Sparse Sinkhorn Attention

Yi Tay [1] **Dara Bahri** [1] **Liu Yang** [1] **Donald Metzler** [1] **Da-Cheng Juan** [1]

Abstract

We propose Sparse Sinkhorn Attention, a new efficient and sparse method for learning to attend. Our method is based on differentiable sorting of internal representations. Concretely, we introduce a meta sorting network that learns to generate latent permutations over sequences. Given sorted sequences, we are then able to compute quasi-global attention with only local windows, improving the memory efficiency of the attention module. To this end, we propose new algorithmic innovations such as Causal Sinkhorn Balancing and SortCut, a dynamic sequence truncation method for tailoring Sinkhorn Attention for encoding and/or decoding purposes. Via extensive experiments on algorithmic seq2seq sorting, language modeling, pixel-wise image generation, document classification and natural language inference, we demonstrate that our memory efficient Sinkhorn Attention method is competitive with vanilla attention and consistently outperforms recently proposed efficient Transformer models such as Sparse Transformers.

1. Introduction

Learning sparse and efficient attention mechanisms has recently garnered considerable interest (Child et al., 2019; Kitaev et al., 2020). While existing state-of-the-art attention models have typically relied on dense, fully-connected attention graphs (Vaswani et al., 2017), these methods are often sub-optimal for two key reasons. First, large memory costs are incurred due to the quadratic complexity at the attention layer. Second, soft dense attention may suffer when ℓ, the sequence length, is large and noisy. Hence, at times, sparse attentive outputs that are reminiscent of hard attention methods, may serve as a desirable inductive bias (Xu et al., 2015).

[1]Google Research. Correspondence to: Yi Tay <yitay@google.com>.

Proceedings of the 37^{th} International Conference on Machine Learning, Vienna, Austria, PMLR 119, 2020.

This paper proposes a new method for (1) reducing the memory complexity of the dot-product attention mechanism and (2) learning sparse attention outputs. Our method is based on a novel idea of differentiable sorting of internal representations *within* the self-attention module. Our method, which we call Sparse Sinkhorn Attention, incorporates a meta sorting network that learns to re arrange and sort input sequences. With this new sorted sequence, attention computation is reduced substantially even when considering computation only within the local neighborhood, emulating a global effect even with solely local computation of context windows.

Our method is comprised of (1) a parameterized meta sorting network S for dynamically generating block-wise permutation matrices and (2) a standard local attention module that receives block-wise permuted input sequences for attention computation. Concretely, at the heart of our sorting network lives a differentiable Sinkhorn balancing mechanism (Adams & Zemel, 2011; Mena et al., 2018), which normalizes permutation matrices to belong to the Birkhoff polytope, the set of doubly stochastic matrices (Sinkhorn, 1964).

As such, given the block-sorted input sequences, the local attention module is able to compute attention weights beyond the default local neighborhood without incurring additional computation costs. Extensive experimental results across a potpourri of language, vision and arithmetic tasks demonstrate that Sparse Sinkhorn Attention outperforms strong baselines such as standard local attention and sparse attention Transformers (Child et al., 2019).

Notably, our proposed method is general purpose in nature and is applicable to sequence encoding, sequence decoding or seq2seq tasks (Sutskever et al., 2014). In order to adapt Sinkhorn balancing to decoding tasks, we propose a causal variant, i.e., Causal Sinkhorn Balancing. Moreover, for further improvement to encoding efficiency, we propose an additional SORTCUT variant of our proposed method, which dynamically truncates sequences in a data-driven manner based on a user-defined budget hyperparameter. Finally, we propose a Mixture model between the Sparse Sinkhorn Attention and standard vanilla attention, leading to further performance improvements.

Our method reduces the memory complexity from $O(\ell^2)$ to

$O(B^2 + N_B^2)$ where $B = \frac{\ell}{N_B}$. When ℓ is large, this factorization of sequence length brings about substantial savings in terms of memory complexity[1]. Our SORTCUT variant further reduces complexity to linear-time, i.e., $O(\ell N_k)$ where N_k is a user defined budget hyperparameter and $N_k <<< \ell$.

We also equip state-of-the-art Transformer models with our proposed Sparse Sinkhorn Attention, evaluating Sinkhorn Transformers on several large-scale sequence modeling tasks including language modeling on the One Billion Word Corpus (Chelba et al., 2013), pixel-wise image generation and document classification. Our proposed Sinkhorn attention remains competitive to the dense fully-connected attention while outperforming local attention and Sparse Transformers. While differentiable neural-based sorting has demonstrated some proof-of-concept promise (Mena et al., 2018), this work demonstrates the first successful application in real large-scale problems.

To summarize, the contributions of this paper are as follows:

- We propose Sparse Sinkhorn Attention, a new attention method based on dynamic, learnable sorting of internal representations. Our method is based on differentiable Sinkhorn balancing and is the first successful application of differentiable sorting on large-scale tasks.
- We also propose (1) Causal Sinkhorn balancing for autoregressive sequence decoding and (2) a new SORTCUT encoding scheme that further improves encoding efficiency by dynamically truncating sequences during attention computation.
- Our proposed methods reduce the memory complexity of dot-product attention while remaining competitive with or outperforming dense vanilla attention.
- We conduct extensive experiments on large-scale generative modeling tasks. On all tasks, Sinkhorn Transformers match and/or outperform vanilla Transformers while consistently outperforming Sparse Transformers (Child et al., 2019) and Local Attention Transformers.

