Automated Concatenation of Embeddings for Structured Prediction

Xinyu Wang[◇‡], **Yong Jiang**[†*], **Nguyen Bach**[†], **Tao Wang**[†],
Zhongqiang Huang[†], **Fei Huang**[†], **Kewei Tu**[◇*]
[◇]School of Information Science and Technology, ShanghaiTech University
Shanghai Engineering Research Center of Intelligent Vision and Imaging
Shanghai Institute of Microsystem and Information Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
[†]DAMO Academy, Alibaba Group
{wangxy1,tukw}@shanghaitech.edu.cn, yongjiang.jy@alibaba-inc.com
{nguyen.bach,leeo.wangt,z.huang,f.huang}@alibaba-inc.com

Abstract

Pretrained contextualized embeddings are powerful word representations for structured prediction tasks. Recent work found that better word representations can be obtained by concatenating different types of embeddings. However, the selection of embeddings to form the best concatenated representation usually varies depending on the task and the collection of candidate embeddings, and the ever-increasing number of embedding types makes it a more difficult problem. In this paper, we propose Automated Concatenation of Embeddings (ACE) to automate the process of finding better concatenations of embeddings for structured prediction tasks, based on a formulation inspired by recent progress on neural architecture search. Specifically, a controller alternately samples a concatenation of embeddings, according to its current belief of the effectiveness of individual embedding types in consideration for a task, and updates the belief based on a reward. We follow strategies in reinforcement learning to optimize the parameters of the controller and compute the reward based on the accuracy of a task model, which is fed with the sampled concatenation as input and trained on a task dataset. Empirical results on 6 tasks and 21 datasets show that our approach outperforms strong baselines and achieves state-of-the-art performance with fine-tuned embeddings in all the evaluations.[1]

1 Introduction

Recent developments on pretrained contextualized embeddings have significantly improved the performance of structured prediction tasks in natural language processing. Approaches based on contextualized embeddings, such as ELMo (Peters et al., 2018), Flair (Akbik et al., 2018), BERT (Devlin et al., 2019), and XLM-R (Conneau et al., 2020), have been consistently raising the state-of-the-art for various structured prediction tasks. Concurrently, research has also showed that word representations based on the concatenation of multiple pretrained 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., 2020b). Given the ever-increasing number of embedding learning methods that operate on different granularities (e.g., word, subword, or character level) and with different model architectures, choosing the best embeddings to concatenate for a specific task becomes non-trivial, and exploring all possible concatenations can be prohibitively demanding in computing resources.

Neural architecture search (NAS) is an active area of research in deep learning to automatically search for better model architectures, and has achieved state-of-the-art performance on various tasks in computer vision, such as image classification (Real et al., 2019), semantic segmentation (Liu et al., 2019a), and object detection (Ghiasi et al., 2019). In natural language processing, NAS has been successfully applied to find better RNN structures (Zoph and Le, 2017; Pham et al., 2018b) and recently better transformer structures (So et al., 2019; Zhu et al., 2020). In this paper, we propose Automated Concatenation of Embeddings (ACE) to automate the process of finding better concatenations of embeddings for structured prediction tasks. ACE is formulated as an NAS problem. In this approach, an iterative search process is guided by a controller based on its belief that models the ef-

[*] Yong Jiang and Kewei Tu are the corresponding authors.
[‡]: This work was conducted when Xinyu Wang was interning at Alibaba DAMO Academy.

[1]Our code is publicly available at https://github.com/Alibaba-NLP/ACE.

fectiveness of individual embedding candidates in consideration for a specific task. At each step, the controller samples a concatenation of embeddings according to the belief model and then feeds the concatenated word representations as inputs to a task model, which in turn is trained on the task dataset and returns the model accuracy as a reward signal to update the belief model. We use the policy gradient algorithm (Williams, 1992) in reinforcement learning (Sutton and Barto, 1992) to solve the optimization problem. In order to improve the efficiency of the search process, we also design a special reward function by accumulating all the rewards based on the transformation between the current concatenation and all previously sampled concatenations.

Our approach is different from previous work on NAS in the following aspects:

1. Unlike most previous work, we focus on searching for better word representations rather than better model architectures.

2. We design a novel search space for the embedding concatenation search. Instead of using RNN as in previous work of Zoph and Le (2017), we design a more straightforward controller to generate the embedding concatenation. We design a novel reward function in the objective of optimization to better evaluate the effectiveness of each concatenated embeddings.

3. ACE achieves high accuracy without the need for retraining the task model, which is typically required in other NAS approaches.

4. Our approach is efficient and practical. Although ACE is formulated in a NAS framework, ACE can find a strong word representation on a single GPU with only a few GPU-hours for structured prediction tasks. In comparison, a lot of NAS approaches require dozens or even thousands of GPU-hours to search for good neural architectures for their corresponding tasks.

Empirical results show that ACE outperforms strong baselines. Furthermore, when ACE is applied to concatenate pretrained contextualized embeddings fine-tuned on specific tasks, we can achieve state-of-the-art accuracy on 6 structured prediction tasks including Named Entity Recognition (Sundheim, 1995), Part-Of-Speech tagging (DeRose, 1988), chunking (Tjong Kim Sang and Buchholz, 2000), aspect extraction (Hu and Liu, 2004), syntactic dependency parsing (Tesnière, 1959) and semantic dependency parsing (Oepen et al., 2014) over 21 datasets. Besides, we also analyze the advantage of ACE and reward function design over the baselines and show the advantage of ACE over ensemble models.

2 Related Work

2.1 Embeddings

Non-contextualized embeddings, such as word2vec (Mikolov et al., 2013), GloVe (Pennington et al., 2014), and fastText (Bojanowski et al., 2017), help lots of NLP tasks. Character embeddings (Santos and Zadrozny, 2014) are trained together with the task and applied in many structured prediction tasks (Ma and Hovy, 2016; Lample et al., 2016; Dozat and Manning, 2018). For pretrained contextualized embeddings, ELMo (Peters et al., 2018), a pretrained contextualized word embedding generated with multiple Bidirectional LSTM layers, significantly outperforms previous state-of-the-art approaches on several NLP tasks. Following this idea, Akbik et al. (2018) proposed Flair embeddings, which is a kind of contextualized character embeddings and achieved strong performance in sequence labeling tasks. Recently, Devlin et al. (2019) proposed BERT, which encodes contextualized sub-word information by Transformers (Vaswani et al., 2017) and significantly improves the performance on a lot of NLP tasks. Much research such as RoBERTa (Liu et al., 2019c) has focused on improving BERT model's performance through stronger masking strategies. Moreover, multilingual contextualized embeddings become popular. Pires et al. (2019) and Wu and Dredze (2019) showed that Multilingual BERT (M-BERT) could learn a good multilingual representation effectively with strong cross-lingual zero-shot transfer performance in various tasks. Conneau et al. (2020) proposed XLM-R, which is trained on a larger multilingual corpus and significantly outperforms M-BERT on various multilingual tasks.

2.2 Neural Architecture Search

Recent progress on deep learning has shown that network architecture design is crucial to the model performance. However, designing a strong neural architecture for each task requires enormous efforts, high level of knowledge, and experiences over the task domain. Therefore, automatic design of neural architecture is desired. A crucial part of

NAS is search space design, which defines the discoverable NAS space. Previous work (Baker et al., 2017; Zoph and Le, 2017; Xie and Yuille, 2017) designs a global search space (Elsken et al., 2019) which incorporates structures from hand-crafted architectures. For example, Zoph and Le (2017) designed a chained-structured search space with skip connections. The global search space usually has a considerable degree of freedom. For example, the approach of Zoph and Le (2017) takes 22,400 GPU-hours to search on CIFAR-10 dataset. Based on the observation that existing hand-crafted architectures contain repeated structures (Szegedy et al., 2016; He et al., 2016; Huang et al., 2017), Zoph et al. (2018) explored cell-based search space which can reduce the search time to 2,000 GPU-hours.

