Enriching Word Vectors with Subword Information and Attention Mechanism

Alexia Wenxin Xu
alexiaxu@stanford.edu

Abstract

Continuous word representations are the key features in many natural language processing tasks. The recent model introduced by (Bojanowski et al., 2016) takes into account the internal structure of words and represents them as a bag of character n-grams. Despite the success of (Bojanowski et al., 2016), their model does not take into account the scenario when a character n-gram occurs in multiple words carrying different meanings. To allow character n-grams weigh differently in different words, we propose a model that represents a word as the sum of the attention of the n-grams as well as the word vector. We further improve the model by adding l_2-regularization and temperature to prevent early convergence to a local optimum. Our model outperforms the model in (Bojanowski et al., 2016) in human similarity judgment tasks. We also evaluate our model on analogy tasks and show it can capture both semantic and syntactic relationships between word pairs.

1 Introduction

Continuous word representations are the key features in many tasks in natural language processing, such as text classification, machine translation, and information retrieval. Back to the 1990s, word representations have been derived from statistical language modeling (Deerwester et al., 1990; Schütze, 1992). More recently, (Mikolov et al., 2013a) proposed a skip-gram model as an efficient method to learn vector representations of words from large amounts of unstructured text data. However, most of the previous models represent each word as one vector without utilizing the internal structure of the word. Without modeling relationship among morphologically related words, rare and complex words are often poorly estimated.

In 2016, (Bojanowski et al., 2016) proposed a new approach based on the skip-gram model of (Mikolov et al., 2013a) that takes into account subword information in computing word representations. It represents each word as the sum of the n-gram vectors and the word vector itself. This approach better tackles rare and out-of-vocabulary (OOV) words. It improves vector representations for not only English but also morphologically rich languages like Turkish or Finnish that contain many word forms that rarely occur in the training corpus.

Despite the success of (Bojanowski et al., 2016), there might still be room for improvement. In reality, each character n-gram may occur in multiple words, and they could weigh differently in those words. For example, "her" is a morpheme and carries significant semantic information in "herself," but not in "where." Ideally, the model should allow each n-gram to vary in significance in different words. To add a dynamic weight to each n-gram, attention mechanism (Bahdanau et al., 2014) is an intuitive solution. We will initialize a trainable bias for each n-gram and represents a word as the sum of the attention to those biases of the n-grams.

We recognize overfitting in training the model of (Bojanowski et al., 2016). To address overfitting in embedding-based models for NLP, (Peng et al., 2015) thoroughly investigated different regularization strategies. They pointed out that penalizing the ℓ_2-norm of embeddings facilitates optimization and improves training accuracy unexpectedly. Based on the results, we also add ℓ_2 regularization to our model. We evaluate our model on both the correlations with human simi-

larity judgment and the analogy tasks proposed by (Mikolov et al., 2013b), and our well-train model manages to outperform that of (Bojanowski et al., 2016) and our previous model in the attached report.

2 Related Work

Morphological Word Representations In the past five years, people have tried various approaches to incorporate subword information in word representations. To model rare words better, (Alexandrescu and Kirchhoff, 2006) introduced factored neural language models, where words are represented as sets of features. These features might include morphological information, and this technique was successfully applied to morphologically rich languages, such as Turkish (Sak et al., 2010).

There are also works proposing different composition functions to derive representations of words from morphemes. (Luong et al., 2013) proposed to train recursive neural networks to build word vectors from the morpheme vectors. Concretely, they combine recursive neural networks (RNNs), where each morpheme is a basic unit, with neural language models (NLMs) to consider contextual information in learning morphologically aware word representations. (Qiu et al., 2014) trained a CBOW model that represents context words as a weighted sum of a bag of words and a bag of morphemes. In their model, the morphological knowledge plays as both additional input representation and auxiliary supervision to the neural network framework. (Botha and Blunsom, 2014) summed up morphological representations to obtain a word vector and output a vocabulary class before predicting an exact word. In their model, word vectors are composed as a linear function of arbitrary sub-elements of the word, e.g., surface form, stem, affixes, or other latent information. The logic is that morphologically related words should share statistical strength despite the differences in surface form. (Cao and Rei, 2016) built a model that takes a sequence of characters as the input to an LSTM and generates the word vector using the attention of the hidden units. The model is built on the intuition that the words with the same stem have similar contexts. Their model splits individual words into segments and weighs each segment according to its ability to predict context words.

Our model is mainly based on that of (Bojanowski et al., 2016), who proposed a novel skip-gram based approach. The model represents each word as the sum of the n-gram vectors and the word vector itself. By taking into account the subword information, it improves vector representations for not only English but also morphologically rich languages like Turkish or Finnish. Those languages typically contain many word forms that rarely occur in the training corpus. We will combine the model of (Bojanowski et al., 2016) with attention mechanism and aim to achieve better performance.

Attention Mechanism Attention mechanism has become an integral part of NLP models for machine translation and dialog agents (Bahdanau et al., 2014). It allows modeling of dependencies without regard to their distance in the input or output sequences. One of the most intriguing results is published by (Vaswani et al., 2017). They proposed a Transformer architecture that eschews recurrence and entirely relies on an attention mechanism to draw global dependencies between input and output. In this project, we will adopt attention mechanism to achieve dynamic weights of character n-grams in different words.

Regularizing Embedding-based Models Embedding-based models for various NLP tasks usually suffer from severe overfitting because of the high dimensionality of parameters. (Peng et al., 2015) explored multiple strategies to regularize embedding-based NLP models to improve generalization. Their conclusions include: (1) The effect of regularization depends on the datasets size. (2) ℓ_2-regularization of embeddings helps with optimization and improves training accuracy. (3) Dropout is not as effective as ℓ_2 regularization, but it can act as a complement for insufficient regularization.

Previous Project Last quarter, we tried to build a model to learn word representations using only subword information (see attached report). In the model, each word vector is computed using the attention of the n-gram representations without using the intact word. Our model achieves a comparable performance to that of (Bojanowski et al., 2016), showing that word vectors are not indispensable in capturing the semantic meaning of the words. However, we were not able to improve the performance enough to pass the significance test.

In this project, we will employ the results of

(Bojanowski et al., 2016) as the baseline, analyze the results of the baseline and try to improve its performance by adding new modules to the model.