2. Related Work

A natural and intuitive yet naive method typically employed for efficiently learning attention involves using a fixed window size. This method, usually referred to as local attention (Luong et al., 2015), has served as a simple and quick fix to run attention models on long sequences. An obvious weakness is that tokens in a window do not have access to context outside the window, restricting the expressiveness and its capability to model long-term dependencies. The study of window (or block-based) local attention has also been an emerging field of research (Shen et al., 2018b; Tay et al., 2019; Qiu et al., 2020; Child et al., 2019; Parmar et al., 2018).

Building upon the notion of local windows, Sparse Transformer (Child et al., 2019) proposed factorizing the attention computation into local and strided operations, delegating different heads to focus on different sparse patterns. They demonstrate promising results, establishing Sparse Transformer as one of the canonical methods[2] for efficient attention computation.

While our method also relies on sequence partitioning, we note that there have been several orthogonal but related efforts. Reformer (Kitaev et al., 2020), proposes locality sensitive hashing as a means to reduce the memory complexity of self-attention. Transformer-XL (Dai et al., 2019) adopts recurrence to cache hidden states across long sequences, which spurred further interest in modeling and compression of long term dependencies (Rae et al., 2020). Star Transformer (Guo et al., 2019) performs attention sparsification by converting the dense graph into a starshaped topology using a shared relay node. However, while this method enables linear-time complexity, its setup makes it difficult for causal masking, making the Star Transformer useful only for encoding.

Learning sparse outputs in attention models has also garnered reasonable interest. The key idea behind sparse weights (i.e., hard attention) is that they enable the model to only focus on a limited number of items at a time (Xu et al., 2015; Shen et al., 2018a). This can be a useful inductive bias when the input sequence is long and/or noisy, serving as a denoising filter. Moreover, hard attention can also improve inference speeds, as demonstrated by methods such as Sparsemax (Martins & Astudillo, 2016). Along a similar vein, this is also reminiscent of Sparse Mixture of Experts (Shazeer et al., 2017), which performs a sparse selection of outputs (experts) for prediction tasks.

Our proposed method is not only a new way of learning efficient attention but also a new way of sparsification. At the core of our approach lies a Sinkhorn ranking operation (Adams & Zemel, 2011) that is used for learning differentiable rankings over internal representations. Leveraging the Gumbel reparameterization trick (Jang et al., 2016), Gumbel Sinkhorn Networks (Mena et al., 2018) proposed stochastic maximization over the set of possible latent permutations. The core novelty of our work lies in the introduction of neural sorting as a means to sparsify and improve the efficiency of well-established attention networks.

[1] As an illustration, when $\ell = 1024$ and $N_B = 64$, this results in a memory saving factor of 240 times.

[2] That said, Sparse Attention requires highly specialized GPU kernels for efficient computation. This generally makes the approach less appealing, e.g., for portability purposes such as running on TPU pods.

3. Sparse Sinkhorn Attention

In this section, we introduce our proposed Sparse Sinkhorn Attention and provide a high-level overview. In our method, the input sequence X of length ℓ is partitioned into N_b blocks in which each block has a length of b tokens. Notably, the original idea of block-based local attention is to allow tokens to only attend to tokens within the same block. However, this restricts the global receptive field and limits the ability for local attention models to model long term dependencies.

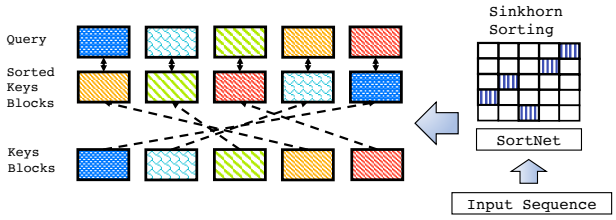

Figure 1. Overview of Sparse Sinkhorn Attention. A Meta Sorting Network learns to sort sequences to enable efficient quasi-global local attention.

Our proposed method mitigates this problem by neural sorting of blocks and receptive fields (neighborhoods). More concretely, instead of attending to tokens in the same block, each token attends to tokens in the newly *sorted* block, which may actually be far apart in the original unsorted sequence. Sorting blocks instead of individual tokens is also more intuitive, since we do not wish to break connections between nearby tokens, i.e., it would be reasonable for each token to still maintain an approximate neighborhood.