In recent NAS research, reinforcement learning and evolutionary algorithms are the most usual approaches. In reinforcement learning, the agent's actions are the generation of neural architectures and the action space is identical to the search space. Previous work usually applies an RNN layer (Zoph and Le, 2017; Zhong et al., 2018; Zoph et al., 2018) or use Markov Decision Process (Baker et al., 2017) to decide the hyper-parameter of each structure and decide the input order of each structure. Evolutionary algorithms have been applied to architecture search for many decades (Miller et al., 1989; Angeline et al., 1994; Stanley and Miikkulainen, 2002; Floreano et al., 2008; Jozefowicz et al., 2015). The algorithm repeatedly generates new populations through recombination and mutation operations and selects survivors through competing among the population. Recent work with evolutionary algorithms differ in the method on parent/survivor selection and population generation. For example, Real et al. (2017), Liu et al. (2018a), Wistuba (2018) and Real et al. (2019) applied tournament selection (Goldberg and Deb, 1991) for the parent selection while Xie and Yuille (2017) keeps all parents. Suganuma et al. (2017) and Elsken et al. (2018) chose the best model while Real et al. (2019) chose several latest models as survivors.

3 Automated Concatenation of Embeddings

In ACE, a task model and a controller interact with each other repeatedly. The task model predicts the task output, while the controller searches for better embedding concatenation as the word representation for the task model to achieve higher accuracy. Given an embedding concatenation generated from the controller, the task model is trained over the task data and returns a reward to the controller. The controller receives the reward to update its parameter and samples a new embedding concatenation for the task model. Figure 1 shows the general architecture of our approach.

3.1 Task Model

For the task model, we emphasis on sequence-structured and graph-structured outputs. Given a structured prediction task with input sentence $\boldsymbol{x}$ and structured output $\boldsymbol{y}$, we can calculate the probability distribution $P(\boldsymbol{y}|\boldsymbol{x})$ by:

$$P(\boldsymbol{y}|\boldsymbol{x}) = \frac{\exp\left(\text{Score}(\boldsymbol{x}, \boldsymbol{y})\right)}{\sum_{\boldsymbol{y}' \in \mathbb{Y}(\boldsymbol{x})} \exp\left(\text{Score}(\boldsymbol{x}, \boldsymbol{y}')\right)}$$

where $\mathbb{Y}(\boldsymbol{x})$ represents all possible output structures given the input sentence $\boldsymbol{x}$. Depending on different structured prediction tasks, the output structure $\boldsymbol{y}$ can be label sequences, trees, graphs or other structures. In this paper, we use sequence-structured and graph-structured outputs as two exemplar structured prediction tasks. We use BiLSTM-CRF model (Ma and Hovy, 2016; Lample et al., 2016) for sequence-structured outputs and use BiLSTM-Biaffine model (Dozat and Manning, 2017) for graph-structured outputs:

$$P^{\text{seq}}(\boldsymbol{y}|\boldsymbol{x}) = \text{BiLSTM-CRF}(\boldsymbol{V}, \boldsymbol{y})$$
$$P^{\text{graph}}(\boldsymbol{y}|\boldsymbol{x}) = \text{BiLSTM-Biaffine}(\boldsymbol{V}, \boldsymbol{y})$$

where $\boldsymbol{V} = [\boldsymbol{v}_1; \cdots; \boldsymbol{v}_n]$, $\boldsymbol{V} \in \mathbb{R}^{d \times n}$ is a matrix of the word representations for the input sentence $\boldsymbol{x}$ with n words, d is the hidden size of the concatenation of all embeddings. The word representation $\boldsymbol{v}_i$ of i-th word 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.

3.2 Search Space Design

The neural architecture search space can be represented as a set of neural networks (Elsken et al., 2019). A neural network can be represented as a directed acyclic graph with a set of nodes and directed edges. Each node represents an operation, while each edge represents the inputs and outputs between these nodes. In ACE, we represent each

embedding candidate as a node. The input to the nodes is the input sentence $\boldsymbol{x}$, and the outputs are the embeddings $\boldsymbol{v}^l$. Since we concatenate the embeddings as the word representation of the task model, there is no connection between nodes in our search space. Therefore, the search space can be significantly reduced. For each node, there are a lot of options to extract word features. Taking BERT embeddings as an example, Devlin et al. (2019) concatenated the last four layers as word features while Kondratyuk and Straka (2019) applied a weighted sum of all twelve layers. However, the empirical results (Devlin et al., 2019) do not show a significant difference in accuracy. We follow the typical usage for each embedding to further reduce the search space. As a result, each embedding only has a fixed operation and the resulting search space contains 2^L-1 possible combinations of nodes.

In NAS, weight sharing (Pham et al., 2018a) shares the weight of structures in training different neural architectures to reduce the training cost. In comparison, we fixed the weight of pretrained embedding candidates in ACE except for the character embeddings. Instead of sharing the parameters of the embeddings, we share the parameters of the task models at each step of search. However, the hidden size of word representation varies over the concatenations, making the weight sharing of structured prediction models difficult. Instead of deciding whether each node exists in the graph, we keep all nodes in the search space and add an additional operation for each node to indicate whether the embedding is masked out. To represent the selected concatenation, we use a binary vector $\boldsymbol{a} = [a_1, \cdots, a_l, \cdots, a_L]$ as an mask to mask out the embeddings which are not selected:

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

where a_l is a binary variable. Since the input $\boldsymbol{V}$ is applied to a linear layer in the BiLSTM layer, multiplying the mask with the embeddings is equivalent to directly concatenating the selected embeddings:

$$\boldsymbol{W}^\top \boldsymbol{v}_i = \sum_{l=1}^{L} \boldsymbol{W}_l^\top \boldsymbol{v}_i^l a_l \quad (2)$$

where $\boldsymbol{W}=[\boldsymbol{W}_1; \boldsymbol{W}_2; \ldots; \boldsymbol{W}_L]$ and $\boldsymbol{W}\in\mathbb{R}^{d\times h}$ and $\boldsymbol{W}_l\in\mathbb{R}^{d^l\times h}$. Therefore, the model weights can be shared after applying the embedding mask to all embedding candidates' concatenation. Another benefit of our search space design is that we can remove the unused embedding candidates and the corresponding weights in $\boldsymbol{W}$ for a lighter task model after the best concatenation is found by ACE.

3.3 Searching in the Space

During search, the controller generates the embedding mask for the task model iteratively. We use parameters $\boldsymbol{\theta} = [\theta_1; \theta_2; \ldots; \theta_L]$ for the controller instead of using the RNN structure applied in previous approaches (Zoph and Le, 2017; Zoph et al., 2018). The probability distribution of selecting an concatenation $\boldsymbol{a}$ is $P^{\text{ctrl}}(\boldsymbol{a};\boldsymbol{\theta}) = \prod_{l=1}^{L} P_l^{\text{ctrl}}(a_l;\theta_l)$. Each element a_l of $\boldsymbol{a}$ is sampled independently from a Bernoulli distribution, which is defined as:

$$P_l^{\text{ctrl}}(a_l;\theta_l) = \begin{cases} \sigma(\theta_l) & a_l{=}1 \\ 1{-}P_l^{\text{ctrl}}(a_l{=}1;\theta_l) & a_l{=}0 \end{cases} \quad (3)$$

where σ is the sigmoid function. Given the mask, the task model is trained until convergence and returns an accuracy R on the development set. As the accuracy cannot be back-propagated to the controller, we use the reinforcement algorithm for optimization. The accuracy R is used as the reward signal to train the controller. 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). In our approach, since calculating the exact expectation is intractable, 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:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) \approx \sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}} \log P_l^{\text{ctrl}}(a_l;\theta_l)(R-b) \quad (4)$$

where b is the baseline function to reduce the high variance of the update function. The baseline usually can be the highest accuracy during the search process. Instead of merely using the highest accuracy of development set over the search process as the baseline, we design a reward function on how each embedding candidate contributes to accuracy change by utilizing all searched concatenations' development scores. We use a binary vector $|\boldsymbol{a}^t - \boldsymbol{a}^i|$ to represent the change between current embedding concatenation $\boldsymbol{a}^t$ at current time step t and $\boldsymbol{a}^i$ at previous time step i. We then define the reward function as:

$$\boldsymbol{r}^t = \sum_{i=1}^{t-1} (R_t - R_i)|\boldsymbol{a}^t - \boldsymbol{a}^i| \quad (5)$$

