Differentiable plasticity: training plastic neural networks with backpropagation

Thomas Miconi
Uber AI Labs
tmiconi@uber.com

Jeff Clune
Uber AI Labs
jeffclune@uber.com

Kenneth O. Stanley
Uber AI Labs
kstanley@uber.com

April 10, 2018

Abstract

How can we build agents that keep learning from experience, quickly and efficiently, after their initial training? Here we take inspiration from the main mechanism of learning in biological brains: synaptic plasticity, carefully tuned by evolution to produce efficient lifelong learning. We show that plasticity, just like connection weights, can be optimized by gradient descent in large (millions of parameters) recurrent networks with Hebbian plastic connections. First, recurrent plastic networks with more than two million parameters can be trained to memorize and reconstruct sets of novel, high-dimensional (1,000+ pixels) natural images not seen during training. Crucially, traditional non-plastic recurrent networks fail to solve this task. Furthermore, trained plastic networks can also solve generic meta-learning tasks such as the Omniglot task, with competitive results and little parameter overhead. Finally, in reinforcement learning settings, plastic networks outperform a non-plastic equivalent in a maze exploration task. We conclude that differentiable plasticity may provide a powerful novel approach to the learning-to-learn problem.

1. Introduction: the problem of "learning to learn"

Many of the recent spectacular successes in machine learning involve learning one complex task very well, through extensive training over thousands or millions of training examples (Krizhevsky et al., 2012; Mnih et al., 2015; Silver et al., 2016). After learning is complete, the agent's knowledge is fixed and unchanging; if the agent is to be applied to a different task, it must be re-trained (fully or partially), again requiring a very large number of new training examples. By contrast, biological agents exhibit a remarkable ability to learn quickly and efficiently from ongoing experience: animals can learn to navigate and remember the location of (and quickest way to) food sources, discover and remember rewarding or aversive properties of novel objects and situations, etc. – often from a single exposure.

Endowing artificial agents with lifelong learning abilities is essential to allowing them to master environments with changing or unpredictable features, or specific features that are unknowable at the time of training. For example, supervised learning in deep neural networks can allow a neural network to identify letters from a specific, fixed alphabet to which it was exposed during its training; however, autonomous learning abilities would allow an agent to acquire knowledge of *any* alphabet, including alphabets that are unknown to the human designer at the time of training.

An additional benefit of autonomous learning abilities is that in many tasks (e.g. object recognition, maze navigation, etc.), the bulk of fixed, unchanging structure in the task can be stored in the fixed knowledge of the agent, leaving only the changing, contingent parameters of the specific situation to be learned from experience. As a result, learning the actual specific instance of the task at hand (that is, the actual latent parameters that do vary across multiple instances of the general task) can be extremely fast, requiring only few or even a single experience with the environment.

Several meta-learning methods have been proposed to train agents to learn autonomously (reviewed shortly). However, unlike in current approaches, in biological brains long-term learning is thought to occur (Martin et al., 2000; Liu et al., 2012) primarily through *synaptic plasticity* – the strengthening and weakening of connections between neurons as a result of neural activity, as carefully tuned by evolution over millions of years to enable efficient learning during the lifetime of each individual. While multiple forms of synaptic plasticity exist, many of them follow the general principle known as Hebb's rule: if a neuron repeatedly takes part in making another neuron fire, the connection between them is strengthened (often roughly summarized as "neurons that fire together, wire together") (Hebb, 1949). This principle underlies several forms of observed plasticity in animal brains, allowing them to learn from experience and adapt to their environment.

Designing neural networks with plastic connections has long been explored with evolutionary algorithms (see Soltoggio et al. 2017 for a recent review), but has been so far relatively less studied in deep learning. However, given the spectacular results of gradient descent in designing traditional non-plastic neural networks for complex tasks, it would be of great interest to expand backpropagation training to networks with plastic connections – optimizing through gradient descent not only the base weights, but also the amount of plasticity in each connection. We previously demonstrated the theoretical feasibility and analytical tractability of this approach (Miconi, 2016).

