AutoTrans: Automating Transformer Design via Reinforced Architecture Search

Wei Zhu[1] **Xiaoling Wang**[1] **Xipeng Qiu**[2] **Yuan Ni**[3] **Guotong Xie**[3]
[1] Shanghai Key Laboratory of Trustworthy Computing, East China Normal University, Shanghai, China
[2] School of Computer Science, Fudan University
[3] Pingan Health Tech, Shanghai, China
52205901018@stu.ecnu.edu.cn, xlwang@cs.ecnu.edu.cn,
xpqiu@fudan.edu.cn, {niyuan442, xieguotong}@pingan.com.cn

Abstract

Though the transformer architectures have shown dominance in many natural language understanding tasks, there are still unsolved issues for the training of transformer models, especially the need for a principled way of warm-up which has shown importance for stable training of a transformer, as well as whether the task at hand prefer to scale the attention product or not. In this paper, we empirically explore automating the design choices in the transformer model, i.e., how to set layer-norm, whether to scale, number of layers, number of heads, activation function, etc, so that one can obtain a transformer architecture that better suits the tasks at hand. RL is employed to navigate along search space, and special parameter sharing strategies are designed to accelerate the search. It is shown that sampling a proportion of training data per epoch during search help to improve the search quality. Experiments on the CoNLL03, Multi-30k, IWSLT14 and WMT-14 shows that the searched transformer model can outperform the standard transformers. In particular, we show that our learned model can be trained more robustly with large learning rates without warm-up.[1]

1 Introduction

The transformer architecture (Vaswani et al., 2017) has achieved great success in not only machine translation but also many other natural language processing (NLP) tasks. Its popularity obtains further increase with the introduction of BERT (Devlin et al., 2018). Meanwhile, there are also many criticisms to the transformer. First, transformers often require a warm-up step to stabilize model training, which is directly affected by how the layer normalization is applied in the transformer architecture. In Vaswani et al. (2017), layer-norm is applied after residual connection (post-LN transformer). Zhang et al. (2019) and Wang et al. (2019) place layer-norm before multi-head attention and activation (prev-LN transformer). Recently Xiong et al. (2020) gives a theoretical explanation of the advantages of the prev-LN transformer. However, these two strategies are the only existing combinations of layer-norms. Second, Yan et al. (2019) points out that some tasks like named entity recognition (NER) may prefer not to scale the attention product. In addition, there are many design choices (or hyper-parameters) in the transformer architecture, e.g., the number of attention heads, layers (Song et al., 2019), and the number of dimensions in the positional feed-forward module, etc. There exists little guidance on how to achieve an optimized transformer architecture for different tasks. Manual tuning or simple heuristic search is time consuming and computationally expensive without any grounded guarantee. Due to its wide applications, optimizing the transformer architectures for specific tasks can be of great importance.

Network Architecture Search (NAS) has achieved great success in image recognition. NAS has helped to discover better models for a variety of vision tasks, from image classification (Cai et al., 2018), semantic segmentation (Liu et al., 2019), to object detection (Ghiasi et al., 2019), etc. Despite its success in vision, there are not enough work to study NAS for NLP. Some efforts have also been invested in searching for sequence models (Zoph and Le, 2017; Pham et al., 2018). In these cases, it has always been to find better RNN architectures. There are few studies in applying NAS into improving the standard transformer architectures. So et al. (2019) employs evolution algorithms for search, but their focuses are to combine convolutions and multi-head attention operations. In addition, the search process they conduct requires enormous computations that is forbidding to most

[1]The source code will be made public available.

researchers or NLP practitioners. In this work, we propose a more efficient yet effective methodology to improve upon the standard transformer architectures.

When trying to improve upon the standard transformers, the above literature fall short in the following aspects.

1) For the settings of layer-norms, no previous literature have proven theoretically or empirically the existing two solutions are optimal.

2) Many other import design choices of transformers architectures like whether to scale, whether to put attention to encoder module before or after self attention, number of layers is not addressed by NAS literature.

3) Existing work on NAS for transformers is extremely time-consuming, making it in-practical.