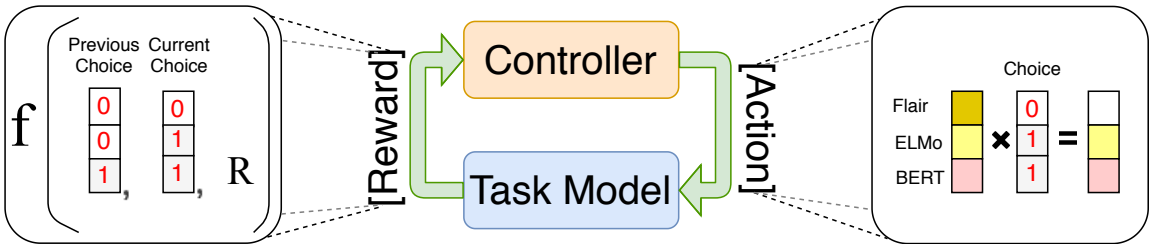

Figure 1: The main paradigm of our approach is shown in the middle, where an example of reward function is represented in the left and an example of a concatenation action is shown in the right.

where $\boldsymbol{r}^t$ is a vector with length L representing the reward of each embedding candidate. R_t and R_i are the reward at time step t and i. When the Hamming distance of two concatenations $Hamm(\boldsymbol{a}^t, \boldsymbol{a}^i)$ gets larger, the changed candidates' contribution to the accuracy becomes less noticeable. The controller may be misled to reward a candidate that is not actually helpful. We apply a discount factor to reduce the reward for two concatenations with a large Hamming distance to alleviate this issue. Our final reward function is:

$$\boldsymbol{r}^t = \sum_{i=1}^{t-1} (R_t - R_i) \gamma^{Hamm(\boldsymbol{a}^t, \boldsymbol{a}^i)-1} |\boldsymbol{a}^t - \boldsymbol{a}^i| \quad (6)$$

where $\gamma \in (0, 1)$. Eq. 4 is then reformulated as:

$$\nabla_{\boldsymbol{\theta}} J_t(\boldsymbol{\theta}) \approx \sum_{l=1}^{L} \nabla_{\boldsymbol{\theta}} \log P_l^{\text{ctrl}}(a_l^t; \theta_l) r_l^t \quad (7)$$

3.4 Training

To train the controller, we use a dictionary $\mathbb{D}$ to store the concatenations and the corresponding validation scores. At $t = 1$, we train the task model with all embedding candidates concatenated. From $t = 2$, we repeat the following steps until a maximum iteration T:

1. Sample a concatenation $\boldsymbol{a}^t$ based on the probability distribution in Eq. 3.
2. Train the task model with $\boldsymbol{a}^t$ following Eq. 1 and evaluate the model on the development set to get the accuracy R_t.
3. Given the concatenation $\boldsymbol{a}^t$, accuracy R_t and $\mathbb{D}$, compute the gradient of the controller following Eq. 7 and update the parameters of controller.
4. Add $\boldsymbol{a}^t$ and R_t into $\mathbb{D}$, set $t = t + 1$.

When sampling $\boldsymbol{a}^t$, we avoid selecting the previous concatenation $\boldsymbol{a}^{t-1}$ and the all-zero vector (i.e., selecting no embedding). If $\boldsymbol{a}^t$ is in the dictionary $\mathbb{D}$, we compare the R_t with the value in the dictionary and keep the higher one.

4 Experiments

We use ISO 639-1 language codes to represent languages in the table[2].

4.1 Datasets and Configurations

To show ACE's effectiveness, we conduct extensive experiments on a variety of structured prediction tasks varying from syntactic tasks to semantic tasks. The tasks are named entity recognition (NER), Part-Of-Speech (POS) tagging, Chunking, Aspect Extraction (AE), Syntactic Dependency Parsing (DP) and Semantic Dependency Parsing (SDP). The details of the 6 structured prediction tasks in our experiments are shown in below:

- **NER**: We use the corpora of 4 languages from the CoNLL 2002 and 2003 shared task (Tjong Kim Sang, 2002; Tjong Kim Sang and De Meulder, 2003) with standard split.
- **POS Tagging**: We use three datasets, Ritter11-T-POS (Ritter et al., 2011), ARK-Twitter (Gimpel et al., 2011; Owoputi et al., 2013) and Tweebank-v2 (Liu et al., 2018b) datasets (Ritter, ARK and TB-v2 in simplification). We follow the dataset split of Nguyen et al. (2020).
- **Chunking**: We use CoNLL 2000 (Tjong Kim Sang and Buchholz, 2000) for chunking. Since there is no standard development set for CoNLL 2000 dataset, we split 10% of the training data as the development set.
- **Aspect Extraction**: Aspect extraction is a subtask of aspect-based sentiment analysis (Pontiki et al., 2014, 2015, 2016). The datasets are from the laptop and restaurant domain of SemEval

[2] https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes

	NER				POS			AE							
	de	en	es	nl	Ritter	ARK	TB-v2	14Lap	14Res	15Res	16Res	es	nl	ru	tr
ALL	83.1	92.4	**88.9**	89.8	90.6	92.1	94.6	82.7	88.5	74.2	73.2	74.6	75.0	67.1	67.5
RANDOM	84.0	92.6	88.8	91.9	91.3	92.6	94.6	83.6	88.1	73.5	74.7	75.0	73.6	68.0	70.0
ACE	**84.2**	**93.0**	**88.9**	**92.1**	**91.7**	**92.8**	**94.8**	**83.9**	**88.6**	**74.9**	**75.6**	**75.7**	**75.3**	**70.6**	**71.1**

	CHUNK	DP		SDP						AVG
	CoNLL 2000	UAS	LAS	DM-ID	DM-OOD	PAS-ID	PAS-OOD	PSD-ID	PSD-OOD	
ALL	96.7	96.7	95.1	94.3	90.8	**94.6**	92.9	82.4	81.7	85.3
RANDOM	96.7	96.8	95.2	94.4	90.8	**94.6**	93.0	82.3	81.8	85.7
ACE	**96.8**	**96.9**	**95.3**	**94.5**	**90.9**	94.5	**93.1**	**82.5**	**82.1**	**86.2**

Table 1: Comparison with concatenating all embeddings and random search baselines on 6 tasks.

14, restaurant domain of SemEval 15 and restaurant domain of SemEval 16 shared task (14Lap, 14Res, 15Res and 16Res in short). Additionally, we use another 4 languages in the restaurant domain of SemEval 16 to test our approach in multiple languages. We randomly split 10% of the training data as the development set following Li et al. (2019).

- **Syntactic Dependency Parsing**: We use Penn Tree Bank (PTB) 3.0 with the same dataset preprocessing as (Ma et al., 2018).
- **Semantic Dependency Parsing**: We use DM, PAS and PSD datasets for semantic dependency parsing (Oepen et al., 2014) for the SemEval 2015 shared task (Oepen et al., 2015). The three datasets have the same sentences but with different formalisms. We use the standard split for SDP. In the split, there are in-domain test sets and out-of-domain test sets for each dataset.

Among these tasks, NER, POS tagging, chunking and aspect extraction are sequence-structured outputs while dependency parsing and semantic dependency parsing are the graph-structured outputs. POS Tagging, chunking and DP are syntactic structured prediction tasks while NER, AE, SDP are semantic structured prediction tasks.

We train the controller for 30 steps and save the task model with the highest accuracy on the development set as the final model for testing. Please refer to Appendix A for more details of other settings.

4.2 Embeddings

Basic Settings: For the candidates of embeddings on English datasets, we use the language-specific model for ELMo, Flair, base BERT, GloVe word embeddings, fastText word embeddings, non-contextual character embeddings (Lample et al., 2016), multilingual Flair (M-Flair), M-BERT and XLM-R embeddings. The size of the search space in our experiments is $2^{11}-1=2047$[3]. For language-specific models of other languages, please refer to Appendix A for more details. In AE, there is no available Russian-specific BERT, Flair and ELMo embeddings and there is no available Turkish-specific Flair and ELMo embeddings. We use the corresponding English embeddings instead so that the search spaces of these datasets are almost identical to those of the other datasets. All embeddings are fixed during training except that the character embeddings are trained over the task. The empirical results are reported in Section 4.3.1.

Embedding Fine-tuning: A usual approach to get better accuracy is fine-tuning transformer-based embeddings. In sequence labeling, most of the work follows the fine-tuning pipeline of BERT that connects the BERT model with a linear layer for word-level classification. However, when multiple embeddings are concatenated, fine-tuning a specific group of embeddings becomes difficult because of complicated hyper-parameter settings and massive GPU memory consumption. To alleviate this problem, we first fine-tune the transformer-based embeddings over the task and then concatenate these embeddings together with other embeddings in the basic setting to apply ACE. The empirical results are reported in Section 4.3.2.

