Training DNNs in O(1) memory with MEM-DFA using Random Matrices

Tien Chu[1]
chutien@protonmail.com
Kamil Mykitiuk[1]
k.mykitiuk@student.uw.edu.pl
Miron Szewczyk[1]
mj.szewczyk2@student.uw.edu.pl
Adam Wiktor[1]
adam.wiktor97@gmail.com
Zbigniew Wojna[2]
zbigniewwojna@gmail.com

[1] Warsaw University
[2] Tensorflight, Inc.

Abstract

This work presents a method for reducing memory consumption to a constant complexity when training deep neural networks. The algorithm is based on the more biologically plausible alternatives of the backpropagation (BP): direct feedback alignment (DFA) and feedback alignment (FA), which use random matrices to propagate error. The proposed method, memory-efficient direct feedback alignment (MEM-DFA), uses higher independence of layers in DFA and allows avoiding storing at once all activation vectors, unlike standard BP, FA, and DFA. Thus, our algorithm's memory usage is constant regardless of the number of layers in a neural network. The method increases the computational cost only by a constant factor of one extra forward pass.

The MEM-DFA, BP, FA, and DFA were evaluated along with their memory profiles on MNIST and CIFAR-10 datasets on various neural network models. Our experiments agree with our theoretical results and show a significant decrease in the memory cost of MEM-DFA compared to the other algorithms.

1 Introduction

State of the art CNN methods strive for high memory usage as in many scenarios, the deeper networks with lager inputs achieve better results. This is exemplified by many successful models such as ResNet [12], Inception [25], or SENet [13].

The backpropagation seems to be an irreplaceable algorithm when it comes to computational cost. However, it imposes substantial memory costs. Apart from storing the model's parameters, the backpropagation stacks all intermediate activation vectors in the forward pass, necessary for gradient calculations during the backward pass.

The memory problem becomes even more apparent for tasks involving processing high-resolution images. For example, the hardware limitations require a decrease in the quality

of images, train the model in small batches, reduce the model's size, or increase the training time.

It is believed the backpropagation is biologically implausible to perform in the brain [17]. Feedback alignment [16, 17] is an alternative method that uses random matrices to propagate error instead of transposed weight matrices. Direct feedback alignment [2, 20], on the other hand, propagates error from the last layer directly to each layer through the random matrix.

This work proposes a method for reducing memory consumption called MEM-DFA. It is preceded by a detailed description of the backpropagation, FA and DFA. Here, we present the results from six experiments with different models on MNIST and CIFAR-10 datasets. We discuss the results and derive conclusions.

The contributions of this work are the following:

- Review and analyze memory usage footprint of BP, FA, DFA, and MEM-DFA algorithms.
- Propose a new method for training DNNs called MEM-DFA that consumes O(1) memory regardless of the number of layers.

2 Related works

2.1 Memory optimization

The first group of related works addresses the memory optimization of neural networks. We can divide them into three categories:

- reducing the number of stored activation vectors at the peak by applying checkpointing method [4, 8, 11, 21],
- neural networks' architecture modifications to decrease the number of activation vectors required in the backpropagation [10, 22, 27],
- reducing the number of bits used to represent activation vectors [5, 6, 14],

The checkpointing method stores a subset of the activations and recalculates the missing activations during backpropagation. This is a trade-off between memory and computational cost. Depending on the algorithm of selection of checkpoints, the memory usage ranges from linear to constant, but the time complexity ranges from linear to quadratic.

Merging batch normalization with activation function [22] is an example of reducing the number of sored activation vectors to decrease memory usage. Another example is a modification of ResNets, called RevNets, by introducing reversible blocks in place of residual blocks [10, 27].

Reducing the number of bits representing values of matrices in a neural network can decrease memory usage and increase calculations speed [5, 14]. In [6], the authors trained a neural network using only one-bit values.

