Effective Word Representation for Named Entity Recognition

Jun-Ting Hsieh
Stanford University

Chengshu Li
Stanford University

Wendi Liu
Stanford University
{junting, chengshu, wendiliu}@stanford.edu

Abstract

Recently, various machine learning models have been built using word-level embeddings and have achieved substantial improvement in NER prediction accuracy. Most NER models only take words as input and ignore character-level information. In this paper, we propose an effective word representation that efficiently includes both the word-level and character-level information by averaging its character n-gram embeddings. Our best performing model uses a bidirectional LSTM with word and character n-gram embeddings, which obtains close to state-of-the-art performance in NER prediction of the English language.

1 Introduction

The use of word vectors has enjoyed great success in various Natural Language Processing (NLP) tasks, including named entity recognition (NER), part-of-speech (POS) tagging, and dependency parsing. Each word is assigned a dense low-dimensional real-valued vector, also called an embedding. Since word vectors are low-dimensional and can capture relations between words, the use of word vectors have become more popular and successful than the traditional one-hot representations.

There are many ways to train word vectors. The word2vec model (Mikolov et al., 2013) introduced the Skip-Gram model, which takes each word and predicts its surrounding context words. The Continuous Bag-of-Words (CBOW) does the inverse by predicting each center word by taking an average of its surrounding context words. Currently, the most commonly used are the GloVe vectors (Pennington et al., 2014), which are trained by using the co-occurrence matrix of the words.

While many NLP models achieve high performance by using only words as inputs, these models suffer from rare words in the dataset. The pre-trained word vectors for rare words are usually poorly represented since they do not occur frequently enough during training. Some words do not even have pre-trained word vectors. As a result, word-based models often perform poorly when there are rare words in the dataset.

Therefore, we suggest a character-based model that will attempt to solve this problem, using character n-grams (we will refer to character n-gram as simply "n-gram" hereafter). For each word, we will construct a new embedding from its n-grams and its original word embedding. The character-level information is captured by averaging the n-gram embeddings, which is supplementary to the word vector. Intuitively, this can improve the representations of rare words. For example, the word "distintiveness" is rare and its pretrained word vector is probably not very accurate. However, its n-grams are likely to capture the information of its sub-string "distinct", which is a common word. More importantly, unlike other more complicated character-based models, our algorithm is very efficient to calculate and train.

Our project is divided into two parts. First, we train our character n-gram embeddings using the Skip-gram model on the WikiText-2 dataset. Second, we analyze the effect of n-gram embeddings on named entity recognition (NER), using the CoNLL-2003 dataset.

2 Approach

2.1 Training n-gram embeddings

Our goal of training n-gram embeddings is to efficiently provide morphological features and information about rare and unknown words, which hopefully will complement existing word vectors and improve end-to-end NLP task such as NER.

2.1.1 Dataset

We used the case sensitive WikiText-2 dataset to train our n-gram embeddings because it has a reasonable size and contains many name entities (see Table 1). We preprocessed the dataset by stripping foreign languages, numbers and titles so that the dataset only have paragraphs of English texts.

	train	dev	test
Articles	600	60	60
Words	2,088,628	217,646	245,569
Unique words		33,278	

Table 1: WikiText-2 Dataset

2.1.2 Model

We train our character n-gram embeddings using the same Skip-Gram objective function as the word2vec embeddings. We construct the vocabulary of the character n-grams in the training data by setting a threshold. Character n-grams that appear less than the threshold times will be considered as UNK n-grams. The final average character embedding for a given word is the average of all character n-gram embeddings it contains. For example, the character n-grams (n = 1, 2, 3) of the word "cat" are c, a, t, ˆc, ca, at, t|, ˆca, cat, at|, where ˆ and | are two special characters we define that represents the start and the end of a word, respectively.

To be more exact, each word w represented with average character embedding is computed as the following:

$$V(w) = \frac{1}{K}\sum_{i=1}^{K} cn_i$$

where cn_i is the n-gram embedding for the i_{th} character n-gram of w.