3 Datasets

Training word embedding We trained our model on enwik8 (also called text8)[1]. enwik8 contains 17M words, 253854 unique words, and 71290 unique frequent words. We normalize enwik8 using Matt Mahoney's preprocessing perl script [2]. The dataset is shuffled before training. We use enwik8 instead of larger datasets such as enwik9 or Wikipedia dumps because we found that the relative performance of models on the three datasets are pretty consistent. Larger datasets will just boost the performance of all models (see Figure 2 of the attached previous report). Due to our limited time and computing resources, we decide to use enwik8 to estimate the relative performance of our model to other models.

Evaluation We evaluate the quality of our representations with two approaches: 1) computing the Spearmans rank correlation coefficient between human similarity judgment; 2) answering analogy questions, of the form A is to B as C is to D, where C must be predicted by the models. In the similarity judgment task, we use the WS353 dataset introduced by (Finkelstein et al., 2001) and the rare word dataset (RW), introduced by (Luong et al., 2013). The two datasets are quite different in terms of the complexity of words. WS353 contains more common words such as $< love, sex >$, $< tiger, cat >$, etc, while RW contains more complex/rare words such as $< squishing, squirt >$, $< undated, undatable >$, etc. Typically, the rare word dataset better evaluates the model in understanding morphemes. In the analogy task, we use the datasets introduced by (Mikolov et al., 2013b) in English.

4 Models

In this section, we will describe how our model learns word representations by gathering morphological information in subwords. In the first subsection, we will show the general framework and the error metric in training word embeddings. In the second subsection, we will present the attention-based model to generate word vectors from subwords.

[1]http://mattmahoney.net/dc/enwik8.zip
[2]http://mattmahoney.net/dc/textdata.html

4.1 General Framework

The model described in this subsection is the skig-gram model with negative sampling proposed in (Mikolov et al., 2013a). We train a skip-gram model to learn word representations, where each word vector is trained to predict the words in its context. Given a corpus with word sequence $\{w_1, w_2, ..., w_T\}$, the objective of the skip-gram model is to maximize the following likelihood:

$$\sum_{t=1}^{T} \sum_{c \in \mathcal{C}_t} logp(w_c|w_t),$$

where $\mathcal{C}_t$ is the context of word w_t, which consists of indices of words surrounding w_t. Assuming we have a similarity function $s(w_t, w_c) \in \mathbb{R}$ that maps the two word vectors to a scalar score, we could define $p(w_c|w_t)$ as follows:

$$p(w_c|w_t) = \frac{e^{s(w_t,w_c)}}{\sum_{j=1}^{W} e^{s(w_t,j))}}$$

However, there are two problems using softmax as the objective function: 1) each center word has more than one context words; 2) to compute the softmax, it needs to calculate the probability of each word in the vocabulary, which is computationally inefficient. In practice, we use negative sampling loss as the objective function in learning word representations. We aim to maximize the score of true context words, while minimizing that of the random negative samples. Thus we have the following loss function:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \left[\sum_{c \in \mathcal{C}_t} \ell(s(w_t, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t, n)) \right],$$

where $\mathcal{N}_{t,c}$ is a random set of negative examples sampled from the vocabulary, and $\ell(x) = log(1 + e^{-x})$. Each word has two vectors. u_{w_t} is the input vector that represent the word when it is the center, and v_{w_c} is the output vector used when the word is in the context. In [1], $s(w_t, w_c) = \boldsymbol{\mu}_{w_t}^{\top} \boldsymbol{\nu}_{w_c}$, while we will re-define $s(w_t, w_c)$ by inducing subword information and attention in the next subsection.

When observing overfitting during training the model, we could also add ℓ_2 regularization to penalize the norm of $\boldsymbol{\mu}_{w_t}$ as suggested by (Peng et al., 2015). With a weight decay α, our final objective becomes:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \left[\sum_{c \in \mathcal{C}_t} \ell(s(w_t, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t, n)) \right] + \alpha \|\boldsymbol{\mu}_{w_t}\|_2^2$$

4.2 Attention-based Subword Model

(Bojanowski et al., 2016) proposed to represent a word w by summing up its word vector with its n-gram subword vectors. They hope to take into account the internal structure of the words by using subword information. We believe their model still has room for improvement because the character n-gram subwords of a word vary in significance in representing the word. "kind" in "kindness", for example, carries more semantic information than "ndnes". Similarly, one character n-gram could occur in multiple words in the vocabulary, and it could weigh differently in representing different words. In this section, we will propose an attention-based subword model to address these issues.

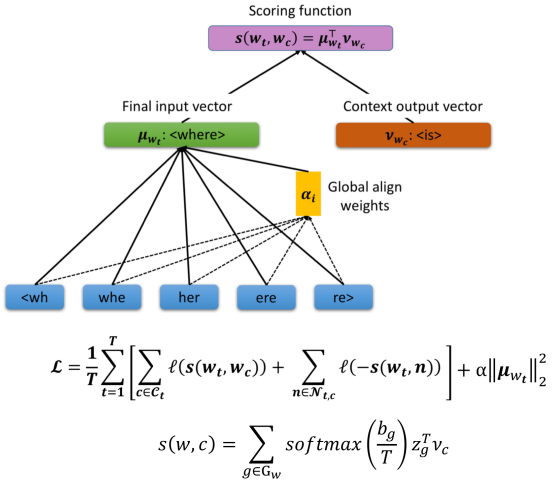

Figure 1: The attention-based skip-gram model with negative sampling, assuming only 3-gram subwords are used. The blue boxes show all character 3-grams of the word $\langle where \rangle$

In our model, each word w is represented as the sum of the attention over its character n-grams as shown in Figure 1. Similar to (Bojanowski et al., 2016), we add special symbols $\langle$ and $\rangle$ at the beginning and the end of each word. These symbols help distinguish the word $\langle her \rangle$ from the "her" in $\langle inherit \rangle$. In practice, we extracted all the character n-grams ($n \in [3, 6]$) plus the word vector itself in computing the input vector $\boldsymbol{\mu}_{w_t}$.

