Point the point : Uyghur morphological segmentation using PointerNetwork with GRU

Yaofei Yang[1], Shupin Li[2], Yangsen Zhang[3], and Hua-Ping Zhang[4] ✉

[1] Beijing Information Science and Technology University
`yangyaofei@gmail.com`
[2] `susanli200808@163.com`
[3] `zhangyangsen@bistu.edu.cn`
[4] Beijing Institute of Technology
`kevinzhang@bit.edu.cn`

Abstract. Uyghur is an agglutinative language that has many morphemes. It is necessary for processing Uyghur to segment words into morphemes. This work is called morphological segmentation. Previous works treat morphological segmentation as a tagging task and classify each character as one of four classes, which are $\{b, m, e, s\}$. However, these labels are not independent from each other, which makes the models easily overfitted. We propose a new method for the segmentation task. Instead of using these labels, we use only segmentation points for modeling. The model used in our method is more robust and easier to train than previous methods. Applying our model to Uyghur morphological segmentation, it achieves high accuracy and higher recall and f1 score than previous models.

Keywords: morphological segmentation · Uyghur · linguist · agglutinative language, PointerNetwork, NLP.

1 Introduction

Morpheme is the smallest grammatical unit in a language. There are two classes of morphemes: 1) free morphemes, which have meaning, and 2) bound morphemes, which have no meaning and need free morphemes to construct words. Language can be classified as two major types according to the ratio of morphemes per word: synthetic language and analytic language. Synthetic language has a high morpheme-per-word ratio. On the contrary, most words are free morphemes in analytic language. More specifically, synthetic language can be classified into inflected language, which combines morphemes by inflection, and agglutinative language, which combines morphemes by concatenation. For example, the English word *unbreakable* is inflected from $un + break + able$. Morphological segmentation is a task that segments words into morphemes, which is a basic natural-language-processing (NLP) task. The superiorities of morphological segmentation before further processing are as follows:

(i) Reduction of vocabulary size and alleviation of the sparsity problem because words can the share same morpheme;

(ii) Alleviation of the out-of-vocabulary (OOV) problem because new words can be constructed with known morphemes.

Uyghur is a typical agglutinative language with numbers of morphemes, and most Uyghur words are combined from numbers of morphemes. It is almost like a phrase in English that leads to it having very low frequency. Moreover, it is difficult to further process the Uyghur language without morphological segmentation, so it is necessary for Uyghur to undergo morphological segmentation.

Parts of previous works use linguists to build a rules-based system (Orhun et al., 2009), which is complex and low-performing. In recent years, a statistics method has been used in this field, in which segmentation is treated as a sequence of classification tasks, using $\{b, m, e, s\}$ to label the sequence (Wang et al., 2016). However, there is a problem with this labeling method, specifically the tags $\{b, m, e, s\}$ are not independent of each other, which may make the model overfitted, and, in turn, cause high accuracy, low recall, and eventually a low f1 score.

The motivation of our work is to find a new method for tagging data in which the tags are independent of each other, and to use this method to build a better model.

In our model, we do not use the aforementioned set of tags; instead, segmentation points that are independent of each other will be used. Further, to fit the shape of data using the proposed method, we use PointerNetwork (Vinyals et al., 2015) as the main modeling framework. Eventually, the results are more improved than those using all previous reported methods of Uyghur morphological segmentation.

2 Related Work

The famous unsupervised tools for morphological segmentation is Morfessor (Creutz and Lagus (2002)), which uses a minimum description length (MDL) algorithm. MDL-based unsupervised methods were an important breakthrough and have been applied to several languages (Goldsmith, 2001). However, this method cannot achieve good performance without manual human retrofitting. Poon et al. (2009) built a log-linear model and used an expectation maximization (EM) algorithm for training, which is a classic model framework for unsupervised learning. Based on this model, Bergmanis and Goldwater (2017) used linguistic information from pre-trained word-vectors as the main feature for their log-linear model and realized better performance.

