Finetuning Pretrained Transformers into RNNs

Jungo Kasai[♡*]   **Hao Peng**[♡]   **Yizhe Zhang**[♠]   **Dani Yogatama**[♣]
Gabriel Ilharco[♡]   **Nikolaos Pappas**[♡]   **Yi Mao**[♠]   **Weizhu Chen**[♠]   **Noah A. Smith**[♡◇]

[♡]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♠]Microsoft   [♣]DeepMind   [◇]Allen Institute for AI
{jkasai,hapeng,gamaga,npappas,nasmith}@cs.washington.edu
{Yizhe.Zhang, maoyi, wzchen}@microsoft.com
dyogatama@google.com

Abstract

Transformers have outperformed recurrent neural networks (RNNs) in natural language generation. But this comes with a significant computational cost, as the attention mechanism's complexity scales quadratically with sequence length. Efficient transformer variants have received increasing interest in recent works. Among them, a linear-complexity *recurrent* variant has proven well suited for autoregressive generation. It approximates the softmax attention with randomized or heuristic feature maps, but can be difficult to train and may yield suboptimal accuracy. This work aims to *convert* a pretrained transformer into its efficient recurrent counterpart, improving efficiency while maintaining accuracy. Specifically, we propose a *swap-then-finetune* procedure: in an off-the-shelf pretrained transformer, we replace the softmax attention with its linear-complexity recurrent alternative and then finetune. With a learned feature map, our approach provides an improved tradeoff between efficiency and accuracy over the standard transformer and other recurrent variants. We also show that the finetuning process has lower training cost relative to training these recurrent variants from scratch. As many models for natural language tasks are increasingly dependent on large-scale pretrained transformers, this work presents a viable approach to improving inference efficiency without repeating the expensive pretraining process.[1]

1 Introduction

Transformer models (Vaswani et al., 2017) have advanced the state of the art beyond recurrent neural network models (e.g., LSTMs, Hochreiter and Schmidhuber, 1997; GRUs, Cho et al., 2014) across a wide range of natural language processing tasks. In particular, the transformer architecture has been widely used in autoregressive modeling such as language modeling (Baevski and Auli, 2019) and machine translation (Vaswani et al., 2017). The transformer makes crucial use of interactions between feature vectors over the input sequence through the attention mechanism (Bahdanau et al., 2015). However, this comes with significant computation and memory footprint during generation. Since the output is incrementally predicted conditioned on the prefix, generation steps cannot be parallelized over time steps and require quadratic time complexity in sequence length. The memory consumption in every generation step also grows linearly as the sequence becomes longer. This bottleneck for long sequence generation limits the use of large-scale pretrained transformers, such as GPT-3 (Brown et al., 2020), Image Transformer (Parmar et al., 2018), and DALL-E (Ramesh et al., 2021).

Recent work aims at reducing the overhead of autoregressive transformers (Child et al., 2019; Kitaev et al., 2020; Beltagy et al., 2020, *inter alia*). Among them are recurrent alternatives that approximate the standard softmax attention (Katharopoulos et al., 2020; Peng et al., 2021; Choromanski et al., 2021; Schlag et al., 2021). Similar to recurrent neural networks (RNNs), those models represent the context by a recurrent state with a fixed size, thereby achieving linear time and constant memory complexity in generation sequence length. When the recurrent state size is smaller than the sequence length, these variants provide substantial speed and memory advantages over the transformer. A small state size, however, tends to deteriorate the generation quality (Peng et al., 2021), leading to a tradeoff between efficiency and accuracy.

This work improves the balance between efficiency and accuracy by a *conversion* approach: instead of training a recurrent alternative from scratch, we develop a method to *convert* a pretrained transformer into an efficient RNN that speeds up generation and reduces memory foot-

---

[*]Work was done during an internship at Microsoft.
[1]https://github.com/jungokasai/T2R/.

10630

prints. Our conversion proceeds with a *swap-then-finetune* process. Specifically, we change the exponential similarity function in the attention mechanism to the dot product after a single-layer MLP feature mapping. We then finetune the MLP parameters and the other network parameters. Our experiments in language modeling and machine translation show that the conversion can compress the context into a much smaller recurrent state than the sequence length (e.g., 1/16 of the sequence length in WikiText-103 language modeling) while retaining high accuracy. In addition, this conversion requires much less GPU time than training randomly initialized models from scratch.

State-of-the-art models in many natural language tasks are increasingly dependent on large-scale pretrained transformer models (e.g., GPT-2, Radford et al., 2019; BERT, Devlin et al., 2019; RoBERTa, Liu et al., 2019; T5, Raffel et al., 2020; BART, Lewis et al., 2020; DeBERTa, He et al., 2021). Converting a large off-the-shelf transformer to a lightweight inference model without repeating the whole training procedure is particularly useful in many downstream applications. Our work focuses on text generation and presents a viable approach towards efficient inference with high accuracy.

2   Convert a Transformer into an RNN

The transformer architecture consists of multihead attention, feedforward, and layer normalization modules (Vaswani et al., 2017). When a transformer is *trained* for a sequence generation task with teacher forcing (Williams and Zipser, 1989), the attention can be parallelized over positions because the target sequence is fully available. During *generation*, on the other hand, the output is incrementally constructed. As a result, the attention becomes an inference bottleneck for long sequences. We present a method to eliminate this bottleneck by converting a pretrained transformer into an efficient RNN of linear time and constant space complexity. We provide a detailed complexity analysis in terms of the sequence length and model dimensions.

2.1   Multihead Attention

The attention module takes as input sequences of *source* and *target* vectors. The source vectors are used to produce *key* and *value* features, while the target vectors are mapped to *query* vectors. More formally, denote by $\{\mathbf{x}_i^{\text{tgt}}\}_{i=1}^{N}$ and $\{\mathbf{x}_j^{\text{src}}\}_{j=1}^{M}$ the target and source vectors, where $\mathbf{x}_i^{\text{tgt}}, \mathbf{x}_j^{\text{src}} \in \mathbb{R}^h$

and h is the model dimensionality. We assume r attention heads of d dimensions ($h = dr$). For each head, the input vectors are first mapped to d dimensional *query*, *key*, and *value* features by learned affine transformations with $\mathbf{W}_* \in \mathbb{R}^{d \times h}$ and $\mathbf{b}_* \in \mathbb{R}^d$:

$$\mathbf{q}_i = \mathbf{W}_q \mathbf{x}_i^{\text{tgt}} + \mathbf{b}_q, \qquad (1a)$$

$$\mathbf{k}_j = \mathbf{W}_k \mathbf{x}_j^{\text{src}} + \mathbf{b}_k, \quad \mathbf{v}_j = \mathbf{W}_v \mathbf{x}_j^{\text{src}} + \mathbf{b}_v. \quad (1b)$$

The similarities of each query vector $\mathbf{q}_i$ with all M key vectors are computed and normalized to produce attention coefficients, which are then used to output a weighted average of the value vectors (Vaswani et al., 2017):

$$\mathbf{x}_i^{\text{out}} = \sum_{j=1}^{M} \frac{\text{sim}(\mathbf{q}_i, \mathbf{k}_j)}{\sum_{\ell=1}^{M} \text{sim}(\mathbf{q}_i, \mathbf{k}_\ell)} \mathbf{v}_j, \qquad (2a)$$

$$\text{sim}(\mathbf{x}, \mathbf{y}) = \exp\left(\mathbf{x} \cdot \mathbf{y} / \sqrt{d}\right). \qquad (2b)$$

Multihead attention runs this procedure for each of the r heads in parallel and concatenates r output vectors to get the final h dimensional vector.[2]

Generation Speed Overhead   Fig. 1 depicts the transformer computation steps from input vectors and their time complexity. We assume that the time complexity of multiplying an $n \times m$ matrix by an $m \times k$ is $\mathcal{O}(nmk)$ as implemented in cuBLAS (NVIDIA, 2014).[3] It consists of the following two stages.