Our model includes a parameter, called bias, for each n-gram to address their different contribution to a word. Given a word w, let's denote the set of n-grams ($n \in [3, 6]$) in w by $\mathcal{G}_w$, the vector representing n-gram $g \in \mathcal{G}_w$ by z_g, and the bias of g by b_g. We define the scoring function as:

$$s(w, c) = \sum_{g\in\mathcal{G}_w} softmax(\frac{b_g}{T}) \boldsymbol{z}_g^\top \boldsymbol{\nu}_c$$

Here T is a hyper-parameter, temperature, that facilitates optimization and prevents the model from falling into a local optimum too early. The temperature decays linearly as follows:

$$T = (T_{max} - 1)(1 - progress) + 1,$$

where $progress \in [0, 1]$. Please refer to Section 6.4 for more details of the motivation to add the temperature. The attention module enables sharing n-grams across words while weighing them differently.

The definition of the bias z_g is the key point of the project. We will try different values and initializations of the bias z_g as shown in Section 6.

5 Experimental Setup

To compare with (Bojanowski et al., 2016), we adopt the same experimental setup. In unsupervised tasks, the dimension of the word representations is 300. The sizes of the n-grams range from 3 to 6. In negative sampling, 5 negative examples are sampled at random for each positive example, with probability proportional to the square root of the uni-gram frequency. When building the word dictionary, we keep the words that appear at least 5 times in the training set. The ceiling of the hash function is 2×10^6. We solve the optimization problem by performing stochastic gradient descent on the loss function, and use a linear decay of the step size. Initialization and the values of hyper-parameters vary in different experiments. Please refer to the corresponding subsections for more settings. The model is implement with C++.

6 Results

We first conduct experiments that facilitate us to interpret the training of the baseline model and answer the following questions:

- Did (Bojanowski et al., 2016) train the character n-grams sufficiently
- Did their model fall into bad local minima and can be improved by locally adjust the word vectors? If so, we may be able to improve its performance via better optimization

Epochs	Training Loss	RW	WS353
1	1.217	39.97	62.05
3	0.709	41.00	64.31
5	0.221	39.69	61.15

Table 1: The performance of the baseline model

- How much will the initialization of bias affect the performance

With conclusions on the previous questions, we start to tune the hyper-parameters of the model and shoot for the best performance

6.1 Baseline

We trained the model of (Bojanowski et al., 2016) on enwik8 without changing the settings in the original paper. Then we evaluate the model on rare word (RW) and WS353 datasets respectively. We trained the model for 1, 3, and 5 epochs, respectively ((Bojanowski et al., 2016) trained the model for 5 epochs). The performance of the baseline is shown in Table 1.

Table 1 shows a clear overfitting in the setting of (Bojanowski et al., 2016) and early stopping helps mitigate the problem. We also print out the rank of significance of the character n-grams calculated from the baseline model. Surprisingly, we find that the baseline model do not put too much weights on the word vectors. The five most significant n-grams of a complex word are typically long, uncommon subwords without including the word vector. Nevertheless, the model of (Bojanowski et al., 2016) usually fails to find the morphemes as the most significant n-gram. Given the observation, we will not remove the intact words in calculating the word vectors.

6.2 Fixed N-gram Weights

We also want to test if there exists a set of bias that could improve the performance when using the pre-trained representations of (Bojanowski et al., 2016). If so, should the "significant" or the previously omitted character n-grams weight more? To address the problem, we fix the weights w_g of the n-gram g to the following values: $< subword, word >$, $cos(subword, word)$, -$< subword, word >$, and -$cos(subword, word)$, respectively, and then evaluate the models on the RW and WS353 dataset. We tuned the hyper-parameters for each model to reach the best performance.

The results are shown in Table 2. We may observe that the performance of the baseline model could be improved by adding a weight $w_i = -cos(subword, word)$ to each n-gram and representing the final word vector as the weighted sum of all character n-grams. It indicates that the baseline model is at some bad local optimum and upweighting the previously unimportant n-grams improves the model.

6.3 Initializing Bias by N-gram Counts

As mentioned in Subsection 6.1, the baseline model tends to view long and uncommon character n-grams as "significant" instead of the real morphemes. To force the model to look at the morphemes, we try to penalize the uncommon character n-grams by initializing the bias of n-gram g as follows:

$$b_g = \beta log(n_g + \epsilon),$$

where n_g is the total counts of n-gram g in the corpus, and $\epsilon = 0.0001$. n_g is in the range of [5, 585379], and a statistics of the n-gram counts in the range of [1000, 12500] is shown in Figure 2. We experiment with different β values and show the results in Table 3. We observe that penalizing the infrequent n-grams could further improve the performance.

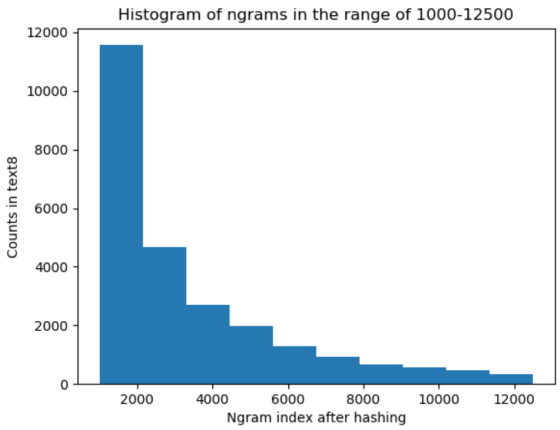

Figure 2: Histogram of the counts of n-grams in enwik8

6.4 Effects of Temperature

Now we have acquire enough understanding on the model of (Bojanowski et al., 2016) and are

	$< subword, word >$	$cos(subword, word)$	-$< subword, word >$	-$cos(subword, word)$
RW	39.3	39.5	41.0	41.3
WS353	63.2	63.4	65.3	65.8

Table 2: Evaluation Results of Fixed N-gram Weights

Experiment	β	RW	WS353
1	0.1	62.9	38.3
2	0.05	67.2	41.2
3	0.01	**67.5**	**41.5**
4	7.5e-3	67.4	41.4
5	5e-3	67.3	41.4

Table 3: Experiment with different β value

ready to train our own model in Section 4.2. We adopt a weight decay of $\alpha = 10^{-4}$ and initialize the $300 - dimensional$ input vectors with a uniform distribution ($\boldsymbol{\mu}_w \sim \mathcal{U}(0, \frac{1}{300})$). All other weights are initialized to zeros since the random input vectors are enough to break the symmetries. Learning rate decays linearly during training.

During training the model, we found that the model falls into a bad local optimum very early. The bias of some character n-gram are optimized to large values after a few iterations. Since the effective weight of a character n-gram is calculated by $softmax(b_i)$, the ones have not occurred and updated yet will never have a chance to be updated again in presence of those updated large biases. To mitigate the problem, we introduce a new hyper-parameter, temperature, to flatten the distribution of biases at the early stage of training.