Here we show that this approach indeed can succeed at training large (millions of parameters) networks for non-trivial tasks. To demonstrate our approach, we apply it to three different types of tasks: complex pattern memorization (including natural images), one-shot classification (on the Omniglot dataset), and reinforcement learning (in a maze exploration problem). We show that plastic networks provide competitive results on Omniglot, improve performance in maze exploration, and outperform advanced non-plastic recurrent networks (LSTMs) by orders of magnitude in complex pattern memorization. This result is interesting not only for opening up a new avenue of investigation in gradient-based neural network training, but also for showing that meta-properties of neural structures normally attributed to evolution or a priori design are in fact amenable to gradient descent, hinting at a whole class of heretofore unimagined meta-learning algorithms.

2. Related work

Designing agents that can quickly learn from ongoing experience is the basic problem of meta-learning, or "learning-to-learn" (Thrun & Pratt, 1998). Several methods already exist to address this problem. One approach is to augment neural networks with external content-addressable memory banks, as in Memory Networks and Neural Turing Machines (Graves et al., 2014; Sukhbaatar et al., 2015; Santoro et al., 2016). The memory bank can be read from and written to by an attentional mechanism within the controller network, enabling fast memorization of ongoing experience. A more straightforward approach is to simply train standard recurrent networks (which, as universal Turing machines, can *in principle* learn any computable function of their inputs) to adequately incorporate past experience in their future responses. With a proper training schedule (e.g. augmenting inputs at time t with the output and error at time $t-1$), recurrent networks can learn to automatically integrate novel information during an episode (Hochreiter et al., 2001; Wang et al., 2016; Duan et al., 2016).

A different approach consists of attaching additional "fast weights" to the connections of a standard neural network. These fast weights automatically grow and decay as a result of ongoing neural activity, essentially reinforcing the influence of recently encountered patterns (Ba et al., 2016). Schmidhuber (1993) has pointed out that this mechanism can in principle be exploited by adequately trained fixed weights to store any desired memory.

Yet another approach is the MAML method (Finn et al., 2017), which performs gradient descent via backpropagation during the episode itself. In this case, the meta-learning consists in training the base network so that it can be "fine-tuned" easily and reliably by few steps (or even just one step) of additional gradient descent while performing the actual task.

For classification problems, one may instead train an embedding to reliably provide adequate discrimination between instances of different classes, over any set of classes within the distribution defined by the task at hand. Then, during each episode, classification is reduced to a comparison between the embedding of the test and example instances. This approach is exemplified both by Matching Networks (Vinyals et al., 2016) (in which the learned embedding is applied to the test and example instances, and the predicted label of the test instance is an average of the examples' labels, weighted by cosine similarity in embedding space with the test instance) and by Prototypical Networks (Snell et al., 2017) (in which the embedded examples are averaged to produce prototypical vectors for each class, with which the test instance is matched by nearest-neighbor classification).

One advantage of using trainable synaptic plasticity as a substrate for meta-learning is the great potential flexibility of this approach. For example, Memory Networks enforce a specific memory storage model in which memories must be embedded in fixed-size vectors and retrieved through some attentional mechanism. In contrast, plasticity at the level of individual synapses may translate into very different forms of memory, the exact implementation of which can be determined by (trainable) network structure. Fixed-weight recurrent networks, meanwhile, require neurons to be used for both storage and computation. While recurrent neural networks are universal Turing machines, and can therefore perform any computable task *in theory*, allowing the connections themselves to store information may reduce the computational burden on neurons. Fast weights can exploit network connectivity for storage of short-term information, but their uniform plasticity (which means all connections have fast weights, which are all updated according to the same parameters) imposes a stereotypical behavior on these memories, essentially "attending to the recent past" (Ba et al., 2016). By contrast, in natural brains, the amount of plasticity in different connections varies widely and (crucially) is actively molded by the long-term learning mechanism (namely, evolution) to perform specific computations; furthermore, biological plasticity is not limited to “fast” weight changes and can sustain memories over years or decades.

3. Differentiable plasticity

To train plastic networks with backpropagation, a plasticity rule must be specified. Many formulations are possible. Here we choose a flexible formulation that keeps separate plastic and non-plastic (baseline) components for each connection, while allowing multiple Hebbian rules to be easily implemented within the framework.