2.2 Biologically motivated methods with random matrices

This paper is inspired by the line of work trying to overcome backpropagation's biological implausibility. Feedback alignment [16, 17] and direct feedback alignment [20] relax information constraints of backpropagation by using random matrices to propagate error. Both

FA and DFA showed promising results on the MNIST dataset achieving close to backpropagation accuracy [16, 17, 20]. FA algorithm in [16] achieves even better results by preserving the signs of the matrices. In [9], the authors try to explain the effectiveness and limitations of DFA.

The works [3, 7, 19, 28] try to scale FA, DFA, and other biologically plausible algorithms such as Target Propagation [15] on large datasets. The authors of [28] show that preserving sign symmetry as in [16] allows scaling FA to satisfying results. The work of [7] shows that sparse matrices of full rank allow training DFA on large datasets by reducing the memory cost of big matrices. Such matrices are a significant obstacle in DFA training on large datasets as observed in [7, 19, 19, 28, 28]. The [7] also transfer BP trained models to DFA training showing comparable results to BP. The sparse matrices can be used in parallel with the MEM-DFA described in this work.

Biologically inspired algorithms are an active area of research, and many works try to improve the performance of biologically motivated methods. Our work does not focus on increasing the accuracy of FA or DFA but proposes an algorithm based on those methods, which reduces memory usage significantly during deep neural networks training.

3 Backpropagation

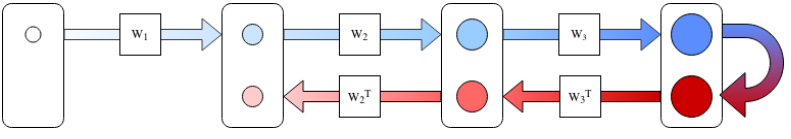


Figure 1: Backpropagation. The gradient is propagated by transposed weight matrices.

This section defines the notation and terminology, starting from the backpropagation algorithm for a sequential network with n layers presented in Figure 1. By layer, we usually mean an affine transformation together with one or more subsequent nonlinear operations. The operations within a layer are often called sublayers.

We denote by x or a_0 an input vector, y an expected output, f an activation function, C a cost function, W_i a weight matrix and b_i a bias where bottom index $i \in \{1..n\}$ means the i-th layer. Forward calculations can be described by

$$z_i = W_i a_{i-1} + b_i \tag{1}$$

$$a_i = f(z_i) \tag{2}$$

where z_i is a weighted vector and a_i is an activation vector. We call the vector a_n the model's prediction. In the i-the layer, we consider gradients with respect to weights W_i, bias b_i and vector z_{i-1} and we denote them by $\frac{\delta C}{\delta W_i}$, $\frac{\delta C}{\delta b_i}$ and $\frac{\delta C}{\delta z_{i-1}}$, respectively.

For simplicity, the following cost function is considered

$$C = \frac{1}{2}||y - a_n||^2 . \tag{3}$$

Hence, gradient with respect to a_n, also called error, is

$$\frac{\delta C}{\delta a_n} = a_n - y . \tag{4}$$

By using the chain rule and by working backward it is possible to effectively calculate gradients with respect to weights and bias [23]. The process can be described inductively. Firstly, based on the gradient with respect to the weighted vector z_{i+1} we calculate the gradient with respect to a_i through weight matrix W_{i+1}

$$\frac{\delta C}{\delta a_i} = W_{i+1}^T \frac{\delta C}{\delta z_{i+1}}. \tag{5}$$

Having $\frac{\delta C}{\delta a_i}$, we propagate through activation function and get gradient with respect to z_i in result

$$\frac{\delta C}{\delta z_i} = \frac{\delta C}{\delta a_i} \odot f'(z_i). \tag{6}$$

Thus we calculate the gradients with respect to weights and bias

$$\frac{\delta C}{\delta W_i} = \frac{\delta C}{\delta z_i} a_{i-1}^T, \tag{7}$$

$$\frac{\delta C}{\delta b_i} = \frac{\delta C}{\delta z_i}. \tag{8}$$

These gradients allow us to update weights and bias, for example, by using stochastic gradient descent method.

