A Sub-Character Architecture for Korean Language Processing

Karl Stratos
Toyota Technological Institute at Chicago
stratos@ttic.edu

Abstract

We introduce a novel sub-character architecture that exploits a unique compositional structure of the Korean language. Our method decomposes each character into a small set of primitive phonetic units called jamo letters from which character- and word-level representations are induced. The jamo letters divulge syntactic and semantic information that is difficult to access with conventional character-level units. They greatly alleviate the data sparsity problem, reducing the observation space to 1.6% of the original while increasing accuracy in our experiments. We apply our architecture to dependency parsing and achieve dramatic improvement over strong lexical baselines.

1 Introduction

Korean is generally recognized as a language isolate: that is, it has no apparent genealogical relationship with other languages (Song, 2006; Campbell and Mixco, 2007). A unique feature of the language is that each character is composed of a small, fixed set of basic phonetic units called **jamo** letters. Despite the important role jamo plays in encoding syntactic and semantic information of words, it has been neglected in existing modern Korean processing algorithms. In this paper, we bridge this gap by introducing a novel compositional neural architecture that explicitly leverages the sub-character information.

Specifically, we perform Unicode decomposition on each Korean character to recover its underlying jamo letters and construct character- and word-level representations from these letters. See Figure 1 for an illustration of the decomposition. The decomposition is *deterministic*; this is a crucial departure from previous work that uses language-specific sub-character information such as radical (a graphical component of a Chinese character). The radical structure of a Chinese character does not follow any systematic process, requiring an incomplete dictionary mapping between characters and radicals to take advantage of this information (Sun et al., 2014; Yin et al., 2016). In contrast, our Unicode decomposition does not need any supervision and can extract correct jamo letters for all possible Korean characters.

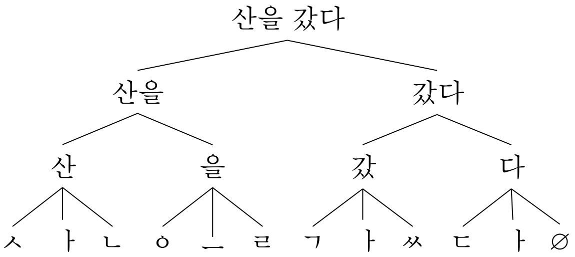

Figure 1: Korean sentence "산을 갔다" (*I went to the mountain*) decomposed to words, characters, and jamos.

Our jamo architecture is fully general and can be plugged in any Korean processing network. For a concrete demonstration of its utility, in this work we focus on dependency parsing. McDonald et al. (2013) note that "Korean emerges as a very clear outlier" in their cross-lingual parsing experiments on the universal treebank, implying a need to tailor a model for this language isolate. Because of the compositional morphology, Korean suffers extreme data sparsity at the word level: 2,703 out of 4,698 word types ($> 57\%$) in the held-out portion of our treebank are OOV. This makes the language challenging for simple lexical parsers even when

Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, pages 730–735
Copenhagen, Denmark, September 7–11, 2017. ©2017 Association for Computational Linguistics

augmented with a large set of pre-trained word representations.

While such data sparsity can also be alleviated by incorporating more conventional character-level information, we show that incorporating jamo is an effective and economical new approach to combating the sparsity problem for Korean. In experiments, we decisively improve the LAS of the lexical BiLSTM parser of Kiperwasser and Goldberg (2016) from 82.77 to 91.46 while reducing the size of input space by 98.4% when we replace words with jamos. As a point of reference, a strong feature-rich parser using *gold* POS tags obtains 88.61.

To summarize, we make the following contributions.

- To our knowledge, this is the first work that leverages jamo in end-to-end neural Korean processing. To this end, we develop a novel sub-character architecture based on deterministic Unicode decomposition.

- We perform extensive experiments on dependency parsing to verify the utility of the approach. We show clear performance boost with a drastically smaller set of parameters. Our final model outperforms strong baselines by a large margin.

- We release an implementation of our jamo architecture which can be plugged in any Korean processing network.[1]

2 Related Work

We make a few additional remarks on related work to better situate our work. Our work follows the successful line of work on incorporating sub-lexical information to neural models. Various *character*-based architectures have been proposed. For instance, Ma and Hovy (2016) and Kim et al. (2016) use CNNs over characters whereas Lample et al. (2016) and Ballesteros et al. (2015) use bidirectional LSTMs (BiLSTMs). Both approaches have been shown to be profitable; we employ a BiLSTM-based approach.

