TRAINING AND GENERATING NEURAL NETWORKS IN COMPRESSED WEIGHT SPACE

Anonymous authors
Paper under double-blind review

ABSTRACT

The inputs and/or outputs of some neural nets are weight matrices of other neural nets. Indirect encodings or end-to-end compression of weight matrices could help to scale such approaches. Our goal is to open a discussion on this topic, starting with recurrent neural networks for character-level language modelling whose weight matrices are encoded by the discrete cosine transform. Our fast weight version thereof uses a recurrent neural network to parameterise the compressed weights. We present experimental results on the enwik8 dataset.

1 INTRODUCTION

Many modern neural nets (NNs) are big. Their number of trainable parameters may easily exceed a few hundred million. This represents a bottleneck for end-to-end differentiable systems that manipulate some NN's weights as input or output variables of another NN, i.e., fast weight programmers (Schmidhuber, 1991; 1992b). This concept has seen a revival under various names such as "hypernetworks" (Ha et al., 2017). The latter were presented as NNs that produce a weight matrix of another NN, but practical implementations only generate a vector that scales the rows of a weight matrix, instead of generating a de nuovo weight matrix. In the original work (Schmidhuber, 1991; 1992b) and related fast weight variants (Schmidhuber, 1993d; Ba et al., 2016; Schlag & Schmidhuber, 2017), the matrix size bottleneck is addressed through outer products of two NN-invented vectors of activations which generate rank-one weight matrices.

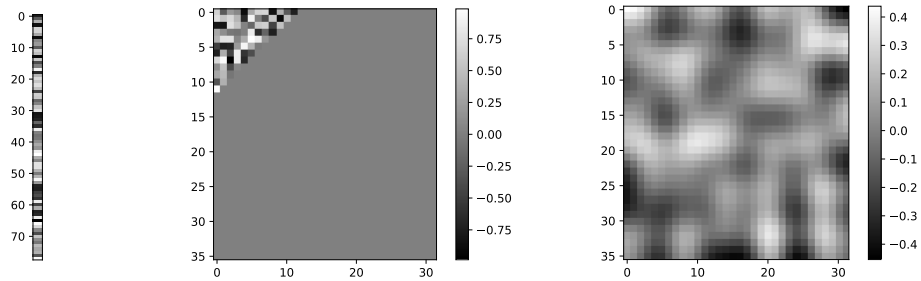
For small NNs, the whole weight matrix can be directly parametrised as the output of another NN (Schmidhuber, 1991; Sitzmann et al., 2020), a natural way of generating context-dependent weight matrices. If we had a differentiable compression method for weight matrices, we could scale up such direct approaches. Compact weight matrix representations could also be useful in scenarios which require a NN to read the weight matrix of another NN or itself (Schmidhuber, 1992a; 1993b;a;c; Unterthiner et al., 2020; Harb et al., 2020; Faccio et al., 2021).

Here we revisit indirect encodings (Koutník et al., 2010a;b) based on the Discrete Cosine Transform (DCT) (Ahmed et al., 1974). DCT offers a simple and differentiable encoding of NN weight matrices. The model can be trained from scratch and end-to-end (including the DCT transformations). While NNs with DCT-encoded weight matrices have been successfully applied in evolutionary settings (Koutník et al., 2010a;b), DCT's effect on model performance has not been clearly quantified in prior work. We evaluate recurrent neural networks (RNNs) with DCT-encoded weight matrices on a character level language modelling task with various compression rates. We also report preliminary results of the fast weight version in which we augment the DCT-based model by another NN which generates the DCT-coefficients of the main NN.

2 NEURAL NETWORKS WITH DCT-ENCODED WEIGHT MATRICES

Here we review neural networks whose weight matrices are encoded as DCT coefficients (Koutník et al., 2010a;b; 2012), and define the type of models considered in this work.

In contrast to regular neural networks whose trainable parameters are a set of weight matrices $\theta = \{\boldsymbol{W}_1, ..., \boldsymbol{W}_N\}$, trainable parameters in a DCT-based neural network are a set of DCT frequency coefficients $\mathcal{G} = \{\boldsymbol{g}_1, ..., \boldsymbol{g}_N\}$ which are used to generate the effective weight matrices



(a) DCT-coefficient vector

(b) Sparse frequency domain weight matrix

(c) Dense time domain weight matrix

Figure 1: Illustration of the decompression steps from a DCT-coefficient vector (model parameters) to an actual weight matrix. The DCT-coefficient vector (Figure 1a) is transformed into a sparse frequency domain weights (Figure 1b) by $\mathrm{TopLeft}$ operation (Eq. 2). Then, inverse DCT (Eq. 3) transforms the frequency matrix of Figure (1b) to the final dense weight matrix of Figure (1c).