In this work, we experiment on making the design choices in the transformer model automatically, i.e., how to set layer-norm, whether to scale, where to put encoder attention, number of heads, number of layers, activation function, etc, so that one can obtain a transformer architecture that better suits the tasks at hand. To navigate on our search space, we employ reinforcement learning (RL) strategy, or more specifically, ENAS by (Pham et al., 2018), and we design the specialized parameter sharing strategies for multi-head attention to help speeding up the search process. And we propose to sample a proportion of the training data at each search epoch, as a way of regularization and speed-up. Experiments on the CoNLL03, Multi-30k, IWSLT-14, WMT-14 dataset shows that the searched transformer models can outperform the standard transformer models significantly. And we will show that this performance advantage is persistent across different learning rates. Note that since they are two phases in ENAS search, the top-ranked model at the search phase may not be the best one, but it can still reliably outperform the standard transformers.

The contributions of the paper can be summarized as:

- We develop a comprehensive search space to improve transformer architecture, especially for the positions of layer-norms.
- We develop efficient search for new transformer architectures (e.g. 5 GPU hours on IWSLT-14), by employing RL strategy and designing specialized parameter sharing strategies for multi-head attention.
- The learned models outperforms the standard transformers, and this performance gain is robust under different learning rates.

2 Related Work

The field of neural architecture search (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 or hyper-parameters 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 understanding (NLU). 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, and the vanilla transformer as the initial population, it generates a better transformer architecture that consistently outperform the vanilla transformer on 4 benchmark machine translation tasks.

Our work also focuses on the transformer architecture, but the difference with the evolved transformer is clear. First, evolved transformer emphasize on combining convolution operation and multi-head attentions, while our work is to optimize the settings of layer-norms, whether to scale, where to place the encoder attentions in the decoder, number of layers, etc, such that the model can converge and generalize well without warm-up. Second, we employ a special designed parameter sharing strategies, and we propose to sample a proportion of training data per search step, such that the search time cost can significantly reduce.

Our work is also closely related to a line of work that try to modify and improve the transformer architecture. Sparse transformer (Correia et al., 2019) uses a sparse alternative of softmax to reduce the head size of self attention. Star Transformer (Guo et al., 2019) uses an intermediate center node to change the fully-connected attention to a more sparse one, thus making the model lighter and can perform better on a series of small or medium sized datasets.The recent Reformer (Kitaev et al., 2020) uses local sensitive hashing and reversible

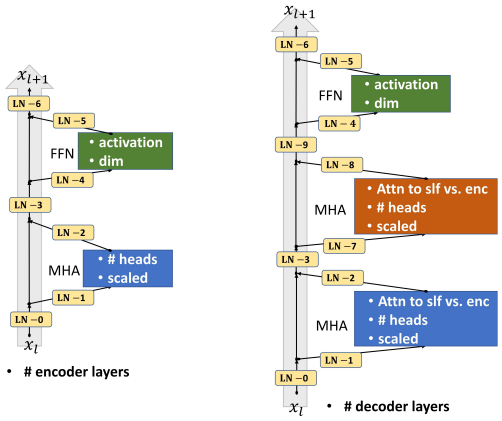


Figure 1: The architecture for AutoTrans.

transformer to significantly reduce the time and space complexity of transformer architectures. Our work contributes to the literature by including many design choices that are ignored by literature into our search space, and making the search for better transformers automatic and efficient.

3 Search Space Design

Now we discuss our search space in detail. Since our goal here is to optimize the transformer architecture, we keep its main bone structure, as shown in Figure 1.

The most import aspect in the search space is how the layer-norm is employed in the transformer block. The original transformer paper (Vaswani et al., 2017) put two layer-norm after the two residue connections, meanwhile in its codebase[2], it suggests that putting the layer-norm before the multi-head attention can make the training more stable. This phenomenon is discussed and explained theoretically by Xiong et al. (2020). However, the simple settings may be sub-optimal, as the search space for layer-norms is large. Figure 1 depicts the possible positions for layer-norms, where $LN - i$ ($i < 7$ for the encoder, and $i < 10$ for the decoder) in yellow boxes means a layer-norm can be put at position i.[3]

The second aspect is whether to scale at the multi-head attention. (Vaswani et al., 2017) argues that scaling makes the gradient more stable. Meanwhile, recent study by (Yan et al., 2019) shows that not to scale results in sparser attention and thus

[2]https://github.com/tensorflow/tensor2tensor

[3]If layer-norm is put at position 0, then there is no need for layer-norm at position 1. This also holds for position 3 and 4, 9 and 4.

