AutoRC: Improving BERT Based Relation Classification Models via Architecture Search

Wei Zhu[1] *
[1] East China Normal University, China

Abstract

Although BERT based relation classification (RC) models have achieved significant improvements over the traditional deep learning models, it seems that no consensus can be reached on what is the optimal architecture, since there are many design choices available. In this work, we design a comprehensive search space for BERT based RC models and employ a modified version of efficient neural architecture search (ENAS) method to automatically discover the design choices mentioned above. Experiments on eight benchmark RC tasks show that our method is efficient and effective in finding better architectures than the baseline BERT based RC models. Ablation study demonstrates the necessity of our search space design and the effectiveness of our search method. We also show that our framework can also apply to other entity related tasks like coreference resolution and span based named entity recognition (NER).

1 Introduction

The task of relation classification (RC) is to predict semantic relations between pairs of entities inside a context. It is an important NLP task since it serves as an intermediate step in variety of NLP applications. There are many works that apply deep neural networks (DNN) to relation classification (Socher et al., 2012; Zeng et al., 2014; Shen and Huang, 2016). With the rise of pre-trained language models (PLMs) (Devlin et al., 2018), a series of literature have incorporated PLMs such as BERT in RC tasks (Baldini Soares et al., 2019; Wu and He, 2019; Eberts and Ulges, 2019; Peng et al., 2019), and shows significant improvements over the traditional DNN models.

Despite great success, there is yet no consensus reached on how to represent the entity pair and their contextual sentence for a BERT based RC model. First, Baldini Soares et al. (2019) and Peng et al. (2019) use different entity identification methods. Second, Baldini Soares et al. (2019) and Wu and He (2019) use different aggregation methods of entity representations and contexts. Third, choosing which features should be considered for the classfication layer should also be determined (Eberts and Ulges, 2019). In addition, previous literature does not consider the interactions between the feature vectors.

In this work, we experiment on making the design choices in the BERT based RC model automatically, so that one can obtain an architecture that better suits the task at hand (Figure 1). Throughout this work, we will refer to our framework as $AutoRC$, which includes our search space and search method. Firstly, a comprehensive search space for the design choices that should be considered in a BERT based RC model is established. Second, to navigate on our search space, we employ reinforcement learning (RL) strategy following ENAS (Pham et al., 2018). That is, a controller generates new RC architectures, receives rewards, and updates its policy via policy gradient method. To stabilize and improve the search results, three non-trivial modifications to ENAS are proposed: a) heterogeneous parameter sharing, which is to share parameters more deeply than ENAS if the modules play similar role, and not to share if not; b) maintain multiple copies of the shared parameters which will be drawn randomly to the child models; c) search warm-ups, which is to generate and update child models without updating the controller at the beginning of the search stage.

Experiments on eight benchmark RC tasks show that our method can outperform the standard BERT based RC models. Transfer of the learned architecture across different tasks is investigated, which shows the transferred architectures can outperform

*Contact: 52205901018@stu.ecnu.edu.cn.

Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: Student Research Workshop, pages 33–43
August 5–6, 2021.

the baseline models but cannot outperform the architecture learned on this task. Ablation study of the search space demonstrates the validity of the search space design. In addition, ablation studies on the search space show the validity of our search space design, and experiments show that our proposed modifications to ENAS are effective. We also show our framework can work effectively on other entity related tasks like coreference resolution and span based NER.

The contributions of the paper can be summarized as:

- We develop a comprehensive search space and improve the BERT based RC models, in which alternatives of the input formats and the aggregation layers are applicable to other tasks.
- As far as we know, we are the first to introduce NAS for BERT based models. Our proposed methods for improving search results are effective and universally applicable.

2 Related Work

Our work is closely related to the literature on neural architecture search (NAS). The field of NAS has attracted a lot of attentions in the recent years. The goal is to find automatic mechanisms for generating new neural architectures to replace conventional handcrafted ones, or automatically deciding optimal design choices instead of manually tuning (Bergstra et al., 2011). Recently, it has been widely applied to computer vision tasks, such as image classification (Cai et al., 2018), semantic segmentation (Liu et al., 2019), object detection (Ghiasi et al., 2019), super-resolution (Ahn et al., 2018), etc. However, NAS is less well studied in the field of natural language processing (NLP), especially in information extraction (IE). Recent works (Zoph and Le, 2017; Pham et al., 2018; Liu et al., 2018) search new recurrent cells for the language modeling (LM) tasks. The evolved transformer (So et al., 2019) employs an evolution-based search algorithm to generate better transformer architectures for machine translation tasks. Zhu et al. (2021) develops a novel search space which incorporates cross-sentence attention mechanism and are able to find novel architectures for natural language understanding (NLU) tasks. In this work, we design a method that incorporate NAS to improve BERT based relation extraction models.

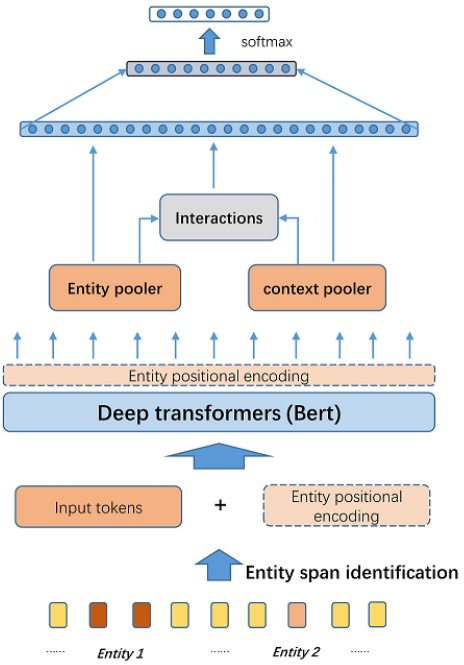

Figure 1: General architecture for a RC model.

