Discovering Better Model Architectures for Medical Query Understanding

Wei Zhu[1,2] [*], **Yuan Ni**[2], **Xiaoling Wang**[1], **Guotong Xie**[2,3,4], **Fang Zhang**[5]
[1] East China Normal University, China
[2] PingAn Health Technology, China
[3] Ping An Health Cloud Company Limited, China
[4] Ping An International Smart City Technology Co., Ltd, China
[5] Shanghai Municipal Center for Disease Control and Prevention, China

Abstract

In developing an online question-answering system for the medical domains, natural language inference (NLI) models play a central role in question matching and intention detection. However, which models are best for our datasets? Manually selecting or tuning a model is time-consuming. Thus we experiment with automatically optimizing the model architectures on the task at hand via neural architecture search (NAS). First, we formulate a novel architecture search space based on the previous NAS literature, supporting cross-sentence attention (cross-attn) modeling. Second, we propose to modify the ENAS method to accelerate and stabilize the search results. We conduct extensive experiments on our two medical NLI tasks. Results show that our system can easily outperform the classical baseline models. We compare different NAS methods and demonstrate that our approach provides the best results.

1 Introduction

Nowadays, online medical question answering (QA) systems are becoming more and more popular. Since the breakout of COVID-19, people grapple with going to the hospital, and hospitals' emergency rooms in some cities are even empty during the daytime.[1] Thus, medical QA systems are of essential importance. NLI models play a central role in such a QA system (Xie et al., 2020). In our QA scenario, we usually use NLI models to determine whether a query has the same intention as some of our labeled questions. For in-domain tasks like ours, modeling experiences are scarce, so selecting a suitable model for our medical NLI tasks becomes a time-consuming procedure. From our experience, different datasets have different optimal models. Thus when a new dataset comes, our engineers usually devote 4 to 5 days experimenting on model tuning and hyper-parameter search with multiple GPU cards.

To speed up the development of our medical QA system and free up the NLP engineers from laborious work, we propose developing a neural architecture search (NAS) framework. Neural architecture search (NAS) has recently attracted intensive attention, both in computer vision and NLP. New RNN models are learned in NASNet (Zoph and Le, 2017), ENAS (Pham et al., 2018), DARTS (Liu et al., 2018), improved DARTS(Jiang et al., 2019) for language modeling. Evolved transformer (So et al., 2019) use the evolution-based NAS algorithm to search for better transformer architectures. TextNAS (Wang et al., 2020) design a new search space for NLU tasks. We will refer to our system as NASQU, shorted for Neural Architecture Search for Query Understanding.

NASQU consists of two parts. First, we design a search space that is suited for NLI tasks. The search space is an extension of the search spaces of TextNAS (Wang et al., 2020). First, for sentence pair modeling, cross-sentence attention plays a central role in aligning the two sentences' contexts and providing a more in-depth understanding of the semantic relations between two sentences (Chen et al., 2016). We add cross-sentence attention operations in the search space, enabling NASQU to search for models with cross-sentence attention mechanisms. Second, aggregating the encoder's outputs to a fixed vector is essential for an NLI model's performance. In this work, we use NAS to decide which layers' outputs are fed into the aggregator and the specific operation in the aggregator layer.

Second, for improving the search results, we employ two modifications to the search method. Our search method mainly follows ENAS (Pham et al., 2018), a reinforcement learning (RL) based search approach. An LSTM controller is employed

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

[1] https://www.cbc.ca/news/health/covid-19-emergency-departments-canada-1.5510778

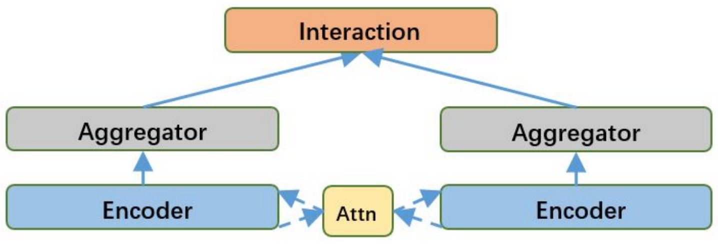


Figure 1: The sentence vector-based framework for the NLI task.

to generate a novel child model, and it will receive a reward based on the child model's performance. Then the controller can update its parameters to improve its ability to generate better child models. A key ingredient for ENAS is weight sharing. We propose further to enhance the weight sharing strategies during architecture search. Besides, we suggest that search warm-ups can stabilize the search processes and provide better search results.