4.3 Results

We use the following abbreviations in our experiments: **UAS**: Unlabeled Attachment Score; **LAS**: Labeled Attachment Score; **ID**: In-domain test set; **OOD**: Out-of-domain test set. We use language codes for languages in NER and AE.

[3]Flair embeddings have two models (forward and backward) for each language.

4.3.1 Comparison With Baselines

To show the effectiveness of our approach, we compare our approach with two strong baselines. For the first one, we let the task model learn by itself the contribution of each embedding candidate that is helpful to the task. We set $\boldsymbol{a}$ to all-ones (i.e., the concatenation of all the embeddings) and train the task model (`All`). The linear layer weight $\boldsymbol{W}$ in Eq. 2 reflects the contribution of each candidate. For the second one, we use the random search (`Random`), a strong baseline in NAS (Li and Talwalkar, 2020). For `Random`, we run the same maximum iteration as in ACE. For the experiments, we report the averaged accuracy of 3 runs. Table 1 shows that ACE outperforms both baselines in 6 tasks over 23 test sets with only two exceptions. Comparing `Random` with `All`, `Random` outperforms `All` by 0.4 on average and surpasses the accuracy of `All` on 14 out of 23 test sets, which shows that concatenating all embeddings may not be the best solution to most structured prediction tasks. In general, searching for the concatenation for the word representation is essential in most cases, and our search design can usually lead to better results compared to both of the baselines.

4.3.2 Comparison With State-of-the-Art approaches

As we have shown, ACE has an advantage in searching for better embedding concatenations. We further show that ACE is competitive or even stronger than state-of-the-art approaches. We additionally use XLNet (Yang et al., 2019) and RoBERTa as the candidates of ACE. In some tasks, we have several additional settings to better compare with previous work. In NER, we also conduct a comparison on the revised version of German datasets in the CoNLL 2006 shared task (Buchholz and Marsi, 2006). Recent work such as Yu et al. (2020) and Yamada et al. (2020) utilizes document contexts in the datasets. We follow their work and extract document embeddings for the transformer-based embeddings. Specifically, we follow the fine-tune process of Yamada et al. (2020) to fine-tune the transformer-based embeddings over the document except for BERT and M-BERT embeddings. For BERT and M-BERT, we follow the document extraction process of Yu et al. (2020) because we find that the model with such document embeddings is significantly stronger than the model trained with the fine-tuning process of Yamada et al. (2020). In SDP, the state-of-the-art approaches used POS tags and lemmas as additional word features to the network. We add these two features to the embedding candidates and train the embeddings together with the task. We use the fine-tuned transformer-based embeddings on each task instead of the pretrained version of these embeddings as the candidates.[4]

We additionally compare with fine-tuned XLM-R model for NER, POS tagging, chunking and AE, and compare with fine-tuned XLNet model for DP and SDP, which are strong fine-tuned models in most of the experiments. Results are shown in Table 2, 3, 4. Results show that ACE with fine-tuned embeddings achieves state-of-the-art performance in all test sets, which shows that finding a good embedding concatenation helps structured prediction tasks. We also find that ACE is stronger than the fine-tuned models, which shows the effectiveness of concatenating the fine-tuned embeddings[5].

5 Analysis

5.1 Efficiency of Search Methods

To show how efficient our approach is compared with the random search algorithm, we compare the algorithm in two aspects on CoNLL English NER dataset. The first aspect is the best development accuracy during training. The left part of Figure 2 shows that ACE is consistently stronger than the random search algorithm in this task. The second aspect is the searched concatenation at each time step. The right part of Figure 2 shows that the accuracy of ACE gradually increases and gets stable when more concatenations are sampled.

5.2 Ablation Study on Reward Function Design

To show the effectiveness of the designed reward function, we compare our reward function (Eq. 6) with the reward function without discount factor (Eq. 5) and the traditional reward function (reward term in Eq. 4). We sample 2000 training sentences on CoNLL English NER dataset for faster training and train the controller for 50 steps. Table 5 shows that both the discount factor and the binary vector $|\boldsymbol{a}^t - \boldsymbol{a}^i|$ for the task are helpful in both development and test datasets.

[4]Please refer to Appendix for more details about the embeddings.

[5]We compare ACE with other fine-tuned embeddings in Appendix.

	NER						POS		
	de	de_{06}	en	es	nl		Ritter	ARK	TB-v2
Baevski et al. (2019)	-	-	93.5	-	-	Owoputi et al. (2013)	90.4	93.2	94.6
Straková et al. (2019)	85.1	-	93.4	88.8	92.7	Gui et al. (2017)	90.9	-	92.8
Yu et al. (2020)	86.4	90.3	93.5	90.3	93.7	Gui et al. (2018)	91.2	92.4	-
Yamada et al. (2020)	-	-	94.3	-	-	Nguyen et al. (2020)	90.1	94.1	95.2
XLM-R+Fine-tune	87.7	91.4	94.1	89.3	95.3	XLM-R+Fine-tune	92.3	93.7	95.4
ACE+Fine-tune	**88.3**	**91.7**	**94.6**	**95.9**	**95.7**	ACE+Fine-tune	**93.4**	**94.4**	**95.8**

Table 2: Comparison with state-of-the-art approaches in NER and POS tagging. †: Models are trained on both train and development set.

	CHUNK		AE							
	CoNLL 2000		14Lap	14Res	15Res	16Res	es	nl	ru	tr
Akbik et al. (2018)	96.7	Xu et al. (2018)†	84.2	84.6	72.0	75.4	-	-	-	-
Clark et al. (2018)	97.0	Xu et al. (2019)	84.3	-	-	78.0	-	-	-	-
Liu et al. (2019b)	**97.3**	Wang et al. (2020a)	-	-	-	72.8	74.3	72.9	71.8	59.3
Chen et al. (2020)	95.5	Wei et al. (2020)	82.7	87.1	72.7	77.7	-	-	-	-
XLM-R+Fine-tune	97.0	XLM-R+Fine-tune	85.9	90.5	76.4	78.9	77.0	77.6	77.7	74.1
ACE+Fine-tune	**97.3**	ACE+Fine-tune	**87.4**	**92.0**	**80.3**	**81.3**	**79.9**	**80.5**	**79.4**	**81.9**

Table 3: Comparison with state-of-the-art approaches in chunking and aspect extraction. †: We report the results reproduced by Wei et al. (2020).

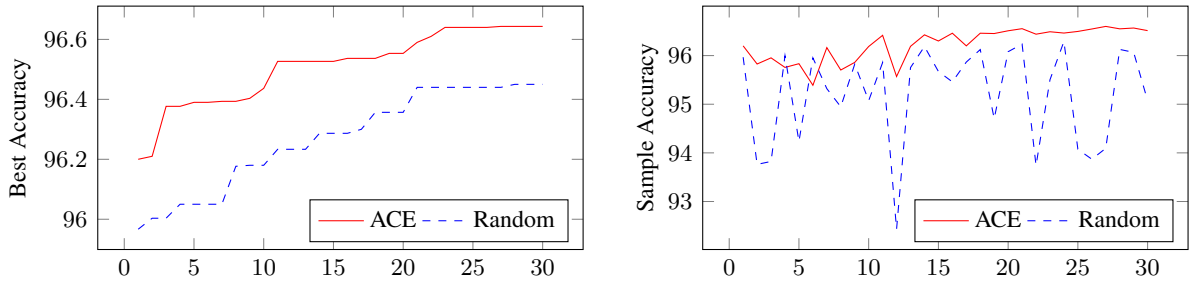

Figure 2: Comparing the efficiency of random search (`Random`) and ACE. The x-axis is the number of time steps. The left y-axis is the averaged best validation accuracy on CoNLL English NER dataset. The right y-axis is the averaged validation accuracy of the current selection.