- **Feature Mapping**: computation of $\{\mathbf{q}_i\}_{i=1}^{N}$, $\{\mathbf{k}_j\}_{j=1}^{M}$, and $\{\mathbf{v}_j\}_{j=1}^{M}$ for all r heads from input vectors (Eqs. 1a-1b). Time complexity of $\mathcal{O}(Nh^2)$, $\mathcal{O}(Mh^2)$, and $\mathcal{O}(Mh^2)$.
- **Attention**: weighted average over the value vectors (Eq. 2a). $\mathcal{O}(MNh)$, quadratic in sequence length (M, N).

Generation Memory Overhead   In autoregressive generation, query, key, and value vectors consume space complexity of $\mathcal{O}(h)$, $\mathcal{O}(Mh)$, and $\mathcal{O}(Mh)$ in every generation step. Every step's attention weight (Eq. 2a) spans over M source positions, taking $\mathcal{O}(Mr)$ space, linear in sequence length M.

2.2   Converting Transformers to RNNs

To address this generation bottleneck of quadratic time and linear space, we propose Transformer-to-RNN (T2R), a method to convert a pretrained

---

[2]Layer normalization (Ba et al., 2016), residual connection (He et al., 2016), and projection are suppressed for brevity.

[3]If the batch size is small enough, parallelization can speed up matrix multiplication.
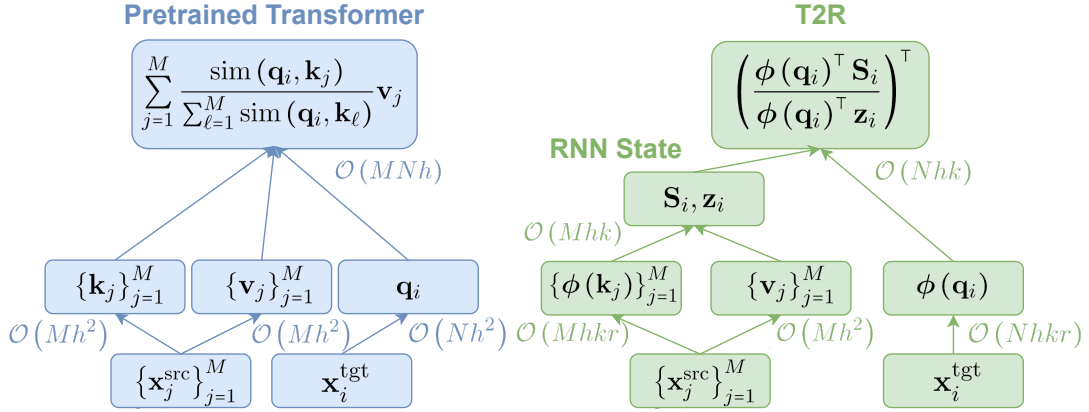
Figure 1: Attention computation steps and their time complexity in pretrained transformer and T2R models during inference generation. Features $\phi(\mathbf{q}_i)$ and $\phi(\mathbf{k}_j)$ are directly computed from input vectors, and $\mathbf{q}_i$ and $\mathbf{k}_j$ are never constructed. M: source length; N: target length; h: model dimensions; k: feature size; r: # heads.

transformer to an RNN inference model of linear time and constant memory complexity in sequence length (Fig. 1). T2R follows a swap-then-finetune procedure that modifies the attention computation of a pretrained transformer, and finetunes the model with the task objective.

We first replace the dot-then-exponential similarity function in a pretrained transformer (Eq. 2b) by

$$\widetilde{\mathrm{sim}}(\mathbf{x}, \mathbf{y}) = \phi(\mathbf{x}) \cdot \phi(\mathbf{y}), \qquad (3a)$$
$$\phi(\mathbf{x}) = \mathrm{relu}(\mathbf{W}_\phi \mathbf{x} + \mathbf{b}_\phi). \qquad (3b)$$

Here $\mathbf{W}_\phi \in \mathbb{R}^{k \times d}$ and $\mathbf{b}_\phi \in \mathbb{R}^k$ are learned parameters of a single-layer MLP. They map a d dimensional vector to a k dimensional kernel feature space. The relu activation (Fukushima, 1980) ensures that the features are non-negative.[4] Different MLP parameters are used for different attention heads, and thus we add a total of $rk(d+1)$ learnable parameters per layer (less than 0.2% parameter increase in our language model, §3). We then finetune all parameters in this modified network, including the MLP parameters, with the original task objective.[5]

During inference generation, we reformulate the attention computation (Eq. 2a) as

$$
\widetilde{\mathbf{x}}_i^{\mathrm{out}} = \sum_{j=1}^{M} \frac{\widetilde{\mathrm{sim}}(\mathbf{q}_i, \mathbf{k}_j)}{\sum_{\ell=1}^{M} \widetilde{\mathrm{sim}}(\mathbf{q}_i, \mathbf{k}_\ell)} \mathbf{v}_j
$$
$$
= \left(\frac{\phi(\mathbf{q}_i) \cdot \sum_{j=1}^{M} \phi(\mathbf{k}_j) \otimes \mathbf{v}_j}{\phi(\mathbf{q}_i) \cdot \sum_{\ell=1}^{M} \phi(\mathbf{k}_\ell)} \right)^\top \qquad (4)
$$

by the associativity of matrix multiplication. This formulation lends itself to recurrent computation. In causal attention where each query only attends to its prefix to predict the next word ($M = i$), define states:

$$\mathbf{S}_i = \sum_{j=1}^{i} \phi(\mathbf{k}_j) \otimes \mathbf{v}_j, \quad \mathbf{z}_i = \sum_{j=1}^{i} \phi(\mathbf{k}_j) \qquad (5)$$

where $\mathbf{S}_i, \mathbf{z}_i \in \mathbb{R}^{k \times d}, \mathbb{R}^k$. These states can be computed recurrently (Katharopoulos et al., 2020):

$$\mathbf{S}_i = \mathbf{S}_{i-1} + \phi(\mathbf{k}_i) \mathbf{v}_i^\top \quad \mathbf{z}_i = \mathbf{z}_{i-1} + \phi(\mathbf{k}_i) \qquad (6)$$

