Outer Dictionary LSTM: An Efficient Network For Named Entity Recognition

Junxiang Ge, Qi Zhang, Haizhou Zhao
AI Research Department, Sogou Inc.
Beijing, China

Abstract

With the risen use of AI application, we need to understand some specific type of message from user's query. Named Entity Recognition(NER) is a basic task for that use. Gazzete is very useful when doing NER, howerver, most of the current method use iit in word embedding format, which will bring out the case, we need to retrain model when the word in gazzete is greatly changed. In this paper, we propose a conception, called Outer Dictionary LSTM (OD-LSTM), which means to use the outer type info of gazzete for each word, without using the specific word. Experiments shows it useful when doing NER task, and acheive a great improvement on task with which gazzete is heavily depended on.

Introduction

Named Entity Recognition (NER) is a base task when doing many Natural Language Processing (NLP) tasks. Usually, we need to extract mainly information from users' sentences to help us understand their meanings, and then use the extracted information to do further process. For example, in weather domain, we may need to extract **date and position** to help us understand when and where the user is asked towards the weather; in music type service, we may need to track **singer, song, style** entities to suit user's command; for many commom use, we need **person, location, organization** message to understand a sentence.

To accomplish this task, many method has been explored. Convolution Neural Network (CNN) is used in Image Classification (Krizhevsky, Sutskever and Geoffrey 2012), and then proved also powerful in Text Classification (Kim 2014). CNN is also useful in NER tasks, whether to capture character-level features (Chiu and Nichols 2016; Ma and Hovy 2016; Peters et al. 2017), or to do word-level sequence labelling (Santos, Xiang, and Zhou 2015; Strubell et al. 2017). To learn the correlation between words in sentence, Recurrent Neural Network (RNN) (Sutskever, Vinyals and Le 2014) and Long Short-Term Memory (LSTM) (Sak, Senior and Beaufays 2014) are proposed, and they achieve great success on machine translation area and so on. They are then used in NER tasks and achieve significant improvement (Chiu and Nichols 2016; Ma and Hovy 2016; Peters et al. 2017). These structures can be use in different situations (Yang, Liang, and Zhang 2018). As the depth of Nerual Network (NN) usually improve the final performance, more and more complicated method has been proposed in recent years. Transformer structure (Vaswani et al. 2017) and its derivative methods, like OpenAI GPT (Radford et al. 2018), BERT (Devlin et al. 2018), XLNet (Yang et al. 2019), make innovative attempt to NN structures.

Among these years, many method focus on base layer of NN structure, usually the embedding input layer, and then directly use it to do NLP tasks (Peters et al. 2018; Radford et al. 2018; Devlin et al. 2018; Yang et al. 2019). Though will it make improvement, it usually pays a long time to train the model. But in these way, it tell us that embedding input can directly infect the result of NLP tasks.

There exists a little difference between English NER tasks and Chinese NER tasks. In Chinsese or other hieroglyphics language, character means a single word; and word is a character phrase which has length longer than one. In English or other letter base language, character means a single letter, and word is a composition of letters. English word is composed by characters, for which character level CNN can help improve NER tasks (Huang et al. 2015; Chiu and Nichols 2016; Ma and Hovy 2016; Peters et al. 2017; Yang, Liang, and Zhang 2018); but that can not be use in Chinese NER tasks, which is hieroglyphics. For this reason, Chinese NER tasks is usually done by word baseline or character baseline structure. Word baseline structure refer to using word segmentor to split sentence into word tokens and then use their word vector (Peng and Dredez 2015; Peng and Dredez 2016); while character structure simply treat each sentence into combination of single Chinese character, and then using character embedding to do NLP tasks (Zhang and Yang 2018; Gui et al. 2019). In these methods, it seems useful when using word embedding within character baseline structure, or the oppisite.

Most of state-of-the-art NER methods uses Conditional Random Fields (CRF) (Lafferty, McCallum, and Pereira 2011) to decode the sequence labelling result. CRF can work even without any NN cells. Hand-writing trait can some-

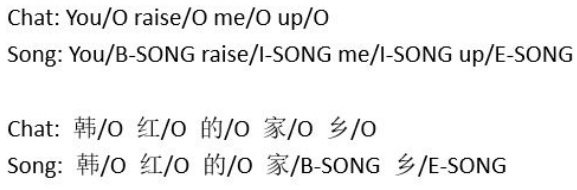

Figure 1: In certain NER situation, dictionary info is neccessary, as cases showed in figure. With knowing the certain name is a song, we tend to denote the word as a song rather a common sentence.

times already achieve high performance (Finkel, Grenager, and Manning 2005; Okazaki 2007).

