AN ATTENTION FREE TRANSFORMER

Anonymous authors
Paper under double-blind review

ABSTRACT

We introduce Attention Free Transformer (AFT), an efficient variant of Transformers (Vaswani et al., 2017) that eliminates the need for spatial attention. AFT offers great simplicity compared with standard Transformers, where the multi-head attention operation is replaced with the composition of element-wise multiplications/divisions and global/local pooling. We provide several variants of AFT along with simple yet efficient implementations that are supported by main stream deep learning libraries. We show that, surprisingly, we are able to train AFT effectively on challenging benchmarks, and also to match or surpass the standard Transformer counterparts.

1 INTRODUCTION

Attention mechanisms, represented by Transformers (Vaswani et al., 2017), have driven the advancement of various machine learning problems, including language modeling (Devlin et al., 2018; Radford et al.), image modeling (Chen et al.), and set modeling (Lee et al., 2019). Different from other well known model architectures such as Convolutional Neural Nets (CNNs) or Recurrent Neural Nets (RNNs), Transformers enable direct interaction between every pair of elements within a sequence, which makes them especially powerful at capturing long term dependencies.

However, Transformers require high computational costs. The root cause of this challenge is its need to perform attention operations which have quadratic time and space complexity w.r.t the context size. This makes it especially difficult for Transformers to scale to inputs with large context sizes.

In this paper, we take a bold step towards improving the efficiency of Transformers. We examine the role of multiple heads in Transformers in its extreme form, where the number of heads is equal to the number of dimensions of the query/key. However, a naive implementation is prohibiting, as the memory cost of a multi-head attention operation scales linearly with the number of heads and quadratically with the context size, which easily becomes infeasible for large models.

Crucially, we show that if we replace the softmax nonlinearity in attention with a $relu$, the extreme multi-head attention amounts to a surprisingly efficient formulation. This gives rise to a new family of functions where the key and value are first combined and reduced along the context dimension, the result of which then interacts with the query in an element-wise fashion. We hence call it the Attention Free Transformers (AFT), where we have effectively eliminated the costly attention operation. Moreover, we show that AFT can be efficiently implemented with a dynamic programming algorithm, which operates in-place (ie., an $O(T \times d)$ space complexity) and has $T \times \log T \times d$ time complexity (in the causal mode) instead of $T \times T \times d$.

The main objective of the paper is to replace the softmax with relu, since the relu seperates the attention product in terms of positive and negative products and summation of that leads to element wise multiplication.More on that later

We perform experiments with AFT on several benchmarks, including unconditional image modeling, image super-resolution, language modeling, machine translation and point cloud generation. We show that AFT works very well as an alternative to the standard Transformer, providing competitive results as well as excellent efficiency when context size is long.

2 MULTI-HEAD ATTENTION

At the core of Transformers is the Multi-Head Attention (MHA) operation. Given three sequences, namely the query $Q \in R^{T\times d}$, key $K \in R^{T\times d}$ and value $V \in R^{T\times d}$, and the number of heads h, MHA performs a scaled dot product attention for each head i, defined as:

$$f_i(Q,K,V) = \sigma\Big(\frac{Q_i'(K_i')^T}{\sqrt{d_k}}\Big)V_i', \text{ s.t. } Q_i' = QW_i^Q, K_i' = KW_i^K, V_i' = VW_i^V, \quad (1)$$

where $W_i^Q \in R^{d\times d_k}$, $W_i^K \in R^{d\times d_k}$, $W_i^V \in R^{d\times d_v}$ are linear transformations for head i, and σ is the non-linearity by default set as the $softmax_r$ function (subscript r indicates softmax is applied to each row of a matrix). d_k, d_v are dimensions for key and value, respectively. MHA concatenates the output of h attention heads along the channel dimension, resulting in feature dimension $h \times d_v$. Unless otherwise mentioned, we assume $d_k = d_v$ and $h = \frac{d}{d_k}$. This means the query, key and value are the same dimension within each head, and output dimension matches that of the input.

