Evaluation of Word Embeddings for Sequence Tagging Tasks

A Anonymous **B Anonymous**

Abstract

Word embeddings are distributed word representations which can be learned from unlabelled data, and have been shown to have high utility in many NLP applications. In this paper, we perform an extrinsic evaluation of five popular word embedding methods in the context of four sequence labelling tasks: POS-tagging, chunking, NER and MWE identification. A particular focus of the paper is analysis of the impact of fine-tuning the word representations during training. We show that when using word embeddings as features, as few as several hundred training instances are sufficient to achieve competitive results, and that word embeddings lead to improvements in results over all tasks. Perhaps more surprisingly, our results indicate that there is very little difference between the different word embedding methods, and indeed, that simple Brown clusters are often competitive with word embeddings across all tasks we consider.

1 Introduction

In recent years, distributed word representations have grown to become a mainstay of natural language processing (NLP), and been show to have empirical utility in a myriad of tasks (**?**; **?**; **?**; **?**). The underlying idea behind distributed word representations is simple: to map each word w in our vocabulary V onto a continuous-valued vector of dimensionality $d \ll |V|$. In this way, words of similar type (syntactic, lexical semantic or otherwise) will ideally be mapped to similar regions of the vector space, implicitly supporting both generalisation across in-vocabulary (IV) items and countering the effects of data sparsity for low-frequency and out-of-vocabulary (OOV) items.

Without some means of automatically deriving the vector representations without reliance on labelled data, however, word embeddings would have little practical utility. Fortunately, it has been shown that they can be "pre-trained" from unlabelled text data in various ways, based on a range of implementations of the distributional hypothesis, i.e. that words which occur in similar contexts tend to be semantically similar. Pre-training methods have been refined considerably in recent years, and scaled up to increasingly large corpora.

As with other machine learning methods, it is well known that the quality of the pre-trained word embeddings depends heavily factors including parameter optimization, the size of the training data, and the fit with the target application. For example, **?**) showed that the optimal dimensionality for word embeddings is task specific. One factor which has received relatively little attention in NLP is the effect of "fine-tuning" the pre-trained word embeddings as part of the training of the end task, based on self-taught learning (**?**). Fine-tuning leads to word representations that are task-specific, but often at the cost of over-fitting low-frequency and OOV words.

In this paper, we perform an extensive evaluation of five word embedding approaches under fixed experimental conditions, applied to four sequence labelling tasks: POS-tagging, full-text chunking, named entity recognition (NER), and multiword expression (MWE) identification. In this, we explore the following research questions:

RQ1: are word embeddings better than baseline approaches of one-hot unigram features and Brown clusters?

RQ2: are some word embeddings better than others in a sequence labelling context?

RQ3: what is the impact of fine-tuning word embeddings in sequence labeling tasks, both

empirically over the target task and geometrically over the vectors?

RQ4: what is the impact of word embeddings (with and without fine-tuning) on both OOV items (relative to the training data) and out-of-domain data?

2 Word Representations

The distributional hypothesis in linguistics suggests that "a word is characterised by the company it keeps" (**?**). Words that are used in the similar contexts tend to have similar semantic and syntactic properties. Capturing distributional similarity is the underlying idea of all word representation learning methods.

2.1 Types of Word Representations

Based on the ways of constructing word representations, **?**) categorise these methods into three types: *Distributinoal representation*, *Cluster-based representation*, and *Distributed representation*.

Distributional representation methods map each word w to its context word vector $\mathbf{C}_w$, which is built based on coocurrence counts between w and the words surrounding it. The learning methods store either directly the cooccurence count between two words w and i in C_{wi} () or project the concurrence counts between words into a lower dimensional space by using techniques such as SVD () and LDA ().

The methods of *Cluster-based representation* build clusters of words by applying either soft- or hard clustering algorithms. Some of them also rely on cooccurence matrix of words (). The Brown algorithm (**?**) is the most famous one in this category.

A *distributed representation* takes the form of a dense, low-dimensional, and continuous-valued vector. It is compact and stores latent features of a word. This kind of representations are also called word embeddings, which are built in the hope of capturing both syntactic and semantic properties of words.