For supervised machine learning, morphological segmentation has been treated as a classification task for each character of the sequence that classifies each character as one of four classes, $\{b, m, e, s\}$. With that approach, the original method is called a conditional random-field- (CRF-) based model. Cotterell et al. (2016) used a hidden Markov model (HMM) and the CRF model for segmentation. For part-of-speech tagging, Plank et al. (2016) used a bidirectional LSTM(Hochreiter and Schmidhuber, 1997) with the CRF model, which also works for morphological segmentation tasks. Similarly, Wang et al. (2016) used a bidirectional window LSTM for modeling and achieved outstanding performance.

For the Uyghur language, Osman et al. (2019) used CRF to build a model, and, in addition, they enlarged the tagging set for orthography. ABUDUKE-LIMU et al. (2017) used a bidirectional GRU(Cho et al., 2014) model for the Uyghur language and achieved the best performance, to date, in Uyghur morphological segmentation.

All of the aforementioned good-performance methods approached the morphological segmentation task as a tagging task. This method can be expressed as shown in Figure 1, where, in a $\{b, m, e, s\}$ tagging method, the sequence is tagged and each character is assigned one of the following pre-determined classes:

(i) **B** represents the beginning of a multi-character segmentation;
(ii) **M** represents the middle of a multi-character segmentation;
(iii) **E** denotes the end of a multi-character segmentation;
(iv) **S** denotes a single-character segmentation.

When we reviewed this tagging method, we found that if we remove the symbols M and E, segmentation can still proceed with the remaining symbols. Thus, the total information in all symbols is equal to the information in symbols B and S; that is to say, those tags are not independent of each other based on information theory. When training the model with the above four classes, the model will learn what the tag is and how to use it in the correct order, which is slightly too much effort for a model to just segment a sequence. In other words, there are rules in those tags that the model must learn, and if it focuses on these rules, overfitting may result.

It is therefore necessary to make the model concentrate on segmentation, and a new method is needed to facilitate that.

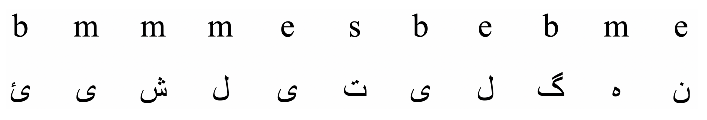

Fig. 1. Tagging sequence with $\{b, m, e, s\}$

3 Segmentation models

3.1 Labeling corpus

In the preceding section, we found that B and S have all of the information, which is the start of each segmented sequence. These tags can be expressed as a pointer to point to where to segment the sequence. Therefore, we treat this task as a generation task that generate a set of pointers pointing to the first character of each segmented sub-sequence, as shown in Figure 2.

This method, which only uses segmentation points, clearly instructs the model how to segment a word and it is thus easier for a model to understand what it should learn from data. It is difficult for the model to make a mistake and this method makes the model more robust.

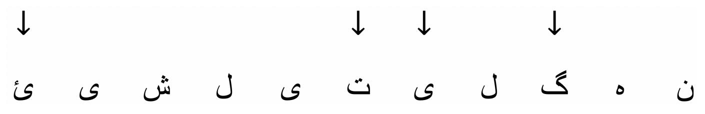

Fig. 2. Tagging sequence with pointer

The two aforementioned labeling methods are formalized in the following.

For the $\{b, m, e, s\}$ method,

$$\theta = \arg\max_{\theta} \sum_{i=1}^{n} \log p(b_i | L : \theta),$$

where $L = (l_1, l_2 ... l_n)$ is a sequence that must be segmented, $B = (b_1, b_2, ...b_n)$ is the $\{b, m, e, s\}$ label, and θ is the model's parameter.

In the proposed method,

$$\theta_p = \arg\max_{\theta_p} \sum_{i=1}^{m} \log p(p_i | L : \theta_p),$$

where $P = (p_1, p_2 ... p_m)$ is the pointer sequence for which $m < n$, and θ_p is the model's parameter.

3.2 Model selection

Because the length of P is not equal to the length of L, we cannot use a recent method like the CRF-based model. To deal with non-aligned data, the best method is to use a sequence to sequence (Seq2Seq) model (Sutskever et al., 2014). A Seq2Seq model is also known as an encoder-decoder (Cho et al., 2014) model. It was proposed to model machine translation with a deep neural network and obtain state-of-the-art performance. A Seq2Seq model has an encoder that can obtain the information of input data and push that information to the decoder, which decodes outputs.

