Enhanced Universal Dependency Parsing with Automated Concatenation of Embeddings

Xinyu Wang$^{\diamond\spadesuit}$, Zixia Jia$^{\diamond\spadesuit}$, Yong Jiang†, Kewei Tu$^{\diamond *}$
$^{\diamond}$School of Information Science and Technology, ShanghaiTech University
Shanghai Engineering Research Center of Intelligent Vision and Imaging
†DAMO Academy, Alibaba Group
{wangxy1,jiazx,tukw}@shanghaitech.edu.cn
{yongjiang.jy}@alibaba-inc.com

Abstract

This paper describes the system used in submission from SHANGHAITECH team to the *IWPT 2021 Shared Task*. Our system is a graph-based parser with the technique of Automated Concatenation of Embeddings (ACE). Because recent work found that better word representations can be obtained by concatenating different types of embeddings, we use ACE to automatically find the better concatenation of embeddings for the task of enhanced universal dependencies. According to official results averaged on 17 languages, our system ranks 2nd over 9 teams.

1 Introduction

Compared to the Universal Dependencies (UD) (Nivre et al., 2016), the Enhanced Universal Dependencies (EUD) (Bouma et al., 2020, 2021)[1] makes some of the implicit relations between words more explicit and augments some of the dependency labels to facilitate the disambiguation of types of arguments and modifiers. The representation of EUD is an enhanced graph with reentrancies, cycles, and empty nodes. Such representation can represent richer grammatical relations than rooted trees, but it is harder to learn. To make the learning process relatively easy, we transfer the enhanced graph to a bi-lexical structure like annotation of semantic dependency parsing (SDP) (Oepen et al., 2015) by reducing reentrancies and empty nodes into new labels. Therefore, many approaches for SDP can be adopted by EUD. Instead of the second-order parser that was used in previous work (Wang et al., 2019, 2020b; Wang and Tu, 2020), we apply the biaffine parser (Dozat and Manning, 2018) which is one of the state-of-the-art approaches of SDP for simplicity.

Recent developments on pre-trained contextualized embeddings have significantly improved the performance of structured prediction tasks in natural language processing. A lot of work has also shown that word representations based on the concatenation of multiple pre-trained contextualized embeddings and traditional non-contextualized embeddings (such as word2vec (Mikolov et al., 2013) and character embeddings (Santos and Zadrozny, 2014)) can further improve performance (Peters et al., 2018; Akbik et al., 2018; Straková et al., 2019; Wang et al., 2020a). Wang et al. (2021) proposed Automated Concatenation of Embeddings to automate the process of finding better concatenations of embeddings and further improved performance of many tasks. We utilize their method to find concatenations of pre-trained embeddings as the input of the biaffine parser for EUD. Because there are many contextualized embeddings, such as XLMR (Conneau et al., 2020a), BERT (Devlin et al., 2018) and Flair (Akbik et al., 2018), non-contextualized embeddings, such as word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), and fastText (Bojanowski et al., 2017), and character embeddings (Santos and Zadrozny, 2014). The search space of embeddings concatenation is large in size, besides, we need to train models of 17 languages respectively. Following Wang et al. (2021), we use reinforcement learning to efficiently find the better embeddings concatenation for each language. Experimental results averaged on 17 languages show the effectiveness of our approach. Our system is ranked 2nd over 9 teams in the official evaluation.

2 System Description

2.1 Data Pre-processing

We adopt the same data pre-processing method as Wang et al. (2020b) which transfers EUD graphs

$^{\spadesuit}$: Equal contributions.

[1] https://universaldependencies.org/u/overview/enhanced-syntax.html

to SDP graphs. For the reentrancies of the same head and dependent on different labels in the EUD graph, we combined these arcs into one and concatenate the labels of these arcs with a special symbol '+' representing the combination of two arcs. For the empty nodes in the EUD graph, there is an official script that can reduce such empty nodes into non-empty nodes with new dependency labels[2].

2.2 Approach

We follow the approach of Wang et al. (2021)[3] to build our system. Our system contains two parts: an ACE module to determine embedding concatenation as inputs, a biaffine parser to predict edges' existence and labels between each word pair. We introduce these two parts respectively.