2.2 Selected Word Representations

In a range of sequence tagging tasks, we evaluated five word representations : Brown clustering (**?**), Collobert & Weston (CW) (**?**), continuous bag-of-words model (CBOW) (), continuous skip-gram model (Skip-gram) (), and Global vectors (Glove) (). Except CW, all other methods are ranked as the best method in different recent empirical studies (). CW is included because it is one of the most influential early work in this area. The trainings methods of these word representations are unsupervised, and are ultimately derived from the word occurrence statistics of a corpus. Another property they shared in common is that all training objectives can be factorised into a sum of local training factors $J(w, \text{context}(w))$.

$$L = -\sum_{w \in V} J(w, \text{context}(w)) \qquad (1)$$

where V is the vocabulary set of a corpus and $\text{context}(w)$ denotes the local context of a word w. For a word w, its local context can either be its previous k words or m words surrounding it. Local training factors are designed to capture the relationship between current words and their local contexts. The underlying ideas of these factors are two-folds: i) predicting current words based on local contexts; ii) using current words to estimate their context words. Except Brown clustering, which utilises cluster-based representation, all other methods employ distributed representation.

The starting point of CBOW and Skip-gram models is to employ softmax for predicting word occurrence.

$$J_{wi} = \frac{\exp(\mathbf{v}_w^{\mathrm{T}} \mathbf{v}_{\text{context}(w)})}{\sum_{j \in V} \exp(\mathbf{v}_j^{\mathrm{T}} \mathbf{v}_{\text{context}(w)})} \qquad (2)$$

where $\mathbf{v}_{\text{context}(w)}$ denotes the distributed representation of the local context of word w. CBOW takes the average of the representation of all context words as $\mathbf{v}_{\text{context}(w)})$. Thus it estimates the probability of current words w given its local context. In contrast, Skip-gram applies softmax for each context word of a word w, in this case, $\mathbf{v}_{\text{context}(w)}$ is the corresponding distributed representation of the context word. This model is interpreted as predicting context words based on current words. In practice, softmax is too expensive to compute thus) propose to use hierarchical softmax and negative sampling to speed up training procedure.

CW considers the local context of a word w as m words to the left and m words to the right of w. The concatenation of embeddings of w and all its context words are taken as input of a neural network with one hidden layer, which produces a higher level representation $f(w) \in R^n$. Then the learning procedure replaces the embedding of w

with that of a randomly sampled word w' and generate another representation $f(w') \in R^n$ with the same neural network. The training objective is to maximise the difference between them.

$$J(w, \text{context}(w)) = \max(0, 1 - f(w) + f(w')) \quad (3)$$

This approach can be regarded as negative sampling with only one negative example.

Glove assumes the dot product of two word embeddings should be similar to logarithm of the co-occurence count X_{ij} of the two words. The local factor $J(w, \text{context}(w))$ becomes

$$g(X_{ij})(\mathbf{v}_i^{\mathrm{T}}\mathbf{v}_j + b_i + b_j - \log(X_{ij}))^2 \quad (4)$$

where b_i and b_j are the bias terms of word i and j, $g(X_{ij})$ is a weighing function based on the co-occurence count. This weighting function controls the degree of agreement between the parametric function $\mathbf{v}_i^{\mathrm{T}}\mathbf{v}_j + b_i + b_j$ and $\log(X_{ij})$. Frequently co-occurred word pairs will gain more weights than infrequent ones but stays the same if it is beyond a threshold.

Brown clustering introduces a finite set of word classes V for all words and partitions all words into these classes. The conditional probability of seeing the next word is defined as

$$p(w_k|w_{k-m}^{k-1}) = p(w_k|h_k)p(h_k|h_{k-m}^{k-1}) \quad (5)$$

where h_k denotes the word class of the word w_k, w_{k-m}^{k-1} are the previous m words and h_{k-m}^{k-1} are their respective word classes. Then $J(w, \text{context}(w)) = -\log p(w_k|w_{k-m}^{k-1})$. Since there is no practical method to find optimal partition of word classes, they consider only bigram class model, and utilise hierarchical clustering as an approximation method to find a sufficiently good partition of words.

2.3 Building Word Representations

For a fair comparison, we train the best performing methods Brown clustering, CBOW, Skip-gram, and Glove on a combination of all corpora in Table 1. The joint corpus was preprocessed with the Stanford sentence splitter and tokenizer. All consecutive digit substrings were replaced by NUM*f*, where *f* is the length of the digit substring (e.g., "10.20" is replaced by "NUM2.NUM2". The word embeddings of CW are downloaded from the website `http://metaoptimize.com/projects/wordreprs`.