3.1. Learning to Sort

In order to learn to sort, we introduce a Sorting Network (SortNet) for learning relaxed permutation matrices. Since our sorting function is differentiable, the parameters of the SortNet are also trained together in an end-to-end fashion. The SortNet accepts an input sequence of ℓ vectors of d dimensions and partitions them into blocks.

$$X' = \psi_P(X) \tag{1}$$

The function $\psi_P(.)$ is a blockwise pooling operation that maps $\mathbb{R}^{\ell \times d} \rightarrow \mathbb{R}^{N_B \times d}$ and $X' \in \mathbb{R}^{N_B \times d}$. In SortNet, we adopt:

$$\psi_P(X)_i = \sum_{j=i*\ell_B}^{(i+1)*\ell_B} (X_j) \tag{2}$$

which is equivalent to taking the sum of embeddings of all tokens belonging to the local window. Our trainable SortNet is defined as follows:

$$R_i = P(X'_i) \tag{3}$$

where i refers to the block index. $P(.)$ is an arbitrary parameterized function which accepts an input vector of d dimensions and returns a vector of N_B dimensions. For example, we may parameterize $P(X)$ using a two layered feed-forward network with ReLU activations.

$$P(X) = \sigma(W_B \sigma(W_P(X) + b_P + b_B) \tag{4}$$

where $W_P \in \mathbb{R}^{d \times d}$ and $W_B \in \mathbb{R}^{d \times \ell_B}$. Essentially, the key idea is that each block learns a projection to N_B other blocks, effectively learning the position that it is supposed to be shifted (or permuted) to.

3.1.1. Sinkhorn Normalization

The matrix R becomes a sorting matrix (or permutation matrix) if it is doubly stochastic (matrix is nonnegative and both rows and columns all sum to 1). More specifically, a permutation matrix is special case of a doubly stochastic matrix (where rows and columns sum to 1 and all entries are either 0 or 1). Since every permutation matrix is a convex combination of doubly stochastic matrices, we consider learning doubly stochastic matrices as a a form of relaxed permutation matrix.

We consecutively normalize the rows and columns of the sorting matrix R, i.e., a process of Sinkhorn normalization (Adams & Zemel, 2011). Here, the number of iterations N_k is a user defined hyperparameter. This procedure is described as follows:

$$S^0(R) = \exp(R)$$
$$S^k(R) = F_c(F_r(S^{k-1}(R)))$$
$$S(R) = \lim_{k \rightarrow \infty} S^K(R)$$

where F_r, F_c are the row and column wise normalization function defined as follows:

$$F_c^k(X) = F_c^{k-1}(X) \oslash (X 1_\ell 1_N^\top)$$
$$F_r^k(X) = F_r^{k-1}(X) \oslash (1_\ell 1_N^\top X)$$

where $\oslash$ is the element-wise division operator, N is the length of the input matrix and $\mathbf{1}$ is a vector of ones. In practice, we perform calculations in log domain for improved stability.

$$F_c^k(X) = F_c^{k-1}(X) - \log(\exp(X 1_\ell) 1_N^\top)$$
$$F_r^k(X) = F_r^{k-1}(X) - \log(1_\ell 1_N^\top \exp(X))$$

To this end, (Sinkhorn, 1964) shows that iterative normalization of R converges to the doubly stochastic limit if R has support, i.e., a nonnegative matrix with a positive diagonal. Note that since R is nonnegative by design due to the usage of ReLU in $P(X)$. Gradients of the iterative Sinkhorn normalization can be computed, enabling end-to-end training.

3.1.2. Neural Sorting of Sequences

The generated permutation matrix is then used to sort the input sequence. This is described by a simple matrix multiplication of R against the blocked input sequence X':

$$X_S = U(RB(X))$$

where $B(.)$ converts input sequence into block-wise representations, i.e., $X' \in \mathbb{R}^{N_B \times (B \times d)}$ and $U(.)$ converts the block-wise sequences back into token-wise sequences. $U(.)$ and $B(.)$ can be interpreted as block-wise reshaping operators. Since R is doubly stochastic, multiplying a partitioned sequence by R is equivalent to sorting it.

3.2. Sparse Sinkhorn Attention

The key idea of the Sparse Sinkhorn Attention is to operate on block sorted sequences. Hence, the revised computation for the attention mechanism can now be written as:

$$A_{ij} = \begin{cases} (Q_i \psi_S(K)_j^\top), & \text{if } \lfloor j/\ell \rfloor = \lfloor i/\ell \rfloor \\ 0 & \text{otherwise} \end{cases}$$

$\psi(.)$ is the neural sorting function. Intuitively, this is identical to only enabling attention without a certain local neighborhood, albeit with key values sorted in a block-wise fashion. Subsequently, to compute and soft-select from the value matrix, we compute:

$$Y = \text{Softmax}(A)\psi_S(V) + \text{Softmax}(\hat{A})(V)$$

Here, the value matrix is also sorted accordingly. In practice, we share the sorting operator between the key and values. The secondary term $\hat{A}$ is the standard local attention which is added to the mixture.

$$\hat{A}_{ij} = \begin{cases} Q_i(K)_j^\top), & \text{if } \lfloor j/\ell \rfloor = \lfloor i/\ell \rfloor \\ 0 & \text{otherwise} \end{cases}$$

In practice, attention weights are only computed when $\lfloor j/\ell \rfloor = \lfloor i/\ell \rfloor$.

