Fast Autoregressive Bit Sequence Modeling for Lossless Compression

Hiroaki Akutsu [1] **Ko Arai** [1]

Abstract

Autoregressive probability estimation of data sequences is a fundamental task in deep neural networks and has been widely used in applications such as lossless data compression. Since it is a sequential iterative process due to causality, there is a problem that its process is slow. In this paper, we propose *Scale Causal Blocks (SCBs)*, which are basic components of deep neural networks that aim to significantly reduce the computational and memory cost compared to conventional techniques. Evaluation results show that the proposed method is one order of magnitude faster than a conventional computationally optimized Transformer-based method while maintaining comparable accuracy.

1. Introduction

One of the basic tasks in deep neural networks is the probability estimation of data sequences. Autoregressive probability estimation, which is a simple task of predicting the next data from past data sequences, is known to achieve high accuracy when implemented by deep neural networks. Autoregressive probability estimation can be applied to image (Mentzer et al., 2018; Minnen et al., 2018), video (Lu et al., 2019; Mentzer et al., 2022), and lossless compression (Bellard, 2021) by combining it with entropy coding (Martin, 1979; Marpe et al., 2003; Duda, 2009).

Autoregressive probability estimation generally suffers from slow processing since it is a sequential iterative process due to the causality. One way to achieve high throughput is multiplexing on a graphics processing unit (GPU). To maximize the throughput of inference processing within the limited resources of the GPU, it is necessary to avoid the increase in computational complexity associated with deeper layers and to reduce the required memory consumption at higher multiplexing.

[1]R&D Group, Hitachi, Ltd., Japan. Correspondence to: Hiroaki Akutsu <hiroaki.akutsu.cs@hitachi.com>.

Proceedings of the Neural Compression Workshop at 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA.

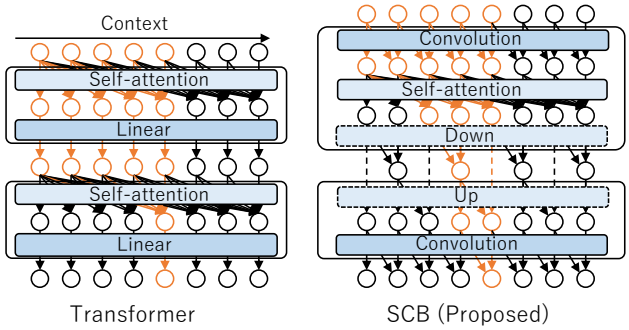

Figure 1. Overview of receptive fields in Transformers and SCBs. In Transformers, past contexts are considered by self-attention, whereas in the proposed SCB, past contexts are considered by both self-attention and convolution with *scaling* to improve computational efficiency.

In this work, we propose *Scale Causal Blocks (SCB)*, which are basic deep neural network components for autoregressive probability estimation that enables faster processing compared to conventional techniques. Our main contributions are as follows.

- We proposed SCBs as the basic components for the autoregressive probability estimation of data sequences. The computational cost was dramatically reduced while maintaining accuracy by combining convolution, scaling, and self-attention.
- We proposed inference algorithms with different parallelization strategies during training and inference. Specifically, during training, convolution is utilized to efficiently train long sequences in the context direction, and during inference, the weights of the convolutional layer are converted into a simple linear layer for faster processing by batch parallelization.
- Through our experiments, we demonstrated that the proposed algorithm can achieve faster inference throughput with comparable accuracy compared to the Linear Transformer.

Again, our goal is to establish efficient network components at a reasonable accuracy, not to achieve state-of-the-art accuracy. This perspective is now particularly important for bit cost reducing tasks such as compression tasks.

2. Related Works

This section describes related research on modeling with deep neural networks for data sequencing.

Transformer-based Models: The conventional Transformer (Vaswani et al., 2017) is computationally inefficient for processing long data sequences, as the computational cost of a self-attention is $O(N^2)$ for the length N of the data sequence. In regard to this issue, several methods have been proposed to improve the efficiency of the self-attention calculation (Child et al., 2019; Kitaev et al., 2020; Katharopoulos et al., 2020; Ma et al., 2021; Tay et al., 2022). In particular, Linear Transformer (Katharopoulos et al., 2020) can reduce the order of computational cost of self-attention on a layer to $O(N)$. Also, research on very deep Transformer methods is progressing, and it is now possible to construct large-scale models (Wang et al., 2022). However, the overall computational cost increases in proportion to the number of layers L, making the overall cost equivalent to $O(NL)$.

CNN-based Models: Since Transformer is based on a linear layer, it cannot consider any other data than the current position (except self-attention). In contrast, convolutional neural network (CNN) can consider the neighboring data sequence if the kernel size K is greater than 2. Wavenet (Oord et al., 2016) is a CNN-based model for modeling long receptive field data sequences K^L through dilated convolution. Caching the intermediate results of the dilated convolution can reduce the redundant computation of the feature map during inference (Ramachandran et al., 2017). However, an exponentially large number of caches relative to the number of layers is required, and it is difficult to increase the multiplicity of inference processing beyond the order of thousands due to the limited amount of memory on GPUs. In addition, the overall computational cost increases in proportion to the number of layers L, that is the same as with Transformers.

3. Scale Causal Blocks

In light of the above background, we proposed SCBs as the basic components of deep neural network for the autoregressive probability estimation of data sequences.

3.1. Scaling Causal Convolution with Self-attention

SCB has a unique feature that combines convolution, scaling, and self-attention for autoregressive probability modeling of data sequences at low computational cost.