We conduct experiments on two medical NLI datasets designated for intent identification in our medical QA system. The experimental results show that NASQU can learn novel models that perform better than baseline models and meet efficiency requirements. Also, a comparison among the search methods shows that our NASQU obtains better results than other search methods. Besides, we show that the search space design of NASQU is essential.

Our work contributes to the field by the following aspects:

- We extend the search space for neural architecture search in NLP tasks by including cross-sentence attention modules and many design choices.
- We experiment on more in-depth parameter sharing strategies than ENAS, which are proven to provide better search results within less time.

2 NASQU

In this section, we first describe the architecture framework of NLI tasks. Then we extend the search space of TextNAS to support cross-attn modeling and aggregator search. And Finally, we elaborate on the search algorithm in NASQU.

2.1 Architecture framework

We adopt the sentence vector-based framework (Bowman et al., 2015) for NLI tasks. The framework is illustrated in Figure 1. The two sentences (i.e., hypothesis and premise) are encoded and aggregated in siamese network architecture. After obtaining the sentence embedding vector u and v, the final feature vector is $[u; v; |u - v|; u \cdot v]$, where $[]$ is concatenation operation.

Note that our framework is different from TextNAS in two ways. First, we enable attention to flow between the two sentences, which we will elaborate on in the next subsection. Second, we add the search space for aggregators, where TextNAS (Wang et al., 2020) fixes the aggregator to the self-attention aggregator.

2.2 Encoder Search space

To ensure efficiency, we do not stack blocks of the same structure in the encoder. Thus the micro and macro search space are the same. The search space for the encoder is depicted as a fully connected DAG. As is shown in Figure 2, the encoder has *K* (= 5) layers. Node *i* in the DAG is a neural network layer, and edge <*i, j*> means the output of layer *i* is fed into layer *j*. If a layer has multiple inputs, then the inputs will be summed.

For each node, the controller first decides whether the node encodes the sentence itself (self-encoding layer), or it encodes the attention from one sentence to each other (cross-attention layer).

If layer *i* is a cross-sentence layer, it will make its input attend to its counterpart's input in the other sentence's encoder. For example, in Figure 2, layer 2 of the premise encoder is a cross-sentence attention layer. So its input will attend to the input of layer 2 in the hypothesis encoder. In addition to the dot product attention used in the multi-head attention of Transformers (denote as **dot**), we incorporate the four attention functions in Tan et al. (2018), referred to as **p_dot**[2], **concat**, **add**, **minus**.

[2]Note that the dot attention in Tan et al. (2018) (denoted as **p_dot** by us) is not the same as the dot product attention in MHA, where the former is a pointwise product ('A * B' in