Table 4 shows the effects of different temperature on the performance on the Human Similarity Judgment. We will fix temperature T_{max} to 25.0 for future experiments.

T_{max}	RW	WS353
10	64.2	41.2
20	66.1	42.2
25	**67.8**	**43.8**
50	63.1	40.7

Table 4: Effects of Temperature on the Human Similarity Judgment

6.5 Training

At the current stage, all hyper-parameters have been fixed. To determine if our model still overfits the dataset and if early stopping is necessary, we train the mode for 15 epochs and print out the learning curve (as shown in Figure 3). From the learning curve, we learn that the negative sampling training loss and the similarity score on both the RW and WS353 datasets are highly correlated. The loss is still decreasing after training for 15 epochs, indicating that longer training might further improve the model. No obvious overfitting is observed, showing that the regularization coefficient we pick is highly effective.

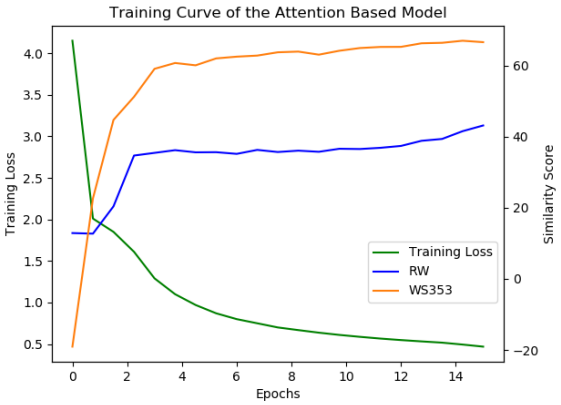

Figure 3: Learning curve of the attention based model

6.6 Human Similarity Judgment

In this subsection, we will compare different models based on their performance on the human similarity judgment tasks. We evaluate the model by computing the Spearmans rank correlation coefficient between human similarity judgment and the cosine similarity between the vector representations. Both the rare word dataset (RW) and the WS353 dataset is employed to cross-validate each other. We also compared our model with the previous works on word vectors incorporating subword information on word similarity tasks. The methods used are: the recursive neural network

Table 5: Correlation Between Human Judgment & Similarity Scores

	WS353	Rare Word
(Luong et al., 2013)	64	34
(Qiu et al., 2014)	65	33
(Bojanowski et al., 2016)	64.3	41.0
Xu report cs224n	65.8	42.5
Attn+Subword	**67.8**	**43.8**

of (Luong et al., 2013), the morpheme cbow of (Qiu et al., 2014) and the calibrated model of (Bojanowski et al., 2016). We also included the attention based model from our report in last quarter. The results are shown in Table 5, where our model outperforms all other models.

In generating this table, the model of (Bojanowski et al., 2016) was retrained on enwik8. The model in the original paper is trained on wikipedia dumps with much better performance. The model from my previous report (Xu report cs224n) was also retrained on enwik8. It was previously trained on enwik9. The model of (Qiu et al., 2014) was not calibrated and was trained on a wikipedia dump. We can imagine its performance could be much worse if retrained on enwik8.

From Table 5, we could reach the conclusions that: 1) Our attention-based model outperforms the model of (Bojanowski et al., 2016) and the model from my previous report. 2) Regularization and temperature help notably improve the optimization. 3) The model takes long time to get fully trained. Increasing the size of the training corpus would improve the performance significantly (also refer to Table 1 of the attached previous report).

6.7 Qualitative Analyis: Analogy Tasks

We also evaluate our attention-based model on word analogy questions, of the form A is to B as C is to D, where C must be predicted by the models. The evaluation scheme based on word analogies is proposed in (Mikolov et al., 2013b). By examining the various dimensions of difference between word vectors, we could peek the finer structure of the word vector space. The most famous example is probably the analogy king is to queen as man is to woman. It should be encoded in the vector space by the vector equation $king - queen = man - woman$. This evaluation scheme favors models that produce dimensions of meaning and reflects the multi-clustering idea of distributed representations.

Table 6 shows a few positive results of the analogy task and negative results with the same relationships. The first three rows are semantic relationships including: the capital of a country, the capital of a state and the terms of men and women of different occupations. The fourth row is about comparative, a syntactic relationship. From the positive examples, we can see that the model is capable of capturing the listed semantic relationships. It indicates that the negative examples with the same relationships may be caused by insufficient context of those words in the training corpus. We conjecture that the performance of the model on the analogy task can be improved significantly by training on the most recent Wikipedia dump. However, due to limited computing resource, we did not have the chance to try it.

7 Conclustion and Discussion

In this project, we build a model incorporating attention mechanism with subword information to learn word representations. Our word vectors were computed using the attention of the n-gram representations. Our model outperforms (Bojanowski et al., 2016) in unsupervised tasks and gained higher correlation with human similarity judgment. We show that both l_2-regularization and temperature improves the optimization by preventing the model from falling into a local optimum in an early stage of training. We also evaluate our model on the analogy task proposed by (Mikolov et al., 2013b) and show that our model is able to capture both semantic and syntactic relationships between words.

We mainly reach three conclusions through the experimental results: 1) It improves the subword model of (Bojanowski et al., 2016) notably to allow character n-grams to weight differently in different words. 2) Adding l_2-regularization and temperature can further improve the subword model through better optimization. 3) The size of training corpus affects the performance a lot. There is a trade-off between computation time and performance in our project.

For potential next steps, we hope to push the limit to see how much improvement can be achieved by tuning hyper-parameters and changing optimization strategies. Figure 3 shows that

Table 6: Positive and negative examples of the analogy tasks: C is predicted

(a) Postive examples

A	B	C	D
glendale	california	denver	colorado
athen	greece	cairo	egypt
sons	daughters	policeman	police-women
fast	faster	loud	louder

(b) Negative examples

A	B	C	D
akron	ohio	angeles	california
athen	greece	zricher	zrich
king	queen	waitressed	waitress
quick	quicker	cool	hotter

the loss decreases much more slowly after training for 3 epochs. We assume this is a sign that learning rate annealing could facilitate the optimization.

Notes

- Since this project is based a previous course project, I attach the previous report at the end
- Our Github repository is currently private since we want to further polish the results to obtain a publishable paper. We will open source the code when we finish training on the Wikipedia Dumps and post it on Arxiv. For now, we will submit our code by sharing a google drive link to a zip file. [3]
- Everything is implemented in C++ upon the code of (Bojanowski et al., 2016).

