Multi-split Reversible Transformers Can Enhance Neural Machine Translation

Yuekai Zhao[1] **Shuchang Zhou**[2] **Zhihua Zhang**[3]
[1] Academy for Advanced Interdisciplinary Studies, Peking University
[2] Megvii Inc.
[3] School of Mathematical Sciences, Peking University
yuekaizhao@pku.edu.cn
zsc@megvii.com
zhzhang@math.pku.edu.cn

Abstract

Large-scale transformers have been shown the state-of-the-art on neural machine translation. However, training these increasingly wider and deeper models could be tremendously memory intensive. We reduce the memory burden by employing the idea of reversible networks that a layer's input can be reconstructed from its output. We design three types of multi-split based reversible transformers. We also devise a corresponding backpropagation algorithm, which does not need to store activations for most layers. Furthermore, we present two fine-tuning techniques: splits shuffle and self ensemble, to boost translation accuracy. Specifically, our best models surpass the vanilla transformer by at least 1.4 BLEU points in three datasets. Our large-scale reversible models achieve 30.0 BLEU in WMT'14 En-De and 43.5 BLEU in WMT'14 En-Fr, beating several very strong baselines with less than half of the training memory.

1 Introduction

Transformers (Vaswani et al., 2017) and their variants (So et al., 2019; Dehghani et al., 2019; Fonollosa et al., 2019; Liu et al., 2020; Zhu et al., 2020) significantly enhance the performance of neural machine translation (NMT). But this often requires a large size of the hidden layer (e.g., Raffel et al. (2019) used a dimension of 65K) or a deeper network by stacking more building blocks (e.g., Liu et al. (2020) used a 60-layer encoder). Training large networks could be extremely memory intensive and might even require model parallelization across multiple GPUs (Brown et al., 2020). As a result, reducing memory consumption is crucial to train wider and deeper networks efficiently.

Backpropagation (BP) is commonly used for training modern neural networks. BP needs to store layer activations to calculate the parameter gradients, which severely increases the memory burden. The idea of reversible networks can be a solution. During training, a reversible network layer's input can be reconstructed from its output. BP is run together with the reconstruction process, removing the need to store all layer activations except for the last layer. We extend the hidden dimension splitting approach by Gomez et al. (2017) and design three types of reversible transformers, namely, simple reversible transformers (**SIM-REV**), single dependent reversible transformers (**SD-REV**) and fully dependent reversible transformers (**FD-REV**). We also devise a corresponding BP algorithm for our reversible models, which is significantly memory efficient compared with the conventional BP.

Our reversible models rely on partitioning each layer's input and output into multiple equal splits. This multi-split feature inspires us to develop two fine-tuning techniques to further enhance translation accuracy. First, we randomly shuffle the output splits to encourage information sharing. Second, we train distinct translation models based on different output splits in the final decoder layer and run model ensemble during inference. These two techniques are applied after model convergence. Only a few epochs of fine-tuning are sufficient for boosting the translation performance. Also, both techniques do not break the reversibility of our proposed models.

We demonstrate that our reversible models can achieve similar or better performance than vanilla transformers do with less memory consumption. Specifically, by employing reversible training and the fine-tuning techniques, our best models can surpass vanilla transformers by 1.5 BLEU (IWSLT'14 De-En), 2.0 BLEU (WMT'19 En-Lt) and 1.4 BLEU (WMT'14 En-De). Our large-scale models also beat several very strong NMT models with less than half the training memory on WMT'14 En-De (30.0 BLEU) and WMT'14 En-Fr (43.5 BLEU).

Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics, pages 244–254
April 19 - 23, 2021.

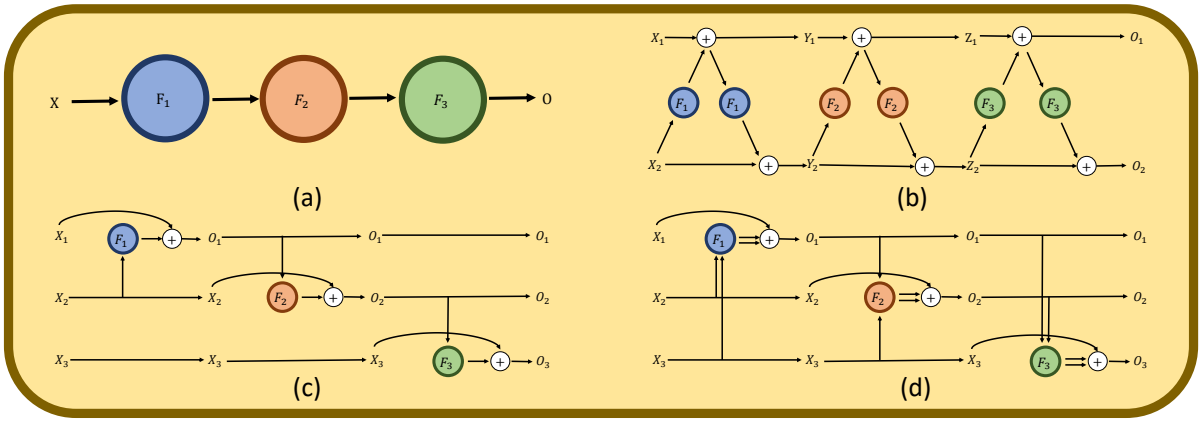

Figure 1: This figure is an illustration of different reversible architectures we explore. (a) shows a vanilla neural network (NN), where F, G and H are modules inside an NN layer. (b) shows **SIM-REV**, which resembles RevNets most. (c) shows a 3-split **SD-REV**. The i-th output split O_i depends only on the i-th input split X_i and O_{i-1}. (d) shows a 3-split **FD-REV**. Each output split O_i depends on all previous output splits $O_{<i}$ and all subsequent input splits $X_{\geq i}$.