Building blocks: The structures of the two basic building blocks that make up the SCB, which we call *Downscale Block (DB)* and *Upscale Block (UB)*, are shown in Fig. 2. DBs/UBs handle *scaling*, the reduction/expansion

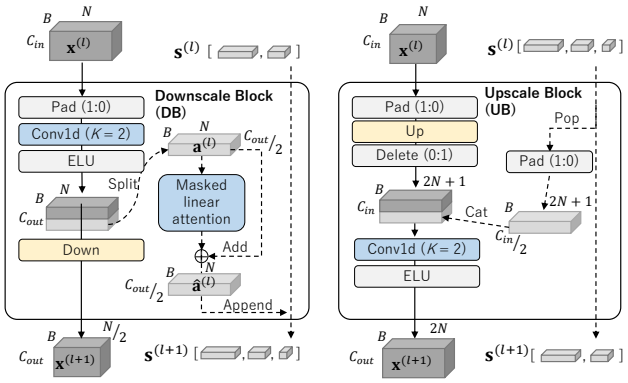

Figure 2. Architecture of SCB.

of a feature map to the context dimension, and it is key function for reducing computational cost. Each block inputs a tensor $\mathbf{x}^{(l)}$ and a list of short-cut path tensors $\mathbf{s}^{(l)} = [\hat{\mathbf{a}}^{(1)}, \cdots, \hat{\mathbf{a}}^{(l)}]$ and outputs $\mathbf{x}^{(l+1)}$ and $\mathbf{s}^{(l+1)}$ at the lth block layer. The DB halves the size of the tensor $\mathbf{x}$ in the context dimension, and the UB doubles its size. A typical configuration using these blocks is to connect multiple layers of DBs followed by multiple layers of an equal number UBs such that the sequence length of the first input N and the last output match.

DBs process an input tensor $\mathbf{x}^{(l)} \in \mathbb{R}^{B\times C_{\text{in}}\times N}$ by a one-dimensional convolutional neural network (1D CNN) where B is the batch size and then produce two types of outputs: half-downscaled tensor $\mathbf{x}^{(l+1)} \in \mathbb{R}^{B\times C_{\text{out}}\times N/2}$ and short-cut path tensor $\hat{\mathbf{a}}^{(l+1)} \in \mathbb{R}^{B\times C_{\text{out}}/2\times N}$ appended on $\mathbf{s}^{(l+1)}$, as shown in Fig. 2. The short-cut path tensor is processed with a masked linear attention (Katharopoulos et al., 2020). UBs first concatenate an input tensor $\mathbf{x}^{(l)} \in \mathbb{R}^{B\times C_{\text{in}}\times N}$ and short-cut path tensor $\hat{\mathbf{a}}^{(l)}$ poped from $\mathbf{s}^{(l)}$ using the padding and deleting tensor operations shown in Fig. 2. $\mathit{Pad}(1:0)$ represents one zero padding on the left side of the context dimension, and $\mathit{Delete}(0:1)$ represents one deletion from the right side. The UBs then process the concatenated tensor by a 1D CNN and finally produce a twice-upscaled tensor $\mathbf{x}^{(l+1)} \in \mathbb{R}^{B\times C_{\text{out}}\times 2N}$. In each block, the kernel size K of the 1D CNN is 2 and the stride size is 1. The exponential linear unit (ELU) (Clevert et al., 2015) is used as the activation function.

Scaling with Down / Up operations: In DBs and UBs, the operations that reduce and expand the context dimension are *Down* and *Up*. Our method aims to achieve these operations fast and without any arithmetic operations by replacing elements of the context dimension with the channel dimension (where C is the size), similar to PixelShuffle (Shi et al., 2016) in the image processing field. For more detail, see the appendix B.

Self-attention on short-cut path: Since SCB has a short-cut path similar to U-net (Ronneberger et al., 2015), we utilize it in the output $\mathbf{s}_l$ of each DB as a direct input to the corresponding UB, without processing of deeper blocks that have been reduced in the context dimension. In this way, it can avoid missing granularity information in the context dimension. We apply masked linear attention (Katharopoulos et al., 2020) to the feature maps of the short-cut path $\mathbf{a}$ (split tensor) to further improve the prediction accuracy at a low computational cost. For more detail, see the appendix C.

Weight sharing of deep layers: Due to the characteristics of SCBs, the data size of the feature map in the context dimension halves each time it goes to deeper layer, which makes it difficult to achieve stable training on these blocks. Therefore, we propose a method for sufficient training with fewer parameters that shares the weights of deep layers of the block by taking advantage of the characteristics of CNNs that can process even if the input size is changed in multi-scale.

3.2. Fast Inference Algorithm

This section describes the processing of SCB during inference. The two main features are explained below.

Convolution into Linear: During learning, the SCB uses convolution to efficiently learn long sequences in the context direction (i.e., the batch multiplicity is relatively small). In contrast, during inference, the batch multiplicity increases to achieve high throughput since it is a iterative sequential process. During inference, the weights are converted to the linear layer format, which enables efficient procesing with a high batch multiplicity.

Minimal caching: SCB is more memory efficient because it does not maintain a large amount of cache even in deep layers. For the dilated convolutions, exponential large cache size of context in deeper layer, whereas for SCBs, each layer requires only constant cache size since the context is extended by scaling. In addition, since attention is applied only to features in the shortcut paths, less memory is required for the iterative attention process in SCBs than Linear Transformers.

The detail algorithms are described in the appendix A.

3.3. Cost Estimations

We investigated the potential of the SCB by estimating its computational and memory costs and comparing it with conventional methods.

Computational Cost: The relationship between the number of layers and floating operations (FLOPs) for SCB and other methods is shown in Fig. 3. The number of channels is assumed to be $C_{\text{in}}, C_{\text{out}} = 256$ and is the same for all network types. This assumption also holds for the experimental results that follow, which show that the networks have approximately the same prediction accuracy. Increasing the number of layers not only increases the nonlinearity of the processing and improves the expressiveness of the network but also increases the receptive field in the convolution, which is advantageous because it means that longer contexts can be considered. The computational complexity increases with the number of layers in general as indicated by orange and grey curves, but as we can see in Fig. 3, the computational cost of the SCBs saturates with respect to the increase in the number of layers.

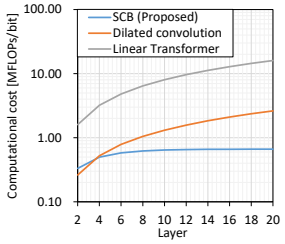

Figure 3. Computational cost.

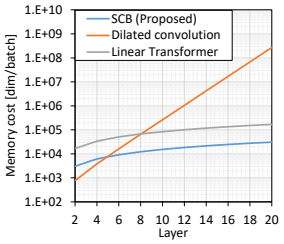

Figure 4. Memory cost.