3 Attention Free Transformer

In practice, given a fixed dimensionality of the output, one often benefits from setting the number of heads to be greater than one. The benefit of doing so is that each head can perform a different aggregation of the context (i.e., the value), thus achieving more flexibility compared to using a single head. We are interested in exploring the limit of number of heads, which amounts to letting $d_k = 1$ for each head. In this case, the dot product operation within each head reduces to a scalar product. However, doing so immediately presents a practical challenge, where the space complexity increases to $O(T \times T \times d)$, where d is potentially a large quantity.

Next, we show that by setting σ to be $relu$ instead of $softmax_r$, the extreme MHA results in a surprisingly simple form. In this case, we have:

$$\begin{aligned} f_i(Q,K,V) &= [Q_i'(K_i')^T]_+V_i' = \big([Q_i']_+[(K_i')^T]_+ + [-Q_i']_+[-(K_i')^T]_+\big)V_i' \\ &= [Q_i']_+\big([K_i']_+^T V_i'\big) + [-Q_i']_+\big([-K_i^T]_+ V_i'\big), \end{aligned} \quad (2)$$

where $[\cdot]_+$ denotes the $relu$ operator, and $Q_i', K_i', V_i' \in R^{T\times 1}$ by definition. In the case when $d_v = 1$ [1], the concatenated output of the attention heads can also be concisely written as:

$$f(Q,K,V) = [Q']_+ \odot \sum_{t=1}^{T}\big([K']_+ \odot V'\big)_t + [-Q']_+ \odot \sum_{t=1}^{T}\big([-K']_+ \odot V'\big)_t \quad (3)$$

Each term has a linear time complexity

where $\odot$ is the element-wise product, with support for broadcasting when the operands' dimensions don't exactly match [2]; $W^Q, W^K, W^V \in R^{d\times d}$ are the combined linear transformation matrices for query, key and value, respectively, and $Q' = QW^Q$, $K' = KW^K$, $V' = VW^V$; A_t denotes the tth row of a matrix A.

Equation 3 is remarkable, as we have shown that what is originally an $O(T^2)$ operation has now become $O(T)$ (wrt both space and time complexity), by only re-arranging the order of computation. In particular, the new operation first combines the key and value by reducing the spatial dimension; the interaction between the query and key simplifies to an element-wise multiplication, which is extremely efficient. By doing so, we have essentially eliminated the spatial attention operation, hence we call it the *Attention Free Transformer* (AFT).

This is not a new technique...

Equation 3 involves the use of two $relu$s, which constitutes a form of sparse interaction between the query and key. For any dimension i, the chance of having a nonzero connectivity between a query point and a context point (key and value) is roughly $p^2 + (1-p)^2$, where p is the average probability of a dimension being positive. Based on this reasoning, it is possible to enforce an even sparser interaction by taking only the first half of Equation 3, resulting in:

$$f(Q,K,V) = [Q']_+ \odot \sum_{t=1}^{T}\big([K']_+ \odot V'\big)_t. \quad (4)$$

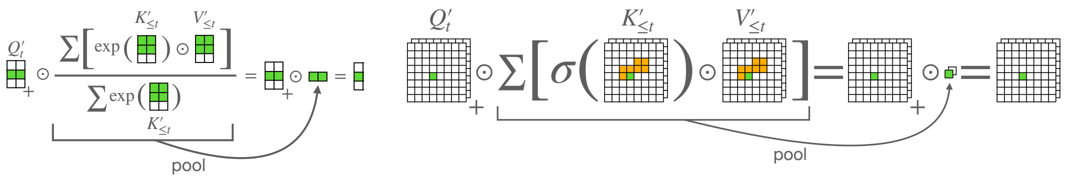
By doing so, the expected probability of query-key interaction along each dimension is reduced to p^2, which can potentially be a small quantity. This simplification also cuts down both time and space

[1]The more general case when $d_v > 1$ can be obtained via broadcasting.