Many previous works have also considered *morphemes* to augment lexical models (Luong et al., 2013; Botha and Blunsom, 2014; Cotterell et al., 2016). Sub-character models are substantially rarer; an extreme case is considered by Gillick et al. (2016) who process text as a sequence of bytes. We believe that such byte-level models are too general and that there are opportunities to exploit natural sub-character structure for certain languages such as Korean and Chinese.

There exists a line of work on exploiting graphical components of Chinese characters called radicals (Sun et al., 2014; Yin et al., 2016). For instance, 足 (*foot*) is the radical of 跑 (*run*). While related, our work on Korean is distinguished in critical ways and should not be thought of as just an extension to another language. First, as mentioned earlier, the compositional structure is fundamentally different between Chinese and Korean. The mapping between radicals and characters in Chinese is nondeterministic and can only be loosely approximated by an incomplete dictionary. In contrast, the mapping between jamos and Korean characters is deterministic (Section 3.1), allowing for systematic decomposition of all possible Korean characters. Second, the previous work on Chinese radicals was concerned with learning word embeddings. We develop an end-to-end compositional model for a downstream task: parsing.

3 Method

3.1 Jamo Structure of the Korean Language

Let $\mathcal{W}$ denote the set of word types and $\mathcal{C}$ the set of character types. In many languages, $c \in \mathcal{C}$ is the most basic unit that is meaningful. In Korean, each character is further composed of a small fixed set of phonetic units called jamo letters $\mathcal{J}$ where $|\mathcal{J}| = 51$. The jamo letters are categorized as head consonants $\mathcal{J}_h$, vowels $\mathcal{J}_v$, or tail consonants $\mathcal{J}_t$. The composition is completely systematic. Given any character $c \in \mathcal{C}$, there exist $c_h \in \mathcal{J}_h$, $c_v \in \mathcal{J}_v$, and $c_t \in \mathcal{J}_t$ such that their composition yields c. Conversely, any $c_h \in \mathcal{J}_h$, $c_v \in \mathcal{J}_v$, and $c_t \in \mathcal{J}_t$ can be composed to yield a valid character $c \in \mathcal{C}$.

As an example, consider the word 갔다 (`went`). It is composed of two characters, 갔, 다 $\in \mathcal{C}$. Each character is furthermore composed of three jamo letters as follows:

- 갔 $\in \mathcal{C}$ is composed of ㄱ $\in \mathcal{J}_h$, ㅏ $\in \mathcal{J}_v$, and ㅆ $\in \mathcal{J}_t$.

- 다 $\in \mathcal{C}$ is composed of ㄷ $\in \mathcal{J}_h$, ㅏ $\in \mathcal{J}_v$, and an empty letter $\varnothing \in \mathcal{J}_t$.

The tail consonant can be empty; we assume a special symbol $\varnothing \in \mathcal{J}_t$ to denote an empty letter.

[1] https://github.com/karlstratos/koreannet

Figure 1 illustrates the decomposition of a Korean sentence down to jamo letters.

Note that the number of possible characters is combinatorial in the number of jamo letters, loosely upper bounded by $51^3 = 132,651$. This upper bound is loose because certain combinations are invalid. For instance, ㅁ $\in \mathcal{J}_h \cap \mathcal{J}_t$ but ㅁ $\notin \mathcal{J}_v$ whereas ㅏ $\in \mathcal{J}_v$ but ㅏ $\notin \mathcal{J}_h \cup \mathcal{J}_t$.

The combinatorial nature of Korean characters motivates the compositional architecture below. For completeness, we describe the entire forward pass of the transition-based BiLSTM parser of Kiperwasser and Goldberg (2016) that we use in our experiments.