References

Andrei Alexandrescu and Katrin Kirchhoff. 2006. Factored neural language models. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 1–4. Association for Computational Linguistics.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information. *arXiv preprint arXiv:1607.04606*.

Jan Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *International Conference on Machine Learning*, pages 1899–1907.

Kris Cao and Marek Rei. 2016. A joint model for word embedding and word morphology. *arXiv preprint arXiv:1606.02601*.

[3] https://drive.google.com/file/d/1o2rikMCqszNnKuBDYlWXMsudM1cPG97r/view?usp=sharing

Scott Deerwester, Susan T Dumais, George W Furnas, Thomas K Landauer, and Richard Harshman. 1990. Indexing by latent semantic analysis. *Journal of the American society for information science*, 41(6):391.

Lev Finkelstein, Evgeniy Gabrilovich, Yossi Matias, Ehud Rivlin, Zach Solan, Gadi Wolfman, and Eytan Ruppin. 2001. Placing search in context: The concept revisited. In *Proceedings of the 10th international conference on World Wide Web*, pages 406–414. ACM.

Thang Luong, Richard Socher, and Christopher Manning. 2013. Better word representations with recursive neural networks for morphology. In *Proceedings of the Seventeenth Conference on Computational Natural Language Learning*, pages 104–113.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013a. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. 2013b. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751.

Hao Peng, Lili Mou, Ge Li, Yunchuan Chen, Yangyang Lu, and Zhi Jin. 2015. A comparative study on regularization strategies for embedding-based neural networks. *arXiv preprint arXiv:1508.03721*.

Siyu Qiu, Qing Cui, Jiang Bian, Bin Gao, and Tie-Yan Liu. 2014. Co-learning of word representations and morpheme representations. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pages 141–150.

Haşim Sak, Murat Saraclar, and Tunga Güngör. 2010. Morphology-based and sub-word language modeling for turkish speech recognition. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 5402–5405. IEEE.

Hinrich Schütze. 1992. Dimensions of meaning. In *Proceedings of the 1992 ACM/IEEE conference on*

Supercomputing, pages 787–796. IEEE Computer Society Press.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 6000–6010.

Representing Words with Only Subword Information

Alexia Wenxin Xu
Department of Computer Science
alexiaxu@stanford.edu

Abstract

Continuous word representations are the key features in many natural language processing tasks. The recent model introduced by Bojanowski *et al.* [2] takes into account the internal structure of words and represents them as a bag of character n-grams. There are two possible limitations in the model: 1) Using the n-grams, word vectors may be redundant representing a word. In addition, the existence of the word vectors may prevent the n-gram vectors from being fully trained 2) Each character n-gram may occur in multiple words, and they could weigh differently in different words. To address the issues, we propose a model that represents a word as the sum of the attention of the n-grams without using the word vector. Our word representation outperforms the model in [2] in unsupervised tasks and is on par with it in text classification. We also show how a well-trained model segments the words and judges the importance of n-grams.

1 Introduction

Continuous word representations are the key features in many tasks in natural language processing, such as text classification, machine translation, and information retrieval. Back to the 1990s, word representations have been derived from statistical language modeling [12, 13]. More recently, Mikolov *et al.* [1] proposed a skip-gram model as an efficient method to learn vector representations of words from large amounts of unstructured text data.

Most of the previous models represent each word as one vector without utilizing the internal structure of the word. Bojanowski *et al.* [2] proposed a model that learns representations for character n-grams. It represents each word as the sum of the n-gram vectors and the word vector itself. By taking into account the subword information, it improves vector representations for not only English, but also morphologically rich languages like Turkish or Finnish that contain many word forms that rarely occur in the training corpus.

In this paper, we propose to improve the model of Bojanowski *et al.* [2] based on two ideas: 1) Word vectors may be redundant in existence of the character n-grams. N-grams should carry sufficient information to represent the words. When the word vectors present, the n-gram vectors may not be fully trained. 2) Each character n-gram may occur in multiple words, and they could weigh differently in those words. For example, ”her” carries significant semantic information in ”herself”, but not in ”inherit”. Ideally, the model should allow an n-gram to vary in significance in different words. To address the two issues, we built a model that represents a word as solely the sum of the attention of the n-grams. To be consistent with [2], we also evaluate our model on both the correlations with human similarity judgment and text classification tasks. We will also show how a well-trained model segments the words and judges the importance of the n-grams.

2 Related Work

There have been many successful previous works on training word embedding. The creation of word2vec [14] allows large improvements in accuracy at much lower computational cost. A year later, Pennington *et al.* [15] introduced GloVe to us, which is a competitive set of pre-trained embeddings that combines the advantages of the two major model families: global matrix factorization and local context window methods. In recent years, there are also works proposing to incorporate morphological information into word representations. [6], [7], and [16] presented approaches to derive representations of words from morphemes.

In 2017, Bojanowski *et al.* [2] proposed a new approach based on the skip-gram model [1], where each word is represented as a bag of character n-grams. Joulin *et al.* [3] built a fast text classifier (fastText) based on the skip-gram model of [2]. fastText is often on par with deep learning classifiers in terms of accuracy and is many orders of magnitude faster for training and evaluation.

Our model was built upon the model in [2] by representing words using only character n-grams. Our model outperforms the model in [2] in correlations with human judgment on word similarity and is competitive with fastText in sentiment analysis.

3 Approach

In this section, we will describe how our model learns word representations by gathering morphological information in subwords. In the first subsection, we will show the general framework and the error metric in training word embeddings. In the second subsection, we will present the attention-based model to generate word vectors from subwords. At last, we will present the modifications to the model in training supervised tasks.