Memory Cost: CNNs and attentions other than simple linear layers require cache memory to hold intermediate data (context) during inference. The capacity of this cache memory is proportional to the multiplicity (number of batches) during inference, so it must be smaller to achieve high multiplicity. The relationship between the number of layers and intermediate cache memory cost for SCB and other methods is shown in Fig. 4. In the case of dilated convolution, the amount of cache memory used increases exponentially with the number of layers, but the cost of the SCBs is only proportional to the number of layers. Furthermore, since attention is applied only to the features in short-cut paths, it reduces both the dimensionality of the channels of the attention networks and the number of attention mechanisms compared to conventional linear attention networks, and thus requires less memory for the iterative attention process in SCBs.

4. Experimental Results

This section presents the results of experimental studies on the effectiveness of SCB. *Lossless block compression* is a simple task that divides data into blocks of a fixed length N, treats each block simply as a bit sequence $x_1, ..., x_N \in \{0, 1\}$, and autoregressively estimates probabilities by a model for entropy coding.

The model θ is trained by an average of the Kullback-Leibler divergence of the ground truth probability distribution $p(x)$ and the estimated probability distribution $q_\theta(x)$,

Table 1. Experimental results of probability estimation of lossless block compression task (values in parentheses are standard deviations).

Method	Bpd (Theoretical compression rate)			Throughput	Cost	Parameters
	Genomics	MRI	Physics	[Mbit/sec]	[MFLOPs/bit]	[Mparms]
Linear Transformer	0.218 (0.004)	0.418 (0.007)	0.213 (0.002)	0.143 (0.0002)	12.8	12.6
SCB (proposed)	**0.217** (0.006)	**0.419** (0.004)	**0.137** (0.004)	**2.613** (0.0039)	**0.7**	**2.8**

Table 2. Experimental results of compression ratio. (values in parentheses are standard deviations)

Method	Compression ratio		
	Genomics	MRI	Physics
gzip -9 with 8192 bits block	0.464 (0.000)	0.749 (0.000)	0.481 (0.000)
gzip -9 with no block	0.332 (0.000)	0.635 (0.000)	0.334 (0.000)
SCB (Proposal)	**0.222** (0.006)	**0.425** (0.004)	**0.143** (0.004)

as

$$\mathcal{L} = \mathbb{E}_i[KL(p(x_i)||q_\theta(x_i|x_1, ..., x_{i-1}))]. \quad (1)$$

Entropy coding using the estimated probability can encode with bitrate approximately equal to the negative log-likelihood (Ho et al., 2019). This loss $\mathcal{L}$ is equal to the negative log-likelihood (and also it is equal to cross-entropy) when $p(x)$ is equal to one-hot encodings of the ground truth bits. So $\mathcal{L}$ can be treated as the theoretical average compression ratio and bits per dimension (bpd). The processing throughput of the compression task is important from a practical point of view because one of the main objectives of compression is to reduce storage costs. If the throughput of the compression process is slow, more time spent to occupy computing resources such as GPUs, which results in the effect of reducing storage costs by compressing data will be offset by the computational costs.

4.1. Experimental Conditions

We experimented with SCB on this task to determine whether SCB can handle probability prediction at high speed. We utilized three different types of open datasets (Genomics (EMBL, 2016), MRI (Alomair et al., 2015), and Physics (Baldi et al., 2016)) to evaluate the bpd and the processing speed of the probability estimation model. We compared the results to the Linear Transformer (Katharopoulos et al., 2020) as a baseline. In our experiment, the size was set to 1,024 bytes ($N = 8192$ bits). In the SCB experiment, the DB and UB were configured to be coupled with ten layers each, and the channel sizes C_{in} and C_{out} were set to 256. And the weights of the 1D CNNs of DBs after the 6th DB layer are shared, and the corresponding UB layers are also shared. We used a single NVIDIA®V100 for each experiments.

4.2. Comparison of Computing Effeciency

We measured throughput with a batch multiplicity of 8,192. Table 1 lists the results. As we can see, the SCB achieves a speedup of more than one order of magnitude over the conventional Linear Transformer with an equivalent bpd and less parameters. These results are supported by the fact that the estimated computational cost (MFLOPs/bit) of SCBs is smaller than that of Linear Transformers. The Physics dataset achieves a lower bpd with SCB. Please see the appendix D for details.

4.3. Comparison of Compression Ratios

A comparison of the compression ratios with gzip (Deutsch, 1996), a common conventional compression, is shown in Table 2. SCB allows partial encode/decode in 8192 bits units due to block compression. The SCB compression ratio includes the coding overhead. SCB has an advantage in the compression ratio even when compared to gzip without block compression (full-file compression) in the highest compression mode (option -9).

5. Conclusion

In this work, we proposed SCBs as the basic components for autoregressive probability estimation of data sequences. The computational cost was dramatically reduced while maintaining accuracy by combining the convolution, scaling, and self-attention. Experimental evaluations demonstrated that the proposed algorithms can achieve faster inference throughput to the Linear Transformer.

We believe it could reduce the environmental impact on society by reducing the consumption of storage and network bandwidth in the future.

References

Alomair, O. I., Brereton, I. M., Smith, M. T., Galloway, G. J., and Kurniawan, N. D. In vivo high angular resolution diffusion-weighted imaging of mouse brain at 16.4 tesla. *PloS one*, 10(6):e0130133, 2015.

Baldi, P., Cranmer, K., Faucett, T., Sadowski, P., and Whiteson, D. Parameterized machine learning for high-energy physics. *arXiv preprint arXiv:1601.07913*, 2016.

Bellard, F. Nncp v2: Lossless data compression with transformer. https://bellard.org/nncp/nncp_v2.1.pdf, February 2021.

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

Clevert, D.-A., Unterthiner, T., and Hochreiter, S. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

Deutsch, P. Gzip file format specification version 4.3. Technical report, 1996.

Duda, J. Asymmetric numeral systems. *arXiv preprint arXiv:0902.0271*, 2009.

EMBL. European nucleotide archive: Illumina hiseq 2000 paired end sequencing; gsm1080195: mouse oocyte 1; mus musculus; rna-seq. https://www.ebi.ac.uk/ena, 6 2016.

Ho, J., Lohn, E., and Abbeel, P. Compression with flows via local bits-back coding. *Advances in Neural Information Processing Systems*, 32, 2019.

