Benchmarking Decoupled Neural Interfaces with Synthetic Gradients

Ekaba O. Bisong∗
Department of Computer Science
Carleton University
Ottawa, ON K1S 5B6
ekaba.bisong@carleton.ca

Abstract

Artificial Neural Networks are a particular class of learning systems modeled after biological neural functions with an interesting penchant for Hebbian learning, that is "neurons that fire together, wire together". However, unlike their natural counterparts, artificial neural networks have a close and stringent coupling between the modules of neurons in the network. This coupling or locking imposes upon the network a strict and inflexible structure that prevent layers in the network from updating their weights until a full feed-forward and backward pass has occurred. Such a constraint though may have sufficed for a while, is now no longer feasible in the era of very-large-scale machine learning, coupled with the increased desire for parallelization of the learning process across multiple computing infrastructures. To solve this problem, synthetic gradients (SG) with decoupled neural interfaces (DNI) are introduced as a viable alternative to the backpropagation algorithm. This paper performs a speed benchmark to compare the speed and accuracy capabilities of SG-DNI as opposed to a standard neural interface using multilayer perceptron MLP. SG-DNI shows good promise, in that it not only captures the learning problem, it is also over 3-fold faster due to it asynchronous learning capabilities.

1 Introduction

Decoupled Neural Interfaces (DNI) is introduced as a novel optimization procedure to minimize the cross-entropy loss or cost function of the weights of the neurons in the module (or layer) of a neural network [Jaderberg et al., 2016a]. This novel system breaks the closed coupled/ locked-in dependency between the various layers of the neural networks by introducing synthetic gradients. Just before we examine synthetic gradients, let us take a closer look at the locking phenomenon in feed-forward/ back propagation procedures.

Locking occurs in feed-forward, and back-propagation techniques as activation units of neurons between layers of the network and the gradient of the loss are computed. This results in both forward, backward and update locking.

In "forward locking" as seen in the feed-forward pass, consequent modules are redundant until the activations of preceding modules are completed. While in update locking, all preceding modules are frozen in state until the gradients of the consequent modules are computed and back-propagated. In backward locking, modules can only update their activations after a complete feed-forward, and backward pass has been executed by the network.

Although this processing scheme has resulted in impressive results across a variety of complex learning tasks, the tight bound system stifles the network to operate sequentially or synchronously
during the learning process. This drawback is manifested in the time complexity of training large-scale neural networks. Moreso, in designing and implementing distributed learning network systems it becomes a waste of computational time/resources for modules to freeze in state until activations or gradients of loss updates are received from other antecedent or consequent layers (Czarnecki et al., 2017). It is in this light that we have the concept of a decoupled neural interface.

The critical algorithmic change in the decoupled neural paradigm is synthetic gradients. Back-propagation is removed from the learning process to break update locking between modules. Synthetic gradients are a parallel sub-modules that are attached to each layer or module of the neural networks. The goals of this synthetic gradients, are to approximate the gradient loss computed by backpropagation (Jaderberg et al., 2016a).

\[
\frac{\partial L}{\partial \theta_i} = f_{Bprop}(h_i, x_i, y_i, \theta_i) \ldots \frac{\partial h_i}{\partial \theta_i} \\
\simeq f_{Bprop}(h_i) \frac{\partial h_i}{\partial \theta_i}
\]

Synthetic gradients (Figure 1) are fixed as a side-module to a network layer and are trained to approximate the gradient of the loss function. To do this, they take as input the activation output of the network layer to which it is attached and uses the labels of the target function from the dataset to approximate the loss function. It then applies the gradient descent learning rule to update the weight of the decoupled layer. Synthetic gradients are trained using the actual labels of the dataset as explained previously or the back-propagated SGs from modules attached to layers higher up in the chain (Figure 2). (Jaderberg, 2016b).

![Figure 1: Synthetic gradients as side-modules to a network layer](https://deepmind.com/blog/decoupled-neural-networks-using-synthetic-gradients/)

![Figure 2: Synthetic gradients trained from back-propagated losses from SGs higher up in the chain](https://deepmind.com/blog/decoupled-neural-networks-using-synthetic-gradients/)

**Paper contribution**

This paper will present some performance benchmark metrics of synthetic gradients for Multi-layer perceptrons in comparison to regular MLP with a standard neural interface.

To that effect, Section 2 will present a briefly highlight some preliminary background information to help the reader comfortably grasp the concept of synthetic gradients with decoupled neural interfaces. Section 3 will present the methodology used in the experiment setup. The base parameters of the
experiment will be culled from (Jaderberg et al., 2016a). Section 4 will discuss the results of the experiments. Section 5 will conclude this paper and 6 will present areas for further investigation that we could not cover.