In some situations, word backgound knowledge, namely dictionary (the same meaning as lexicon and gazette), is neccessary to do NER jobs. For example, in music service case, You Raise Me Up" would be treated more likely as a chat information rather than a song, unless the model is given that knowledge (see in Figure 1). Most of the state-of-the-art methods using word embedding to complete these task. But it then gives two problems:

- Out Of Vacabulary (OOV) problem. As for **song** entity in music service, word embedding size will be too large; and sometimes will the length of song be too long to make word vector.
- Retrain problem. When the content of dictionary changes abundantly, the trained model would have poor performance doing the same jobs. Still use the **song** entity as an example, the dictionary would be changed monthly or even daily, thus we need to retrain our model synchronously.

Faced with this problem, we proposed a new conception, called Outer Dictionary (OD) models, which means using the dictionary's outer trait, without using its specific content. For example, we use the word type " song ", rather than a specific song name, to help model understand the outer information. In these way, word embedding is no longer needed for Chinese NER task. Exprement result show that in gazzete depended-on task, OD model gives the best result among state-of-the-art method. And when the content of dictionary changed, the model can synchronously update its result without retraining.

Contributions

In summary, we make these contributions:

- We proposed a new conception: Outer Dictionary (OD) model, which is used to update the content of dictionary without retraining the model.
- In order to realize OD model, we explored Tag Embedding LSTM (TE-LSTM), using the word type, rather the word itself, to improve the performance of NER tasks.
- Experiment result shows our model gives significant improvements on gazzete depended-on tasks among state-of-the-art models.
- We release the music service dataset, which we use in our paper, for further scientific research.

Related Work

Success of BERT (Devlin et al. 2018) and XLNet (Yang et al. 2019) show that word embedding is important and useful for many upper NLP tasks. More useful information is there in the vector, more efficiently the model would work. To a great degree, embedding layer decides the performance. Same conclusion can be made from a lot of NER models, as for character level message, word level message, position message, handwriting message and so on. Especially in some specific NER task, like music service, gazzete is heavilly depended on to make better performance.

Most of the recent models use gazette as word embedding input, or its string format, to give the model related message. Lattice LSTM (Zhang and Yang 2018) use the embedding as additional input to character baseline LSTM, which can give the model more sentence information. Lexivon Rethinking CNN (Gui et al. 2019), corparate the word embedding vector to correlated CNN layer, and the coflict word segment could be learned by rethinking. Stanford NER (Finkel, Grenager, and Manning 2005) can use the gazette information, as string format, in CRF model.

However, in some pratical use, the content of the gazette will change greatly and constantly. In this way, most of the recent model need to be retrained after every abundantly change of gazette. To solve this problem, we need to construct a model which can still be used after such change. We then came out with the OD models, and use TE-LSTM to realize these model. As a result, with no more word embedding is needed, and model needed only to be trained once. What we need is the word tag information to help the model better understand user's intention.

Outer Dictionary LSTM

In this chapter, we illustrate the model we use to realize the OD model. Our framework is based on Tensorflow (Abadi et al. 2015). We use the character baseline model for Chinese NER task, inspired by the results of Lattice LSTM (Zhang and Yang 2018) and Lexicon Rething CNN (Gui et al. 2019). In order to better understand the model, we make a appointment that: In Chinsese or other hieroglyphics language, character means a single word; and word is a character phrase which has length longer than one. In English or other letter base language, character means a single letter, and a word is a composition of letters; then a phrase is composited of a set of words. For Chinese NER jobs, only character embedding will we use, and no more word embedding is needed, for we hold the following opinions towards these jobs:

- Common characters in hieroglyphics language, can be exhaustive, especially for Chinese, because there are no more than 100, 000 characters in simplified Chinese.
- Words in hieroglyphics language, will be uncountable, because there are still many new words or pharases are still created every day. Thus, OOV error can not be avoided in these case.