Name	Function
relu	$\max(x, 0)$
leaky_relu	x if $x \geq 0$ else 1e-2 * x
elu	x if $x \geq 0$ else $e^x - 1$
swish	$x * sigmoid(x)$
gelu	$0.5 * x * (1 + erf(x/\sqrt{2}))$
gelu_new	$0.5x(1 + tanh(\sqrt{2/\pi}(x + 0.044715x^3)))$

Table 1: Activation functions in the search space.

help to improve the transformer's performance on NER tasks. Third, the number of layers is also import as it can not only affect performance during training, but also how many GPU resources are needed for deploying the model. In addition, it is common to set the number of layers in encoder to be equal to that in the decoder (Vaswani et al., 2017), however in Song et al. (2019) the decoder has fewer layers than the encoder. In this paper, we let the search procedure to decide whether to have different numbers of layers in the encoder and decoder. The fourth design choice we include in our search space is whether the attention to encoder module is placed after the self-attention or before.

Similar to So et al. (2019), the search space also include the number of heads, and the activation function used in the positional feed-forward layer, and the relative dimension for the intermediate hidden states. The activation function we consider is listed in Table 1.

Now we formally introduce our search space. For the encoder, the search space is as follows.

- Layer-norm at position i ($i < 7$) = True, False;
- Embedding layer-norm = True, False;
- Final output layer-norm = True, False;
- Number of self-attention heads = 1, 2, 4, 8, 16;
- Attention scaled = True, False;
- Activation = relu, leaky_relu, elu, swish, gelu, gelu_new;
- Relative dimension = 0.5, 1, 2, 4, 8;[4]
- Number of layers = 1, 2, 3, 4, 6, 9;

For the decoder, most of the search space is the same, except the following three items.

[4]Here the relative dimension being equal to 0.5 means halving the hidden dimensions.

- Layer-norm at position i ($i < 10$) = True, False;
- Encoder attention after self-attention = True, False;

In total, our search space contains 1.5e+6 number combinations of possible transformer encoder architectures and 1.2e+8 for encoder-decoder architectures, which is quite a large search space. Next, we will show how to navigate through this enormous search space and obtain architectures that are better than standard transformers efficiently.

4 Architecture Search

We elaborate on the search algorithm that exploits the defined search space, the parameter sharing strategies and data sampling strategy controlling the search time within a GPU day.

4.1 Search Algorithm

We employ a controller to do a guided exploitation in search space, which is similar to NASNet (Zoph et al., 2018) and ENAS (Pham et al., 2018). The controller is an LSTM network with 100 units, with parameters θ. The output hidden state is fed into a classifier to decide the action at each step. The shared parameters of the child models are denoted by ω, which will be discussed in detail in the next subsection.

The architecture search procedure consists of two interleaving phases. The first phase trains ω, the shared parameters of the child models, on a pass through the training data set. In ENAS (Pham et al., 2018), this step use the whole pass of the training data set. However, we argue that for each pass, randomly sampling a proportion of the training data not only saves time, but also provides extra regularization so that the parameters do not over-fit and the search can obtain better architectures. The second phase trains θ, the parameters of the controller, via optimizing the expected reward function using the REINFORCE algorithm (Williams, 1992). The reward function is the negative of the perplexity on the valid set for the machine translation task, and F1 score for NER.

4.2 Deriving Architectures

We discuss how to derive novel architectures from a trained ENAS model. We rst sample several models from the trained policy $\pi(m, \theta)$. For each sampled model, we compute its reward on the validation set. Then we take the top model(s) with the highest rewards to re-train from scratch. The number of top models to select is based on our computational resources, which will be shown in detail in the next section. We find that it is not guaranteed that the top-ranked model generated here will turn out to be the best model in certain tasks. But, selecting only a few top models to train from scratch makes the search procedure reasonably economical, and is shown to be able to improve the transformer architecture.

Table 2: Overview of used datasets in experiments.