[2]We adopt Numpy styled broadcasting convention: https://numpy.org/doc/stable/user/theory.broadcasting.html

complexity by half, resulting in a more efficient operation. We denote Equation 3 and Equation 4 as AFT-relu2 and AFT-relu, respectively.

Empirically, we have found a better working variant where we replace the $relu$ non-linearity on K' in AFT-relu with $softmax$, resulting in $f(Q,K,V) = [Q']_+ \odot \sum_{t=1}^{T} \left(\text{softmax}_c(K') \odot V'\right)_t$, where $softmax_c$ indicates softmax applied to each column of a matrix. We denote this version AFT-softmax. Compared with AFT-relu, AFT-softmax provides an additional level of non-linearity and allows for a smoother gradient flow on K'. Softmax also provides more numerical stability due to its bounded activation when used in a multiplicative fashion.



(a) Causal attention free operation. (b) Local causal attention free operation.

Figure 1: AFT blocks require only element-wise and pooling operations.

3.1 Causal Attention Free Transformers

One particularly useful variant of MHA is masked attention, oftentimes presented in the form of causal attention. Specifically, in auto-regressive models, queries are constrained to not be able to interact with keys and values beyond the curret position. In standard attention, this is usually implemented with an explicit binary masking matrix of shape $T \times T$, with non-causal entries masked as 0. We show that it is also straightforward to extend AFT to the causal mode while maintaining its efficiency. We use AFT-softmax as the basic AFT version (other variants can be implemented similarly), and denote its output as $Y_t = f(Q_{\le t}, K_{\le t}, V_{\le t}),\ t = 1, ..., T$ [3]. We formulate the casual AFT as:

$$Y_t = [Q'_t]_+ \odot \sum_{t'=1}^{t} \left(\text{softmax}_c(K'_{\le t}) \odot V'_{\le t}\right)_{t'},\ t = 1, ..., T, \tag{5}$$

where the subscript X_t indexes the tth row of matrix X. A naive implementation of Equation 5 amounts to a sequential algorithm, where one iterates each spatial index t. This can be very costly as the context size T increases, preventing efficient usage of parallel computation on modern GPU like hardware. However, we show that there exists an efficient dynamic programming solution that cuts down the number of sequential operations to $O(\log T)$. We show the pseudo code of our algorithm in Algorithm 1 and a diagram of the operations in Figure 1a. We have effectively utilized the fact that the context aggregation part of Equation 5 is amendable to a recursive update rule which breaks the sequential bottleneck of the naive implementation. Moreover, the solution provided in Algorithm 1 can be efficiently implemented as an *in-place* operation [4], which results in a spatial cost of only $O(Td)$.

3.2 Local Causal Attention Free Transformers

In all versions of AFT, the interaction between the query and the context (key and value) happens after the the pooling operation is applied along the spatial dimension. When the context is extremely long, the pooling operation might become the bottleneck and cause difficulty of learning. We hence propose a local variant of AFT, where the context size that each query interacts with is limited. Luckily, doing so for 1d sequences requires only minimum change to the full AFT version, which we also illustrate in Algorithm 1. We denote this version AFT-softmax-local1d.

For data with more sophisticated layouts, e.g., images, AFT-softmax-local1d is not optimal as it does not utilize the 2d layouts of pixels, which is a valuable prior. We then accordingly propose an AFT-softmax-local2d counterpart, shown in Algorithm 2, with a diagram of the operations in

[3]We assume here that Y_t includes input information at the current position t, the version where the current position is excluded can be obtained by shifting the outputs to the right.

[4]We implemented it in Pytorch.

Figure 1b. Again, our local2d version enjoys exactly the same time complexity and the benefit of in-place operations. All the operations involved are basic tensor manipulations supported by all modern deep learning libraries. This is in drastic contrast to alternative approaches such as (Child et al., 2019; Parmar et al., 2018) which typically involves non-trivial implementations.