5.3 Comparison with Embedding Weighting & Ensemble Approaches

We compare ACE with two more approaches to further show the effectiveness of ACE. One is a variant of `All`, which uses a weighting parameter $\boldsymbol{b} = [b_1, \cdots, b_l, \cdots, b_L]$ passing through a sigmoid function to weight each embedding candidate. Such an approach can explicitly learn the weight of each embedding in training instead of a binary mask. We call this approach `All+Weight`. Another one is model ensemble, which trains the task model with each embedding candidate individually and uses the trained models to make joint prediction on the test set. We use voting for ensemble as it is simple and fast. For sequence labeling tasks, the models vote for the predicted label at each position. For DP, the models vote for the tree of each sentence. For SDP, the models vote for each potential labeled arc. We use the confidence of model predictions to break ties if there are more than one agreement with the same counts. We call this approach `Ensemble`. One of the benefits of voting is that it combines the predictions of the task models efficiently without any training process. We can search all possible 2^L-1 model ensembles in a short period of time through caching the outputs of the models. Therefore, we search for the best ensemble of models on the development set and then evaluate the best ensemble on the test set ($\texttt{Ensemble}_{\texttt{dev}}$). Moreover, we additionally search for the best ensemble on the test set for reference ($\texttt{Ensemble}_{\texttt{test}}$), which is the upper bound of the approach. We use the same setting as in Section 4.3.1 and select one of the datasets from each task. For NER, POS tagging, AE, and SDP, we use CoNLL 2003 English, Ritter, 16Res, and DM datasets, respectively. The results are shown in Table 6. Empirical results show that ACE out-

	DP			SDP					
	PTB			DM		PAS		PSD	
	UAS	LAS		ID	OOD	ID	OOD	ID	OOD
Zhou and Zhao (2019)†	97.2	95.7	He and Choi (2020)‡	94.6	90.8	96.1	94.4	86.8	79.5
Mrini et al. (2020)†	97.4	96.3	D & M (2018)	93.7	88.9	93.9	90.6	81.0	79.4
Li et al. (2020)	96.6	94.8	Wang et al. (2019)	94.0	89.7	94.1	91.3	81.4	79.6
Zhang et al. (2020)	96.1	94.5	Jia et al. (2020)	93.6	89.1	-	-	-	-
Wang and Tu (2020)	96.9	95.3	F & G (2020)	94.4	91.0	95.1	93.4	82.6	82.0
XLNET+Fine-tune	97.0	95.6	XLNet+Fine-tune	94.2	90.6	94.8	93.4	82.7	81.8
ACE+Fine-tune	**97.2**	**95.8**	ACE+Fine-tune	**95.6**	**92.6**	**95.8**	**94.6**	**83.8**	**83.4**

Table 4: Comparison with state-of-the-art approaches in DP and SDP. †: For reference, they additionally used constituency dependencies in training. We also find that the PTB dataset used by Mrini et al. (2020) is not identical to the dataset in previous work such as Zhang et al. (2020) and Wang and Tu (2020). ‡: For reference, we confirmed with the authors of He and Choi (2020) that they used a different data pre-processing script with previous work.

.

	DEV	TEST
ACE	**93.18**	**90.00**
No discount (Eq. 5)	92.98	89.90
Simple (Eq. 4)	92.89	89.82

Table 5: Comparison of reward functions.

	NER	POS	AE	CHK	DP		SDP	
					UAS	LAS	ID	OOD
All	92.4	90.6	73.2	96.7	96.7	95.1	94.3	90.8
Random	92.6	91.3	74.7	96.7	96.8	95.2	94.4	90.8
ACE	**93.0**	**91.7**	**75.6**	**96.8**	**96.9**	**95.3**	**94.5**	**90.9**
All+Weight	92.7	90.4	73.7	96.7	96.7	95.1	94.3	90.7
Ensemble	92.2	90.6	68.1	96.5	96.1	94.3	94.1	90.3
$\text{Ensemble}_{\text{dev}}$	92.2	90.8	70.2	96.7	96.8	95.2	94.3	90.7
$\text{Ensemble}_{\text{test}}$	92.7	91.4	73.9	96.7	96.8	95.2	94.4	90.8

Table 6: A comparison among `All`, `Random`, ACE, `All+Weight` and `Ensemble`. CHK: chunking.

performs all the settings of these approaches and even $\texttt{Ensemble}_{\texttt{test}}$, which shows the effectiveness of ACE and the limitation of ensemble models. `All`, `All+Weight` and $\texttt{Ensemble}_{\texttt{dev}}$ are competitive in most of the cases and there is no clear winner of these approaches on all the datasets. These results show the strength of embedding concatenation. Concatenating the embeddings incorporates information from all the embeddings and forms stronger word representations for the task model, while in model ensemble, it is difficult for the individual task models to affect each other.

6 Discussion: Practical Usability of ACE

Concatenating multiple embeddings is a commonly used approach to improve accuracy of structured prediction. However, such approaches can be computationally costly as multiple language models are used as input. ACE is more practical than concatenating all embeddings as it can remove those embeddings that are not very useful in the concatenation. Moreover, ACE models can be used to guide the training of weaker models through techniques such as knowledge distillation in structured prediction (Kim and Rush, 2016; Kuncoro et al., 2016; Wang et al., 2020a, 2021b), leading to models that are both stronger and faster.

7 Conclusion

In this paper, we propose Automated Concatenation of Embeddings, which automatically searches for better embedding concatenation for structured prediction tasks. We design a simple search space and use the reinforcement learning with a novel reward function to efficiently guide the controller to search for better embedding concatenations. We take the change of embedding concatenations into the reward function design and show that our new reward function is stronger than the simpler ones. Results show that ACE outperforms strong baselines. Together with fine-tuned embeddings, ACE achieves state-of-the-art performance in 6 tasks over 21 datasets.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61976139) and by Alibaba Group through Alibaba Innovative Research Program. We thank Chengyue Jiang for his comments and suggestions on writing.

References

Alan Akbik, Tanja Bergmann, and Roland Vollgraf. 2019. Pooled contextualized embeddings for named entity recognition. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Lan-*

guage Technologies, Volume 1 (Long and Short Papers), pages 724–728, Minneapolis, Minnesota. Association for Computational Linguistics.

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.

Peter J Angeline, Gregory M Saunders, and Jordan B Pollack. 1994. An evolutionary algorithm that constructs recurrent neural networks. *IEEE transactions on Neural Networks*, 5(1):54–65.

Alexei Baevski, Sergey Edunov, Yinhan Liu, Luke Zettlemoyer, and Michael Auli. 2019. Cloze-driven pretraining of self-attention networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5360–5369, Hong Kong, China. Association for Computational Linguistics.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2017. Designing neural network architectures using reinforcement learning. In *International Conference on Learning Representations*.

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.

Sabine Buchholz and Erwin Marsi. 2006. CoNLL-X shared task on multilingual dependency parsing. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 149–164, New York City. Association for Computational Linguistics.

Luoxin Chen, Weitong Ruan, Xinyue Liu, and Jianhua Lu. 2020. SeqVAT: Virtual adversarial training for semi-supervised sequence labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8801–8811, Online. Association for Computational Linguistics.

Kevin Clark, Minh-Thang Luong, Christopher D. Manning, and Quoc Le. 2018. Semi-supervised sequence modeling with cross-view training. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1914–1925, Brussels, Belgium. 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. 2020. 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.

Steven J. DeRose. 1988. Grammatical category disambiguation by statistical optimization. *Computational Linguistics*, 14(1):31–39.

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. 2017. Deep biaffine attention for neural dependency parsing. In *International Conference on Learning Representations*.

Timothy Dozat and Christopher D. Manning. 2018. Simpler but more accurate semantic dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 484–490, Melbourne, Australia. Association for Computational Linguistics.

Thomas Elsken, Jan-Hendrik Metzen, and Frank Hutter. 2018. Simple and efficient architecture search for convolutional neural networks. In *International Conference on Learning Representations workshop*.

Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. 2019. Neural architecture search: A survey. *Journal of Machine Learning Research*, 20:1–21.

Daniel Fernández-González and Carlos Gómez-Rodríguez. 2020. Transition-based semantic dependency parsing with pointer networks. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7035–7046, Online. Association for Computational Linguistics.

Dario Floreano, Peter Dürr, and Claudio Mattiussi. 2008. Neuroevolution: from architectures to learning. *Evolutionary intelligence*, 1(1):47–62.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. 2019. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7036–7045.

Kevin Gimpel, Nathan Schneider, Brendan O'Connor, Dipanjan Das, Daniel Mills, Jacob Eisenstein, Michael Heilman, Dani Yogatama, Jeffrey Flanigan, and Noah A. Smith. 2011. Part-of-speech tagging for Twitter: Annotation, features, and experiments. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 42–47, Portland, Oregon, USA. Association for Computational Linguistics.