Katharopoulos, A., Vyas, A., Pappas, N., and Fleuret, F. Transformers are rnns: Fast autoregressive transformers with linear attention. In *International Conference on Machine Learning*, pp. 5156–5165. PMLR, 2020.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Kitaev, N., Kaiser, Ł., and Levskaya, A. Reformer: The efficient transformer. *arXiv preprint arXiv:2001.04451*, 2020.

Lu, G., Ouyang, W., Xu, D., Zhang, X., Cai, C., and Gao, Z. Dvc: An end-to-end deep video compression framework. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11006–11015, 2019.

Ma, X., Kong, X., Wang, S., Zhou, C., May, J., Ma, H., and Zettlemoyer, L. Luna: Linear unified nested attention. *Advances in Neural Information Processing Systems*, 34: 2441–2453, 2021.

Marpe, D., Schwarz, H., and Wiegand, T. Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard. *IEEE Transactions on circuits and systems for video technology*, 13(7):620–636, 2003.

Martin, G. N. N. Range encoding: an algorithm for removing redundancy from a digitised message. In *Proc. Institution of Electronic and Radio Engineers International Conference on Video and Data Recording*, volume 2, 1979.

Mentzer, F., Agustsson, E., Tschannen, M., Timofte, R., and Van Gool, L. Conditional probability models for deep image compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4394–4402, 2018.

Mentzer, F., Toderici, G., Minnen, D., Caelles, S., Hwang, S. J., Lucic, M., and Agustsson, E. Vct: A video compression transformer. In *Advances in Neural Information Processing Systems*, 2022.

Minnen, D., Ballé, J., and Toderici, G. D. Joint autoregressive and hierarchical priors for learned image compression. *Advances in neural information processing systems*, 31, 2018.

Oord, A. v. d., Dieleman, S., Zen, H., Simonyan, K., Vinyals, O., Graves, A., Kalchbrenner, N., Senior, A., and Kavukcuoglu, K. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

Ramachandran, P., Paine, T. L., Khorrami, P., Babaeizadeh, M., Chang, S., Zhang, Y., Hasegawa-Johnson, M. A., Campbell, R. H., and Huang, T. S. Fast generation for convolutional autoregressive models. *arXiv preprint arXiv:1704.06001*, 2017.

Ronneberger, O., Fischer, P., and Brox, T. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241. Springer, 2015.

Shi, W., Caballero, J., Huszár, F., Totz, J., Aitken, A. P., Bishop, R., Rueckert, D., and Wang, Z. Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1874–1883, 2016.

Tay, Y., Dehghani, M., Bahri, D., and Metzler, D. Efficient transformers: A survey. *ACM Computing Surveys*, 55 (6):1–28, 2022.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Wang, H., Ma, S., Dong, L., Huang, S., Zhang, D., and Wei, F. Deepnet: Scaling transformers to 1,000 layers. *arXiv preprint arXiv:2203.00555*, 2022.

A. Details of Inference Argorithms

Algorithms 1 and 2 are the inference algorithms of SCB. The architecture in Fig. 2 is essentially a process during learning, while these algorithms are a process during inference. Although they have different parallelization policies, they are equivalent in terms of results and computational complexity. The details are omitted since the operation has already been outlined in Fig. 2, but we briefly go over the notable parts of the inference process in the following.

Since this is an autoregressive inference, the tensor $\mathbf{x} \in \mathbb{R}^{B \times C_{in}}$ at a certain location in the context dimension is used as the input of DBs and UBs. The initialize process sets initial values for the variables ($\mathbf{c}_{\mathrm{x}}$, $\mathbf{c}_{\mathrm{s}}$, $\mathbf{c}_{\mathrm{x1}}$, and $\mathbf{c}_{\mathrm{x2}}$) used as the caches of intermediate data in DBs and UBs before data sequence processing (corresponding to the *Pad* operations in Fig. 2). These variables are internal static variables in each block and are maintained during data sequence processing. By using these caches and concatenating the calculation results, equivalent processing to convolution can be done in the sequential processing of inference while increasing the multiplicity in the batch dimension (e.g., to several thousand or more). The *cat* and *split* are operated for the context dimension. The *linearfromconv* is a process that replaces the convolution layer with a linear layer. Specifically, the convolution process can be viewed as a linear layer with KC_{in} channels as input and C_{out} channels as output, so the parameters of the convolution kernel are converted into those of the linear layer. This eliminates the process for dimensional conversion of tensors, and allows for faster processing. The effect of reducing the computational complexity of the deeper layers by *scale* with decreasing the size of the feature map of the context dimension corresponds to the fact that, in the inference, the *linearfromconv* process and the attention process are executed less frequently as the layers of the block become deeper (due to the conditional branching of the algorithm), so the frequency of execution decreases.

Algorithm 1 Inference of down-scale block.

Require: $\mathbf{x} \in \mathbb{R}^{B \times C_{\mathrm{in}}}, \mathbf{s}$
(Initial: $\mathbf{c}_{\mathrm{x}} \leftarrow \mathbf{0}^{B \times C_{\mathrm{in}}}, \mathbf{c}_{\mathrm{s}} \leftarrow \mathrm{None}$)
Ensure: $\mathbf{x} \in \mathbb{R}^{B \times C_{\mathrm{out}}}, \mathbf{s}$
if $\mathbf{x}$ is None **then**
  $\mathrm{append}(\mathbf{s}, \mathrm{None})$
  return None, $\mathbf{s}$
end if
$\mathbf{t} \leftarrow \mathrm{cat}(\mathbf{c}_{\mathrm{x}}, \mathbf{x})$ $\{\mathbf{t} \in \mathbb{R}^{B \times 2C_{\mathrm{in}}}\}$
$\mathbf{c}_{\mathrm{x}} \leftarrow \mathbf{x}$
$\mathbf{x} \leftarrow \mathbf{t}$
$\mathbf{x} \leftarrow \mathrm{linearfromconv}(\mathbf{x})$ $\{\mathbf{x} \in \mathbb{R}^{B \times C_{\mathrm{out}}}\}$
$\mathbf{x} \leftarrow \mathrm{elu}(\mathbf{x})$
$\mathbf{x}, \mathbf{a} \leftarrow \mathrm{split}(\mathbf{x})$ $\{\mathbf{x}, \mathbf{a} \in \mathbb{R}^{B \times C_{\mathrm{out}}/2}\}$
$\hat{\mathbf{a}} \leftarrow \mathrm{attention}(\mathbf{a}) + \mathbf{a}$
$\mathrm{append}(\mathbf{s}, \hat{\mathbf{a}})$
if $\mathbf{c}_{\mathrm{s}}$ is None **then**
  $\mathbf{c}_{\mathrm{s}} \leftarrow \mathbf{x}$
  $\mathbf{x} \leftarrow \mathrm{None}$
else
  $\mathbf{x} \leftarrow \mathrm{cat}(\mathbf{c}_{\mathrm{s}}, \mathbf{x})$
  $\mathbf{c}_{\mathrm{s}} \leftarrow \mathrm{None}$
end if
return $\mathbf{x}, \mathbf{s}$