ACE Given a sentence with n words $\mathbf{w} = [w_1, w_2, ..., w_n]$, we first get the input representations $\boldsymbol{V} = [\boldsymbol{v}_1; \cdots; \boldsymbol{v}_i; \cdots; \boldsymbol{v}_n]$, $\boldsymbol{V} \in \mathbb{R}^{d \times n}$ for the sentence, where $\boldsymbol{v}_i$ is word representation of i-th word and it is a concatenation of L types of word embeddings:

$$\boldsymbol{v}_i^l = \text{embed}_i^l(\boldsymbol{x}); \quad \boldsymbol{v}_i = [\boldsymbol{v}_i^1; \boldsymbol{v}_i^2; \ldots; \boldsymbol{v}_i^L]$$

where embed^l is the model of l-th embeddings, $\boldsymbol{v}_i \in \mathbb{R}^d$, $\boldsymbol{v}_i^l \in \mathbb{R}^{d^l}$. d^l is the hidden size of embed^l. Our ACE use a binary vector $\boldsymbol{a} = [a_1, \cdots, a_l, \cdots, a_L]$ as an mask to choose a subset of embeddings of L types and mask out the rest. Thus, the embeddings become:

$$\boldsymbol{v}_i = [\boldsymbol{v}_i^1 a_1; \ldots; \boldsymbol{v}_i^l a_l; \ldots; \boldsymbol{v}_i^L a_L]$$

where a_l is a binary variable.

To learn this mask (i.e., embeddings concatenation), we set a controller which interact with our EUD parser to iteratively generate the embedding mask from the search space. Defined the probability distribution of selecting an concatenation $\boldsymbol{a}$ as $P^{\text{ctrl}}(\boldsymbol{a}; \boldsymbol{\theta}) = \prod_{l=1}^{L} P_l^{\text{ctrl}}(a_l; \theta_l)$. Each element a_l of $\boldsymbol{a}$ is sampled independently from a Bernoulli distribution, which is defined as:

$$P_l^{\text{ctrl}}(a_l; \theta_l) = \begin{cases} \sigma(\theta_l) & a_l = 1 \\ 1 - P_l^{\text{ctrl}}(a_l = 1; \theta_l) & a_l = 0 \end{cases} \quad (1)$$

where σ is the sigmoid function.

We use reinforcement learning and take the accuracy on development set of our EUD parser as reward signal R. The controller's target is to maximize the expected reward $J(\boldsymbol{\theta}) = \mathbb{E}_{P^{\text{ctrl}}(\boldsymbol{a};\boldsymbol{\theta})}[R]$ through the policy gradient method (Williams, 1992). We defined the reward function as:

$$\boldsymbol{r}^t = \sum_{i=1}^{t-1} (R_t - R_i) \gamma^{Hamm(\boldsymbol{a}^t, \boldsymbol{a}^i) - 1} |\boldsymbol{a}^t - \boldsymbol{a}^i| \quad (2)$$

Where $\gamma \in (0, 1)$. $|\boldsymbol{a}^t - \boldsymbol{a}^i|$ is a binary vector, representing the change between current embedding concatenation $\boldsymbol{a}^t$ at current time step t and $\boldsymbol{a}^i$ at previous time step i. R_t and R_i are the reward at time step t and i. $Hamm(\boldsymbol{a}^t, \boldsymbol{a}^i)$ is the Hamming distance of two concatenations.

Since calculating the exact expectation is intractable in our approach, the gradient of $J(\boldsymbol{\theta})$ is approximated by sampling only one selection following the distribution $P^{\text{ctrl}}(\boldsymbol{a}; \boldsymbol{\theta})$ at each step for training efficiency. With the reward function, the final formulation is:

$$\nabla_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}) \approx \sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}} \log P_l^{\text{ctrl}}(a_l^t; \theta_l) r_l^t \quad (3)$$

EUD Parser After getting the representation $\boldsymbol{V}$ of the sentence $\boldsymbol{w}$, we use a three-layer BiLSTM taking the representation as input:

$$\boldsymbol{R} = \text{BiLSTM}(\boldsymbol{V})$$

Where $\boldsymbol{R} = [\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n]$ represents the output from the BiLSTM. For the arc prediction and label prediction, we use two different feed-forward networks and biaffine functions:

$$\begin{aligned} s_{ij}^{(\text{arc})} &= \text{FNN\_Biaffine}^{(\text{arc})}(\boldsymbol{r}_i, \boldsymbol{r}_j) \\ \mathbf{s}_{ij}^{(\text{label})} &= \text{FNN\_Biaffine}^{(\text{label})}(\boldsymbol{r}_i, \boldsymbol{r}_j) \end{aligned}$$