Algorithm 1: Pseudo code of an efficient, in-place causal AFT-softmax/AFT-softmax-local1d.

```
Input: Query, key and value $Q', K', V' \in R^{T \times d}$; optionally context size $s \in \{2^n, n \in N\}$.
Output: Causal AFT output $Y \in R^{T \times d}$.
1  $KK = \exp(K')$ // new memory allocation
2  $KV = \exp(K') * V'$ // new memory allocation
   // if s is not provided default to $\lceil \log_2(T) \rceil$ iterations
3  for $j = 1, ..., \min(\lceil \log_2(T) \rceil, \log_2(s))$ do
4      stride = $2^{j-1}$
5      KV[stride:, :] = KV[stride:, :] + KV[:T-stride, :] // in-place op
       // now $KV[i] = \sum_{k=max(0,i-2^j+1)}^{i} (\exp(K') * V')[k], \forall i$
6      KK[stride:, :] = KK[stride:, :] + KK[:T-stride, :] // in-place op
       // now $KK[i] = \sum_{k=max(0,i-2^j+1)}^{i} (\exp(K'))[k], \forall i$
   // normalize according to $softmax_c$ and multiply with query
7  $Y = relu(Q') * KV/KK$
```

Algorithm 2: Pseudo code of an efficient, in-place causal AFT-softmax-local2d.

```
Input: Query, key and value $Q', K', V' \in R^{H \times W \times d}$, context sizes $s_h, s_w \in \{2^n, n \in N^+\}$
Output: Causal AFT output $Y \in R^{H \times W \times d}$.
1  $KK = \exp(K')$ // new memory allocation
2  $KV = \exp(K') * V'$ // new memory allocation
   // first aggregate locally across rows; pass if $s_w \leq 2$.
3  for $j = 1, ..., \log_2(s_w) - 1$ do
4      stride = $2^{j-1}$
5      KV[:, stride:, :] = KV[:, stride:, :] + KV[:, :W-stride, :] // in-place op
       // now $KV[:, i] = \sum_{k=max(0,i-2^j+1)}^{i} (\exp(K') * V')[:, k], \forall i$
6      KK[:, stride:, :] = KK[:, stride:, :] + KK[:, :W-stride, :] // in-place op
       // now $KK[:, i] = \sum_{k=max(0,i-2^j+1)}^{i} (\exp(K'))[:, k], \forall i$
   // then aggregate locally across columns
7  for $j = 1, ..., \log_2(s_h)$ do
8      stride = $2^{j-1}$
9      KV[stride:, :, :] = KV[stride:, :, :] + KV[:H-stride, :, :] // in-place op
       // now
           $KV[i_h, i_w] = \sum_{k_h=max(0,i_h-2^j+1)}^{i_h} \sum_{k_w=max(0,i_w-\frac{s_w}{2}+1)}^{i_w} (\exp(K') * V')[k_h, k_w], \forall i_h, i_w$
10     KK[stride:, :, :] = KK[stride:, :, :] + KK[:H-stride, :, :] // in-place op
       // now $KK[i_h, i_w] = \sum_{k_h=max(0,i_h-2^j+1)}^{i_h} \sum_{k_w=max(0,i_w-\frac{s_w}{2}+1)}^{i_w} (\exp(K'))[k_h, k_w], \forall i_h, i_w$
   // incorporate contexts to the right
11 idx = min(arange(W) + $\frac{s_w}{2}$, W-1) // arange(W) = [0, 1, ..., W-1]
12 KV[1:, :, :] = KV[1:, :, :] + KV[:H-1, idx, :] // in-place op
13 KK[1:, :, :] = KK[1:, :, :] + KK[:H-1, idx, :] // in-place op
   // normalize according to $softmax_c$ and multiply with query
14 $Y = relu(Q') * KV/KK$
```

4 RELATED WORK

Since the Transformer was introduced, there have been numerous attempts to address the major source of inefficiency in the architecture, the quadratic cost of the attention operation. Improving this operation can enable larger context sizes and more efficient implementations. For a comprehensive, recent survey of efficient transformers, see (Tay et al., 2020c).

