Generating Long Sequences with Sparse Transformers

Rewon Child [1] **Scott Gray** [1] **Alec Radford** [1] **Ilya Sutskever** [1]

Abstract

Transformers are powerful sequence models, but require time and memory that grows quadratically with the sequence length. In this paper we introduce sparse factorizations of the attention matrix which reduce this to $O(n\sqrt{n})$. We also introduce a) a variation on architecture and initialization to train deeper networks, b) the recomputation of attention matrices to save memory, and c) fast attention kernels for training. We call networks with these changes Sparse Transformers, and show they can model sequences tens of thousands of timesteps long using hundreds of layers. We use the same architecture to model images, audio, and text from raw bytes, setting a new state of the art for density modeling of Enwik8, CIFAR-10, and ImageNet-64. We generate unconditional samples that demonstrate global coherence and great diversity, and show it is possible in principle to use self-attention to model sequences of length one million or more.

1. Introduction

Estimating complex, high-dimensional data distributions is a central problem in unsupervised learning, as many downstream applications of interest involve generation of text, images, audio, and other data. Additionally, it is believed to be a key component of unsupervised representation learning.

Recently, neural autoregressive models have achieved impressive results in this domain, achieving state-of-the-art in modeling natural language (Jozefowicz et al., 2016) (Radford et al., 2018) (Dai et al., 2018), raw audio (Van Den Oord et al., 2016) (Mehri et al., 2016), and images (Oord et al., 2016) (Menick & Kalchbrenner, 2018) (Salimans et al., 2017) (Reed et al., 2017) (Chen et al., 2017).

These methods decompose a joint probability distribution into a product of conditional ones. Modeling these conditional distributions is extremely challenging, however, as they contain many complex, long-range dependencies and require a suitably expressive model architecture to learn them.

Figure 1. Unconditional samples from our neural autoregressive model on ImageNet 64 and a classical music dataset. We used the same self-attention based architecture for audio, images, and text. The samples above were generated with softmax temperature 1.0, and had lengths 12,288 and 65,536. Audio samples be listened to at `https://bit.ly/2DnJx34`

Architectures based off CNNs (Oord et al., 2016) have made great progress in this direction, but require significant depth to expand their receptive field. To address this, WaveNet (Van Den Oord et al., 2016) introduced dilated convolutions, which allowed the network to model long-range dependencies in a logarithmic number of layers.

Separately, the Transformer (Vaswani et al., 2017) has been shown to excel on many natural language tasks, which may

[1]OpenAI, San Francisco, California, United States. Correspondence to: Rewon Child <rewon@openai.com>.

be in part due to its ability to model arbitrary dependencies in a constant number of layers. As each self-attention layer has a global receptive field, the network can allocate representational capacity to the input regions for which it is most useful. Thus the architecture may be more flexible at generating diverse data types than networks with fixed connectivity patterns.

However, the memory and computational requirements of such networks grows quadratically with sequence length, which excludes their use on long sequences.

The main contribution of this work is to introduce several sparse factorizations of the attention matrix, which scale as $O(n\sqrt[p]{n})$ with the sequence length without sacrificing performance. These work by separating the full attention computation into several faster attention operations which, when combined, can approximate the dense attention operation. We use this to apply self-attention to sequences of unprecedented length.

Additionally, we introduce several other changes to the Transformer, including:

- A restructured residual block and weight initialization to improve training of very deep networks
- A set of sparse attention kernels which efficiently compute subsets of the attention matrix
- Recomputation of attention weights during the backwards pass to reduce memory usage

We empirically validate that models augmented in this manner can achieve state-of-the-art compression and generation of natural language, raw audio, and natural images. The simplicity of the architecture leads us to believe it may be useful for many problems of interest.

2. Related Work

The most related work involves other techniques for scaling up autoregressive generative models. For images, (Reed et al., 2017) models conditional independence between the pixels in order to generate many locations in parallel, and (Menick & Kalchbrenner, 2018) imposes an ordering and multi-scale upsampling procedure to generate high fidelity samples. (Parmar et al., 2018) uses blocks of local attention to apply Transformers to images. For text, (Dai et al., 2018) introduces a state reuse "memory" for modeling long-term dependencies. And for audio, in addition to (Van Den Oord et al., 2016), (Mehri et al., 2016) used a hierarchical structure and RNNs of varying clock-rates to use long contexts during inference, similar to (Koutnik et al., 2014).

Our work is simpler than many of the techniques above and can be applied equally across images, text, and audio. Many of the above techniques are orthogonal to ours, moreover, and could be used in conjunction with ours.

Outside of generative modeling, there are several works relevant to improving the efficiency of attention based off chunking (Chiu & Raffel, 2017) or using fixed length representations (Britz et al., 2017). Other works have investigated attention with multiple "hops", such as (Sukhbaatar et al., 2015) and (Gehring et al., 2017).

It is worth noting that the Gated Pixel CNN (Oord et al., 2016) and WaveNet (Van Den Oord et al., 2016) use multiplicative interactions in their networks, which are related to self-attention.