In the self-attention or encoder-to-decoder (cross) attention of a sequence-to-sequence model, $\mathbf{S}_i$ and $\mathbf{z}_i$ are constant with respect to i and only need to be computed once. Given the two states at position i, we can obtain the output vector:

$$\widetilde{\mathbf{x}}_i^{\mathrm{out}} = \left(\frac{\phi(\mathbf{q}_i)^\top \mathbf{S}_i}{\phi(\mathbf{q}_i)^\top \mathbf{z}_i} \right)^\top \qquad (7)$$

This avoids quadratic computation with respect to the input sequence length. We also speed up inference by merging the MLP feature map with the affine feature maps that produce queries and keys.

$$\phi(\mathbf{q}_i) = \mathrm{relu}(\widetilde{\mathbf{W}}_q \mathbf{x}_i^{\mathrm{tgt}} + \widetilde{\mathbf{b}}_q), \qquad (8a)$$
$$\phi(\mathbf{k}_j) = \mathrm{relu}(\widetilde{\mathbf{W}}_k \mathbf{x}_j^{\mathrm{src}} + \widetilde{\mathbf{b}}_k), \qquad (8b)$$

where $\quad \widetilde{\mathbf{W}}_q = \mathbf{W}_\phi \mathbf{W}_q, \quad \widetilde{\mathbf{W}}_k = \mathbf{W}_\phi \mathbf{W}_k, \quad (8c)$
$$\widetilde{\mathbf{b}}_q = \mathbf{b}_\phi + \mathbf{W}_\phi \mathbf{b}_q, \quad \widetilde{\mathbf{b}}_k = \mathbf{b}_\phi + \mathbf{W}_\phi \mathbf{b}_k. \quad (8d)$$

After the model is trained, Eqs. 8c–8d are computed once before generation; the intermediate features of $\mathbf{q}_i$ and $\mathbf{k}_j$ are never computed during inference.

Generation Speed Overhead The time complexity of each step in a T2R model is shown in Fig. 1. Similar to the transformer, it proceeds over

[4] We found that relu stabilized training by prohibiting negative similarities $\phi(\mathbf{q}) \cdot \phi(\mathbf{k})$. Other activation functions, such as $\cos$, $\tanh$, and elu, did not improve performance.

[5] We tried training the MLP parameters only, but this setting resulted in degraded development performance.

two stages.

- **Feature Mapping**: computation of $\{\phi(\mathbf{q}_i)\}_{i=1}^N$, $\{\phi(\mathbf{k}_j)\}_{j=1}^M$, and $\{\mathbf{v}_j\}_{j=1}^M$ for all r heads (Eqs. 8a–8b). Time complexity of $\mathcal{O}(Nhkr)$, $\mathcal{O}(Mhkr)$, and $\mathcal{O}(Mh^2)$.
- **Attention**: the RNN states and the outputs for r heads (Eqs. 5–7) are computed with $\mathcal{O}(Mhk)$ and $\mathcal{O}(Nhk)$.

Comparing this with the pretrained transformer, we see that if the feature size is much smaller than input sequence lengths ($k \ll M, N$), the change in the attention stage from $\mathcal{O}(MNh)$ to $\mathcal{O}(hk(M + N))$ in T2R brings a substantial speedup.

Generation Memory Overhead T2R only needs to store the RNN state, and thus its space complexity is $\mathcal{O}(hk)$, constant in sequence length. This implies reduction in memory footprint when $k \ll M$, compared to the transformer's $\mathcal{O}(Mh)$.

2.3 Autoregressive Linear Transformers

In principle, any kernel function can be used as the similarity function in Eq. 2a (Tsai et al., 2019). Previous work proposed several untrainable feature map functions ϕ and developed autoregressive transformer variants with linear time and constant space complexity in sequence length (Katharopoulos et al., 2020; Peng et al., 2021; Choromanski et al., 2021). While those models follow similar computation steps to T2R, there are several differences in generation efficiency. Since the feature map in Katharopoulos et al. (2020) preserves input dimensions, the feature size is always the same as the head dimensions ($k = d$). This means that the speedup and memory savings from using a small feature size are restricted by design. In our experiments (§3.3), our T2R models gain further efficiency by using a feature size that is even smaller than the head dimensions ($k = 32$ and $d = 128$ for language modeling). Peng et al. (2021) and Choromanski et al. (2021) scale query and key vectors by their norms before the random approximation to bound the error. Consequently, the feature mapping stage needs additional steps of producing intermediate $\mathbf{q}$ and $\mathbf{k}$ and scaling them. T2R suppresses these steps and speeds up generation further (§3.3).

3 Experiments

We present extensive experiments on standard benchmarks for language modeling and machine translation. Our results show that T2R achieves efficient autoregressive generation while retaining high accuracy.

3.1 Baselines and Comparison

We compare performance with previous transformer models for autoregressive generation with linear time and constant space complexity in input sequence length.[6] As discussed in §2.3, those prior methods correspond to two different untrainable feature maps ϕ. We experiment with two types of feature maps for comparisons: **ELU** ($\phi(\mathbf{x}) = \text{elu}(\mathbf{x}) + 1$, Katharopoulos et al., 2020); **RFA** (random feature approximation with softmax temperature reparameterization, Peng et al., 2021). Each feature map is evaluated in two settings: random initialization and pretrain. Random initialization is our reimplementation of the experiments in Katharopoulos et al. (2020) and Peng et al. (2021). The pretrain setting follows the same protocol as T2R except that we use different feature maps ϕ than our proposed one-layer MLP with relu activation. Positive orthogonal random features (**Performer**, Choromanski et al., 2021) provide similar random approximation to RFA and were evaluated in the biology domain, but we found that this method caused training divergence in the language modeling task.[7]

3.2 Setup and Implementations

We apply our method to causal attention in language models and both cross and causal attention in machine translation. For language modeling, we use a 32-dimensional feature map function. We do not modify the encoder in machine translation as its generation speed overhead is much less significant than the decoder (Kasai et al., 2021). Our exploration showed that reducing the feature size of causal attention tends to have less impact on the final translation accuracy as opposed to cross attention; we use feature sizes of 32 and 4 for cross and causal attention, respectively. This observation

---

[6]See §5 for our discussion on more transformer variants with linear time complexity, but most of those variants need modifications for autoregressive modeling and have yet to be empirically evaluated in autoregressive generation tasks.

[7]Our implementation closely follows the code released by the authors (https://github.com/lucidrains/performer-pytorch/blob/main/performer_pytorch/performer_pytorch.py#L75-L81), but does *not* subtract the maximum logit; otherwise it would disallow the linear complexity in causal attention. We conjecture that this is the reason why Performer becomes less stable in our experiments. We suspect that some techniques are necessary to improve numerical stability in language modeling and machine translation.

is consistent with previous work that showed that causal attention can be more drastically simplified than cross attention in transformer machine translation models (You et al., 2020; Tay et al., 2021).