The main reason for memory consumption in the backpropagation algorithm is the need to store activation vectors. Hence the memory usage is growing with the number of layers. More specifically, we need z_i and a_{i-1} in order to calculate gradients in i-the layer as shown in equations (6) and (7). However, before calculating gradients in i-th layer, the gradients in $i+1$-th layer have to be calculated first. Therefore, before we can forget z_i and a_{i-1} we need to use z_{i+1} and a_i in calculations of gradients in the $i+1$-th layer. Thus, in the stack's peak moment, all activation and weighted vectors are kept in memory to calculate gradients in each layer efficiently.

4 Feedback Alignment

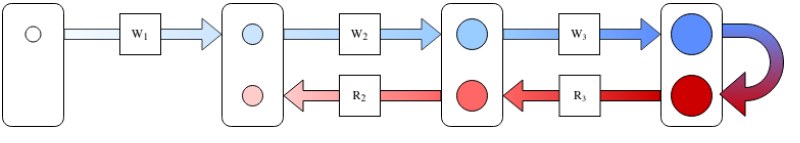


Figure 2: Feedback Alignment. The error is propagated by random matrices instead transposed weight matrices

The Feedback Alignment (FA) introduced in [16, 17] and presented in Figure 2 is a biologically-motivated algorithm. It targets the conjectured implausibility of the symmetric backward connectivity pattern in the backpropagation, i.e., the use of transpose of a weight matrix in the backward phase. The authors propose random matrices to propagate error instead and show that such an algorithm provides learning.

Since we propagate error using random matrices instead of transposed weight matrix, in FA we have vectors which are not strictly gradients but which play role of the BP's gradients $\frac{\delta C}{\delta v}$. We denote them by δv. The FA algorithm differs from BP by the equation (5).

$$\delta a_i = R_{i+1}\delta z_{i+1} \tag{9}$$

where R_i is random matrix with the same dimensions as W_i^T. The R_i matrix can be generated once at the beginning or newly generated for each iteration. Moreover, by preserving the signs of matrix values in R_i, we can achieve better results as shown in [16].

Although there is information alleviation between the forward and backward pass, the memory usage is roughly the same as for backpropagation since we still need to store all intermediate vectors.

5 Direct Feedback Alignment

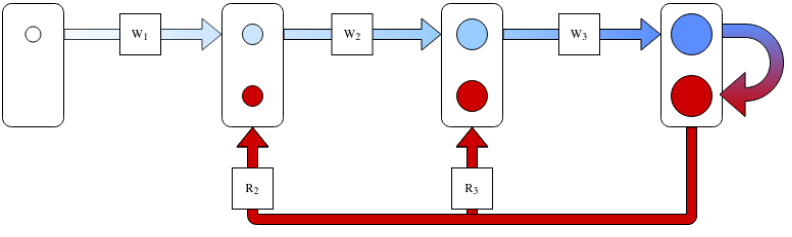


Figure 3: Direct Feedback Alignment. The error is propagated directly from the last year by random matrices.

Direct feedback alignment (DFA) described in [2, 20] further develops the FA algorithm's idea. A random matrix in DFA skips the dependence on the subsequent layer - it directly propagates the error from the last layer, as illustrated in figure 3. Hence, the R_i matrix has the input dimension of the model's output and output dimension as W_i^T. The DFA differs from BP and FA respectively by the equations (5) and (9)

$$\delta a_i = R_{i+1}\delta a_n \tag{10}$$

DFA still needs to store all intermediate vectors at the peak. Therefore it has similar memory complexity as BP and FA. However, DFA allows parallelizing some operations in the backward pass.