3.2.1. Gumbel Noise

For $S(X)$ to approximate the doubly-stochastic permutation matrix, we leverage the Gumbel categorical reparameterization trick (Jang et al., 2016). Concretely, we inject Gumbel noise into our sorting operator, i.e., $S(X) = S(\frac{(X+\epsilon)}{\tau})$ where ϵ is the injected standard i.i.d Gumbel noise and τ is the temperature hyperparameter. Intuitively, lowering the temperature brings $S(X)$ to be closer to a permutation matrix with discrete 1s and 0s.

3.2.2. Multihead Sparse Sinkhorn Attention

We have previously described the computation of a single Sinkhorn attention head. Similar to dot product attention, utilizing the multi-headed variation is straightforward.

$$Y_G = F_H([Y_1 \cdots Y_{N_H}])$$

where Y_i is the output of the i-th attention head. F_H is a linear transform layer with kernels $W \in \mathbb{R}^{(N_H \times d) \times d}$. Notably, our implementation learns a sorting network on a per head basis, i.e., we do not share the same permutation matrix R across all heads.

3.2.3. Mixture Model

Finally, we also consider a variant where the Sinkhorn Attention is used to model an alternate view of the input sequence. Concretely, we leverage the combination of the Sinkhorn attention by mixing it with the vanilla standard dot product attention.

$$Y = \text{Softmax}(A)\psi_S(V) + \text{Softmax}(QK^\top)V$$

Notably, the mixture mode regresses to the same quadratic complexity of vanilla self-attention. However, we hypothesize that the side network may provide an alternative and diverse view, ultimately improving performance.

3.3. Causal Sparse Sinkhorn Attention

Our Sinkhorn Attention not only involves sorting sequences but also learning the sorting order in a content-based fashion. To this end, pertaining to learning causal attention (i.e., no information from the future should leak to the present) there are two cases that we have to be careful about. The first is that current time steps should never have access to future time steps. Hence, if block i is sorted into a new position $p < i$, then it is being masked out. This produces an inductive bias that favors sorting orders that produce sorting between nearby blocks.

3.3.1. Causal Sorting Networks

The second case is the content-based and dynamic learning of sorting networks. To maintain the causal property, it would not be plausible to generate permutation matrices based on global information such as the sum of tokens in a sequence. Hence, the matrix R is generated using the cumulative sum of embeddings instead. This is described as follows:

$$\psi_P(X)_i = \sum_{j=0}^{(i*\ell_B+1)} (X_j) \quad \text{and} \quad R = P(\psi_P(X)) \tag{5}$$

Since our attention operates based on the idea of blocks, we use the first token in the block as its representative embedding. The cumulative sum operator allows the model to learn a permutation matrix conditioned on all previous context information leading up to the current block.

3.3.2. CAUSAL SINKHORN BALANCING

We note that the original Sinkhorn balancing requires knowledge of the future tokens for normalization. For causal self-attention, this is undesirable and non-permissible. Hence, we develop a causal variation of the typical Sinkhorn Balancing method which performs masking of the future while performing iterative normalization.

$$F_c^k(X) = F_c^{k-1}(X) - \log(\exp(M(X)1_\ell)1_N^\top)$$
$$F_{kr}(X) = F_r^{k-1}(X) - \log(1_\ell 1_N^\top M(\exp(X)))$$

where $M(.)$ is a masking function.

$$M(x) = \begin{cases} 0, & \text{if } j \geq i \\ 1, & \text{otherwise} \end{cases} \tag{6}$$

3.3.3. CONNECTIONS TO LEARNABLE SPARSITY

Due to the computation of causal Sinkhorn, it is expected that some blocks may be masked out, i.e., a block is masked out if it is sorted into an earlier position (i.e, $i' < i$). Essentially, the Sorting Network is also learning which tokens to mask by determining the sorting sequence.

3.4. SORTCUT Sinkhorn Attention

We propose an additional variant of our Sparse Sinkhorn Attention which we call SORTCUT. In this method, we propose a post-sorting truncation of the input sequence, essentially performing a hard top-k operation on the input sequence blocks within the computational graph. While most attention models mainly re-weight or assign near-zero weights during training, our method enables us to explicitly and dynamically truncate the input sequence. Specifically,

$$Y = Softmax(Q\psi_S(K)_{[:n]}^\top)\psi_S(V)_{[:n]}$$

where n is the SORTCUT budget hyperparameter. A caveat is that this mode may only be performed on the the Transformer encoder unless self-attention is explicitly computed again for every time-step in autoregressive decoding.

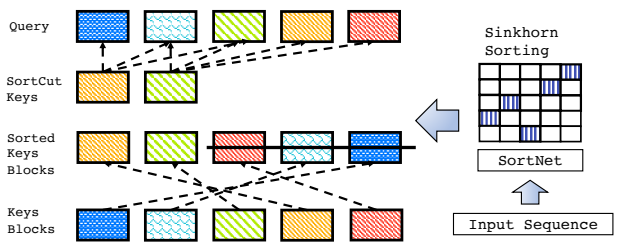

Figure 2. Overview of the proposed SortCut Encoding Scheme.