2 Methodology

We introduce reversible transformers in this section. The definition and benefits of layer reversibility are given in Section 2.1. Section 2.2 shows three types of reversible architectures based on partitioning the layer input along the hidden/embedding dimension. Section 2.3 details the backpropagation algorithm we use. Finally, in Section 2.4, two techniques that can fit into the reversible training framework are introduced for further boosting model performance.

2.1 Reversible Architectures

A neural network layer is said to be reversible if its input can be reconstructed from its output. Usually, a network is trained in a forward-backward fashion. The activations in each layer are calculated in the forward process and stored for gradient computation in the backward process. The requirement for storing activations is memory intensive and often becomes a bottleneck for network training. However, if a network has reversible building blocks, we do not need to store the activations for most layers since they can be computed during the backward process.

A reversible layer can be designed in two ways. The first way is that this layer has an analytical inverse (Gomez et al., 2017; Jacobsen et al., 2018; Chang et al., 2018; MacKay et al., 2018). The second way is to compute the layer input via numerical methods, e.g., the fixed-point iteration (Behrmann et al., 2019). We focus on the first way following the dimension-splitting approach proposed by RevNets (Gomez et al., 2017). We give a brief review of RevNets. X is the network input, which is split into two halves X_1 and X_2. F and G are modules inside a layer (e.g., 3×3 convolutions). The forward process is as follows:

$$O_1 = X_1 + F(X_2); \quad O_2 = X_2 + G(O_1). \quad (1)$$

X_1, X_2 can be reconstructed from O_1, O_2 by:

$$X_2 = O_2 - G(O_1); \quad X_1 = O_1 - F(X_2). \quad (2)$$

2.2 Reversible Transformers

Transformers (Vaswani et al., 2017) achieved the state of the art performance in several tasks (Edunov et al., 2018; Brown et al., 2020). Despite its success, training transformers is memory intensive. We propose three reversible transformers inspired by RevNets (Gomez et al., 2017) to reduce the training memory consumption. X is the layer input. O is the layer output. X and O are partitioned into n equal splits along the hidden/embedding dimension. $X = \{X_1, X_2, \cdots, X_n\}$, $O = \{O_1, O_2, \cdots, O_n\}$. $F_1, F_2, \cdots, F_n$ are modules inside a layer (e.g. self-attention, fully connected layer).

Simple Reversible Transformer (SIM-REV) X is split into two halves X_1, X_2. For each module F_i inside a layer, the forward process resembles RevNets by changing F and G in Equation (1) into F_i. Part (b) of Figure 1 demonstrates the case of a layer with three modules:

$$\begin{aligned} Y_1 &= X_1 + F_1(X_2); \quad Y_2 = X_2 + F_1(Y_1) \\ Z_1 &= Y_1 + F_2(Y_2); \quad Z_2 = Y_2 + F_2(Z_1) \\ O_1 &= Z_1 + F_3(Z_2); \quad O_2 = Z_2 + F_3(O_1) \end{aligned} \quad (3)$$

This is the simplest way to introduce reversibility into a transformer layer. But the computation complexity is doubled compared with vanilla transformers since each module function F_i is used twice.

Single Dependent Reversible Transformer (SD-REV) We propose another reversible architecture to reduce the computational complexity. The i-th output split O_i depends only on X_i and O_{i-1}. The forward process is as follows:

$$\begin{aligned} O_1 &= X_1 + F_1(X_2) \\ O_2 &= X_2 + F_2(O_1) \\ &\dots \\ O_n &= X_n + F_n(O_{n-1}) \end{aligned} \tag{4}$$

The reconstruction of X given O is also straightforward:

$$\begin{aligned} X_n &= O_n - F_n(O_{n-1}) \\ X_{n-1} &= O_{n-1} - F_{n-1}(O_{n-2}) \\ &\dots \\ X_1 &= O_1 - F_1(X_2) \end{aligned} \tag{5}$$

Part (c) of Figure 1 shows a 3-split example of **SD-REV**. With only half of **SIM-REV**'s computational complexity, experiments show that **SD-REV** can achieve similar or even better performance as **SIM-REV** does.