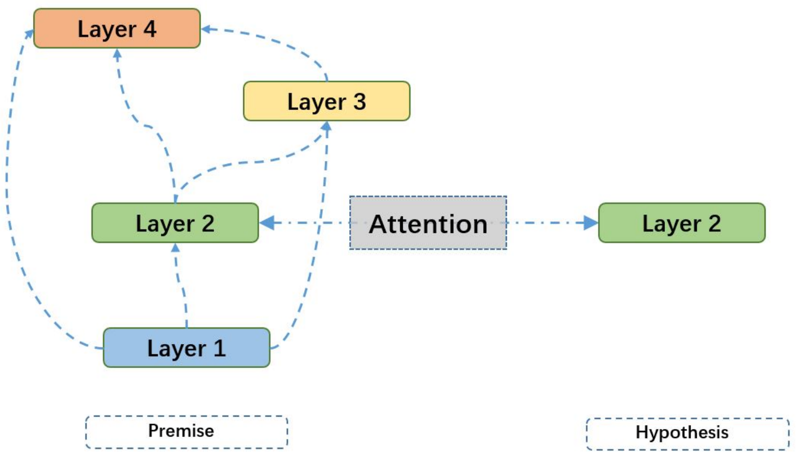


Figure 2: The DAG for the encoder. The layers can be self-encoding layers or cross-attention layers. If layer i is a cross-sentence layer, it will make its input attend to its counterpart's input in the other sentence's encoder.

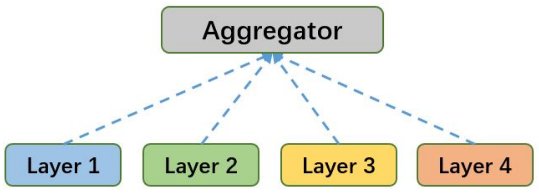


Figure 3: The DAG for the aggregation layer.

For the self-encoding layer, we incorporate four categories of candidate layers which are commonly used for text representation, namely convolutional layers with kernel size 1, 3, 5 (denoted as **conv1**, **conv3**, **conv5**), recurrent layers such as LSTM/GRU (**lstm**, **gru**), max pooling layers with window size 3 and 5 (denoted as **pool_3**, **pool_5**), and multi-head self-attention layers with number of heads 4 and 8 (**mha_4**, **mha_8**). Skip connection (**skip**) is also included to support residual layers. Zero layer (**zero**), which is to output zero tensors, is also included, so that the final model can be sparser than is shown in Figure 2.

2.3 Aggregator search space

As depicted in Figure 3, the DAG for aggregator is much simpler. One has to decide whether each layer's output in the encoder DAG should be fed into the aggregation layer. If multiple layers are selected, their outputs are summed. There are several different aggregation operations. The most common two are the max-pooling aggregator (**max_agg**) and the average-pooling aggregator (**avg_agg**). The self-attention (**sa_agg**) technique is also used for aggregation (Gong et al., 2018; Chen et al., 2018) . We also include dynamic routing (Gong et al., 2018) (henceforth *dr_agg*) into our aggregator operation space.[3]

2.4 Architecture search algorithm

We adopt the ENAS (Pham et al., 2018) framework for search since it is one of the most effective and efficient among all state-of-the-art search algorithms. ENAS searches for the best network architecture via reinforcement learning with weight sharing. ENAS leverages an LSTM as the controller. In each step, the controller samples several child networks from the search space. The child networks share the same set of parameters with the global super-graph to accelerate the evaluation procedure. After child models' performances are obtained, they are fed back to the controller as reward signals. The parameters of the controller are updated through policy gradients based on REINFORCE (Williams, 1992). We implement ENAS via NNI[4].

In this work, we try to improve the search results by deeper parameter sharing and search warm-up. First, the parameter sharing in ENAS is relatively shallow. The parameters are shared only when precisely the same operation is used in the same position of the DAG. We share the parameters across related operations. For convolutional layers, we use depthwise separable convolution networks since they are more parameter efficient. And the point-

PyTorch), and the latter is (batch-wise) matrix multiplication ('torch.matmul(A, B)').

[3]Following Gong et al. (2018), we set the number of capsules as 4 and the number of iterations as 3.

[4]https://github.com/microsoft/nni

wise convolution inside each convolutional layer is shared across **conv1**, **conv3**, and **conv5**. The key, query and value matrices inside **mha_4** and **mha_8** are shared for multi-head attention layers. The key, query, and value matrices for cross-attn modules of different attention functions are shared.

Second, we add a search warm-up phase before the search begins, and the learning rate of the RL controller is also gradually increased to the maximum value and follows a linear decay schedule. The intuition is that when the shared parameters are trained for pre-specified steps, the controller can receive much more reliable reward signals.