David E Goldberg and Kalyanmoy Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, pages 69–93. Elsevier.

Tao Gui, Qi Zhang, Jingjing Gong, Minlong Peng, Di Liang, Keyu Ding, and Xuanjing Huang. 2018. Transferring from formal newswire domain with hypernet for Twitter POS tagging. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2540–2549, Brussels, Belgium. Association for Computational Linguistics.

Tao Gui, Qi Zhang, Haoran Huang, Minlong Peng, and Xuanjing Huang. 2017. Part-of-speech tagging for Twitter with adversarial neural networks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2411–2420, Copenhagen, Denmark. Association for Computational Linguistics.

Han He and Jinho Choi. 2020. Establishing strong baselines for the new decade: Sequence tagging, syntactic and semantic parsing with bert. In *The Thirty-Third International Flairs Conference*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '04, page 168–177, New York, NY, USA. Association for Computing Machinery.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. 2017. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708.

Zixia Jia, Youmi Ma, Jiong Cai, and Kewei Tu. 2020. Semi-supervised semantic dependency parsing using CRF autoencoders. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6795–6805, Online. Association for Computational Linguistics.

Rafal Jozefowicz, Wojciech Zaremba, and Ilya Sutskever. 2015. An empirical exploration of recurrent network architectures. In *International conference on machine learning*, pages 2342–2350.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1317–1327, Austin, Texas. Association for Computational Linguistics.

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

Dan Kondratyuk and Milan Straka. 2019. 75 languages, 1 model: Parsing Universal Dependencies universally. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2779–2795, Hong Kong, China. Association for Computational Linguistics.

Adhiguna Kuncoro, Miguel Ballesteros, Lingpeng Kong, Chris Dyer, and Noah A. Smith. 2016. Distilling an ensemble of greedy dependency parsers into one MST parser. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1744–1753, Austin, Texas. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Liam Li and Ameet Talwalkar. 2020. Random search and reproducibility for neural architecture search. In *Uncertainty in Artificial Intelligence*, pages 367–377. PMLR.

Xin Li, Lidong Bing, Wenxuan Zhang, and Wai Lam. 2019. Exploiting BERT for end-to-end aspect-based sentiment analysis. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 34–41, Hong Kong, China. Association for Computational Linguistics.

Zuchao Li, Hai Zhao, and Kevin Parnow. 2020. Global greedy dependency parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8319–8326.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L Yuille, and Li Fei-Fei. 2019a. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 82–92.

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2018a. Hierarchical representations for efficient architecture search. In *International Conference on Learning Representations*.

Yijia Liu, Yi Zhu, Wanxiang Che, Bing Qin, Nathan Schneider, and Noah A. Smith. 2018b. Parsing tweets into Universal Dependencies. 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 965–975, New Orleans, Louisiana. Association for Computational Linguistics.

Yijin Liu, Fandong Meng, Jinchao Zhang, Jinan Xu, Yufeng Chen, and Jie Zhou. 2019b. GCDT: A global context enhanced deep transition architecture for sequence labeling. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2431–2441, Florence, Italy. Association for Computational Linguistics.

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

Ilya Loshchilov and Frank Hutter. 2018. Decoupled weight decay regularization. In *International Conference on Learning Representations*.

Jouni Luoma and Sampo Pyysalo. 2020. Exploring cross-sentence contexts for named entity recognition with BERT. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 904–914, Barcelona, Spain (Online). International Committee on Computational Linguistics.

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.

Xuezhe Ma, Zecong Hu, Jingzhou Liu, Nanyun Peng, Graham Neubig, and Eduard Hovy. 2018. Stack-pointer networks for dependency parsing. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1403–1414, Melbourne, Australia. Association for Computational Linguistics.

Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajič. 2005. Non-projective dependency parsing using spanning tree algorithms. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 523–530, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

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.

Geoffrey Miller, Peter Todd, and Shailesh Hegde. 1989. Designing neural networks using genetic algorithms. In *3rd International Conference on Genetic Algorithms*, pages 379–384.

Khalil Mrini, Franck Dernoncourt, Quan Hung Tran, Trung Bui, Walter Chang, and Ndapa Nakashole. 2020. Rethinking self-attention: Towards interpretability in neural parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 731–742, Online. Association for Computational Linguistics.

Dat Quoc Nguyen, Thanh Vu, and Anh Tuan Nguyen. 2020. BERTweet: A pre-trained language model for English Tweets. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*.

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.

Stephan Oepen, Marco Kuhlmann, Yusuke Miyao, Daniel Zeman, Dan Flickinger, Jan Hajic, Angelina Ivanova, and Yi Zhang. 2014. Semeval 2014 task 8: Broad-coverage semantic dependency parsing. *SemEval 2014*.

Olutobi Owoputi, Brendan O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. 2013. Improved part-of-speech tagging for online conversational text with word clusters. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 380–390, Atlanta, Georgia. Association for Computational Linguistics.

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.

Hieu Pham, Melody Guan, Barret Zoph, Quoc Le, and Jeff Dean. 2018a. Efficient neural architecture search via parameters sharing. In *International Conference on Machine Learning*, pages 4095–4104.

Hieu Pham, Melody Y Guan, Barret Zoph, Quoc V Le, and Jeff Dean. 2018b. Efficient neural architecture search via parameter sharing. In *International Conference on Machine Learning*.

Telmo Pires, Eva Schlinger, and Dan Garrette. 2019. How multilingual is multilingual BERT? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4996–5001, Florence, Italy. Association for Computational Linguistics.

Maria Pontiki, Dimitris Galanis, Haris Papageorgiou, Ion Androutsopoulos, Suresh Manandhar, Mohammad AL-Smadi, Mahmoud Al-Ayyoub, Yanyan Zhao, Bing Qin, Orphée De Clercq, Véronique Hoste, Marianna Apidianaki, Xavier Tannier, Natalia Loukachevitch, Evgeniy Kotelnikov, Nuria Bel, Salud María Jiménez-Zafra, and Gülşen Eryiğit. 2016. SemEval-2016 task 5: Aspect based sentiment analysis. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval-2016)*, pages 19–30, San Diego, California. Association for Computational Linguistics.

Maria Pontiki, Dimitris Galanis, Haris Papageorgiou, Suresh Manandhar, and Ion Androutsopoulos. 2015. SemEval-2015 task 12: Aspect based sentiment analysis. In *Proceedings of the 9th International Workshop on Semantic Evaluation (SemEval 2015)*, pages 486–495, Denver, Colorado. Association for Computational Linguistics.

Maria Pontiki, Dimitris Galanis, John Pavlopoulos, Harris Papageorgiou, Ion Androutsopoulos, and Suresh Manandhar. 2014. SemEval-2014 task 4: Aspect based sentiment analysis. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 27–35, Dublin, Ireland. Association for Computational Linguistics.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. 2019. Regularized evolution for image classifier architecture search. In *Proceedings of the aaai conference on artificial intelligence*, volume 33, pages 4780–4789.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911.

Alan Ritter, Sam Clark, Mausam, and Oren Etzioni. 2011. Named entity recognition in tweets: An experimental study. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 1524–1534, Edinburgh, Scotland, UK. Association 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.

Tal Schuster, Ori Ram, Regina Barzilay, and Amir Globerson. 2019. Cross-lingual alignment of contextual word embeddings, with applications to zero-shot dependency parsing. 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 1599–1613, Minneapolis, Minnesota. Association for Computational Linguistics.

David R So, Chen Liang, and Quoc V Le. 2019. The evolved transformer. In *International Conference on Machine Learning*.

Kenneth O Stanley and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation*, 10(2):99–127.

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.

Masanori Suganuma, Shinichi Shirakawa, and Tomoharu Nagao. 2017. A genetic programming approach to designing convolutional neural network architectures. In *Proceedings of the genetic and evolutionary computation conference*, pages 497–504.

Beth M. Sundheim. 1995. Named entity task definition, version 2.1. In *Proceedings of the Sixth Message Understanding Conference*, pages 319–332.

Richard S Sutton and Andrew G Barto. 1992. *Reinforcement learning: An introduction*. MIT press.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. 2016. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826.

Lucien Tesnière. 1959. éléments de syntaxe structurale. *Editions Klincksieck*.