Our work is closely related to literatures on relation extraction, especially the recent ones that take advantages of the pre-trained language models (PLMs). In terms of entity span identification, Baldini Soares et al. (2019) argues that adding entity markers to the input tokens works best, while Peng et al. (2019) shows that some RC tasks are in favor of replace entity mentions with special tokens. For feature selection, Baldini Soares et al. (2019) shows that aggregating the entity representations via start pooling works best across a panel of RC tasks. Meanwhile, Wu and He (2019) chooses average pooling for entity features. In addition, it argues that incorporating the representation of the [CLS] token is beneficial. Eberts and Ulges (2019) shows that the context between two entities serves as a strong signal on some RC task. Zhu (2020) shows that pre-training with entity spans can benefit the downstream tasks. In this work, we provide a more comprehensive overview of the design choices in BERT based RC models, and provide a solution for efficient and task-specific architecture discovery, thus alleviating NLP practitioner in the field of RE from manually or simple heuristic model tuning.

3 Search space for RC model

An overall architecture design for a RC model is shown in Figure 1. Following its bottom-up work-

(a) standard input

(b) entity_markers

(c) entity_tokens

Figure 2: How to make changes to the input sequence for entity span identification.

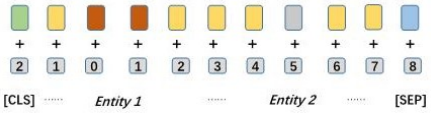

Figure 3: An example of the entity positional encoding.

flow, we will define the search space for $AutoRC$.

3.1 Formal definition of task

In this paper, we focus on learning mappings from relation statements to relation representations. Formally, let $x = [x_0, ..., x_n]$ be a sequence of tokens, and entity 1 (e_1) and entity 2 (e_2) to be the entity mentions, which is depicted at the bottom of Figure 1. The position of e_i in x is denoted by the start and end position, $s_i = (e_i^s, e_i^e)$. A relation statement is a triple $r = (x, e_1, e_2)$. Our goal is to learn a function f_θ that maps the relation statement to a fixed-length vector $h_r = f_\theta(r) \in R^d$ that represents the relation expressed in r.

Note that the two entities divide the sentence into five parts, e_1 and e_2 as entity mentions, and three contextual pieces, denoted as c_0, c_1 and c_2.

3.2 Entity span identification

In this work, we employ BERT (Devlin et al., 2018) as the encoder for the input sentences. The BERT encoder may need to distinguish the entity mentions from the context sentence to properly model the semantic representations of a relation statement. We present three different options for getting information about the entity spans s_1 and s_2 into our BERT encoder, which are depicted in Figure 2.

standard, that is, not to make any change to the input sentence (Figure 2(a)).

entity_markers. We add special tokens at the start and end of the entities to inform BERT where the two entities are in the sentence, as depicted by Figure 2(b). Formally, the sentence x becomes $[[CLS], x_0...[E1]...[/E1]...[E2]...[/E2]...x_n, [SEP]]$.

entity_tokens. This approach (Figure 2(c)) replaces the entity mentions in the sentence with special tokens. Formally, x becomes $[[CLS]...[ENTITY - 1]...[ENTITY - 2]...[SEP]]$.

3.3 Entity positional encoding

To make up for the standard input's lack of entity identification, or to further address the position of entities, one can add special entity positional encoding accompany input sequence x. As is shown in Figure 3, for entity 1, the entity positional encoding will be the distance to entity 1's starting token. [1]

Now there are two design choices. First is whether to use entity positional encoding at all. Second, as is shown in Figure 1 if using entity positional encoding, do we add this extra embedding to the embedding layer of the BERT (denoted as add_to_embedding), or do we concatenate this embedding to the output of BERT encoder (denoted as concat_to_output)?

3.4 Pooling layer

How to aggregate the entities' and contexts' hidden representations into fixed length feature vectors, i.e., what kind of poolers are used becomes the core part of the RC model architecture. In this work, we investigate 5 different poolers: average pooling (avg_pool), max pooling (denoted as max_pool), self-attention pooling (denoted as self_attn_pool), dynamic routing pooling (dr_pool) (Gong et al., 2018), and start pooling (start_pool), which is to use the reprsentation of the starting token as in Baldini Soares et al. (2019).

3.5 Output features

To select appropriate features for classifying relation types, there are many design choices. First, whether the two entity vectors should be used as features. Second, whether each contextual piece

[1] Entity positional encoding corresponds to two (one for either entity) entity positional embedding modules in the RC model, and they are randomly initialized and fine-tuned during BERT fine-tuning.

(c_0, c_1, c_2) should be added as features (Eberts and Ulges, 2019; Wu and He, 2019).

We notice that the literature does not consider the interactions of the features from different parts of the sentence, which proves to be useful in other tasks such as natural language inference (NLI) (Chen et al., 2016). Here, we consider the interaction between the two entities, and their interactions with contextual pieces. The interaction can be dot product (denoted as dot) or absolute difference (denoted as minus) between two feature vectors.

