Memory-Efficient Differentiable Transformer Architecture Search

Yuekai Zhao[†], **Li Dong**[‡], **Yelong Shen**[‡], **Zhihua Zhang**[§], **Furu Wei**[‡], **Weizhu Chen**[‡]
[†]Academy for Advanced Interdisciplinary Studies, Peking University
[§]School of Mathematical Sciences, Peking University
[‡]Microsoft Corporation
`{yuekaizhao@,zhzhang@math.}pku.edu.cn`
`{lidong1,yeshe,fuwei,wzchen}@microsoft.com`

Abstract

Differentiable architecture search (DARTS) is successfully applied in many vision tasks. However, directly using DARTS for Transformers is memory-intensive, which renders the search process infeasible. To this end, we propose a multi-split reversible network and combine it with DARTS. Specifically, we devise a backpropagation-with-reconstruction algorithm so that we only need to store the last layer's outputs. By relieving the memory burden for DARTS, it allows us to search with larger hidden size and more candidate operations. We evaluate the searched architecture on three sequence-to-sequence datasets, i.e., WMT'14 English-German, WMT'14 English-French, and WMT'14 English-Czech. Experimental results show that our network consistently outperforms standard Transformers across the tasks. Moreover, our method compares favorably with big-size Evolved Transformers, reducing search computation by an order of magnitude.

1 Introduction

Current neural architecture search (NAS) studies have produced models that surpass the performance of those designed by humans (Real et al., 2019; Lu et al., 2020). For sequence tasks, efforts are made in reinforcement learning-based (Pham et al., 2018) and evolution-based (So et al., 2019; Wang et al., 2020) methods, which suffer from the huge computational cost. Instead, gradient-based methods (Liu et al., 2018; Jiang et al., 2019; Yang et al., 2020) are less demanding in computing resources and easy to implement, attracting many attentions recently.

The idea of gradient-based NAS is to train a super network covering all candidate operations. Different sub-graphs of the super network form the search space. To find a well-performing subgraph, Liu et al. (2018) (DARTS) introduced search parameters jointly optimized with the network weights. Operations corresponding to the largest search parameters are kept for each intermediate node after searching. A limitation of DARTS is its memory inefficiency because it needs to store the intermediate outputs from all its candidate operations. This is much more pronounced when we apply Transformers (Vaswani et al., 2017) as the backbone of DARTS (the operation set is detailed in Section 2.5). As shown in Figure 1, memory consumption grows extremely fast as we increase the hidden size d, quickly running out of memory as $d > 400$. As a result, we can only use a limited operation set or a small hidden size, which may lead to worse model performance.

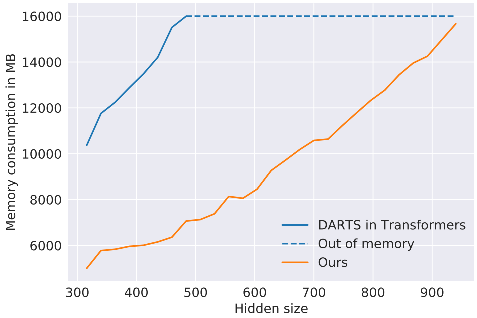


Figure 1: Memory comparison of using our reversible networks and Transformers as the backbone model of DARTS. Experiments are run on a single step of forward-backward pass on a batch of 3584 tokens with a NVIDIA P100 GPU. Limited by GPU memory, DARTS in Transformers has to search in small sizes while evaluating in large sizes, which will cause performance gaps (Chen et al., 2019).

To address the unfavorable memory consumption issue in DARTS, we propose a variant of reversible networks. Each input of a reversible network layer can be reconstructed from its outputs. Thus, it is unnecessary to store intermediate outputs except for the last layer because we can reconstruct

Findings of the Association for Computational Linguistics: ACL-IJCNLP 2021, pages 4254–4264
August 1–6, 2021. ©2021 Association for Computational Linguistics

them during backpropagation (BP). Inspired by the idea of RevNets (Gomez et al., 2017), we devise a multi-split reversible network. Each split contains a mixed operation search node to enable DARTS. Also, only a small modification of BP is needed to enable gradient calculation with input reconstruction. We show the memory consumption of our method in Figure 1, which on average halves the amount of memory required in the vanilla DARTS. We can search larger, deeper networks with a richer candidate operation set under the same memory constraint.

Our method is generic to handle various network structures. In this work, we focus on the sequence-to-sequence task. We first perform the architecture search using the WMT'14 English-German translation task. The resulting architecture is then re-trained on three datasets: WMT'14 English-German, WMT'14 English-French, and WMT'14 English-Czech. We achieve consistent improvement over standard Transformers in all tasks. At a medium model size, we can have the same translation quality as the original "big" Transformer with 69% fewer parameters. At a big model size, we exceed the performance of the Evolved Transformer (So et al., 2019), with the computational cost lowered by an order of magnitude. We will make our code and models publicly available.

2 Methodology

We give a detailed description of our method. In Section 2.1, we introduce DARTS and its memory inefficiency when applying in Transformers. In Section 2.2, we propose a multi-split reversible network, which works as the backbone of our memory-efficient architecture search approach. Section 2.3 shows a backpropagation-with-reconstruction algorithm. In Section 2.4, we manage to combine DARTS with our reversible networks. Finally, in Section 2.5, we summarize the proposed algorithms with more details.