Algorithm 2 Inference of up-scale block.

Require: $\mathbf{x} \in \mathbb{R}^{B \times C_{\mathrm{in}}}, \mathbf{s}$
(Initial: $\mathbf{c}_{\mathrm{x1}}, \mathbf{c}_{\mathrm{x2}}, \mathbf{c}_{\mathrm{s}} \leftarrow \mathbf{0}^{B \times C_{\mathrm{in}}/2}$)
Ensure: $\mathbf{x} \in \mathbb{R}^{B \times C_{\mathrm{out}}}, \mathbf{s}$
$\hat{\mathbf{a}} \leftarrow \mathrm{pop}(\mathbf{s})$
if $\hat{\mathbf{a}}$ is None **then**
  return $\mathbf{x}, \mathbf{s}$
end if
if $\mathbf{x}$ is None **then**
  $\mathbf{x} \leftarrow \mathrm{cat}(\mathbf{c}_{\mathrm{x1}}, \mathbf{c}_{\mathrm{s}}, \mathbf{c}_{\mathrm{x2}}, \hat{\mathbf{a}})$ $\{\mathbf{x} \in \mathbb{R}^{B \times 2C_{\mathrm{in}}}\}$
else
  $\mathbf{t} \leftarrow \mathbf{c}_{\mathrm{x2}}$
  $\mathbf{c}_{\mathrm{x1}}, \mathbf{c}_{\mathrm{x2}} \leftarrow \mathrm{split}(\mathbf{x})$
  $\mathbf{x} \leftarrow \mathrm{cat}(\mathbf{t}, \mathbf{c}_{\mathrm{s}}, \mathbf{c}_{\mathrm{x1}}, \hat{\mathbf{a}})$ $\{\mathbf{x} \in \mathbb{R}^{B \times 2C_{\mathrm{in}}}\}$
end if
$\mathbf{c}_{\mathrm{s}} \leftarrow \hat{\mathbf{a}}$
$\mathbf{x} \leftarrow \mathrm{linearfromconv}(\mathbf{x})$ $\{\mathbf{x} \in \mathbb{R}^{B \times C_{\mathrm{out}}}\}$
$\mathbf{x} \leftarrow \mathrm{elu}(\mathbf{x})$
return $\mathbf{x}, \mathbf{s}$

B. Details of Up/Down Operation

In DBs and UBs, the operations that reduce and expand the context dimension are *Down* and *Up*. Assume a tensor $\mathbf{x} \in \mathbb{R}^{C \times N}$ (batch dimension is omitted because it is simply multiplexed), where each element of $\mathbf{x}$ is denoted as $x_{1,1} \cdots x_{C,N}$. The $Down$ operation is then defined as

$$Down(\mathbf{x}) = \begin{bmatrix} x_{1,1} & x_{1,3} & \cdots & x_{1,N-1} \\ \vdots & \vdots & \ddots & \vdots \\ x_{C,1} & x_{C,3} & \cdots & x_{C,N-1} \\ x_{1,2} & x_{1,4} & \cdots & x_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{C,2} & x_{C,4} & \cdots & x_{C,N} \end{bmatrix}_{2C \times N/2}, \tag{2}$$

And the Up operation is defined as

$$Up(\mathbf{x}) = \begin{bmatrix} x_{1,1} & x_{C/2+1,1} & \cdots & x_{1,N} & x_{C/2+1,N} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{C/2,1} & x_{C,1} & \cdots & x_{C/2,N} & x_{C,N} \end{bmatrix}_{C/2 \times 2N}. \tag{3}$$

C. Details of Masked Linear Attention

We apply masked linear attention (Katharopoulos et al., 2020) to the feature maps of the short-cut path $\mathbf{a}$ (split tensor) to further improve the prediction accuracy at a low computational cost. In this section, Masked Linear Attention reported in (Katharopoulos et al., 2020) is explained for the estimation of the computational and memory costs in the next section. The batch dimension and layer notations are omitted for simplicity. C_{att}, H, D, and $M = C_{\text{att}}/H$ denote the number of the channel dimensions, heads, query dimensions, and value dimensions, respectively. Note that C_{att} is equivalent to C_{in} for Linear Transformer, while for SCBs, C_{att} is equivalent to $C_{\text{out}}/2$.

First, an input tensor $\mathbf{a} \in \mathbb{R}^{C_{\text{att}} \times N}$ is projected to the queries $\mathbf{Q}^{(h)} \in \mathbb{R}^{N \times D}$, the keys $\mathbf{K}^{(h)} \in \mathbb{R}^{N \times D}$, and the values $\mathbf{V}^{(h)} \in \mathbb{R}^{N \times M}$ by weight matrices $\mathbf{W}_Q^{(h)}, \mathbf{W}_K^{(h)} \in \mathbb{R}^{C_{\text{att}} \times D}$ and $\mathbf{W}_V^{(h)} \in \mathbb{R}^{C_{\text{att}} \times M}$ for each head $h = 0, \cdots, (H-1)$ as follows:

$$\begin{aligned} \mathbf{Q}^{(h)} &= \mathbf{a}^\mathsf{T} \mathbf{W}_Q^{(h)} \\ \mathbf{K}^{(h)} &= \mathbf{a}^\mathsf{T} \mathbf{W}_K^{(h)} \\ \mathbf{V}^{(h)} &= \mathbf{a}^\mathsf{T} \mathbf{W}_V^{(h)}. \end{aligned} \tag{4}$$