Erik F. Tjong Kim Sang. 2002. Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*.

Erik F. Tjong Kim Sang and Sabine Buchholz. 2000. Introduction to the CoNLL-2000 shared task chunking. In *Fourth Conference on Computational Natural Language Learning and the Second Learning Language in Logic Workshop*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

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, Fei Huang, and Kewei Tu. 2020a. Structure-level knowledge distillation for multilingual sequence labeling. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3317–3330, Online. Association for Computational Linguistics.

Xinyu Wang, Yong Jiang, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021a. Improving Named Entity Recognition by External Context Retrieving and Cooperative Learning. 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. 2020b. More embeddings, better sequence labelers? In *Findings of EMNLP*, Online.

Xinyu Wang, Yong Jiang, Zhaohui Yan, Zixia Jia, Nguyen Bach, Tao Wang, Zhongqiang Huang, Fei Huang, and Kewei Tu. 2021b. Structural Knowledge Distillation: Tractably Distilling Information for Structured Predictor. 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 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.

Zhenkai Wei, Yu Hong, Bowei Zou, Meng Cheng, and Jianmin Yao. 2020. Don't eclipse your arts due to small discrepancies: Boundary repositioning with a pointer network for aspect extraction. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3678–3684, Online. Association for Computational Linguistics.

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

Martin Wistuba. 2018. Deep learning architecture search by neuro-cell-based evolution with function-preserving mutations. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 243–258. Springer.

Shijie Wu and Mark Dredze. 2019. Beto, bentz, becas: The surprising cross-lingual effectiveness of BERT. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 833–844, Hong Kong, China. Association for Computational Linguistics.

L. Xie and A. Yuille. 2017. Genetic cnn. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 1388–1397.

Hu Xu, Bing Liu, Lei Shu, and Philip Yu. 2019. BERT post-training for review reading comprehension and aspect-based sentiment analysis. 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 2324–2335, Minneapolis, Minnesota. Association for Computational Linguistics.

Hu Xu, Bing Liu, Lei Shu, and Philip S. Yu. 2018. Double embeddings and CNN-based sequence labeling for aspect extraction. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 592–598, Melbourne, Australia. Association for Computational Linguistics.

Ikuya Yamada, Akari Asai, Hiroyuki Shindo, Hideaki Takeda, and Yuji Matsumoto. 2020. LUKE: Deep contextualized entity representations with entity-aware self-attention. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6442–6454, Online. Association for Computational Linguistics.

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.

Juntao Yu, Bernd Bohnet, and Massimo Poesio. 2020. Named entity recognition as dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6470–6476, Online. Association for Computational Linguistics.

Yu Zhang, Zhenghua Li, and Min Zhang. 2020. Efficient second-order TreeCRF for neural dependency parsing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 3295–3305, Online. Association for Computational Linguistics.

Zhao Zhong, Junjie Yan, Wei Wu, Jing Shao, and Cheng-Lin Liu. 2018. Practical block-wise neural network architecture generation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2423–2432.

Junru Zhou and Hai Zhao. 2019. Head-Driven Phrase Structure Grammar parsing on Penn Treebank. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2396–2408, Florence, Italy. Association for Computational Linguistics.

Wei Zhu, Xiaoling Wang, Xipeng Qiu, Yuan Ni, and Guotong Xie. 2020. Autotrans: Automating transformer design via reinforced architecture search. *arXiv preprint arXiv:2009.02070*.

Barret Zoph and Quoc V Le. 2017. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*.

Barret Zoph, Vijay Vasudevan, Jonathon Shlens, and Quoc V Le. 2018. Learning transferable architectures for scalable image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8697–8710.

A Detailed Configurations

Evaluation To evaluate our models, We use F1 score to evaluate NER, Chunking and AE, use accuracy to evaluate POS Tagging, use unlabeled attachment score (UAS) and labeled attachment score (LAS) to evaluate DP, and use labeled F1 score to evaluate SDP.

Task Models and Controller For sequence-structured tasks (i.e., NER, POS tagging, chunking, aspect extraction), we use a batch size of 32 sentences and an SGD optimizer with a learning rate of 0.1. We anneal the learning rate by 0.5 when there is no accuracy improvement on the development set for 5 epochs. We set the maximum training epoch to 150. For graph-structured tasks (i.e., DP and SDP), we use Adam (Kingma and Ba, 2015) to optimize the model with a learning rate of 0.002. We anneal the learning rate by 0.75 for every 5000 iterations following Dozat and Manning (2017). We set the maximum training epoch to 300. For DP, we run the maximum spanning tree (McDonald et al., 2005) algorithm to output valid trees in testing. We fix the hyper-parameters of the task models.

We tune the learning rate for the controller among $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ and the discount factor among $\{0.1, 0.3, 0.5, 0.7, 0.9\}$ on the same dataset in Section 5.2. We search for the hyper-parameter through grid search and find a learning rate of 0.1 and a discount factor of 0.5 performs the best on the development set. The controller's parameters are initialized to all 0 so that each candidate is selected evenly in the first two time steps. We use Stochastic Gradient Descent (SGD) to optimize the controller. The training time depends on the task and dataset size. Take the CoNLL English NER dataset as an example. It takes 45 GPU hours to train the controller for 30 steps on a single Tesla P100 GPU, which is an acceptable training time in practice.

Sources of Embeddings The sources of the embeddings that we used are listed in Table 7.

B Additional Analysis

B.1 Document-Level and Sentence-Level Representations

Recently, models with document-level word representations extracted from transformer-based embeddings significantly outperform models with sentence-level word representations in NER (Devlin et al., 2019; Yu et al., 2020; Yamada et al., 2020). However, there are a lot of application scenarios that document contexts are unavailable. We replace the document-level word representations from transformer-based embeddings (i.e., XLM-R and BERT embeddings) with the sentence-level word representations. Results are shown in Table 8. We report the test results of `All` to show how the gap between ACE and `All` changes with different kinds of representations. We report the test accuracy of the models with the highest development accuracy following Yamada et al. (2020) for a fair comparison. Empirical results show that the document-level representations can significantly improve the accuracy of ACE. Comparing with models with sentence-level representations, the averaged accuracy gap between ACE and `All` is enhanced from 0.7 to 1.7 with document-level representations, which shows that the advantage of ACE becomes stronger with document-level representations.

B.2 Fine-tuned Models Versus ACE

To fine-tune the embeddings, we use AdamW (Loshchilov and Hutter, 2018) optimizer with a learning rate of 5×10^{-6} and trained the contextualized embeddings with the task for 10 epochs. We use a batch size of 32 for BERT, M-BERT and use a batch size of 4 for XLM-R, RoBERTa and XLNet. A comparison between ACE and the fine-tuned embeddings that we used in ACE is shown in Table 9, 10. Results show that ACE can further improve the accuracy of fine-tuned models.

EMBEDDING	RESOURCE	URL
GloVe	Pennington et al. (2014)	nlp.stanford.edu/projects/glove
fastText	Bojanowski et al. (2017)	github.com/facebookresearch/fastText
ELMo	Peters et al. (2018)	github.com/allenai/allennlp
ELMo (Other languages)	Schuster et al. (2019)	github.com/TalSchuster/CrossLingualContextualEmb
BERT	Devlin et al. (2019)	huggingface.co/bert-base-cased
M-BERT	Devlin et al. (2019)	huggingface.co/bert-base-multilingual-cased
BERT (Dutch)	wietsedv	huggingface.co/wietsedv/bert-base-dutch-cased
BERT (German)	dbmdz	huggingface.co/bert-base-german-dbmdz-cased
BERT (Spanish)	dccuchile	huggingface.co/dccuchile/bert-base-spanish-wwm-cased
BERT (Turkish)	dbmdz	huggingface.co/dbmdz/bert-base-turkish-cased
XLM-R	Conneau et al. (2020)	huggingface.co/xlm-roberta-large
RoBERTa	Liu et al. (2019c)	huggingface.co/roberta-large
XLNet	Yang et al. (2019)	huggingface.co/xlnet-large-cased

Table 7: The embeddings we used in our experiments. The URL is where we downloaded the embeddings.

	de	de_{06}	en	es	nl
All+sent	86.8	90.1	93.3	90.0	94.4
ACE+sent	87.1	90.5	93.6	92.4	94.6
BERT (2019)	-	-	92.8	-	-
Akbik et al. (2019)	-	88.3	93.2	-	90.4
Yu et al. (2020)	86.4	90.3	93.5	90.3	94.7
Yamada et al. (2020)	-	-	94.3	-	-
Luoma and Pyysalo (2020)	87.3	-	93.7	88.3	93.5
Wang et al. (2021a)	-	-	93.9	-	-
All+doc	87.5	90.8	94.0	90.7	93.7
ACE+doc	**88.3**	**91.7**	**94.6**	**95.9**	**95.7**

Table 8: Comparison of models with and without document contexts on NER. +sent/+doc: models with sentence-/document-level embeddings.