Dataset	Task	Train	Dev	Test	Metrics
Multi-30k (en-de)	MT	30k	1.03k	1.0k	BLEU
IWSLT (de-en)	MT	160k	7.2k	6.7k	BLEU
WMT-14 (en-de)	MT	4.5M	39k	3k	BLEU
CoNLL2003	NER	14k	3.2K	3.4k	F1

4.3 Cross-operation Parameter Sharing

In this work, the embeddings for the source language and the target language are shared for all the child models. For more efficient parameter sharing and ease of training, we constrain that the hidden dimension of each attention head is equal to the hidden size divided by the number of heads, so that multi-head attention block with different number of heads have parameters of the same size, and thus are possible to share the query, key and value matrices. Note that the encoder and decode does not share weights.

4.4 Cross-layer Parameter Sharing

Recently ALBERT (Lan et al., 2019) shows that cross-layer parameter sharing can provide regularization for training and stabilize the gradients, thus are beneficial for training deep models. In this paper, unless otherwise stated, during search the parameters in the previous transformer layer is shared to the next one. Cross-layer parameter sharing is beneficial in the scenario of NAS since it notably reduce the number of shared weights, thus greatly accelerating the search process.

5 Experiments and Results

For each search or evaluation, we assign 1 Tesla V100 GPU card(s) for CoNLL03 and Multi-30k, and 4 for IWSLT14.

5.1 Datasets

We conduct experiments on two different tasks with 4 benchmark datasets, whose statistics are

shown in Table 2. To verify the validity of our method on tasks of different scales, we select three machine translation tasks, Multi-30K (Elliott et al., 2016), IWSLT (German to English) (Cettolo et al.) and the standard WMT-14 (English-German) dataset, which represent machine translation tasks of different sizes. We also experiment on CoNLL2003 (Sang and De Meulder, 2003), a benchmark NER datasets, to showcase that our method works under different NLP tasks.

Multi-30K This task involves translating English sentences that describe an image into German.

IWSLT14 This dataset focuses on the translation of TED Talks, a collection of public speeches covering many different topics.

WMT-14 It consists of about 4.5 million EN-DE sentence pairs. Sentences were encoded using byte-pair encoding (Sennrich et al., 2016), which has a shared source target vocabulary of about 37000 tokens. We use *newstest2013* for validation and *newstest2014* for test, which is in consistence with Vaswani et al. (2017).

CoNLL2003 This dataset consists of 200k training words which have been annotated as Person, Organization, Location, Miscellaneous, or Other (non-named entity).

5.2 Architecture Search Protocols

During search phase, the interleaving optimization process is run 100 times. For each search epoch, a proportion r of the train data is passed to a child model, where $r = 0.05, 0.2, 0.5$ *or* 1. For WMT-14 task, we only consider $r = 0.2$. For the CoNLL03 task, we use the pre-trained embedding from Glove (Pennington et al., 2014) (840B, 300d), and the 300-d embedded input is reshaped to 512d with a separable convolution with kernel size 1. For the three MT tasks, the embedding is randomly initialized, and the dimensions for the embedding and for the hidden states are all set to 512. Due to the resource limitation, for the WMT-14 dataset, we limit the number of layers to be less or equal to 6, so that the size and number of parameters of the new transformers will not be larger than the transformer base setting in Vaswani et al. (2017), thus for comparison of performances, we will only compare with transformer base. The hidden dim for the controller is set to 100. After manually fine-tuning, the learning rate for the search is set at 1e-4 for CoNLL03 and IWSLT14, and 1e-3 for Multi-30k and WMT-14. The batch-size is set at 64 per GPU.

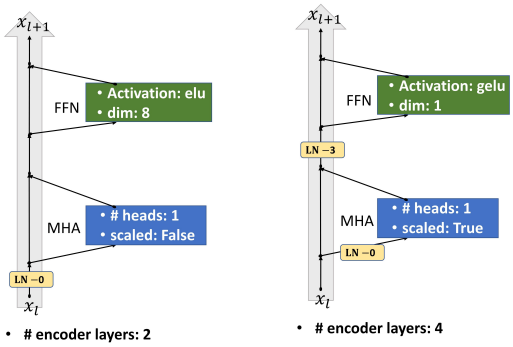
After the search phase, 30 model architectures are sampled from the trained controller, and they are ranked via their performance on the valid data when they are initialized using the shared parameters. Then the top-ranked 5 models (2 models for WMT-14 task) are trained from scratch to convergence on the whole training data of the task to formally evaluate their performances. The training is also repeated for n runs to calculate the fluctuation of performances. n is set in consideration of replication and our resource limitations. For CoNLL03, Multi-30k and IWSLT14, n is set to be 10. For WMT-14, n is set to be 5.