Fully Dependent Reversible Transformer (FD-REV) The **SD-REV** only encodes information in neighbour splits. The lack of interaction between distant splits may make the model less expressive. We force each output split O_i to depend on all previous output splits $O_{<i}$ and all subsequent input splits $X_{\geq i}$, while preserving the reversibility of the network layer. Despite the increased computational complexity, we hope the model to have a better generalization ability. A detailed description of the **FD-REV**'s forward process is as follows:

$$\begin{aligned} O_1 &= X_1 + \sum_{i=2}^{n} F_1(X_i) \\ &\dots \\ O_k &= X_k + \sum_{i=k+1}^{n} F_k(X_i) + \sum_{j=1}^{k-1} F_k(O_j) \\ &\dots \\ O_n &= X_n + \sum_{j=1}^{n-1} F_n(O_j) \end{aligned} \tag{6}$$

The reversibility of **FD-REV** is ensured by:

$$\begin{aligned} X_n &= O_n - \sum_{j=1}^{n-1} F_n(O_j) \\ &\dots \\ X_k &= O_k - \sum_{i=k+1}^{n} F_k(X_i) - \sum_{j=1}^{k-1} F_k(O_j) \\ &\dots \\ X_1 &= O_1 - \sum_{i=2}^{n} F_1(X_i) \end{aligned} \tag{7}$$

Part (d) of Figure 1 illustrates **FD-REV** in a 3-split case. Experiments in Section 3 shows that allowing interaction between distant splits is beneficial for translation performance.

Instantiation The building blocks of transformers are attention based modules and position-wise feed-forward layers:

$$\begin{aligned} &\text{Encoder} : \text{Self-Attn} \rightarrow \text{FFN} \\ &\text{Decoder} : \text{Self-Attn} \rightarrow \text{Cross-Attn} \rightarrow \text{FFN} \end{aligned}$$

For **SIM-REV**, each of the above modules are transformed into reversible modules, where S is an input/output split:

$$\begin{aligned} \text{Encoder: } F_1(S) &= \alpha(S + \text{Self-Attn}(S)), \\ F_2(S) &= \alpha(S + \text{FFN}(S)) \\ \text{Decoder: } F_1(S) &= \alpha(S + \text{Self-Attn}(S)), \\ F_2(S) &= \alpha(S + \text{Cross-Attn}(S)), \\ F_3(S) &= \alpha(S + \text{FFN}(S)) \end{aligned}$$

For an n-split **SD-REV** or **FD-REV** (actually it has an n-split encoder and an $(n+1)$-split decoder), we use multiple Self-Attn modules and a single Cross-Attn/FFN module within each layer:

$$\begin{aligned} \text{Encoder: } F_{k<n}(S) &= \alpha(S + \text{Self-Attn}_k(S)), \\ F_n(S) &= \alpha(S + \text{FFN}(S)) \\ \text{Decoder: } F_{k<n}(S) &= \alpha(S + \text{Self-Attn}_k(S)), \\ F_n(S) &= \alpha(S + \text{Cross-Attn}(S)), \\ F_{n+1}(S) &= \alpha(S + \text{FFN}(S)) \end{aligned}$$

Applying layer normalization (LN) to the layer output O is crucial to better convergence in training transformers. However, it requires extra storage to calculate the reverse of LN. We use the ReZero (Bachlechner et al., 2020) technique as a substitution of LN. Each layer has a distinct re-scaling

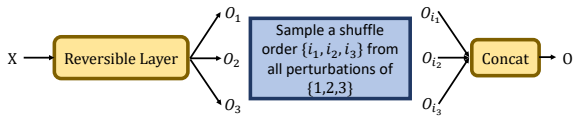

Figure 2: Illustration of shuffling splits

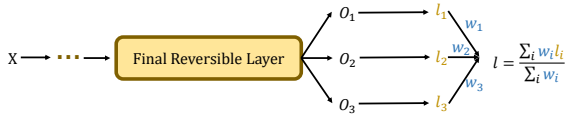

Figure 3: Illustration of self ensemble

weight α which is initialized to zero. α is trained together with other network parameters using Algorithm 1 in Section 2.3.

2.3 Backpropagation with Reconstructing Activations

In the backward pass, we are given the activations $O = \{O_1, \cdots, O_n\}$ and their total derivatives $dO = \{dO_1, \cdots, dO_n\}$. We wish to compute the inputs $X = \{X_1, \cdots, X_n\}$, their total derivatives $dX = \{dX_1, \cdots, dX_n\}$ and the derivatives of model parameters in $F_1, \cdots, F_n$. For **SIM-REV**, the backpropagation (BP) algorithm has no difference with that in Gomez et al. (2017). Such that our main focus is to derive the resulting BP algorithm for **SD-REV** and **FD-REV**.

The forward pass of **SD-REV** and **FD-REV** can be combined into a more general form:

$$O_k = X_k + G_k(X_{i>k}, O_{j<k}, \theta_k) \quad \text{where } k \in \{1, \cdots, n\}. \tag{8}$$

SD-REV corresponds to $G_k = F_k(O_{k-1})$ and **FD-REV** corresponds to $G_k = \sum_{i=k+1}^{n} F_k(X_i) + \sum_{j=1}^{k-1} F_k(O_j)$. Algorithm 1 defines the BP rule of this general form reversible network. Gradients for model parameters are computed in line 9 of Algorithm 1 as a side effect. A repeated apply of Algorithm 1 allows us to perform BP through a sequence of reversible layers, only requiring the activations and their derivatives of the top layer. In this way, the storage cost for activations can be small and independent of network depth.

2.4 Splits Shuffle and Self Ensemble

In this section, we propose two multi-split based fine-tuning methods that can enhance model performance, namely, splits shuffle and self ensemble.