For each pair of center word v_c and context word u_o, we compute the objective function for Skip-Gram model using negative sampling:

$$\sum_{(v_c,u_o)} \Big(\log \sigma(V(v_c) \cdot V(u_o)) - \sum_{i=1}^{N} \log \sigma(-V(v_c) \cdot V(u_i)) \Big)$$

where $\sigma(\bullet)$ is the logistic sigmoid function and u_i is a negative sample.

After the training, each word can then be represented as the concatenation of its corresponding word and average character embeddings.

2.1.3 Implementation Details

Built on top of the code base from StanfordSentiment, we create a (word, word id) mapping, a (n-ngram, n-gram id) mapping, and a (word id, a list of n-gram ids) mapping. Then we modified the original word2vec code base, which uses Skip-Gram model and negative sampling, to train our

	train	dev	test
Sentences	14041	3250	3453
Words	203621	51362	46435
Unique words	17485	7848	7256
O	169578	42759	38323
PER	11128	3149	2773
ORG	10025	2092	2496
LOC	8297	2094	1925
MISC	4593	1268	918

Table 2: CoNLL-2003 Dataset

n-gram embeddings. The key `tf.Variable` now is an n-gram embedding matrix rather than a word embedding matrix.

2.2 Named Entity Recognition

We will use the NER task to analyze the effect of using n-gram embeddings because it has been shown that character-level features can improve the accuracy of NER tasks (dos Santos and Guimaraes, 2015)

2.2.1 Dataset

We use the standard CoNLL-2003 dataset. This dataset concentrates on four types of named entities: persons, locations, organizations, and names of miscellaneous entities that do not belong to the previous three groups. Therefore, each word in the dataset is labeled as one of the following: `PER, LOC, ORG, MISC, O`.

The dataset provides us the split of train, dev, and test data. The statistics of the dataset are shown in Table 2.

2.2.2 Model

For each word, we construct a new embedding using the word itself and its n-grams. We extract the n-grams, look up the n-gram embeddings for each of them, and average them to get an averaged n-gram embedding. Then, we concatenate the original word embedding and the averaged n-gram embedding to form a new embedding, which has twice of its original dimension.

In addition, since the words and n-grams are lower cased, when looking up the embeddings, we add a spelling feature to represent the original casing of the word. There are four types: all uppercase, all lowercase, first character is upper case, and others. We randomly initialized 4 vectors for these features, which we call "spelling embedding". Therefore, in total, our newly constructed embedding for a word is the concatenation of its word embeddings, averaged n-gram embeddings, and spelling embeddings.

We use these new embeddings as inputs to a recurrent neural network (RNN) model. We did experiments using different cells, including RNN and LSTM, and we also experimented with bidirectional LSTM. As expected, we found that bidirectional LSTM performed the best. Thus, bidirectional LSTM will be our backbone model in this project. Finally, we take the output states of the RNN and use a simple transformation to predict the NER label of each word.

2.2.3 Implementation Details

We only include 2-grams, 3-grams, and 4-grams. We exclude the unigrams (single characters) based on the assumption that the individual characters don't really contribute to the meaning of the word. For the embeddings, our word embeddings, n-gram embeddings, and spelling embeddings are all 50-dimensional, and our hidden states in the LSTM (or bidirectional LSTM) are 300-dimensional.

Every word has different number of n-grams but the n-gram input tensors have to be of the same shape. Thus, we set the maximum number of n-grams to be 36, and pad the n-gram input vector with

0's (which represent the "UNK" token) or truncate the input. As a result, we also need an n-gram boolean mask which tells us the actual number of n-grams and which n-grams to ignore.

During training, we apply dropout with 0.5 probability to the output states of the RNN for regularization. We turn off the dropout during evaluation. For gradient descent, we use the Adam algorithm with 0.001 learning rate. When fine-tuning the embeddings, we tried using the vanilla gradient descent optimizer for the embeddings and the Adam optimizer for the rest of the variables. However, we found that this approach only slows down the training and does not affect the results, so we stay with the Adam optimizer only.

2.2.4 Evaluation

The standard way to evaluate an NER model is to look at the F1 score. The F1 score is the harmonic mean of the precision and recall. After each epoch, we evaluate our model on the dev set and calculate the F1 score. If we find a new best score, we save the tensorflow session and the model. At test time, we restore the session and the model, and evaluate it on the test set.