To compare our methods with random search, for task CoNLL03, Multi-30k and IWSLT14, we randomly samples 7 different models, since the GPU time for training 7 models is slightly larger than an entire search and evaluation process. Due to the same reason, we also randomly samples 3 different models for evaluation on the WMT-14 task. We train them from scratch, and report the performance of the best dag as the performance of this random search run. The results of random search will be the average of 5 such runs.

The key hyper-parameters for all the learned models and baseline transformers are learning rate, and for standard transformers the number of layers, number of heads, relative dimension are also considered. Learning rate is selected from 3e-3, 1e-3, 1e-4, and the other hyper-parameters are consistent with that in our search space. The optimal hyper-parameters are determined via exhaustive search over the search space.

5.3 Main Results

Results on CoNLL-03. First, we report the results on the CoNLL 2003 task in Table 3. The prev-LN transformers perform slightly better than the post-LN version, achieving around 67.89 F1 on the test set. Random search on our search space obtains worse average results, and the performances are quite volatile. Now we look at our learned models. The top-ranked dag $\text{AT}_{conll,0.2,0}$ by the search procedure when using only 20% of the training data is depicted in Figure 2(a). It only uses layer-norm once at the beginning of the transformer block, and it uses a relative dim of 8 and stack two transformers layers. Note that it does not scale at the MHA. This model significantly outperforms the two ver-



(a) $\text{AT}_{conll,0.2,0}$ (b) $\text{AT}_{conll,0.5,1}$

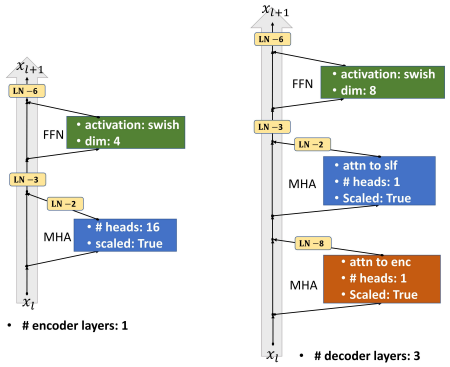
Figure 2: New transformer architectures learned on CoNLL03.

Model	dev F1 (%)	test F1 (%)
prev-LN Transformer	75.77 ± 0.354	67.89 ± 0.204
post-LN Transformer	75.46 ± 0.217	67.11 ± 0.262
random search	74.89 ± 0.768	66.7 ± 0.645
$\text{AT}_{conll,0.2,0}$	78.15 ± 0.178	68.81 ± 0.189
$\text{AT}_{conll,0.5,1}$	**78.64** ± 0.204	**69.75** ± 0.197

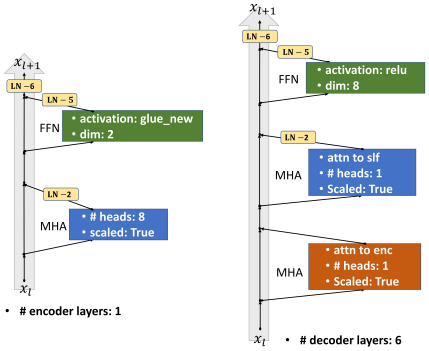
Table 3: Results on the CoNLL2003 dataset.

sions of standard transformers by achieving 68.81 F1 score on the test set. The best model the search gives out is $\text{AT}_{conll,0.5,1}$, a 4-layer transformer discovered when using 50% of the training data for search. If the resources allows, training more top-rated dags can help to discover better architectures. The results show that our search method is able to discover new transformer architectures efficiently.