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: Corpora used to learn word embeddings

Dimension of word embedding and context window size are the key hyperparameters of the learning methods for distributed representation. We use all possible combinations from the following ranges to train word embeddings on the combined corpus.

- **Word dimension**: [25, 50, 100, 200].
- **Context window size**: [5, 10, 15].

Brown clustering requires only the number of clusters as hyperparameter. We train word clusters with 250, 500, 1000, 2000, and 4000 clusters respectively.

3 Sequence Tagging Tasks

We evaluate different word representations in four different sequence tagging tasks: POS-tagging, chunking, NER and MWE identification. For the POS-tagging, chunking and MWE tasks, we used the same features as the state-of-the-art approaches, **?**), **?**) and **?**), respectively. For the NER, we used the same feature space as in **?**), except for the previous two predictions.aa

For each sequence tagging task, we feed learned word representations into a first order linear-chain graph transformer (**?**), and trained them by using the online learning algorithm AdaGrad (**?**). For each model taking distributed word representations as word features, we consider two settings:

- Graph transformer *does not* fine tune word representations during training (this is equivalent to a linear-chain CRF);
- Graph transformer fine tunes the word representations during training.

We consider also CRF models with hand-crafted features, which use one-hot representation for each unigram.

We split the task specific corpus into a training set, validation set, and a test set (see Table 2). If a corpus already provides fixed splits, we reuse them. For POS-tagging and NER, we also evaluated the models with a out-of-domain corpus (English Web-Treebank and MUC-7, respectively), which has similar annotation schema as the respective training corpus.

Table 2: Datasets splits and feature space for each sequence tagging task.

	Training set	**Validation set**	***in-domain* Test set**	***out-of-domain* Test set**
POS-Tagging	0-18 WSJ	19-21 WSJ	22-24 of WSJ	English Web-Treebank
Chunking	WSJ	1000 sentences WSJ	CoNLL-2000	Brown corpus
NER	CoNLL-2003 train set	CoNLL-2003 dev. set	CoNLL-2003 test set	MUC7
MWE	500 documents from	100 documents from	123 documents	-

In order to have fair and reproducible experimental results, we tuned the hyperparameters with random search (**?**). We randomly sampled 50 distinct hyperparameter sets with the same random seed for the models that do not update word embeddings, and sampled 100 distinct hyperparameter sets for the models that update word embeddings. For each set of hyperparameters, we train a model on its training set and pick up the best one based on its performance on its validation set (**?**). Note that, we also consider word vector size and context window size of distributed word representation, and number of clusters of brown clustering as the hyperparameters. This is achieved by mapping each possible hyperparameter combination to the word representation files trained with these parameters.

However, for the models that update word representations, we always found under-performed hyperparameters after trying out all hyperparameter combinations, because they have more hyperparameters than the models that do not update word representations. Then, for each distributed word representations, we reuse all hyperparameters of the models that do not update word representations, only tune the hyperparameters of AdaGrad for the word representation layer. This method requires only 32 additional runs for each model updating embeddings and achieves consistently better results than 100 random draws.

The final evaluation is carried out in a semi-supervised setting. We split the training set into 10 partitions at log scale. That means, the second smallest partition will be twice the size of the smallest partition. We created 10 training sets with incremental size by merging these partitions from the smallest one to the largest one, and each of them on the same designated test sets.

We adopted the most commonly used F1 measure as the evaluation metric for all tasks except POS tagging, for which we use per-word accuracy. In order to evaluate model performance on out-of-vocabulary (unknown) words, we reported also the accuracy for the words that do not occur in the training set.

In addition, we also set up experiments to verify if CRF/graph transformer requires different feature design for different kinds of pre-trained word embeddings. This is achieved by adding a hidden layer between CRF and distributed word representations. For each context word, the hidden layer computes the element-wise multiplication of its embedding with the embedding of the current word embedding, and the representation of current word stays the same. The results of this approach are not plotted because this method leads only to marginal improvement.

4 Experimental Results and Discussion

During the evaluation, we focus in addressing the following questions:

(i) Are the evaluated word embedding methods better than unigram features? To answer this question, we systematically compared the usefulness of word embedding versus unigram features for sequence tagging and noted that word embedding methods always outperform unigram features (Figures 1, **??**).