3. Background

We consider the task of autoregressive sequence generation, where the joint probability of a sequence $\mathbf{x} = \{x_1, x_2, ..., x_n\}$ is modeled as the product of conditional probability distributions and parameterized by a network θ.

$$p(\mathbf{x}) = \prod_{i=1}^{n} p(x_i | x_1, ..., x_{i-1}; \theta) \tag{1}$$

We treat images, text, and audio as a sequence of discrete tokens, typically raw bytes. The network θ takes in the sequence of tokens and outputs a categorical distribution over the v possible values of the next token using the $\mathrm{softmax}$ function, where v is the size of the *vocabulary*. The training objective is to maximize the log-probability of the data with respect to θ.

A simple and powerful choice for model θ is a Transformer (Vaswani et al., 2017) in decoder-only mode, as demonstrated by (Radford et al., 2018) and (Liu et al., 2018). These models transform the input sequence with blocks of multihead self-attention over the entire sequence, followed by dense transformations over each sequence element. The self-attention portion of the network must compute n weightings for each of n elements, however, which can quickly become intractable as the sequence length grows.

In the following sections, we describe our modifications to the Transformer architecture which make it more suitable for modeling long sequences.

4. Factorized Self-Attention

Sparse Transformers separate the full self-attention operation across several steps of attention, as visualized in Figure 3(b) and 3(c). To motivate our approach, we first perform a qualitative assessment of attention patterns learned by a standard Transformer on an image dataset.

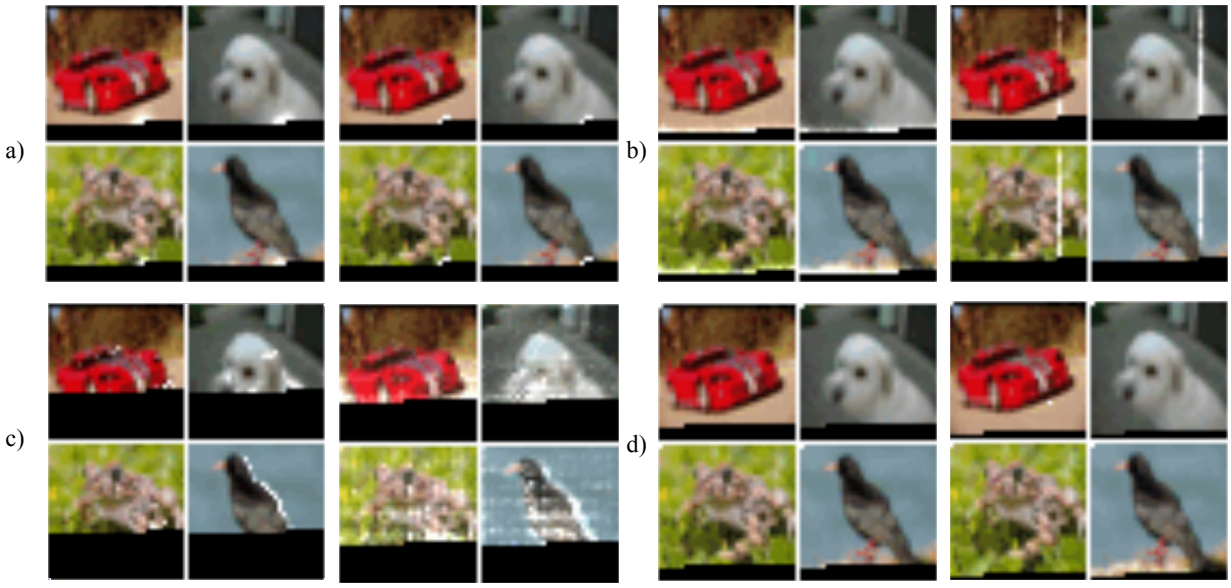
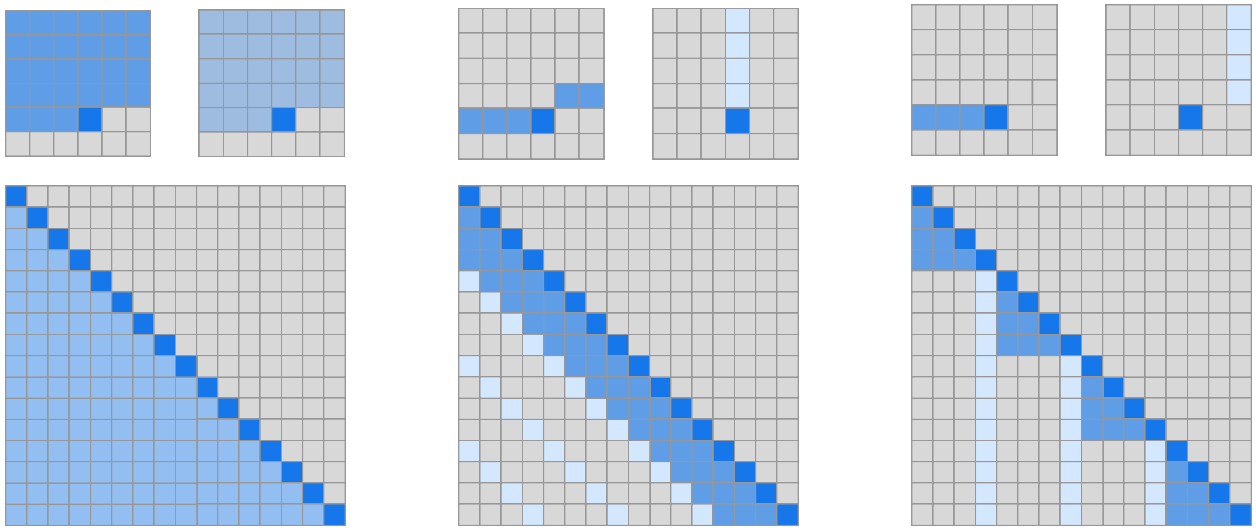