3.6 Search space

Now we are ready to define the search space formally. The search space is as follows:

- entity span identification = entity_markers, entity_tokens, standard;
- how to use entity positional embedding = null, add_to_embedding, concat_to_output;
- poolers for entity or contextual piece = avg_pool, max_pool, self_attn_pool, dr_pool, start_pool;
- whether to use the representation of entity e_i = True, False, where $i = 1, 2$;
- whether to use the representation of context c_i = True, False, where $i = 0, 1, 2$;
- Interaction between the two entities = dot, minus, null, where null means no interaction;
- Interaction between entity and contextual piece c_i = dot, minus, null, where null means no interaction, and $i = 0, 1, 2$.

Our search space contains 1.64e+8 combinations of design choices, which makes manually fine-tuning or random search impractical.

4 Search method

In this section, we first formally formulate the problem of architecture search with reinforcement learning. Then, , we discuss the search algorithm based on policy gradient. At the last part, we discuss our modifications to stabilize the search outputs.

4.1 Problem formulation

Given a search space $\mathcal{M}$ of neural architectures, and a dataset split into train set $\mathcal{D}_{train}$ and $\mathcal{D}_{valid}$, we aim to find the best architecture $m^* \in \mathcal{M}$ that

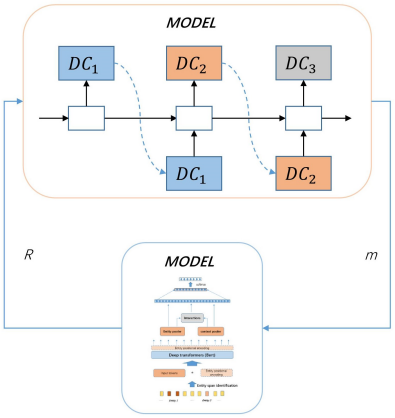

Figure 4: An illustration of the RL mechanism for architecture search.

maximizes the expected reward $\mathrm{E}[\mathcal{R}_{\mathcal{D}_{valid}}(m)]$ on the validation set $\mathcal{D}_{valid}$, i.e.,

$$m^* = \arg\max_{m \in \mathcal{M}} \mathrm{E}[R_{\mathcal{D}_{valid}}(m)]. \quad (1)$$

Figure 4 shows the reinforcement learning framework used to solve Eq 1 by continuously sampling architectures $m \in \mathcal{M}$ and evaluating the reward (performance score) $\mathcal{R}$ on the validation set $\mathcal{D}_{valid}$. First, the recurrent network generates a network description $m \in \mathcal{M}$ that corresponds to a RC model. Then, the generated model m is trained on $\mathcal{D}_{train}$ and tested on the validation set $\mathcal{D}_{valid}$. The test result is taken as a reward signal R to update the controller.

4.2 Search and evaluation

The whole procedure for model search can be divided into the search phase and evaluation phase. The search phase updates the shared parameters and the parameters for the controller in an interleaving manner, while the evaluation phase obtains multiple top-ranked models from the controller and train them till convergence on the task dataset for proper evaluations of the learned architectures.

Parameter sharing. In order to avoid training from scratch to obtain reward signals, parameter sharing is applied. The same operator is re-used for a child model if it is chosen. Specific to our architecture, the BERT encoder and the final classifier are shared for all child models. We denote the collection of all the parameters shared as Φ.

Search phase. Now we describe the interleaving optimization procedure. First, an architecture is sampled by the controller, and its network parameters are initialized with Φ. It is trained for n_c steps

(which is usually a small integer), during which Φ is updated. Then, the reward of this model is obtained on $\mathcal{D}_{valid}$. With n reward signals receive, Θ is updated using policy gradients following REINFORCE (Williams, 1992):

$$\nabla_\Theta \hat{J}(\Theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\Theta \log \pi(a_i, \Theta)(R(\Theta) - b), \tag{2}$$

where b denotes a moving average of the past rewards and it is used to reduce the variance of gradient approximation. In this work, we find $n = 1$ already works quite well. Repeating this interleaving optimization procedure for N times till the controller is well trained, then we generate k candidate architectures, evaluate them using the shared parameters, and then select the top-ranked k_e models for architecture evaluation.

Evaluation phase. In this phase, the top-ranked models are trained with the whole train set, and validated on the dev set to select the best checkpoint for prediction on the test set. Note that the shared parameters Φ are discarded in this phase, and the learned architecture is trained from scratch. To fully evaluate each architecture, we run a grid search for the optimal hyper-parameters including learning rate, batch size and warm-up steps. After the optimal combination of hyper-parameters is selected, the model is run several times to ensure replication.

4.3 Improving search

Now we propose a few methods to stabilize the search results and improve the search performance.

Heterogeneous parameter sharing. First, the reward signals directly relies on the parameter sharing mechanism, thus we should think deeper into how to design proper parameter sharing strategies for RC model search. Parameter sharing in ENAS is unconditional. Note that to much or too little parameter sharing can generate un-reliable reward signals, guiding the controller to wrong directions. Thus based on our extensive experiments, we now present our parameter sharing strategies, which we will call heterogeneous parameter sharing, since our idea is to share parameters among modules that plays similar roles in the model architectures. The details are as follows: (a) first, note that the entity span identification method entity_tokens significantly alter the original sentence, thus, it is natural for it to use a different BERT encoder in the child models. (b) since entities and contexts play quite different roles in the RC tasks, the aggregators for entities and contexts will not share parameters. Note that start_pooler and dr_pooler have a common component, which is a linear layer followed by a non-linear module, thus the linear layer will be shared in these two aggregators for entities or for contexts. However, we will use the linear layer of the BERT pooler to initialize all the linear layers of start_pooler and dr_pooler.