(ii) How does the size of labelled training data affect the experimental results? We observed that embedding methods are especially helping POS-tagging and NER when there are only several hundreds of training instances. Therefore, confirming that any of the evaluated word embedding methods should be used when POS-tagging or NER labelled data is limited. These early improvements are less evident for chunking and MWE, and according to our results, all the methods, including unigram, needed the same amount of training data to reach a decent performance.

(iii) How well do the word embeddings perform when evaluated with *out-of-domain* test sets? We measure the performance for *out-of-domain* of all the tasks, except MWE identification for which there is no other data set available (see Table 2). As expected, the performance goes down, more dramatically for NER for which

it drops about 1 point. The best performing embedding method turn to be skip-gram without fine-tuning and the worst method is unigram.

(iv) How well do the word embeddings perform for unknown words? As already mentioned, we measure the performance for out-of-vocabulary words (OOV) in two settings: with *in-domain* and *out-of-domain* corpora, for all the tasks, except MWE identification for which there is no other data set available (see Table 2). As expected, word embeddings and Brown clustering excel in *out-of-domain* performance. Word embeddings without fine-tuning enhance even more the performance of OOV for the *in-domain* and *out-of-domain* settings (Figure 2) since fine-tuned word representations become task-specific, hence performing worst for OOV.

(v) How well do different word embeddings perform in all tasks when semi-supervised fine-tuning is not performed?, and **(vi) and how well do different word embeddings perform in all tasks when semi-supervised fine-tuning is performed?** Across all the methods, fine-tuning is helping POS-Tagging and MWE, where the CW method has been found to be the most sensible to tuning, reaching almost 3 points more, when tuning is performed. For chunking and NER, the best results are fine-tuned, but the difference across all the methods and updated features versus not-updated ones, is not significant. We also found that fine-tuning can correct poorly learned word representations, but can be overfitted if unsupervisely learned ones are already good.

Finally, we address the following question: **(vii) It has been shown that Brown clusters are useful features for MWE identification but, are also word embeddings helping MWE identification?** According to our experiments, the word embedding features distilled by fine-tuned CW reached the best results, beating the state-of-the-art performance (see Table 3). However, between Brown clusters and fine-tuned CW, learned under the same settings, the difference is not impressive, suggesting that distributional word representations and cluster-based representations captures similar features for the MWE identification task. Thus, a natural question: would it be better to learn distributional representations for MWE, instead of representations of single words?

As a reference, we compared our best results for each task with their corresponding benchmarks (Table3). For POS-tagging and chunking, we reach a comparable performance to the state-of-the-art methods. The difference between our NER system and its baseline is most obvious, as we are 0.025 points below them, but the comparison is not fair considering that their algorithm is much complex (**?**) used a 2^{nd} order CRF, while we used a 1^{rst} order CRF). For the task of MWE identification, our implementation and settings beat the baseline. However, in this paper, we do not aim to maximize the absolute performance of the tasks under study, but rather to study the impact of word embeddings for sequence tagging tasks under control settings.

Table 3: Benchmark results vs. our best results

Task	Benchmark	Us
POS-Tagging	(Accuracy) 97.24 (**?**)	0.9592 (skip-gram negsam+up)
Chunking	(F1) 0.9429 (**?**)	0.9386 (Brown cluster v2000+)
NER	(F1) 0.8931 (**?**)	0.8686 (skip-gram negsam+noup)
MWE	(F1) 0.6253 (**?**)	0.6546 (cw+up)

5 Related Work

Word embedding learning methods 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 Semantic Role Labelling. 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.