To make the model efficient enough for online deployment, we add efficiency constraints to the generated child models, which will be specified in the next section. Child models that fail these requirements will be assigned a zero reward so that the controller will learn how to generate models that meet the requirements.

3 Experiments and Discussion

3.1 Datasets

We conduct experiments on two medical NLI datasets we build for developing our medical dialogue system, whose statistics and metrics for evaluation are shown in Table 1.

Chinese Medical Frequent Asked Queries (CMFAQ). These datasets contain pairs of Chinese medical frequently asked questions (FAQs), and the model has to determine whether the two queries contain the same meaning. The dataset is collected from the logs of an online medical consultation provider. The sentences in this dataset are usually general health-related questions.[5]

Chinese Medical Query Patterns (CMQP). This dataset is designated for semantic matching of the query patterns. The model has to determine whether two patterns have the same intention. We collect queries that are related to medical entities and annotate their NER tags. Then we replace the named entities with special tokens that reflect the entity types in the sentence and transform the query into a query pattern.[6]

The train/valid/test split is defined by randomly splitting the whole annotated dataset with a ratio 7:1:2.

3.2 Experimental settings

To improve the performances of models that are not pre-trained on a large corpus, we will distill knowledge from a pre-trained large teacher model both during search and model evaluation. The knowledge distillation method follows Liu et al. (2019), and the distillation temperature is set to be 10. We select the Chinese BERT-wwm-ext (Cui et al., 2019) as the teacher model. To make it more suitable for our domain applications (Gu et al., 2020), we further pre-train it on our 2.3GB medical corpus.

In our experiments, the encoder has five layers at most. The 128d word embedding vectors are initialized by a pre-trained Word2Vec (Mikolov et al., 2013) model trained on our medical corpus and are fine-tuned during training. The hidden dimension is kept to 256 in the model. During the architecture search, the batch size is 128, max input length is 64 for both premise and hypothesis, the dropout ratio is 0.2, and the weight decay is 2e-6. For both model weights and controller, we utilize Adam optimizer and learning rate decay with cosine annealing. The maximum learning rate l_{max} is 3e-3, and the minimum learning rate l_{min} is 1e-6, and the cosine cycle is 10. For model weights, at the beginning of training, the learning rate is linearly warmed up for 0.8 of one epoch to l_{max}. For search warm-up, in the first 3 epochs, the controller is not updated, and at the beginning of the fourth epoch, the controller learning rate is also linearly warmed up for 0.8 of one epoch to l_{max}.

After each epoch, ten candidate architectures are generated by the controller and evaluated on the validation set. The inference batch size is 1, which mimic the scenario for online deployment. When obtaining the validation performance, we also calculate the inference speed and memory consumption. The efficiency requirement is that the model's GPU memory consumption is less than 1 GB, and the per-sample inference time is lower than 20ms. If the efficiency requirements are not satisfied, the child model's reward is set to be zero. After train-

[5]For examples, "现在在居家隔离，我要怎么保持健康？" (Now in isolation at home, how can I keep healthy?) is a quite popular query during the breakout of COVID-19, and it is matched to one of our collected FAQs, "居家隔离如何养生？" (How to keep healthy when quarantined at home?)

[6]For example, a common question we received is "立普妥是饭前吃吗？" (Is Lipitor taken before meals?). In this sentence, "立普妥" (Lipitor) is a drug entity, so the query is transformed into a query pattern "<drug>是饭前吃吗?" (<drug> should be taken before meals?), and it is matched to one of our collected query patterns "<drug>应该饭前吃还是饭后吃？" (<drug> should be taken before meals or after meals?).

Dataset	avg seq_len	Train #	Dev #	Test #	Label #	Metrics
CMFAQ	32.5	37676	5382	10764	2	F1
CMQP	19.6	32476	4639	9279	2	F1

Table 1: Overview of medical NLI datasets in experiments.

ing 150 epochs, the architecture with the highest evaluation F1 is chosen as the final network. And this model is retrained from scratch with optimal hyper-parameters tuned using NNI's implementation of Bayesian optimization. We mainly focus on three hyper-parameters: (1) batch size, (2) learning rate, (3) dropout rate.