A connection between any two neurons i and j has both a fixed component and a plastic component. The fixed part is just a traditional connection weight $w_{i,j}$. The plastic part is stored in a *Hebbian trace* $\mathrm{Hebb}_{i,j}$, which varies during a lifetime according to ongoing inputs and outputs (note that we use “lifetime” and “episode” interchangeably). In the simplest case studied here, the Hebbian trace is simply a running average of the product of pre- and post-synaptic activity. The relative importance of plastic and fixed components in the connection is structurally determined by the plasticity coefficient $\alpha_{i,j}$, which multiplies the Hebbian trace to form the full plastic component of the connection. Thus, at any time, the total, effective weight of the connection between neurons i and j is the sum of the baseline (fixed) weight $w_{i,j}$, plus the Hebbian trace $\mathrm{Hebb}_{i,j}$ multiplied by the plasticity coefficient $\alpha_{i,j}$. The precise network equations for the output $x_j(t)$ of neuron j are:

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

$$\mathrm{Hebb}_{i,j}(t+1) = \eta x_i(t-1) x_j(t) + (1-\eta)\mathrm{Hebb}_{i,j}(t) \quad (2)$$

Here σ is a nonlinear function (we use $\tanh$ throughout this paper), and ’inputs’ denotes the set of all neurons providing input to neuron j.

In this way, depending on the values of $w_{i,j}$ and $\alpha_{i,j}$, a connection can be fully fixed (if $\alpha = 0$), or fully plastic with no fixed component (if $w = 0$), or have both a fixed and a plastic component.

The Hebbian trace $\mathrm{Hebb}_{i,j}$ is initialized to zero at the beginning of each lifetime/episode: it is purely a lifetime quantity. The parameters $w_{i,j}$ and $\alpha_{i,j}$, on the other hand, are the structural parameters of the network that are conserved across lifetimes, and optimized by gradient descent between lifetimes (descending the gradient of the error computed during episodes), to maximize expected performance over a lifetime/episode. Note that η, the “learning rate” of plasticity, is also an optimized parameter of the algorithm (for simplicity, in this paper, all connections share the same value of η, which is thus a single scalar parameter for the entire network).

In Equation 2, the weight decay term η prevents runaway positive feedback on Hebbian traces. However, because of this weight decay term, Hebbian traces (and thus memories) decay to zero in the absence of input. Fortunately, other, more complex Hebbian rules can maintain stable weight values indefinitely in the absence of stimulation, thus allowing stable long-term memories, while still preventing runaway divergences. One well-known example is Oja’s rule (Oja, 2008). To incorporate Oja’s rule in our framework, we simply replace Equation 2 above with the following (note the absence of a decay term):

$$\mathrm{Hebb}_{i,j}(t+1) = \mathrm{Hebb}_{i,j}(t) + \eta x_j(t)(x_i(t-1) - x_j(t)\mathrm{Hebb}_{i,j}(t)) \quad (3)$$

To illustrate the flexibility of our approach, we demonstrate both rules in the experiments reported below.

This overall framework for differentiable plasticity is easily implemented in automatic differentiation packages. All experiments reported here use the PyTorch package to compute gradients. Implementing this formulation of plasticity only requires a couple additional lines of code on top of a standard PyTorch network implementation.

4. Experiments and Results

The experiments in this section are designed to show both that differentiable plasticity actually works within a meta-learning framework, and that in some cases it provides a definitive advantage over alternative options. The hope is that these investigations help to establish the value of this research direction; while it is reasonable to expect that no meta-learning approach is definitively superior over all problems, the existence of an entirely new option should help to stimulate new ideas and research directions in this area.

4.1. Pattern memorization: Binary patterns

To demonstrate the differentiable plasticity approach, we first apply it to the task of quickly memorizing sets of arbitrary high-dimensional patterns (including novel patterns never seen during training), and reconstructing these patterns when exposed to partial, degraded versions of them. Networks that can perform this task are known as content-addressable memories, or auto-associative networks (Dayan & Abbott, 2001). This task is a useful test because it is

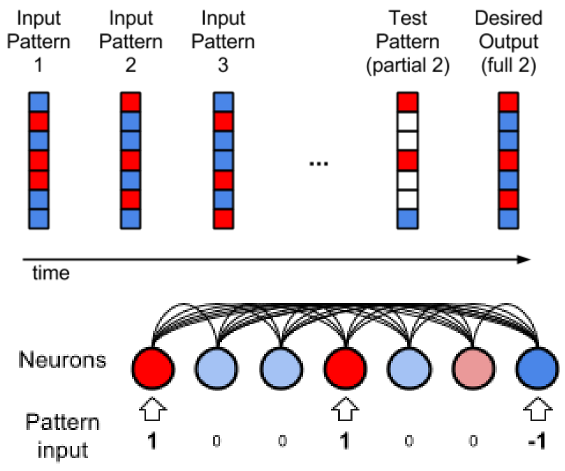


Figure 1: Top: Conceptual description of the task. Bottom: depiction of the architecture.

known that hand-designed recurrent networks with (usually homogenous) Hebbian plastic connections can successfully solve it for binary patterns (Hopfield, 1982). Thus, if differentiable plasticity is to be of any help, it should be able to automatically solve this task – that is, to automatically design networks that can perform the task just like existing hand-designed networks can.

