Sorting Numbers Using Plastic Neural Networks
CSCI 2951X

Sean Segal SEAN_SEGAL@BROWN.EDU
Brown University, 69 Brown St. RI 02912 USA

Ansel Vahle ANSEL_VAHLE@BROWN.EDU
Brown University, 69 Brown St. RI 02912 USA

Abstract

This paper explores recent attempts to model synaptic plasticity in artificial neural networks. In particular, we replicate the architecture proposed by Miconi et al. (2018) in which traditional neural networks are augemented so that each neuron's output is determined by both a traditional fixed compoenent in addition to a fast-changing plastic component. We test the effectiveness of this new architecture on a sorting task — a domain that has not yet been openly published before. The inspiration for this task comes from the intersection of logical problems and connectionism, with many connectionsist systems being explicitly defined to solve these problems rather than learned. We find that plastic networks offer a slight improvement over traditional neural networks, demonstrating the flexibility and context dependent computation introduced by this model.

1. Introduction

The beauty of intelligence is the ability to adapt. Humans themselves are able to learn new concepts, actions, and structures very quickly and are able to apply previous experience and context in order to do this more efficiently. Likewise, humans are most impressed when other forms of life are able to handle uncertainty. Many computer scientists believe that understanding how humans and other biological agents rapidly learn from their experiences will help push research towards creating more effective agents. This growing field is referred to as meta-learning (Thrun & Pratt, 1998).

In order to replicate function, one often tries to replicate the form. As modern research moves forward in search of artificial intelligence, scientists have attempted to mimic the biological conception of intelligence, the brain. Neural networks, meant to loosely replicate the action of neurons firing in the brain, have achieved unparalleled performance in many different domains (Krizhevsky et al., 2012; Mnih et al., 2013; Silver et al., 2017; Sutskever et al., 2014; Wu et al., 2016). Recurrent neural networks, in particular long short-term memory networks (Hochreiter & Schmidhuber, 1997), have achieved a high level of performance on account of their ability to take into account the recent past and can effectively take context into account when operating. Assuming a general knowledge of neural networks — this memory comes in the form of the form of the weights, often referred to as synapses or neurons.

While RNNs and hidden Markov models are able to capture a hidden state, RNN's were a large step forward, as they have a higher computational capacity and a greater memory. Alex Graves (Graves et al., 2014) points out the necessity of this "dynamic" state as it allows for context dependent computation, i.e. allowing recent information gained to affect the output of a current input.

The amount of information they can store is inherently limited and it can take a significant amount of time for these networks to account for new information. Ba and Hinton bound this long term memory at $O(H2) + O(IH) + O(HO)$ where H, I, and O are the sizes of hidden, input, and output units, respectively (Ba et al., 2016); this memory capacity, while powerful, is slow to update and does not actively reflect new information, but instead reflects inferred patterns from the data. They bound the amount of short term memory at just $O(H)$. Regardless of memory capac-

ity, a different neural net must be trained for each task and there is little flexibility in function even if there is a consistently defined input and output set.

Ba and Hinton (2016) additionally bound the timescale for different mechanisms used to reflect the context of different samples; each of the proposed methods, e.g. attention (Bahdanau et al., 2014), is not fast enough for both computational efficiency and actively reflecting the context. As the field stood a few years ago, there was a question of how to increase the memory capacity of these nets a) without having to create arbitrarily deep nets, b) sacrifice computational efficiency, c) build long term memory while also being able to heavily weight new information.

In human brains, there is some question about how long term human memory is stored. Some (Martin & Morris, 2000; Liu & Tonegawa, 2018) believe that it is achieved through synaptic plasticity, i.e. the process in which neurons become more heavily connected the more they interact and vice versa. This theory is attributed to Hebbs and is best described in his aphorism, "neurons that fire together, wire together" (Hebb, 1949). However, there is also evidence that the brain utilizes properties of synaptic plasticity in the short term to maintain context and rapidly adjust to new information (Tsodyks et al., 1998; F Abbott & Regehr, 2004; Barak & Tsodyks, 2007).