3.2.1 Language Modeling

We use the WikiText-103 benchmark, which consists of 103M tokens sampled from English Wikipedia (Merity et al., 2017). We choose similar hyperparameters to prior work (Baevski and Auli, 2019; Fan et al., 2020): 32 layers, 8 heads, 128 head dimensions, 1024 model dimensions, 4096 fully connected dimensions and dropout (Srivastava et al., 2014) and layer dropout rates of 0.2. We partition the training data into non-overlapping blocks of 512 contiguous tokens ignoring document boundaries and train the model to predict each token from left to right (Baevski and Auli, 2019). Validation and test perplexity are measured by predicting the last 256 words out of the input of 512 consecutive words to avoid evaluating tokens in the beginning with limited context (*early token curse*, Press et al., 2021). We generally follow the optimization method from Baevski and Auli (2019), but some hyperparameters, such as the learning rate for the T2R finetuning, are adjusted for better convergence than randomly initialized training. See Appendix A.1 for more details.

3.2.2 Machine Translation

We experiment with 3 translation benchmarks: WMT14 EN-DE (4.5M train pairs, Bojar et al., 2016), WMT14 EN-FR (36M, Bojar et al., 2014), and WMT17 ZH-EN (20M, Bojar et al., 2017). We follow the preprocessing and data splits by previous work (EN-DE: Vaswani et al., 2017; EN-FR: Gehring et al., 2017; EN-ZH: Hassan et al., 2018). We use the hyperparameters of the large sized transformer (Vaswani et al., 2017): 6 layers, 16 attention heads, 1024 model dimensions, and 4096 hidden dimensions for both the encoder and decoder. We apply dropout with 0.3 and label smoothing with $\varepsilon = 0.1$. Following Ott et al. (2018), we use an increased batch size of approximately 460K tokens. Each randomly initialized model is trained for 30K (60K for the large EN-FR dataset) steps using Adam with a learning rate of $5 \cdot 10^{-4}$ and $\beta = (0.9, 0.98)$ (Kingma and Ba, 2015). We observed that convergence of the T2R conversion can be achieved with 20K (40K for EN-FR) steps and a reduced learning rate of $2 \cdot 10^{-4}$. We average the checkpoints from the last five epochs to obtain the

final model (Vaswani et al., 2017). In inference, we apply beam search with size 5 and length penalty 0.6. Consistent with previous practice, we evaluate with tokenized BLEU (Papineni et al., 2002). Further details are described in Appendix A.1.

| | | ppl. | | train |
Model	k	dev.	test	time
ELU + Random Init.	128	22.0	22.8	470h
RFA + Random Init.	32	20.4	21.3	512h
T2R + Random Init.	32	20.1	20.8	474h
ELU + Pretrain	128	21.5	22.2	97h
RFA + Pretrain	32	20.8	21.6	104h
T2R + Pretrain	32	**19.0**	**19.6**	98h
T2R 75% + Pretrain	32	17.9	18.5	95h
Pretrained Transformer	–	17.9	18.5	–
Baevski and Auli (2019)	–	–	18.7	–

Table 1: WikiText-103 language modeling results (perplexity). Train time is measured in GPU hours. The top two rows are our reimplementations of Katharopoulos et al. (2020) and Peng et al. (2021). Pretrain indicates initialization with a pretrained transformer for language modeling. T2R 75% indicates a model where every fourth layer from the top is kept as the original transformer layer. Perplexity (ppl.) is measured by predicting the last 256 words out of the input of 512 consecutive words. All models use 128 head dimensions. We assume access to a pretrained transformer model and measure the finetuning time in GPU hours.

3.3 Results

Language Modeling Seen in Table 1 are language modeling results in perplexity. We observe that T2R with the learnable MLP feature map outperforms the other two linear transformer models by more than 2.0 perplexity points in the pretrain setting. Unlike the other linear transformer models, T2R greatly benefits from pretraining (T2R + Pretrain: 19.6 vs. T2R + Random Init.: 20.8 test perplexity points). We attribute this advantage of T2R to the fact that the MLP feature map is able to learn attention patterns that are similar to those of the pretrained transformer, as evidenced in §4. Notice also that the T2R conversion is ~5x faster (measured in GPU hours) than training a model from scratch. These results illustrate that a lightweight model can be obtained without repeating the expensive training of large-scale pretrained language models such as GPT-2 and GPT-3 (Radford et al., 2019; Brown et al., 2020). T2R's generation speedup (~4x when producing 512 consecutive words) and

Model	Feature Size k		WMT14		WMT17	Train Time
	cross	causal	EN-DE	EN-FR	ZH-EN	(GPU hours)
ELU + Random Init.	64	64	28.4	*	23.4	120h
RFA + Random Init.	32	4	28.1	41.7	23.4	135h
T2R + Random Init.	32	4	27.5	39.8	23.1	123h
ELU + Pretrain	64	64	28.4	41.8	**23.8**	80h
RFA + Pretrain	32	4	27.6	41.8	23.2	90h
T2R + Pretrain	32	4	**28.7**	**42.1**	**23.8**	82h
Pretrained Transformer Large	–	–	28.9	42.2	24.2	–
Vaswani et al. (2017)	–	–	28.4	41.8	–	–

Table 2: Machine translation test results in BLEU scores. The top two rows are our reimplementations of Katharopoulos et al. (2020) and Peng et al. (2021). Pretrain indicates initialization with a trained transformer-large model. *: diverged even when running with multiple random seeds and smaller learning rates. We assume access to a pretrained transformer model and measure the finetuning time in GPU hours.

memory savings are later benchmarked with varying sequence lengths. There remains a gap of 1.1 perplexity points between the T2R and pretrained transformer models (19.6 vs. 18.5). However, the gap can be closed when every fourth layer from the top is kept as the original transformer layer and the model is finetuned in the same way (T2R 75%). This suggests that keeping a small fraction of the quadratic attention layers can provide an effective middle ground between efficiency and accuracy.[8]

Machine Translation Seen in Table 2 are machine translation results in BLEU from various configurations. Departing from the language modeling experiments, the T2R model underperforms the other two linear transformer models when initialized randomly. However, consistent with language modeling, the T2R model substantially benefits from pretraining (e.g., 28.7 vs. 27.5 BLEU points in EN-DE). As a result, the T2R model achieves similar BLEU scores to the original transformer across all language pairs. ELU trained from the pretrained transformer yields comparable performance to T2R, but the feature size is much larger (64 vs. 32 and 64 vs. 4 in cross and causal attention), thus leading to increased overhead, as shown later. Note that the T2R finetuning time is only moderately smaller than that of randomly initialized training here, but further speedup in conversion can be potentially achieved with more extensive hyperparameter tuning.[9]
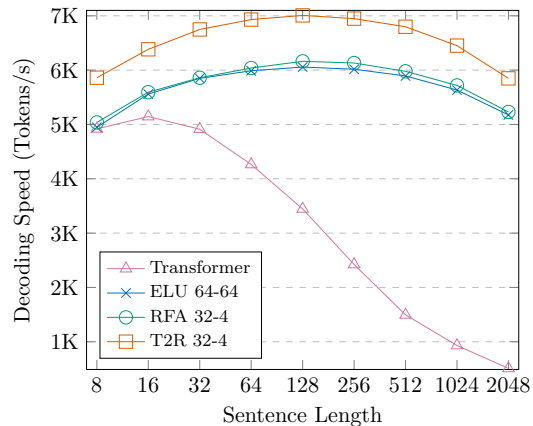

Figure 2: Machine translation speed of various models. Speed is measured on a single TPU v2 accelerator with batch size 16 and beam size 1, following Peng et al. (2021). 32-4 indicates the feature sizes of 32 and 4 for cross and causal attention, respectively.