3.2 Jamo Architecture

The parameters associated with the jamo layer are

- Embedding $e^l \in \mathbb{R}^d$ for each letter $l \in \mathcal{J}$
- $U^{\mathcal{J}}, V^{\mathcal{J}}, W^{\mathcal{J}} \in \mathbb{R}^{d \times d}$ and $b^{\mathcal{J}} \in \mathbb{R}^d$

Given a Korean character $c \in \mathcal{C}$, we perform Unicode decomposition (Section 3.3) to recover the underlying jamo letters $c_h, c_v, c_t \in \mathcal{J}$. We compose the letters to induce a representation of c as

$$h^c = \tanh\left(U^{\mathcal{J}} e^{c_h} + V^{\mathcal{J}} e^{c_v} + W^{\mathcal{J}} e^{c_t} + b^{\mathcal{J}}\right)$$

This representation is then concatenated with a character-level lookup embedding, and the result is fed into an LSTM to produce a word representation. We use an LSTM (Hochreiter and Schmidhuber, 1997) simply as a mapping $\phi : \mathbb{R}^{d_1} \times \mathbb{R}^{d_2} \rightarrow \mathbb{R}^{d_2}$ that takes an input vector x and a state vector h to output a new state vector $h' = \phi(x, h)$. The parameters associated with this layer are

- Embedding $e^c \in \mathbb{R}^{d'}$ for each $c \in \mathcal{C}$
- Forward LSTM $\phi^f : \mathbb{R}^{d+d'} \times \mathbb{R}^d \rightarrow \mathbb{R}^d$
- Backward LSTM $\phi^b : \mathbb{R}^{d+d'} \times \mathbb{R}^d \rightarrow \mathbb{R}^d$
- $U^{\mathcal{C}} \in \mathbb{R}^{d \times 2d}$ and $b^{\mathcal{C}} \in \mathbb{R}^d$

Given a word $w \in \mathcal{W}$ and its character sequence $c_1 \ldots c_m \in \mathcal{C}$, we compute

$$f_i^c = \phi^f\left(\begin{bmatrix} h^{c_i} \\ e^{c_i} \end{bmatrix}, f_{i-1}^c\right) \qquad \forall i = 1 \ldots m$$

$$b_i^c = \phi^b\left(\begin{bmatrix} h^{c_i} \\ e^{c_i} \end{bmatrix}, b_{i+1}^c\right) \qquad \forall i = m \ldots 1$$

and induce a representation of w as

$$h^w = \tanh\left(U^{\mathcal{C}} \begin{bmatrix} f_m^c \\ b_1^c \end{bmatrix} + b^{\mathcal{C}}\right)$$

Lastly, this representation is concatenated with a word-level lookup embedding (which can be initialized with pre-trained word embeddings), and the result is fed into a BiLSTM network. The parameters associated with this layer are

- Embedding $e^w \in \mathbb{R}^{d_{\mathcal{W}}}$ for each $w \in \mathcal{W}$
- Two-layer BiLSTM Φ that maps $h_1 \ldots h_n \in \mathbb{R}^{d+d_{\mathcal{W}}}$ to $z_1 \ldots z_n \in \mathbb{R}^{d^*}$
- Feedforward for predicting transitions

Given a sentence $w_1 \ldots w_n \in \mathcal{W}$, the final d^*-dimensional word representations are given by

$$(z_1 \ldots z_n) = \Phi\left(\begin{bmatrix} h^{w_1} \\ e^{w_1} \end{bmatrix} \ldots \begin{bmatrix} h^{w_n} \\ e^{w_n} \end{bmatrix}\right)$$

The parser then uses the feedforward network to greedily predict transitions based on words that are active in the system. The model is trained end-to-end by optimizing a max-margin objective. Since this part is not a contribution of this paper, we refer to Kiperwasser and Goldberg (2016) for details.

By setting the embedding dimension of jamos d, characters d', or words $d_{\mathcal{W}}$ to zero, we can configure the network to use any combination of these units. We report these experiments in Section 4.