Results on Multi-30k. The results on the Multi-30k English-German translation task is reported on Table 4. Random search on our search space can not result in good performances and is quite unreliable in finding good architectures. The top-ranked architecture at the search phase when using 20% of the training data for search achieves average ppl of 17.63 on dev set and BLEU score of 33.55, which outperforms the two standard transformers. However, this is not the best architecture we obtained. When using all the train data per search epoch, the second best dag by the search phase is the best architecture we find. Figure 3(a) and 3(b) depict the two learned models. Note that in the two models: i) a layer-norm is placed right after the multi-head self attention, and after the FFN module; ii) the decoder has more layers than the encoder layer; iii) the attn to encoder module is placed before the self attn in the decoder layer. The two learned models



(a) $\text{AT}_{multi30k,0.2,0}$



(b) $\text{AT}_{multi30k,1,1}$

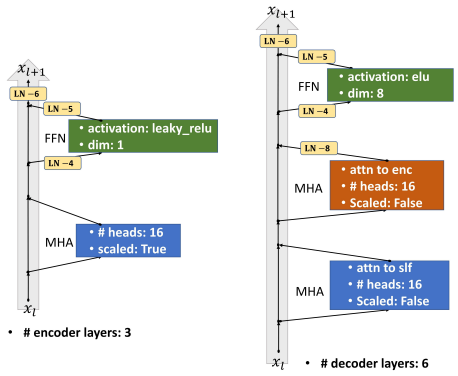
Figure 3: New transformer architectures learned on Multi-30K.

Model	dev ppl	test BLEU
prev-LN Transformer	22.33 ± 0.123	29.47 ± 0.426
post-LN Transformer	19.64 ± 0.569	31.20 ± 0.089
random search	24.76 ± 1.987	28.08 ± 1.863
$\text{AT}_{multi30k,0.2,0}$	17.54 ± 0.090	33.55 ± 0.310
$\text{AT}_{multi30k,1,1}$	**16.63** ± 0.143	**34.56** ± 0.325

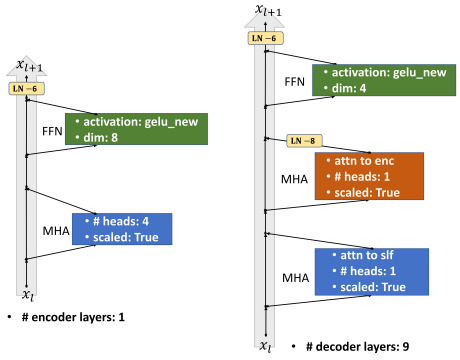
Table 4: Results on the Multi-30k dataset.

verify the necessity of our search space design.

Results on IWSLT-14 As is shown in Table 5, different from the learned models on Multi-30K, the ones we discovered on IWSLT-14 tends to favor placing the attention to encoder behind the self attention operation, as depicted in Figure 4(a) and 4(b). But again, we see that the decoder tends to have more layers than the encoder, which is similar to $\text{AT}_{multi30k,0.2,0}$ and $\text{AT}_{multi30k,1,1}$. As for the positions of layer-norms, both $\text{AT}_{iwslt14,0.2,0}$ and $\text{AT}_{iwslt14,0.2,3}$ place layer-norms after the attention to encoder, and place one at the end of an layer. Both models outperform the standard transformers significantly. Notably, the best models

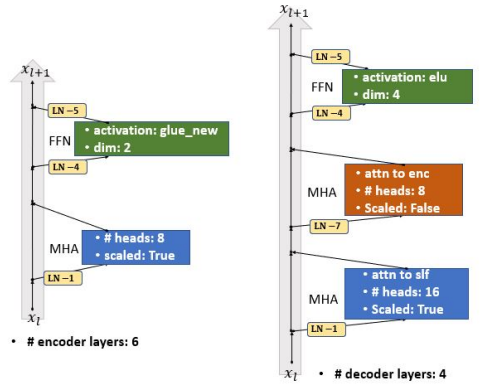


(a) $\text{AT}_{iwslt14,0.2,0}$

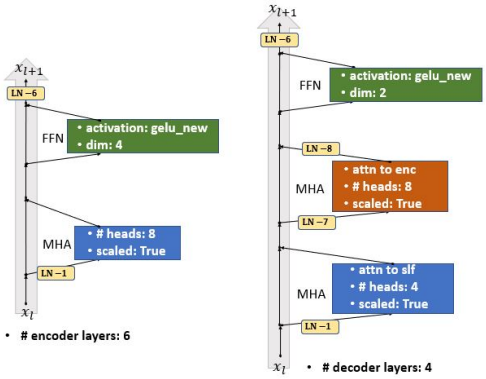


(b) $\text{AT}_{iwslt14,0.2,3}$

Figure 4: New transformer architectures learned on IWSLT14.