Figure 1 (top) depicts an episode in this task. The network is shown a set of 5 binary patterns in succession. Each binary pattern is composed of 1,000 elements, each of which is either 1 (dark red) or -1 (dark blue). Each pattern is shown for 10 time steps, with 3 time steps of zero input between presentations, and the whole sequence of patterns is presented 3 times in random order (few-shot learning). Then, one of the presented patterns is chosen at random and degraded, by setting half of its bits to zero (i.e. no input - white). This degraded pattern is then fed as an input to the network. The task of the network is to reproduce the correct full pattern in its outputs, drawing on its memory to complete the missing bits of the degraded pattern (pale blue and red in the bottom panel).

The architecture (Figure 1, bottom) is a fully recurrent neural network with one neuron per pattern element, plus one fixed-output ("bias") neuron, for a total of 1,001 neurons. Input patterns are fed by clamping the value of each neuron to the value of the corresponding element in the pattern, if this value is not zero (i.e. 1 or -1); for zero-valued inputs in degraded patterns, the corresponding neurons do not receive pattern input, and get their inputs solely from lateral connections, from which they must reconstruct the correct, expected output values. Outputs are read directly from the activation of the neurons. The network's performance is evaluated only on the final time step, by computing the loss as the summed squared error between the final network output and the correct expected pattern (that is, the non-degraded version of the degraded input pattern). The gradient of this error over the $w_{i,j}$ and $\alpha_{i,j}$ coefficients is then computed by backpropagation, and these coefficients are optimized through an Adam solver (Kingma & Ba, 2015) with learning rate 0.001. For this experiment, we use the simple decaying Hebbian formula for updating Hebbian traces (equation 2). Note that the network has two trainable parameters (w and α) for each connection, summing up to $1{,}001 \times 1{,}001 \times 2 = 2{,}004{,}002$ trainable parameters.

Figure 2 shows the result of 10 runs with different random seeds. Error (defined as the proportion of bits that have the wrong sign) converges to a low, residual value (<1%) within about 200 episodes.

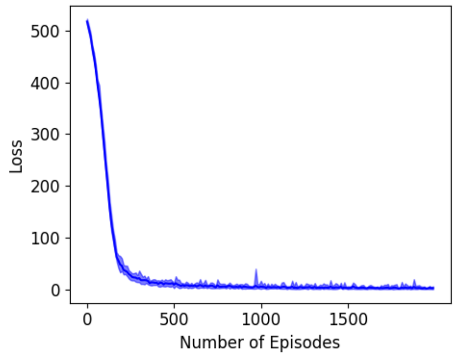


Figure 2: Learning curve for 1,000-bit pattern memorization (10 runs shown: shaded area indicates minimum and maximum loss, thick curve indicates mean loss).

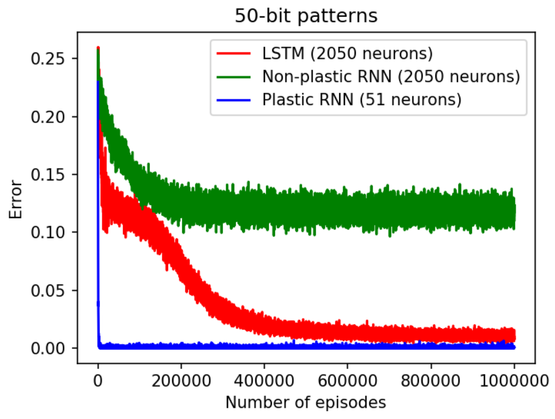


Figure 3: Learning curve for typical runs for 50-bit patterns, using a non-plastic RNN with 2,050 neurons (green curve), an LSTM with 2,050 neurons (red curve), and a differentiable plastic-weight network with the same parameters but only 51 neurons (blue curve).