2.2.5 Tasks

We perform several tasks to analyze the effect of using n-grams embeddings. For our word embedding matrix, we can initialize it randomly or with the GloVe (50-dimensional) vectors. Similarly, we can initialize our n-gram embedding matrix randomly or with our own pre-trained embeddings from WikiText dataset. our tasks consist of different combinations of word and n-gram initializations.

Note that the GloVe vectors are trained on billions of words and have coverage over 400,000 unique words, whereas our CoNLL-2003 dataset (see Table 2) has less than 20,000 unique words. Also, due to the limit of time and computing power, our n-gram embeddings were trained on around only 2 million words and 30,000 unique words. Thus, we anticipate that the GloVe vectors alone can achieve very high performance, and adding our pre-trained n-gram embeddings is not likely to have a huge impact.

Therefore, in order to analyze the effect of n-gram embeddings, we add tasks that include GloVe vectors but only the top 10,000 or 5,000 frequent words. This way, we limit the coverage of GloVe vectors, hoping that the effect of including n-gram embeddings can be more significant. Thus, in total, we have the following tasks:

1. Random word
2. Random n-gram
3. Random word + Random n-gram
4. GloVe word
5. pre-trained n-gram
6. GloVe word + pre-trained n-gram
7. GloVe word (top 10,000)
8. GloVe word (top 10,000) + pre-trained n-gram
9. GloVe word (top 5,000)
10. GloVe word (top 5,000) + pre-trained n-gram

We expect to see a big improvement from Task 1 to Task 3, but with a low F1 score. We also expect to see a slight improvement from Task 7 to Task 8 and from Task 9 to Task 10, while still achieving a high F1 score. And finally, Task 4 and Task 6 should have the highest scores, but since GloVe vectors are so powerful, there might be little or no improvement from adding n-gram embeddings.

Structure	score
Forward RNN	0.769
Bidirectional RNN	0.880
Forward LSTM	0.782
Bidirectional LSTM	0.919

Table 3: Dev F1 score with GloVe word embeddings

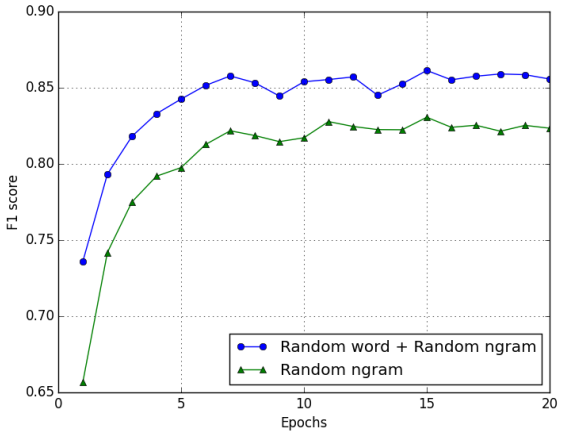

Figure 1: Dev F1 score after each epoch, bidirectional LSTM

3 Experiment Setup

3.1 Trained n-gram embeddings

We train our model for 20,000 steps. The original word2vec Skip-Gram model was trained for 80,000 steps. We decided to reduce the number of steps because we found that the loss seemed to converge after around 4,000 steps (see below for Figure 3).

3.2 Named Entity Recognition

We trained and tested our models with the CoNLL-2003 dataset. Moreover, we did several experiments to find the best model for NER.

3.2.1 RNN Structure

We experimented with forward RNN, forward LSTM, bidirectional LSTM. The results are shown in Table 3. As expected, we found that bidirectional LSTM gives the best performance. Thus, we use bidirectional LSTM for the rest of the project.

3.2.2 Number of epochs

Most machine learning models will converge after a certain number of epochs. Further epochs do not increase accuracy, and we might start to see overfitting. Therefore, we first tested the number of epochs our model needs to converge. We ran our model and looked at the dev set F1 score after each epoch. Intuitively, the random initialized embeddings should take longer to converge.

As shown in Figure 1, we found that on all tasks, our model converges at or before the 10th epochs. Therefore, we set the number of epochs to 15, just in case 10 isn't enough.