Figure 2. Learned attention patterns from a 128-layer network on CIFAR-10 trained with full attention. White highlights denote attention weights for a head while generating a given pixel, and black denotes the autoregressive mask. Layers are able to learn a variety of specialized sparse structures, which may explain their ability to adapt to different domains. a) Many early layers in the network learn locally connected patterns, which resemble convolution. b) In layers 19 and 20, the network learned to split the attention across a row attention and column attention, effectively factorizing the global attention calculation. c) Several attention layers showed global, data-dependent access patterns. d) Typical layers in layers 64-128 exhibited high sparsity, with positions activating rarely and only for specific input patterns.



(a) Transformer (b) Sparse Transformer (strided) (c) Sparse Transformer (fixed)

Figure 3. Two 2d factorized attention schemes we evaluated in comparison to the full attention of a standard Transformer (a). The top row indicates, for an example 6x6 image, which positions two attention heads receive as input when computing a given output. The bottom row shows the connectivity matrix (not to scale) between all such outputs (rows) and inputs (columns). Sparsity in the connectivity matrix can lead to significantly faster computation. In (b) and (c), full connectivity between elements is preserved when the two heads are computed sequentially. We tested whether such factorizations could match in performance the rich connectivity patterns of Figure 2.

4.1. Qualitative assessment of learned attention patterns

We visualized the attention patterns learned by a 128-layer self-attention network on CIFAR-10, and present several examples in Figure 2. Visual inspection showed that most layers had sparse attention patterns across most data points, suggesting that some form of sparsity could be introduced without significantly affecting performance. Several layers (Figure 2c) clearly exhibited global patterns, however, and others exhibited data-dependent sparsity (Figure 2d), both of which would be impacted by introducing a predetermined sparsity pattern into all of the attention matrices.

In this paper, we restricted our investigation to a class of sparse attention patterns that have connectivity between all positions over several steps of attention. These methods can be more efficient than full attention while still providing global context to any given position. We aimed to empirically validate the performance of these factorized patterns on a range of tasks, given that they are unable to learn the exact same mappings as those in Figure 2. We present the formulation of factorized attention below.

4.2. Factorized self-attention

A self-attention layer maps a matrix of input embeddings X to an output matrix and is parameterized by a connectivity pattern $S = \{S_1, ..., S_n\}$, where S_i denotes the set of indices of the input vectors to which the ith output vector attends. The output vector is a weighted sum of transformations of the input vectors:

$$\text{Attend}(X, S) = \left(a(\mathbf{x}_i, S_i) \right)_{i \in \{1,...,n\}} \tag{2}$$

$$a(\mathbf{x}_i, S_i) = \text{softmax}\left(\frac{(W_q \mathbf{x}_i) K_{S_i}^T}{\sqrt{d}} \right) V_{S_i} \tag{3}$$

$$K_{S_i} = \left(W_k \mathbf{x}_j \right)_{j \in S_i} \quad V_{S_i} = \left(W_v \mathbf{x}_j \right)_{j \in S_i} \tag{4}$$

Here W_q, W_k, and W_v represent the weight matrices which transform a given $\mathbf{x}_i$ into a *query*, *key*, or *value*, and d is the inner dimension of the queries and keys. The output at each position is a sum of the values weighted by the scaled dot-product similarity of the keys and queries.

Full self-attention for autoregressive models defines $S_i = \{j : j \leq i\}$, allowing every element to attend to all previous positions and its own position.

Factorized self-attention instead has p separate attention heads, where the mth head defines a subset of the indices $A_i^{(m)} \subset \{j : j \leq i\}$ and lets $S_i = A_i^{(m)}$. We are chiefly interested in *efficient* choices for the subset A, where $|A_i^{(m)}| \propto \sqrt[p]{n}$.

Additionally, for the time being we consider *valid* choices of A, where all input positions are connected to all future output positions across the p steps of attention.

For every $j \leq i$ pair, we set every A such that i can attend to j through a path of locations with maximum length $p + 1$. Specifically, if $(j, a, b, c, ..., i)$ is the path of indices, then $j \in A_a^{(1)}$, $a \in A_b^{(2)}$, $b \in A_c^{(3)}$, and so forth.

These two criteria allow us keep the ability of Transformers to propagate signals from arbitrary input positions to arbitrary output positions in a constant number of steps, while reducing the total effective computation to $O(n \sqrt[p]{n})$. We also note that softening the validity criterion (for instance, having a series of only locally connected layers) may be a useful inductive bias for certain domains.

In this work, we explore two factorizations for $p = 2$, which we describe in the following section, though we note that the same techniques can be easily extended to higher dimensions.

4.3. Two-dimensional factorized attention

A natural approach to defining a factorized attention pattern in two dimensions is to have one head attend to the previous l locations, and the other head attend to every lth location, where l is the *stride* and chosen to be close to $\sqrt{n}$, a method we call *strided* attention.