Multiple copies of shared parameters. Note that all child models have a BERT encoder and a classifier layer, thus parameters in these modules may over-fit quickly. Thus, during search training, we maintain multiple copies of these modules, and each time we initialize a child model, a copy of BERT encoder and classifier layer will be randomly selected from shared parameters Φ. After updating, these copies will be stored back to Φ.

Search warm-ups At the beginning of training, the shared parameters are not trained, thus reward signals generated are unreliable. Thus, at the first few epochs, the controller will generate child models to train on the dataset, but it will not be updated.

5 Experiments

Due to resource limitations, we assign up to 2 NVIDIA V100 GPU cards to each tasks.

5.1 Datasets

We run experiments on 8 different benchmark datasets, semeval10 (Hendrickx et al., 2009),[2] tacred (Zhang et al., 2017), kbp37 (Zhang and Wang, 2015), wiki80 (Han et al., 2019), deft2020 (Spala et al., 2019), i2b2 (zlem et al., 2011), ddi (Herrero-Zazo et al., 2013), chemprot (Krallinger et al., 2017). These tasks are from various domains and are different in the respects of dataset sizes, sentence length, entity mention length, etc, to demonstrate that our method is robust for various RC tasks. Detailed descriptions and statistics are provided in the Appendix.

5.2 Search protocol

During search phase, the interleaving optimization process is run 100 epochs. Throughout this work, we use the base uncased version of BERT (Devlin et al., 2018) as the sentence encoder, and its

[2]This dataset does not establish a default split for development, so for this work we adopt the same train/dev split with that provided by OpenNRE (Han et al., 2019). Thus, we cannot adopt the reported results for semeval10 on Table 1 of Baldini Soares et al. (2019).

parameters are fine-tuned to better adjust to downstream tasks. During search, 4 copies of BERT model checkpoints are maintained, 2 for method entity_tokens and 2 for the other two entity span identifiers, so each time we initialize a child model, a BERT checkpoint is randomly selected and its parameters can be updated. If the entity position embedding is concatenated after the BERT output, its size is set to be 12.

During search, each child model is trained with 4 batches of training data and evaluated on a single batch of valid data, and the evaluation batch size is 4 times the training batch size. The learning rate for the controller is set at 1e-4, and the learning rate and batch size for the sampled architectures are manually tuned to obtain better search results. During search, the number of warm-up steps for the BERT encoders is set to be equal to 0.8 of a epoch, and the warm-up steps for search is set to be 1.5 epochs.

5.3 Architecture evaluation protocol

In this work, we differentiate between a NAS method's performance and that of a learned model. We obtain the former by running architecture search 5 times. The best learned model's performance will be regarded as the NAS method's performance in each run. The best learned model in each search is also run for 10 times.

To make our results more reproducible, each learned model or each baseline model is trained for 10 times, and the mean and variance of the performance will be reported. And for evaluating the search method, after the search phase, 30 model architectures are sampled from the trained controller, and they are ranked via their performance on the valid data when they are initialized using the shared parameters. Then the top-ranked 5 models are trained from scratch till convergence on the whole training data of the task to formally evaluate their performances. The best learned model's performance of a search run is regarded as the search method's performance. In this work, we will report the mean and standard deviation of the search method performances in 5 independent runs.

To compare our methods with random search, for each task, we randomly samples 10 different models with a randomly initialized controller, since the GPU time for training 10 models is guaranteed to be larger than an entire search and evaluation process described above.

To thoroughly evaluate a learned model or a baseline model, we run a random search of 10 times on the following space for the optimal combination of the following key hyper-parameters:

- learning rate = 1e-4, 5e-5, 2e-5, 1e-5;
- training batch size = 128, 64, 32;
- warm-up steps = 0.8, 1.0 of the number of steps in an epoch.

The hyper-params for the baseline models are reported in the Appendix.

5.4 Baseline models

In this work, we select two strong baselines for comparison. The first one is **BERT-entity**, the best model from Baldini Soares et al. (2019). The second is **R-BERT** by Wu and He (2019). **BERT-entity** and **R-BERT** are implemented by OpenNRE (Han et al., 2019). The two models are special cases in our search space. The baseline models also have to go through the above reproducibility protocols. We will not compare with traditional deep-learning based model in the pre-BERT era, since **BERT-entity** significantly outperforms them.[3]

5.5 Results on Benchmark datasets

The results on the 8 benchmarks RC datasets are reported in Table 1. We report both the performance of the search methods and the performance of the best model learned on each task using $AutoRC$. For all eight tasks, $AutoRC$ successfully obtains higher average scores than the baseline models. In addition, we find that $AutoRC$ outperforms naive ENAS and random search and its results are more stable. In addition, we can see that the best learned model outperforms the baseline models significantly. One observation can be made is that the test results of the search architectures are consistently stable than the baseline, which also validates that our method are efficient in finding a task-specific model for the task at hand.

Figure 5, 6 and 7 report the best searched architectures for the deft2020, i2b2 and kbp37 tasks. We can see that learned architectures can be quite

[3]This work only considers the effects of architecture design, thus some of the SOTAs may not provide fair comparison. KnowBert (Peters et al., 2019) explicitly incorporates external KGs. Tao et al. (2019) take advantage of syntactic priors. Before submission, we run the REDN (Li and Tian, 2020) model (by using their code and re-implement by our self), but the results are not comparable to the results in their paper.