In typical fashion, many computer scientists have attempted to replicate this aspect of the human brain. Two papers (Ba et al., 2016; Miconi et al., 2018) have taken similar approaches to this — effectively creating a separate weight matrix that accounts for the interaction between neurons (see section 2.1 for more detail). This weight matrix is updated significantly after each sample and information from previous samples exponentially decays if it is not being used. As these weights are rapidly generated and allow for context dependent computation, it is a logical step to consider the implications of these weights on the efficacy of these methods in meta-learning (see section 2.2).

As mentioned previously, neural nets have achieved success in many different domains; the majority of these domains do not include logical problems, such as sorting, memorization, and boolean. It has been demonstrated that RNNs are Turing-Complete (Siegelmann & Sontag, 1995), which implies that these can (not necessarily should) be used for accomplishing these tasks. Some have attempted to create end-to-end systems entirely composed of neural nets. The implications of this are these operations can be differentiated. Many of these networks are explicitly constructed by hand and are therefore not flexible. One such task is using neural nets to sort lists of numbers (Takefuji & Lee, 1990; Chen & Hsieh, 1990). In this paper, we propose testing a plasticity method on the domain of sorting numbers.

2. Related Work

2.1. Fast Weights

The idea of fast weights was in response to two new developments in the field of deep learning: new variations in types of memory and new types of context computing. Graves' Neural Turing Machines (Graves et al., 2014) were reading and writing to static memory using a method he developed specifically for his machine; Grefenstette built RNNs on top of Stacks, Queues, and DeQues (Grefenstette et al., 2015); and Facebook had just developed a new type of neural network called a Memory Network that was meant to explicitly relate input features to output features and to contain a memory of previous responses (Weston et al., 2014).

Ba, Hinton, and others at Google (2016) believed that these structures were not realistically based on the structure of the brain and thus could never achieve the same success. They additionally found fault in these methods as there were particular rules concerning when and how to update these weights. While they did want to increase the memory capacity of these nets, they placed a heavier emphasis on "having a simple, non-iterative storage rule." They settled on using "an outer product rule to store hidden activity vectors in fast weights that decay rapidly."

They achieve this by initially defining some fast weights matrix, A. They then define a decay rate λ and a learning rate η. At each iteration, t, the matrix A decays and the new fast weights matrix, i.e. the outer product of the hidden state vector, $h(t)$, scaled by η.

$$A(t) = \lambda A(t-1) + \eta h(t)h(t)^T \tag{1}$$

They utilize this fast weights matrix when computing the hidden vector for each input. They do this in a multi step process, where they repeatedly apply the fast weights matrix, the hidden weights W and C, and the nonlinear function σ to generate s intermediate states.

$$h_{s+1}(t+1) = f([Wh(t) + Cx(t)] + A(t)h_s(t+1)) \tag{2}$$

From here, they go on to show that at any given step A still reflects previous inputs and that you can rewrite

the fast weights matrix and the product of it and the hidden state as:

$$A(t) = \eta \sum_{\tau=1}^{\tau=t} \lambda^{t-\tau} h(\tau) h(\tau)^T \quad (3)$$

$$A(t)h_s(t+1) = \eta \sum_{\tau=1}^{\tau=t} \lambda^{t-\tau} h(\tau)[h(\tau)^T h_s(t+1)] \quad (4)$$

For a more detailed explanation of the algorithm and the proof, please see the paper.

2.2. Differentiable Plasticity

Researchers at Uber AI Labs (Miconi et al., 2018), compelled by other research in meta-learning, sought to apply the principles of synaptic plasticity to these domains. Relying on the work on their previous demonstration of feasibility and tractability for this type of model (Miconi, 2016), they increased the scope of the domains explored to show that plasticity methods are incredibly effective even when millions of parameters are required.