However, a weakness is revealed upon reviewing the difference between the output of the Seq2Seq model and our proposed model. A Seq2Seq model can actually contain our outputs, but there is no strong relation between the input set and output set in the original Seq2Seq model. However, in this segmentation task, our output is a pointer pointing to the input that has a strong relation to the input.

In Vinyals et al. (2015)'s work, their PointerNetwork is a special Seq2Seq network. Each of the outputs of the PointerNetwork is a pointer that points to an input sequence that is usually used on text summarization and Q&A tasks that need the model to point out something from the input.

3.3 PointerNetwork with scaled attention

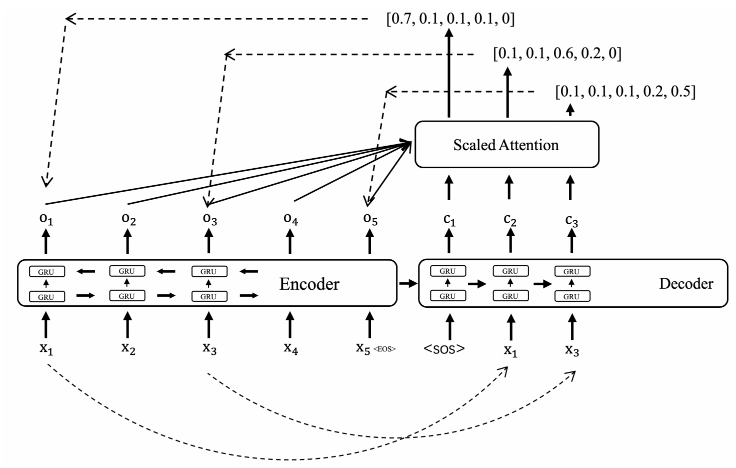

Fig. 3. Proposed modified PointerNetwork with scaled attention with example input $(x_1, x_2, x_3, x_4, x_5)$ in which x_5 is an end-of-sequence symbol <eos>

In our work, we use a modified PointerNetwork with scaled attention for the proposed method. There are three parts in our scaled attention PointerNetwork: encoder, decoder, and attention. In the encoder and decoder, we replace the LSTM (Hochreiter and Schmidhuber, 1997) with GRU (Cho et al., 2014) to reduce the size of the model. After passing through the encoder and decoder, the data flow into the attention sub-model and the probability of pointing to each input item is calculated. The entire process is detailed below.

Figure 3 shows a sample procedure for the modified PointerNetwork that we propose. Suppose the input of the PointerNetwork is $X = (x_1, x_2, x_3, x_4, x_5)$, in which x_5 is an end-of-sequence symbol <eos>. In addition, the sequence X is to be segmented into (x_1, x_2) and (x_3, x_4).

First, X goes through the encoder to obtain the output O and a hidden state h that will be sent to the decoder. The first input of the decoder is a symbol <sos>, which means the start of the sequence.

The decoder then calculates an output c_1 that will be sent to the scaled attention sub-model, and the probability vector with O will be calculated. The probability vector's dimension is equal to the length of the input X, which is 5 in this case. The largest probability in this vector will be the pointer pointing to the input, which is x_1 in this first step. The character that is pointed out by the output of the attention mechanism is sent into the decoder, which is output c_2 in the second step. Through use of the attention model, the output in this case is a pointer pointing to x_3.

Finally, in the third step, the decoder and attention model with the x_3 output pointer points to x_5, which is an end-of-sequence symbol. Then, the entire decoding stops, and the output 3 pointer segments the sequence $(x_1, x_2, x_3, x_4, x_5)$ into (x_1, x_2) and (x_3, x_4).

The entire procedure is demonstrated over, and the three parts of the proposed model are detailed below.

Encoder and decoder The encoder and decoder have a similar structure, which lets data go through them in each time step. Both also have a stacked GRU. Details are as follows.