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

Algorithm 1 BP Algorithm for Multi-Split Reversible Networks

Input:
Layer output: $O = \{O_1, \cdots, O_n\}$;
Derivatives of O: $dO = \{dO_1, \cdots, dO_n\}$;
Modules: $G_1, \cdots, G_n$;
Output:
Layer input: $X = \{X_1, \cdots, X_n\}$;
Derivative of X: $dX = \{dX_1, \cdots, dX_n\}$;

1: $X = \{\}; dX = \{\}$
2: **for** k in n to 1 **do**
3: $C = O_k$; $O = O \setminus \{O_k\}$
4: **if** $k == n$ **then**
5: $\text{grad}_k = dO_k$
6: **else**
7: $\text{grad}_k = dO_k + C.\text{grad}$
8: **end if**
9: $g_k = G_k(X, O, \theta_k)$; g_k.backward[1](grad_k)
10: $X_k = C - g_k$; $X = X \cup \{X_k\}$
11: **end for**
12: $dX_1 = \text{grad}_1, dX = \{dX_1\}$
13: **for** k in 2 to n **do**
14: $dX_k = X_k.\text{grad} + \text{grad}_k$
15: $dX = dX \cup \{dX_k\}$
16: **end for**

First, we train a reversible transformer till convergence. Then, several epochs of fine-tuning with one of these techniques can improve model accuracy.

Splits Shuffle A reversible transformer consists of several reversible layers. The inputs of a certain layer are the outputs of its preceding layer, which we denote as $O = \{O_1, \cdots, O_n\}$. Note that if the order of O_i is randomly shuffled, the whole network is still reversible as long as we keep a record of the shuffling order. This property inspires us to do the following fine-tuning technique:

- For each layer in the reversible network, sample $b \sim \text{Bernoulli}(p)$.
- If $b = 1$, uniformly sample a shuffle order $\{i_1, \cdots, i_n\}$ from all perturbations of $\{1, \cdots, n\}$.
- Next layer's input is $O = \{O_{i_1}, \cdots, O_{i_n}\}$.

Figure 2 shows the splits shuffle process. At inference time, we set p to 0. The idea behind splits shuffle is to apply dropout in the structure level. Splits shuffle provides a way to combine exponentially many network architectures efficiently. In

order to let each structure perform well, each split is forced to become more expressive.

Self Ensemble Model ensemble is a commonly used method for boosting translation performance (Zhou et al., 2017; Wang et al., 2020b). Model ensemble usually requires multiple distinct models to output their probability distributions over the vocabulary. The ensemble process is both computational and memory intensive. Our multi-split model offers a new chance that we can view each split of the final output as an independent model. Our self ensemble technique works as follows:

- O_i is a split in the final layer output, y is the translation target, FC stands for a fully connected layer: $P_i = \text{Softmax}(\text{FC}(O_i)); l_i = \text{loss}(P_i, y)$
- Sample a weight $w_i \sim \text{uniform}(0, 1)$. Final loss $l = \sum_{i=1}^{n} w_i l_i / \sum_{i=1}^{n} w_i$.

Figure 3 illustrates the self ensemble method. At inference time, the final distribution is an average of P_i output by each split O_i. We manage to learn an ensemble of transformers in a single reversible transformer. The inference-time memory and computational consumption are largely reduced compared with conventional ensemble methods.

3 Experiments

3.1 Datasets, Architectures and Training

Datasets We experiment on four standard corpora to demonstrate reversible transformers' effectiveness: (1) IWSLT'14 German-English (De-En), which consists of 160K training sentence pairs. (2) WMT'19 English-Lithuanian (En-Lt), which consists of 800K training sentence pairs. (3) WMT'14 English-German (En-De), which consists of 4.5M training sentence pairs. (4) WMT'14 French-English (En-Fr), which consists of 36M 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 10K (IWSLT'14 De-En), 20K (WMT'19 En-Lt), 32K (WMT'14 En-De), 40K (WMT'14 En-Fr). The evaluation metric is *BLEU* (Papineni et al., 2002). We use beam search for test datasets with a beam size of 8 and a length penalty of 0.7.

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

Architectures We experiment with all architectures proposed in Section 2.2. Multi-split based models have larger hidden dimensions than vanilla transformers do. We use a smaller embedding size than the hidden size by factorizing the word embedding matrix. N is the vocabulary size, d is the hidden size. The original word embedding matrix $E \in R^{N \times d}$ is factorized into a multiplication of two matrices of size $N \times l$ and $l \times d$, where $l \ll d$. We denote l as the embedding size. The embedding size for each language pair is 128 (IWSLT'14 De-En), 256 (WMT'19 En-Lt, WMT'14 En-De base models), 512 (WMT'14 En-De and WMT'14 En-Fr large models). For a specific language pair, we manage to ensure almost identical parameter sizes across different model architectures. One can refer to Appendix A for some details.

Training All models are trained on 8 RTX 2080Ti GPU cards with a mini-batch of 3584 tokens unless otherwise stated. We use the same learning rate scheduling strategy as (Vaswani et al., 2017) does with a warmup step of 4000. The learning rates are set to 5×10^{-4} (IWSLT'14 De-En), 7×10^{-4} (WMT'19 En-Lt, WMT'14 En-De base). The dropout probability and label smoothing factor are all set to 0.1. For training large models in Section 3.4, we increase the dropout probability to 0.3 and the learning rate to 1×10^{-3}. We also accumulate gradients for 16 batches.