$\{\boldsymbol{W}^{(g_1)}, ..., \boldsymbol{W}^{(g_N)}\}$. In prior work (Koutník et al., 2010a;b; 2012; Gomez et al., 2012; Srivastava et al., 2012; van Steenkiste et al., 2016), $\mathcal{G}$ is referred to as *genome*, and each $\boldsymbol{g}_i$ as *chromosome*.

As a formal illustration let us consider a regular feed-forward layer (where we omit bias) which transforms an input vector $\boldsymbol{x} \in \mathbb{R}^m$ to an output $\boldsymbol{y} \in \mathbb{R}^n$ with an element-wise activation σ and a trainable weight matrix $\boldsymbol{W} \in \mathbb{R}^{n \times m}$ as:

$$\boldsymbol{y} = \sigma(\boldsymbol{W}\boldsymbol{x}) \tag{1}$$

Its DCT-based counterpart is a layer with trainable parameters $\boldsymbol{g} \in \mathbb{R}^c$ with $c \ll n \times m$ which transforms $\boldsymbol{x} \in \mathbb{R}^m$ to $\boldsymbol{y} \in \mathbb{R}^n$ as:

$$\boldsymbol{W}_f = \mathrm{TopLeft}_{n,m}(\boldsymbol{g}) \tag{2}$$

$$\boldsymbol{W}^{(\boldsymbol{g})} = \mathrm{DCT}^{-1}(\boldsymbol{W}_f) \tag{3}$$

$$\boldsymbol{y} = \sigma(\boldsymbol{W}^{(\boldsymbol{g})}\boldsymbol{x}) \tag{4}$$

where operations in Eq. 2 and 3 correspond to a decompression process which transforms a DCT-coefficient vector $\boldsymbol{g} \in \mathbb{R}^c$ to a weight matrix $\boldsymbol{W}^{(\boldsymbol{g})} \in \mathbb{R}^{n \times m}$. First, $\mathrm{TopLeft}_{n,m}$ operation generates a sparse matrix $\boldsymbol{W}_f \in \mathbb{R}^{n \times m}$ from $\boldsymbol{g} \in \mathbb{R}^c$. $\boldsymbol{W}_f$ represents the weight matrix in the frequency domain. Its coefficients are all zero except the c highest frequency coefficients (anti-diagonals in the top left corner of the matrix) which are filled using the coefficients of $\boldsymbol{g}$. This mapping is tied to the model definition, and the model is trained end-to-end under the corresponding weight pattern in the time domain. The inverse of DCT, DCT^{-1} is then applied to the sparse frequency domain matrix $\boldsymbol{W}_f$ to obtain a dense matrix $\boldsymbol{W}^{(\boldsymbol{g})} \in \mathbb{R}^{n \times m}$. Figure 1 illustrates these intermediate steps of the decompression process.

We also note that both DCT and its inverse DCT^{-1} can be expressed as simple matrix multiplications (therefore differentiable) using fixed DCT matrices:

$$\mathrm{DCT}^{-1}(\boldsymbol{W}_f) = D_n^{-1}\boldsymbol{W}_f D_m^{-1} \tag{5}$$

where $D_n \in \mathbb{R}^{n \times n}$ and $D_m \in \mathbb{R}^{m \times m}$ are fixed orthonormalised DCT matrices.

The size c of DCT-coefficient vector is a model hyper-parameter. With $c \ll n \times m$, where $n \times m$ is the size of the original matrix, $\boldsymbol{g}$ effectively offers a compressed representation of $\boldsymbol{W}^{(\boldsymbol{g})}$. In our experiments, instead of explicitly fixing c, we fix a compression rate and derive the value of c.

In practice, we allocate separate DCT-coefficient vectors for different weight matrices (i.e. matrices which have different roles), e.g., in an RNN:

$$h_t = \sigma(\boldsymbol{W}^{(\boldsymbol{g}_i)}x_t + \boldsymbol{R}^{(\boldsymbol{g}_r)}h_{t-1}) \tag{6}$$

$\boldsymbol{W}^{(\boldsymbol{g}_i)}$ and $\boldsymbol{R}^{(\boldsymbol{g}_r)}$ are generated by parameters $\boldsymbol{g}_i$ and $\boldsymbol{g}_r$ respectively via two separate transformations. Similarly for a Long Short-Term Memory (LSTM) (Hochreiter & Schmidhuber, 1997),

separate DCT parameters are allocated for each gate and each weight matrix and decompressed separately.