Speedup and Memory Savings in Generation We run a conditional generation experiment to compare the decoding speed of the models in Table 2 (Fig. 2). Here we assume the input and output sequences are of the same length. All models are tested using greedy decoding with the same batch size of 16 on a TPU v2 accelerator.[10] We see that indeed the linear transformer models can generate an almost constant number of tokens per second regardless of the sequence length and outpace the transformer model dramatically as the sequence becomes longer. The T2R model achieves a 15%+

---

[8]Concurrent work (Lei, 2021) also explores reducing the number of attention layers for efficiency.

[9]We found that the batch size could be reduced for T2R conversion without hurting accuracy, while randomly initialized models deteriorate with small batch sizes. This suggests

that the computational cost for conversion can be much lighter than training from scratch, and T2R is advantageous when only a limited number of GPUs are available.
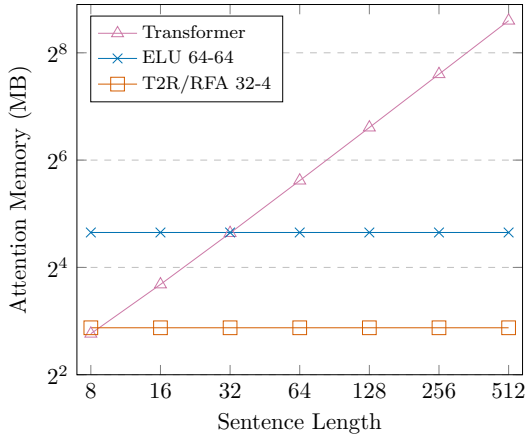
[10]https://opensource.google/projects/jax.

Figure 3: Memory consumption from the attention computation of various machine translation models in inference with batch size 16 and beam size 1.
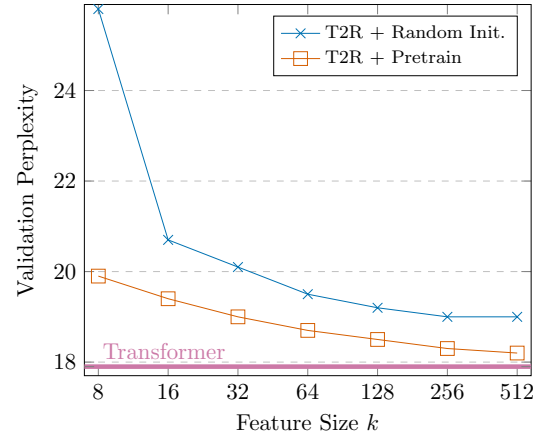

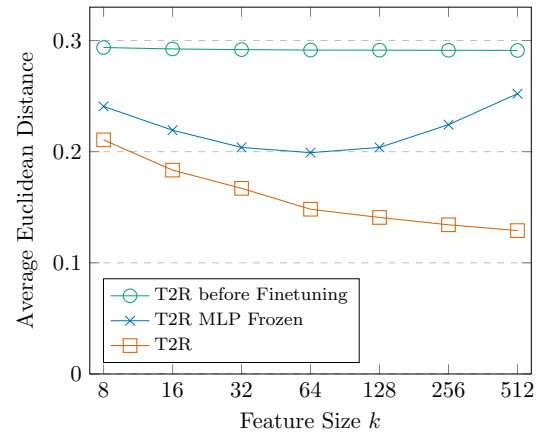Figure 4: WikiText-103 validation perplexity with varying feature sizes.

Figure 5: Average Euclidean distance of T2R models from the transformer attention weights with varying feature sizes. The distances are computed on the Wikitext-103 validation data for predicting a word given the preceding 512 words. All models are initialized with a pretrained transformer model.

speedup over ELU and RFA due to its smaller feature sizes and faster feature mapping respectively; this confirms our analysis on T2R's speed advantage over them (§2.3). Fig. 3 plots memory consumption from the attention computation during decoding for machine translation. Since the T2R, RFA, and ELU models compress keys and values into a $k \times d$ matrix $\mathbf{S}$ and a k dimensional vector $\mathbf{z}$ (§2.2), the required memory at each decoding step is constant over varying sequence lengths. It is also roughly proportional to the feature size k. The MLP feature map in the T2R model allows for small feature dimensions than the ELU feature of the head dimensions, resulting in a 70% memory reduction. The attention computation in the standard transformer, on the other hand, consumes memory linearly in sequence length at each decoding step because all previous key and value vectors have to be stored. We also found a similar speedup and memory savings in unconditional generation with the T2R language model (~4x speedup in generating 512 consecutive words over the transformer).

4   Analysis and Ablations

We presented T2R, a method to convert a pretrained transformer into an efficient RNN. In this section, we analyze our conversion approach by examining the impact of the feature size and induced attention weight distributions. Our analysis shows that T2R implicitly learns attention distributions similar to the original transformer.

Feature Size and Pretraining   We saw that T2R benefits substantially from transformer pretraining. Fig. 4 compares T2R with pretraining and random

initialization in terms of the relation between the validation perplexity from WikiText-103 and the feature sizes. We see that as the feature size (RNN state size) becomes smaller, pretraining becomes particularly important to achieve low perplexity. Transformer pretraining achieves a Pareto improvement over random initialization in the tradeoff between efficiency (small feature size) and accuracy (low perplexity).

Attention Distribution   T2R is not explicitly trained to mimic the original attention distributions, and there is no guarantee that the MLP feature map approximates the exponential similarity function, unlike previous approximation approaches (Peng et al., 2021; Choromanski et al., 2021). Here, we analyze the properties of the attention weight dis-

10636

tributions that are induced by finetuning. We use the validation data from WikiText-103 and run language models to predict the next word given the input of 512 contiguous words. We compute the attention weight distribution over the 512 words for each attention head in the model layers.

Fig. 5 compares the attention distributions from T2R in various configurations. T2R MLP frozen indicates a model that is finetuned with the MLP parameters frozen. Euclidean distances in attention distributions between the original transformer and each model are averaged across validation samples, model layers, and attention heads.[11] Comparing T2R before finetuning and the full T2R model, we see that the finetuning process induces much more similar attention distributions, and the distance diminishes as the feature size increases (and the perplexity approaches the original transformer, Fig. 4). We also observed that when the MLP parameters are not trained (T2R MLP frozen), the distance from the original attention distributions increases. These results suggest that finetuning of the whole network in T2R implicitly develops similar attention distributions to the original transformer even though the training supervision comes solely from language modeling.