Thus, in Chinese NER jobs, we denote a sentence as a composition of many characters: $s = (c_1, c_2, ..., c_n)$ as the base input to model.

Character-Base LSTM

We use character embedding lookup table to turn character id to vector:

$$a_i = e_c(c_i) \tag{1}$$

Usually a dropout layer is connected with the embedding output, which is proved useful in NER jobs (Ma and Hovy 2016). After that, bi-directional LSTM (Bi-LSTM), the most common structure used in NER tasks, is applied to learn the character connection within the sentence. We use the hidden vector for LSTM output, that is, for input: $x = (a_1, a_2, ..., a_n)$, we get output: $h = (h_1, h_2, ..., h_n)$, where h_i is the concatenation of forward ($\overrightarrow{h_i}$) and backward ($\overleftarrow{h_i}$) LSTM hidden vector:

$$h_i = [\overrightarrow{h_i}; \overleftarrow{h_i}] \tag{2}$$

Tag Embedding Model

As case show in Figure 1, we need to use dictionary for certain NER tasks. However, the constant variation of dictionary may cause constant retrain problem. Therefore, we may not use the specific word in each dictionary, instead, only the outer information will be used, which is called Outer Dictionary (OD) model. Here, we implement our OD model using Tag Embedding (TE), which uses the tag embedding of the dictionary. Cause we use LSTM structure to utilize tag embedding info, we called our model TE-LSTM.

Specifically, we do this by two step:

a) Match sentence with the word in dictionary using certain outer model method.

b) Turn match result into embedding information, used as the input of LSTM.

Match Policy. In this part, we need to match all the word in given dictionary occured in current sentence. We impletement this by using Aho-Corasick Automation Algorithm (Aho and Corasick 1975). In summary, we complete a KMP (Knuth, Morris, and Pratt 1977) + Trie Tree (Briandais 1959) algorithm to accomplish this task. In methematical, given sentence $s = (c_1, c_2, ..., c_n)$, and a dictionary d_i, which is composed of word set $D_i = \{w_1, w_2, ..., w_V\}$. We denote the result of match policy as:

$$r_{d_i} = [(f_1, b_1), (f_2, b_2), ..., (f_k, b_k)] \tag{3}$$

Here, f_i and b_i denote the front and back position of each match word. Let W_{f_i,b_i} represent the substring of sentence s with the rank $[f_i, b_i]$, then each (f_i, b_i) tuple forms a word in D_i. Parameter k means the total number of such word matched in d_i, which is uncertain.

Tag Embedding Scheme. After match policy is fullfilled, we need to turn that result into vector so that it can be understood by model. Cause the result of match policy is denoted as position, we can simply use it to get a new sequence form from original sentence. With dictionary d_i and a matched word position rank range $[f_i, b_i]$, the original sentence can be written as: $s = (c_1, c_2, ..., c_{f_i-1}, c_{f_i}, ..., c_{b_i}, c_{b_i+1}, ..., c_n)$, we then apply tag scheme for these sequence: turn the match word to correspond tag as shown in Figure 2, the unmatche part is turn to tag ' O '.

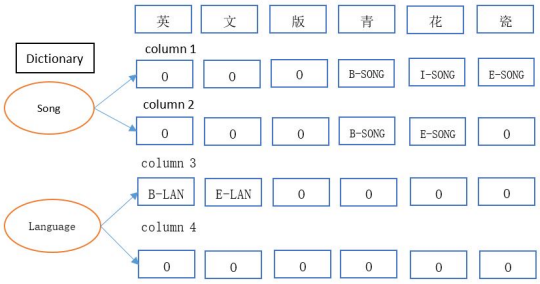

Figure 2: Illustration for tag embedding scheme. For dictionary 'song', there are two possible songs, the longer one is put in the first place (column 1). For dictionary 'language', only one possible result will be matched, however in this case, since we set the tag blank number to 2, we should add one more padding blank to fill the position, which is showed as column 4.

Here, we choose **IOBES** scheme for it is proved better than IOB1 and IOB2 tag scheme in result by previous works (Chiu and Nichols 2016; Ma and Hovy 2016). Yet we need to emphasis that: the dictionary here need not to be the same with the result tag species. As an illustration, to extract **name** entity, we can use dictionary **first name** and dictionary **last name** to do this transformation.

