in all tasks when supervised fine-tuning is *not* performed?
- How well do different word embeddings perform in all tasks when supervised fine-tuning is performed?
- How does the size of labeled training data affect the experimental results?
- How well do the word embeddings perform for unknown words?
- How do the key parameters of each word learning algorithms affect the experimental results?

6 Conclusion

Acknowledgments

Anonymised
Anonymised
Anonymised
Anonymised

	Training	Validation	Test	Feature space
POS-Tagging	0-18 of WSJ	19-21 of WSJ	22-24 of WSJ and English Web Treebank	as in (**?**)
Chunking	WSJ	1000 sentences WSJ	CoNLL2000	as (**?**)
NER	CoNLL2003 train set	CoNLL2003 dev. set	CoNLL2003 test set and MUC7	as in (**?**)
MWE	500 documents from	100 documents from	123 documents	as in (**?**)

Figure 1: Best results for each method for POS and Chunking

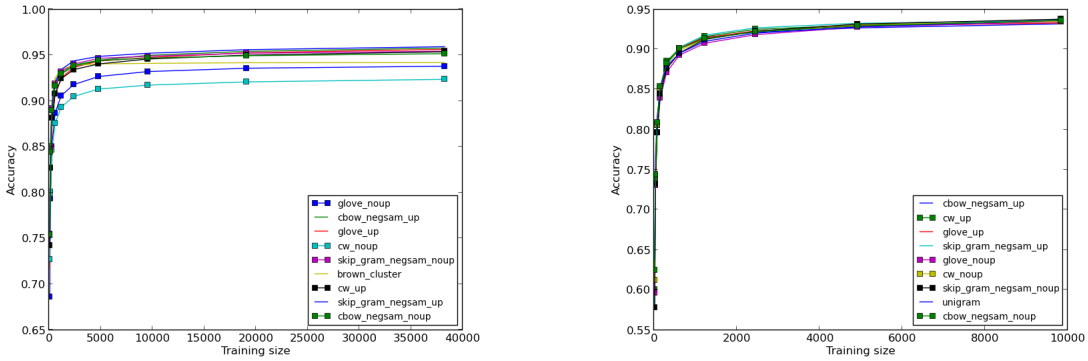


Figure 2: Best results for each method for NER and MWE

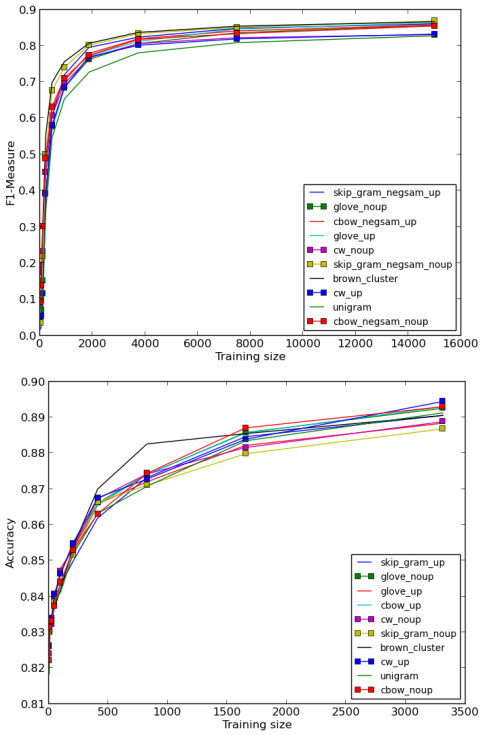
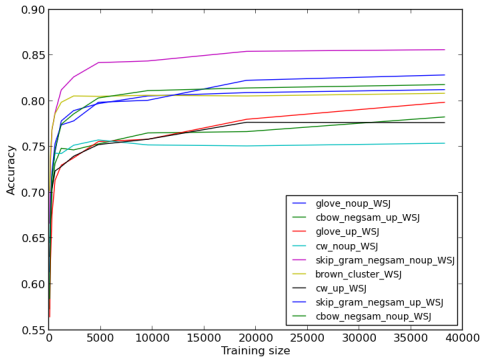
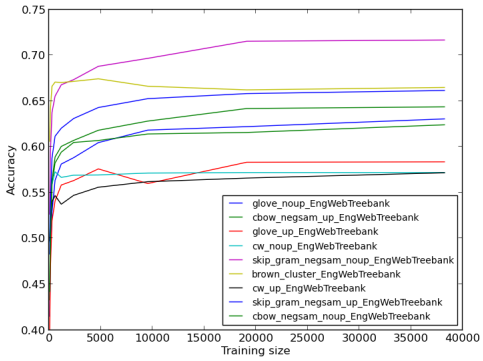


Figure 3: Out-of-vocabulary-words for *in domain* results for each method for POS

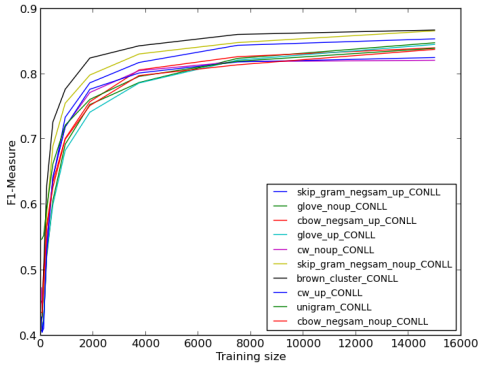


(a) *In domain*

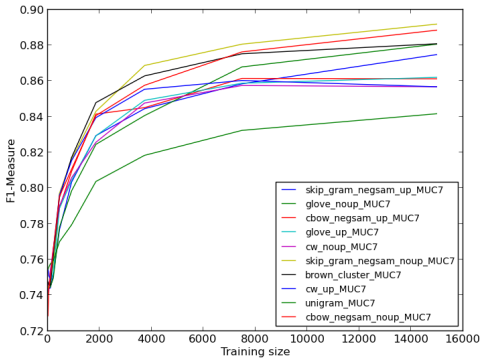


(b) *Out of domain*

Figure 4: Out-of-vocabulary-words for *in domain* results for each method for NER

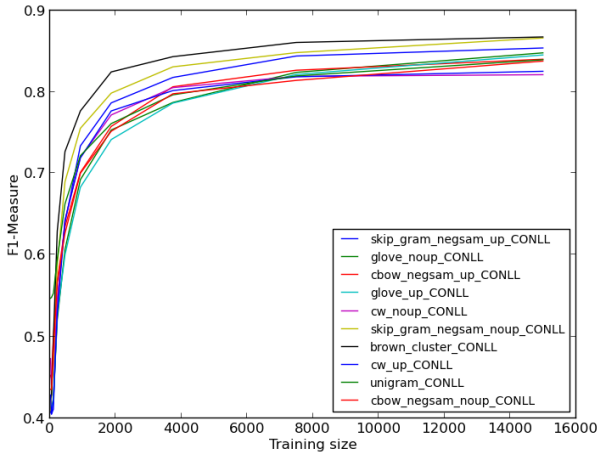


(a) *In domain*



(b) *Out of domain*

Figure 5: Out-of-vocabulary-words for *in domain* results for each method for MWE



Anonymised
Anonymised