We assign 2 CPU cores, 8G memory, and 1 Tesla V100 GPU card for each search or evaluation in our experiments. The search lasts 12 hours and 4 hours for CMFAQ and CMQP, respectively.

3.3 Baseline methods

The best learned architectures are evaluated by training from scratch (with hyper-parameter search). The baseline models include: (a) LSTM + *max_agg*; (b) LSTM + *sa_agg*, which are evaluated by Wang et al. (2018); (c) LSTM/Transformer + *dr_agg* (Gong et al., 2018); (d) ESIM (Chen et al., 2016); (e) Decomposable attention (DecompAttn) model (Parikh et al., 2016). The number of encoder layers is treated as a hyper-parameter and are tuned together with other parameters. We also compare our search space with that in TextNAS. We also compare different search algorithms that have similar time complexities as ENAS, including DARTS (Liu et al., 2018), One-Shot (Luo et al., 2019), and Random Search with Weight Sharing (RandomSA) (Li and Talwalkar, 2019). [7] The BERT-wwm-ext teacher model's performances are also reported.

3.4 Search Results

As depicted in Figure 4(a) and 4(b), the learned architectures consist of different layers categories. For convenience, we will refer to the best model learned on CMFAQ as NASQU-1 and the best model learned on CMQP as NASQU-2. NASQU-1's encoder has 3 self-encoding encoders, 2 of which are two convolutional layers, and the other one is a MHA layer, and it has a cross-attn layer with the **add** attention function. Note that NASQU-1 discards the 4th layer in the DAG since it is **zero** operation. The aggregator of NASQU-1 takes layers 2 and 4 as input, and the aggregator is **sa_agg**. Meanwhile, NASQU-2 is more lightweight than NASQU-1 since it discards the 3rd and 5th layer. NASQU-2's encoder has a GRU layer, a conv layer, and a cross-attn layer with **p_dot** attention function. The aggregator of NASQU-2 takes all valid layers as input, and the aggregator is **dr_agg**.

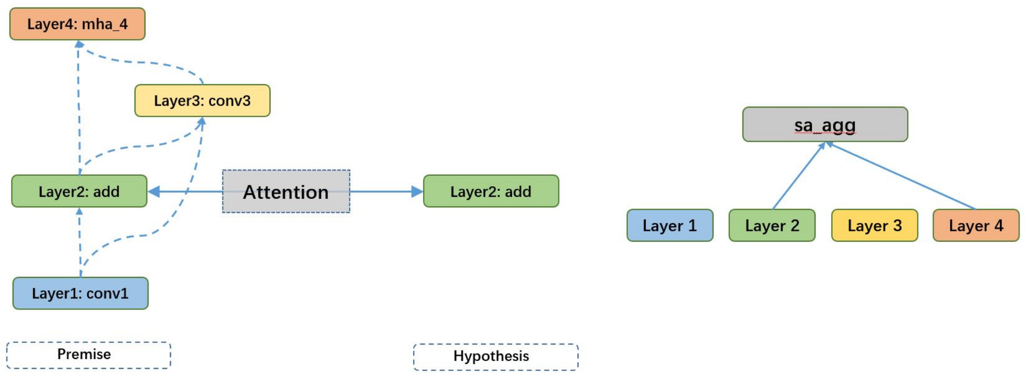
MODEL	CMFAQ	CMQP
Baseline models		
GRU + *max_agg*	81.8	83.6
LSTM + *max_agg*	82.3	84.6
LSTM + *sa_agg*	83.6	85.4
LSTM + *dr_agg*	85.3	**87.7**
Transformer + *dr_agg*	84.3	86.8
ESIM	**85.8**	87.6
DecompAttn	84.5	86.9
NAS models		
DARTS	**86.4**	87.9
One-Shot	86.3	88.1
RandomSA	85.5	87.2
TextNAS	86.3	**88.6**
Our models		
NASQU-1	**87.1***	88.5
NASQU-2	86.2	**89.5***
BERT-wwm-ext	88.9	90.5

Table 2: Results of the two medical NLI dataset. For each dataset, we conduct a significance test against the best reproducible model, and * means that the improvement is significant at the level of 0.05 significance level.