3.1 General Framework

The model described in this subsection is the skig-gram model with negative sampling proposed in [1] and [2]. We trained a skip-gram model to learn word representations, where each word vector is trained to predict the words in its context. Given a corpus with word sequence $\{w_1, w_2, ..., w_T\}$, the objective of the skip-gram model is to maximize the following likelihood:

$$\sum_{t=1}^{T} \sum_{c \in \mathcal{C}_t} logp(w_c|w_t),$$

where $\mathcal{C}_t$ is the context of word w_t, which consists of indices of words surrounding w_t. Assuming we have a similarity function $s(w_t, w_c) \in \mathbb{R}$ that maps the two word vectors to a scalar score, we could define $p(w_c|w_t)$ as follows:

$$p(w_c|w_t) = \frac{e^{s(w_t,w_c)}}{\sum_{j=1}^{W} e^{s(w_t,j))}}$$

However, there are two problems using softmax as the objective function: 1) each center word has more than one context words; 2) to compute the softmax, it needs to calculate the probability of each word in the vocabulary, which is computationally inefficient. In practice, we use negative sampling loss as the objective function in learning word representations. We aim to maximize the score of true context words, while minimizing that of the random negative samples. Thus we have the following loss function:

$$\mathcal{L} = \frac{1}{T} \sum_{t=1}^{T} \Bigg[\sum_{c \in \mathcal{C}_t} \ell(s(w_t, w_c)) + \sum_{n \in \mathcal{N}_{t,c}} \ell(-s(w_t, n)) \Bigg],$$

where $\mathcal{N}_{t,c}$ is a random set of negative examples sampled from the vocabulary, and $\ell(x) = log(1 + e^{-x})$. Each word has two vectors. u_{w_t} is the input vector that represent the word when it is the center, and v_{w_c} is the output vector used when the word is in the context. In [1], $s(w_t, w_c) = \boldsymbol{\mu}_{w_t}^{\top} \boldsymbol{\nu}_{w_c}$, while we will re-define $s(w_t, w_c)$ by inducing subword information and attention in the next subsection.

3.2 Attention-based Subword Model

Bojanowski *et al.* [2] proposed to represent a word w by summing up its word vector with its n-gram subword vectors. They hope to take into account the internal structure of the words by using subword information. We believe that the n-gram subwords carry enough information to represent a word both syntactically and semantically. Thus the word vector of w itself is not necessary in the model. Another important point is that the n-grams of a word vary in significance in representing the word. "kind" in "kindness", for example, carries more semantic information than "ndnes". Similarly, one n-gram could occur in multiple words in the vocabulary, and it could weigh differently in representing different words. In this section, we will propose an attention-based subword model to address these issues.

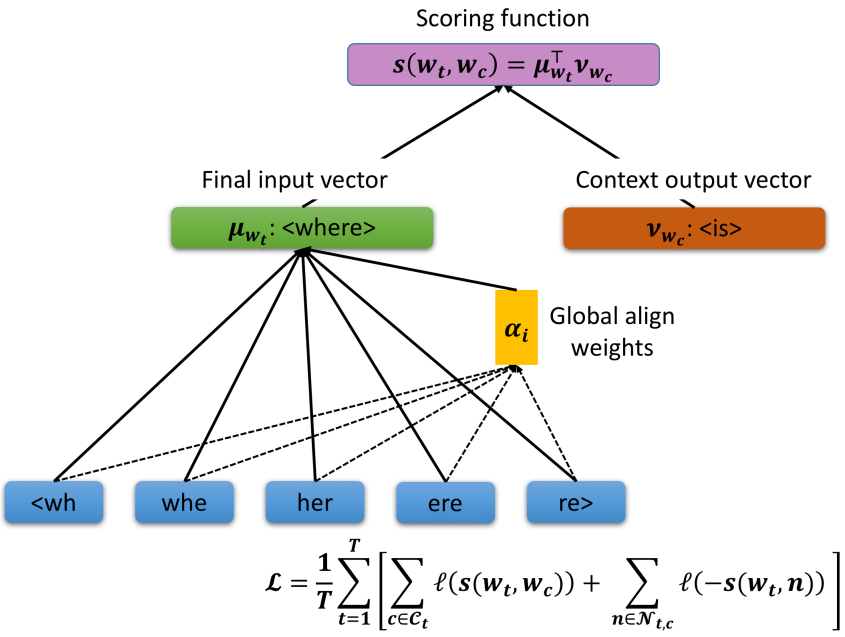

Figure 1: The attention-based skip-gram model with negative sampling, assuming only 3-gram subwords are used. The blue boxes show all character 3-grams of the word $\langle where \rangle$

As shown in Figure 1, each word w is represented as the sum of the attention over its character n-grams in our model. Similar to [2], we add special symbols $\langle$ and $\rangle$ at the beginning and the end of each word. These symbols help distinguish the word $\langle her \rangle$ from the "her" in $\langle inherit \rangle$. In practice, we extracted all the character n-grams ($n \in [3, 6]$) in computing the input vector $\boldsymbol{\mu}_{w_t}$.

Our model includes a trainable parameter, called bias, for each n-gram to address their different contribution to a word. Given a word w, let's denote the set of n-grams ($n \in [3, 6]$) in w by $\mathcal{G}_w$, the vector representing n-gram $g \in \mathcal{G}_w$ by z_g, and the trainable bias of g by b_g. We define the scoring function as:

$$s(w, c) = \sum_{g \in \mathcal{G}_w} softmax(b_g) \boldsymbol{z}_g^\top \boldsymbol{\nu}_c,$$

The attention module enables sharing n-grams across words while weighing them differently.

As in [2], we use a hash function that maps n-grams to integers in $[1, 2 \times 10^6]$ to bound the memory. In this setting, each word is represented as a list of n-gram indices.

3.3 Text Classification

We also tested our model on a supervised task, sentiment analysis, as in [3]. In sentiment analysis, we represent a sentence as the mean of the bigrams it contains, which means every pair of adjacent words in the sentence, and denote the sentence vector as $\boldsymbol{\mu}_s$. We denote the set of the bigrams by $\mathcal{G}_s$. To embed a bigram, we first transform it into a set of character n-grams, denoted by $\mathcal{G}_c$. Then we represent the word bigram as the sum of the attention of the character n-grams, and denote the bigram vector by $\boldsymbol{\mu}_{bigram}$. Labels are also embedded as vectors, denoted by $\boldsymbol{\nu}_L$. The scoring function of each (sentence, label) pair is calculated as the dot product of the two vectors. We use softmax cross-entropy as the loss function:

$$\mathcal{L} = CE(\boldsymbol{y}, softmax(\boldsymbol{\mu}_s^\top \boldsymbol{\nu}_L)),$$

where

$$\boldsymbol{\mu}_s = \frac{1}{|\mathcal{G}_s|} \sum_{bigram \in \mathcal{G}_s} \boldsymbol{\mu}_{bigram},$$

$$\boldsymbol{\mu}_{bigram} = \sum_{g \in \mathcal{G}_c} softmax(b_g) \boldsymbol{z}_g$$

All models were trained asynchronously on multiple CPUs using stochastic gradient descent and a linearly decaying learning rate.