6 Our method MEM-DFA

Let us observe that since an error in DFA is propagated directly through the random matrix from the last layer, the gradient estimations are independent of gradients from the layers after. In other words, we could do the backpropagation independently from each layer. So assuming we have input to the first operation in the layer i and error propagated through random matrix R_i, we could proceed with backpropagation within that layer without waiting for the gradient from the layer after.

dataset	exp. no.	model	BP	FA	DFA / MEM-DFA
MNIST	I	3 x FC	97,6%	97,1%	97,1%
MNIST	II	2 x Conv + 2 x FC	99,0%	98,7%	98,9%
CIFAR-10	I	2 x Conv + 2 x FC	70,1%	63,7%	64,3%
CIFAR-10	II	3 x Conv + 2 x FC	74,8%	51,5%	51,2%
CIFAR-10	III	VGG-16	77,8%	55,0%	54,8%

Table 1: The accuracy of the models trained with different learning methods on MNIST and CIFAR-10.

Our method makes use of these observations. During the forward pass, we do not store any intermediate vectors. Instead of one backward pass, we alternate the second forward pass and backward pass in the following way. We do forward pass within the layer, storing the intermediate vectors. Then we propagate the error directly to the last operation in that layer and then proceed with backpropagation within that layer. We do it starting from the first layer to use the output of the layer i as input to the layer $i+1$.

We called this method MEM-DFA. The algorithm is illustrated in figure 4. We can describe it as follows.

Algorithm 1 MEM-DFA

Calculate a_n in the forward pass without storing intermediate vectors apart from input a_0.
Calculate δa_n.
for $i \in \{1..n\}$ **do**
1. Using a_{i-1} calculate a_i in the forward pass within the i-th layer storing intermediate vectors.
2. Calculate δa_i.
3. Backpropagate locally within i-th layer and forget intermediate vectors apart from the a_i.

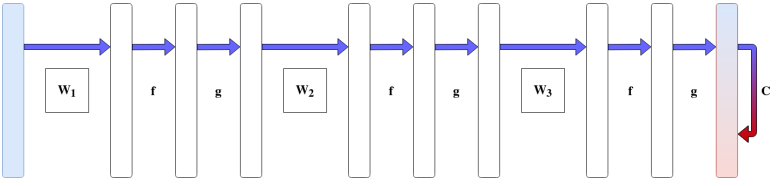
end for

Our method stores at most $k+1$ intermediate vectors, where k is the number of operations in a layer. Usually, k is insignificant compared to the number of layers. Thus, MEM-DFA achieves constant memory cost, regardless of the number of weight matrices in a neural network.

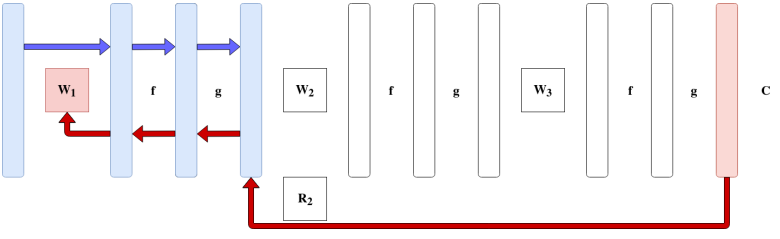
The computational cost of MEM-DFA is equivalent to two forward passes plus one backward pass of the DFA, FA, or BP. All operations are the same as in DFA but in a different order or eventually recalculated. The MEM-DFA method is numerically equivalent to DFA.

7 Experiments

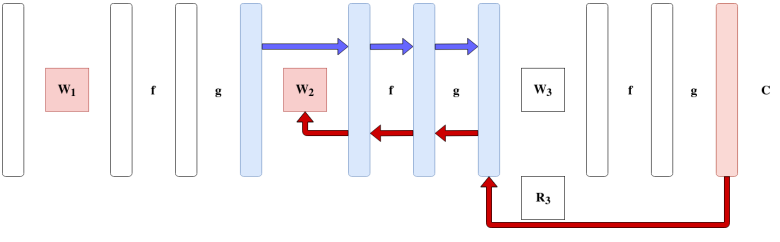
Our experiments were conducted on MNIST and CIFAR-10 with various neural network models. Each model was trained with BP, FA, DFA, and MEM-DFA methods. We measured the memory usage and computation time per training iteration. The accuracy of our FA and DFA implementations were close to those from the original works [16, 17, 20]. The results are presented in Table 1. Throughout the experiments, we used the ReLU activation function and softmax cross-entropy cost function. Weight updates followed the stochastic gradient descent algorithm.