Since there will be one or more matched word for one dictionary, in order to fix the matched word of d_i in certain position, we choose w words from result list. Let us refer w as **tag blank number**. We choose the best w results from the matched list (in our experiment, here we use the word length, the longer, the prior). Then for dictionary d_i, if the number of match results isn't less than w, then the best w results will be choose and turn to a new sequence; or the full results will be included, and some padding blank will added to that dictionary's leave position to fill the blank, until the number of sequence of that dictionary is fullfilled. The full process of producing tag sequence is shown in Figure 2. We do the tag scheme in this way cause we think the position is important for model to understand, Of course, w could be variable for different dictionary, but that needs a bit more efforts and seems low performance increase.

With the tag embedding sequence produced, denoting each tag sequence as: $s_t = (t_1, t_2, ..., t_n)$, we turn the result to vector using:

$$g_i = e_t(t_i) \tag{4}$$

where e_t is the tag embedding lookup table. We use random generated number to produce this lookup table. Then the char embedding output a_i and tag embeddding output g_i is concatenated together, used as additional property to enrich the input information (Collobert et al. 2011), and then fed into the Bi-LSTM layer.

Table 1: Statistics of Music dataset.

Count	Sentences	Entities
Train	7,177	8,585
Dev	897	1,057
Test	897	1,047

Loss and Training

CRF layer is applied on the top, same with many previous works and proved to be useful (Collobert et al. 2011; Chiu and Nichols 2016; Ma and Hovy 2016; Yang, Liang, and Zhang 2018; Zhang and Yang 2018; Gui et al. 2019). Denote y_j as a possible label sequence for sentence s and Y as the total set of all possible y_j, then the probability of y_j is:

$$p(y_j|s) = softmax(\sum_i W_{c_i} * h_i + B_{c_i})_j \quad (5)$$

W_{c_i}, B_{c_i} is respectively the emission matrix and transformation matrix correlate to h_i, and j is the rank for y_j in all possible label sequence set Y. While training, we take the L_2 regularization for all parameters into account together with CRF loss. With batch number N and corresponding gold set (s_i, y_i), the final loss can be computed as:

$$L = \sum_{i=1}^{N} log(p(y_i|s_i)) + \frac{\lambda}{2}||\theta||^2 \quad (6)$$

λ is the L_2 hyper parameter, and θ stands for all the training parameters. At decode step, we use the Viterbi algorithm (Viterbi 1967) to decode result, finding the label sequence with the highest score.

Experiments

Settings

Here displays the experiment settings within our work.

Dataset. We use the music domain dataset obtained on China Conference on Knowledge Graph and Semantic Computing (CCKS) 2018 music dataset. Cause we can only get the training dataset, we split that file into three part with portion 8:1:1 as train set, dev set and test set. Call this dataset as **CCKS** dataset.

Another music domain dataset is annotated by ourselves, and the tag defined is more rich than CCKS. The statistics information of this dataset is listed in Table 1.

Character Embedding. In order to compare with Lattice LSTM (Zhang and Yang 2018) and Lexicon Rethinking CNN (Gui et al. 2019), we use the word vector provided in their papers (these two paper use the same character embedding and word embedding). There are character embedding and word embedding, we use the latter one as character embedding for we find it perform better in both character-level LSTM and OD LSTM.

Tag Embedding. Given tag embedding length l_t as hyper parameter, we use uniform random policy to generate the tag embedding lookup table.

Word Embedding. For Lattice LSTM and Lexicon Rethinking CNN, they use the same word embedding file, which can not incorparate the word of dictionary directly. For Stanford NER (Finkel, Grenager, and Manning 2005) model and OD model, we use the dictionary file, which in true is a list of word name. For the same dataset, the dictionary they use keep the same.

Table 2: Hyperparameters

Layer	Hyper-parameters	Value
Char-layer	embeddding size	50
Tag-layer	embeddding size	10
	tag blank number	3
LSTM	state size	200
	number of layers	1
Dropout	dropout rate	[0.0, 0.5]
	batch size	[1, 10]
	initial learning rate	0.015
	decay rate	0.05
	momentum number	0.9
	gradient clipping	5.0
	L2 λ	1e-8
	fine tune	True

Compared Models

For compared purposed, we impletement these models below:

Lattice LSTM. The author of Lattice LSTM (Zhang and Yang 2018) has generously provided his code on website. We direcly use his code and settings, and run on experiment dataset. We denote it as La-LSTM.