Formally, $A_i^{(1)} = \{t, t+1, ..., i\}$ for $t = \max(0, i - l)$ and $A_i^{(2)} = \{j : (i - j) \bmod l = 0\}$. This pattern can be visualized in Figure 3(b).

This formulation is convenient if the data naturally has a structure that aligns with the stride, like images or some types of music. For data without a periodic structure, like text, however, we find that the network can fail to properly route information with the strided pattern, as spatial coordinates for an element do not necessarily correlate with the positions where the element may be most relevant in the future.

In those cases, we instead use a *fixed* attention pattern (Figure 3(c)), where specific cells summarize previous locations and propagate that information to all future cells.

Formally, $A_i^{(1)} = \{j : (\lfloor j/l \rfloor = \lfloor i/l \rfloor)\}$, where the brackets denote the floor operation, and $A_i^{(2)} = \{j : j \bmod l \in \{t, t+1, ..., l\}$, where $t = l - c$ and c is a hyperparameter.

Concretely, if the stride is 128 and $c = 8$, then all future positions greater than 128 can attend to positions 120-128, all positions greater than 256 can attend to 248-256, and so forth.

A fixed-attention pattern with $c = 1$ limits the expressivity of the network significantly, as many representations in

the network are only used for one block whereas a small number of locations are used by all blocks. We instead found choosing $c \in \{8, 16, 32\}$ for typical values of $l \in \{128, 256\}$ to perform well, although it should be noted that this increases the computational cost of this method by c in comparison to the strided attention.

Additionally, we found that when using multiple heads, having them attend to distinct subblocks of length c within the block of size l was preferable to having them attend to the same subblock.

In the subsequent section, we describe how to incorporate factorized attention into the Sparse Transformer architecture.

5. Sparse Transformer

Here we fully describe the Sparse Transformer architecture, which is a modified version of the Transformer (Vaswani et al., 2017).

5.1. Factorized attention heads

Standard dense attention simply performs a linear transformation of the attend function defined in Equation 2:

$$\text{attention}(X) = W_p \cdot \text{attend}(X, S) \tag{5}$$

where W_p denotes the post-attention weight matrix. The simplest technique for integrating factorized self-attention is to use one attention type per residual block, and interleave them sequentially or at a ratio determined as a hyperparameter:

$$\text{attention}(X) = W_p \cdot \text{attend}(X, A^{(r \bmod p)}) \tag{6}$$

Here r is the index of the current residual block and p is the number of factorized attention heads.

A second approach is to have a single head attend to the locations of the pixels that both factorized heads would attend to, which we call a *merged* head:

$$\text{attention}(X) = W_p \cdot \text{attend}(X, \bigcup_{m=1}^{p} A^{(m)}) \tag{7}$$

This is slightly more computationally intensive, but only by a constant factor. A third approach is to use multi-head attention (Vaswani et al., 2017), where n_h attention products are computed in parallel, then concatenated along the feature dimension:

$$\text{attention}(X) = W_p \Big(\text{attend}(X, A)^{(i)} \Big)_{i \in \{1, \ldots, n_h\}} \tag{8}$$

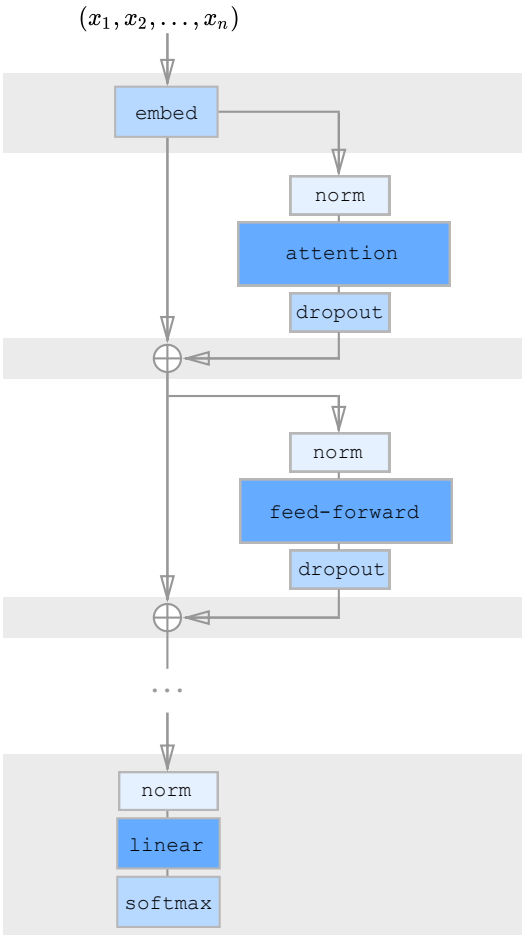


Figure 4. Diagram depicting one residual block of the Sparse Transformer. The shaded background indicates tensors which are *checkpointed* (Chen et al., 2016) and stored in GPU memory. The other tensors, including the attention weights and feedforward network activations, are recomputed during the calculation of gradients, reducing memory usage substantially.

Here, the A can be the separate attention patterns, the merged patterns, or interleaved as in Eq. 2. Also, the dimensions of the weight matrices inside the attend function are reduced by a factor of $1/n_h$, such that the number of parameters are invariant across values of n_h.