From here, a subscript $*_i$ is introduced to represent the i-th position in the context dimension (e.g. $\mathbf{Q}_i^{(h)}$ is a vector whose shape is $\mathbb{R}^D$). Next, the attention memory $\mathbf{S}_i^{(h)} \in \mathbb{R}^{D \times M}$ and the normalizer memory $\mathbf{Z}_i^{(h)} \in \mathbb{R}^D$ is calculated as

$$\mathbf{S}_i^{(h)} = \sum_{j=1}^{i} \phi(\mathbf{K}_j^{(h)}) \mathbf{V}_j^{(h)\mathsf{T}} \tag{5}$$

$$\mathbf{Z}_i^{(h)} = \sum_{j=1}^{i} \phi(\mathbf{K}_j^{(h)}), \tag{6}$$

where $\phi(x)$ is a function defined by $\phi(x) = \text{elu}(x) + 1$. Note that $\phi(\mathbf{K}_j^{(h)}) \mathbf{V}_j^{(h)\mathsf{T}}$ in Eq. (5) is an outer product, not a matrix multiplication.

After that, the scaled dot-product attentions $\mathbf{A}_i^{(h)} \in \mathbb{R}^M$ are calculated for each head and the self attention $\mathbf{A}_i \in \mathbb{R}^{C_{\text{att}}}$ is

projected by a weight matrix $\mathbf{W}_A \in \mathbb{R}^{C_{\text{att}} \times C_{\text{att}}}$ as

$$\mathbf{A}_i^{(h)} = \tilde{\mathbf{V}}_i^{(h)} = \frac{\phi(\mathbf{Q}_i^{(h)})^\top \mathbf{S}_i^{(h)}}{\phi(\mathbf{Q}_i^{(h)})^\top \mathbf{Z}_i^{(h)}} \tag{7}$$

$$\mathbf{A}_{\text{cat}} = \text{cat}(\mathbf{A}_i^{(0)}, \cdots, \mathbf{A}_i^{(H-1)})$$

$$\mathbf{A}_i = \mathbf{W}_A \mathbf{A}_{\text{cat}}, \tag{8}$$

where $\tilde{\mathbf{V}}_i^{(h)}$ represents the updated values.

In the case of SCBs, the short-cut path tensor $\hat{\mathbf{a}}$ is calculated as

$$\hat{\mathbf{a}}_i = \mathbf{A}_i + \mathbf{a}_i. \tag{9}$$

D. Analysis of Prediction Accuracy

In the Physics dataset, SCBs showed singularly higher prediction accuracy than Linear Transformers. We therefore analyzed the prediction accuracy in the Physics dataset, which is csv files consist of floating-point data. Figure 5 shows the input data sequence of floating-point data in ascii text format (horizontal axis) and theoretical compression ratio (vertical axis) in the Physics dataset. For visibility, bitwise compression ratios are averaged into bytes. The latter half of the floating point data beyond the number of significant digits of the floating point formed a pattern that appeared in common with other data points, indicating that SCB was able to learn longer patterns than Linear Transformers, resulting in improved prediction accuracy, i.e., a higher theoretical compression ratio.

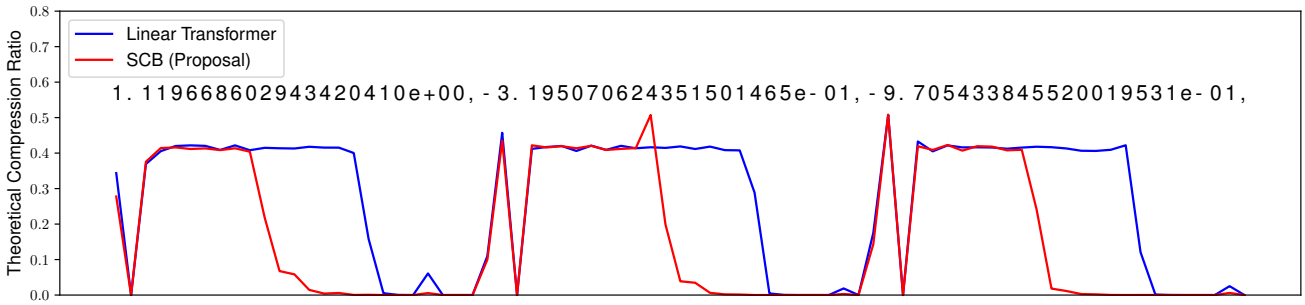

Figure 5. Physics (HEPMASS) dataset analysis.

E. Preliminary Experimental Results

Table 3 also shows a comparison of the experimental results when SCB scale is disabled/enabled and when self-attention is disabled/enabled. The results show that scaling and self-attention were effective for bpd reduction. This is because the SCB scale has the effect of expanding the receptive field and also self-attention works efficiently.

Table 3. Preliminary experimental results of scale and self-attention using Genomics dataset (values in parentheses are standard deviations).

Methods	Bpd	Cost [Mflop/dim]
No scale and attention	0.291 (0.002)	1.4
No scale	0.259 (0.005)	2.1
No attention	0.241 (0.009)	0.5
Full (Proposal)	**0.217** (0.006)	0.7

F. Details of Estimations

F.1. Computational costs

Operations in the Masked Linear Attention and LinearFromConv are provided in Table 4 and Table 5, respectively.

Table 4. Computational costs of Masked Linear Attention.