3.2 Machine Translation Results

To make comparisons between various architectures, we carry experiments on all corpora except WMT'14 En-Fr. Results are summarized in Table 1. In general, the best reversible architecture can outperform the transformer baseline by 1.1 (IWSTL'14 De-En), 1.5 (WMT'19 En-Lt) and 0.8 (WMT'14 En-De) BLEU points.

All models we propose deliver similar or superior performance to the vanilla transformer. **SD-REV-2** (2 means the split number is 2) is almost as good as **SIM-REV** with only half the computational complexity. For **SD-REV**, translation performance increases as the split number becomes larger. A higher split number means more interaction between separate splits, which may benefit the translation quality. The good performance of **FD-REV** further indicates that interactions between splits should be encouraged. **FD-REV** translates best among different architectures. Increasing the split number is not necessary for **FD-REV**, since it

Model	IWSLT'14 De-En		WMT'19 En-Lt		WMT'14 En-De	
	Parameters	BLEU	Parameters	BLEU	Parameters	BLEU
Transformer (Baseline)					61.0 M	27.3
Transformer (Our impl.)	20.3 M	33.7	38.3 M	20.1	62.9 M	27.4
SIM-REV (Ours)	20.3 M	33.8	38.3 M	20.6	62.9 M	27.4
SD-REV-2 (Ours)	20.7 M	33.7	38.0 M	20.5	51.9 M	27.4
SD-REV-3 (Ours)	20.6 M	34.0	38.0 M	20.9	51.6 M	27.8
SD-REV-4 (Ours)	20.2 M	34.2	38.0 M	21.1	51.9 M	28.0
FD-REV-2 (Ours)	20.7 M	**34.8**	38.0 M	21.0	51.9 M	**28.2**
FD-REV-3 (Ours)	20.6 M	34.3	38.0 M	21.0	51.6 M	27.7
FD-REV-4 (Ours)	20.2 M	34.1	38.0 M	**21.6**	51.9 M	27.7
Best + Shuffle Splits (Ours)	20.7 M	35.2	38.0 M	22.0	51.9 M	28.8
Best + Self Ensemble (Ours)	20.7 M	35.1	38.0 M	22.1	51.9 M	28.3

Table 1: Machine translation results on different test sets. **SD-REV-n** represents an n-split single dependent reversible transformer, as is the case with **FD-REV-n**. The best results are all achieved by **FD-REV**. We explore splits shuffle and self ensemble techniques with the best architecture for each language pair. Both of the techniques benefit translation performance. The bold numbers are the best BLEU scores without using the fine-tuning techniques. The numbers with an underline are the overall best BLEU scores for a certain language pair.

Model	WMT'14 En-De		WMT'14 En-Fr	
	Parameters	BLEU	Parameters	BLEU
Ott et al. (2018)	214.0 M	29.3	214.0 M	43.2
Wu et al. (2019)	213.0 M	29.7	213.0 M	43.2
So et al. (2019)	218.1 M	29.8	221.2 M	41.3
Kitaev et al. (2020)	204.0 M	29.1		
Lioutas and Guo (2020)	209.0 M	29.6	209.6 M	43.2
Best	188.5 M	29.3	193.0 M	42.8
Best + Shuffle Splits	188.5 M	30.0	193.0 M	43.5
Best + Self Ensemble	188.5 M	29.7	193.0 M	43.5

Table 2: Machine Translation Accuracy

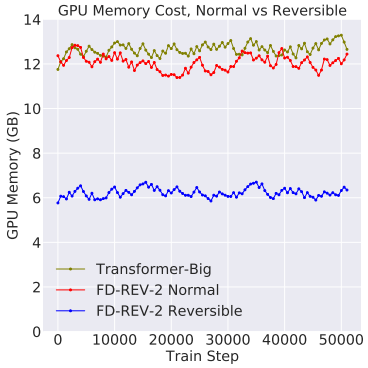

Figure 4: GPU Memory Consumption Comparison

already encodes information from different splits.

The remaining experiments are organized as follows: (1) In Section 3.3, we apply splits shuffle and self ensemble to the best models for each language pair. (2) In Section 3.4, we experiment with large model size for two large corpora, namely, WMT'14 En-De and WMT'14 En-Fr. We also try splits shuffle and self ensemble to validate their effectiveness.

3.3 Splits Shuffle and Self Ensemble

In this section, we focus on fine-tuning techniques to boost translation performance. After model convergence when training with Algorithm 1, we apply splits shuffle or self ensemble for fifteen epochs (IWSLT'14 De-En), five epochs (WMT'19 En-Lt) and one epoch (WMT'14 En-De). For the shuffle probability p, we use 0.3. Results are also summarized in Table 1.

Both fine-tuning techniques yield a performance gain over the original model. Splits shuffle is slightly better than self ensemble. Also, splits shuffle does not increase inference-time computational cost while self ensemble does. Several interesting phenomena are worth mentioning. First, the final validation perplexity decreases for splits shuffle and increases for self ensemble. Since both techniques are helpful, it is reasonable to think that splits shuffle indeed enhances model performance while self

Methods	De-En	En-Lt
ReZero (+)	34.8	21.6
ReZero (−)	32.8	19.9
Reversible Training (+)	34.8	21.6
Reversible Training (−)	34.8	21.6

Table 3: Ablation Study of ReZero and Reversible Training. **De-En** represents IWSLT'14 De-En. **En-Lt** represents WMT'19 En-Lt. + represents using a certain method. − stands for training without a certain method.

ensemble benefits more from the ensemble process. Second, a combination of splits shuffle and self ensemble fails to converge. The combination task may be too challenging for the model to learn even the model is already in a sub-optimal state. Third, we can use splits shuffle and self ensemble from the beginning. However, such complicated training objectives also bring no performance gain. Details can be found in Appendix B.