This method has been previously investigated for RNNs in evolutionary settings (Koutník et al., 2010a;b) and for small feedforward NNs in supervised settings of toy binary classification tasks (Fabisch et al., 2013). We revisit this method for RNNs in a language modelling setting. In the experimental Sec. 3, we evaluate the performance of LSTM-RNNs with DCT-encoded weights under different compression rates.

Fast weight version. Once we have a compact representation of weight matrices, an attractive extension is to generate the weights in the compressed space depending on the context by parameterising it by other neural networks (Schmidhuber, 1991; 1992b; 1993d; Gomez & Schmidhuber, 2005; Ha et al., 2017). This approach can be considered as an alternative to outer-product based fast weights, with a potentially full rank weight matrix update at each time step.

We define our main network to be a regular RNN and parameterise its two weight matrices: input-to-hidden weights and hidden-to-hidden weights, using an LSTM-RNN. An input vector $\boldsymbol{x}_t \in \mathbb{R}^m$ at time step t is mapped to a recurrent hidden state $\boldsymbol{h}_t \in \mathbb{R}^n$ by:

$$[\boldsymbol{g}_t^i, \boldsymbol{g}_{t-1}^h], \boldsymbol{c}_t^i = \mathrm{LSTM}([\boldsymbol{g}_{t-1}^i, \boldsymbol{g}_{t-1}^h], \boldsymbol{c}_{t-1}^i, \boldsymbol{x}_t,) \tag{7}$$

$$\boldsymbol{W}_t^{(\boldsymbol{g}_t^i)} = \mathrm{DCT}^{-1} \circ \mathrm{TopLeft}_{n,m}(\boldsymbol{g}_t^i) \tag{8}$$

$$\boldsymbol{R}_t^{(\boldsymbol{g}_t^h)} = \mathrm{DCT}^{-1} \circ \mathrm{TopLeft}_{n,n}(\boldsymbol{g}_t^h) \tag{9}$$

$$\boldsymbol{h}_t = \sigma(\boldsymbol{W}_t^{(\boldsymbol{g}_t^i)} \boldsymbol{x}_t + \boldsymbol{R}_t^{(\boldsymbol{g}_t^h)} \boldsymbol{h}_{t-1}) \tag{10}$$

where $\circ$ denotes the composition. Figure 2 illustrates the corresponding model.

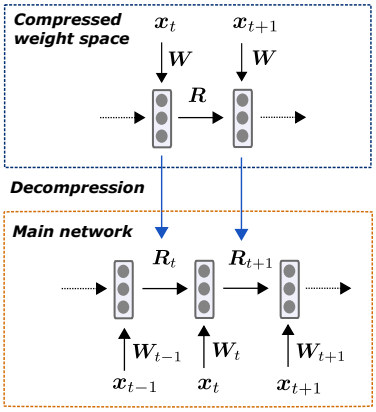


Figure 2: Illustration of a network with weight matrices parametrised by an RNN in the compressed space (weight generation for $\boldsymbol{W}_t$ as well as super-scripts on weights are omitted for readability).

3 Experimental results

We train character level LSTM-RNN language models on the standard `enwik8` dataset (Hutter, 2012) which contains about 90 M characters for training with a character level vocabulary size of 205. We follow the baseline configuration of Merity et al. (2018) which has three LSTM-RNN layers with dimensions of 1840 for the two first layers and 400 for the third one, with an embedding layer of size 400 whose parameters are shared for input and output embeddings. In the DCT models, we encode each weight matrix in an LSTM layer by DCT as described in Sec. 2. We apply the same compression rate for each weight matrix. All models are trained with the same training configuration. To initialise the model parameters which are DCT-coefficient vectors, we first generate and initialise a weight matrix in the time domain, and apply the DCT compression to obtain the corresponding initialisation, which we found to work well in practice.

Table 1 presents the effects of different compression rates on model performance. We obtain degradation of about 14% relative in bits-per-character with a compression rate of 90%. This has to be compared to the performance of baseline models with smaller layer sizes (the simplest way to reduce the model size) presented in Table 2. The performance of the baseline model with a layer size of 465 and an embedding size of 400 from Table 2 with 4.8 M parameters (the one in the second to last row in Table 2) is similar to the performance of the DCT model with a compression rate of 90% in Table 1. Apparently a large model with compressed weights does not improve much over a small model with a comparable number of parameters. However, reducing the number of parameters by only reducing the recurrent layer size without reducing the embedding dimension has its limits. For example, with an LSTM layer size of 16 (which is extremely small) and an embedding size of 400, the corresponding model size is 780 K. Further parameter reduction requires to reduce the

Table 1: Bits-per-character performance of **DCT-based** character level LSTM language models on `enwik8`. Results are shown for different levels of compression rate. The embedding size is 400 in all cases.