The arc probability distribution and the label probability distribution for each potential arc are:

$$\begin{aligned} P^{(\text{arc})}(y_{ij}^{(\text{arc})} | \text{w}) &= \text{softmax}([s_{ij}^{(\text{arc})}; 0]) \\ P^{(\text{label})}(y_{ij}^{(\text{label})} | \text{w}) &= \text{softmax}(\mathbf{s}_{ij}^{(\text{label})}) \end{aligned}$$

According to $s_{ij}^{(\text{arc})}$, we first use MST (McDonald et al., 2005) algorithm to get a tree structure, then we additionally add arcs for the positions that $s_{ij}^{(\text{arc})} > 0$. Such method can get a EUD graph and ensure the connectivity of the graph. Wang

[2]For more details, please refer to https://universaldependencies.org/iwpt20/task_and_evaluation.html.

[3]https://github.com/Alibaba-NLP/ACE. Our code will be released here as well.

Embedding (Language)	Resource	URL
fastText (all)	Bojanowski et al. (2017)	github.com/facebookresearch/fastText
M-BERT (all)	Devlin et al. (2019)	huggingface.co/bert-base-multilingual-cased
BERT (en, et, sk, ta, uk)	Devlin et al. (2019)	huggingface.co/bert-base-cased
BERT (ar)	Safaya et al. (2020)	huggingface.co/asafaya/bert-large-arabic
BERT (bg, cs, pl, ru)	Arkhipov et al. (2019)	huggingface.co/DeepPavlov/bert-base-bg-cs-pl-ru-cased
BERT (fi)	Virtanen et al. (2019)	huggingface.co/TurkuNLP/bert-base-finnish-cased-v1
BERT (fr)	Martin et al. (2020)	huggingface.co/camembert-base
BERT (it)	dbmdz	huggingface.co/dbmdz/bert-base-italian-cased
BERT (lt)	U&R(2020)	huggingface.co/EMBEDDIA/litlat-bert
BERT (lv)	U&R(2020)	huggingface.co/EMBEDDIA/litlat-bert
BERT (nl)	wietsedv	huggingface.co/wietsedv/bert-base-dutch-cased
BERT (sv)	Malmsten et al. (2020)	huggingface.co/KB/bert-base-swedish-cased
XLM-R (all)	Conneau et al. (2020b)	huggingface.co/xlm-roberta-large
RoBERTa (uk)	youscan	huggingface.co/youscan/ukr-roberta-base
RoBERTa (ru)	Blinov and Avetisian (2020)	huggingface.co/blinoff/roberta-base-russian-v0
RoBERTa (nl)	Delobelle et al. (2020)	huggingface.co/pdelobelle/robbert-v2-dutch-base
RoBERTa (others)	Liu et al. (2019)	huggingface.co/roberta-large
XLNet (en)	Yang et al. (2019)	huggingface.co/xlnet-large-cased

Table 1: The embeddings we used in our system. The URL is where we downloaded the embeddings. 'all' means we use the model for all the languages. 'other' means we use this RoBERTa model for all the languages except the uk, ru and nl.

et al. (2020b) shows that the non-projective tree algorithm (MST) is better than the projective tree algorithm (Eisner's) for the EUD task. We select the label with the highest score of each potential arc.

Given any labeled sentence $(\boldsymbol{w}, \boldsymbol{Y}^{\star})$, where $\boldsymbol{Y}^{\star}$ stands for a gold parse graph, to train the system, we follow the approach of Wang et al. (2019) with the cross entropy loss:

$$\mathcal{L}^{(\text{arc})}(\Lambda) = -\sum_{i,j} \log(P_{\Lambda}(y_{ij}^{\star(\text{arc})}|\boldsymbol{w}))$$

$$\mathcal{L}^{(\text{label})}(\Lambda) = -\sum_{i,j} \mathbb{1}(y_{ij}^{\star(\text{arc})}) \log(P_{\Lambda}(y_{ij}^{\star(\text{label})}|\boldsymbol{w}))$$

where Λ is the parameters of our system, $\mathbb{1}(y_{ij}^{\star(\text{arc})})$ denotes the indicator function and equals 1 when edge (i, j) exists in the gold parse and 0 otherwise, and i, j ranges over all the tokens $\boldsymbol{w}$ in the sentence. The two losses are combined by a weighted average.

$$\mathcal{L} = \lambda\mathcal{L}^{(label)} + (1-\lambda)\mathcal{L}^{(arc)}$$

Where λ is a hyper-parameter.