2.1 Differentiable Architecture Search in Transformers

Following (Liu et al., 2018), we explain the idea of differentiable architecture search (DARTS) within a one-layer block. Let $\mathscr{O}$ be the candidate operation set (e.g., Self Attention, FFN, Zero). Each operation $o \in \mathscr{O}$ represents some function that can be applied to the layer inputs or hidden states (denoted X). The key of DARTS is to use a mixed operation search node $f(X)$ to relax the categorical choice of a specific operation to a softmax over all candidate operations:

$$f(X) = \sum_{o \in \mathscr{O}} \frac{\exp(\alpha_o)}{\sum_{o' \in \mathscr{O}} \exp(\alpha_{o'})} o(X), \qquad (1)$$

where the α are trainable parameters of size $|\mathscr{O}|$ that determines the mixing weights. During searching, a one-layer block contains several search nodes. The task is to find a suitable set of α for each search node. At the end of the search, the resulting operation in each node is determined by:

$$f = \arg\max_{o \in \mathscr{O}} \alpha_o. \qquad (2)$$

We optimize the α together with network weights θ by gradient descent. A good architecture means performing well on the searching validation set, such that we optimize α with validation loss L_{val} and θ with training loss L_{train}:

$$\begin{aligned} &\min_{\alpha} L_{val}(\theta^*(\alpha), \alpha), \\ &\text{s.t.} \;\; \theta^*(\alpha) = \arg\min_{\theta} L_{train}(\theta, \alpha). \end{aligned}$$

In practice, we update α by $\nabla_\alpha L_{val}$ and θ by $\nabla_\theta L_{train}$ in each step.

It is easy to directly apply DARTS in Transformers by replacing some or all operations in a Transformer block with mixed operation search nodes. For example, we can change the transformer decoder block from Self Attn → Cross Attn → FFN to Search Node 1 → Cross Attn → Search Node 2. Note that a search node outputs a weighted sum of different operations. To enable gradient calculation in the backward pass, we need to store every operation's output, which results in a steep rise in memory consumption during searching. Figure 1 shows the memory consumption of using 2 search nodes in both Transformer encoder and decoder. DARTS run out of memory easily, even at a small hidden size.

2.2 Multi-split Reversible Networks

To relieve the memory burden of DARTS in Transformers, we use reversible networks. A reversible network layer's input can be reconstructed from its output. Suppose a network is comprised of several reversible layers. We do not need to store intermediate outputs except the last layer, because we can reconstruct them from top to bottom during backpropagation (BP). Denote by X and $f(X)$ the

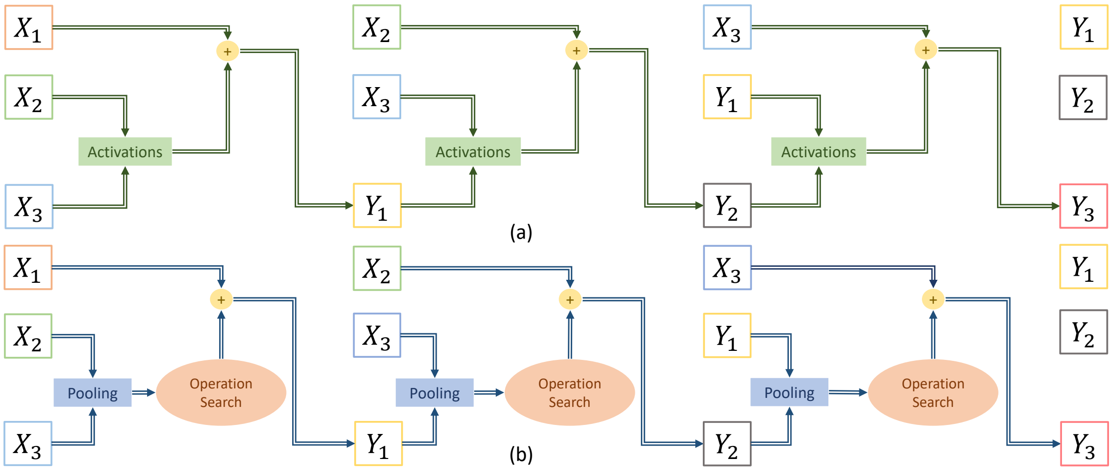


Figure 2: This figure is a demonstration of DARTSformer. (a) shows an n-split ($n = 3$) reversible network, which serves the backbone of our method. (b) shows the design of activations to enable differentiable architecture search. Each X_k and Y_k are in $R^{l \times d}$. The k-th pooling takes the concatenation of $X_{i>k}$ and $Y_{i<k}$ as the input, and outputs a tensor in $R^{l \times d}$. The operation search gives a weighted average of the outputs of each candidate operation.

layer input and the layer output, respectively. X is first split along the embedding/channel dimension into n equal parts $\{X_1, \cdots, X_n\}$. A RevNets (Gomez et al., 2017) alike operation is applied to each X_k, which yields Y_k. $f(X)$ is a concatenation of $\{Y_1, \cdots, Y_n\}$ along the split dimension:

$$\begin{aligned} Y_1 &= X_1 + G_1(X_{i>1}, \theta_1), \\ &\cdots \\ Y_k &= X_k + G_k(X_{i>k}, Y_{i<k}, \theta_k), \qquad (3) \\ &\cdots \\ Y_n &= X_n + G_n(Y_{i<n}, \theta_n). \end{aligned}$$

G_k is a mixed operation node during the architecture search process. After searching, G_k is a deterministic operation given by $\arg\max_{o \in \mathcal{O}} \alpha_o$. Detailed discussions can be found in Section 2.4.