Some approaches focus on increased efficiency through restricting the number of keys against which the query is compared. Sparse Transformers (Child et al., 2019) use fixed context patterns derived from analysis of fully-trained attention heads. The Reformer (Kitaev et al., 2020) uses approximate-nearest-neighbor search and locality-sensitive hashing to select relevant keys. Attention models in vision tasks use image structure to help handcraft relevant spatial patterns to attend (Wang et al., 2020a; Huang et al., 2019b; Zhu et al., 2019; Huang et al., 2019a; Ramachandran et al., 2019).

Other approaches try to learn these patterns. Adaptive-Span Transformers (Sukhbaatar et al., 2019) learn a range for each attention head within which to attend. Routing transformers (Roy et al., 2020) use clustering to compute dot-product attention only over a subset of elements within the same cluster. The Linformer (Wang et al., 2020b) reduces the length of the context by compressing the keys and values with a linear layer. The Sinkhorn Transformer (Tay et al., 2020b) uses a differentiable sorting operation to identify relevant comparisons that may not be local in the original sequence order. Compressive Transformers (Rae et al., 2020) compute and update reduced representations of the input that are far enough back in the input sequence, and attend to those compressed representations.

Instead of limiting the number of comparisons, other methods change the operation used to compute attention. Efficient Attention (Britz et al., 2017) notes that changing the order of matrix multiplications and the softmax nonlinearity results in a matrix quadratic on the feature dimension of the inputs rather than on the length of the context. The Synthesizer (Tay et al., 2020a) uses attention weights predicted from inputs, rather than derived from dot-product interactions.

AFT differs from the aforementioned approaches fundamentally, as we have addressed the efficiency issue by replacing attention with a new operation, rather than using an approximation. AFT also enjoys great simplicity, which makes it to be readily adopted as a plug in alternative to Transformers.

5 Experiments

We conduct experiments on five tasks: unconditional image modeling (Sec. 5.1), image super resolution (Sec. 5.2), language modeling (Sec. 5.3), machine translation (Sec. 5.4) and point cloud generation (Sec. A.4). We focus on the causal mode of AFT, while leaving systematic evaluation of the non-causal version as future work. Unless otherwise mentioned, all experiments are conducted on 8$\times$V100 GPU machines.

5.1 Unconditional Image Modeling

In our first set of experiments, we consider the problem of image modeling by minimizing the negative log likelihood (NLL). Similar to (Parmar et al., 2018), we represent an RGB image as a sequence of length $H \times W \times 3$, with H, W being the height and width, respectively. Each sub-pixel is represented as a 256-way discrete variable. We use CIFAR10 for image density modeling.

5.1.1 AFT-relu2 vs AFT-relu vs AFT-softmax

We first conduct experiments validating the legitimacy of AFT and its three nonlinearity variants. Our reference Transformer design largely follows that of (Chen et al.), where a transformer block consists of a MHA layer with residual connection and a 2 layer MLP with residual connections. Layer Normalization (LN) (Ba et al., 2016) is applied in a "pre-act" fashion. We adopt learned position embeddings, and use a set of shared token embeddings and prediction heads across RGB.

Our base architecture consists of 24 Transformer blocks, each with d=256 dimensional features. The hidden layer of the MLP per block has $4\times$ dimensionality of its input. We use Adam, and follow a standard warmup learning rate schedule as in (Vaswani et al., 2017). We use an initial learning rate of 3×10^{-3} and a weight decay of 0.1 applied to all linear transformations weights, and no dropout.