3.4 Performance and Memory Consumptions of Large Models

In this section, we investigate large-scale reversible transformers. Experiments focus on two aspects. First, whether reversible models' performance is comparable or even better than the non-reversible models? Second, how much GPU memory can be saved when using Algorithm 1 for backpropagation (BP). We choose **FD-REV-2** which performs best for WMT'14 En-De in Section 3.2. The hidden dimension is doubled, resulting in a similar parameter size with other large-scale models.

As shown in Table 2, **FD-REV-2** achieves comparable results in both datasets. The fine-tuning techniques in Section 2.4 offer a chance to enhance model performance further. We follow the settings for WMT'14 En-De in Section 3.3 and find out that large-scale models benefit more from splits shuffle and self ensemble. We can surpass various strong baselines by using splits shuffle for only one epoch of fine-tuning. Specifically, we achieve 30.0 BLEU points for WMT'14 En-De and 43.5 BLEU points for WMT'14 En-Fr.

We also compare the training memory consumption between three different settings: (1) Training Transformer-Big with conventional BP. (2) Training **FD-REV-2** with conventional BP. (3) Training **FD-REV-2** with Algorithm 1. WMT'14 En-De is

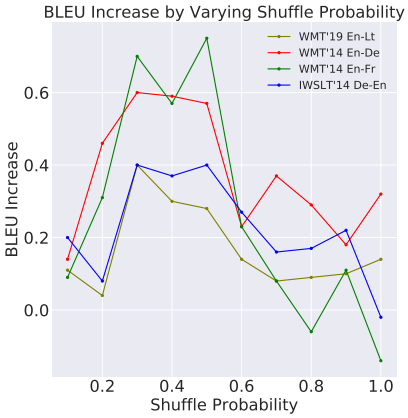

Figure 5: Effects of Shuffle Probability p

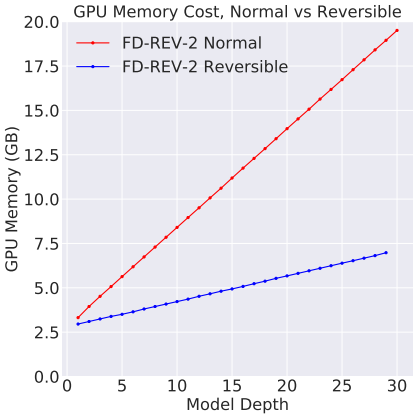

Figure 6: GPU Memory Consumption vs. Depth

the training dataset. We use a batch size of 2390 tokens and carry out one epoch of training. Figure 4 illustrates the memory consumptions of three training settings. Transformer-Big and **FD-REV-2** are similar to each other in GPU memory consumption. Reversible BP with Algorithm 1 removes the need to store activations for most layers, requiring about half of the GPU memory as conventional BP does.

4 Analysis

Splits Shuffle Probability We study the impact of splits shuffle probability p. For all language pairs, we use the best models as mentioned in Section 3.3 and Section 3.4. The results are summarized in Figure 5. We find out that a medium p value ($0.3 \leq p \leq 0.5$) yields the largest BLEU increase. A small p value is insufficient to enhance model generalization ability since the model has already been optimized for several epochs. Meanwhile, a very large p value makes the model highly unstable and hard to converge.

ReZero We study the impact of the ReZero technique. ReZero works as a substitution for layer

normalization (LN). We can not apply LN to each layer's output O due to the requirement of reversibility. Instead, we can apply LN to each output split O_i. We compare using ReZero with using LN on O_i. Table 3 shows the results. Translation performance drops severely and the model converges more slowly. Therefore we choose to use ReZero throughout our experiments.

Reversible Training vs. Normal Training Reversible training saves GPU memory. However, reconstructing activations over many layers can introduce numerical errors. Inaccurate gradients may hurt model performance, so that it is important to compare the model performance between using reversible training and conventional backpropagation (BP). As shown in Table 3, reversible training does not hurt model performance. Also, since we update the model parameters for the same number of times, the convergence speed is almost identical between reversible training and conventional BP.

Memory Consumption of Deep Models Reversible transformers are more memory efficient when the model gets deeper. We validate this argument with a simple experiment. First, we get a mini-batch of 2390 tokens. Then, one step of parameter update is done by conventional BP or reversible training. We gradually increase the model depth and keep a record of the corresponding memory consumption in a single RTX TITAN GPU card. Results are shown in Figure 6. Deeper models mean more activations to store when using conventional BP, while reversible training only needs to store the extra model parameters. The memory consumption gap can be up to 12.6 GB when we use a 30-layers encoder and a 30-layers decoder.