Language	Fine-tuned XLM-R	ACE
	ELAS	ELAS
Arabic	76.07	82.90
Bulgarian	87.92	91.46
Czech	91.64	92.95
Dutch	87.11	92.33
English	86.04	89.24
Estonian	87.13	89.37
Finnish	86.00	91.66
French	74.74	93.65
Italian	89.31	93.03
Latvian	84.83	90.11
Lithuanian	68.92	85.48
Polish	87.98	90.90
Russian	91.52	93.22
Slovak	85.26	90.92
Swedish	76.02	88.04
Tamil	38.66	69.84
Ukrainian	79.60	90.87
Average	81.10	87.98

Table 2: Compared ELAS scores on development set of fine-tuning single XLM-R embedding and ACE.

3 Settings and Results

3.1 Experimental Settings

In training, we use the official development set as the development set. We tune the hyper-parameters on the development set and determine the hyper-parameter values according to the labeled F1 score (LF1) which is the evaluation metric used in SDP. LF1 measures the correctness of each arc-label pair. We use a batch size of 2000 tokens with the Adam (Kingma and Ba, 2015) optimizer. We set 30 steps of reinforcement learning, and the time of each reinforcement learning step depends on the size of data set. The hyper-parameters of our biaffine parser are shown in Table 5, which are mostly adopted from previous work on dependency parsing. For the hyper-parameters of our ACE module, we follow the settings of Wang et al. (2021). We only use the tokenized words as the model input. For the sentence

	Team Name								
Language	TGIF	**Ours**	ROBERTNLP	COMBO	UNIPI	DCU EPFL	GREW	FASTPARSE	NUIG
Arabic	81.23	**82.26**	81.58	76.39	77.17	71.01	71.13	53.74	0.00
Bulgarian	**93.63**	92.52	93.16	86.67	90.84	92.44	88.83	78.73	78.45
Czech	**92.24**	91.78	90.21	89.08	88.73	89.93	87.66	72.85	0.00
Dutch	**91.78**	88.64	88.37	87.07	84.14	81.89	84.09	68.89	0.00
English	**88.19**	87.27	87.88	84.09	87.11	85.70	85.49	73.00	65.40
Estonian	**88.38**	86.66	86.55	84.02	81.27	84.35	78.19	60.05	54.03
Finnish	**91.75**	90.81	91.01	87.28	89.62	89.02	85.20	57.71	0.00
French	**91.63**	88.40	88.51	87.32	87.43	86.68	83.33	73.18	0.00
Italian	**93.31**	92.88	93.28	90.40	91.81	92.41	90.98	78.32	0.00
Latvian	**90.23**	89.17	88.82	84.57	83.01	86.96	77.45	66.43	56.67
Lithuanian	**86.06**	80.87	80.76	79.75	71.31	78.04	74.62	48.27	59.13
Polish	**91.46**	90.66	89.78	87.65	88.31	89.17	78.20	71.52	0.00
Russian	**94.01**	93.59	92.64	90.73	90.90	92.83	90.56	78.56	66.33
Slovak	**94.96**	90.25	89.66	87.04	86.05	89.59	86.92	64.28	67.45
Swedish	**89.90**	86.62	88.03	83.20	84.91	85.20	81.54	67.26	63.12
Tamil	**65.58**	58.94	59.33	52.27	51.73	39.32	58.69	42.53	0.00
Ukrainian	**92.78**	88.94	88.86	86.92	87.51	86.09	83.90	63.42	0.00
Avg.	**89.24**	87.07	86.97	83.79	83.64	83.57	81.58	65.81	30.03

Table 3: Official results of all systems.