## 2 Background

Neural networks also known as "connectionist" architectures (McCulloch & Pitts, 1943; Hebb, 1949) are constructed as an interconnection of simple blocks or neurons each having a weight that captures a knowledge unit as activations in the system (Rosenblatt, 1958). The interest in connectionist systems are intensified by the use of massively parallel computing structures to simulate the deep representations of the cerebral cortex (Rochester et al., 1956) to realize automatic machine intelligence. This practice escapes some of the drawbacks of the heretofore symbolic processing (Smolensky, 1987).

In artificial neural network (ANN) design, the system is loosely categorized into three layers (or modules) called the input, hidden and output layers. Each layer is consisted of a set of \( i \) neurons, where \( \{i \in \mathbb{N} \mid i > 1\} \). The neurons in the input layer receive information, \( x, x \in \mathbb{R}^n \) from an external preceptor; this information is parameterized by a weight function, \( \theta \) via matrix multiplication to compute the activation of the neuron.

This activation is then forward-propagated to the hidden layers of the network acted upon by an activation function (or non-linearity) that streamlines the propagated signal. This function is carefully chosen to prevent the vanishing or explosion of gradients when learning the network (Pascanu et al., 2012). The hidden layers can consist of one or more modules of neurons. These computations come together in a fully-connected output layer to approximate a transformation or a map from the output to the target function (Widrow & Lehr, 1992b; Hecht-Nielsen, 1988):

\[
\hat{y} = f(x; \theta)
\]

The ANN model architecture described above is known as feedforward neural networks or multilayer perceptrons (MLP) (Figure 3) (Bebis & Georgiopoulos, 1994; Hornik et al., 1989). This name is so-called as information travels from the input to the output through the hidden layers. A more sophisticated type of network exists where there is a cyclic or feedback loop from the output into the model input.

These networks are called recurrent neural networks (RNN) (Rumelhart et al., 1986; Funahashi & Nakamura et al., 1993). RNN can process sequences of information or memories by sharing parameters or weights across several time frames (or position in sequence). In RNNs each output can be seen as a recursive input into the next state or time quata in a dynamical system. We can formally express RNNs as:

\[
h^{(t)} = f(h^{(t-1)}, x^t; \theta)
\]

Where \( h^{(t)} \) are the hidden states of a dynamic continuous time recurrent network system. Given a finite time step, \( \tau \) we can unfold the above equation into a recurrent computational graph each representing a position in the recursive sequence (Figure 4).

Recurrent networks can be designed to produce outputs at each time step while maintaining cyclic connections to the hidden states, or they can be made to only keep feedback loops between the outputs in successive dependent time-steps. Also, they can be designed to receive the entire input, \( x \), while maintaining cyclic loops between hidden states and generate a single output (Pascanu et al., 2013).

Backpropagation (Rumelhart et al., 1986) has long been hailed as the workhorse of neural networks in computing the gradient of the loss function given the weight of the network (Widrow & Lehr, 1992a). The algorithm commonly known as backprop was mostly responsible for renewing and flaming interest again in ANN (Werbos, 1981) by solving the XOR problem (Minsky, 1969) in learning non-linear representations by adjusting the weights of the network. Backpropagation have been shown to be "computationally efficient" (LeCun et al., 1998) with a robust set of heuristics for boosting performance when designing a network architecture.

---

1Figure 3 source: https://www.mql5.com/es/code/9002
2Figure 4 source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/
As earlier noted, the problem of vanishing and exploding gradients in RNN and MLP has been an issue when using a learning algorithm like stochastic gradient descent to backpropagate the partial derivates of the loss function to update the weights of the neurons in each preceding layer (Hochreiter et al., 2001; Pascanu et al., 2012). However, this problem is mitigated by using activation functions like ReLu or Leaky ReLu (Agostinelli et al., 2014).

In resolving the exploding and vanishing gradient problem, instead of the ReLu and Leaky ReLu functions squashing values between 0 and 1 as seen in the Sigmoid activation function, ReLu will threshold at 0 when \( x < 0 \), while Leaky ReLU will instead have a small negative slope when \( x < 0 \). Both ReLu and Leaky ReLu are linear with a slope of 1 when \( x > 0 \) (Karpathy et al., 2016).

In decoupled neural interfaces, backpropagation will only occur in the synthetic gradients and leave the main layers of the network update unlocked for asynchronous learning.

3 Methodology

The initial MLP setup architecture is similar to (Jaderberg et al., 2016a) to form a base benchmark of our results. The base MLP is made up of a 4-layered fully connected network on a MINIST dataset. Each hidden layer has 256 neurons, with a batch-normalization and ReLu non-linearity. We made our synthetic gradient architecture to be structurally identical to the inference network.

The experiments are run for 500k iterations and are optimized using adaptive moment estimation (Adam) (Kingma and Ba, 2014). The learning rate is optimized at \( 3 \times 10^{-5} \) and decreased by a factor of 10 at 300k and 400k steps. We set a batch size of 256 inputs.