Model	semeval10	tacred	kbp37	wiki80	deft2020	i2b2	ddi	chemprot
R-BERT	88.19±0.234	69.63±0.178	64.15±0.285	85.38±0.158	60.12±0.875	81.88±0.547	75.73±0.786	66.77±0.336
BERT-entity	88.35±0.159	69.97 ± 0.198	64.20±0.273	85.35±0.141	60.19±0.723	81.94±0.691	75.66±0.712	66.86±0.393
random search	87.61±0.316	69.15±0.376	63.90±0.516	83.46±0.378	58.19±1.968	81.33±1.364	74.23±0.653	66.04±0.873
naive ENAS	88.23±0.256	69.98±0.267	64.25±0.412	85.38±0.286	61.57±0.727	82.18±0.632	75.57±0.598	66.94±0.453
AutoRC	88.53±0.212	70.06±0.242	64.32±0.414	85.46±0.143	62.87±0.632	82.76±0.587	75.72±0.532	67.15±0.367
$AR_{semeval10}$	**88.89**±0.165	-	-	-	-	-	-	-
AR_{tacred}	-	**70.87**±0.167	-	-	-	-	-	-
AR_{kbp37}	-	-	**64.96**±0.185	85.63±0.175	-	81.87±0.778	75.58±0.704	-
AR_{wiki80}	-	-	64.58±0.169	**85.98**±0.134	-	82.32±0.604	75.89±0.633	-
$AR_{deft2020}$	-	-	-	-	**63.82**±0.593	-	-	-
AR_{i2b2}	-	-	64.43±0.166	85.46±0.164	-	**83.59**±0.478	76.05±0.658	-
AR_{ddi}	-	-	64.37±0.172	85.39±0.159	-	82.92±0.454	**76.73**±0.475	-
$AR_{chemprot}$	-	-			-	-	-	**67.95**±0.283

Table 1: Test results for eight relation classification tasks. The performance metric is micro F1 for all tasks except for deft2020 which uses macro F1. Results from the baseline model are obtained with the help of OpenNRE (Han et al., 2019).

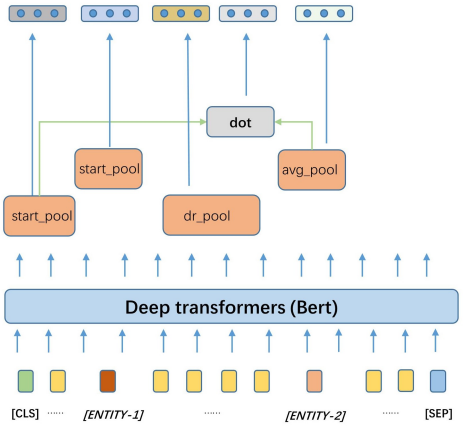

Figure 5: $AR_{deft2020}$, the best learned architecture on deft2020.

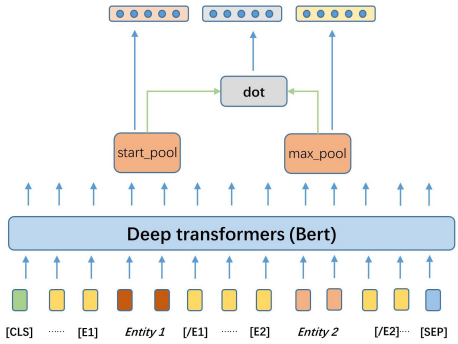

Figure 6: AR_{i2b2}, the best learned architecture on i2b2.

different, thus validating the necessity of task specificity. The learned models are different in the following three aspects. First, $AR_{deft2020}$ choose to replace entity mentions with entity tokens. We hypothesis that in deft-2020, the entities are often quite long, thus replacing entity mentions with entity tokens is beneficial for the model to understand

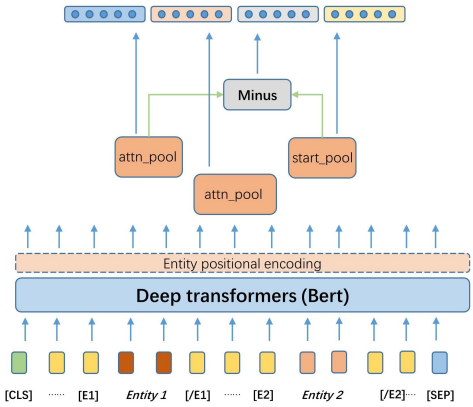

Figure 7: AR_{kbp37}, the best learned architecture on kbp37.

the contexts' structural patterns. Second, note that $AR_{deft2020}$ uses start_pool to aggregate context piece c_0, which is the representation of [CLS] token. In addition, it includes the representation of context c_1, which is also used in AR_{kbp37}. Third, $AR_{deft2020}$ incorporates the interaction between context c_0 and the two entities, while AR_{i2b2} and AR_{kbp37} include the interaction between the two entities. Differences in the learned architectures for different tasks indicate the necessity of task specific architectures, which is challenging without the help of NAS. We believe there are two aspects that can affect the learned models. First, different domains have different contexts, which may lead to different models. Second, the formulation of data. For example, in deft-2020, some extended definitions of scientific concepts are annotated as entities. Thus, the avg entity mention length (18.5) is quite different from other tasks (2.3 in "ddi").

In Table 1, we also study how does an architecture learned on one task performs on another.

Search space	deft2020	i2b2
$\mathcal{M}$	**63.82** $\pm$ 0.593	**83.59** $\pm$ 0.478
$\mathcal{M}_1$	63.45 $\pm$ 0.698	83.22 $\pm$ 0.514
$\mathcal{M}_2$	62.31 $\pm$ 0.423	82.68 $\pm$ 0.483
$\mathcal{M}_3$	61.78 $\pm$ 0.893	82.35 $\pm$ 0.558
BERT-entity	60.19 $\pm$ 0.723	81.94 $\pm$ 0.691

Table 2: Results of ablation study on the search space.