3.3 Unicode Decomposition

Our architecture requires dynamically extracting jamo letters given any Korean character. This is achieved by simple Unicode manipulation. For any Korean character $c \in C$ with Unicode value $U(c)$, let $\overline{U}(c) = U(c) - 44032$ and $T(c) = \overline{U}(c) \bmod 28$. Then the Unicode values $U(c_h)$, $U(c_v)$, and $U(c_t)$ corresponding to the head consonant, vowel, and tail consonant are obtained by

$$\begin{aligned} U(c_h) &= 1 + \left\lfloor \frac{\overline{U}(c)}{588} \right\rfloor + \texttt{0x10ff} \\ U(c_v) &= 1 + \left\lfloor \frac{(\overline{U}(c) - T(c)) \bmod 588}{28} \right\rfloor + \texttt{0x1160} \\ U(c_t) &= 1 + T(c) + \texttt{0x11a7} \end{aligned}$$

where c_t is set to $\varnothing$ if $T(c_t) = 0$.

	Training	Development	Test
# projective trees	5,425	603	299
# non-projective trees	12	0	0

	#	# Ko	Examples
word	31,060	–	프로그램보다 갈비 booz
char	1,772	1,315	최 귤 흠 냥 셧 캐 쪽 @ 正 a
jamo	500	48	ㄱ ㄳ ㄼ ㅏ ㅠ ㅢ @ 正 a

Table 1: Treebank statistics. Upper: Number of trees in the split. Lower: Number of unit types in the training portion. For simplicity, we include non-Korean symbols (e.g., @, 正, a) as characters/jamos.

3.4 Why Use Jamo Letters?

The most obvious benefit of using jamo letters is alleviating data sparsity by flattening the combinatorial space of Korean characters. We discuss some additional explicit benefits. First, jamo letters often indicate syntactic properties of words. For example, a tail consonant ㅆ strongly implies that the word is a past tense verb as in 갔다 (went), 왔다 (came), and 했다 (did). Thus a jamo-level model can identify unseen verbs more effectively than word- or character-level models. Second, jamo letters dictate the sound of a character. For example, 갔 is pronounced as got because the head consonant ㄱ is associated with the sound g, the vowel ㅏ with o, and the tail consonant ㅆ with t. This is clearly critical for speech recognition/synthesis and indeed has been investigated in the speech community (Lee et al., 1994; Sakti et al., 2010). While speech processing is not our focus, the phonetic signals can capture useful lexical correlation (e.g., for onomatopoeic words).

4 Experiments

Data We use the publicly available Korean treebank in the universal treebank version 2.0 (McDonald et al., 2013).[2] The dataset comes with a train/development/test split; data statistics are shown in Table 1. Since the test portion is significantly smaller than the dev portion, we report performance on both.

As expected, we observe severe data sparsity with words: 24,814 out of 31,060 elements in the vocabulary appear only *once* in the training data. On the dev set, about 57% word types and 3% character types are OOV. Upon Unicode decomposition, we obtain the following 48 jamo types:

ㄱ ㄳ ㄲ ㄵ ㄴ ㄷ ㄶ ㄹ ㄸ ㄻ ㄺ ㄼ ㅁ
ㅀ ㅃ ㅂ ㅅ ㅄ ㅇ ㅆ ㅉ ㅈ ㅋ ㅊ ㅍ ㅌ
ㅏ ㅎ ㅑ ㅐ ㅓ ㅒ ㅕ ㅔ ㅗ ㅖ ㅙ ㅘ ㅛ ㅚ
ㅝ ㅜ ㅟ ㅞ ㅡ ㅠ ㅣ ㅢ

none of which is OOV in the dev set.

Implementation and baselines We implement our jamo architecture using the DyNet library (Neubig et al., 2017) and plug it into the BiLSTM parser of Kiperwasser and Goldberg (2016).[3] For Korean syllable manipulation, we use the freely available toolkit by Joshua Dong.[4] We train the parser for 30 epochs and use the dev portion for model selection. We compare our approach to the following baselines:

- McDonald13: A cross-lingual parser originally reported in McDonald et al. (2013).
- Yara: A beam-search transition-based parser of Rasooli and Tetreault (2015) based on the rich non-local features in Zhang and Nivre (2011). We use beam width 64. We use 5-fold jackknifing on the training portion to provide POS tag features. We also report on using *gold* POS tags.
- K&G16: The basic BiLSTM parser of Kiperwasser and Goldberg (2016) without the sublexical architecture introduced in this work.
- Stack LSTM: A greedy transition-based parser based on stack LSTM representations. Dyer15 denotes the word-level variant (Dyer et al., 2015). Ballesteros15 denotes the character-level variant (Ballesteros et al., 2015).