4 Experiments

4.1 Datasets

Training word embeddings We trained our model respectively on enwik8 (also called text8)[1], enwik9 (also called text9)[2], and the English Wikipedia data released by Shaoul and Westbury (2010) [4]. We normalize all datasets using Matt Mahoney's preprocessing perl script [3]. All the datasets are shuffled. We evaluate the quality of our representations on the task of word similarity / relatedness. In evaluation, we use the WS353 dataset introduced by Finkelstein *et al.* [5] and the rare word dataset (RW), introduced by Luong *et al.* [6].

Text classification We employ the same 8 datasets and evaluation protocol as [3], which are the Ag News, Sogou News, DBpedia, Yelp Review Polarity, Yelp Review Full, Yahoo Answers, Amazon Review Full, and Amazon Review Polarity.

4.2 Experimental Setup

To compare with [2], we adopt the same experimental setup. In unsupervised tasks, the dimension of the word representations is 300. The sizes of the n-grams range from 3 to 6. In negative sampling, 5 negative examples are sampled at random for each positive example, with probability proportional to the square root of the uni-gram frequency. When building the word dictionary, we keep the words that appear at least 5 times in the training set. The ceiling of the hash function is 2×10^6. We solve the optimization problem by performing stochastic gradient descent on the loss function, and use a linear decay of the step size. Each dataset is trained for 5 epochs. In text classification tasks, we set the dimension of the vectors to 10. The initial learning rate of the eight datasets were $\{0.25\ 0.5\ 0.5\ 0.1\ 0.1\ 0.1\ 0.05\ 0.05\}$.

5 Results

For simplicity, we will refer to the model of [2] and [3] as fastText (the name of the library based on [2] and [3]) and our attention-based model as fastAttn in this section.

[1] http://mattmahoney.net/dc/enwik8.zip

[2] http://mattmahoney.net/dc/enwik9.zip

[3] http://mattmahoney.net/dc/textdata.html

5.1 Effect of the Corpus Size

In this subsection, we explore the effect of the size of the training sets. By training on a large set, n-gram vectors capture the semantics and syntax more precisely. On the other hand, training becomes much slower. To balance between training time and quality, we trained our model on enwik8 (17M words), enwik9 (124M words), and the Shaoul Wikipedia (990M words) for 5 epochs, respectively. The result is shown in Figure 2. We evaluate the models on the rare word dataset (RW) by calculating the correlation between human judgment and similarity scores. We observe that the performance of the model trained on enwik9 is similar to that on wikipedia (the correlation score is ~ 45 vs. ~ 47 on wikipedia), but it takes less time to train. We will use the models trained on enwik9 to generate the results in Section 5.2 and Section 5.4.

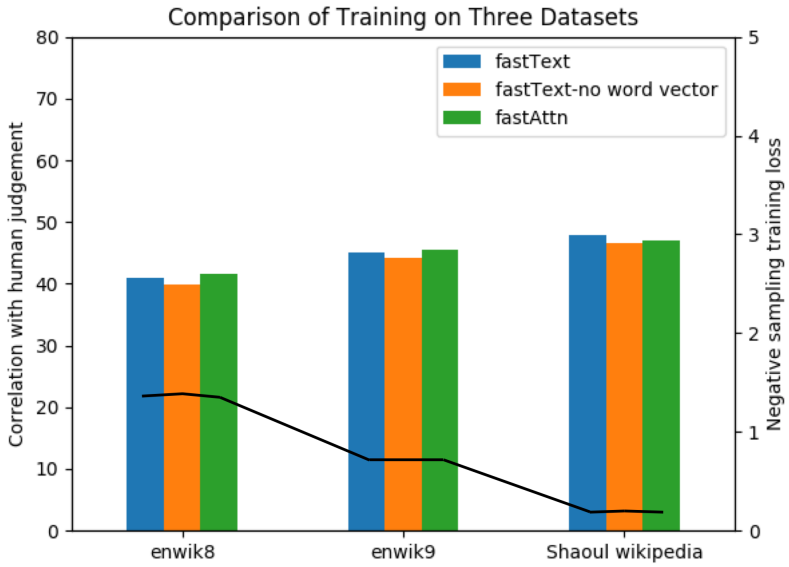

Figure 2: Comparison of training on enwik8, enwik9, and the Shaoul Wikipedia. The bars show the Spearmans rank correlation coefficient between human judgement and simlarity scores computed from different models. The black line show the negative sampling training loss of the models on the three datasets

5.2 Human Similarity Judgement

In this subsection, we will show that subword representation is sufficient to carry the semantic meaning of the words. To compare with Bojanowski *et al.* [2], we similarly evaluate our word representations on the task of word similarity / relatedness. We evaluate the model by computing the Spearmans rank correlation coefficient between human similarity judgement and the cosine similarity between the vector representations. Both the rare word dataset (RW) and the WS353 dataset is employed to cross-validate each other. We also compared our model with the previous works on word vectors incorporating subword information on word similarity tasks. The methods used are: the recursive neural network of Luong et al. (2013) [6], the morpheme cbow of Qiu et al. (2014) [7] and the morphological transformations of Soricut and Och (2015) [8]. The results are shown in Table 1, where our model outperforms that of Bojanowski *et al.* [2].