The reversibility of Eq. (3) needs rigorous validation, such that the input X can be easily reconstructed from $f(X)$:

$$\begin{aligned} X_n &= Y_n - G_n(Y_{i<n}, \theta_n), \\ &\cdots \\ X_k &= Y_k - G_k(X_{i>k}, Y_{i<k}, \theta_k), \qquad (4) \\ &\cdots \\ X_1 &= Y_1 - G_1(X_{i>1}, \theta_1). \end{aligned}$$

Part (a) of Figure 2 illustrates a 3-split reversible network, which we frequently employ throughout our experiments for simplicity.

2.3 Backpropagation with Reconstruction

Consider the problem of backpropagating (BP) through a reversible layer. Based on the layer output $f(X) = \text{Concat}(Y_1, \cdots, Y_n)$ and its total derivative $df(X) = \text{Concat}(dY_1, \cdots, dY_n)$, we need to calculate the layer input $X = \text{Concat}(X_1, \cdots, X_n)$, its total derivative $dX = \text{Concat}(dX_1, \cdots, dX_n)$, and the derivatives of the network weights $d\theta_1, \cdots, d\theta_n$.

We show the BP-with-reconstruction through a single layer in Algorithm 1. $[\cdot]$ represents Concat$(\cdot)$ for simplicity reasons. In Line 9 of Algorithm 1, $d\theta_k$ is calculated as a side effect. Line 10 shows the reconstruction process, where each split X_k is recovered in the order of n to 1. In Algorithm 1, grad$_k$ works as a gradient accumulator, which keeps track of all derivatives associated with X_k. A repetitive application of Algorithm 1 enables us to backpropagate through a sequence of reversible layers. Only the top layer's outputs require storage, which makes it much more memory-efficient.

Roughly speaking, for a network with N connections, the forward and backward passes require approximately N and $2N$ add-multiply operations, respectively. Since we need to reconstruct X from $f(X)$, the re-calculation requires another N add-multiply operations, making it 33% slower. Fortunately, we can only need Algorithm 1 for architecture search and will re-train the resulting network with ordinary BP. The search process turns out to converge fast. The computational overhead does not become a severe problem.

[1] Automatic differentiation routines, e.g. `tf.gradient`, `torch.autograd.backward`

Algorithm 1 BP-with-reconstruction Algorithm for Multi-Split Reversible Networks

```
Input:
    Layer output: $f(X) = [Y_1, \cdots, Y_n]$;
    Total derivatives: $df(X) = [dY_1, \cdots, dY_n]$;
    Operations: $G_1, \cdots, G_n$;
Output:
    Layer input: $X = [X_1, \cdots, X_n]$;
    Derivatives of $X$: $dX = [dX_1, \cdots, dX_n]$;
$X = \{\}; dX = \{\}; Y = \{Y_1, \cdots, Y_n\}$
for k in $n$ to 1 do
    $C = Y_k; Y = Y \setminus \{Y_k\}$
    if $k == n$ then
        grad$_k = dY_k$
    else
        grad$_k = dY_k + C$.grad
    end if
    $g_k = G_k(X, Y, \theta_k)$; $g_k$.backward[1](grad$_k$)
    $X_k = C - g_k; X = X \cup \{X_k\}$
end for
$dX_1 =$ grad$_1, dX = \{dX_1\}$
for k in 2 to $n$ do
    $dX_k = X_k$.grad + grad$_k$
    $dX = dX \cup \{dX_k\}$
end for
```

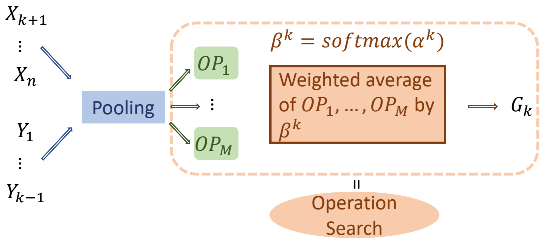
2.4 DARTS with Multi-split Reversible Networks

Performing DARTS based on n-split reversible networks only requires specifying each G_k in Eq. (3). Suppose that each $X_k \in R^{l \times d_n}$ (l is the sequence length and d is the hidden size, $d_n = \frac{d}{n}$), and that each Y_k has the same size as X_k. The input of G_k contains $n-1$ tensors in $R^{l \times d_n}$. To enable element-wise addition with X_k, the output of G_k must also be in $R^{l \times d_n}$.

G_k is factorized into two parts. The first part is a pooling operation, which takes an $l \times d_n \times (n-1)$ tensor as input, and outputs an $l \times d_n \times 1$ tensor. The second part is a mixed operation search node. G_k is calculated as follows:

$$
\begin{aligned}
H_k &= \text{Pooling}(X_{i>k}, Y_{i<k}), \\
G_k &= \sum_{o \in \mathscr{O}} \frac{\exp(\alpha_o^k)}{\sum_{o' \in \mathscr{O}} \exp(\alpha_{o'}^k)} o(H_k),
\end{aligned} \tag{5}
$$

where α_k is randomly initialized. Figure 3 shows the design of G_k. By substituting each G_k in Eq. (3) with Eq. (5), we are able to use Algorithm 1 to perform memory-efficient DARTS. We call this method *DARTSformer*, which is illustrated by Part (b) of Figure 2 in a 3-split case.