4. Complexity Analysis

The vanilla Transformer has a self-attention memory complexity of $O(\ell^2)$ where ℓ is the input sequence length. Our proposed Sinkhorn model reduces this to $O(B^2 + (\frac{\ell}{N_B})^2)$ where $B = \frac{\ell}{N_B}$. Essentially, this is equivalent to the memory complexity of local attention models. The SORTCUT Sinkhorn encoder has a memory complexity of $O(\ell N_k + (N_B)^2)$ where N_k is the budget hyperparameter. Since $\frac{\ell}{\ell_B} << \ell$, the complexity of the SORTCUT encoder can be reduced to $O(\ell)$.

5. Experiments

We evaluate the effectiveness of our proposed method for five tasks, including algorithmic sorting, language modeling, pixel-wise image generation, document classification and natural language inference. All our experiments are run on the open source Tensor2Tensor framework (Vaswani et al., 2018). If not stated otherwise, our Sinkhorn Transformers adopt the following global hyperparameters - temperature τ tuned among $\{0.25, 0.50, 0.75, 1.0\}$, number of sort iterations tuned among $\{2, 5, 10, 20\}$. Block size is largely dependent on the maximum length of the problem domain. Our tasks are designed to capture a wide and diverse range of scenarios such as medium to long range (e.g., 256 to 2048) and also covering a range of encoding focused and/or decoding focused tasks.

5.1. Algorithmic Tasks

We first test our model on a toy algorithmic sorting task. The task is cast as a sequence transduction problem (seq2seq) where the model is tasked to output a sorted sequence of an integer sequence.

Experimental Setup We use the `algorithmic_sort_problem` task in Tensor2Tensor. In this task, we train our models to sort sequences of $\ell = 256$ and evaluate on sequences of length 2ℓ (i.e., 512) to probe for generalization ability and ensure the models are not just simply memorizing. We evaluate based on exact match (EM) and edit distance (the lower the better). The exact match metric is defined by the number of test sequences that the model gets entirely correct. The dataset consists of $100K$ train examples and 1000 test examples. For Sinkhorn Transformers, we utilize Sparse Sinkhorn Attention for both encoding and decoding. We train all models for 200k steps using the default Transformer base hyperparameter. We compare against vanilla Transformers, local attention Transformers and Sparse Transformers.

Results on Sorting Task Table 1 reports our results on the algorithmic sorting task. Sinkhorn Transformers outper-

Model	Edit Dist.	EM
Transformer	0.4252	45.69
Local Attention (32)	0.4340	21.12
Sparse Transformer (32)	0.4176	46.88
Sinkhorn Transformer (8)	0.4156	43.65
Sinkhorn Transformer (16)	0.4071	48.23
Sinkhorn Transformer (32)	**0.4054**	**49.24**

Table 1. Evaluation on Algorithmic Sequences of length 256.

form all other Transformer variants, including the vanilla attention model. Sparse Transformer outperforms dense attention, which demonstrates the usefulness of sparse inductive biases. The local attention performs the worst, which demonstrates that some extent of global knowledge is required to solve this task.

5.2. Language Modeling

We evaluate on the LM1B (Language Modeling One Billion) dataset (Chelba et al., 2013), a large-scale language modeling benchmark. We evaluate on subword-level and character level language modeling on this task.

Experimental Setup We implement our model in the Tensor2Tensor framework, using the packed TPU setting. Tokens are split into 32k word pieces and sentences are shuffled. For word level language modeling, we use the default Tensor2Tensor hyperparameters[3]. Concretely, we evaluate two model sizes, BASE and BIG corresponding to `lmx_base` and `lmx_h2k_f8k` respectively. All models are trained for $300K$ steps on 16 TPU V2 Chips. For Sinkhorn Transformers and the corresponding Local Attention baseline, we tune the block size $B \in \{16, 32, 64\}$. We compare against the vanilla Transformer baseline, Local Attention Transformers, and Sparse Transformers (Child et al., 2019). We implement Sparse Transformers in Tensor2Tensor by referencing the original open source code using the fixed attention scheme (Child et al., 2019). However, instead of integrating the specialized cuda kernels, we manually simulated masking to achieve an equivalent implementation. We use a block length of $N_B = 64$ and fixed stride of $c = 8$. For character-level language modeling, owing to the overall larger sequence length, we use a maximum sequence length of 1024 and use a fixed block length of 128.

Results on Subword level Language Modeling Table 2 reports results on subword level language modeling[4]. On both parameter settings, Sinkhorn Transformers outperform all local attention models and Sparse Transformer. Pertaining to relative performance between Sinkhorn and local attention, the performance gain at each B setting ranges from $2-3$ perplexity points. Notably, on the base setting, Sinkhorn Transformer outperforms the vanilla Transformer at $B = 32$ and $B = 64$. At $B = 16$, Sinkhorn Transformers remain competitive to base Transformers. On the big setting, Sinkhorn Transformers fail to outperform vanilla Transformers, but still perform reasonably well despite being more memory efficient. Finally, the Sinkhorn Mixture model outperforms all models.