We typically find multiple heads to work well, though for extremely long sequences where the attention dominates the computation time, it is more worthwhile to perform them one at a time and sequentially.

5.2. Scaling to hundreds of layers

We found that Transformers were difficult to train with many layers, as noted by (Al-Rfou et al., 2018). Instead of incorporating auxillary losses, we adopted the following

architectural changes.

First, we use the pre-activation residual block of (He et al., 2016), defining a network of N layers in the following way:

$$H_0 = \text{embed}(X, W_e) \tag{9}$$

$$H_k = H_{k-1} + \text{resblock}(H_{k-1}) \tag{10}$$

$$y = \text{softmax}(\text{norm}(H_N)W_{out}) \tag{11}$$

where embed is a function we describe in the next section, W_{out} is a weight matrix, and $\text{resblock}(h)$ normalizes the input to the attention block and a positionwise feedforward network in the following way:

$$a(H) = \text{dropout}(\text{attention}(\text{norm}(H))) \tag{12}$$

$$b(H) = \text{dropout}(\text{ff}(\text{norm}(H + a(H)))) \tag{13}$$

$$\text{resblock}(H) = a(H) + b(H) \tag{14}$$

The norm function denotes Layer Normalization (Ba et al., 2016), and $\text{ff}(x) = W_2\, f(W_1 x + b_1) + b_2$. Our choice of f is the Gaussian Error Linear Unit (Hendrycks & Gimpel, 2016), $f(X) = X \odot \text{sigmoid}(1.702 \cdot X)$, as used in (Radford et al., 2018). The output dimension of W_1 is 4.0 times the input dimension, unless otherwise noted.

Observe that H_N is the sum of N applications of functions a and b, and thus each function block receives a gradient directly from the output layer . We scale the initialization of W_2 and W_p in Eq. 5 by $\frac{1}{\sqrt{2N}}$ to keep the ratio of input embedding scale to residual block scale invariant across values of N.

5.3. Modeling diverse data types

In addition to the embedding of input symbols, positional embeddings are typically used in Transformers and other location-agnostic architectures to encode the spatial relationships of data (Gehring et al., 2017), (Parmar et al., 2018).

We found using learned embeddings which either encoded the structure of the data or the factorized attention patterns were important for performance of our models.

We added either $n_{emb} = d_{data}$ or $n_{emb} = d_{attn}$ embeddings to each input location, where d_{data} refers to the number of dimensions of the data, and d_{attn} is the number of dimensions of the factorized attention. If $\mathbf{x}_i$ is the one-hot encoded ith element in the sequence, and $\mathbf{o}_i^{(j)}$ represents the one-hot encoded position of $\mathbf{x}_i$ in the jth dimension ($1 \leq j \leq n_{emb}$), then:

$$\text{embed}(X, W_e) = \left(\mathbf{x}_i W_e + \sum_{j=1}^{n_{emb}} \mathbf{o}_i^{(j)} W_j \right)_{\mathbf{x}_i \in X} \tag{15}$$

For images, we used data embeddings, where $d_{data} = 3$ for the row, column, and channel location of each input byte. For text and audio, we used two-dimensional attention embeddings, where $d_{attn} = 2$ and the index corresponds to each position's row and column index in a matrix of width equal to the stride.

5.4. Saving memory by recomputing attention weights

Gradient checkpointing has been shown to be effective in reducing the memory requirements of training deep neural networks (Chen et al., 2016), (Gruslys et al., 2016). It is worth noting, however, that this technique is particularly effective for self-attention layers when long sequences are processed, as memory usage is high for these layers relative to the cost of computing them.

Using recomputation alone, we are able to train dense attention networks with hundreds of layers on sequence lengths of 16,384, which would be infeasible on modern hardware otherwise.

In our experiments, we recompute the attention and feedforward blocks during the backwards pass. To simplify our implementation, we do not apply dropout within the attention blocks, as in (Vaswani et al., 2017), and instead only apply it at the end of each residual addition, as seen in Figure 4.

5.5. Efficient block-sparse attention kernels

The sparse attention masks in 3(b) and 3(c) can be efficiently computed by slicing out sub-blocks from the query, key, and value matrices and computing the product in blocks. Attention over a local window can be computed as-is, whereas attention with a stride of k can be computed by transposing the matrix and computing a local window. Fixed attention positions can be aggregated and computed in blocks.

In order to ease experimentation, we implemented a set of GPU kernels which efficiently perform these operations. The softmax operation is fused into a single kernel and also uses registers to eliminate loading the input data more than once, allowing it to run at the same speed as a simple nonlinearity. The upper triangle of the attention matrix is never computed, moreover, removing the need for the negative bias term of (Vaswani et al., 2017) and halving the number of operations to be performed.