X_{k+1}
X_n
Y_1
Y_{k-1}
Pooling
OP_1
OP_M
$\beta^k = softmax(\alpha^k)$
Weighted average of $OP_1, \ldots, OP_M$ by β^k
G_k
Operation Search

Figure 3: Pooling and operation search in each split.

The overall search space size is critical to the performance of DARTSformer. In our experiments, we focus on sequence-to-sequence tasks where the encoder and the decoder are searched simultaneously. Suppose that we have an m-split encoder and an n-split decoder. We search s consecutive layers. For example, $s = 2$ means that we search within a 2-layer encoder block. Each layer in the block is an m-split reversible layer. The encoder contains several identical 2-layer blocks, the same to the decoder. The search space is of size $|\mathscr{O}|^{s(m+n)}$. If $|\mathscr{O}|$ is large, it can easily introduce a large search space even with small m, n and s.

2.5 Instantiation

We describe the instantiation of DARTSformer in this section.

Operation Set The candidate operation set $\mathscr{O}$ is defined as follows:

- Standard Conv $w \times 1$: for $w \in \{3, 5, 7, 11\}$.
- Dynamic Conv $w \times 1$: for $w \in \{3, 7, 11, 15\}$.
- Self Attention.
- Cross Attention: Only available to decoder.
- Gated Linear Unit (GLU).
- FFN.
- Zero: Return a zero tensor of the input size.
- Identity: Return the input.

The Dynamic Conv is from Wu et al. (2019). The Self Attention, Cross Attention and FFN are from Vaswani et al. (2017). We use 8 attention heads. The GLU is from Dauphin et al. (2017).

Residual connections (He et al., 2016) and layer normalization (Ba et al., 2016) are crucial for convergence in training Transformers (Vaswani et al., 2017). To make our network fully reversible, these two tricks can not be used directly. Instead, we put

Algorithm 2 The framework of DARTSformer

```
Input:
    Operation set: $\mathscr{O}$, Search parameters: $\alpha$;
    Network weights: $\theta$;
Output:
    Best candidate network: $\mathcal{N}_{final}$;
1: Setup a multi-split reversible network with operation search nodes $\mathcal{N}_{super}(\mathscr{O}, \alpha, \theta)$.
2: while $\alpha$ not converge do
3:     Update $\theta$ by Algorithm 1 with $L_{train}$.
4:     Update $\alpha$ by Algorithm 1 with $L_{val}$.
5: end while
6: Get $\mathcal{N}_{final}(\mathscr{O}, \alpha, \theta)$ with Eq. (2).
```

the residual connections and layer normalization within each operation $\tilde{o}(X) = \text{LayerNorm}(X + o(X))$, except for Zero and Identity.

Encoder and Decoder We use an n-split encoder and an $(n+1)$-split decoder for DARTSformer. Each G_k in the encoder takes the format of Eq. (5). Instead for the decoder, $G_{k<n+1}$ still follows Eq. (5), but the operation for the last split G_{n+1} is fixed as Cross Attention. Our experiments show that this constraint on the decoder yields architectures with better performances.

Search and Re-train We summarize the entire framework of DARTSformer in Algorithm 2. Note that the search process is the most memory intensive part, such that we use BP-with-reconstruction as shown in Line 2-5 of Algorithm 2.

3 Experiment Setup

3.1 Datasets

We use three standard datasets to perform our experiments as So et al. (2019): (1) WMT'18 English-German (En-De) without ParaCrawl, which consists of 4.5 million training sentence pairs. (2) WMT'14 French-English (En-Fr), which consists of 36 million training sentence pairs. (3) WMT'18 English-Czech (En-Cs), again without ParaCrawl, which consists of 15.8 million training sentence pairs. Tokenization is done by Moses[2]. We employ BPE (Sennrich et al., 2016) to generate a shared vocabulary for each language pair. The BPE merge operation numbers are 32K (WMT'18 En-De), 40K (WMT'14 En-Fr), 32K (WMT'18 En-Cs). We discard sentences longer than 250 tokens. For the re-training validation set, we randomly choose 3300 sentence pairs from the training set. The evaluation metric is *BLEU* (Papineni et al., 2002). We use beam search for test sets with a beam size of 5, and we tune the length penalty parameter from 0.5 to 1.0. Suppose the input length is m, and the maximum output length is $1.2m + 10$.