5 Further Related Work

In addition to the work we already discussed, we highlight related methods from prior work that make transformer models efficient.

5.1 Knowledge Distillation

Knowledge distillation (Hinton et al., 2015) is closely related to our T2R conversion and uses a similar pipeline: a teacher model with large capacity is first trained and is used to generate *silver* training data for a new lightweight inference model. It has been successfully applied to machine translation (e.g., Kim and Rush, 2016; Gu et al., 2018) to make generation efficient. In particular, several prior works distill a transformer translation model to an RNN (Senellart et al., 2018; Kim et al., 2019). We share the same motivation toward fast generation with light memory, but our approach differs in two ways: the original training data are used for finetuning an RNN model, and its model parameters are initialized with the "teacher" transformer.

Our method does not use the computationally expensive teacher model to generate new training data. While data generation is a one-time computational cost, it becomes expensive as the teacher model size and training data increase. Moreover, since the pretrained parameters can be directly used, conversion requires fewer GPU hours than training a brand new lightweight model from scratch (§3.3).

5.2 Efficient Transformers

Prior work suggested many other strategies to improve efficiency in transformers, such as weight sharing and factorization (Dehghani et al., 2019; Lan et al., 2020), weight and layer pruning (Michel et al., 2019; Fan et al., 2020), quantization (Zafrir et al., 2019; Shen et al., 2020), and modifying the combination of sublayers (Press et al., 2020; Mandava et al., 2020). Some of these methods present orthogonal design choices and can be integrated into our T2R model to gain further efficiency. For a more comprehensive survey, see Tay et al. (2020b). Below we describe several prior works along two major strategies: compressing the attention context and sparsifying the attention patterns.

Attention Context Compression This strand of methods compresses the context that is attended to, thereby reducing the time and memory overhead in the attention. RNN models that we converted pretrained transformers into compress the context into a recurrent state. Other approaches include low rank approximation of the attention computation (Wang et al., 2020; Tay et al., 2021) and adding a memory module that can access multiple tokens at once (Liu et al., 2018; Dai et al., 2019; Lee et al., 2019; Ainslie et al., 2020; Rae et al., 2020; Beltagy et al., 2020; Zaheer et al., 2020).

Sparse Attention Patterns Another approach to reducing the time and memory overhead from the attention computation is to limit the tokens that are attended to by sparsifying the attention patterns. These patterns can be set in advance or learned during training (Tay et al., 2020b). For example, prior works introduced fixed patterns of blockwise attention (Qiu et al., 2020) and strided attention (Child et al., 2019; Beltagy et al., 2020; Zaheer et al., 2020). Other previous works presented methods to learn attention patterns from data (Sukhbaatar et al., 2019; Roy et al., 2020; Tay et al., 2020a).

It should be noted that significant modifications are necessary to apply many of these methods to autoregressive generation tasks such as language

modeling and machine translation, and their empirical evaluation in these generation settings has yet to be conducted (Peng et al., 2021). This work presents extensive empirical evaluation in autoregressive generation settings.

6 Conclusion and Future Work

We present T2R, a method that converts a pretrained transformer to a recurrent neural network that reduces the time and memory cost of autoregressive generation. Our experiments in language modeling and machine translation demonstrated that our model produces an improved trade-off between efficiency and accuracy over randomly initialized training and previous models with lightweight attention. Our work provides further support for the claim that large-scale pretrained models can be compressed into efficient inference models that facilitate downstream applications.

Acknowledgments

References

Joshua Ainslie, Santiago Ontanon, Chris Alberti, Vaclav Cvicek, Zachary Fisher, Philip Pham, Anirudh Ravula, Sumit Sanghai, Qifan Wang, and Li Yang. 2020. ETC: Encoding long and structured inputs in transformers. In *Proc. of EMNLP*.

Jimmy Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer normalization.

Alexei Baevski and Michael Auli. 2019. Adaptive input representations for neural language modeling. In *Proc. of ICLR*.

Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Proc. of ICLR*.

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

Ondřej Bojar, Christian Buck, Christian Federmann, Barry Haddow, Philipp Koehn, Johannes Leveling, Christof Monz, Pavel Pecina, Matt Post, Herve Saint-Amand, Radu Soricut, Lucia Specia, and Aleš Tamchyna. 2014. Findings of the 2014 workshop on statistical machine translation. In *Proc. of WMT*.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. 2017. Findings of the 2017 conference on machine translation (WMT17). In *Proc. of WMT*.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurélie Névéol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proc. of WMT*.

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. In *Proc. of NeurIPS*.

Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. 2019. Generating long sequences with sparse transformers.

Kyunghyun Cho, Bart van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. 2014. On the properties of neural machine translation: Encoder–decoder approaches. In *Proc. of SSST-8*.

Krzysztof Choromanski, Valerii Likhosherstov, David Dohan, Xingyou Song, Andreea Gane, Tamás Sarlós, Peter Hawkins, Jared Davis, Afroz Mohiuddin, Lukasz Kaiser, David Belanger, Lucy Colwell, and Adrian Weller. 2021. Rethinking attention with Performers. In *Proc. of ICLR*.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive language models beyond a fixed-length context. In *Proc. of ACL*.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Lukasz Kaiser. 2019. Universal transformers. In *Proc. of ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proc. of NAACL*.

Angela Fan, Edouard Grave, and Armand Joulin. 2020. Reducing transformer depth on demand with structured dropout. In *Proc. of ICLR*.

Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*.

Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann Dauphin. 2017. Convolutional sequence to sequence learning. In *Proc. of ICML*.

Jiatao Gu, James Bradbury, Caiming Xiong, Victor O. K. Li, and Richard Socher. 2018. Non-autoregressive neural machine translation. In *Proc. of ICLR*.

Hany Hassan, Anthony Aue, Chang Chen, Vishal Chowdhary, Jonathan Clark, Christian Federmann, Xuedong Huang, Marcin Junczys-Dowmunt, William Lewis, Mengnan Li, Shujie Liu, Tie-Yan Liu, Renqian Luo, Arul Menezes, Tao Qin, Frank Seide, Xu Tan, Fei Tian, Lijun Wu, Shuangzhi Wu, Yingce Xia, Dongdong Zhang, Zhirui Zhang, and Ming Zhou. 2018. Achieving human parity on automatic Chinese to English news translation.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *Proc. of CVPR*.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2021. DeBERTa: Decoding-enhanced BERT with disentangled attention.

Geoffrey E. Hinton, Oriol Vinyals, and Jeffrey Dean. 2015. Distilling the knowledge in a neural network. In *Proc. of NeurIPS Deep Learning and Representation Learning Workshop*.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*.

Hakan Inan, Khashayar Khosravi, and Richard Socher. 2017. Tying word vectors and word classifiers: A loss framework for language modeling. In *Proc. of ICLR*.