Model	Perplexity Base	Perplexity Big
Transformer (Vaswani et al., 2017)	41.57	27.59
Local Attention (16)	44.62	30.14
Local Attention (32)	44.23	29.32
Local Attention (64)	44.23	28.97
Sparse Transformer (64)	41.89	28.77
Sinkhorn Transformer (16)	42.64	29.42
Sinkhorn Transformer (32)	41.29	28.48
Sinkhorn Transformer (64)	40.79	28.39
Sinkhorn Mixture	**40.11**	**27.34**

Table 2. Experimental results on Language Model One Billion (LM1B) benchmark using the Base (50M parameter) and Big (430M) setting.

Results on Character-level Language Modeling Table 4 reports our experimental results (bytes per char) on character level language modeling. On both settings (base/big), our proposed Sinkhorn Transformer outperforms both local attention and Sparse Transformer, which affirms its effectiveness as an efficient attention method. On the contrary, local attention performs substantially worse compared to its counterparts, likely due to not having much global context. From this set of experiments, the vanilla full attention Transformer outperforms all efficient attention methods. However, our Sinkhorn Mixture model outperforms the Transformer baseline, achieving the best performance for both parameterizations.

Comparison with the State-of-the-art Table 3 reports our best scores relative to the state-of-the-art[5]. Notably, our best performing Sinkhorn Transformer remains competitive with the High Budget MoE (Shazeer et al., 2017) and Evolved Transformer (So et al., 2019) models. This demonstrates the overall competitiveness of Sinkhorn Transformers. Unfortunately, we were unable to outperform Mesh

[3] `tensor2tensor/models/research/lm_experiments.py`

[4] We also run experiments on the recently proposed Reformer (Kitaev et al., 2020) model, using the JAX framework. However, we were unable to obtain good performance. The best performance obtained is about 47 PPL. We are still extensively tuning the model and will provide an update in the final camera ready version.

[5] To the best of our knowledge, (Shazeer et al., 2018) is the best performing model on per-word perplexity. (Baevski & Auli, 2018) and (Dai et al., 2019) report per-token perplexity

Model	# Params	Perplexity
Low Budget MoE	5.0B	34.10
Transformer (Big)	141M	30.44
Evolved Transformer (Big)	151M	28.60
High Budget MoE	5.0B	28.00
Mesh Tensorflow	4.9B	**24.00**
Sinkhorn Transformer	450M	28.39
Sinkhorn Transformer	1.9B	27.34

Table 3. Comparison with other published works on LM1B that uses per-word Perplexity. Sinkhorn Transformer remains competitive to other Transformer models and High Budget MoE models.

	Bytes per char (Bpc)	
Model	Base	Big
Local Attention	2.559	1.825
Transformer	1.283	1.121
Sparse Transformer	1.300	1.134
Sinkhorn Transformer	1.295	1.132
Sinkhorn Mixture	**1.270**	**1.119**

Table 4. Experimental results on character level language modeling on LM1B with sequence lengths of 1024 characters.

Tensorflow (Shazeer et al., 2018) on our setup, which consists of 5 billion parameters. Nevertheless, we consider our results to be reasonable given the improved memory complexity.

5.3. Pixel-wise Image Generation

This section introduces and reports results on pixel-wise image generation task. This task models unconditional generative modeling of images by modeling images as flat sequences of pixels.

Experimental Setup We evaluate our model on pixel-by-pixel image generation using the Tensor2Tensor framework. We use the CIFAR-10 dataset. Similar to language modeling, we evaluate using bytes per dimension (Bpd), a common metric for evaluating generative modeling of images. In this task, images are flatted to sequences of 3076 bits which probes for long-term sequence modeling capabilities. We train all models using the base parameterization for $500K$ steps with a batch size of 1.

Model	Bpd
Local Attention	4.200
Transformer (Vaswani et al., 2017)	3.198
Sparse Transformer (256)	3.227
Sinkhorn Transformer (256)	**3.197**
Sinkhorn Mixture	3.199

Table 5. Experimental results on pixel-wise image generation (CIFAR-10)

Results on Image Generation Table 5 reports our results on the pixel-wise image generation task. Our proposed Sinkhorn Transformer outperforms all baselines. The local attention model performs the worst, which can be intuitively attributed to lack of global knowledge. While keeping the local window identical, our model also outperforms Sparse Transformer which demonstrates its utility as an efficient attention method. Finally, the Sinkhorn Mixture performs worse than the ordinary Sinkhorn Transformer, suggesting that a restricted (and learned) global view may serve as a useful inductive bias.

5.4. Text Classification

We evaluate several text classification benchmarks from the Tensor2Tensor framework. These tasks are mainly encoding only tasks, which allows us to benchmark the SORTCUT encoding scheme.

Experimental Setup We experiment on both sentiment analysis and natural language inference. For the former, we use the standard open source IMDb sentiment (Maas et al., 2011) and Sentiment Treebank (SST) dataset (Socher et al., 2013). For the latter, we use two natural language inference (NLI) datasets, i.e., Stanford NLI (Bowman et al., 2015) and MultiNLI (Williams et al., 2017).

For sentiment analysis, we evaluate on both character and word level. We set the maximum length of tokens to be 512/2048 for word/character level tasks respectively. We implement our models using Tensor2Tensor using the TINY Transformer setting (2 layers). Hyperparameters between our Sinkhorn Transformer and the vanilla Transformer remains identical. Token embeddings are initialized randomly. We train our models for 15000 steps for IMDb and SST and 500000 steps for NLI tasks. For all experiments, we use a batch size of 4096 tokens per batch. Models are trained with a single V100 GPU.