We adopt simple data augmentation. During training, we first randomly flip each image horizontally, then add or subtract a value in the range $[-10, 10]$ from all its subpixels, and clip resulting pixel values to $[0, 255]$. We use cross entropy loss, and a default batch size of 128 for 200 training epochs. We train three versions of AFT, namely AFT-relu2, AFT-relu, AFT-softmax, all of which use full contexts. We show the training and test loss curves in Figure 2. All versions of AFT are trainable with standard optimization techniques for Transformers. In particular, AFT-relu2 performs slightly worse than AFT-relu, and both are significantly worse than AFT-softmax. Based on this observation, we use AFT-softmax as the default setting for all remaining experiments.

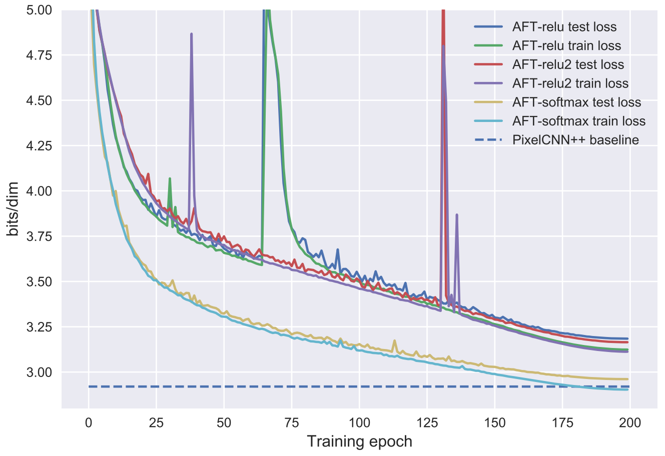


Figure 2: Proof of concept experiments for AFT-relu2, AFT-relu and AFT-softmax, tested on CIFAR10. All three versions train well with standard optimization settings. AFT-relu2 and AFT-relu perform similarly, while AFT-softmax is more stable and yields significantly better results.

Table 1: Training and test loss for AFT-softmax-local2d variants with different context sizes. We see a U-shaped distribution of both losses as context size increases.

Context size	(4,4)	(4,8)	(8,8)	(8,16)	(16, 16)	(16, 32)	full
Training loss	2.948	2.844	2.826	**2.811**	2.821	2.853	2.903
Test loss	3.017	2.901	2.888	**2.871**	2.880	2.912	2.961

5.1.2 AFT-LOCAL2D

Another important variant of AFT is the local2d version. In order to show its impact on the model's performance, we conducted further experiments with the same base architecture. The only change we make is to replace AFT-softmax with its local version. In particular, we reshape each image of size $(H, W, 3)$ to a 2d matrix with H rows and $3W$ columns, and apply a designated local context size shared across all AFT blocks. We search over context sizes in the range $\{(4,4),(4,8),(8,8),(8,16),(16,16),(16,32)\}$ and show the results in Table 1.

Finally, we trained a set of deeper models for 500 epochs, and report the final results in Table 3. We see a clear gain of increasing depth, and our best model is comparable with the state-of-the-art result on CIFAR10, which is set by a very deep Sparse Transformer.

5.1.3 TIME AND MEMORY EFFICIENCY

In order to evaluate the efficiency of AFT, we have conducted an additional set of experiments by varying the sequence length, model depth and width. We use a fixed batch size (128) and measure each configuration's training speed and memory usage. We show the results in Table 2. We see that AFT provides significant memory efficiency, especially for long sequences. Also, the memory footprint of AFT is roughly linear w.r.t. T, Layers (L) and d. E.g., doubling the depth and halving the width results in identical memory usage, thanks to the in-place operations. Local1d and Local2d variants both provide effective speedup compared to the full AFT version. There is a trend towards improved time efficiency for AFT over the baseline Transformer as sequence length grows, while the baseline exceeds GPU memory at the largest sequence length, preventing further comparison.

5.2 IMAGE SUPER RESOLUTION

We also consider a super-resolution task based on pixel-wise image generation. Following (Dahl et al., 2017; Parmar et al., 2018), we enlarge an 8×8 sized image to 32×32.