Jungo Kasai, Nikolaos Pappas, Hao Peng, James Cross, and Noah A. Smith. 2021. Deep encoder, shallow decoder: Reevaluating non-autoregressive machine translation. In *Proc. of ICLR*.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. 2020. Transformers are RNNs: Fast autoregressive transformers with linear attention. In *Proc. of ICML*.

Yoon Kim and Alexander M. Rush. 2016. Sequence-level knowledge distillation. In *Proc. of EMNLP*.

Young Jin Kim, Marcin Junczys-Dowmunt, Hany Hassan, Alham Fikri Aji, Kenneth Heafield, Roman Grundkiewicz, and Nikolay Bogoychev. 2019. From research to production and back: Ludicrously fast neural machine translation. In *Proc. of WNGT*.

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A method for stochastic optimization. In *Proc. of ICLR*.

Nikita Kitaev, Lukasz Kaiser, and Anselm Levskaya. 2020. Reformer: The efficient transformer. In *Proc. of ICLR*.

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 *Proc. of ICLR*.

Juho Lee, Yoonho Lee, Jungtaek Kim, Adam Kosiorek, Seungjin Choi, and Yee Whye Teh. 2019. Set transformer: A framework for attention-based permutation-invariant neural networks. In *Proc. of ICML*.

Tao Lei. 2021. When attention meets fast recurrence: Training language models with reduced compute.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising sequence-to-sequence pretraining for natural language generation, translation, and comprehension. In *Proc. of ACL*.

Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. 2018. Generating Wikipedia by summarizing long sequences. In *Proc. of ICLR*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke S. Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A robustly optimized BERT pretraining approach.

Ilya Loshchilov and Frank Hutter. 2017. SGDR: stochastic gradient descent with restarts.

Swetha Mandava, Szymon Migacz, and Alex Fit Florea. 2020. Pay attention when required.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proc. of ICLR*.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? In *Proc. of NeurIPS*.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. In *Proc. of ICLR*.

NVIDIA. 2014. The NVIDIA CUDA basic linear algebra subroutines (CUBLAS).

Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng, David Grangier, and Michael Auli. 2019. fairseq: A fast, extensible toolkit for sequence modeling. In *NAACL Demonstrations*.

Myle Ott, Sergey Edunov, David Grangier, and Michael Auli. 2018. Scaling neural machine translation. In *Proc. of WMT*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: a method for automatic evaluation of machine translation. In *Proc. of ACL*.

Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam Shazeer, Alexander Ku, and Dustin Tran. 2018. Image transformer. In *Proc. of ICML*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *Proc. of NeurIPS*.

Hao Peng, Nikolaos Pappas, Dani Yogatama, Roy Schwartz, Noah A. Smith, and Lingpeng Kong. 2021. Random feature attention. In *Proc. of ICLR*.

Ofir Press, Noah A. Smith, and Omer Levy. 2020. Improving transformer models by reordering their sublayers. In *Proc. of ACL*.

Ofir Press, Noah A. Smith, and Mike Lewis. 2021. Shortformer: Better language modeling using shorter inputs. In *Proc. of ACL*.

Ofir Press and Lior Wolf. 2017. Using the output embedding to improve language models. In *Proc. of EACL*.

Jiezhong Qiu, Hao Ma, Omer Levy, Wen-tau Yih, Sinong Wang, and Jie Tang. 2020. Blockwise self-attention for long document understanding. In *Proc. of EMNLP*.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, Chloe Hillier, and Timothy P. Lillicrap. 2020. Compressive transformers for long-range sequence modelling. In *Proc. of ICLR*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*.

Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. 2021. Zero-shot text-to-image generation.

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

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. 2021. Linear transformers are secretly fast weight memory systems. In *Proc. of ICML*.

Jean Senellart, Dakun Zhang, Bo Wang, Guillaume Klein, Jean-Pierre Ramatchandirin, Josep Crego, and Alexander Rush. 2018. OpenNMT system description for WNMT 2018: 800 words/sec on a single-core CPU. In *Proc. of WNG*.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proc. of ACL*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W. Mahoney, and Kurt Keutzer. 2020. Q-BERT: hessian based ultra low precision quantization of BERT. In *Proc. of AAAI*.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *JMLR*.

Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. 2019. Adaptive attention span in transformers. In *Proc. of ACL*.

Yi Tay, Dara Bahri, Donald Metzler, Da-Cheng Juan, Zhe Zhao, and Che Zheng. 2021. Synthesizer: Rethinking self-attention in transformer models. In *Proc. of ICML*.

Yi Tay, Dara Bahri, Liu Yang, Donald Metzler, and Da-Cheng Juan. 2020a. Sparse sinkhorn attention. In *Proc of ICML*.

Yi Tay, M. Dehghani, Dara Bahri, and Donald Metzler. 2020b. Efficient Transformers: A survey.

Yao-Hung Hubert Tsai, Shaojie Bai, Makoto Yamada, Louis-Philippe Morency, and Ruslan Salakhutdinov. 2019. Transformer dissection: An unified understanding for transformer's attention via the lens of kernel. In *Proc. of EMNLP*.

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 *Proc. of NeurIPS*.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020. Linformer: Self-attention with linear complexity.

Ronald J. Williams and David Zipser. 1989. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*.

Felix Wu, Angela Fan, Alexei Baevski, Yann Dauphin, and Michael Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *Proc. of ICLR*.

Weiqiu You, Simeng Sun, and Mohit Iyyer. 2020. Hard-coded Gaussian attention for neural machine translation. In *Proc. of ACL*.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: quantized 8bit BERT. In *Proc. of EMC²*.

Manzil Zaheer, Guru Guruganesh, Kumar 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. In *Proc. of NeurIPS*.

A Appendix

A.1 Hyperparameters and Setting

All training is implemented in `fairseq` (Ott et al., 2019) and run with PyTorch 1.7.1 (Paszke et al., 2019), 8 Telsa V100 GPUs, and CUDA 11.0. We used mixed precision and distributed training over 8 GPUs (Micikevicius et al., 2018; Ott et al., 2018). Apart from EN→ZH where we used separate BPE operations and only tied the decoder input and output embeddings, we tie all embeddings (Press and Wolf, 2017; Inan et al., 2017). We experimented with feature sizes of [16, 32, 64] and [4, 8, 16, 32] for language modeling and machine translation respectively, and chose the smallest feature sizes that retained the development performance compared to the standard transformer.

A.1.1 Language Modeling

We generally follow the optimization method from Baevski and Auli (2019). For optimizing a model from random initialization, the learning rate is linearly warmed up from 10^{-7} to 1 for the initial 16K steps and then annealed using a cosine learning rate schedule with cycles (Loshchilov and Hutter, 2017). Each period lasts for twice the number of updates than the previous cycle, and we lower the maximum and minimum learning rates by 25% compared to the previous cycle. The initial minimum and maximum learning rates are 10^{-5} and 1 respectively (Baevski and Auli, 2019). We train the model with a batch size of about 74K tokens with a total of 286K steps (Baevski and Auli, 2019). When we convert a pretrained transformer to an RNN model by finetuning, we found that we could speed up training by reducing the warm-up steps, total update steps, maximum and minimum rates, and batch size to 8K steps, 142K steps, $5 \cdot 10^{-6}$, 0.5, and 25K tokens without loss in validation perplexity.