For natural language inference, experimental setup follows the Tensor2Tensor setup where premise and hypothesis are concatenated into one long input sequence. Word embeddings are also randomly initialized. We use the Transformer tiny hyperparameter for this task. We would like to emphasize that our experimental setup for these tasks differs from the standard usage of these datasets.

Results on Sentiment Analysis Table 6 reports results on sentiment analysis. Sinkhorn Transformers demonstrate promising results on sentiment analysis datasets on both word and character level. Even with significant memory savings, Sinkhorn Transformers are able to outperform or remain competitive with the baseline Transformer model. We also take this chance to benchmark the SORTCUT encoder, which further reduces memory complexity. On all

Model	IMDb		SST	
	Word	Char	Word	Char
(Vaswani et al., 2017)	**85.12**	62.77	76.83	57.45
Sinkhorn (8)	82.51	63.78	74.08	**62.27**
Sinkhorn (16)	82.00	62.05	76.15	56.08
Sinkhorn (32)	83.54	62.87	**77.52**	58.14
SORTCUT (2x8)	84.32	64.53	73.85	56.65
SORTCUT (2x16)	80.12	**64.87**	74.31	58.14
SORTCUT (2x32)	84.43	62.80	75.81	56.42

Table 6. Experimental results on word and character level document classification on IMDb dataset and SST datasets.

Model	SNLI	MNLI
Transformers (Vaswani et al., 2017)	78.87	53.69
Sinkhorn (8)	68.34	52.15
Sinkhorn (16)	77.77	52.09
Sinkhorn (32)	78.62	54.25
Sortcut Sinkhorn (2x8)	75.84	48.88
Sortcut Sinkhorn (2x16)	**80.30**	49.78
Sortcut Sinkhorn (2x32)	79.39	**55.80**

Table 7. Experimental results on natural language inference.

settings, we find that the SORTCUT variation can achieve similar performance to not only the standard Sinkhorn Transformer but also the vanilla Transformer.

Results on Natural Language Inference Table 7 reports results on SNLI and MNLI tasks. We find that both Sinkhorn and Sortcut Sinkhorn are able to outperform the vanilla Transformer. This task demonstrates the effectiveness of the SortCut variant despite the improvement of memory complexity over the standard Sinkhorn Transformer.

6. Analysis

In this section, we study the effect of certain modeling choices.

Modeling Choice	Perplexity
(1) $P(X) = \sigma(F_2(\sigma(F_1(X))))$	41.70
(2) $P(X) = F_2(\sigma(F_1(X)))$	41.38
(3) $P(X) = \sigma(F_1(X))$	41.34
(4) $P(X) = F_1(X)$	41.29
(5) $K = V$	42.26
(6) N_k=0 (no sinkhorn)	52.40

Table 8. Effect of different Sorting Network variants on $b = 32$ on LM1B (lower is better). $F(.)$ refers to linear transformation layers.

6.1. Effect of Modeling Choices

We are mainly interested in the effects of varying the Sorting Network model. Table 8 reports ablation studies on various model configurations. In (1) to (4), we vary the sorting network model. In (5), we experiment with a scheme to tie the weights of K and V (this is because they share the same permutation matrix). From Table 8, the best model for learning the sorting matrix is a linear layer, which signifies that the sorting network can be a simple model. We also observed that, more often than not, sharing the key-values seem to hurt performance. Finally in (6), we set $N_k = 0$ which is equivalent to not performing Sinkhorn normalization on R. We observe that performance degrades substantially and performs the worse of all ablative variations.

6.2. Hard or Soft Sorting?

Figure 3 reports the effect of varying Sinkhorn balancing temperatures. Keeping all other variables constant, we varied the temperature of the Gumbel Sinkhorn balancing mechanism. Overall, we find that maintaining a high temperature (inclined towards soft sorting) works better than a more discrete (hard) form of sorting. On this task, the optimal temperature is at $\tau = 0.75$.

6.3. Effect of Sorting Iterations

Figure 4 reports the performance trend with respect to N_k, the number of sorting iterations. Overall, a small number of sorting iterations is sufficient for good performance. No sorting at all performs extremely bad while the optimal number of sorting iterations seems to be $5 - 10$. Conversely, increasing the number of sorting iterations (beyond 20) seem to hurt perplexity scores.

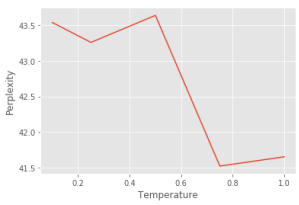

Figure 3. Effect of τ on perplexity scores (LM1B).

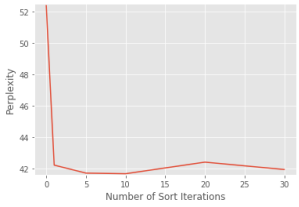

Figure 4. Effect of sorting iterations k on perplexity scores (LM1B).