Encoder:

In the encoder, we define $X = (x_1, x_2, x_3 ... x_n)$ as an input sequence and, through the encoder $f_encoder$, an output sequence $O = (o_1, o_2, o_3 ... o_n)$ and a hidden state h_n. The encoder can be represented as

$$O, h_n = f_{encoder}(X),$$

and for each time step,

$$o_i, h_i = f_{encoder_{i-1}}(x_{i-i}, h_{i-1}).$$

Decoder:

The input of the decoder in each time step is a character that is pointed out in the last time step. Because the encoder and decoder use the same charset for input, we share char embedding between encoder and decoder.

Each time step can be represented as follows. In time step j, we assume the output of time step $j-1$ points to x_{i-1}, and the hidden state is h_{j-1},

$$c_j = f_{decoder_j}(x_{i-1}, h_{j-1}),$$

where the output of the decoder is c_j, which will be sent to the attention model to calculate the probability of each input pointer.

Scaled attention The attention model uses one of the decoder's outputs and all of the encoder's output to calculate the probability of pointing to each input item. Suppose that the length of input is n, and then the attention model's output vector has n dimension. Each dimension represents a probability of an input item:

$$y_i = f_{atten}(O, c_i),$$

where y_i has n dimension.

In the original PointerNetwork, Bahdanau-style attention (Bahdanau et al., 2014) is used. This attention mechanism is calculated as follows:

$$f_{atten}(o_i, c_j) = V_a^{\top} tanh(W_1 o_i + W_2 c_j),$$

where V is a learned vector parameter and W_1 and W_2 are two learned matrix parameters. o_i and c_j are the outputs of the encoder and decoder, respectively.

In our work, we use dot-product attention instead, which is much faster and more space-efficient in practice, since it can be implemented using highly optimized matrix multiplication code. In addition, we follow Vaswani et al. (2017)'s scaled-attention mechanism. According to the latter paper, for large dimension d_K, dot-product attention obtains small gradients when using a softmax function in this part without scaling. The calculation is expressed as

$$\begin{aligned} f_{atten} &= softmax(\frac{QK^{\top}}{\sqrt{d}_K}), \\ Q &= W_Q C, \\ K &= W_K O, \end{aligned}$$

where C is the encoder's output and O the decoder's output; W_Q and W_K are two independent learnable matrices.

In practice, we mask out (set to ∞) all values larger than the length of input after the softmax function with a batch because we need to pad the batch sequence to the same length for batch input.

3.4 Decoding

Our model is different when inference. For the previous model, the input goes through the model and gets output for each character. But our model is a Seq2Seq model, the length of outputs are not equal to the input. And the decoder will emit each output at each timestamp. The important is in each timestamp decoder need previous output to calculate the output in current timestamp. In another words, the decoder needs a decoding procedure which is not needed in previous model.

There is a flaw in our model if we use previous decoding method. In decoding, the previous output which the decoder need is the previous pointer points to the previous segmentation point. And the output in this timestamp should be a pointer after the previous pointer, which can be a pointer before the previous one. It is an error in that situation.

To fix that flaw, we mask the output which let the probability to point before the previous one be zero, when decoding.

4 Experimentation

4.1 Dataset

For Uyghur morphological segmentation, we use the THUUyMorph (Halidanmu et al., 2018) dataset. For the comparison experiment, we process this corpus into two files: one is a source-words list and the other is a label list that uses $\{b, m, e, s\}$ symbols.

In our model experiment, the corpus is processed into three files: a source-words list, a pointer list, and a character list that are pointers pointing to the source words.

The dataset was split into two parts, one for training (70%) and one for testing (30%).

4.2 Training

For the sequence classification method, we trained the CRF model as the baseline model and the bidirectional GRU model as represented by ABUDUKELIMU et al. (2017) as the state of the art. For the CRF model, we use CRF++. In contrast, for the bidirectional GRU model, we follow the setting in ABUDUKELIMU et al. (2017), which has a 1024 GRU dimension and a 300-char embedding size.