For pre-trained word embeddings, we apply the spectral algorithm of Stratos et al. (2015) on a 2015 Korean Wikipedia dump to induce 285,933 embeddings of dimension 100.

Parsing accuracy Table 2 shows the main result. The baseline test LAS of the original cross-lingual parser of McDonald13 is 55.85. Yara achieves 85.17 with predicted POS tags and 88.61 with gold POS tags. The basic BiLSTM model of K&G16 obtains 82.77 with pre-trained word embeddings (78.95 without). The stack LSTM parser is comparable to K&G16 at the word level

[2] https://github.com/ryanmcd/uni-dep-tb

[3] https://github.com/elikip/bist-parser

[4] https://github.com/JDongian/python-jamo

System	Features	Feature Representation	Emb	POS	Dev		Test	
					UAS	LAS	UAS	LAS
McDonald13	cross-lingual features	large sparse matrix	–	PRED	–	–	71.22	55.85
Yara (beam 64)	features in Z&N11	large sparse matrix	–	PRED	76.31	62.83	91.19	85.17
				GOLD	79.08	68.85	92.93	88.61
K&G16	word	31060 × 100 matrix	–	–	68.87	48.25	88.61	78.95
		298115 × 100 matrix	YES		76.30	60.88	90.00	82.77
Dyer15	word, transition	31067 × 100 matrix	–	–	69.40	48.46	88.41	78.22
		298122 × 100 matrix	YES		75.99	59.38	90.73	83.89
Ballesteros15	char, transition	1779 × 100 matrix	–	–	84.22	76.41	91.27	86.25
KoreanNet	char	1772 × 100 matrix	–	–	84.76	76.95	94.75	90.81
		1772 × 200 matrix			84.83	77.29	94.55	91.04
	jamo	500 × 100 matrix	–	–	84.27	76.07	94.59	90.77
		500 × 200 matrix			84.68	77.27	94.86	91.46
	char, jamo	2272 × 100 matrix	–	–	85.35	78.18	94.79	91.19
		2272 × 200 matrix			85.74	78.76	94.55	91.31
	word, char, jamo	302339 × 200 matrix	YES	–	**86.39**	**79.68**	**95.17**	**92.31**

Table 2: Main result. Upper: Accuracy with baseline models. Lower: Accuracy with different configurations of our parser network (word-only is identical to K&G16).

(Dyer15), but it performs significantly better at the character level (Ballesteros15) reaching 86.25 test LAS.

We observe decisive improvement when we incorporate sub-lexical information into the parser of K&G16. In fact, a *strictly* sub-lexical parser using only jamos or characters clearly outperforms its lexical counterpart despite the fact that the model is drastically smaller (e.g., 90.77 with 500 × 100 jamo embeddings vs 82.77 with 298115 × 100 word embeddings). Notably, jamos alone achieve 91.46 which is not far behind the best result 92.31 obtained by using word, character, and jamo units in conjunction. This demonstrates that our compositional architecture learns to build effective representations of Korean characters and words for parsing from a minuscule set of jamo letters.

5 Discussion of Future Work

We have presented a natural sub-character architecture to model the unique compositional orthography of the Korean language. The architecture induces word-/sentence-level representations from a small set of phonetic units called jamo letters. This is enabled by efficient and deterministic Unicode decomposition of characters.

We have focused on dependency parsing to demonstrate the utility of our approach as an economical and effective way to combat data sparsity. However, we believe that the true benefit of this architecture will be more evident in speech processing as jamo letters are definitions of sound in the language. Another potentially interesting application is informal text on the internet. Ill-formed words such as ㅎㅎㅎ (shorthand for 하하하, an onomatopoeic expression of laughter) and ㄴㄴ (shorthand for 노노, a transcription of `no no`) are omnipresent in social media. The jamo architecture can be useful in this scenario, for instance by correlating ㅎㅎㅎ and 하하하 which might otherwise be treated as independent.

Acknowledgments

The author would like to thank Lingpeng Kong for his help with using the DyNet library, Mohammad Rasooli for his help with using the Yara parser, and Karen Livescu for helpful comments.