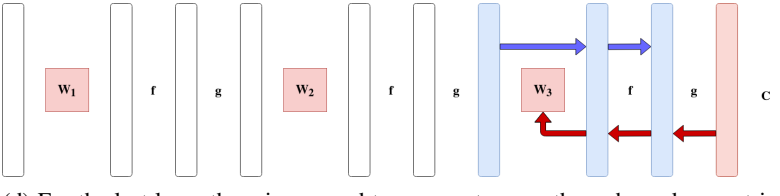
(a) During the first phase do not store vectors apart from input and output vectors.



(b) Propagate error to the first layer through random matrix, calculate intermediate vectors within layer and store them. Then proceed with local backpropagation within layer.



(c) Continue this process for the next layers by using the calculated activation vector from the previous layers.

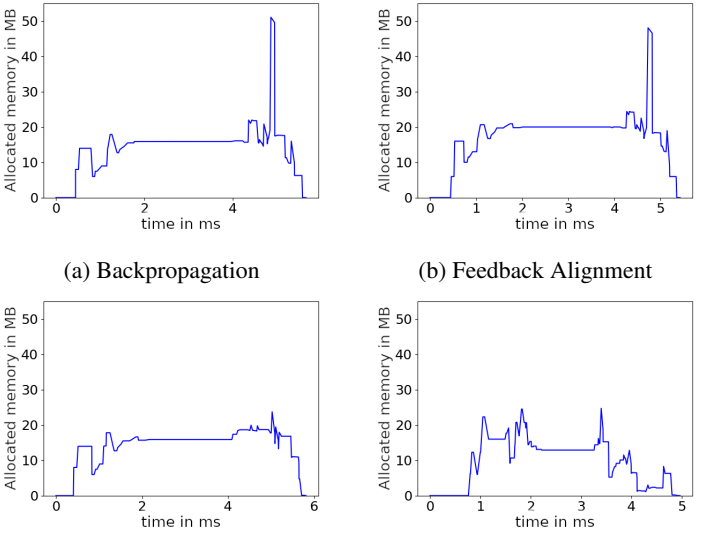


(d) For the last layer there is no need to propagate error through random matrix, just backpropagate within that layer.

Figure 4: MEM-DFA algorithm. Each layer is composed of W_i, f, and g operations. We can see that we do exactly two full forward and one full backward passes, but without the need to store all intermediate activations.

The first experiment has been conducted on MNIST with a model consisted of 3 fully connected layers of size 100, 30, and 10, respectively. We trained the model using a learning rate of 0.01, batch size of 100, and 100 epochs.

In the next experiment on MNIST, we trained a convolutional neural network. The first convolution layer contained 20 filters of size 5x5, and the second layer 50 filters of the same size. Additionally, the max-pooling of size 2x2 and stride 2x2 was applied after each Conv layer. In the end, there were two fully connected layers of sizes 500 and 10. The architecture is from [16]. We trained for 150 epochs with a learning rate of 0.005. Measurements of memory usage of this model are compiled in figure 5.



(a) Backpropagation (b) Feedback Alignment

(c) Direct Feedback Alignment (d) MEM-DFA

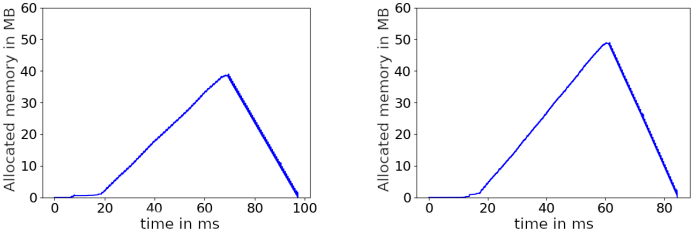
Figure 5: Memory usage of convolution network on MNIST with 100 batch size.