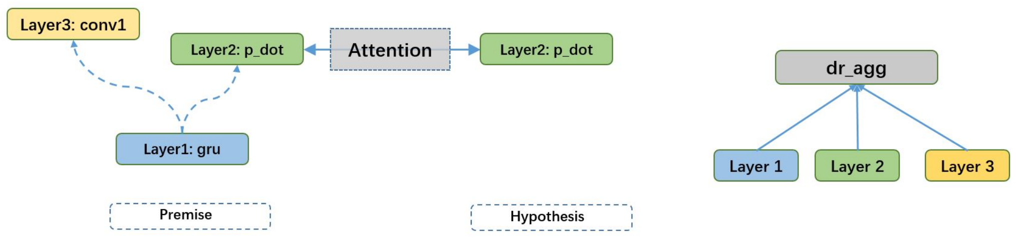
Although the learned model seems to be more involved than manual architectures, we still find that there are some design principles in line with the commonsense and previously established observations:

- Convolution layers are combined with GRU and multi-head self-attention layers, which are similar to C-LSTM (Zhou et al., 2015) and Transformer (Vaswani et al., 2017). Intuitively, convolution operations extract local features similar to n-gram, which complement long-term dependency features captured by GRU/self-attention.

[7] Unless specified, the default settings of their open-source codes are used.



(a) NASQU_1



(b) NASQU_2

Figure 4: Learned architectures on the two medical NLI datasets.

- We find that the best learned architectures include cross-attn modules, which are in line with observations in the previous literature (Parikh et al., 2016; Chen et al., 2016). Cross sentence attention can align the semantics of two text inputs and provide better feature extraction for NLI tasks.

- Different tasks result in quite different architectures, emphasizing the importance of task specificity. Besides, a smaller dataset size prefers a more light-weighted architecture, since intuitively, a heavier architecture is more prone to over-fitting on a small dataset.

The performance results are shown in Table 2. The learned architecture discovered by NASQU achieves higher average F1 scores than all the baseline models. Also, it outperforms other network architectures found automatically by other search spaces and algorithms. Note that on CMQP, the improvement over different baselines is statistically significant. We also evaluate the transferring ability of the two best learned architectures. Although NASQU-1 also performs well on CMQP compared with baselines, it is significantly worse than NASQU-2. This observation also stands for NASQU-2 on the CMFAQ dataset. These observations validate the importance of customizing different architecture for different tasks.

Model	GPU memory	inference speed
BERT-wwm-ext	4.8 GB	66ms/it
NASQU-1	0.84 GB	14ms/it
NASQU-2	0.71 GB	11ms/it
LSTM + *dr_agg*	0.72GB	15ms/it
ESIM	1.12GB	21ms/it

Table 3: Comparison of GPU memory consumption and inference speed between the teacher model BERT Large and NASQU-1, and NASQU-2.

Table 2 also shows that the performances of the two learned models is close to the BERT teacher. We now compare the inference speed and GPU memory consumptions of BERT and the learned models. The results are reported in Table 3. We can see that the learned models achieve significant speed-up over the BERT teacher models without too much performance loss, and they are more efficient than ESIM. LSTM + **dr_agg** has comparable efficiency, but the performance is significantly worse.

3.5 Ablation studies

We now conduct extensive ablation studies to demonstrate our search space design, and modi-

strategies	test F1
NASQU	**87.1***
- deeper weight sharing	85.9
- search warm-up	84.5

Table 4: Results for ablations studies on the strategies we propose for search.

search space	test F1
NASQU	**87.1***
- cross sentence attention	86.2
- aggregator search space	84.6

Table 5: Results for ablations studies on the search space we construct for architecture search for NLI tasks.

fications to the search algorithm are essential. The ablation studies are performed on CMFAQ.