Language	**Stanza**				**Trankit**			
	Tokens	**Words**	**Sentences**	**ELAS**	**Tokens**	**Words**	**Sentences**	**ELAS**
Arabic	99.97	87.32	84.57	63.70	99.95	99.39	96.79	82.26
Bulgarian	99.93	99.93	97.49	92.59	99.78	99.78	98.79	92.52
Czech	99.92	99.92	95.03	91.50	99.93	99.92	97.56	91.78
Dutch	99.94	99.94	82.32	89.62	99.00	99.00	83.48	88.64
English	98.95	98.97	91.28	86.92	98.63	98.87	94.29	87.27
Estonian	99.68	99.68	90.26	86.44	99.39	99.39	94.85	86.66
Finnish	99.65	99.63	91.02	90.21	99.63	99.63	96.39	90.81
French	99.60	99.39	95.61	87.60	99.76	99.75	97.23	88.40
Italian	99.95	99.59	98.76	92.18	99.88	99.86	99.07	92.88
Latvian	99.78	99.78	98.85	89.26	99.74	99.74	98.69	89.17
Lithuanian	99.92	99.92	88.13	80.43	99.84	99.84	95.72	80.87
Polish	99.51	99.54	98.26	89.58	99.47	99.92	99.05	90.66
Russian	99.58	99.58	99.04	93.34	99.70	99.70	99.45	93.59
Slovak	99.96	99.96	86.27	89.01	99.95	99.94	95.31	90.25
Swedish	99.44	99.44	93.64	85.80	99.78	99.78	98.25	86.62
Tamil	99.92	86.95	98.76	46.70	98.33	94.19	100.00	58.94
Ukrainian	99.76	99.75	96.02	88.79	99.77	99.76	97.55	88.94
Average	99.73	98.19	93.25	84.92	99.56	99.32	96.62	87.07

Table 4: Comparison of different tokenization toolkits.

Hidden Layer	**Hidden Sizes**
BiLSTM LSTM	3*768
Arc/Label	500
Embedding/LSTM Dropouts	33%
Loss Interpolation (λ)	0.025
Adam β_1	0.9
Adam β_2	0.9
Learning rate	$2e^{-3}$
LR decay	0.5

Table 5: Hyper-parameters for our system.

and word segmentation, we used the pretrained large model of *trankit* (Nguyen et al., 2021). The embeddings we used in the ACE module for each language are shown in Table 1. For transformer-style embeddings, we only take the hidden states of the topmost layer and we only take the first piece subword representation as the multi-pieces word representation. We built our codes based on PyTorch (Paszke et al., 2019), and trained the model for each language on a single Tesla V100 GPU.

3.2 Main Results

Table 2 shows the ELAS scores (defined as F1-score over the set of enhanced dependencies in the system output and the gold standard) on development set of biaffine parser with fine-tuning single XLM-R embedding and with our ACE module. We can see that with ACE, the performance of most languages models is improved a lot.

Table 3 shows the results of official evaluations of all teams. We only show the ELAS in the results. We can see that our model gets the 1st on the Arabic language and gets the 2nd on averaged ELAS over 17 languages.

3.3 Tokenization Performances of Different Toolkits

In our experiments, we have tried two different tokenization toolkits. One is *stanza* (Qi et al., 2020) which is from Standford NLP Group, the others is *trankit* (Nguyen et al., 2021) which is a lightweight Transformer-based Python Toolkit for multilingual NLP. We use pretrained models of the two toolkits respectively. Furthermore, We train tokenization model of *stanza* for each language. Both settings of *stanza* are worse than *trankit* on sentence segmentation score. Table 4 shows the sentences and words segmentation scores of *stanza* trained on each language and pretrained *trankit*. We see that although *stanza* is better than *trankit* on segmentation score of tokens, there is a huge performance gap on segmentation score of sentences between *trankit* and *stanza*. Therefore, the final ELAS on test set tokenized by *trankit* is better than *stanza*.

4 Conclusion

Our system is a parser with automated embeddings concatenation and a biaffine encoder. Empirical results show the effectiveness of ACE to enhanced universal dependencies. Our system ranks 2nd over 9 teams according to the official ELAS.

References

Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual string embeddings for sequence labeling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1638–1649, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Mikhail Arkhipov, Maria Trofimova, Yuri Kuratov, and Alexey Sorokin. 2019. Tuning multilingual transformers for language-specific named entity recognition. In *Proceedings of the 7th Workshop on Balto-Slavic Natural Language Processing*, pages 89–93, Florence, Italy. Association for Computational Linguistics.