We use CelebA dataset (Liu et al., 2015) as the benchmark. Our baseline model is the Image Transformer (Parmar et al., 2018) with its encoder and decoder connected through the attention mech-

Table 2: Training speed and memory consumption of different model configurations, measured in iterations/second (IPS) and GB/GPU (GPG), respectively. Default batch size is set at 128, tested on an 8xV100 GPU machine. Both local2d and local1d variants apply a window size of 128 (8×16). For local2d, the input sequence is reshaped to a 2d matrix with 32 rows. OOM denotes out of memory error. Each entry shows result in the IPS/GPG format.

	L=12, d=512, heads=8 (for Transformer)				L=24, d=256, heads=8 (for Transformer)			
		AFT-softmax				AFT-softmax		
T	Transformer	full	local2d	local1d	Transformer	full	local2d	local1d
3072	- / OOM	0.45 / 26	0.52 / 26	0.55 / 26	- / OOM	0.52 / 25	0.62 / 25	0.67 / 25
1024	1.43 / 22	1.36 / 10	1.48 / 10	1.55 / 10	- / OOM	1.47 / 10	1.60 / 10	1.70 / 10
512	3.20 / 9	2.47 / 6	2.69 / 6	2.74 / 6	2.61/12	2.55 / 6	2.62 / 6	2.75 / 6

Table 3: NLL results on CIFAR10, evaluated by bits/dim, the lower the better.

Method	train loss	test loss	#params
PixelCNN	3.08	3.14	
PixelCNN++	-	2.92	
PixelSNAIL	-	2.85	
Image Transformer L=12, d=512	-	2.90	40M
Sparse Transformer L=128, d=256	-	**2.80**	59M
AFT-local2d L=24, d=256 (Ours)	2.76	2.85	20M
AFT-local2d L=28, d=256 (Ours)	2.75	2.84	23M
AFT-local2d L=35, d=256 (Ours)	**2.70**	**2.81**	29M

anism. Both the 1D and 2D local Image Transformer models have $L = 12$ layers, $d = 512$ and attention heads=8, and are trained under the DMOL (discretized mixture of logistics) loss for 200 epochs. We experimented by replacing the standard attention blocks in decoder with our AFT-local1d or -local2d, keeping other modules the same. We follow similar training schemes, but with tuned dropout and learning rate. Evaluation is performed in terms of NLL measured in bits/dim, with the sampling temperature fixed at 0.8.

Table 4 shows results of our best AFT-local2d model (context size 16×16), in comparison to the PixelRecursive baseline and 1D/2D local Image Transformer models. Note the large consumption of model capacity and memory from 2D local Image Transformer, while our AFT-local2d shows clear advantages with no loss in model quality (see Figure 3 for visual comparison).

5.3 Language Modeling

We apply AFT to language modeling on WikiText-103 (Merity et al., 2016). This dataset consists of 103M word tokens extracted from Wikipedia articles. Given a word sequence, the task is to predict the next word. Standard transformer-based neural language models use multiple transformer layers, each with causal attention. In this set of experiments, we replace the transformer modules with AFT-softmax (the full version), keeping the rest of the model design the same. Our baseline is the architecture proposed by Baevski & Auli (2019) which has 16 1024-d Transformer layers and uses a maximum context of 3072 words. It achieves close to state-of-the-art results and was easily reproduced using the fairseq library (Ott et al., 2019). Training the model to convergence takes a long time, and in order to facilitate our exploration and analysis, we compare results when all models are trained with 70K updates. As shown in Table 5, replacing transformer modules with the same number of AFT modules increases perplexity. However, this loss in performance can be made up for by making the models deeper. Moreover, the models can be made narrower to reduce the computational cost without significantly sacrificing performance. In fact, all AFT models trained achieved lower training perplexity than the baseline, indicating that AFT has no problem fitting the long sequence. AFT-based transformers are much more memory-efficient as seen by the maximum batch-size that we could accommodate for each of the models on a 32GB GPU. The baseline allows for only a batch size of 1 per GPU, but our models which achieve comparable results can accommodate a batch size of up to 3, indicating a much smaller memory footprint. We also report the number of training updates per minute for a batch-size of 32. Our deep and narrow AFT-based models can be updated at a similar rate as the baseline.