The final experiment on MNIST was conducted on a larger neural network with 50 fully connected layers, each of size 500. We trained with a batch of size 100. Figure 6 presents network's allocated memory with BP, FA, DFA, and MEM-DFA algorithms. This model was used only for memory measurements.

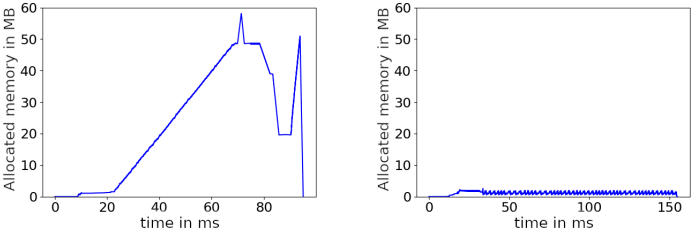
The next three experiments were conducted on CIFAR-10. In the first, we used the same CNN as before. Hyperparameters used were also very similar.

In the second experiment, we used a model with convolutions with 32 filters of size 5x5 in the first layer and 64 filters with the same dimensions in the next two. In the end, two FC layers with 128 and 10 sizes were used. The max and two average pooling of size 2x2 and stride 2x2 were put after respective Conv layers. The model also comes from [16].

The VGG-16 network architecture [24] was used in the final experiment on CIFAR-10. The models were trained using batch size 200 for 80 epochs with a learning rate of 0.001 for backpropagation and 0.00005 for FA and MEM-DFA. The memory measurements for various methods on VGA-16 are shown in figure 7.

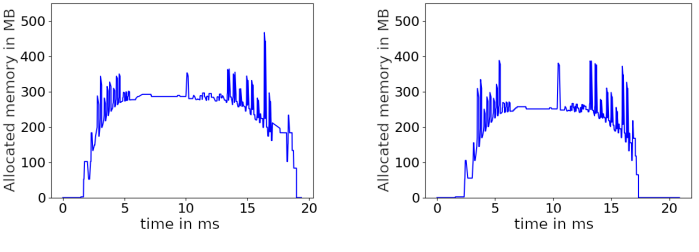


(a) Backpropagation (b) Feedback Alignment

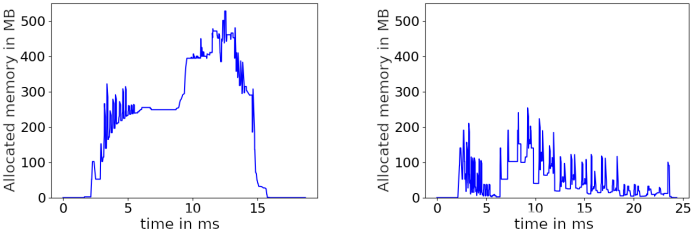


(c) Direct Feedback Alignment (d) MEM-DFA

Figure 6: Memory usage of the model with 50 FC layers on MNIST with 100 batch size.



(a) Backpropagation (b) Feedback Alignment



(c) Direct Feedback Alignment (d) MEM-DFA

Figure 7: Memory usage of VGG-16 on CIFAR-10 with 200 batch size.

8 Measuring memory usage

8.1 Memory management in TensorFlow framework

TensorFlow 1.12 [1] has been used to create our neural network models. One of the main features of this technology is a static definition of a computation graph. The user defines the sequence of operations to be performed on input data at a later time. The computation graph is an abstract representation of the computation process, which allows preserving the relationship between input data and the results of applied operations. Thanks to it, it is possible to perform the symbolic computation of gradients. Such static construction allows the analysis, optimization, and compilation of the computation graph to utilize resources efficiently.

TensorFlow uses many techniques to minimize needed time to perform computation and used memory. XLA compiler [26] can replace consecutively appearing operations by optimized equivalent. It is also possible to detect repeating sequences of operations and introduce a common copy.