Search Method	deft2020	i2b2
naive ENAS	61.57 $\pm$ 0.727	82.18 $\pm$ 0.632
$AutoRC$	**62.87** $\pm$ 0.632	**82.76** $\pm$ 0.587
$AutoRC_1$	62.38 $\pm$ 0.689	82.42 $\pm$ 0.616
$AutoRC_2$	62.53 $\pm$ 0.672	82.56 $\pm$ 0.595
$AutoRC_3$	62.49 $\pm$ 0.708	82.61 $\pm$ 0.614

Table 3: Ablation study on the search methods.

Method	OntoNotes	CoNLL04
SpanBERT	85.3	-
SpERT	-	88.94
$AutoRC$	86.1	89.87

Table 4: Experiments on the coreference resolution and span based NER.

Note that when evaluated on a different task, an architecture's hyper-parameters are tuned again, following the procedure described in subsection 5.3. The architecture learned on kbp37, which is an open-domain dataset, AR_{kbp37}, transfer well on wiki80. But it does not perform well on the two tasks of medical domain, i2b2 and ddi. However, the learned architectures learned on i2b2 and ddi transfer well on each other and perform comparably well. The above results demonstrate that the learned models have certain ability for task transfer, but its suitability is significantly affected by the domains of the tasks.

5.6 Ablation study on the search space

We further investigate the specific contributions by the different components of the search space. For this purpose, we create three smaller search space. The first one, denoted as $\mathcal{M}_1$, which does not allow any interactions among entity features and context features. The second one, $\mathcal{M}_2$ further reduce $\mathcal{M}_1$ by limiting that the pooling operation available is the start pooling operation. The third one, $\mathcal{M}_3$, further forbid contextual features. If further limit the entity span identification method to be entity markers, the search space is reduced to the baseline **BERT-entity** model. The search and evaluation protocols on the reduced search space strictly follow the previous subsections.

Ablation study for the search space is done on deft2020 and i2b2. Results are reported in Table 2. For deft2020, alternating the method for span identification provides significant performance gain on deft2020, and interaction among features is also important. For i2b2, the most significant performance drop occurs when the pooling operations are limited, indicating that even for powerful bi-directional context encoder like BERT, considering different pooling operations are beneficial.

5.7 Ablations on the modifications for search method

In this subsection, we will show that our modifications to the search method, i.e., the naive E-NAS, are indeed effective and necessary. Here we use $AutoRC$ to denote our method, which is the combination of ENAS and our proposed modifications. We now experiment on three variations to $AutoRC$. First, $AutoRC_1$ drops heterogeneous parameter sharing, that is, all input formats share the same BERT encoder, and all context and all entity representations share the same aggregators. The second variant, $AutoRC_2$, is to maintain single copies of shared weights. The third variant, $AutoRC_3$, is the one that drops search warm-ups.

The average search performance, which is the average score of the best learned model at each search run, and their standard deviations are reported on Table 3. From the results, dropping any of three strategies we propose results in performance drop and increased variance in results. And changing the parameter sharing strategies cause the most significant performance drops on both tasks. The above results demonstrate that our proposed modifications make the reward signal during search more reliable, thus resulting in better searched architectures.

5.8 Applications to other entity related tasks

In Table 4, we apply our $AutoRC$ framework to the other two entity related tasks, i.e., coreference resolution and span based NER. $AutoRC$ can directly apply to coreference resolution since it essentially asks the model to determine whether an expression refers to an entity. It can also be applied to span based NER since it asks the model to determine whether a span in the sentence is an entity.

We experiment on the OntoNotes coreference resolution benchmark (Pradhan et al., 2012). The metric is MUC F1 and we choose the state-of-the-art (SOTA) SpanBERT (Joshi et al., 2019) as base-

line. The results show that our $AutoRC$ framework can effectively improve the performances of the SpanBERT checkpoint.

We experiment on the NER task of CoNLL04 (Roth and tau Yih, 2004), which uses entity level F1 as metric. Eberts and Ulges (2020) provides a SOTA baseline. The results show that performance improves via $AutoRC$.

6 Conclusion

In this work, we first construct a comprehensive search space to include many import design choices for a BERT based RC model. Then we design an efficient search method with the help of RL to navigate on this search space. To improve the search results, parameter sharing strategies different from ENAS are designed. To avoid over-fitting, we maintain multiple copies of shared weights during search. To stabilize the reward signal, search warm-ups are applied. Experiments on eight benchmark RC tasks show that our method can outperform the standard BERT based RC model significantly. Ablation study shows our search space design and proposed modifications are effective.

References

Namhyuk Ahn, Byungkon Kang, and Kyung-Ah Sohn. 2018. Fast, accurate, and lightweight super-resolution with cascading residual network. In *ECCV*.

Livio Baldini Soares, Nicholas FitzGerald, Jeffrey Ling, and Tom Kwiatkowski. 2019. Matching the blanks: Distributional similarity for relation learning. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2895–2905, Florence, Italy. Association for Computational Linguistics.

James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. 2011. Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

Han Cai, Ligeng Zhu, and Song Han. 2018. Proxylessnas: Direct neural architecture search on target task and hardware. *arXiv preprint arXiv:1812.00332*.