In the original PointerNetwork, Vinyals et al. (2015) did not use the output of the hidden state from the encoder. We double it because the hidden state has rich linguistic features that should help performance. Therefore, we train our model using the two aforementioned models separately. In addition, we use the scaled-attention mechanism. To determine whether our procedure works, the attention-without-scaling mechanism is also trained. Moreover, to realize a better hyperparameter for our model, we trained various sizes of embeddings and GRUs. For the char embedding, the encoder and decoder share the same embedding weight because they use the same char vocabulary. For optimization, all neural network models are optimized by stochastic gradient descent (SGD)(Kiefer and Wolfowitz, 1952) with learning rate 0.1 and decay to 0.9 per 10000 steps. For normalization, all neural network models include bi-GRU model are used dropout with rate 0.3.

Table 1. Results of comparisons to baseline

Method	Accuracy	Precision	Recall	f1
CRF	94.20	**98.42**	95.23	96.80
Bi-GRU	**98.5**	96.05	96.15	96.15
Small pointer*	97.80	96.39	**98.70**	97.50
Large pointer**	98.03	96.52	98.56	**97.53**

* Uses size-256 GRU and size-32 embedding.
** Uses size-1024 GRU and size-64 embedding.

4.3 Analysis

The results are shown in Tables 1 and 2.

Table 2. Results using various hyperparameters

GRU size	Embedding	Hidden state[*]	Scaled[**]	Accuracy	Precision	Recall	f1
256	32	F	F	97.63	95.61	97.48	96.35
256	32	F	T	97.60	95.76	98.24	96.98
256	32	T	F	97.69	95.99	97.27	96.63
256	32	T	T	97.70	96.24	98.70	96.80
512	32	F	F	97.92	96.29	97.66	96.97
512	32	F	T	97.96	96.29	97.78	97.03
512	32	T	F	97.79	96.38	97.21	96.79
512	32	T	T	97.80	96.39	**98.70**	97.50
1024	32	F	F	97.84	96.58	97.17	96.87
1024	32	F	T	97.98	96.35	97.75	97.05
1024	32	T	F	97.72	96.04	97.32	96.67
1024	32	T	T	97.91	**96.89**	98.48	**97.68**
1024	64	F	F	97.30	95.48	97.76	96.61
1024	64	F	T	97.72	96.15	98.16	97.15
1024	64	T	F	97.41	95.86	97.69	96.76
1024	64	T	T	**98.03**	96.52	98.56	97.53
1024	128	T	T	97.93	96.43	98.42	97.41
2048	128	T	T	97.92	96.36	98.44	97.39

[*] Uses encoder's hidden state output or not that used by the decoder as initiation state.
[**] Uses scaled attention in attention mechanism or not.

In Table 1, we apply two models to facilitate a comparison with the baseline and SoTA. As we expected, we obtain a higher recall and f1 score. Because we use a more independent label to train our model, it is difficult to overfit the data and the former output would not affect the subsequent output. It makes our model less error-prone and we eventually obtain a low recall and f1 score, which makes our model more robust.

In Table 2, we compare different hyperparameters of our models. We train several models by using hidden state and scaled attention in various hyperparameters. In all hyperparameters, we can get the same conclusion. First, as mentioned above, using the encoder's hidden state is definitely helpful because there is useful information in it. The scaled-attention mechanism used in our model is more important than the hidden state, since the performance is significantly reduced without scaled attention. We believe that scaled attention can make the attention model more clear when the lengths of input are varied and the pointers need to point to them. In other words, it is a kind of dimension-wise normalization. Regarding the size of GRU and embedding, bigger is not better. A GRU size of 1024 and an embedding size of 64 obtains the best result; larger values of either of these two hyperparameters will degrade the performance.

5 Conclusions

In previous work, morphological segmentation was viewed as a classification problem for each item in a sequence using the set $\{b, m, e, s\}$ to label the sequence. We found this method to have a label-dependent problem that is damaging to training and performance. To solve this problem, we propose using fewer independent labels to model morphological segmentation tasks. We first apply PointerNetwork, modify it with a scaled attention mechanism, and use the hidden state of the encoder, in contrast to the original PointerNetwork. We thus obtain a higher recall and f1 score compared with the previous baseline and the SoTA Uyghur morphological segmentation model.