3.2 Search Configuration

The architecture searches are all run on WMT'14 En-De. DARTS is a bilevel optimization process, which updates network weights θ on one dataset and search parameters α on another dataset. We split the 4.5 million sentence pairs into 2.5/2.0 million for θ and α. Both L_{train} and L_{val} are cross entropy loss with a label smoothing factor of 0.1. The split number n is 2 for the encoder and 3 for the decoder. We set s to 1 or 2, which means the super network contains several identical 1-layer or 2-layer blocks. The candidate operations are detailed in Section 2.5, where $|\mathscr{O}| = 13/14$ for encoder and decoder, respectively. Along the analysis in Section 2.4, the largest size of the search space is around 1 billion. We use a factorized word embedding matrix to save memory. $|V|$ is the vocabulary size, and d is the hidden size. The original word embedding matrix $E \in R^{|V| \times d}$ is factorized into a multiplication of two matrices of size $|V| \times e$ and $e \times d$, where $e \ll d$. We let e denote the embedding size. We set $e = 256, d = 960$. During searching, we set the dropout probability to 0.1. Two Adam optimizers (Kingma and Ba, 2015) are used for updating θ and α, with $\beta_1 = 0.9$ and $\beta_2 = 0.98$. For θ, we use the same learning rate scheduling strategy as done in Vaswani et al. (2017) with a warmup step of 10000. The maximum learning rate is set to 5×10^{-4}. For α, we fix the learning rate to 3×10^{-4} with a weight decay of 1×10^{-3}, which is the same as Liu et al. (2018) does.

DARTSformer requires us to specify a pooling operation as stated in Eq. (5). We experiment with both max pooling and average pooling. All searches run on the same 8 NVIDIA V100 hardware. We use a batch size of 5000 tokens per GPU and save a checkpoint every 10,000 updates (5000 for θ and 5000 for α). Our search process finalizes after 60,000 updates.

3.3 Training Details

All the networks derived from the saved checkpoints are re-trained on WMT'14 En-De to select the best performing one. We then train the selected network on all datasets in Section 3.1 to

[2] https://github.com/moses-smt/mosesdecoder

Model	Pooling	Search s Layers	Model Size	BLEU
Transformer	-	-	61.1M	27.7
ET	-	-	64.1M	28.2
Sampling	max	2	60.1M	18.7
Sampling	avg	2	61.6M	16.8
DARTSformer	max	1	64.5M	27.9
DARTSformer	max	2	65.2M	**28.4**
DARTSformer	avg	1	66.0M	28.3
DARTSformer	avg	2	63.4M	28.3

Table 1: BLEU scores of various search setups on WMT'14 En-De test set. ET is the Evolved Transformer (So et al., 2019). We use a 2-split encoder and a 3-split decoder.

Model	Pooling	Splits	BLEU
DARTSformer	max	2,3	**28.4**
DARTSformer	max	3,4	28.0
DARTSformer	max	4,5	27.4
DARTSformer	avg	2,3	**28.3**
DARTSformer	avg	3,4	27.9
DARTSformer	avg	4,5	27.1

Table 2: BLEU scores of DARTSformer with different split numbers on WMT'14 En-De test set. We use an n-split encoder and an $n+1$-split decoder. We searching through 2 consecutive layers.

verify its generalization ability. We follow the settings of So et al. (2019) with both a base model and a big model. For the base model, we still use $e = 256, d = 960$ without re-scaling. For the big model, we set $e = 512, d = 1824$. Unless otherwise stated, all the training run on 8 Tesla V100 GPU cards with the batch size of 5000 tokens per card.

4 Results

4.1 Comparison Between Search Setups

We search through a different number of consecutive layers with different pooling operations. For retraining, we use the same learning rate scheduling strategy as in searching. We also keep the dropout rate unchanged. Results are summarized in Table 1. DARTSformers yields better results than standard Transformers in all experimental setups. The maximum performance gain is 0.7 BLEU with max pooling when searching through 2 consecutive layers. Also, DARTSformer achieves slightly better results than the Evolved Transformer in three out of four runs.

Model	Price	Steps	Hardware
ET	\$150k	4.2×10^8	200 TPUs
DARTSformer	\$1.25k	4.8×10^5	8 V100

Table 3: Comparison for search cost between Evolved Transformer (ET; So et al. 2019) and DARTSformer. The price for ET is from Strubell et al. (2019).

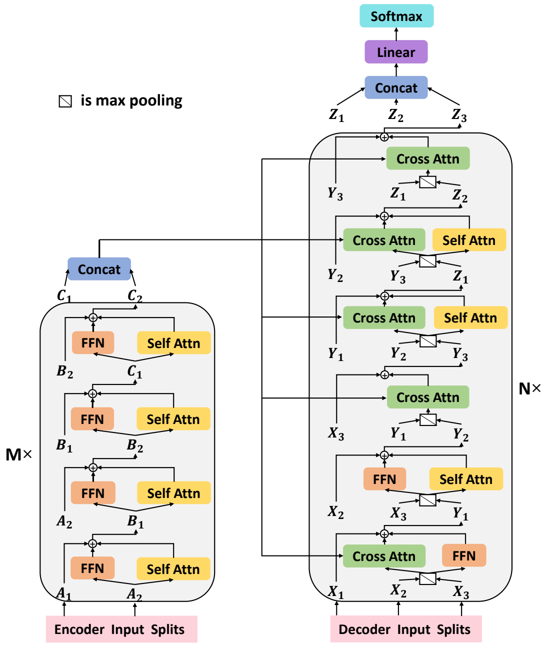


Figure 4: Architecture searched by DARTSformer.

We compare the search cost between the Evolved Transformers and DARTSformer from various aspects. DARTSformer takes about 40 hours to run on an AWS p3dn.24xlarge node[3]. The price for a single run of search is about \$1.25k. As reported by Strubell et al. (2019), the search process of Evolved Transformer takes up to \$150k, which is extremely expensive. As for hardware, the evolutionary search employs 200 TPU V.2 chips to run, while our method only uses 8 NVIDIA V100 cards. The reason for the evolutionary search algorithm's huge cost is that it requires training multiple candidate networks from scratch. We compare the number of parameter update steps in Table 3. The evolutionary search needs approximately 874 times more update steps than our method.