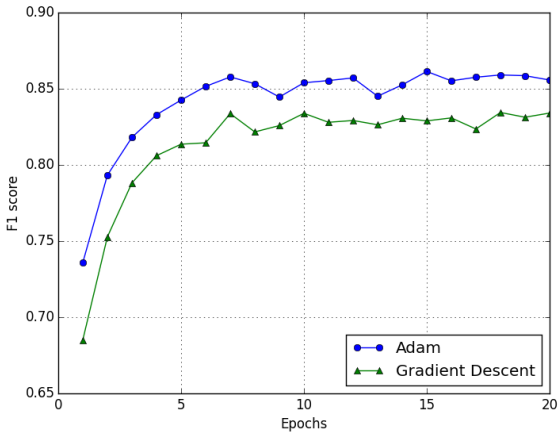

Figure 2: Dev F1 scores for Random word + Random n-gram, bidirectional LSTM. The embeddings are updated by either Adam or vanilla Gradient descent.

3.2.3 Gradient Descent Optimizer

It is common practice to use the vanilla gradient descent optimizer for embeddings, while using the Adam optimizer for the rest of the variables. The reason is that the embeddings might fluctuate too much due to the adaptive learning rate of the Adam algorithm. Thus, we tried updating n-gram embeddings using a vanilla gradient descent optimizer with 0.1 learning rate, while updating the rest of the variables using Adam optimizer.

However, as shown in Figure 2, we found that using gradient descent results in worse performance, and this is consistent across other tasks as well. Therefore, we decided to stay with the Adam optimizer.

3.2.4 N-gram Embeddings

We ran each task (described in Section 2.2.4) for 5 times and looked at the dev scores and test scores. The results are shown in Table 4 and Table 5. Interestingly, we saw that our pre-trained n-gram embeddings alone can achieve a decent F1 score, and more importantly, in most cases, adding n-gram embeddings does improve the performance. We will give a thorough analysis in the next section.

4 Experiment Results and Analysis

4.1 Trained n-gram embeddings

From the 2-million-word Wiki corpus, we extracted 33,000 unique four-grams. A visualization of t-SNE analysis of the 100 most frequent 4-grams are shown in Figure 4. In this figure, we can see that prepositions ('of', 'to', 'at'), punctuations (',', '.') and conjunctions ('and', 'and—') are all clustered together. In Figure 5, we represent the 100 most frequent words as the average vector of all the n-grams they contain, and this shows that our training is quite successful. Pronouns with similar n-grams ('who', 'when', 'which', 'where'), numbers (00, 0000) and punctuations (',', '.', '(', ')') are all clustered together. In Figure 6, we pick a few common prefixes and visualize their embeddings. We would expect that positive prefixes ('re', 'en', 'pro') would fall close to each other, so would negative prefixes ('im', 'in', 'il', 'ir') and numerical prefixes ('mon', 'bi', 'tri', 'qua'). However, our result slightly deviates from our expectation, which means there is room for improvement in our char2vec model.

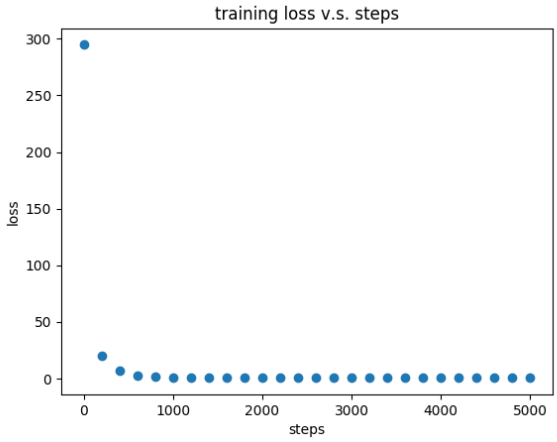

Figure 3: character n-gram embedding training loss

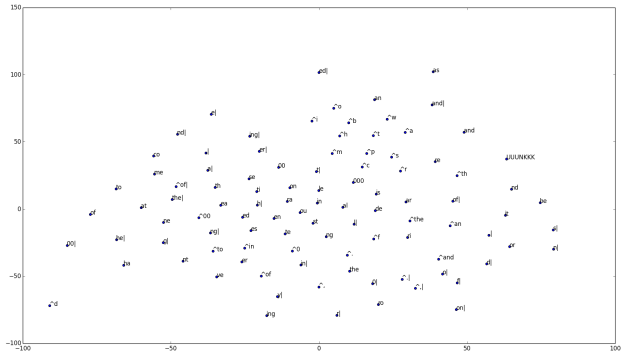

Figure 4: t-SNE visualization of 4-gram embeddings trained on WikiText-2

4.2 Named Entity Recognition

4.2.1 Effects of n-grams on NER

The dev and test scores of all tasks are shown in Table 4 and Table 5. First, we found that the test scores are consistently lower than the dev scores. This is consistent with the results of previous works using the CoNLL-2003 dataset.