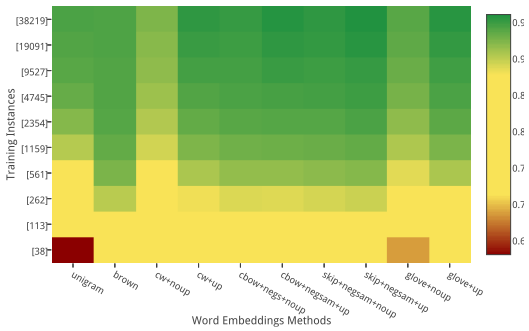
Word embeddings have been also used as features for syntactic dependency parsing and constituent parsing. **?**) used word embeddings as features for dependency parsing, which used the syntactic dependency context instead of the linear context in raw text. They found that simple attempts based on discretization 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 like grammar induction (**?**) and semantic tasks such as semantic relatedness, synonymy detection, concept categorization selection preferences and analogy (**?**)

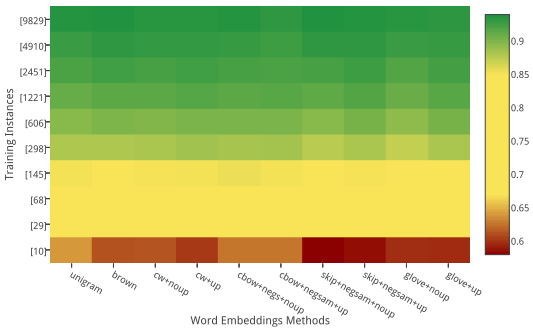
6 Conclusion

We have performed an extensive extrinsic evaluation of five word embeddings methods approaches under fixed experiments conditions, and evaluate them over different sequence tagging tasks: POS-tagging, chunking, NER and MWE identification. We found that word embeddings methods always outperformed unigram features, especially when the training size is small, but no method was consistently better than the others across the different tasks and settings. Word representations were also found useful for improving the accuracy of OOV. We expected to see an important gap between the performance of fine-tuned features in a semi-supervised setting and no fine-tuned ones, but the difference is marginal. Nevertheless, we found that fine-tuning can result in over-fitting, when the ones learned unsupervisely were already good. Finally, by using word embeddings as features for MWE identification, we outperformed the state-of-the-art system. We could not find any trend that suggest that a word embedding method is better than other, thus suggesting that ... Future studies include learning representation of complex sequences such as MWEs.

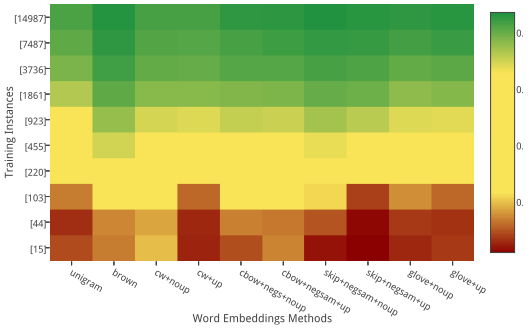
Figure 1: Best results for each method for POS-Tagging and Chunking. The x-axis correspond to the different word embeddings methods and the y-axis to the 10 training partitions at log scale. Green color stand for high performance, while red color stands for low performance. The methods are in chronological order