Qian Chen, Xiaodan Zhu, Zhenhua Ling, Si Wei, Hui Jiang, and Diana Inkpen. 2016. Enhanced lstm for natural language inference. *arXiv preprint arXiv:1609.06038*.

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*.

Markus Eberts and A. Ulges. 2020. Span-based joint entity and relation extraction with transformer pre-training. In *ECAI*.

Markus Eberts and Adrian Ulges. 2019. Span-based Joint Entity and Relation Extraction with Transformer Pre-training. *arXiv e-prints*, page arXiv:1909.07755.

Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V. Le. 2019. Nas-fpn: Learning scalable feature pyramid architecture for object detection. In *CVPR*.

Jingjing Gong, Xipeng Qiu, Shaojing Wang, and Xuanjing Huang. 2018. Information aggregation via a dynamic routing for sequence encoding. *arXiv preprint arXiv:1806.01501*.

Xu Han, Tianyu Gao, Yuan Yao, Deming Ye, Zhiyuan Liu, and Maosong Sun. 2019. OpenNRE: An open and extensible toolkit for neural relation extraction. In *Proceedings of EMNLP-IJCNLP: System Demonstrations*, pages 169–174.

Xu Han, Hao Zhu, Pengfei Yu, Ziyun Wang, Yuan Yao, Zhiyuan Liu, and Maosong Sun. 2018. FewRel: A large-scale supervised few-shot relation classification dataset with state-of-the-art evaluation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4803–4809, Brussels, Belgium. Association for Computational Linguistics.

Iris Hendrickx, Su Nam Kim, Zornitsa Kozareva, Preslav Nakov, Diarmuid Ó Séaghdha, Sebastian Padó, Marco Pennacchiotti, Lorenza Romano, and Stan Szpakowicz. 2009. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In *Proceedings of the Workshop on Semantic Evaluations: Recent Achievements and Future Directions*, SEW '09, page 94–99, USA. Association for Computational Linguistics.

María Herrero-Zazo, Isabel Segura-Bedmar, Paloma Martínez, and Thierry Declerck. 2013. The ddi corpus: An annotated corpus with pharmacological substances and drug–drug interactions. *Journal of biomedical informatics*, 46(5):914–920.

Mandar Joshi, Danqi Chen, Y. Liu, Daniel S. Weld, Luke Zettlemoyer, and Omer Levy. 2019. Spanbert: Improving pre-training by representing and predicting spans. *Transactions of the Association for Computational Linguistics*, 8:64–77.

Martin Krallinger, Obdulia Rabal, Saber A Akhondi, et al. 2017. Overview of the biocreative vi chemical-protein interaction track. In *Proceedings of the sixth BioCreative challenge evaluation workshop*, volume 1, pages 141–146.

Cheng Li and Ye Tian. 2020. Downstream Model Design of Pre-trained Language Model for Relation Extraction Task. *arXiv e-prints*, page arXiv:2004.03786.

Chenxi Liu, Liang-Chieh Chen, Florian Schroff, Hartwig Adam, Wei Hua, Alan L. Yuille, and Li Fei-Fei. 2019. Auto-deeplab: Hierarchical neural architecture search for semantic image segmentation. In *CVPR*.

H. Liu, K. Simonyan, and Y. Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Yifan Peng, Shankai Yan, and Zhiyong Lu. 2019. Transfer learning in biomedical natural language processing: An evaluation of bert and elmo on ten benchmarking datasets. In *Proceedings of the 2019 Workshop on Biomedical Natural Language Processing (BioNLP 2019)*, pages 58–65.

Matthew E. Peters, Mark Neumann, Robert L Logan, Roy Schwartz, Vidur Joshi, Sameer Singh, and Noah A. Smith. 2019. Knowledge enhanced contextual word representations. In *EMNLP*.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient Neural Architecture Search via Parameter Sharing. *arXiv e-prints*, page arXiv:1802.03268.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. 2018. Efficient neural architecture search via parameter sharing. In *ICML*.

Sameer Pradhan, Alessandro Moschitti, N. Xue, O. Uryupina, and Yuchen Zhang. 2012. Conll-2012 shared task: Modeling multilingual unrestricted coreference in ontonotes. In *EMNLP-CoNLL Shared Task*.

D. Roth and Wen tau Yih. 2004. A linear programming formulation for global inference in natural language tasks. In *CoNLL*.

Yatian Shen and Xuanjing Huang. 2016. Attention-based convolutional neural network for semantic relation extraction. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2526–2536, Osaka, Japan. The COLING 2016 Organizing Committee.

David R So, Chen Liang, and Quoc V Le. 2019. The evolved transformer. *arXiv preprint arXiv:1901.11117*.

Richard Socher, Brody Huval, Christopher D. Manning, and Andrew Y. Ng. 2012. Semantic compositionality through recursive matrix-vector spaces. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

Sasha Spala, Nicholas A. Miller, Yiming Yang, Franck Dernoncourt, and Carl Dockhorn. 2019. DEFT: A corpus for definition extraction in free- and semi-structured text. In *Proceedings of the 13th Linguistic Annotation Workshop*, pages 124–131, Florence, Italy. Association for Computational Linguistics.

Qiongxing Tao, Xiangfeng Luo, and Hao Wang. 2019. Enhancing Relation Extraction Using Syntactic Indicators and Sentential Contexts. *arXiv e-prints*, page arXiv:1912.01858.

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

Shanchan Wu and Yifan He. 2019. Enriching Pre-trained Language Model with Entity Information for Relation Classification. *arXiv e-prints*, page arXiv:1905.08284.