The memory management system of TensorFlow is implementing best-fit with the coalesce algorithm. It is reducing the fragmentation of memory and overall usage. TensorFlow chooses the smallest block during memory allocation, which still satisfies memory requirements from the memory block pool of predefined sizes. The block is kept alive until its reference counter has a positive value. When the counter becomes zero, all computations needed to access this particular memory are finished. When deallocating, the adjacent memory blocks are checked whether they are free, and if possible, they fuse into one chunk of a bigger size. Thus memory fragmentation is reduced.

Research on the memory management systems [18] has shown that calculating each memory fragment's liveliness based on the computation graph allows using a memory swapping strategy between GPU DRAM and RAM limiting peak usage of VRAM without sacrificing computation time.

Static definition of the computation graph allows for repeating calculations whose computational cost is smaller than keeping its results in memory for a prolonged time. Checkpointing algorithm introduced in [4, 8] frees memory blocks if re-computation costs are below some specified threshold despite the positive reference counter for that memory block.

This research is limited to native TensorFlow optimizations. It allows measuring performance in a relatively easy to reconstruct manner. Many of the described above methods are perpendicular to each other and could be used simultaneously to limit memory usage significantly.

8.2 Measuring memory usage

The TensorFlow framework allows performing computations on CPU and GPU units. Unfortunately, its implementation cannot measure in detail allocated and deallocated RAM since messages created by allocator objects are stripped off requested memory size values. Tools such as htop, ps, valgrind, or docker containers with limited memory did not bring satisfactory results. TensorFlow is using a lazy strategy for deallocating memory to reduce interactions with the underlying operating system. When freeing a memory fragment, it is put back to memory poll without reporting to the operating system limiting these operations' overhead. Above all, CPU computations are usually used for fast prototype solutions. Thus this paper focused on measuring memory usage on GPU.

TensorFlow has a built-in operation to measure peak memory usage, which gives comparable results to other methods. Detailed time analysis is necessary to measure the proposed algorithm's performance, so this method was not used.

In the conducted experiments, Nvidia graphics cards have been used. Nvidia-Smi toolkit for this hardware allows observing card memory usage in real-time. Regrettably, TensorFlow is using a lazy strategy in memory management by maintaining an internal buffer of memory. It reduces interactions with firmware and operating system, which boosts the speed of running neural networks but blocks proper measurements from the outside tools like Nvidia-Smi, which shows constant memory usage during runtime and one deallocation at the end of it.

However, the GPU allocator in TensorFlow is emitting messages about the requested memory blocks. It is possible to extract information about total allocated memory in a given time and a chronological list of performed operations from the memory allocator's logs. To measure memory usage in our experiments, we adopted an existing code from [21] on TensorFlow's memory optimization, which performs such data manipulations.

9 Discussion

Even on the shallow convolutional network on MNIST the memory decrease of MEM-DFA is apparent from figure 5. At the same time, the computational cost of one training iteration stays close to the other methods. The decrease in memory usage of MEM-DFA is further exemplified by VGG-16 on CIFAR-10, as shown by figure 7. The computational time increased from about 20 ms to about 25 ms here. Moreover, we can also observe on this Figure the parallelization in the backward pass for DFA.

The memory usage and computational time of 50 layers fully connected model entirely agree with our theoretical expectations. In Figure 6, we can see precisely the linear memory allocation during the forward pass and linear memory freeing during the backward pass of BP, FA, and partially DFA. In contrast, the memory usage of MEM-DFA stays roughly constant throughout the calculations. The computation time is about 50% longer as predicted.

10 Conclusion

Biologically inspired algorithms in deep neural networks are an immensely active and fascinating area of research. Our work shows that they could provide insights regarding the nature of our brain and practical optimizations.