Lexicon Rethinking CNN. Same with Lattice LSTM, we use the code provided by the author (Gui et al. 2019) and run on our dataset. We denote it as LR-CNN.

Character LSTM. We basicly impletement the character-level LSTM as the first model, no any other properties (like bichar, softword, position etc.) used. We use the word embedding in Lattice LSTM as character embedding here for it gives better performance. We abbreviate it as C-LSTM.

Stanford NER model. Stanford NER tool (Finkel, Grenager, and Manning 2005) is used in our work to do comparision, for it is convenient to add handcrafted features into model, and use dictionary (called gazette in its term) directly, which can be compared to our OD model intuitively. For short, denote it as S-NER.

Tag Embedding LSTM. This is the model which we proposed in this paper. The character embedding is the same with the word embedding provided by the author of Lattice LSTM (Zhang and Yang 2018). The dictionary used is the same with S-NER model for comparision. We denote it as TE-LSTM.

Hyper-parameters

Shown in Table 2, we choose the hyper-parameters referring to BLSTM-CNNs-CRF structure (Ma and Hovy 2016).

Table 3: Main Results

Model	CCKS			Music		
	Precision	Recall	F1	Precision	Recall	F1
La-LSTM	87.34%	75.03%	80.72%	86.24%	84.43%	85.33%
LR-CNN	88.17%	76.98%	82.2%	85.99%	85.00%	85.49%
C-LSTM	75.66%	71.12%	73.32%	84.11%	82.50%	83.30%
ST-NER	91.00%	76.15%	82.92%	88.00%	85.48%	86.72%
TE-LSTM	84.28%	84.51%	84.40%	93.09%	92.82%	92.96%

- **Fine-tuning**.We apply fine-tuning on all the embedding vector, which is proved useful by previous works (Collobert et al. 2011; Peng and Dredze 2015; Ma and Hovy 2016).
- **Dropout**. We choose the dropout rate between 0.0 and 0.5, for different dataset, the best result will occur at different rate.
- **SGD policy**. We use stachastic gradient descent (SGD) policy to train model, with momentum set as 0.9.

Main Results

Conclusion

pass

Acknowledgments

Especially grateful to Can Cui for denoting the music ner data, to Jindou Wu for advice on data processing of CCKS 2018 music dataset.

References

Krizhevsky, A.; Sutskever, I.; and Geoffrey E. H. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of Advances in Neural Information Processing Systems 25 (NIPS 2012).

Kim, Y. 2014. Convolutional Neural Networks for Sentence Classification. arXiv:1408.5882.

Sutskever, I.; Vinyals, O.; and Le, Q.V. 2014. Sequence to Sequence Learning with Neural Networks. In Proceedings of Advances in Neural Information Processing Systems 27 (NIPS 2014).

Sak, H.; Senior, A.; and Beaufays, F. 2014. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. arXiv:1402.1128.

Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; and Kuksa, P. 2011. Natural Language Processing (Almost) from Scratch. Journal of Machine Learning Research 12 (2011) 2493-2537.

Santos. C. N.; Xiang, B.; and Zhou, B. 2015. Classifying Relations by Ranking with Convolutional Neural Networks. arXiv:1504.06580.

Strubell, E.; Verga, P.; Belanger, D.; and McCallum, A. 2017. Fast and Accurate Entity Recognition with Iterated Dilated Convolutions. arXiv:1702.02098.

Peters, M. E.; Ammar, W.; Bhagavatula, C.; and Power. R. 2017. Semi-supervised sequence tagging with bidirectional language models. arXiv:1705.00108.

Peters, M. E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; and Zettlemoyer, L. 2018. Deep contextualized word representations. arXiv:1802.05365.

Ma, X.; and Hovy, E. 2016. End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. arXiv:1603.01354.

Yang, J.; Liang, S.; and Zhang, Y. 2018. Design Challenges and Misconceptions in Neural Sequence Labeling. arXiv:1806.04470.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, L.; and Polosukhin, I. 2017. Attention Is All You Need. arXiv:1706.03762.