A simple sampling-based NAS method (Guo et al., 2020) can also reduce memory consumption.

[3] https://aws.amazon.com/ec2/instance-types/p3/

For each batch of training data, we set G_k in Eq. (5) as a uniformly sampled operation from the candidate set $\mathscr{O}$. The search parameters α are discarded, and the resulting network is produced from an evolutionary search by evaluating on the re-training validation set. This method performs poorly in machine translation, as shown in Table 1. We find that sampling-based methods favor large-kernel convolutions and that the resulting architectures tend to generate repetitive sentences.

We also experiment with increased split numbers. As shown in Table 2, an increased split number hurts the translation performance. The best results are all achieved by the smallest split. Also, the search process is harder to converge as the search space becomes too large. The re-training and inference speed will slow down when increasing the split number because more recurrence are introduced in the calculation as shown in Eq. (3).

In the following sections, we try the best search result (DARTSformer + search 2 layers + 2 split + max pooling) in various sequence-to-sequence tasks to see its generalization ability. We show this searched architecture in Figure 4.

4.2 Performance of DARTSformer on Other Datasets

First, we train DARTSformer with a base model size on three translation tasks in Section 3.1. We would like to see whether DARTSformer only performs well on the task used for architecture search or generalizes to related tasks. Second, we scale up the model size and the batch size to see whether the performance gain of DARTSformer still exists. We compare DARTSformer with standard Transformers and Evolved Transformers with similar model sizes. Following Vaswani et al. (2017), the parameter size is around 62.5M/214.7M for the base model and big model, respectively. To match the settings of So et al. (2019) when training big models, we increase the dropout rate to 0.3 and the learning rate to 1×10^{-3}. We also accumulate gradients for two batches.

Results are shown in Table 4. At the base model size, DARTSformer steadily outperforms standard Transformers. We achieved the same translation quality (28.4 BLEU, reported by Vaswani et al. (2017)) as the original big Transformer in WMT'14 En-De, with about 69% fewer parameters. Also, the maximum BLEU gain is 0.9 in WMT'14 En-Cs, which is not the dataset we conduct our architecture search on. As for Evolved Transformers, we surpass their performance in two out of three datasets, and our search algorithm is more computationally efficient. At the big model size, DARTSformer exceeds both standard Transformers and Evolved Transformers, which indicates the good generalization ability of DARTSformer.

Models	En-De	En-Fr	En-Cs
Transformer	27.7	40.0	27.0
ET (So et al., 2019)	28.2	**40.6**	27.6
DARTSformer	**28.4**	40.1	**27.9**

(a) Comparison for Base Model Size

Models	En-De	En-Fr	En-Cs
Transformer	29.1	41.2	28.1
ET (So et al., 2019)	29.3	**41.3**	28.2
DARTSformer	**29.8**	**41.3**	**28.5**

(b) Comparison for Big Model Size

Table 4: BLEU scores on WMT'14 translation tasks. ET is the Evolved Transformer. We use the best search result from different DARTSformer search setups.

4.3 Performance of DARTSformer vs. Parameter Size

In Section 4.2, DARTSformer consistently improves the performance with a model size comparable to the base and big Transformers. We are wondering whether the performance increase exists with smaller model sizes. We experiment with a spectrum of model sizes for standard Transformers and DARTSformer on WMT'14 En-De. Specifically, we use four embedding sizes for standard Transformers, [small:128, medium:256, base:512, big:1024], where its hidden size is identical to the embedding size. We also adjust the model size of DARTSformer accordingly. For base and big models, we use the results from Section 4.2. For small and medium models, we set the learning rate to 5×10^{-4}, the dropout probability to 0.1, and update the model parameters for 200,000 steps on the same 8 NVIDIA V100 hardware.

Figure 5 shows the results for both architectures. DARTSformer performs better than standard Transformers at all sizes. The BLEU increase is [1.3/0.9/0.7/0.7] for [small/medium/base/big] models. An interesting fact is that the performance gap between two models tends to be smaller as we increase the model size, which is also observed in So

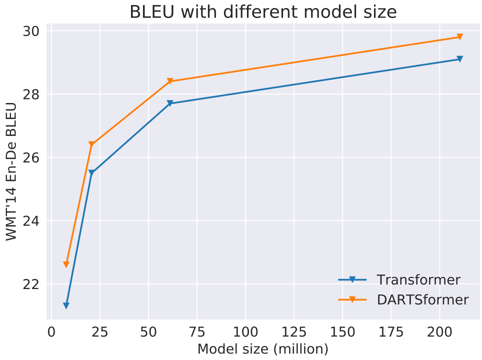


Figure 5: BLEU comparison between DARTSformer and standard Transformers with different model sizes.