4.2. The importance of being plastic: a comparison with non-plastic recurrent networks

In principle, this task (like any computable task) could be solved by a non-plastic recurrent network, although the non-plastic networks will require additional neurons to store previously seen patterns. However, despite much exploration, we were unable to succeed in solving this task with a non-plastic RNN or LSTM (Hochreiter & Schmidhuber, 1997). We could only succeed by reducing the pattern size to 50 bits (down from 1,000), showing only 2 patterns per episode (rather than 5), and presenting them for only 3 time steps. The best results required adding 2,000 extra neurons (for a total of 2,050 neurons). Training error over episodes is shown in Figure 3. For the non-plastic RNN, the error essentially flatlines at a high level (green curve). The LSTM solves the task, imperfectly, after about 500,000 episodes (red curve). For comparison, the blue curve shows performance on the exact same problem, architecture, and parameters, but restoring plastic connections. The network solves the task very quickly, reaching mean error below .01 within 2,000 episodes, which is 250 times faster than the LSTM.

Thus, for this specific task, plastic recurrent networks seem considerably more powerful than LSTMs. Although this task is known to be well-suited for plastic recurrent networks (Hopfield, 1982), this result raises the question of which other domains might benefit from the differentiable plasticity approach over current LSTM models (or even by adding plasticity to LSTM models).

4.3. Pattern memorization: Natural images

As a more challenging test, we applied our method to the problem of memorizing natural images with graded pixel values, which contain much more information per element. Images are from the CIFAR-10 database, which contains 60,000 images of size 32 by 32 pixels (i.e. 1,024 pixels in total), converted to grayscale pixels between 0 and 1.0. The architecture is largely similar to the one described above, with 1,025 neurons in total, leading to $2 \times 1{,}025 \times 1025 = 2{,}101{,}250$ parameters. Each episode included 3 pictures, shown 3 times (in random order each time) for 20 timesteps each time, with 3 time steps of zero input between image presentations. To prevent a trivial solution consisting in simply reconstructing each missing pixel as the average of its neighbors (which the high autocorrelation of natural images might make viable), images are degraded by zeroing out one full contiguous half of the image (either top or bottom half).

Figure 4a shows the behavior of the trained network on a withheld test set of images not seen during training. The last column shows the final output of the network, i.e. the reconstructed image. The model has successfully learned to perform the non-trivial task of memorizing and reconstructing previously unseen natural images.

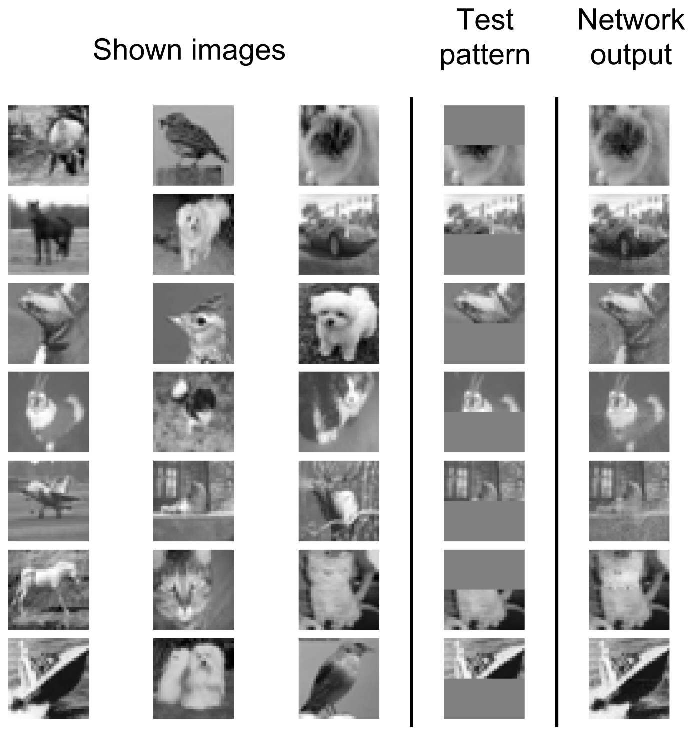
Figure 4b shows the final matrices of weights (top) and plasticity coefficients (bottom) produced by training. The plasticity matrix (bottom) shows considerable structure, in contrast to the homogenous plasticity of traditional Hopfield networks (Hopfield, 1982). Some of the structure (diagonal lines) is related to the high correlation of neighboring pixels, while other aspects (alternating bands near the midsection) result from the choice to use half-field zeroing in test images. We hypothesize that the wide alternating bands near the midsection support fast clearing of reverberating network activity when a test stimulus is presented (see Supplementary Materials).