We investigated the test dataset and found that a majority of the sentences in the test set are sports related, for example "SOCCER - JAPAN GET LUCKY WIN , CHINA IN SURPRISE DEFEAT." In this example, the gold label for Japan is LOC but is PER for China, even though they should both be LOC. We also found inconsistency within the dataset itself. For example, a sentence that consists of only two words, "PACIFIC DIVISION", is labeled as "LOC O", but another sentence "MIDWEST DIVISION" is labeled as "O O". They are obviously incorrectly labeled as both should be labeled MISC.

Next, we looked at the results of our n-gram embeddings. We found that the pre-trained n-gram embeddings alone achieved a score of 0.835. Moreover, for Task 3, 8, and 10, we saw that including our pre-trained n-gram embeddings does improve the score significantly. This shows that by extracting n-grams of a word and averaging the n-gram embeddings, we can boost the performance of a word-based model.

For Task 6 (using the GloVe vectors), we found that using n-gram embeddings does not boost the test score. We suspect that this is due to difference between the amount of training between the GloVe vectors and our n-gram embeddings. As described in Section 2.1.1, we only have 33,000

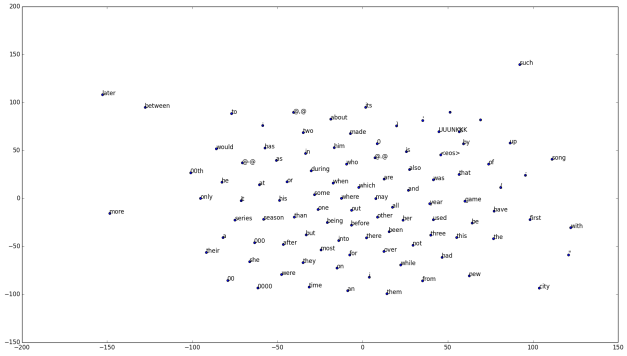

Figure 5: t-SNE visualization of word embeddings represented as the average of their n-gram vectors

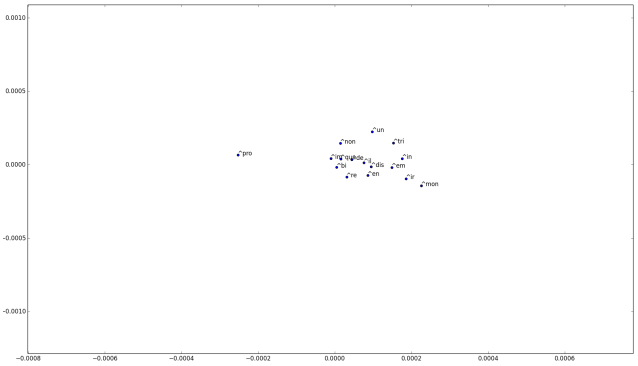

Figure 6: t-SNE visualization of prefix embeddings

unique n-grams and their embeddings are trained on 2 million words. In contrast, GloVe vectors cover 400,000 unique words and are trained on billions of words. Since our CoNLL-2003 dataset is relatively small (Table 2), the GloVe vectors already have big coverage over the words in the dataset, and adding our n-gram embeddings does not improve the performance.

This argument can be justified by Task 8 and Task 10, where we limit the GloVe vectors to only cover the top 10,000 or 5,000 frequent words. The difference between Task 9 and 10 is larger than the difference between Task 7 and 8. This shows that as we increase the coverage of GloVe vectors, the improvement from n-grams decreases.

We suggest that if we train our n-gram embeddings on a large dataset, for example, the WikiText-103 dataset, we will have more n-gram coverage and more accurate vectors. Then, we could see a significant improvement from adding n-gram embeddings.