The fast weights algorithm described a fixed weighting of the plastic components when applying it to the non-plastic components. The authors of this paper made an intentional choice to redefine the computations such that the plastic and non-plastic components would be separate. They define the plastic components as a Hebbian trace. This is defined in a very similar manner to the fast weights but instead of using a fixed learning rate, they instead use a convex combination of the previous Hebbian matrix and the new update.

$$\text{Hebb}_{i,j}(t+1) = \eta x_i(t1) x_j(t) + (1-\eta)\text{Hebb}_{i,j}(t) \quad (5)$$

The real optimization they introduced is a weight matrix, α, that regulates how plastic each neuron should be, i.e. how much weight to give to the plastic component of each neuron. The α matrix can additionally be optimized via gradient descent. The following equation expresses how the value of each output is determined.

$$x_j(t) = \sigma\{\sum_{i \in \text{inputs}} [w_{i,j} x_i(t-1)] + \alpha_{i,j}\text{Hebb}_{i,j}(t) x_i(t-1)]\} \quad (6)$$

σ here is again the nonlinear function defined earlier; the authors use `tanh` throughout the paper. The authors then made another jump — the alphas maintain state across many episodes, effectively reflecting the strengthening of synaptic connections. The Hebbian trace, on the other hand, is reset to 0 at the end of each episode. Thus, in order to rebuild any benefit of plasticity in the short term, several examples must be input. As the authors point out, this rule effectively demonstrates that over time these traces go to zero (assuming time increments without new input). They additionally experiment with another rule, Oja's rule (Oja, 2008), to calculate the Hebbian traces

$$\text{Hebb}_{i,j}(t+1) = \text{Hebb}_{i,j}(t) + \eta x_j(t)(x_i(t-1) - x_j(t)\text{Hebb}_{i,j}(y)) \quad (7)$$

For a more detailed explanation of the algorithm, please see the paper.

2.3. Neural Sorting

There have been several papers the discuss the feasibility of neural sorting (Takefuji & Lee, 1990; Chen & Hsieh, 1990). These networks are specifically constructed to output the proper answer. This provides a powerful intution that demonstrates the differentiability of a logical process. However, neural networks are best known for being able to take in data and learn to output the correct answer, through either supervised or unsupervised learning. Teaching a neural network to sort is a different task than building one that works.

3. Our Experiment

The goal of our experiements is to replicate the network architecture of (Miconi et al., 2018) and test the network's performance on a sorting task - a task that was not explored in their original work.

While our experiment is primarily a supervised learning task, it can help to view the task through the lens of reinforcement learning. At any point it time, the agent is in one of two states: A or D. In state A the goal of the agent is to sort a given sequence of numbers in ascending order. Alternatively, in state D, the goal of the agent is to sort the given sequence in descending order. After every `num_steps` samples, there is a probability p that the agent will transition to state D and a probability $1-p$ that the agent will transition to state A. (Note that this transition is indepdent of the current state.) Finally, the reward is defined as (the negative) of the mean squared error of each element in the sequence.

The challenge here is that the same network must learn to sort numbers in increasing and decreasing order. Additionally, given an input, the network *does not know* which order it should be sorted (ie: it does not know the current state). When designing this experiement, the hope is that the plastic neural network will be able to adapt to the different settings and

quickly learn whether to sort numbers in ascending or descending order from only a few examples. Furthermore, one would expect that a network trained to sort values in ascending order and one trained to sort numbers in descending order to share some weights. A plastic neural network should be able to learn these connections and set them to be fixed across episodes while settings connections that must be different between the two tasks to be more plastic.

We implemented plastic neural networks in Python using Tensorflow. We used a three-layer feedforward neural network as a baseline. We used `tanh` as an activation function and varied the hidden sizes of the layers between 20 to 80. We used a learning rate of `1e-5` with the built in Adam optimizer (Kingma & Ba, 2014).

In order to ensure a fair comparison, we used an identical architecture and parameters for the plastic network. The only difference was that we added the α paramaeters described above and kept track of a Hebbian trace for each plastic layer. We experimented with a neural network in which only the final layer was plastic, only the first layer was plastic and all layers were plastic.

4. Results

In all the experiments below we set `num_steps` $= 50$ and `num_epsiodes` $= 5000$. For each epsiode, the inputs of each sequence were selected uniformly at random from the integers between 0-100. Then, with probability $p = 0.5$, the labeled examples were those sequences sorted in descending order and with probability $1 - p$ those sequences were sorted in ascending order. The networks were training on 50 sequences (or steps) per epsiode.

After training, we repeated this proceess on 50 test episodes. During testing, the gradient descent optimizer was no longer run. However, the Hebbian traces were still updated and passed to the neural net at each step. The loss report below can be calculated as

$$\sum_{s=1}^{\texttt{num\_steps}} \sum_{i=1}^{\texttt{sequence\_len}} (y_{si} - \hat{y}_{si})^2$$

where y_s is the true sorted sequence for step s and $\hat{y}_s$ is the output of the neural network for step s.