5.6. Mixed-precision training

We store network weights in single-precision floating-point, but otherwise compute network activations and gradients in half-precision, as in (Micikevicius et al., 2017). This accelerates our training due to the usage of Tensor Core operations on the V100 GPU. During the gradient calculation, we use

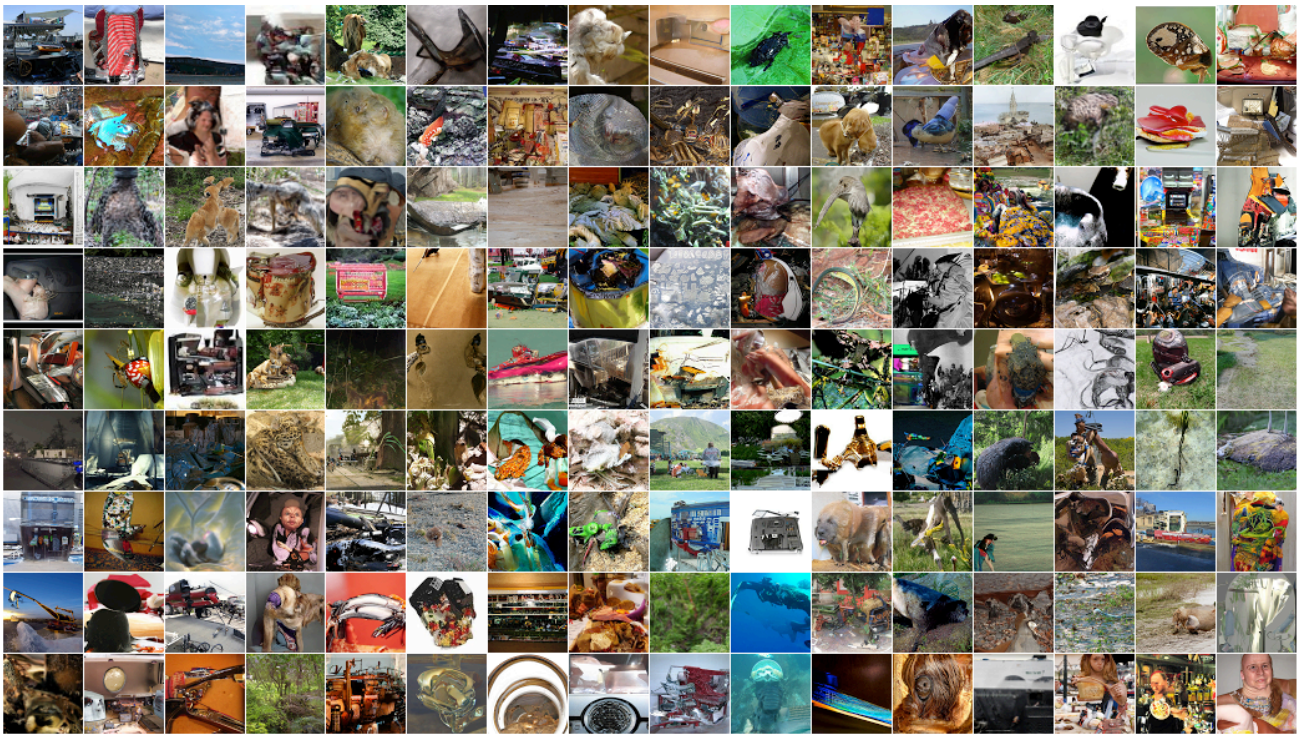


Figure 5. Unconditional samples from ImageNet 64x64, generated with an unmodified softmax temperature of 1.0. We are able to learn long-range dependencies directly from pixels without using a multi-scale architecture.

dynamic loss scaling to reduce numerical underflow, and we communicate half-precision gradients when averaging across multiple GPUs. When sampling, we cast the queries and keys to single-precision, as the query-key product can sometimes overflow the max value of half-precision.

6. Training

We use the Adam optimizer with a linear warmup of 5000 iterations and a gradient clipping of 1.0, both of which we found important for model stability. We use a weight decay penalty of 0.01. We annealed the learning rate according to a cosine decay as in (Radford et al., 2018). We train on 8 V100 GPUs unless otherwise noted.

All embeddings are of a constant dimension d, usually one of $\{256, 512, 1024\}$. By default, all linear transforms are to the same dimension, with the exception of the feed-forward network, which projects the input to $4d$, unless we use "half-size" transformations, where it is $2d$. Additionally, sometimes we halve the size of the query and key transformations.

We initialize the token embedding W_e from $\mathcal{N}(0, \frac{0.125}{\sqrt{d}})$ and the position embeddings from $\mathcal{N}(0, \frac{0.125}{\sqrt{dn_{emb}}})$. Within the attention and feedforward components, all biases are initialized to 0 and all weights are initialized from $\mathcal{N}(0, \frac{0.125}{\sqrt{d_{in}}})$ where d_{in} is the fan-in dimension. The weight matrix for the output logits was initialized to 0.

7. Experiments

We empirically test our architecture on density modeling tasks including natural images, text, and raw audio. A summary of the results is available in Table 1. We found that, in addition to running significantly faster than full attention, sparse patterns also converged to lower error, as shown in Table 2. This may point to a useful inductive bias from the sparsity patterns we introduced, or an underlying optimization issue with full attention.

7.1. CIFAR-10

We train strided Sparse Transformers on CIFAR-10 images represented as sequences of 3072 bytes. Models have 2 heads, 128 layers, d = 256, half-size feedforward network and query-key projections, and are trained for 120 epochs with a learning rate of 0.00035 and a dropout rate of 0.25 until validation error stops decreasing.