Randomly Initialized Training We generally follow the hyperparameters chosen in Baevski and Auli (2019); Fan et al. (2020). Specifically, we list the hyperparameters in Table 3 for easy replication. All other hyperparameter options are left as default values in `fairseq`.

Finetuning Pretrained Transformer Seen in Table 4 are the hyperparameters for finetuning a pretrained transformer to RNN models. The learning rates, the max number of updates, and the learning period length are all reduced.

architecture	transformer_lm_wiki103
criterion	adaptive_loss
tokens-per-sample	512
sample-break-mode	none
# max tokens	3072
dropout rate	0.2
layer dropout rate	0.2
decoder embed dim	1024
decoder ffn dim	4096
# decoder attn heads	8
optimizer	nag
lr-scheduler	cosine
lr-period-updates	270K
lr-shrink	0.75
t-mult	2
max-lr	1
min-lr	1e-9
lr	1e-4
clip-norm	0.1
warm-up lr	1e-7
# warmup updates	16K
# max updates	286K
# GPUs	8
update-freq	3

Table 3: Language modeling hyperparameters when randomly initialized in the `fairseq` library.

A.1.2 Machine Translation

We experiment with 3 translation benchmarks: WMT14 EN-DE (4.5M train pairs, Bojar et al., 2016), WMT14 EN-FR (36M, Bojar et al., 2014), and WMT17 ZH-EN (20M, Bojar et al., 2017). We follow the preprocessing and data splits by previous work (EN-DE: Vaswani et al., 2017; EN-FR: Gehring et al., 2017; EN-ZH: Hassan et al., 2018; Wu et al., 2019). These datasets are all encoded into subwords by BPE (Sennrich et al., 2016). We run joint BPE on all language pairs except EN-ZH. We use the hyperparameters of the large sized transformer (Vaswani et al., 2017): 6 layers, 16 attention heads, 1024 model dimensions, and 4096 hidden dimensions for both the encoder and decoder. We apply dropout with 0.3, weight decay with 0.01 and label smoothing with $\varepsilon = 0.1$. Following Ott et al. (2018), we use an increased batch size of approximately 460K tokens by accumulating gradients without updating parameters.

Randomly Initialized Training We generally follow the hyperparameters chosen in Vaswani et al. (2017); Ott et al. (2018). Specifically, we list the hyperparameters in Table 5 for easy replication. All other hyperparamter options are left as default values in `fairseq`. The parameters from the last five epochs were averaged to obtain the final model.

Finetuning Pretrained Transformer Seen in Table 6 are the hyperparameters for finetuning a

architecture	transformer_lm_wiki103
criterion	adaptive_loss
tokens-per-sample	512
sample-break-mode	none
# max tokens	3072
dropout rate	0.2
layer dropout rate	0.2
decoder embed dim	1024
decoder ffn dim	4096
# decoder attn heads	8
optimizer	nag
lr-scheduler	cosine
lr-period-updates	135K
lr-shrink	0.75
t-mult	2
max-lr	0.5
min-lr	1e-9
lr	5e-5
clip-norm	0.1
warm-up lr	1e-7
# warmup updates	8K
# max updates	142K
# GPUs	8
update-freq	1

Table 4: Finetuning language modeling hyperparameters in the `fairseq` library. The learning rates are smaller than randomly initialized training.

pretrained transformer to RNN models. The learning rate and the max number of updates are reduced. The parameters from the last five epochs were again averaged to obtain the final model.

A.2 Attention Distribution

Peakiness of Attention Fig. 6 plots the average entropy of the T2R models with and without pretraining. Entropy is averaged across validation samples, layers, and attention heads. Comparing Figs. 4 and 6, we see that there is strong correlation between validation perplexity and entropy. The entropy decreases (and thus the attention distribution gets peakier) when a large feature size is used or the transformer pretraining is applied. This observation hints at potential future improvement of linear transformer models by introducing an inductive bias towards peaky attention distributions.

architecture	transformer_vaswani_en_de_big
criterion	label_smoothed_cross_entropy
label smoothing	0.1
# max tokens	3584
dropout rate	0.3
weight decay	0.0
encoder embed dim	1024
encoder ffn dim	4096
# encoder attn heads	16
# encoder layers	6
decoder embed dim	1024
decoder ffn dim	4096
# decoder attn heads	16
# decoder layers	6
max source positions	1024
max target positions	1024
Adam lrate	5e-4, 3e-4 (T2R)*
Adam β_1	0.9
Adam β_2	0.98
lr-scheduler	inverse square
warm-up lr	1e-7
# warmup updates	4000
# max updates	30K, 60K (EN-FR)
length penalty	0.6
beam size	5
# GPUs	8
update-freq	16

Table 5: Machine translation hyperparameters when randomly initialized in the `fairseq` library. *: we reduced the learning rate for T2R to avoid training divergence.

architecture	transformer_vaswani_en_de_big
criterion	label_smoothed_cross_entropy
label smoothing	0.1
# max tokens	3584
dropout rate	0.3
weight decay	0.0
encoder embed dim	1024
encoder ffn dim	4096
# encoder attn heads	16
# encoder layers	6
decoder embed dim	1024
decoder ffn dim	4096
# decoder attn heads	16
# decoder layers	6
max source positions	1024
max target positions	1024
Adam lrate	2e-4
Adam β_1	0.9
Adam β_2	0.98
lr-scheduler	inverse square
warm-up lr	1e-7
# warmup updates	4000
# max updates	20K, 40K (EN-FR)
length penalty	0.6
beam size	5
# GPUs	8
update-freq	16

Table 6: Finetuning machine translation hyperparameters. The learning rate is smaller than randomly initialized training.
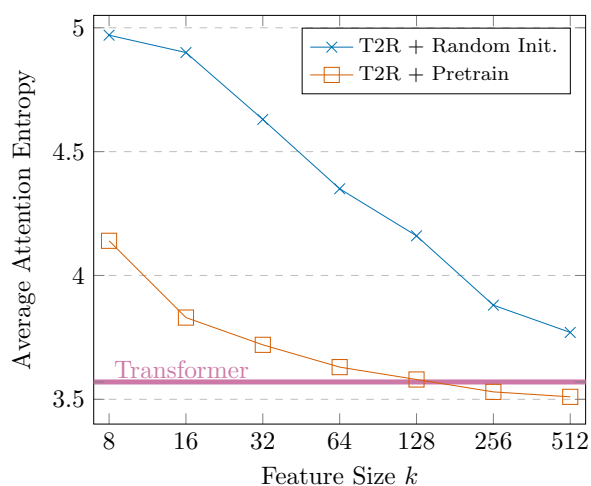

Figure 6: Average entropy of the attention weights. They are computed on the Wikitext-103 validation data for predicting a word given the preceding 512 words.