Our results are available in Table 1. We can see that in all trials, the plastic networks modestly outperformed the non-plastic networks. Given that the plastic neural networks have twice as many parameters as the regular networks, the gain in performance is minimal and may not be worth the additional memory and computational resources required.

Table 1. Total mean squared error (over 500 sequences) of sorted sequences on test dataset

Data set	Non-plastic	Plastic
Hidden Size 20 (Trial 1)	1.2339	1.5495
Hidden Size 20 (Trial 2)	1.2082	1.4281
Hidden Size 40 (Trial 1)	1.3864	1.6192
Hidden Size 40 (Trial 2)	1.2738	1.2973
Hidden Size 40 (Trial 3)	1.2745	1.4842

It is important to note that we also tried networks with larger hidden sizes (80 and above). Results for these networks were worse for both the plastic and non-plastic networks, most likely because they required more data to effectively train. Given more computational resources, we would retrain the larger networks with more episodes and compare the results of those nets.

5. Future Work

The breadth of our experiments were restricted due to limited computational resources. It would be interesting to run these experiements with networks with more complicated architectures, including deeper networks and potentially recurrent neural networks. Additionally, given that differentiable plasticity is very recent development, there are many other domains in which it would be interesting to test the performance of this new architecture.

Applying this framework to a transfer learning problem would be quite interesting. Given a reinforcement learning problem with a fixed agent space, it would be interesting to see how neurons again rely on each other as the problem space changes. As Miconi et al. demonstrate, this framework is effective on time specific domains and can be applied to reinforcement learning problems, implying that this task is feasible. A similar experiment would be to consider the plasticity weights generated from different networks trained in the same environments, but with different goals, to see what information the plasticity weights seem to represent.

6. Conclusion

In the past year, neural networks that model synaptic plasticity have become a promising new research area. The power of synaptic plasticity is clear when

observing human beings' abilities to quickly adapt to new environments and this is clearly a property that one would expect to exist in intelligent agents. Our work surveys recent work in the area and tests Uber AI Lab's architecture on a new domain. Our tests demonstrate that plasticity offers some improvements over regular neural networks in the domain of neural sorting. Given more computational resources, it would be interesting to continue testing how plasticity affects the performance of different network architectures in the sorting domain. Overall, however, our experiment suggests that plasticity may only provide a significant performance boost in "Learning to Learn" domains rather than in supervised domains.

6.1. Software and Data

The code from our experiments are open source and can be found on Github, here: https://github.com/seansegal/cs2951x-finalproject.

References

Ba, J., Hinton, G., Mnih, V., Leibo, J. Z., and Ionescu, C. Using Fast Weights to Attend to the Recent Past. *ArXiv e-prints*, October 2016.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014. URL http://arxiv.org/abs/1409.0473.

Barak, Omri and Tsodyks, Misha. Persistent activity in neural networks with dynamic synapses. *PLOS Computational Biology*, 3(2):1–1, 02 2007. doi: 10.1371/journal.pcbi.0030035. URL https://doi.org/10.1371/journal.pcbi.0030035.

Chen, W. T. C. Wen-Tsuen and Hsieh, K. R. H. Kuen-Rong. A neural sorting network with o(1) time complexity. In *1990 IJCNN International Joint Conference on Neural Networks*, pp. 87–95 vol.1, June 1990. doi: 10.1109/IJCNN.1990.137551.

F Abbott, L and Regehr, Wade. Synaptic computation. 431:796–803, 11 2004.

Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural turing machines. *CoRR*, abs/1410.5401, 2014. URL http://arxiv.org/abs/1410.5401.

Grefenstette, Edward, Hermann, Karl Moritz, Suleyman, Mustafa, and Blunsom, Phil. Learning to transduce with unbounded memory. *CoRR*, abs/1506.02516, 2015. URL http://arxiv.org/abs/1506.02516.

Hebb, Donald O. The organization of behavior: a neuropsy- chological theory. 1949.