Radford, A.; Narasimhan, K.; Salimans, T.; and Sutskever, I. 2018. Improving Language Understanding by Generative Pre-Training. Technical report, Open AI.

Devlin, J.; Chang, M.; Lee, K.; and Toutanova, K. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv:1810.04805.

Yang, Z.; Dai, Z.; Yang, Y.; Carbonell, J.; Salakhutdinov, R.; and Le, Q. V. 2019. XLNet: Generalized Autoregressive Pretraining for Language Understanding. arXiv:1906.08237.

Sang, E. F.; and Meulder, F. D. 2003. Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition. arXiv:cs/0306050.

Chiu, J. P.C.; and Nichols, E. 2016. Named Entity Recognition with Bidirectional LSTM-CNNs. Transactions of the Association for Computational Linguistics, Volume 4, 2016, p.357-370.

Huang, Z.; Xu, W.; and Yu, K. 2015. Bidirectional LSTM-CRF Models for Sequence Tagging. arXiv:1508.01991.

Peng, N.; and Dredze, M. 2015. Named Entity Recognition for Chinese Social Media with Jointly Trained Embeddings. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing (EMNLP).

Peng, N.; and Dredze, M. 2016. Improving Named Entity Recognition for Chinese Social Media with Word Segmentation Representation Learning. arXiv:1603.00786.

He, H.; and Sun, X. 2016. F-Score Driven Max Margin Neural Network for Named Entity Recognition in Chinese Social Media. arXiv:1611.04234.

Chen, Q.; Zhuo, Z.; and Wang, W. 2019. BERT for Joint Intent Classification and Slot Filling. arXiv:1902.10909.

Zhang, Y.; and Yang, J. 2018. Chinese NER Using Lattice LSTM. arXiv:1805.02023.

Gui, T.; Ma, R.; Zhang, Q.; Zhao, L.; Jiang, Y.; and Huang, Y. 2019. CNN-Based Chinese NER with Lexicon Rethinking. In Proceedings of the International Joint Conference on Artificial Intelligence 2019 (IJCAI 2019).

Lafferty, J.; McCallum, A.; and Pereira, F. 2001. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In Proceedings of the 18th International Conference on Machine Learning 2001 (ICML 2001), pages 282-289.

Socher, R.; Manning, C. D.; and Andrew Y. Ng. 2010. Learning Continuous Phrase Representations and Syntactic Parsing with Recursive Neural Networks. In Proceedings

of the NIPS-2010 Deep Learning and Unsupervised Feature Learning Workshop.

Finkel, J. R.; Grenager, T.; and Manning, C. 2005. Incorporating non-local information into information extraction systems by Gibbs sampling. In Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics (ACL 2005), pp. 363-370.

Okazaki, N. 2007. CRFsuite: a fast implementation of Conditional Random Fields (CRFs). URL http://www. chokkan. org/software/crfsuite.

Mikolov, T.; Chen, K.; Corrado, G.; and Dean, J. 2013. Efficient Estimation of Word Representations in Vector Space. In Proceedings of Workshop at ICLR.

Mikolov, T.; Sutskever, I.; Chen, K.; Corrado, G.; and Dean, J. 2013. Distributed Representations of Words and Phrases and their Compositionality. In Proceedings of NIPS.

Mikolov, T.; Yih, W.; and Zweig, G. 2013. Linguistic Regularities in Continuous Space Word Representations. In Proceedings of NAACL HLT.

Pennington, J.; Socher, R.; and Manning, C.D. 2014. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 15321543.

Mikolov, T.; Grave, E.; Bojanowski, P.; Puhrsch, C.; and Joulin, A. 2017. Advances in Pre-Training Distributed Word Representations. arXiv:1712.09405.

Aho, A. V.; and Corasick, M. J. 1975. Efficient string matching: An aid to bibliographic search. Communications of the ACM. June 1975, 18 (6): 333340.

Knuth, D.; Morris, J. H.; and Pratt, V. 1977. Fast pattern matching in strings. SIAM Journal on Computing. 6 (2): 323350.

Briandais, D. L. R. 1959. File searching using variable length keys. Proc. Western J. Computer Conf. pp. 295298.

Viterbi AJ. 1967. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. IEEE Transactions on Information Theory. 13 (2): 260269. doi:10.1109/TIT.1967.1054010.

Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vigas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. https://www.tensorflow.org.