4.2.2 Effects on UNK words

Intuitively, our character n-gram approach should help the most when it comes to NER classification for UNK words. UNK words can't take advantage of the GloVe pre-trained word embeddings. On the other hand, they may contain common character n-grams that could provide useful information about their meaning.

Table 6 shows the F1 scores on test set for UNK words. As we expect, n-gram embeddings performs better than word embeddings or word embeddings + n-gram embeddings because word embedding for UNK word really provides zero information about the actual word. We are not quite sure why the F1 score results for UNK words are better when we limit our vocabulary to the top 5,000 and 10,000 Glove words.

Task	Task	0	1	2	3	4	Average
1	Random word	0.853	0.851	0.848	0.848	0.848	0.850
2	Randon n-gram	0.829	0.825	0.828	0.83	0.827	0.828
3	Random word + Random n-gram	0.865	0.86	0.861	0.862	0.864	0.862
4	GloVe word	0.918	0.918	0.919	0.921	0.919	0.919
5	pre-trained n-gram	0.835	0.837	0.834	0.837	0.834	0.835
6	GloVe word + pre-trained n-gram	0.923	0.921	0.923	0.920	0.921	0.922
7	GloVe word (top 10000)	0.894	0.890	0.892	0.895	0.896	0.893
8	GloVe word (top 10000) + pre-trained n-gram	0.897	0.899	0.896	0.898	0.894	0.897
9	GloVe word (top 5000)	0.881	0.881	0.88	0.881	0.879	0.880
10	GloVe word (top 5000) + pre-trained n-gram	0.888	0.89	0.887	0.887	0.888	0.888

Table 4: F1 scores on dev set

Task	Task	0	1	2	3	4	Average
1	Random word	0.788	0.797	0.798	0.789	0.786	0.792
2	Randon n-gram	0.771	0.773	0.768	0.766	0.776	0.771
3	Random word + Random n-gram	0.804	0.798	0.803	0.809	0.803	0.803
4	GloVe word	0.861	0.876	0.864	0.865	0.866	0.866
5	pre-trained n-gram	0.779	0.780	0.779	0.778	0.777	0.779
6	GloVe word + pre-trained n-gram	0.862	0.870	0.864	0.866	0.867	0.866
7	GloVe word (top 10000)	0.839	0.833	0.840	0.835	0.833	0.836
8	GloVe word (top 10000) + pre-trained n-gram	0.843	0.837	0.835	0.840	0.839	0.839
9	GloVe word (top 5000)	0.818	0.82	0.824	0.815	0.83	0.821
10	GloVe word (top 5000) + pre-trained n-gram	0.825	0.825	0.826	0.824	0.826	0.825

Table 5: F1 scores on test set

5 Related Works

There has been a large interest in building language models using letter-based word embeddings. In order to address the rare word problem, Bilmes and Kirchhoff (2003) came up with a count-based n-gram language model. Building on top of their work, Alexandrescu and Kirchhoff (2006) represent each word as a set of shared factor embeddings. Gillick et al. (2015) model the task of sequence labeling as a sequence to sequence learning problem and incorporate character-based representations into their encoder model. Chiu and Nichols (2015) designed an architecture using CNNs to learn character-level features, which is similar to the work by Santos and Guimaraes (2015). Zhang, Zhao and LeCun (2015) used an architecture that does not need word embeddings at all and show that for text classification, a deep CNN over characters performs well.

A lot of interesting work has been done using deep learning for named entity recognition. Passos et al. (2014) extend the Skip-Gram language model (Mikolov et al., 2013) to produce phrase embeddings used in a linear-chain CRF to perform NER. As mentioned above, Santos and Guimaraes (2015) show that using neural character embeddings largely boosted their prediction accuracy. Our work is similar to theirs in that we accomplish the full task of identifying and classifying named entities using only features automatically learned.

Ling et al. (2015) apply a bidirectional LSTM over characters to use as inputs for language modeling and part-of-speech tagging. They show improvements on various languages (English, Portuguese, Catalan, German, Turkish). We are able to show that bidirectional LSTM outperforms vanilla LSTM yet it remains open as to whether CNN or LSTM performs better on this task.