Compression Rate	# Params in M	Valid	Test
0	75.0	1.28	1.29
0.5	23.7	1.33	1.33
0.7	14.2	1.36	1.36
0.8	9.5	1.40	1.40
0.9	**4.8**	1.43	1.44
0.99	567 K	1.75	1.74
0.999	144 K	2.23	2.20

Table 2: Bits-per-character performance of **baseline** character level LSTM language models on `enwik8` with different number of parameters.

Embed. Size	Layer Size	# Params in M	Valid	Test
64	672	5.8	1.47	14.9
400	16	780 K	1.74	1.75
	465	**4.8**	1.46	1.47
	512	5.5	1.46	1.47

Table 3: Bits-per-character performance on `enwik8` of DCT-based models with **fast weights**.

Compression Rate	Embed. Size	Layer Size	# Params in M	Valid	Test
0.9	64	64	5.7	1.59	1.60
0.99	128	256	13.2	1.66	1.67

embedding size. In contrast, the DCT-based model can be configured to have an arbitrarily small number of parameters, without requiring to modify other model hyper-parameters, as shown in the lower part of Table 1.

We also present preliminary experimental results for the fast weight variant. Table 3 presents two models. The small 5.7 M-parameter model has an embedding size of 64 and two RNN layers of 64 dimension each. With a DCT compression rate of 90%, the number of parameters in each RNN layer is 812. We directly parameterise them through an LSTM-RNN with a hidden size of 812. Its current performance lags behind that of the baseline model (5.5 M-parameter model in the last row) from Table 2 with a comparable number of parameters. We are currently investigating the impact of tuning the dropout rate in the weight generator LSTMs and plan to update our result section in the final version. We also observe that the compression rate has an impact on the performance of fast weight models. The model with 90% compression rate outperforms another model with a rate of 99% while the former has less parameters.

4 Discussion and Future Work

We revisited DCT-based indirect encodings of neural network weight matrices in a character level language modelling task. Our experiments seem to indicate that at least in the case of language modelling, large architectures with DCT-compressed weights don't improve much over small networks with a similar number of parameters. However, we also showed that the DCT-based encoding allows for controlling the number of parameters in each layer flexibly and independently of hyper-parameters in other layers. This property may be of practical interest in some applications.

What is the best way of encoding neural network weights? This fundamental question remains unanswered, but we hope our work will help trigger a discussion of this topic.

Keeping in mind our goal of building highly adaptive language models (Lazaridou et al., 2021; Irie et al., 2018), our fast weight variant can be extended in various ways. In particular, the weight generator networks could be augmented by additional inputs for error signals (Schmidhuber, 1993b; Hochreiter et al., 2001; Rocki, 2016) to facilitate the weight update generation process. Future work will evaluate our models on tasks which explicitly require context-adaptive behaviour, e.g. texts containing multiple domain shifts.

REFERENCES

Nasir Ahmed, T. Natarajan, and Kamisetty R Rao. Discrete cosine transform. *IEEE Transactions on Computers*, 100(1):90–93, 1974.

Jimmy Ba, Geoffrey E Hinton, Volodymyr Mnih, Joel Z Leibo, and Catalin Ionescu. Using fast weights to attend to the recent past. In *Proc. Advances in Neural Information Processing Systems (NIPS)*, pp. 4331–4339, Barcelona, Spain, December 2016.

Alexander Fabisch, Yohannes Kassahun, Hendrik Wöhrle, and Frank Kirchner. Learning in compressed space. *Neural networks*, 42:83–93, 2013.

Francesco Faccio, Louis Kirsch, and Jürgen Schmidhuber. Parameter-based value functions. In *Int. Conf. on Learning Representations (ICLR)*, Virtual only, May 2021.

Faustino Gomez, Jan Koutník, and Jürgen Schmidhuber. Compressed network complexity search. In *Proc. Int. Conf. on Parallel Problem Solving from Nature*, pp. 316–326, Taormina, Italy, September 2012.

Faustino J. Gomez and Jürgen Schmidhuber. Evolving modular fast-weight networks for control. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, pp. 383–389, Warsaw, Poland, September 2005.