Table 4: Image super resolution results on CelebA. Our AFT models outperform the PixelRecursive baseline (Dahl et al., 2017) in bits/dim (the lower the better), and show clear advantages in parameter efficiency and memory saving over Image Transformers (Parmar et al., 2018), with comparable or even better performance.

Method	L	d	Train bits/dim	Dev bits/dim	Params	Iters /sec	GB /GPU
PixelRecursive	-	-	-	2.81	54M	-	-
1D local Transformer	12	512	2.54	2.68	39M	2.98	9.5G
2D local Transformer	12	512	2.43	2.61	42M	1.45	21.2G
AFT-local2d (Ours)	32	256	2.39	2.59	25M	1.43	10.4G

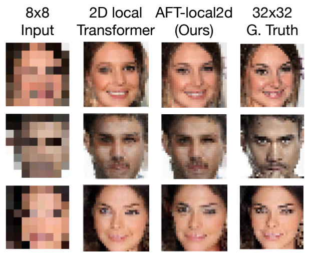


Figure 3: Upscaled images from baseline and our 2D local transformers on CelebA.

Table 5: Language Modeling Results on WikiText-103 with the Baseline from Baevski & Auli (2019)

Method	L	d	Train PPL	Val PPL	Test PPL	#params	Iters per min	Largest b-size/GPU
Baseline (at convergence)	16	1024	16.21	17.97	18.70	247M	22.2	1
Baseline (at 70K updates)	16	1024	26.21	22.80	23.63	247M	22.2	1
AFT (Ours, 70K)	16	1024	20.55	25.27	25.93	247M	29.3	4
AFT (Ours, 70K)	32	1024	21.35	23.78	24.11	448M	14.5	2
AFT (Ours, 70K)	48	1024	21.34	23.55	24.07	650M	9.3	1
AFT (Ours, 70K)	32	724	22.66	24.12	24.64	233M	23.8	3
AFT (Ours, 70K)	48	512	24.38	23.79	24.46	173M	25.9	3

We also experiment with a different baseline architecture using a shorter 512 word context (see supplementary material). In that setting as well, AFT matches baseline performance.

5.4 Machine Translation

As a machine translation benchmark, we show experiments with the WMT 2014 English to German translation task. The training set contains approximately 4.5 million sentence pairs. We compare against a Transformer architecture baseline using the OpenNMT implementation (Klein et al., 2017). For translation, the standard architecture is an encoder-decoder structure, where the encoder uses non-causal attention to encode the input sentence. The decoder uses two different types of attention. The first, self attention, sequentially attends to the output translation as it is being generated token by token. The second attends to the translation and the context from the encoder.

In our experiments, we replace the multi-headed decoder self-attention blocks. We compare perplexity (PPL), BLEU score, and efficiency between the Transformer base and AFT in Table 6. In this task, we see that AFT performs on par with the Transformer. As expected for the small context size, typically around 50 tokens, AFT does not show improvements in speed or memory.

Table 6: WMT 2014 English-to-German Translation.

Method	Training PPL	Validation PPL	Test BLEU score	tokens/sec	GB/GPU
Transformer	4.38	4.06	27.32	56.6K	7.80
AFT (Ours)	4.34	4.10	27.70	51.1K	8.04

6 Conclusions

We have introduced the Attention Free Transformer that replaces attention with an efficient, easy-to-implement new operation. We have demonstrated strong results on challenging benchmarks, despite of the simplicity of our design. We believe that our model opens a new design space for Transformer-like models, and will see impact in various areas where Transformers are applied.