6 Conclusion

In this project, we trained n-gram embeddings using the Skip-Gram model, and then use them as additional input features to an NER model using bidirectional LSTM. We successfully trained our n-gram embeddings such that similar character n-grams are clustered together. For NER, we found

Run	0	1	2	3	4	Average
Random word	0.786	0.790	0.791	0.789	0.779	0.787
Randon n-gram	0.819	0.821	0.819	0.815	0.823	0.819
Random word + Random n-gram	0.804	0.804	0.795	0.805	0.812	0.804
GloVe word	0.798	0.797	0.760	0.762	0.773	0.778
pre-trained n-gram	0.820	0.825	0.823	0.817	0.822	0.821
GloVe word + pre-trained n-gram	0.772	0.783	0.760	0.758	0.775	0.770
GloVe word (top 10000)	0.826	0.824	0.836	0.828	0.829	0.828
GloVe word (top 10000) + pre-trained n-gram	0.839	0.837	0.831	0.829	0.834	0.834
GloVe word (top 5000)	0.821	0.821	0.825	0.807	0.830	0.827
GloVe word (top 5000) + pre-trained n-gram	0.820	0.822	0.819	0.829	0.833	0.822

Table 6: F1 scores on test set for UNK words

that by simply averaging the n-gram embeddings of a word, we can boost the performance of a word-only model.

There are a few future extensions we have in mind. We are interested in training our n-gram embeddings on a larger dataset such as WikiText-103 to cover more n-grams. Moreover, we are considering using a CRF-based model. Finally, we can enhance our model structure by adding a local or global CNN layer and compare that result with our current LSTM approach.

7 References

Alexandrescu A. & Kirchhoff K. (2006). Factored Neural Language Models. *NAACL-Short '06 Proceedings of the Human Language Technology Conference of the NAACL*, Companion Volume: Short Papers Pages 1-4.

Bilmes, J.A. & Kirchhoff K. (2003). Factored Language Models and Generalized Parallel Backoff. *NAACL-Short '03 Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*. Companion volume of the Proceedings of HLT-NAACL 2003 short papers - Volume 2, Pages 4-6.

Chiu, J. P. C. & Nichols, E. (2015). Named Entity Recognition with Bidirectional LSTM-CNNs. arXiv:1511.08308.

Gillick D. & Brunk C. & Vinyals O. & Subramanya A. (2015). Multilingual Language Processing From Bytes. arXiv:1512.00103.

Hashimoto, K. & Xiong, C. & Tsuruoka, Y. & Socher, R. (2016). A Joint Many-Task Model: Growing a Neural Network for Multiple NLP Tasks. arXiv:1611.01587v3.

Kim, Y. & Rush, S. & Jernite, Y. & Sontag, D. (2015). Character-Aware Neural Language Models. arXiv:1508.06615v4.

Lample, G. & Sandeep, S. & Kawakam, K. & Dyer, Chris & Ballesteros, M. (2016). Neural Architectures for Named Entity Recognition. arXiv:1603.01360v3.

Le, Q. V. & Mikolov, T. (2014). Distributed Representations of Sentences and Documents. International Conference on Machine Learning (ICML).

Ling, W. & Luis, T. & Marujo, L. & Astudillo, R. F. & Amir, S. & Dyer, C. & Trancoso, I. (2015). Finding Function in Form: Compositional Character Models for open vocabulary word representation. Conference on Empirical Methods in Natural Language Processing (pp. 1520-1530). Lissbon: Association for Computational Linguistics.

Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. (2013). Distributed representations of words and phrases and their compositionality. In Advances in Neural Information Processing Systems. pages 31113119.

Passos, A. & Kumar, V. & McCallum, A. (2014). Lexicon Infused Phrase Embeddings for Named Entity Resolution. Conference on Natural Language Learning (CoNLL).

Pennington, Jeffrey, Richard Socher, and Christopher D Manning. (2014). Glove: Global vectors for word representation. Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014) 12.

Santos, C. N. & Guimaraes V. (2015). Boosting Named Entity Recognition with Neural Character Embeddings. arXiv:1505.05008v2.

Zhang, X. & Zhao, J. & LeCun, Y. (2015). Character-level Convolutional Networks for Text Classification. arXiv:1509.01626.