D. Zeng, K. Liu, S. Lai, G. Zhou, and J. Zhao. 2014. Relation classification via convolutional deep neural network. *the 25th International Conference on Computational Linguistics: Technical Papers*, pages 2335–2344.

Dongxu Zhang and Dong Wang. 2015. Relation Classification via Recurrent Neural Network. *arXiv e-prints*, page arXiv:1508.01006.

Yuhao Zhang, Victor Zhong, Danqi Chen, Gabor Angeli, and Christopher D. Manning. 2017. Position-aware attention and supervised data improve slot filling. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 35–45, Copenhagen, Denmark. Association for Computational Linguistics.

Wei Zhu. 2020. Mvp-bert: Redesigning vocabularies for chinese bert and multi-vocab pretraining. *ArXiv*, abs/2011.08539.

Wei Zhu, Yuan Ni, Xiaoling Wang, and Guotong Xie. 2021. Discovering better model architectures for medical query understanding. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 230–237, Online. Association for Computational Linguistics.

Uzuner zlem, Brett R South, Shuying Shen, and DuVall Scott L. 2011. 2010 i2b2/va challenge on concepts, assertions, and relations in clinical text. *Journal of the American Medical Informatics Association Jamia*, 1(5):5.

B. Zoph and Q.V. Le. 2017. Neural architecture search with reinforcement learning. In *ICLR*.

A Benchmark datasets

Here we include introductions to the benchmark datasets we investigate. And the basic statistics and performance metrics are included in Table 5.

SemEval-2010 Task 8 (Hendrickx et al., 2009) (denoted as semeval10) This dataset does not establish a default split for development, so for this work we adopt the same train/dev split with that provided by OpenNRE (Han et al., 2019).

Dataset	# labels	Train	Dev	Test	sent length	Metrics
semeval2010	19	6508	1494	2718	19.09	micro F1
tacred	42	75,050	25,764	18,660	36.2	micro F1
kbp37	37	15917	1724	3405	31.09	micro F1
wiki80	80	40320	10080	5600	24.93	micro F1
deft2020	6	16727	963	1139	72.11	macro F1
i2b2	8	2496	624	6293	24.33	micro F1
ddi	5	18779	7244	5761	45.03	micro F1
chemprot	6	19460	11820	16943	49.69	micro F1

Table 5: Overview of datasets in experiments.

Wiki80 (denoted as wiki80) This dataset (Han et al., 2019) is derived from FewRel (Han et al., 2018), a large scale few-shot dataset. Since Wiki80 only has a train/val split, we randomly split the train set into a train set and val set (with 8:2 ratio), and treat the original validation set as the test set.

KBP-37 (Zhang and Wang, 2015) (denoted as kbp37). This dataset is a revision of MIML-RE annotation dataset, provided by Gabor Angeli et al. (2014). They use both the 2010 and 2013 KBP official document collections, as well as a July 2013 dump of Wikipedia as the text corpus for annotation.

DEFT-2020 Subtask 3 (denoted as deft2020) This dataset also serves as the task 6 of SemEval 2020 shared tasks. This RC task have to overcome longer contexts, longer entity mentions, and more imbalanced relation types. (Spala et al., 2019)

i2b2 2010 (denoted as i2b2) shared task collection consists of 170 medical documents for training and 256 documents for testing, which is the subset of the original dataset (zlem et al., 2011).

ChemProt (denoted as chemprot) consists of 1,820 PubMed abstracts with chemical-protein interactions annotated by domain experts and was used in the BioCreative VI text mining chemical-protein interactions shared task (Krallinger et al., 2017) [4].

DDI extraction 2013 corpus (denoted as ddi) is a collection of 792 texts selected from the DrugBank database and other 233 Medline abstracts (Herrero-Zazo et al., 2013).[5]

B Hyper-params for models on different tasks

Now we report the hyper-parameters for the baseline models and the learned models (for architecture evaluation phase). The main hyper-parameters are learning rate (lr), batch size (bsz) and warm-up steps (warm-up) for finetuning. Warm-up is reported as the proportion of steps in one epoch. One common hyper-parameter is the max sequence length, which is set as 256.

Dataset	model	lr	bsz	warm-up
semeval10	R-BERT	2e-5	64	0.8
	BERT-entity	5e-5	64	1.0
	$AR_{semeval10}$	1e-5	64	0.8
tacred	R-BERT	1e-4	128	0.8
	BERT-entity	5e-5	128	0.8
	AR_{tacred}	5e-5	128	0.8
kbp37	R-BERT	1e-5	64	0.8
	BERT-entity	2e-5	64	0.8
	AR_{kbp37}	5e-5	64	1.0
wiki80	R-BERT	5e-5	128	0.8
	BERT-entity	2e-5	64	1.0
	AR_{wiki80}	2e-5	64	1.0
deft2020	R-BERT	1e-4	64	0.8
	BERT-entity	5e-5	64	1.0
	$AR_{deft2020}$	1e-4	64	0.8
i2b2	R-BERT	2e-5	32	0.8
	BERT-entity	5e-5	32	0.8
	AR_{i2b2}	1e-5	32	0.8
ddi	R-BERT	5e-5	64	0.8
	BERT-entity	2e-5	32	0.8
	AR_{ddi}	5e-5	64	1.0
chemprot	R-BERT	5e-5	64	0.8
	BERT-entity	1e-5	128	0.8
	$AR_{chemprot}$	5e-5	64	1.0

Table 6

[4] https://biocreative.bioinformatics.udel.edu/news/corpora/
[5] http://labda.inf.uc3m.es/ddicorpus