Computational Overhead Roughly speaking, our proposed network is composed mostly of fully connected (FC) layers. For an FC layer with N connections, the forward and backward passes require approximately N and $2N$ add-multiply operations, respectively. As the reconstruction during backpropagation (BP) adds another N add-multiply operations, training with Algorithm 1 will be 33% slower. We compare the training speed of 4 structures, namely, **FD-REV-2-base**, **FD-REV-2-big**, **Transformer-base**, **Transformer-big**. We train on WMT'14 En-De for one epoch, using a batch size of 3584 tokens. Experiments are done on a single RTX Titan GPU card with 24 GB memory. The speed measurement is words per second (wps).

Structures	use-rev	Speed (wps)
FD-REV-2-base	×	10320
FD-REV-2-base	✓	7831
FD-REV-2-big	×	3735
FD-REV-2-big	✓	2788
Transformer-base	×	10081
Transformer-big	×	3879

Table 4: Training speed comparison between Transformers and our best reversible networks. The term **use-rev** being ✓ means training with Algorithm 1. Otherwise, we train the network with conventional backpropagation.

From Table 4, we can see that **FD-REV-2** trains almost as fast as vanilla Transformers when employing conventional BP. The apply of Algorithm 1 adds about 33% to 38% training time, which in turn saves about half the memory consumption.

5 Related Work

Reversible Networks The idea of reversible training without storing activations was first introduced by RevNets (Gomez et al., 2017). Jacobsen et al. (2018) attempted to use RevNets to learn representations without loss of information. Chang et al. (2018) associated well-posed ODEs with reversible networks. MacKay et al. (2018) extended RevNets to recurrent networks. Behrmann et al. (2019) showed that a simple normalization step could make standard ResNet architectures invertible. Generative flows are often combined with reversible networks. Kingma and Dhariwal (2018) used invertible convolutions to train a generative model. Later works (Huang et al., 2018; Tran et al., 2019; Ma et al., 2019) studied generative flows with reversible networks in discrete data.

Memory-Efficient Transformers Creating memory-efficient transformers has attracted immense interest in recent years. Most works focus on proposing an optimized version of the attention module. Liu et al. (2018); Child et al. (2019); Kitaev et al. (2020); Ainslie et al. (2020); Tay et al. (2020b) limited the attention span to local neighborhoods. Wang et al. (2020a); Tay et al. (2020a) employed low rank approximations for attention matrices. Roy et al. (2020) achieved sparse attention by k-mean clustering. Katharopoulos et al. (2020) used a kernel-based self-attention to reduce memory consumption. (Ho et al., 2019)

operates on multi-dimensional tensors and applies multiple attentions, each along a single axis of the input tensor. Several works (Wu et al., 2019; Lioutas and Guo, 2020; Beltagy et al., 2020; Zaheer et al., 2020; Wu et al., 2020) incorporate convolution networks into transformers. Except for inventing new attention modules, weight sharing (Dehghani et al., 2019; Bai et al., 2019; Lan et al., 2020) is another practical approach to decreasing the memory burden. Reversible models are orthogonal to these approaches. A combination of reversible models and variants of transformers can further reduce memory consumption.

6 Conclusion

We have presented three types of multi-split based reversible transformers which outperform vanilla transformers. During backpropagation, activations for most layers need not be stored in memory because they can be reconstructed. Furthermore, we have proposed two fine-tuning techniques, namely, splits shuffle and self ensemble. Both techniques are easy to implement, and only a few fine-tuning epochs are sufficient for boosting translation performance. Our approach has beaten several strong baselines in two large datasets with fewer model parameters and much less training memory. Specifically, we have achieved 30.0 BLEU points in WMT'14 En-De and 43.5 BLEU points in WMT'14 En-Fr. Also, one can transform other network structures into their reversible versions by applying our methods. We would explore more computer vision or natural language processing tasks to widen reversible networks' applicability.

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

Joshua Ainslie, Santiago Ontañón, Chris Alberti, Philip Pham, Anirudh Ravula, and Sumit Sanghai. 2020. Etc: Encoding long and structured data in transformers. *CoRR*, abs/2004.08483.

Thomas Bachlechner, Huanru Henry Majumder, Bodhisattwa Prasad Mao, Garrison W. Cottrell, and Julian McAuley. 2020. Rezero is all you need: Fast convergence at large depth. In *arXiv*.

Shaojie Bai, J. Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Jens Behrmann, Will Grathwohl, Ricky T. Q. Chen, David Duvenaud, and Joern-Henrik Jacobsen. 2019. Invertible residual networks. volume 97 of *Proceedings of Machine Learning Research*, pages 573–582, Long Beach, California, USA. PMLR.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners.

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

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. 2018. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium. Association for Computational Linguistics.

José A. R. Fonollosa, Noe Casas, and Marta R. Costa-jussà. 2019. Joint source-target self attention with locality constraints. *arXiv preprint arXiv:1905.06596*.

Aidan Gomez, Mengye Ren, Raquel Urtasun, and Roger Grosse. 2017. The reversible residual network: Backpropagation without storing activations.

Jonathan Ho, Nal Kalchbrenner, Dirk Weissenborn, and Tim Salimans. 2019. Axial attention in multidimensional transformers. *arXiv preprint arXiv:1912.12180*.

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 Smeulders, and Edouard Oyallon. 2018. i-revnet: Deep invertible networks.

A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret. 2020. Transformers are rnns: Fast autoregressive transformers with linear attention. In *Proceedings of the International Conference on Machine Learning (ICML)*.