David Ha, Andrew Dai, and Quoc V Le. Hypernetworks. In *Int. Conf. on Learning Representations (ICLR)*, Toulon, France, April 2017.

Jean Harb, Tom Schaul, Doina Precup, and Pierre-Luc Bacon. Policy evaluation networks. Preprint arXiv:2002.11833, 2020.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Sepp Hochreiter, A. Steven Younger, and Peter R. Conwell. Learning to learn using gradient descent. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, volume 2130, pp. 87–94, Vienna, Austria, August 2001.

Marcus Hutter. The human knowledge compression contest. *URL http://prize. hutter1. net*, 6, 2012.

Kazuki Irie, Shankar Kumar, Michael Nirschl, and Hank Liao. RADMM: Recurrent adaptive mixture model with applications to domain robust language modeling. In *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 6079–6083, Calgary, Canada, April 2018.

Jan Koutník, Faustino Gomez, and Jürgen Schmidhuber. Evolving neural networks in compressed weight space. In *Proc. Conference on Genetic and Evolutionary Computation (GECCO)*, pp. 619–626, 2010a.

Jan Koutník, Faustino Gomez, and Jürgen Schmidhuber. Searching for minimal neural networks in fourier space. In *Proc. Conf. on Artificial General Intelligence*, Lugano, Switzerland, March 2010b.

Jan Koutník, Juergen Schmidhuber, and Faustino Gomez. A frequency-domain encoding for neuroevolution. *Preprint arXiv:1212.6521*, 2012.

Angeliki Lazaridou, Adhiguna Kuncoro, Elena Gribovskaya, Devang Agrawal, Adam Liska, Tayfun Terzi, Mai Gimenez, Cyprien de Masson d'Autume, Sebastian Ruder, Dani Yogatama, et al. Pitfalls of static language modelling. *Preprint arXiv:2102.01951*, 2021.

Stephen Merity, Nitish Shirish Keskar, and Richard Socher. An analysis of neural language modeling at multiple scales. *Preprint arXiv:1803.08240*, 2018.

Kamil M Rocki. Surprisal-driven feedback in recurrent networks. *Preprint arXiv:1608.06027*, 2016.

Imanol Schlag and Jürgen Schmidhuber. Gated fast weights for on-the-fly neural program generation. In *NIPS Metalearning Workshop*, Long Beach, CA, USA, December 2017.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to recurrent nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, March 1991.

Jürgen Schmidhuber. Steps towards "self-referential" learning. Technical Report CU-CS-627-92, Dept. of Comp. Sci., University of Colorado at Boulder, November 1992a.

Jürgen Schmidhuber. Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, 4(1):131–139, 1992b.

Jürgen Schmidhuber. An introspective network that can learn to run its own weight change algorithm. In *Proc. IEE Int. Conf. on Artificial Neural Networks*, pp. 191–195, Brighton, UK, May 1993a.

Jürgen Schmidhuber. A self-referential weight matrix. In *Proc. Int. Conf. on Artificial Neural Networks (ICANN)*, pp. 446–451, Amsterdam, Netherlands, September 1993b.

Jürgen Schmidhuber. A neural network that embeds its own meta-levels. In *Proc. IEEE Int. Conf. on Neural Networks (ICNN)*, San Francisco, CA, USA, March 1993c.

Jürgen Schmidhuber. Reducing the ratio between learning complexity and number of time varying variables in fully recurrent net s. In *International Conference on Artificial Neural Networks (ICANN)*, pp. 460–463, Amsterdam, Netherlands, September 1993d.

Vincent Sitzmann, Julien Martel, Alexander Bergman, David Lindell, and Gordon Wetzstein. Implicit neural representations with periodic activation functions. In *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, pp. 7462–7473, Virtual only, 2020.

Rupesh Kumar Srivastava, Jürgen Schmidhuber, and Faustino J. Gomez. Generalized compressed network search. In *Proc. Int. Conf. on Parallel Problem Solving from Nature*, pp. 337–346, Taormina, Italy, September 2012.

Thomas Unterthiner, Daniel Keysers, Sylvain Gelly, Olivier Bousquet, and Ilya Tolstikhin. Predicting neural network accuracy from weights. Preprint arXiv:2002.11448, 2020.

Sjoerd van Steenkiste, Jan Koutník, Kurt Driessens, and Jürgen Schmidhuber. A wavelet-based encoding for neuroevolution. In *Proc. Genetic and Evolutionary Computation Conference (GECCO)*, pp. 517–524, Denver, CO, USA, July 2016.