Search Settings	e	d	**BLEU**
Tiny	128	120	24.2
Small	128	240	26.3
Medium	256	480	27.5
DARTSformer	256	960	**28.4**
DARTS + Transformer	320	320	27.7
Transformer	-	-	27.7

Table 5: BLEU scores of DARTS with different search hidden sizes on WMT'14 En-De test set. All searched architectures are re-trained with a parameter size similar to DARTSformer.

et al. (2019). Based on this observation, DARTSformer is more pronounced for environments with resource limitations, such as mobile phones. A possible reason for the decreased performance gap at larger model sizes is that the effect of overfitting becomes more important. We expect that some data augmentation skills (Sennrich et al., 2015; Edunov et al., 2018; Qu et al., 2020) might be of help.

4.4 The Impact of Search Hidden Size

The main motivation for our presented method is that we want to search in a large hidden size to reduce the performance gap between searching and re-training. However, whether this gap exists needs rigorous validation. Otherwise, it would suffice to instead use a small hidden size d in architecture search, and then increase d after search for training the actual model. We experiment with 4 search hidden sizes, namely, $e = 128, d = 120$ (tiny), $e = 128, d = 240$ (small), $e = 256, d = 480$ (medium), $e = 256, d = 960$ (DARTSformer). e is the word embedding size and d is the hidden size as described in Section 3.2. After obtaining the searched model, we set the model size to $e = 256, d = 960$, and re-train it on WMT'14 En-De.

The results are summarized in Table 5, which clearly shows that the translation quality is improving as the search hidden size gets larger. Also, note that when searching with tiny, small and medium settings, the final BLEU scores fall behind that of standard transformers. We argue that if one wants to evaluate the searched model in large model sizes, it is important to search with large hidden sizes. Further more, we directly apply DARTS with standard transformer as the backbone model. We set $e = 320, d = 320$. A larger search hidden size often causes memory failure due to the storage of many intermediate hidden states. As shown in Table 5, We can see that searching with a small hidden size yields no performance gain on the standard transformer.

5 Related Work

Architecture Search The field of neural architecture search (NAS) has seen advances in recent years. In the early stage, researchers focus on the reinforcement learning-based approaches (Baker et al., 2016; Zoph and Le, 2016; Cai et al., 2018a; Zhong et al., 2018) and evolution-based approaches (Liu et al., 2017; Real et al., 2017; Miikkulainen et al., 2019; So et al., 2019; Wang et al., 2020). These methods can produce architectures that outperform human-designed ones (Zoph et al., 2018; Real et al., 2019). However, the computational cost is almost unbearable since it needs to fully train and evaluate every candidate network found in the search process. Weight sharing (Brock et al., 2017; Pham et al., 2018) is a practical solution where a super network is trained, and its sub-graphs form the search space. Liu et al. (2018) proposed DARTS to use search parameters together with a super network, which allows searching with gradient descent. Gradient-based methods (Cai et al., 2018b; Xie et al., 2018; Chen et al., 2019; Xu et al., 2019; Yao et al., 2020) attracts researchers' attention since it is computationally efficient and easy to implement. We base our method on DARTS and take one step further to reduce the memory consumption of training the super network. Another recent trend is the one-stage NAS (Cai et al., 2019; Mei et al., 2019; Hu et al., 2020; Yang et al., 2020). Many NAS algorithms are in two stages. In the first stage, one searches for a good candidate network. In the second stage, the resulting network is re-initialized and re-trained. One-stage NAS tries to search and optimize the network weights simultaneously. After searching, one can have a ready-to-run network. We use a simple one-stage NAS algorithm (Guo et al., 2020) as a baseline in Section 4.1.

Reversible networks The idea of reversible networks is first introduced by RevNets (Gomez et al., 2017). Later on, Jacobsen et al. (2018); Chang et al. (2018); Behrmann et al. (2019) invented different reversible architectures based on the ResNet (He et al., 2016). MacKay et al. (2018) extended RevNets to the recurrent network, which is particularly memory-efficient. Bai et al. (2019, 2020) conducted experiments with reversible Transformers by fixed point iteration. Kitaev et al. (2020) combined local sensitive hashing attention with reversible transformers to save memory in training with long sequences. An important application of reversible networks is the flow-based models (Kingma and Dhariwal, 2018; Huang et al., 2018; Tran et al., 2019). For sequence tasks, Ma et al. (2019) achieved success in non-autoregressive machine translation.

6 Conclusion

We have proposed a memory-efficient differentiable architecture search (DARTS) method on sequence-to-sequence tasks. In particular, we have first devised a multi-split reversible network whose intermediate layer outputs can be reconstructed from top to bottom by the last layer's output. We have then combined this reversible network with DARTS and developed a backpropagation-with-reconstruction algorithm to significantly relieve the memory burden during the gradient-based architecture search process. We have validated the best searched architecture on three translation tasks.

Our method consistently outperforms standard Transformers. We can achieve the same BLEU score as the original big Transformer does with 69% fewer parameters. At a large model size, we surpass Evolved Transformers with a search cost lower by an order of magnitude. Our method is generic to handle other architectures, and we plan to explore more tasks in the future.

Acknowledgments

Yuekai Zhao and Zhihua Zhang have been supported by the Beijing Natural Science Foundation (Z190001), National Key Research and Development Project of China (No. 2018AAA0101004), and Beijing Academy of Artificial Intelligence (BAAI).

References

Lei Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization. *CoRR*, abs/1607.06450.

Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. In *Advances in Neural Information Processing Systems*, pages 690–701.

Shaojie Bai, Vladlen Koltun, and J Zico Kolter. 2020. Multiscale deep equilibrium models. *arXiv preprint arXiv:2006.08656*.

Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. 2016. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*.

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. 2019. Invertible residual networks. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 573–582. PMLR.

Andrew Brock, Theodore Lim, James M Ritchie, and Nick Weston. 2017. Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*.

Han Cai, Jiacheng Yang, Weinan Zhang, Song Han, and Yong Yu. 2018a. Path-level network transformation for efficient architecture search. In *International Conference on Machine Learning*, pages 678–687. PMLR.

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

B. Chang, L. Meng, E. Haber, Lars Ruthotto, David Begert, and E. Holtham. 2018. Reversible architectures for arbitrarily deep residual neural networks. In *AAAI*.

Xin Chen, Lingxi Xie, Jun Wu, and Qi Tian. 2019. Progressive differentiable architecture search: Bridging the depth gap between search and evaluation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1294–1303.

Yann N Dauphin, Angela Fan, Michael Auli, and David Grangier. 2017. Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*.

Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. 2017. The reversible residual network: Backpropagation without storing activations. In *Advances in neural information processing systems*, pages 2214–2224.

Zichao Guo, Xiangyu Zhang, Haoyuan Mu, Wen Heng, Zechun Liu, Yichen Wei, and Jian Sun. 2020. Single path one-shot neural architecture search with uniform sampling. In *European Conference on Computer Vision*, pages 544–560. Springer.

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

Shoukang Hu, Sirui Xie, Hehui Zheng, Chunxiao Liu, Jianping Shi, Xunying Liu, and Dahua Lin. 2020. Dsnas: Direct neural architecture search without parameter retraining. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 12084–12092.

Chin-Wei Huang, David Krueger, Alexandre Lacoste, and Aaron Courville. 2018. Neural autoregressive flows. volume 80 of *Proceedings of Machine Learning Research*, pages 2078–2087, Stockholmsmässan, Stockholm Sweden. PMLR.

Jörn-Henrik Jacobsen, Arnold W.M. Smeulders, and Edouard Oyallon. 2018. i-revnet: Deep invertible networks. In *International Conference on Learning Representations*.

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 3576–3581.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*.

Durk P Kingma and Prafulla Dhariwal. 2018. Glow: Generative flow with invertible 1x1 convolutions. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 10215–10224. Curran Associates, Inc.

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

Hanxiao Liu, Karen Simonyan, Oriol Vinyals, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Hierarchical representations for efficient architecture search. *arXiv preprint arXiv:1711.00436*.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. 2018. Darts: Differentiable architecture search. *arXiv preprint arXiv:1806.09055*.

Zhichao Lu, Gautam Sreekumar, Erik Goodman, Wolfgang Banzhaf, Kalyanmoy Deb, and Vishnu Naresh Boddeti. 2020. Neural architecture transfer. *arXiv preprint arXiv:2005.05859*.

Xuezhe Ma, Chunting Zhou, Xian Li, Graham Neubig, and Eduard Hovy. 2019. FlowSeq: Non-autoregressive conditional sequence generation with generative flow. 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 4282–4292, Hong Kong, China. Association for Computational Linguistics.

Matthew MacKay, Paul Vicol, Jimmy Ba, and Roger Grosse. 2018. Reversible recurrent neural networks. In *Neural Information Processing Systems (NeurIPS)*.

Jieru Mei, Yingwei Li, Xiaochen Lian, Xiaojie Jin, Linjie Yang, Alan Yuille, and Jianchao Yang. 2019. Atomnas: Fine-grained end-to-end neural architecture search. *arXiv preprint arXiv:1912.09640*.

Risto Miikkulainen, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. Evolving deep neural networks. In *Artificial intelligence in the age of neural networks and brain computing*, pages 293–312. Elsevier.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

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

Yanru Qu, Dinghan Shen, Yelong Shen, Sandra Sajeev, Jiawei Han, and Weizhu Chen. 2020. Coda: Contrast-enhanced and diversity-promoting data augmentation for natural language understanding.

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

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

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2015. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*.

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*.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. 2019. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*.

Dustin Tran, Keyon Vafa, Kumar Agrawal, Laurent Dinh, and Ben Poole. 2019. Discrete flows: Invertible generative models of discrete data. In *Advances in Neural Information Processing Systems*, pages 14719–14728.

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

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020. HAT: Hardware-aware transformers for efficient natural language processing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7675–7688, Online. Association for Computational Linguistics.

Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*.

Sirui Xie, Hehui Zheng, Chunxiao Liu, and Liang Lin. 2018. Snas: stochastic neural architecture search. *arXiv preprint arXiv:1812.09926*.

Yuhui Xu, Lingxi Xie, Xiaopeng Zhang, Xin Chen, Guo-Jun Qi, Qi Tian, and Hongkai Xiong. 2019. Pc-darts: Partial channel connections for memory-efficient architecture search. *arXiv preprint arXiv:1907.05737*.

Yibo Yang, Hongyang Li, Shan You, Fei Wang, Chen Qian, and Zhouchen Lin. 2020. Ista-nas: Efficient and consistent neural architecture search by sparse coding. *arXiv preprint arXiv:2010.06176*.

Quanming Yao, Ju Xu, Wei-Wei Tu, and Zhanxing Zhu. 2020. Efficient neural architecture search via proximal iterations. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 6664–6671.

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

Barret Zoph and Quoc V Le. 2016. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*.

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.