B.3 Retraining

Most of the work (Zoph and Le, 2017; Zoph et al., 2018; Pham et al., 2018b; So et al., 2019; Zhu et al., 2020) in NAS retrains the searched neural architecture from scratch so that the hyper-parameters of the searched model can be modified or trained on larger datasets. To show whether our searched embedding concatenation is helpful to the task, we retrain the task model with the embedding concatenations on the same dataset from scratch. For the experiment, we use the same dataset settings as in Section 4.3.1. We train the searched embedding concatenation of each run from ACE 3 times (therefore, 9 runs for each dataset).

Table 12 shows the comparison between retrained models with the searched embedding concatenation from ACE and `All`. The results show that the retrained models are competitive with ACE in SDP and in chunking. However, in another three tasks, the retrained models perform inferior to ACE. The possible reason is that the model at each step is initialized by the trained model of previous step. The retrained models outperform `All` in all tasks, which shows the effectiveness of the searched embedding concatenations.

B.4 Effect of Embeddings in the Searched Embedding Concatenations

There is no clear conclusion on what concatenation of embeddings is helpful to most of the tasks. We analyze the best searched embedding concatenations by ACE over different structured outputs, semantic/syntactic type, and monolingual/multilingual tasks. The percentage of each embedding selected by the best concatenations from all experiments of ACE are shown in Table 13. The best embedding concatenation varies over the output structure, syntactic/semantic level of understanding, and the language. The experimental results show that it is essential to select embeddings for each kind of task separately. However, we also find that the embeddings are strong in specific settings. In comparison to the sequence-structured and graph-structured tasks, we find that M-BERT and ELMo are only frequently selected in sequence-structured tasks while XLM-R embeddings are always selected in graph-structured tasks. For Flair embeddings, the forward and backward model are evenly selected. We suspect one direction of Flair embeddings is strong enough. Therefore concatenating the embeddings from two directions together cannot further improve the accuracy. For non-contextualized embeddings, pretrained word embeddings are frequently selected in sequence-structured tasks, and character embeddings are not. When we dig deeper into the semantic and syntactic type of these two structured outputs, we find that in all best concatenations, BERT embeddings are selected in all syntactic sequence-structured tasks, and Flair, M-Flair, word, and XLM-R embeddings are selected in syntactic graph-structured tasks. In multilingual tasks, all best concatenations in multilingual NER tasks select M-BERT embeddings while M-BERT is rarely selected in multilingual AE tasks. The monolingual Flair embeddings are always selected in NER tasks, and XLM-R is more

frequently selected in multilingual tasks than monolingual sequence-structured tasks (**SS**).

	NER					POS		
	de	de (Revised)	en	es	nl	Ritter	ARK	TB-v2
BERT+Fine-tune	76.9	79.4	89.2	83.3	83.8	91.2	91.7	94.4
MBERT+Fine-tune	81.6	86.7	92.0	87.1	87.2	90.8	91.5	93.9
XLM-R+Fine-tune	87.7	91.4	94.1	89.3	95.3	92.3	93.7	95.4
RoBERTa+Fine-tune	-	-	93.9	-	-	92.0	93.9	95.4
XLNET+Fine-tune	-	-	93.6	-	-	88.4	92.4	94.4
ACE+Fine-tune	**88.3**	**91.7**	**94.6**	**95.9**	**95.7**	**93.4**	**94.4**	**95.8**

Table 9: A comparison between ACE and the fine-tuned embeddings that are used in ACE for NER and POS tagging.

	Chunk	AE							
	CoNLL 2000	14Lap	14Res	15Res	16Res	es	nl	ru	tr
BERT+Fine-tune	96.7	81.2	87.7	71.8	73.9	76.9	73.1	64.3	75.6
MBERT+Fine-tune	96.6	83.5	85.0	69.5	73.6	74.5	72.6	71.6	58.8
XLM-R+Fine-tune	97.0	85.9	90.5	76.4	78.9	77.0	77.6	77.7	74.1
RoBERTa+Fine-tune	97.2	83.9	90.2	78.5	80.7	-	-	-	-
XLNET+Fine-tune	97.1	84.5	88.9	72.8	73.4	-	-	-	-
ACE+Fine-tune	**97.3**	**87.4**	**92.0**	**80.3**	**81.3**	**79.9**	**80.5**	**79.4**	**81.9**

Table 10: A comparison between ACE and the fine-tuned embeddings we used in ACE for chunking and AE.

	DP		SDP					
	PTB		DM		PAS		PSD	
	UAS	LAS	ID	OOD	ID	OOD	ID	OOD
BERT+Fine-tune	96.6	95.1	94.4	91.4	94.4	93.0	82.0	81.3
MBERT+Fine-tune	96.5	94.9	93.9	90.4	93.9	92.1	81.2	80.0
XLM-R+Fine-tune	96.7	95.4	94.2	90.4	94.6	93.2	82.9	81.7
RoBERTa+Fine-tune	96.9	95.6	93.0	89.3	94.3	92.8	82.0	80.6
XLNET+Fine-tune	97.0	95.6	94.2	90.6	94.8	93.4	82.7	81.8
ACE+Fine-tune	**97.2**	**95.7**	**95.6**	**92.6**	**95.8**	**94.6**	**83.8**	**83.4**

Table 11: A comparison between ACE and the fine-tuned embeddings that are used in ACE for DP and SDP.

	NER	POS	Chunk	AE	DP-UAS	DP-LAS	SDP-ID	SDP-OOD
All	92.4	90.6	96.7	73.2	96.7	95.1	94.3	90.8
Retrain	92.6	90.8	**96.8**	73.6	96.8	95.2	**94.5**	**90.9**
ACE	**93.0**	**91.7**	**96.8**	**75.6**	**96.9**	**95.3**	**94.5**	**90.9**

Table 12: A comparison among retrained models, All and ACE. We use the one dataset for each task.

	BERT	M-BERT	Char	ELMo	F	F-bw	F-fw	MF	MF-bw	MF-fw	Word	XLM-R
SS	0.81	0.74	0.37	0.85	0.70	0.48	0.59	0.78	0.59	0.41	0.81	0.70
GS	0.75	0.17	0.50	0.25	0.83	0.75	0.42	0.83	0.58	0.58	0.50	1.00
Sem. SS	0.67	0.73	0.40	0.80	0.60	0.40	0.53	0.87	0.60	0.53	0.80	0.60
Syn. SS	1.00	0.75	0.33	0.92	0.83	0.58	0.67	0.67	0.58	0.25	0.83	0.83
Sem. GS	0.78	0.22	0.67	0.33	0.78	0.67	0.56	0.78	0.56	0.67	0.33	1.00
Syn. GS	0.67	0.00	0.00	0.00	1.00	1.00	0.00	1.00	0.67	0.33	1.00	1.00
M-NER	0.67	1.00	0.56	0.83	1.00	0.78	1.00	0.89	0.78	0.44	0.78	0.89
M-AE	1.00	0.33	0.75	0.33	0.58	0.42	0.42	0.75	0.25	0.75	0.50	0.92

Table 13: The percentage of each embedding candidate selected in the best concatenations from ACE. **F** and **MF** are monolingual and multilingual Flair embeddings. We count these two embeddings are selected if one of the forward/backward (**fw/bw**) direction of Flair is selected in the concatenation. We count the **Word** embedding is selected if one of the fastText/GloVe embeddings is selected. **SS**: sequence-structured tasks. **GS**: graph-structured tasks. **Sem.**: Semantic-level tasks. **Syn.**: Syntactic-level tasks. **M-NER**: Multilingual NER tasks. **M-AE**: Multilingual AE tasks. We only use English datasets in **SS** and **GS**. English datasets are removed for **M-NER** and **M-AE**.