Results show that our new treatment of the segmentation task works and can achieve a more robust model. Our new method can be ported to other segmentation or tagging tasks, e.g., Chinese segmentation or morphological segmentation of the English, Urdu, and Turkish languages. Combining previous methods and our method, a higher performance and more robust ensemble model can be realized.

6 Acknowledge

This work was supported by National Science Foundation of China (Grant No. 61772075), National Science Foundation of China (Grant No. 61772081), Scientific Research Project of Beijing Educational Committee (Grant No. KM201711232022), Beijing Municipal Education Committee (Grant No. SZ20171123228), Beijing Institute of Computer Technology and Application (Grant by Extensible Konwlege Graph Construction Technique Project).

Bibliography

ABUDUKELIMU, H., CHENG, Y., LIU, Y., SUN, M.: Uyghur morphological segmentation with bidirectional GRU neural networks. Journal of Tsinghua University(Science and Technology) 57(1), 1–6 (Jan 2017)

Bahdanau, D., Cho, K., Bengio, Y.: Neural Machine Translation by Jointly Learning to Align and Translate. arXiv:1409.0473 [cs, stat] (Sep 2014)

Bergmanis, T., Goldwater, S.: From Segmentation to Analyses: A Probabilistic Model for Unsupervised Morphology Induction. In: Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 1, Long Papers. pp. 337–346. Association for Computational Linguistics, Valencia, Spain (Apr 2017)

Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., Bengio, Y.: Learning Phrase Representations using RNN Encoder–Decoder for Statistical Machine Translation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). pp. 1724–1734 (Oct 2014)

Cotterell, R., Vieira, T., Schütze, H.: A Joint Model of Orthography and Morphological Segmentation. In: Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. pp. 664–669. Association for Computational Linguistics, San Diego, California (2016)

Creutz, M., Lagus, K.: Unsupervised Discovery of Morphemes. In: Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning (2002)

Goldsmith, J.: Unsupervised Learning of the Morphology of a Natural Language. Comput. Linguist. 27(2), 153–198 (Jun 2001)

Halidanmu, A., Abudukelimu, A., SUN, M., LIU, Y.: THUUyMorph: An Uyghur Morpheme Segmentation Corpus. Journal of Chinese Information Processing 32(2), 81 (2018)

Hochreiter, S., Schmidhuber, J.: Long Short-Term Memory. Neural Comput. 9(8), 1735–1780 (Nov 1997)

Kiefer, J., Wolfowitz, J.: Stochastic Estimation of the Maximum of a Regression Function. The Annals of Mathematical Statistics 23(3), 462–466 (1952)

Orhun, M., Tantug, A.C., Adali, E.: Rule Based Analysis of the Uyghur Nouns. Int. J. of Asian Lang. Proc. 19(1), 33–44 (2009)

Osman, T., Yang, Y., Tursun, E., Cheng, L.: Collaborative Analysis of Uyghur Morphology Based on Character Level. Beijing Daxue Xuebao (Ziran Kexue Ban)/Acta Scientiarum Naturalium Universitatis Pekinensis 55, 47–54 (Jan 2019)

Plank, B., Søgaard, A., Goldberg, Y.: Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). pp. 412–418 (Aug 2016)

Poon, H., Cherry, C., Toutanova, K.: Unsupervised Morphological Segmentation with Log-Linear Models. In: Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics. pp. 209–217. Association for Computational Linguistics, Boulder, Colorado (2009)

Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to Sequence Learning with Neural Networks. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27, pp. 3104–3112. Curran Associates, Inc. (2014)

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., Polosukhin, I.: Attention is All you Need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems 30, pp. 5998–6008. Curran Associates, Inc. (2017)

Vinyals, O., Fortunato, M., Jaitly, N.: Pointer Networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems 28, pp. 2692–2700. Curran Associates, Inc. (2015)

Wang, L., Cao, Z., Xia, Y., de Melo, G.: Morphological Segmentation with Window LSTM Neural Networks. In: Thirtieth AAAI Conference on Artificial Intelligence (Mar 2016)