The proposed MEM-DFA algorithm allows achieving constant memory complexity regardless of the number of linear operations in a neural network by using the higher independence between layers in DFA. The method increases the computational cost by a constant factor equal to one extra forward pass. Our experiments confirmed our theoretical results.

References

[1] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. TensorFlow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[2] Pierre Baldi, Peter Sadowski, and Zhiqin Lu. Learning in the machine: Random backpropagation and the deep learning channel. *Artificial intelligence*, 260:1–35, 2018.

[3] Sergey Bartunov, Adam Santoro, Blake Richards, Luke Marris, Geoffrey E Hinton, and Timothy Lillicrap. Assessing the scalability of biologically-motivated deep learning algorithms and architectures. In *Advances in Neural Information Processing Systems*, pages 9368–9378, 2018.

[4] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[5] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015.

[6] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to +1 or -1. *arXiv preprint arXiv:1602.02830*, 2016.

[7] Brian Crafton, Abhinav Parihar, Evan Gebhardt, and Arijit Raychowdhury. Direct feedback alignment with sparse connections for local learning. *Frontiers in neuroscience*, 13:525, 2019.

[8] Jianwei Feng and Dong Huang. Cutting down training memory by re-fowarding. *arXiv preprint arXiv:1808.00079*, 2018.

[9] Justin Gilmer, Colin Raffel, Samuel S Schoenholz, Maithra Raghu, and Jascha Sohl-Dickstein. Explaining the learning dynamics of direct feedback alignment. *ICLR 2017 workshop*, 2017.

[10] Aidan N Gomez, Mengye Ren, Raquel Urtasun, and Roger B Grosse. The reversible residual network: Backpropagation without storing activations. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 2214–2224. Curran Associates, Inc., 2017.

[11] Audrunas Gruslys, Remi Munos, Ivo Danihelka, Marc Lanctot, and Alex Graves. Memory-efficient backpropagation through time. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29*, pages 4125–4133. Curran Associates, Inc., 2016.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[13] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7132–7141, 2018.

[14] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[15] Dong-Hyun Lee, Saizheng Zhang, Asja Fischer, and Yoshua Bengio. Difference target propagation. In *Joint european conference on machine learning and knowledge discovery in databases*, pages 498–515. Springer, 2015.

[16] Qianli Liao, Joel Z Leibo, and Tomaso Poggio. How important is weight symmetry in backpropagation? In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[17] Timothy P Lillicrap, Daniel Cownden, Douglas B Tweed, and Colin J Akerman. Random feedback weights support learning in deep neural networks. *arXiv preprint arXiv:1411.0247*, 2014.

[18] Chen Meng, Minmin Sun, Jun Yang, Minghui Qiu, and Yang Gu. Training deeper models by GPU memory optimization on TensorFlow. *Proc. of ML Systems Workshop in NIPS*, 2017.

[19] Theodore H Moskovitz, Ashok Litwin-Kumar, and LF Abbott. Feedback alignment in deep convolutional networks. *arXiv preprint arXiv:1812.06488*, 2018.

[20] Arild Nøkland. Direct feedback alignment provides learning in deep neural networks. In *Advances in neural information processing systems*, pages 1037–1045, 2016.

[21] OpenAI. Gradient checkpointing. https://github.com/openai/gradient-checkpointing, 2017.

[22] Samuel Rota Bulò, Lorenzo Porzi, and Peter Kontschieder. In-place activated batchnorm for memory-optimized training of dnns. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5639–5647, 2018.

[23] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[24] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR 2015 Workshop*, 2015.

[25] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2818–2826, 2016.

[26] XLA team. Accelerated linear algebra compiler. https://www.tensorflow.org/xla, 2017.

[27] Sil C van de Leemput, Jonas Teuwen, and Rashindra Manniesing. Memcnn: a framework for developing memory efficient deep invertible networks. *ICLR 2018 Workshop*, 2018.

[28] Will Xiao, Honglin Chen, Qianli Liao, and Tomaso A. Poggio. Biologically-plausible learning algorithms can scale to large datasets. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*, 2019.