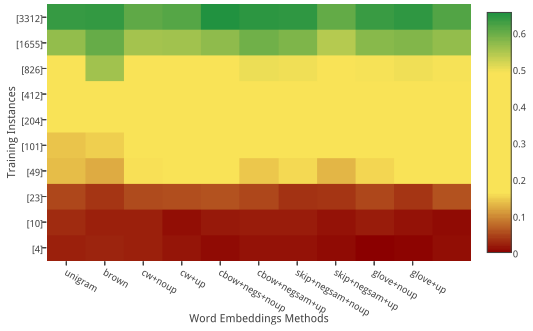
(a) POS-Tagging Accuracy



(b) Chunking F1-Measure

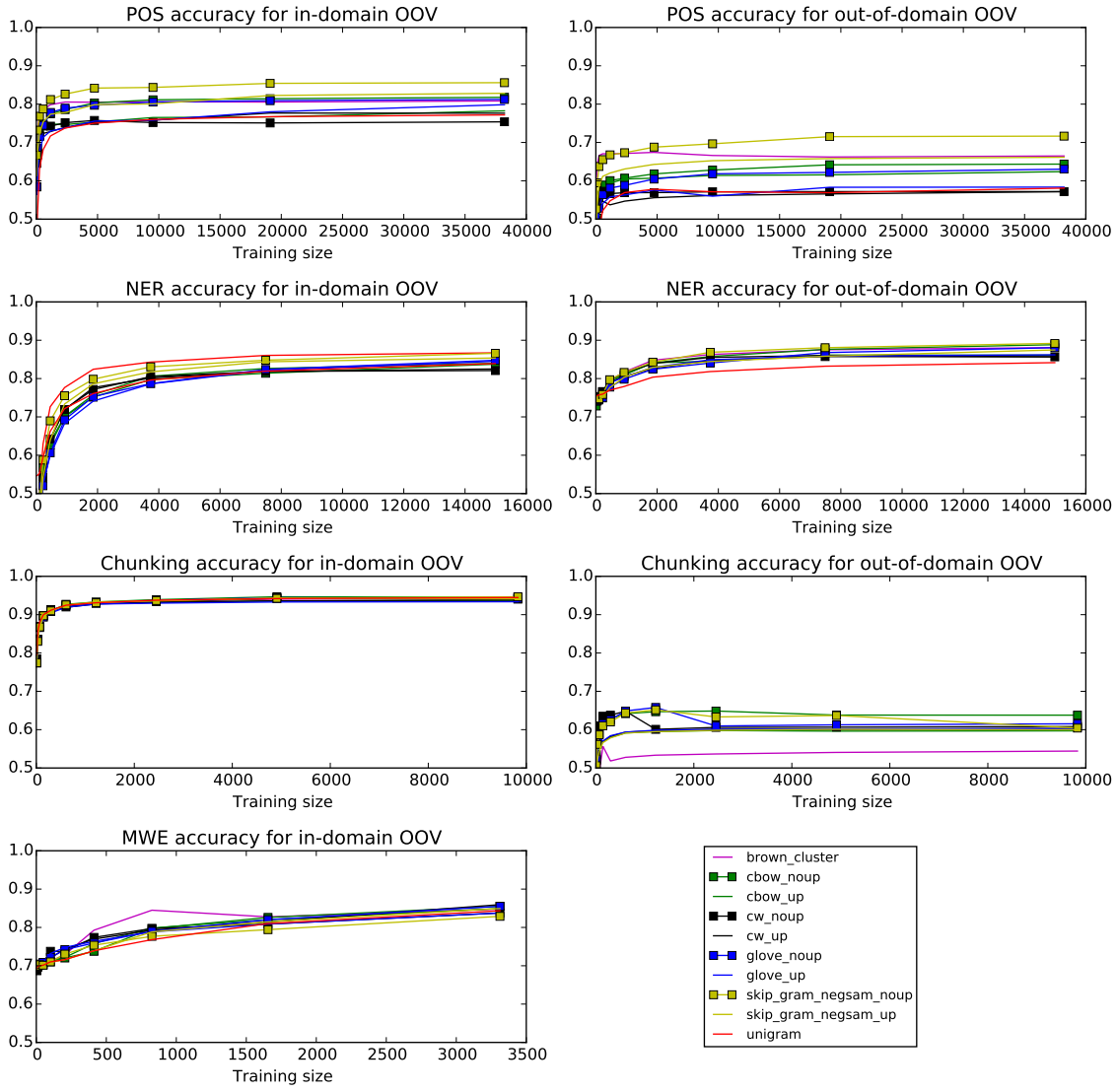


(c) NER F1-Measure



(d) MWEs F1-Measure

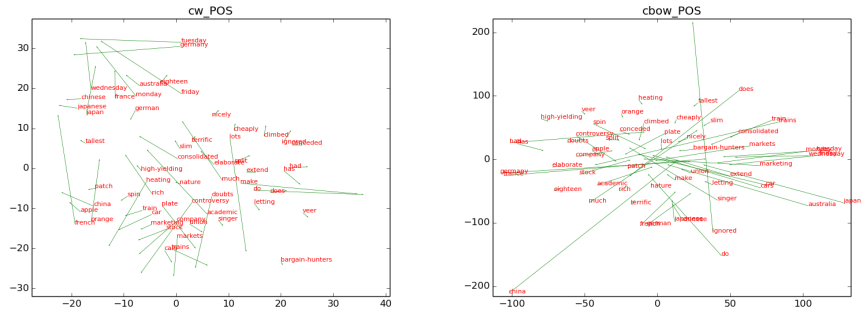
Figure 2: Out-of-vocabulary-words (OOV) accuracy for *in-domain* and *out-of-domain* test sets



Acknowledgments

Anonymised
Anonymised
Anonymised
Anonymised
Anonymised
Anonymised

Figure 3: updated vs. no-updated word representations for POS



(a) (b)

(c) (d)