(a) $\text{AT}_{wmt14,0.2,0}$



(b) $\text{AT}_{wmt14,0.2,1}$

Figure 5: New transformer architectures learned on WMT-14.

Model	dev ppl	test BLEU
prev-LN Transformer	33.21 ± 0.083	22.95 ± 0.292
post-LN Transformer	30.68 ± 0.086	23.14 ± 0.248
random search	35.63 ± 0.654	19.63 ± 1.278
$\text{AT}_{iwslt14,0.2,0}$	28.86 ± 0.097	25.04 ± 0.187
$\text{AT}_{iwslt14,0.2,3}$	**27.53** ± 0.081	**25.79** ± 0.206

Table 5: Results on the IWSLT14 dataset.

Model	dev ppl	test BLEU
prev-LN Transformer (base)	-	27.1
post-LN Transformer (base)	-	27.3
Evolved Transformer (base)	**4.03**	**28.4**
random search	4.56 ± 0.764	26.85 ± 0.698
$\text{AT}_{wmt14,0.2,0}$	4.18 ± 0.092	27.68 ± 0.164
$\text{AT}_{wmt14,0.2,1}$	4.09 ± 0.076	28.05 ± 0.115

Table 6: Results on the WMT-14 dataset.

from the search comes from sampling 20% of training data per search step, which again verifies that sampling a proportional data per search epoch is beneficial both to efficiency and performance.

Results on WMT-14 For WMT-14 EN-DE task, the results for prev-LN and post-LN transformers are referenced from Wang et al. (2019) and Vaswani et al. (2017). As we can see in Figure 5, the learned models $\text{AT}_{wmt14,0.2,0}$ and $\text{AT}_{wmt14,0.2,1}$ places encoder attention after self-attention, which is consistent with IWSLT14. But now the best discovered models on WMT-14 tends to place layer-norm before the self-attention module, and before and after the feed-forward layer. Here we find that the decoder has less layers than the encoder. According to Table 5, both models outperform the standard transformers significantly, and performs comparably with the evolved transformer. Note that the evolved transformer includes branches in its architecture to allow more variable feature extraction, and the GPU time cost is 1000 times more than ours. The result shows that our approach is efficient and effective on machine translation tasks of various sizes.

5.4 Effects of Proportions of Training Data

We conduct a series of experiments on the Multi-30k dataset, trying to study the effects of using only a proportion of training data in a search epoch.

Model	dev ppl	test BLEU
prev-LN Transformer	22.33 ± 0.123	29.47 ± 0.426
post-LN Transformer	19.64 ± 0.569	31.20 ± 0.089
$AT_{multi30k,0.05,0}$	21.85 ± 0.162	29.39 ± 0.395
$AT_{multi30k,0.2,0}$	17.54 ± 0.090	33.55 ± 0.310
$AT_{multi30k,0.5,0}$	17.95 ± 0.205	31.12 ± 0.363
$AT_{multi30k,1,0}$	**17.27** ± 0.095	**33.72** ± 0.232

Table 7: Multi-30k: different proportions of train data for search.

Table 7 gives out the results for the top-ranked models for the proportion of training data being 5%, 20%, 50% and 100%, respectively. When using only 20% of the data for search, we can already learn a good architecture that outperforms the standard transformer. Using the whole training data for search can generate better performance, but the search cost is higher. Note that, when given only 5% of the training data per search epoch, the controller fails to obtain a better model. We believe the intuition behind this phenomenon is that when fed with not enough data, the reward signals the controller receives can not well represent a model's performance, thus making it difficult to design good models for the dev set. However, our experiments shows that when the resources is limited, one can use a proper proportion of data for search.

5.5 Effects of Different Learning Rates on the Learned Architecture

We study how different learning rates affect the performances of our learned architecture $AT_{multi30k,1,1}$, which is obtained by setting the learning rate to be 1e-3 during search. The learning rate is set to be 3e-3, 1e-3, 1e-4, and as always, no warm-up is used for all models. The post-LN transformer is the most sensitive to learning rate, and prev-LN transformer is robust with different learning rate, but it does not result in good performance. Our learned model $AT_{multi30k,1,1}$ is affected by learning rate, but it outperforms the two baselines significantly.

5.6 Effects of Learning Rate on Search

As shown in Table 9, learning rate affects the search results significantly. For different learning rates, the searched models given by using 20% of training data are different, and the performance difference is significant. Thus, how to incorporate the learning rate into the search space or search for models that are robust to different learning rates is an important issue we would like to further investigate.