5.3 Text Classification

We also evaluate our model and compare it to fastText and other existing text classifiers on sentiment analysis tasks. We employ the same 8 datasets and evaluation protocol as in [3]. Other models that we compare with are the baseline and the character level convolutional model (char-CNN) of Zhang and LeCun (2015) [9], the character based convolution recurrent network (char-CRNN) of (Xiao and Cho, 2016) [10] and the very deep convolutional network (VDCNN) of Conneau et al. (2016)

Table 1: Correlation Between Human Judgment & Similarity Scores

	WS353	Rare Word
Luong et al. (2013)	64	34
Qiu et al. (2014)	65	33
Soricut and Och (2015)	71	42
fastText	71.8	45.0
fastText-no word vector	71.3	44.7
fastAttn	**73.6**	**45.4**

Table 2: Accuracy on Sentiment Datasets

Model	AG	Sogou	DBP	Yelp P.	Yelp F.	Yah.A.	Amz.F.	Amz.P.
ngrams TFIDF (Zhang et al., 2015)	92.4	97.2	98.7	95.4	54.8	68.5	52.4	91.5
char-CNN (Zhang et al., 2015)	87.2	95.1	98.3	94.7	62.0	71.2	59.5	94.5
char-CRNN (Xiao and Cho, 2016)	91.4	95.2	98.6	94.5	61.8	71.7	59.2	94.1
VDCNN (Conneau et al., 2016)	91.3	96.8	98.7	95.7	64.7	73.4	63.0	95.7
fastText	92.3	96.8	98.6	95.7	63.9	72.5	60.3	94.6
fastAttn	92.4	96.3	97.9	95.9	62.2	71.8	61.4	94.5

[11]. Table 2 shows that the performance of our model is comparable to fastText, the char-CNN, and the char-CRNN models, but worse than the VDCNN. However, since no neural networks are involved, our model should train much faster than VDCNN. We used word bigrams to train both fastText and our fastAttn model. If we use trigrams, the performance of both models will increase. Table 2 indicates that removing word vectors from fastText still allows the model to achieve similar performance.

5.4 Qualitative analysis

Most significant character n-grams In a view of linguistics, it is interesting to show how a well trained model weighs the character n-grams, and if the most significant n-grams correspond to morphemes. We will quantify the significance of the n-grams using a modified method of [2]. We define the significance of an n-gram g as follows:

$$sig_g = cos(\boldsymbol{\mu}_w, \boldsymbol{\mu}_{w\setminus g}),$$

where $\boldsymbol{\mu}_{w\setminus g}$ is the the restricted representation obtained by omitting n-gram g in the word vector

$$\boldsymbol{\mu}_{w\setminus g} = \boldsymbol{\mu}_w - softmax(b_g)\boldsymbol{z}_g,$$

We picked some words that contains clear morphemes and show the most important 3 n-grams in these words in Table 3. We show that the model mostly "correctly" segments the words based on morphemes. However, we observe that our model tends to up-weight longer n-grams without clear semantic meaning like "teness" in "kindness". We doubt that in absence of the whole word vector, the model could "cheat" by learning long uncommon n-grams as the whole word vector. Training the model on larger dataset might mitigate the problem when all n-grams occur in multiple words with different meanings.

Nearest neighbors We hope to explore if our word representations carry enough semantic information to find synonyms, even though they are assembled from n-gram vectors. We show the nearest neighbors of some relatively uncommon words based the cosine similarity of word vectors in Table 4. The trained model was able to character semantic similarities between words even they don't share n-grams in common.

6 Conclusion

In this project, we build a model to learn word representations using only subword information. Our word vectors were computed using the attention of the n-gram representations. Our model

Table 3: The Most Significance N-grams in Words

	1st	2nd	3rd
anarchy	narchy	⟨anarc	⟨anar
monarchy	onarch	monarc	⟨monar
kindness	⟨kindn	⟨kind	ness⟩
politeness	polite	ness⟩	teness
lifetime	etime	⟨life	time⟩

Table 4: Nearest Neighbors Based on Cosine Similarity

Query Word	Nearest Neighbor
eateries	restaurant
tiling	tessellation
honda	toyota*
dna	ribonucleotide*
badminton	racquetball

* toyota is actually the second nearest neighbor, and the first was "hondas"
* riboneucleotide is RNA, so it's slightly different from DNA

outperforms Bojanowski *et al.* [2] in unsupervised tasks and gained higher correlation with human similarity judgment. The performance of our model on sentiment analysis on par with the fastText classifier in [3], showing that n-gram subwords are sufficient to represent the semantic meaning of words. We also showed that qualitatively, our model segments words mostly based on morphemes, and it could find synonyms for relatively less common words even though they don't share any n-grams in common.

For next steps, we are going to train the model on Wikipedia and repeat the experiments in Section 5.2 and Section 5.4. We hope that with a larger dataset, our model could put more significance on the morphemes and further down weight other long n-grams without specific meaning. We will also further test our model on multi-labeled text classification tasks other than sentiment analysis.

References

[1] Mikolov, Tomas, *et al. Distributed representations of words and phrases and their compositionality.* Advances in neural information processing systems. 2013.

[2] Bojanowski, Piotr, *et al. Enriching word vectors with subword information*. CoRR abs/1607.04606. (2016).

[3] Joulin, Armand, *et al. Bag of tricks for efficient text classification.* arXiv preprint arXiv:1607.01759 (2016).

[4] Shaoul, Cyrus. *The westbury lab wikipedia corpus.* Edmonton, AB: University of Alberta (2010).

[5] Finkelstein, Lev, *et al. Placing search in context: The concept revisited.* Proceedings of the 10th international conference on World Wide Web. ACM, 2001.

[6] Luong, Thang, Richard Socher, and Christopher Manning. *Better word representations with recursive neural networks for morphology.* Proceedings of the Seventeenth Conference on Computational Natural Language Learning. 2013.

[7] Qiu, Siyu, *et al. Co-learning of word representations and morpheme representations.* Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers. 2014.

[8] Soricut, Radu, and Franz Och. *Unsupervised morphology induction using word embeddings.* Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. 2015.

[9] Zhang, Xiang, and Yann LeCun. *Text understanding from scratch.* arXiv preprint arXiv:1502.01710 (2015).

[10] Xiao, Yijun, and Kyunghyun Cho. *Efficient character-level document classification by combining convolution and recurrent layers.* arXiv preprint arXiv:1602.00367 (2016).

[11] Conneau, Alexis, et al. *Very deep convolutional networks for natural language processing.* arXiv preprint arXiv:1606.01781 (2016).

[12] Deerwester, Scott, *et al. Indexing by latent semantic analysis.* Journal of the American society for information science 41.6 (1990): 391.

[13] Schtze, Hinrich. *Dimensions of meaning.* Proceedings of the 1992 ACM/IEEE conference on Supercomputing. IEEE Computer Society Press, 1992.

[14] Mikolov, Tomas, et al. *Efficient estimation of word representations in vector space.* arXiv preprint arXiv:1301.3781 (2013).

[15] Pennington, Jeffrey, Richard Socher, and Christopher Manning. *Glove: Global vectors for word representation.* Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). 2014.

[16] Botha, Jan, and Phil Blunsom. *Compositional morphology for word representations and language modelling.* International Conference on Machine Learning. 2014.