References

Miguel Ballesteros, Chris Dyer, and Noah A. Smith. 2015. Improved transition-based parsing by modeling characters instead of words with lstms. In *Proc. EMNLP*.

Jan A Botha and Phil Blunsom. 2014. Compositional morphology for word representations and language modelling. In *ICML*, pages 1899–1907.

Lyle Campbell and Mauricio J Mixco. 2007. *A glossary of historical linguistics*. Edinburgh University Press.

Ryan Cotterell, Hinrich Schütze, and Jason Eisner. 2016. Morphological smoothing and extrapolation of word embeddings. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*, volume 1, pages 1651–1660.

Chris Dyer, Miguel Ballesteros, Wang Ling, Austin Matthews, and Noah A. Smith. 2015. Transition-based dependency parsing with stack long short-term memory. In *Proc. ACL*.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2016. Multilingual language processing from bytes. In *Proceedings of NAACL*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Yoon Kim, Yacine Jernite, David Sontag, and Alexander M Rush. 2016. Character-aware neural language models. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Eliyahu Kiperwasser and Yoav Goldberg. 2016. Simple and accurate dependency parsing using bidirectional lstm feature representations. *Transactions of the Association for Computational Linguistics*, 4:313–327.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of NAACL*.

Geunbae Lee, Jong-Hyeok Lee, and Kyunghee Kim. 1994. Phonemie-level, speech and natural, language integration for agglutinative languages. *GGGGGGGG 0*.

Thang Luong, Richard Socher, and Christopher D Manning. 2013. Better word representations with recursive neural networks for morphology. In *CoNLL*, pages 104–113.

Xuezhe Ma and Eduard Hovy. 2016. End-to-end sequence labeling via bi-directional lstm-cnns-crf. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1064–1074, Berlin, Germany. Association for Computational Linguistics.

Ryan T McDonald, Joakim Nivre, Yvonne Quirmbach-Brundage, Yoav Goldberg, Dipanjan Das, Kuzman Ganchev, Keith B Hall, Slav Petrov, Hao Zhang, Oscar Täckström, et al. 2013. Universal dependency annotation for multilingual parsing. In *ACL (2)*, pages 92–97.

Graham Neubig, Chris Dyer, Yoav Goldberg, Austin Matthews, Waleed Ammar, Antonios Anastasopoulos, Miguel Ballesteros, David Chiang, Daniel Clothiaux, Trevor Cohn, Kevin Duh, Manaal Faruqui, Cynthia Gan, Dan Garrette, Yangfeng Ji, Lingpeng Kong, Adhiguna Kuncoro, Gaurav Kumar, Chaitanya Malaviya, Paul Michel, Yusuke Oda, Matthew Richardson, Naomi Saphra, Swabha Swayamdipta, and Pengcheng Yin. 2017. Dynet: The dynamic neural network toolkit. *arXiv preprint arXiv:1701.03980*.

Mohammad Sadegh Rasooli and Joel R. Tetreault. 2015. Yara parser: A fast and accurate dependency parser. *CoRR*, abs/1503.06733.

Sakriani Sakti, Andrew Finch, Ryosuke Isotani, Hisashi Kawai, and Satoshi Nakamura. 2010. Korean pronunciation variation modeling with probabilistic bayesian networks. In *Universal Communication Symposium (IUCS), 2010 4th International*, pages 52–57. IEEE.

Jae Jung Song. 2006. *The Korean language: Structure, use and context*. Routledge.

Karl Stratos, Michael Collins, and Daniel Hsu. 2015. Model-based word embeddings from decompositions of count matrices. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 1282–1291, Beijing, China. Association for Computational Linguistics.

Yaming Sun, Lei Lin, Nan Yang, Zhenzhou Ji, and Xiaolong Wang. 2014. Radical-enhanced chinese character embedding. In *International Conference on Neural Information Processing*, pages 279–286. Springer.

Rongchao Yin, Quan Wang, Rui Li, Peng Li, and Bin Wang. 2016. Multi-granularity chinese word embedding. In *Proceedings of the Empiricial Methods in Natural Language Processing*.

Yue Zhang and Joakim Nivre. 2011. Transition-based dependency parsing with rich non-local features. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies: short papers-Volume 2*, pages 188–193. Association for Computational Linguistics.