We use 48000 examples for training and 2000 examples for validation, evaluating the performance of our best models on

Table 1. Summary of our findings for density modeling tasks. Results are reported in bits per byte, which is equivalent to bits per dim for image tasks. M refers to millions of parameters.

Model	Bits per byte
CIFAR-10	
PixelCNN (Oord et al., 2016)	3.03
PixelCNN++ (Salimans et al., 2017)	2.92
Image Transformer (Parmar et al., 2018)	2.90
PixelSNAIL (Chen et al., 2017)	2.85
Sparse Transformer 59M (strided)	**2.80**
Enwik8	
Deeper Self-Attention (Al-Rfou et al., 2018)	1.06
Transformer-XL 88M (Dai et al., 2018)	1.03
Transformer-XL 277M (Dai et al., 2018)	**0.99**
Sparse Transformer 95M (fixed)	**0.99**
ImageNet 64x64	
PixelCNN (Oord et al., 2016)	3.57
Parallel Multiscale (Reed et al., 2017)	3.7
Glow (Kingma & Dhariwal, 2018)	3.81
SPN 150M (Menick & Kalchbrenner, 2018)	3.52
Sparse Transformer 152M (strided)	**3.44**
Classical music, 5 seconds at 12 kHz	
Sparse Transformer 152M (strided)	**1.97**

Table 2. Sparse patterns showed increased speed and also better loss on the datasets where we could compare both, which may point to a useful inductive bias in the patterns we learned or an underlying optimization issue with full attention.

Model	Bits per byte	Time/Iter
Enwik8 (12,288 context)		
Dense Attention	1.00	1.31
Sparse Transformer (Fixed)	**0.99**	0.55
Sparse Transformer (Strided)	1.13	0.35
CIFAR-10 (3,072 context)		
Dense Attention	2.82	0.54
Sparse Transformer (Fixed)	2.85	0.47
Sparse Transformer (Strided)	**2.80**	0.38

Table 3. We observe increased compression of Enwik8 with longer contexts, suggesting the Sparse Transformer can effectively incorporate long-term dependencies.

Minimum context length during evaluation	Bits per byte
6,144 tokens	0.9952
9,216 tokens	0.9936
10,752 tokens	0.9932
11,904 tokens	0.9930
12,096 tokens	0.9922
12,160 tokens	**0.9908**

the test set. The model achieves **2.80** bits per dim (2.798 ± 0.004 over seeds 1, 2, 3) versus the previous 2.85 state of the art (Chen et al., 2017). We also compare performance of different attention patterns in Table 2. The strided attention reaches the lowest error in the shortest amount of time, surpassing the error of dense attention at 2.82 bits per dim.

7.2. Text

In order to assess Sparse Transformers on datasets without a strong two-dimensional structure, we trained models on the EnWik8 dataset, which represents the first 10^8 bytes of Wikipedia and contains a great degree of variability in periodic structure. We trained with a context length of 12,288, which is longer than previous approaches.

We trained on the first 90 million tokens and reserved the last 10 million for validation and test. We used 30-layer fixed Sparse Transformers with 8 heads, d = 512, and a dropout rate of 0.40. We trained for 80 epochs until validation loss stopped decreasing. We used a stride of 128, $c = 32$, and merged the factorized attention heads.

Our best model reached **0.99** bits per dim (0.992 ± 0.001 over seeds 1, 2, 3), surpassing the 1.03 state-of-the-art for a similarly-sized Transformer-XL (Dai et al., 2018) and matching the 0.99 of a model trained with more than double the number of parameters. Strided attention failed to do well on this dataset, whereas fixed patterns were able to recover and surpass the performance of dense attention, as listed in Table 2.

Additionally, during evaluation of the test set, we modified the minimum context length the network could use by evaluating fewer tokens in parallel. We saw monotonic increases in performance with more tokens used, up to 12,160 out of the 12,288 tokens used for training (see Table 3), which suggests the network is effectively incorporating long-term dependencies.

7.3. ImageNet 64x64

In order to test the ability of the model to learn long range dependencies and scale to a large dataset, we train on the version of downsampled ImageNet released by (Oord et al., 2016) and evaluate on the validation set. We used a 48 layer strided Sparse Transformer with 16 attention heads and d = 512, totaling 152 million parameters. We used a stride of 128, a dropout of 0.01, and trained for 70 epochs, which took 7 days on 64 V100 GPUs.

Our model achieves a loss of **3.44** bits per dim (3.437 across 1 run), in comparison to the previous 3.52 (Menick & Kalchbrenner, 2018).

Additionally, we generate unconditional samples (Figure 5) from a model with twice the number of layers (300m parameters total) at an unmodified softmax temperature of 1.0. On visual assessment we find no artifacts from the sparsity patterns and see evidence of long-term structure in most images.

7.4. Classical music from raw audio

To test the extent to which Sparse Transformers are able to scale to very long contexts, we trained models on the classical music dataset released by (Dieleman et al., 2018). As details of the dataset processing are unavailable, we omit any direct comparison to other work and instead study what size of Sparse Transformer we can train with increasing context size. For each sequence length, we attempted to train the largest model which could entirely fit into 16GB V100 accelerators without model parallelism.

Overall, we found that increasing the sequence length by a factor of 4 requires a reduction in model capacity of approximately $4\sqrt{4} = 8$. Thus we found we could use factorized self-attention on sequences over 1 million timesteps long, albeit with extremely few parameters (3 million).