Hochreiter, Sepp and Schmidhuber, Jrgen. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. doi: 10.1162/neco.1997.9.8.1735. URL https://doi.org/10.1162/neco.1997.9.8.1735.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL http://arxiv.org/abs/1412.6980.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In Pereira, F., Burges, C. J. C., Bottou, L., and Weinberger, K. Q. (eds.), *Advances in Neural Information Processing Systems 25*, pp. 1097–1105. Curran Associates, Inc., 2012. URL http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neu pdf.

Liu, Xu, Ramirez Steve-Pang Petti T Puryear Corey B Govindarajan Arvind Deisseroth Karl and Tonegawa, Susumu. Optogenetic stimulation of a hippocampal engram activates fear memory recall. *Nature*, 484:381–385, 2018.

Martin, Stephen J, Grimwood-Paul D and Morris, Richard GM. Synaptic plasticity and memory: an evaluation of the hypothesis. *Annual review of neuroscience*, 23:649 – 711, 2000.

Miconi, Thomas. Backpropagation of hebbian plasticity for lifelong learning. *CoRR*, abs/1609.02228, 2016. URL http://arxiv.org/abs/1609.02228.

Miconi, Thomas, Clune, Jeff, and Stanley, Kenneth O. Differentiable plasticity: training plastic neural networks with backpropagation. *CoRR*, abs/1804.02464, 2018. URL http://arxiv.org/abs/1804.02464.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Graves, Alex, Antonoglou, Ioannis, Wierstra, Daan, and Riedmiller, Martin A. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013. URL http://arxiv.org/abs/1312.5602.

Oja, E. Oja learning rule. *Scholarpedia*, 3(3):3612, 2008. doi: 10.4249/scholarpedia.3612. revision #91606.

Siegelmann, H.T. and Sontag, E.D. On the computational power of neural nets. *Journal of Computer and System Sciences*, 50(1):132 – 150, 1995. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.

1995.1013. URL http://www.sciencedirect.com/science/article/pii/S0022000085710136.

Silver, David, Hubert, Thomas, Schrittwieser, Julian, Antonoglou, Ioannis, Lai, Matthew, Guez, Arthur, Lanctot, Marc, Sifre, Laurent, Kumaran, Dharshan, Graepel, Thore, Lillicrap, Timothy P., Simonyan, Karen, and Hassabis, Demis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017. URL http://arxiv.org/abs/1712.01815.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014. URL http://arxiv.org/abs/1409.3215.

Takefuji, Y. and Lee, K. C. A super-parallel sorting algorithm based on neural networks. *IEEE Transactions on Circuits and Systems*, 37(11):1425–1429, Nov 1990. ISSN 0098-4094. doi: 10.1109/31.62417.

Thrun, Sebastian and Pratt, Lorien. Learning to learn: Introduction and overview. In Thrun, Sebastian and Pratt, Lorien (eds.), *Learning to Learn*, pp. 3–17. Kluwer Academic Publishers, orwell, MA, USA, 1998.

Tsodyks, Misha, Pawelzik, Klaus, and Markram, Henry. Neural networks with dynamic synapses. *Neural Comput.*, 10(4):821–835, May 1998. ISSN 0899-7667. doi: 10.1162/089976698300017502. URL http://dx.doi.org/10.1162/089976698300017502.

Weston, Jason, Chopra, Sumit, and Bordes, Antoine. Memory networks. *CoRR*, abs/1410.3916, 2014. URL http://arxiv.org/abs/1410.3916.

Wu, Yonghui, Schuster, Mike, Chen, Zhifeng, Le, Quoc V., Norouzi, Mohammad, Macherey, Wolfgang, Krikun, Maxim, Cao, Yuan, Gao, Qin, Macherey, Klaus, Klingner, Jeff, Shah, Apurva, Johnson, Melvin, Liu, Xiaobing, Kaiser, Lukasz, Gouws, Stephan, Kato, Yoshikiyo, Kudo, Taku, Kazawa, Hideto, Stevens, Keith, Kurian, George, Patil, Nishant, Wang, Wei, Young, Cliff, Smith, Jason, Riesa, Jason, Rudnick, Alex, Vinyals, Oriol, Corrado, Greg, Hughes, Macduff, and Dean, Jeffrey. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016. URL http://arxiv.org/abs/1609.08144.