Pavel Blinov and Manvel Avetisian. 2020. Transformer models for drug adverse effects detection from tweets. In *Proceedings of the Fifth Social Media Mining for Health Applications Workshop & Shared Task*, pages 110–112, Barcelona, Spain (Online). Association for Computational Linguistics.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Gosse Bouma, Djamé Seddah, and Daniel Zeman. 2020. Overview of the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, Seattle, US. Association for Computational Linguistics.

Gosse Bouma, Djamé Seddah, and Daniel Zeman. 2021. From raw text to enhanced universal dependencies: The parsing shared task at iwpt 2021. In *Proceedings of the 17th International Conference on Parsing Technologies (IWPT 2021)*, pages 146–157, Bangkok, Thailand (online). Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020a. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020b. Unsupervised cross-lingual representation learning at scale. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8440–8451, Online. Association for Computational Linguistics.

Pieter Delobelle, Thomas Winters, and Bettina Berendt. 2020. RobBERT: a Dutch RoBERTa-based Language Model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3255–3265, Online. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Timothy Dozat and Christopher D Manning. 2018. Simpler but more accurate semantic dependency parsing. *arXiv preprint arXiv:1807.01396*.

Diederik P Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Martin Malmsten, Love Börjeson, and Chris Haffenden. 2020. Playing with words at the national library of sweden – making a swedish bert.

Louis Martin, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric de la Clergerie, Djamé Seddah, and Benoît Sagot. 2020. CamemBERT: a tasty French language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219, Online. Association for Computational Linguistics.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL'05)*, pages 91–98.

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

Minh Van Nguyen, Viet Lai, Amir Pouran Ben Veyseh, and Thien Huu Nguyen. 2021. Trankit: A light-weight transformer-based toolkit for multilingual natural language processing. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*.

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajič, Christopher D. Manning, Ryan McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC'16)*, pages 1659–1666, Portorož, Slovenia. European Language Resources Association (ELRA).

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Silvie Cinková, Dan Flickinger, Jan Hajic, and Zdenka Uresova. 2015. Semeval 2015 task 18: Broad-coverage semantic dependency parsing. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 915–926.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Peng Qi, Yuhao Zhang, Yuhui Zhang, Jason Bolton, and Christopher D Manning. 2020. Stanza: A python natural language processing toolkit for many human languages. *arXiv preprint arXiv:2003.07082*.

Ali Safaya, Moutasem Abdullatif, and Deniz Yuret. 2020. KUISAIL at SemEval-2020 task 12: BERT-CNN for offensive speech identification in social media. In *Proceedings of the Fourteenth Workshop on Semantic Evaluation*, pages 2054–2059, Barcelona (online). International Committee for Computational Linguistics.

Cicero D Santos and Bianca Zadrozny. 2014. Learning character-level representations for part-of-speech tagging. In *Proceedings of the 31st international conference on machine learning (ICML-14)*, pages 1818–1826.

Jana Straková, Milan Straka, and Jan Hajic. 2019. Neural architectures for nested NER through linearization. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 5326–5331, Florence, Italy. Association for Computational Linguistics.

Matej Ulčar and Marko Robnik-Šikonja. 2020. LitLat BERT. CLARIN-LT digital library in the Republic of Lithuania.

Antti Virtanen, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. Multilingual is not enough: BERT for finnish. *CoRR*, abs/1912.07076.

Xinyu Wang, Jingxian Huang, and Kewei Tu. 2019. Second-order semantic dependency parsing with end-to-end neural networks. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4609–4618, Florence, Italy. Association for Computational Linguistics.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021. Automated Concatenation of Embeddings for Structured Prediction. In *the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (**ACL-IJCNLP 2021**)*. Association for Computational Linguistics.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Huang Zhongqiang, Fei Huang, and Kewei Tu. 2020a. More embeddings, better sequence labelers? In *Findings of EMNLP*, Online.

Xinyu Wang, Yong Jiang, and Kewei Tu. 2020b. Enhanced Universal Dependency parsing with second-order inference and mixture of training data. In *Proceedings of the 16th International Conference on Parsing Technologies and the IWPT 2020 Shared Task on Parsing into Enhanced Universal Dependencies*, pages 215–220, Online. Association for Computational Linguistics.

Xinyu Wang and Kewei Tu. 2020. Second-order neural dependency parsing with message passing and end-to-end training. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing*, pages 93–99, Suzhou, China. Association for Computational Linguistics.

Ronald J Williams. 1992. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5753–5763.