Samples are available for sequences of length 65,536, which correspond to around 5 seconds of generated audio at 12kHz. The samples clearly demonstrate global coherence over the sampled period, and exhibit a variety of play styles and tones, swapping from rhythmic playing to forceful. To listen to samples, visit `https://bit.ly/2DnJx34`. Sample quality quickly degrades for greater sequence lengths due to reduced model capacity.

Table 4. Performance of a strided Sparse Transformer on a classical audio dataset (μ-law encoded at 12 kHz) as a function of sequence length and model size.

Sequence length	Parameters	Bits per byte
65,536	152M	1.97
262,144	25M	2.17
1,048,576	3M	2.99

8. Conclusion

We introduced Sparse Transformers and showed they attain equivalent or better performance on density modeling of long sequences than standard Transformers while requiring significantly fewer operations. This performance is state-of-the-art in images and text and is easily adaptable to raw audio. The model demonstrates usage of long-term context and generates globally coherent samples.

9. Acknowledgements

We would like to thank Ashish Vaswani for insightful discussions during the genesis of the project. We also thank Joshua Meier and Mark Chen for helpful discussions, and Johannes Otterbach, Prafulla Dhariwal, and David Luan for feedback on drafts of this paper.

References

Al-Rfou, R., Choe, D., Constant, N., Guo, M., and Jones, L. Character-level language modeling with deeper self-attention. *arXiv preprint arXiv:1808.04444*, 2018.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

Britz, D., Guan, M. Y., and Luong, M.-T. Efficient attention using a fixed-size memory representation. *arXiv preprint arXiv:1707.00110*, 2017.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

Chen, X., Mishra, N., Rohaninejad, M., and Abbeel, P. Pixelsnail: An improved autoregressive generative model. *arXiv preprint arXiv:1712.09763*, 2017.

Chiu, C.-C. and Raffel, C. Monotonic chunkwise attention. *arXiv preprint arXiv:1712.05382*, 2017.

Dai, Z., Yang, Z., Yang, Y., Cohen, W. W., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Language modeling with longer-term dependency. 2018.

Dieleman, S., van den Oord, A., and Simonyan, K. The challenge of realistic music generation: modelling raw audio at scale. In *Advances in Neural Information Processing Systems*, pp. 8000–8010, 2018.

Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. Convolutional sequence to sequence learning. *arXiv preprint arXiv:1705.03122*, 2017.

Gruslys, A., Munos, R., Danihelka, I., Lanctot, M., and Graves, A. Memory-efficient backpropagation through time. In *Advances in Neural Information Processing Systems*, pp. 4125–4133, 2016.

He, K., Zhang, X., Ren, S., and Sun, J. Identity mappings in deep residual networks. *arXiv preprint arXiv:1603.05027*, 2016.

Hendrycks, D. and Gimpel, K. Bridging nonlinearities and stochastic regularizers with gaussian error linear units. *arXiv preprint arXiv:1606.08415*, 2016.

Jozefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., and Wu, Y. Exploring the limits of language modeling. *arXiv preprint arXiv:1602.02410*, 2016.

Kingma, D. P. and Dhariwal, P. Glow: Generative flow with invertible 1x1 convolutions. In *Advances in Neural Information Processing Systems*, pp. 10236–10245, 2018.

Koutnik, J., Greff, K., Gomez, F., and Schmidhuber, J. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

Liu, P. J., Saleh, M., Pot, E., Goodrich, B., Sepassi, R., Kaiser, L., and Shazeer, N. Generating wikipedia by summarizing long sequences. *arXiv preprint arXiv:1801.10198*, 2018.

Mehri, S., Kumar, K., Gulrajani, I., Kumar, R., Jain, S., Sotelo, J., Courville, A., and Bengio, Y. Samplernn: An unconditional end-to-end neural audio generation model. *arXiv preprint arXiv:1612.07837*, 2016.

Menick, J. and Kalchbrenner, N. Generating high fidelity images with subscale pixel networks and multidimensional upscaling. *arXiv preprint arXiv:1812.01608*, 2018.

Micikevicius, P., Narang, S., Alben, J., Diamos, G., Elsen, E., Garcia, D., Ginsburg, B., Houston, M., Kuchaev, O., Venkatesh, G., et al. Mixed precision training. *arXiv preprint arXiv:1710.03740*, 2017.

Oord, A. v. d., Kalchbrenner, N., and Kavukcuoglu, K. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016.

Parmar, N., Vaswani, A., Uszkoreit, J., Kaiser, Ł., Shazeer, N., and Ku, A. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/research-covers/language-unsupervised/language_ understanding_paper. pdf*, 2018.

Reed, S., Oord, A. v. d., Kalchbrenner, N., Colmenarejo, S. G., Wang, Z., Belov, D., and de Freitas, N. Parallel multiscale autoregressive density estimation. *arXiv preprint arXiv:1703.03664*, 2017.

Salimans, T., Karpathy, A., Chen, X., and Kingma, D. P. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017.

Sukhbaatar, S., Weston, J., Fergus, R., et al. End-to-end memory networks. In *Advances in neural information processing systems*, pp. 2440–2448, 2015.

Van Den Oord, A., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.