The program was executed on a NVIDIA GeForce GTX 1080 Ti with TensorFlow CUDA GPU.
4 Discussion

Table 1: **Time/ Accuracy** - Decoupled MLP vs. Standard MLP :: 500k iterations

| Name        | Test Accuracy | Training Time     |
|-------------|---------------|-------------------|
| Standard MLP| 0.976300007105 | 7290.029 seconds  |
| SG-DNI MLP  | 0.957999996841 | 2496.592 seconds  |

In the above table, we ran the standard and SG-DNI MLP algorithms for 50,000 iterations, with 4-layers of FCN as specified in the methodology. From the result, we see that decoupling the MLP model has a significant increase in execution time over standard Multi-layer perceptrons. This result is anticipated because individual modules of the layers are constantly asynchronously adjusting their weight parameters based on the synthetic gradient approximation. The consequent result is a speed-up in the overall learning time because modules are no longer locked in step waiting for a full forward and backward pass before updating. However, we all observe that the test set accuracy is lower when compared to the standard MLP implementation.

Figure 5 and 6 below shows a plot of the accuracy and the loss of the training and validation datasets using decoupled neural interfaces with synthetic gradients.

![Figure 5: Train/ validation set accuracy :: SG-DNI multilayer perceptron.](image)

![Figure 6: Train/ validation loss plot :: SG-DNI multilayer perceptron.](image)
Computational Graph for Synthetic Gradients  Figure 7 below shows the TensorFlow computational graph for synthetic gradients. From the graph, we can see a synthetic gradient module attached to each layer of the network. The synthetic modules here are using the real output from the dataset to compute the loss function and the error gradient.

![Computational Graph for Synthetic Gradients](image)

**Figure 7: Computational Graph for Synthetic Gradients**

Table 2: **Time/ Accuracy Benchmark** - Decoupled MLP vs. Standard MLP :: 100k iterations

| Name           | Test Accuracy | Training Time  |
|----------------|---------------|----------------|
| Standard MLP   | 0.978100006282 | 823.099 seconds |
| SG-DNI MLP     | 0.933599999547 | 596.972 seconds |

In Table 2 above, we adjusted the number of iterations to 100k, to better observe the speed and performance measures at a lower termination condition. From the results, we observe that synthetic gradients with decoupled neural interfaces still performs remarkably better in terms of speed metrics, although it still lags behind the standard backpropagation neural interface with respect to performance accuracy.

5 Conclusion

From the results seen we can conclude that synthetic gradients hold an upper-hand when it comes to speed of execution and this is vital when we have to train very-large-scale neural networks learning systems. An interesting observation to make from the above experiments is that the difference in training time between synthetic gradients with DNIs and standard neural MLP networks grows as the number of iterations increases. This clearly puts synthetic gradients as a winner for speed gain due to its asynchronous update schemes by decoupling the network layers.

Backpropagation is still superior with respect to the accuracy of the network by making a better computation of the gradient of the loss function. However, this apparent advantage in accuracy can be matched by synthetic gradients in practice due to the humongous datasets available when training.
large-scale networks. Also, the over 3-fold increase in training speed can allow training for even more prolonged iterations for a shorter period. An increased training time and large dataset can more than compensate for the slight dip in gradient error computation as synthetic gradient attempts to approximation the exact backpropagation gradient.

6 Future Work

For further exploration of performance benchmarks, given more time and resources, it will be beneficial to train synthetic gradients on the CIFAR-100 dataset and compare its performance with standard neural networks.

Further, another important metric that should be carried out in a future study is to run synthetic gradients across parallel computing infrastructures (perhaps on GPUs if possible) to see how they perform in a distributed fashion. Asynchronous parallel learning is one of the major promises of decoupled neural interfaces. Running of select convolutional and recurrent neural architectures in a distributed setting will provide further context into the performance of synthetic gradients against standard backpropagation.

7 Acknowledgments

The author will like to thank Andrew Miles for help and support with the GPU computing facilities at the Carleton School of Computer Science.

References

Agostinelli, F., Hoffman, M., Sadowski, P., & Baldi, P. (2014). Learning Activation Functions to Improve Deep Neural Networks. ArXiv e-prints. arXiv: 1412.6830

Bebis, G. and Georgiopoulos, M. (1994). "Feed-forward neural networks," in IEEE Potentials, vol. 13, no. 4, pp. 27-31. doi: 10.1109/45.329294

Czarnecki, W. M., Swirszcz, G., Jaderberg, M., Osindero, S., Vinyals, O. and Kavukcuoglu, K. (2017). Understanding Synthetic Gradients and Decoupled Neural Interfaces. ArXiv e-prints. arXiv: 1703.00522.