First, we conduct ablation on the proposed modifications to the search method. As we can see from Table 4, deeper weight sharing is beneficial. Intuitively, deeper parameter sharing reduces the number of shared parameters during the search phase, making the reward signal more reliable. The results also show that the search warm-up also contributes to better search results.

We now conduct an ablation study on our entire search space. The results are shown in Table 5. Note that for our NLI tasks, when we drop the cross attention mechanism from the search space, the search results drop from 87.1 to 86.2. And we can see that dropping the aggregator search space (fixing the aggregator to be **sa_agg**) also results in worse performances.

4 Conclusion and Future Work

In this work, we experiment on modeling NLI tasks via NAS on our medical NLI tasks. Our search space is an extension to TextNAS and ENAS, which enable us to search for novel models with cross sentence attention and suitable aggregators. To improve the search results, we also propose a more in-depth weight sharing strategy than ENAS and search warm-up steps. Experiments on our NLI tasks demonstrate that our search space is beneficial for NLI tasks.

References

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 632–642, Lisbon, Portugal. Association for Computational Linguistics.

Qian Chen, Zhen-Hua Ling, and Xiaodan Zhu. 2018. Enhancing sentence embedding with generalized pooling. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 1815–1826, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

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*.

Yiming Cui, W. Che, T. Liu, B. Qin, Ziqing Yang, S. Wang, and G. Hu. 2019. Pre-training with whole word masking for chinese bert. *ArXiv*, abs/1906.08101.

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

Yu Gu, Robert Tinn, Hao Cheng, Michael Lucas, Naoto Usuyama, Xiaodong Liu, Tristan Naumann, Jianfeng Gao, and Hoifung Poon. 2020. Domain-specific language model pretraining for biomedical natural language processing.

Yufan Jiang, Chi Hu, Tong Xiao, Chunliang Zhang, and Jingbo Zhu. 2019. Improved differentiable architecture search for language modeling and named entity recognition. 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 3585–3590, Hong Kong, China. Association for Computational Linguistics.

Liam Li and Ameet Talwalkar. 2019. Random search and reproducibility for neural architecture search. *ArXiv*, abs/1902.07638.

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

Xiaodong Liu, Pengcheng He, W. Chen, and Jianfeng Gao. 2019. Multi-task deep neural networks for natural language understanding. In *ACL*.

Renqian Luo, Tao Qin, and E. Chen. 2019. Understanding and improving one-shot neural architecture optimization. *ArXiv*, abs/1909.10815.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS*.

Ankur P Parikh, Oscar Täckström, Dipanjan Das, and Jakob Uszkoreit. 2016. A decomposable attention model for natural language inference. *arXiv preprint arXiv:1606.01933*.

H. Pham, M.Y. Guan, B. Zoph, Q.V. Le, and J. Dean. 2018. Efficient neural architecture search via parameter sharing. *arXiv preprint arXiv:1802.03268*.

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.

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

Chuanqi Tan, Furu Wei, Wenhui Wang, Weifeng Lv, and Ming Zhou. 2018. Multiway attention networks for modeling sentence pairs. In *IJCAI*, pages 4411–4417.

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 *NIPS*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*.

Yujing Wang, Yaming Yang, Yiren Chen, Jing Bai, Ce Zhang, Guinan Su, Xiaoyu Kou, Yunhai Tong, Mao Yang, and Lidong Zhou. 2020. Textnas: A neural architecture search space tailored for text representation. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(05):9242–9249.

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

Ruobing Xie, Yanan Lu, F. Lin, and Leyu Lin. 2020. Faq-based question answering via knowledge anchors. In *NLPCC*.

Chunting Zhou, Chonglin Sun, Zhiyuan Liu, and F. C. M. Lau. 2015. A c-lstm neural network for text classification. *ArXiv*, abs/1511.08630.

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