Model	dev ppl	test BLEU
lr = 1e-3		
prev-LN Transformer	23.66 ± 0.686	29.41 ± 0.405
post-LN Transformer	19.64 ± 0.569	31.20 ± 0.089
$AT_{multi30k,1,1}$	**16.63** ± 0.143	**34.56** ± 0.325
lr = 3e-3		
prev-LN Transformer	22.33 ± 0.123	29.47 ± 0.426
post-LN Transformer	34.44 ± 9.31	22.62 ± 6.462
$AT_{multi30k,1,1}$	17.358 ± 0.290	33.253 ± 0.199
lr = 1e-4		
prev-LN Transformer	25.07 ± 0.239	28.97 ± 0.422
post-LN Transformer	22.25 ± 0.166	30.29 ± 0.202
$AT_{multi30k,1,1}$	18.50 ± 0.094	34.08 ± 0.481

Table 8: Multi-30k: different learning rates for model training.

Model	dev ppl	test BLEU
lr = 1e-3		
$AT_{multi30k,0.2,0}$	**17.54** ± 0.094	**33.55** ± 0.310
lr = 3e-3		
$AT_{multi30k,0.2,0}$	18.62 ± 0.115	33.08 ± 0.385
lr = 1e-4		
$AT_{multi30k,0.2,0}$	21.29 ± 0.079	32.88 ± 0.286

Table 9: Multi-30k: different learning rates to search.

6 Conclusions and Discussions

In this work, we have investigated how neural architecture search can improve the standard transformer architectures efficiently. We focus on the design choices that are not well studied in literature, such as how to place layer-norms, number of layers, how to place encoder attention in the decoder, etc. By applying parameter sharing and training data sampling, we can obtain improved transformer models within a couple of hours on a single GPU. Our experiments on CoNLL03, Multi-30K, IWSLT14 shows that our methodology works on different tasks of different sizes. In addition, our learned model can perform more robustly when trained with different learning rate.

There are possibilities for future work. First, how to make the transformer architectures more robust to different learning rate or minimize the effects of learning rate is an important direction. Second, although the top-ranked model during search is better than standard transformers, it may not be the best one. Thus, minimizing the gap between the search and training is a challenging and worth efforts.

References

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

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*.

Mauro Cettolo, Jan Niehues, Sebastian Stüker, Luisa Bentivogli, and Marcello Federico. Report on the 11th iwslt evaluation campaign, iwslt 2014.

Gonçalo M Correia, Vlad Niculae, and André FT Martins. 2019. Adaptively sparse transformers. 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 2174–2184.

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*.

D. Elliott, S. Frank, K. Sima'an, and L. Specia. 2016. Multi30k: Multilingual english-german image descriptions. pages 70–74.

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

Qipeng Guo, Xipeng Qiu, Pengfei Liu, Yunfan Shao, Xiangyang Xue, and Zheng Zhang. 2019. Star-transformer. *arXiv preprint arXiv:1902.09113*.

Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

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*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

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

Erik F Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. *arXiv preprint cs/0306050*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

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

Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. 2019. Mass: Masked sequence to sequence pre-training for language generation. In *International Conference on Machine Learning*, pages 5926–5936.

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*.

Qiang Wang, Bei Li, Tong Xiao, Jingbo Zhu, Changliang Li, Derek F. Wong, and Lidia S. Chao. 2019. Learning deep transformer models for machine translation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 1810–1822, Florence, Italy. Association for Computational Linguistics.

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

Ruibin Xiong, Yunchang Yang, Di He, Kai Zheng, Shuxin Zheng, Huishuai Zhang, Yanyan Lan, Liwei Wang, and Tie-Yan Liu. 2020. On layer normalization in the transformer architecture.

Hang Yan, Bocao Deng, Xiaonan Li, and Xipeng Qiu. 2019. TENER: Adapting Transformer Encoder for Named Entity Recognition. *arXiv e-prints*, page arXiv:1911.04474.

Biao Zhang, Ivan Titov, and Rico Sennrich. 2019. Improving deep transformer with depth-scaled initialization and merged attention. 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 898–909, Hong Kong, China. Association for Computational Linguistics.

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

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.