7. Conclusion

We proposed Sparse Sinkhorn Attention, a new efficient and sparse method for attention computation. Our work demonstrates the utility of neural sorting of internal representations within the attention module on a multitude of large-scale generative modeling and classification tasks. On these benchamrks, our proposed Sinkhorn Transformer outperforms or remains competitive to vanilla Transformer and sparse Transformer models on a multitude of applications while being memory efficient.

References

Adams, R. P. and Zemel, R. S. Ranking via sinkhorn propagation. *arXiv preprint arXiv:1106.1925*, 2011.

Baevski, A. and Auli, M. Adaptive input representations for neural language modeling. *arXiv preprint arXiv:1809.10853*, 2018.

Bowman, S. R., Angeli, G., Potts, C., and Manning, C. D. A large annotated corpus for learning natural language inference. *arXiv preprint arXiv:1508.05326*, 2015.

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., and Robinson, T. One billion word benchmark for measuring progress in statistical language modeling. 2013.

Child, R., Gray, S., Radford, A., and Sutskever, I. Generating long sequences with sparse transformers. *arXiv preprint arXiv:1904.10509*, 2019.

Dai, Z., Yang, Z., Yang, Y., Carbonell, J., Le, Q. V., and Salakhutdinov, R. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Guo, Q., Qiu, X., Liu, P., Shao, Y., Xue, X., and Zhang, Z. Star-transformer. *arXiv preprint arXiv:1902.09113*, 2019.

Jang, E., Gu, S., and Poole, B. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

Kitaev, N., Kaiser, L., and Levskaya, A. Reformer: The efficient transformer. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rkgNKkHtvB.

Luong, M.-T., Pham, H., and Manning, C. D. Effective approaches to attention-based neural machine translation. *arXiv preprint arXiv:1508.04025*, 2015.

Maas, A. L., Daly, R. E., Pham, P. T., Huang, D., Ng, A. Y., and Potts, C. Learning word vectors for sentiment analysis. In *Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies-volume 1*, pp. 142–150. Association for Computational Linguistics, 2011.

Martins, A. and Astudillo, R. From softmax to sparsemax: A sparse model of attention and multi-label classification. In *International Conference on Machine Learning*, pp. 1614–1623, 2016.

Mena, G., Belanger, D., Linderman, S., and Snoek, J. Learning latent permutations with gumbel-sinkhorn networks. *arXiv preprint arXiv:1802.08665*, 2018.

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

Qiu, J., Ma, H., Levy, O., tau Yih, S. W., Wang, S., and Tang, J. Blockwise self-attention for long document understanding, 2020. URL https://openreview.net/forum?id=H1gpET4YDB.

Rae, J. W., Potapenko, A., Jayakumar, S. M., Hillier, C., and Lillicrap, T. P. Compressive transformers for long-range sequence modelling. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=SylKikSYDH.

Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.

Shazeer, N., Cheng, Y., Parmar, N., Tran, D., Vaswani, A., Koanantakool, P., Hawkins, P., Lee, H., Hong, M., Young, C., et al. Mesh-tensorflow: Deep learning for supercomputers. In *Advances in Neural Information Processing Systems*, pp. 10414–10423, 2018.

Shen, T., Zhou, T., Long, G., Jiang, J., Wang, S., and Zhang, C. Reinforced self-attention network: a hybrid of hard and soft attention for sequence modeling. *arXiv preprint arXiv:1801.10296*, 2018a.

Shen, T., Zhou, T., Long, G., Jiang, J., and Zhang, C. Bi-directional block self-attention for fast and memory-efficient sequence modeling. *arXiv preprint arXiv:1804.00857*, 2018b.

Sinkhorn, R. A relationship between arbitrary positive matrices and doubly stochastic matrices. *The annals of mathematical statistics*, 35(2):876–879, 1964.

So, D. R., Liang, C., and Le, Q. V. The evolved transformer. *arXiv preprint arXiv:1901.11117*, 2019.

Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A. Y., and Potts, C. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pp. 1631–1642, 2013.

Sutskever, I., Vinyals, O., and Le, Q. V. Sequence to sequence learning with neural networks. *arXiv preprint arXiv:1409.3215*, 2014.

Tay, Y., Wang, S., Tuan, L. A., Fu, J., Phan, M. C., Yuan, X., Rao, J., Hui, S. C., and Zhang, A. Simple and effective curriculum pointer-generator networks for reading comprehension over long narratives. *arXiv preprint arXiv:1905.10847*, 2019.

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.

Vaswani, A., Bengio, S., Brevdo, E., Chollet, F., Gomez, A. N., Gouws, S., Jones, L., Kaiser, L., Kalchbrenner, N., Parmar, N., Sepassi, R., Shazeer, N., and Uszkoreit, J. Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416, 2018. URL http://arxiv.org/abs/1803.07416.

Williams, A., Nangia, N., and Bowman, S. R. A broad-coverage challenge corpus for sentence understanding through inference. *arXiv preprint arXiv:1704.05426*, 2017.

Xu, K., Ba, J., Kiros, R., Cho, K., Courville, A., Salakhudinov, R., Zemel, R., and Bengio, Y. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pp. 2048–2057, 2015.