Operation	Details	Computational costs [FLOPs/bit/layer]	Note
$\mathbf{a}^\mathsf{T}\mathbf{W}_Q^{(h)}$	$\mathbb{R}^{C_{\mathrm{att}}} \times \mathbb{R}^{C_{\mathrm{att}} \times D}$ (H times)	$C_{\mathrm{att}}DH$	Eq. (4)
$\mathbf{a}^\mathsf{T}\mathbf{W}_K^{(h)}$	$\mathbb{R}^{C_{\mathrm{att}}} \times \mathbb{R}^{C_{\mathrm{att}} \times D}$ (H times)	$C_{\mathrm{att}}DH$	Eq. (4)
$\mathbf{a}^\mathsf{T}\mathbf{W}_V^{(h)}$	$\mathbb{R}^{C_{\mathrm{att}}} \times \mathbb{R}^{C_{\mathrm{att}} \times M}$ (H times)	$C_{\mathrm{att}}MH$	Eq. (4)
$\phi(\mathbf{K}_j^{(h)})\mathbf{V}_j^{(h)\mathsf{T}}$	$\mathbb{R}^{D} \otimes \mathbb{R}^{M}$ (H times)	DMH	Eq. (5)
$\phi(\mathbf{Q}_i^{(h)})^\mathsf{T}\mathbf{S}_i^{(h)}$	$\mathbb{R}^{D} \times \mathbb{R}^{D \times M}$ (H times)	DMH	Eq. (7)
$\mathbf{W}_A\mathbf{A}_{\mathrm{cat}}$	$\mathbb{R}^{C_{\mathrm{att}}} \times \mathbb{R}^{C_{\mathrm{att}} \times C_{\mathrm{att}}}$	C_{att}^2	Eq. (8)

Table 5. Computational costs of LineaFromConv.

Operation	Details	Computational costs [FLOPs/bit/layer]
LinearFromConv	$\mathbb{R}^{(K*C_{\mathrm{in}})} \times \mathbb{R}^{(K*C_{\mathrm{in}}) \times C_{\mathrm{out}}}$	$KC_{\mathrm{in}}C_{\mathrm{out}}$

Under the conditions used in the main text, the parameters shown in Table 4 and Table 5 can be described by only using

C_{in} and H as follows:

$$\begin{aligned} D^{(\text{LT})} &= C_{\text{in}}/H \\ M^{(\text{LT})} &= C_{\text{in}}/H \\ C_{\text{att}}^{(\text{LT})} &= C_{\text{in}} \\ \\ D^{(\text{SCB})} &= C_{\text{in}}/2H \\ M^{(\text{SCB})} &= C_{\text{in}}/2H \\ C_{\text{att}}^{(\text{SCB})} &= C_{\text{in}}/2 \\ C_{\text{out}} &= C_{\text{in}} \end{aligned} \tag{10}$$

First, we explain the computational costs of the Linear Transformer $\eta^{(\text{LT})}$. In addition to the operations provided in Table 4, there is a FeedForward operation in each layer. The FeedForward operation consists of $\mathbb{R}^{C_{\text{att}}^{(\text{LT})}} \times \mathbb{R}^{C_{\text{att}}^{(\text{LT})} \times 4C_{\text{att}}^{(\text{LT})}}$ and $\mathbb{R}^{4C_{\text{att}}^{(\text{LT})}} \times \mathbb{R}^{4C_{\text{att}}^{(\text{LT})} \times C_{\text{att}}^{(\text{LT})}}$, which is equivalent to $2 \times 4(C_{\text{att}}^{(\text{LT})})^2$ FLOPs per bit per layer. From the above, $\eta^{(\text{LT})}$ is calculated as

$$\begin{aligned} \eta^{(\text{LT})} &= [2C_{\text{att}}^{(\text{LT})} D^{(\text{LT})} H + C_{\text{att}}^{(\text{LT})} M^{(\text{LT})} H + 2D^{(\text{LT})} M^{(\text{LT})} H + (C_{\text{att}}^{(\text{LT})})^2 \\ &\quad + 2 \times 4(C_{\text{att}}^{(\text{LT})})^2] L \text{ [FLOPs/bit]}. \end{aligned} \tag{11}$$

By applying Eq. (10), $\eta^{(\text{LT})}$ is summarized as

$$\eta^{(\text{LT})} = \left(12 + \frac{2}{H}\right) C_{\text{in}}^2 L \text{ [FLOPs/bit]}. \tag{12}$$

Next, we explain the computational costs of SCBs $\eta^{(\text{SCB})}$. The UB process consists of just a LinearFromConv while the DB process consists of a LinearFromConv and a Masked Linear Attention. Additionally, lth layer is computed $1/2^l$ times per bit in SCBs. From the above, $\eta^{(\text{SCB})}$ is calculated as

$$\begin{aligned} \eta^{(\text{SCB})} &= [2C_{\text{att}}^{(\text{LT})} D^{(\text{LT})} H + C_{\text{att}}^{(\text{LT})} M^{(\text{LT})} H + 2D^{(\text{LT})} M^{(\text{LT})} H + (C_{\text{att}}^{(\text{LT})})^2] \\ &\quad \times \left(1 + \frac{1}{2} + \cdots \frac{1}{2^{L/2-1}}\right) \\ &\quad + (KC_{\text{in}}C_{\text{out}}) \times 2 \left(1 + \frac{1}{2} + \cdots \frac{1}{2^{L/2-1}}\right) \text{ [FLOPs/bit]}. \end{aligned} \tag{13}$$

By applying Eq. (10), $\eta^{(\text{SCB})}$ is summarized as

$$\eta^{(\text{SCB})} = \left(2 + \frac{1}{H} + K\right) \left(2 - \frac{1}{2^{L/2-1}}\right) C_{\text{in}}^2 \text{ [FLOPs/bit]}. \tag{14}$$

Lastly, we explain the computational costs of dilated convolutions $\eta^{(\text{DC})}$. Since each layer consists of just a LinearFromConv, $\eta^{(\text{DC})}$ is calculated as

$$\begin{aligned} \eta^{(\text{DC})} &= KC_{\text{in}}C_{\text{out}} \times L \\ &= KC_{\text{in}}^2 L \text{ [FLOPs/bit]}. \end{aligned} \tag{15}$$

F.2. Memory costs

The required memory for Masked Linear Attention, SCBs, and dilated convolutions is listed in Table 6, Table 7, and Table 8, respectively.

Table 6. Memory costs of Masked Linear Attention.

Description	Shape	Requried memory [dim/batch/layer]
Attention memory $\mathbf{S}_i^{(h)}$	$\mathbb{R}^{D\times M}$ ($\times H$ pcs.)	DMH
Normalizer memory $\mathbf{Z}_i^{(h)}$	$\mathbb{R}^{D}$ ($\times H$ pcs.)	DH

Table 7. Memory costs of the cached results in SCBs.