Fei-Fei Li, Andrej Karpathy & Justin Johnson (2016). “Training Neural Networks, Part I”. Lecture Notes. [http://cs231n.stanford.edu/slides/2016/winter1516_lecture5.pdf](http://cs231n.stanford.edu/slides/2016/winter1516_lecture5.pdf)

Hebb, Donald (1949). *The Organization of Behavior*. New York: Wiley. ISBN 978-1-135-63190-1.

Hecht-Nielsen, Robert, (1988). *Neurocomputing*, IEEE Spectrum. Volume 25 Issue 3, March 1988. Page 36-41. http://dx.doi.org/10.1109/6.4520. doi: 10.1109/6.4520

Jaderberg, M., (2016). Decoupled Neural Interfaces using Synthetic Gradients. https://deepmind.com/blog/decoupled-neural-networks-using-synthetic-gradients/

Jaderberg, M., Czarnecki, W. M., Osindero, S., Vinyals, O., Graves, A., Silver, D., and Kavukcuoglu, K. (2017). Decoupled Neural Interfaces using Synthetic Gradients. ArXiv e-prints. arXiv: 1608.05343.

Ken-ichi Funahashi, Yuichi Nakamura, (1993). *Approximation of dynamical systems by continuous time recurrent neural networks*, In Neural Networks, Volume 6, Issue 6. Pages 801-806, ISSN 0893-6080, https://doi.org/10.1016/S0893-6080(05)80125-X.

Kingma, D.P. and Ba, J. (2014). Adam: A Method for Stochastic Optimization. ArXiv e-prints. arXiv: 1412.6980.

Kurt Hornik, Maxwell Stinchcombe, Halbert White, (1989) *Multilayer feedforward networks are universal approximators*, In Neural Networks, Volume 2, Issue 5. Pages 359-366, ISSN 0893-6080, https://doi.org/10.1016/0893-6080(89)90020-8.

[^5]: https://www.cs.toronto.edu/~kriz/cifar.html
LeCun Y., Bottou L., Orr G.B., Müller K.R. (1998) Efficient BackProp. In: Orr G.B., Müller KR. (eds) Neural Networks: Tricks of the Trade. Lecture Notes in Computer Science, vol 1524. Springer, Berlin, Heidelberg.

McCulloch, Warren; Walter Pitts (1943). "A Logical Calculus of Ideas Immanent in Nervous Activity". Bulletin of Mathematical Biophysics. 5 (4): 115–133. doi:10.1007/BF02478259

Minsky, Marvin; Papert, Seymour (1969). Perceptrons: An Introduction to Computational Geometry. MIT Press. ISBN 0-262-63022-2

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. (2012). On the difficulty of training recurrent neural networks. arXiv preprint arXiv:1211.5063.

Pascanu, Razvan, Gulcehre, Caglar, Cho, Kyunghyun, and Bengio, Yoshua. (2013) How to construct deep recurrent neural networks. CoRR, abs/1312.6026, 2013. URL http://arxiv.org/abs/1312.6026.

Rochester, N.; J.H. Holland; L.H. Habit; W.L. Duda (1956). "Tests on a cell assembly theory of the action of the brain, using a large digital computer". IRE Transactions on Information Theory. 2 (3): 80–93. doi:10.1109/TIT.1956.1056810

Rosenblatt, F. (1958). "The Perceptron: A Probabilistic Model For Information Storage And Organization In The Brain". Psychological Review. 65 (6): 386–408. CiteSeerX: 10.1.1.588.3775. doi:10.1037/h0042519. PMID 13602029

Rumelhart, David E.; Hinton, Geoffrey E.; Williams, Ronald J. (1986). "Learning representations by back-propagating errors". Nature. 323 (6088): 533–536. doi:10.1038/323533a0

Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber (15 January 2001). "Gradient flow in recurrent nets: the difficulty of learning long-term dependencies". In Kolen, John F.; Kremer, Stefan C. A Field Guide to Dynamical Recurrent Networks. John Wiley & Sons. ISBN 978-0-7803-5369-5

Smolensky, P. (1987) “Connectionist AI, Symbolic AI, and the Brain ”. Artificial Intelligence Review (1987) 1, 95-109. https://doi.org/10.1007/BF00130011

Werbos, P. J. (1981). Applications of advances in nonlinear sensitivity analysis. In Proceedings of the 10th IFIP Conference, 31.8 - 4.9, NYC, pages 762–770.

Widrow, B. and Lehr, M.A.(1992). Backpropagation and its Applications,” Proceedings of the INNS Summer Workshop on Neural Network Computing for the Electric Power Industry, Stanford, pp.21-29, August 1992.

Widrow, B. and Lehr, M.A.Feedforward Networks, in INNS Above Threshold, December 1992.