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, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *International Conference on Learning Representations*.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

Vasileios Lioutas and Yuhong Guo. 2020. Time-aware large kernel convolutions. In *Proceedings of the 37th International Conference on Machine Learning (ICML)*.

Peter J. Liu, Mohammad Ahmad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating wikipedia by summarizing long sequences.

Xiaodong Liu, Kevin Duh, Liyuan Liu, and Jianfeng Gao. 2020. Very deep transformers for neural machine translation. *arXiv preprint arXiv:2008.07772*.

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)*.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proceedings of the Third Conference on Machine Translation: Research Papers*, pages 1–9, Brussels, Belgium. Association for Computational Linguistics.

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.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv e-prints*.

Aurko Roy, Mohammad Taghi Saffar, David Grangier, and Ashish Vaswani. 2020. Efficient content-based sparse attention with routing transformers.

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 So, Quoc Le, and Chen Liang. 2019. The evolved transformer. In *Proceedings of the 36th International Conference on Machine Learning*, pages 5877–5886.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2020a. Synthesizer: Rethinking self-attention in transformer models. *arXiv preprint arXiv:2005.00743*.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020b. Sparse sinkhorn attention. *arXiv preprint arXiv:2002.11296*.

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, undefinedukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

Sinong Wang, Belinda Li, Madian Khabsa, Han Fang, and Hao Ma. 2020a. Linformer: Self-attention with linear complexity. *arXiv preprint arXiv:2006.04768*.

Yiren Wang, Lijun Wu, Yingce Xia, Tao Qin, ChengXiang Zhai, and Tie-Yan Liu. 2020b. Transductive ensemble learning for neural machine translation. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI*

2020, New York, NY, USA, February 7-12, 2020, pages 6291–6298. AAAI Press.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. 2020. Lite transformer with long-short range attention. In *International Conference on Learning Representations*.

Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. 2020. Big bird: Transformers for longer sequences.

Long Zhou, Wenpeng Hu, Jiajun Zhang, and Chengqing Zong. 2017. Neural system combination for machine translation. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 378–384, Vancouver, Canada. Association for Computational Linguistics.

Jinhua Zhu, Yingce Xia, Lijun Wu, Di He, Tao Qin, W. Zhou, H. Li, and T. Liu. 2020. Incorporating bert into neural machine translation. *ArXiv*, abs/2002.06823.

A Model Configuration

Table 5 details the model hyper-parameters. As we use a factorized word embedding matrix, the embedding size l is smaller than the hidden dimension d. The hidden size d increases with the number of splits n to ensure a similar parameter size for a certain dataset. Another thing worth mentioning is that **SD-REV** and **FD-REV** have identical parameter sizes as long as they have the same number of splits n, embedding size l and hidden size d. Thus, we do not differentiate between **SD-REV** and **FD-REV** in Table 5.

B More Results on Splits Shuffle and Self Ensemble

We provide more experimental results on splits shuffle and self ensemble as shown in Figure 7 and Figure 8. Using any of the techniques from the beginning tend to hurt the model performance. Meanwhile, fine-tuning with splits shuffle or self ensemble after model convergence can bring some performance gain. A combination of splits shuffle and self ensemble fails to converge when limiting the number of fine-tuning epochs (15 epochs for IWSLT'14 De-En, 5 epochs for WMT'19 En-Lt, 1 epoch for WMT'14 En-De).

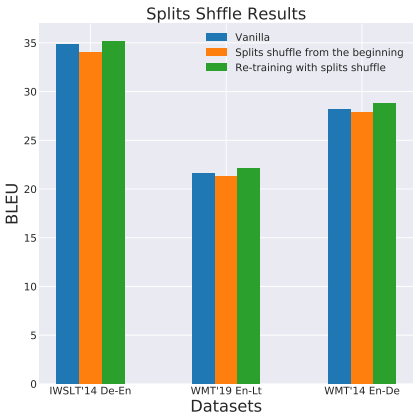

Figure 7: Splits shuffle results. No splits shuffle vs. Fine-tuning vs. Training from the beginning

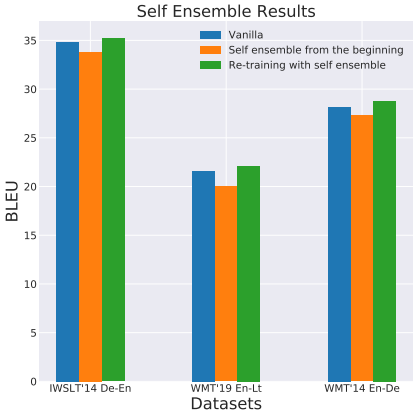

Figure 8: Self ensemble results. No self ensemble vs. Fine-tuning vs. Training from the beginning

Dataset	n	l	d
IWSLT'14 De-En	2	128	708
IWSLT'14 De-En	3	128	768
IWSLT'14 De-En	4	128	800
WMT'19 En-Lt	2	256	900
WMT'19 En-Lt	3	256	960
WMT'19 En-Lt	4	256	1040
WMT'14 En-De	2	256	1152
WMT'14 En-De	3	256	1200
WMT'14 En-De (Base)	4	256	1280
WMT'14 En-De (Big)	2	512	2304
WMT'14 En-Fr	2	512	2304

Table 5: Model configuration for **SD-REV** and **FD-REV**.n represents number of splits. l is the embedding size as mentioned in Section 3.1. d is the hidden size.