Block	Description	Shape	Requried memory [dim/batch/layer]
DB	$\mathbf{c}_{\mathrm{x}}$	$\mathbb{R}^{C_{\mathrm{in}}}$	C_{in}
	$\mathbf{c}_{\mathrm{s}}$	$\mathbb{R}^{C_{\mathrm{out}}/2}$	$C_{\mathrm{out}}/2$
	$\mathbf{s}^{(l)}$	$\mathbb{R}^{C_{\mathrm{out}}/2}$	$C_{\mathrm{out}}/2$
UB	$\mathbf{c}_{\mathrm{x1}}$	$\mathbb{R}^{C_{\mathrm{in}}/2}$	$C_{\mathrm{in}}/2$
	$\mathbf{c}_{\mathrm{x2}}$	$\mathbb{R}^{C_{\mathrm{in}}/2}$	$C_{\mathrm{in}}/2$
	$\mathbf{c}_{\mathrm{s}}$	$\mathbb{R}^{C_{\mathrm{in}}/2}$	$C_{\mathrm{in}}/2$

Table 8. Memory costs of dilated convolutions.

Description	Shape	Requried memory [dim/batch/layer]
Caluculated results of each layer	C_{in} ($\times 2^l$ pcs.)	$C_{\mathrm{in}} \times 2^l$

Similar to the previous section, the memory costs (elements per dim) of the Linear Transformer $\beta^{(\mathrm{LT})}$, SCBs $\beta^{(\mathrm{SCB})}$, and the dilated convolutions $\beta^{(\mathrm{DC})}$ is calculated as follows:

$$\begin{aligned}
\beta^{(\mathrm{LT})} &= (D^{(\mathrm{LT})}M^{(\mathrm{LT})}H + D^{(\mathrm{LT})}H) \times L \\
&= \left(\frac{C_{\mathrm{in}}^2}{H} + C_{\mathrm{in}}\right) L \ [\mathrm{dim/batch}], \qquad (16) \\
\beta^{(\mathrm{SCB})} &= (D^{(\mathrm{SCB})}M^{(\mathrm{SCB})}H + D^{(\mathrm{SCB})}H) \times \frac{L}{2} \\
&\quad + \left(C_{\mathrm{in}} + 2\frac{C_{\mathrm{out}}}{2}\right) \times \frac{L}{2} \\
&\quad + \left(3\frac{C_{\mathrm{in}}}{2}\right) \times \frac{L}{2} \\
&= \left(\frac{C_{\mathrm{in}}^2}{8H} + 2C_{\mathrm{in}}\right) L \ [\mathrm{dim/batch}], \qquad (17) \\
\beta^{(\mathrm{DC})} &= C_{\mathrm{out}} \times (2^0 + 2^1 + \cdots + 2^{L-1}) \\
&= C_{\mathrm{in}}(2^L - 1) \ [\mathrm{dim/batch}]. \qquad (18)
\end{aligned}$$

Note that the memory costs per layer in DBs ($L/2$ layers) and in UBs ($L/2$ layers) differ since only DBs have the linear attention process.

G. Expremental details

G.1. Expremental Conditions

We experimented with SCB on this task to determine whether SCB can handle probability prediction at high speed. We utilized three different types of open datasets (Genomics (EMBL, 2016), MRI (Alomair et al., 2015), and Physics (Baldi et al., 2016)) to evaluate the bpd and the processing speed of the probability estimation model. We compared the results to the Linear Transformer (Katharopoulos et al., 2020) as a baseline. Lossless block compression divides chunks of data into blocks of a fixed size for faster loading by partial decoding and parallel processing. In our experiment, the size was set to 1,024 bytes ($N = 8192$ bits). In the SCB experiment, the DB and UB were configured to be coupled with ten layers each, and the channel sizes C_{in} and C_{out} were set to 256. And the weights of the 1D CNNs of DBs after the 6th DB layer are shared, and the weights of the UB layers corresponding to the short-cut path of the shared DB layers are also shared among these UBs. In the Linear Transformer experiment, we set the embedding size to 256, the number of heads to 8, and the number of layers to 16, as in the experimental configuration described in (Katharopoulos et al., 2020). In both experiments, as with the general Transformers (Vaswani et al., 2017), the input bits were embedded to a 256-dimensional value and positional encoding was added. As a final layer, a linear layer with one output channel and a Sigmoid function were applied. The model was trained to output a probability that the bit is 1. We used the ADAM optimizer (Kingma & Ba, 2014) with a learning rate of $1.0e^{-4}$ for training and a batch size of 8. The training iteration was 200,000. We implemented the experimental code on Pytorch and used the library for fast transformer implementations (Katharopoulos et al., 2020) for the masked linear attention part. The experiments were performed with 32-bit floating-point arithmetic simply for the sake of pure method comparison. We used a single NVIDIA®V100 for each experiments.

G.2. Evaluation Datasets

The details of the datasets used in the lossless block compression evaluation experiments are shown in Table 9.

Table 9. Details of evaluation datasets.

Dataset	Items	Description
Genomics	Name	Illumina HiSeq 2000 paired end sequencing GSM1080195: mouse oocyte 1 Mus musculus RNA-Seq (EMBL, 2016)
	URL	https://www.ebi.ac.uk/ena/
	File (train)	SRR689233_1.fastq (3.87 GB) (md5: 56cb883e8b42344384b9e4ccc90ec9db)
	File (test)	SRR689233_2.fastq (3.87 GB) (md5: 92439bb6745f4abbf46b99efcbf20a02)
MRI	Name	In vivo High Angular Resolution Diffusion-weighted Imaging of Mouse Brain at 16.4 Tesla (Alomair et al., 2015)
	URL	https://dataverse.harvard.edu/
	File (train)	in-vivo-DWI-EPI.tar (0.94 GB) (md5: 4b247a403110dceb9631b365cee42813)
	File (test)	invivo-insitu-experiment.tar (0.76 GB) (md5: 5eb5203b0fca67411f39c2377336605b)
Physics	Name	HEPMASS Dataset (Baldi et al., 2016)
	URL	http://archive.ics.uci.edu/
	File (train)	all_train.csv (5.18 GB) (md5: 5b1fc2dafe14aa2f661cc3de5ccf3984)
	File (test)	all_test.csv (2.59 GB) (md5: 414f886d007f18b1eb97257a36120389)