While Figure 4b shows that the learned network has structure, it could be that this structure is merely an artifact of the learning process, with no inherent usefulness. To test this possibility, we compare the full plastic network against a similar architecture with yoked weights – that is, all connections share the same α coefficient. Thus plasticity is still trainable, but as a single parameter that is shared across all connections. Interestingly, because we use the simple decaying Hebbian formulation here, this yoked-plasticity architecture is equivalent to an "optimal" fast-weights network (Ba et al., 2016) – that is, a fast-weights network in which both the relative importance of the fast weights (α) and their decay rate (η) are trained by gradient descent. The result of this comparison is shown in Figure 5. Independent plasticity coefficients for each connection improve performance, showing that the structure observed in Figure 4a is actually useful, and constitutes a novel architecture for memorization and completion of natural images under these settings.

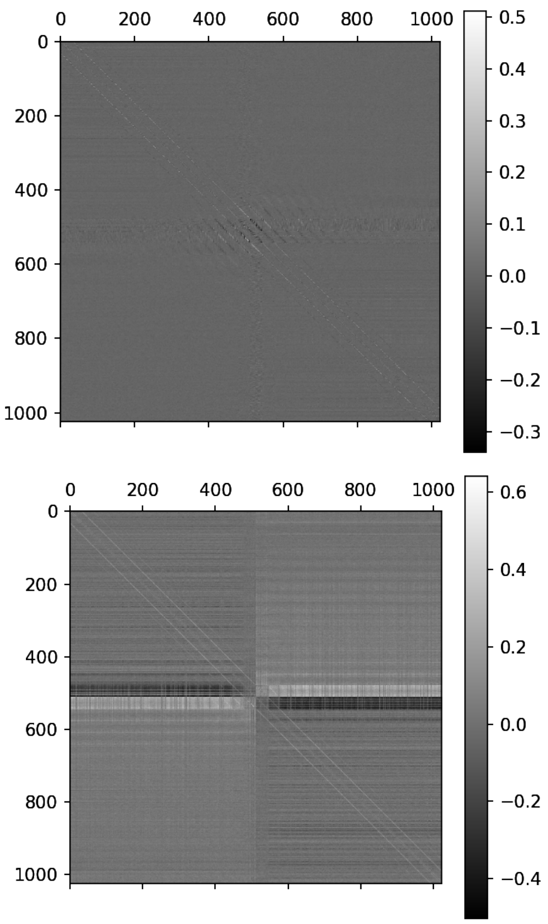
4.4. One-shot pattern classification: Omniglot task

While fast pattern memorization is a complex task, it is important to assess whether differentiable plasticity can handle a wider range of tasks. To test this, we first apply our approach to the standard task for one-shot and few-shot learning, namely, the Omniglot task.

The Omniglot dataset (Lake et al., 2015) is a collection of handwritten characters from various writing systems, including 20 instances each of 1,623 different handwritten characters, written by different subjects. This dataset is the basis for a standard one-shot and few-shot learning task, organized as follows: in each episode, we randomly select N character classes, and sample K instances from each class (here we use N=5 and K=1, i.e. five-way, one-shot learning). We show each of these instances, together with the class label (from 1 to N), to the model. Then, we sample a new, unlabelled instance from one of the N classes and show it to the model. Model performance is defined as the model's



(a) Typical image reconstruction results from a withheld test set (not seen during training). Each row is a full episode.



(b) Matrices of baseline weights $w_{i,j}$ (top) and plasticity coefficients $\alpha_{i,j}$ (bottom) after training. Each column describes the input to a single cell, and vertically adjacent entries describe inputs from horizontally adjacent pixels in the image. Notice the significant structure present in both matrices (best viewed electronically by zooming).

Figure 4

accuracy in classifying this unlabelled example.

Our base model uses the same base architecture as in previous work (Vinyals et al., 2016; Finn et al., 2017; Snell et al., 2017; Mishra et al., 2017) : 4 convolutional layers with 3×3 receptive fields and 64 channels. As in (Finn et al., 2017), all convolutions have a stride of 2 to reduce dimensionality between layers. The output of this network is a single vector of 64 features, which feeds into a N-way softmax. Concurrently, the label of the current character is also fed as a one-hot encoding to this softmax layer, guiding the correct output when a label is present.

There are several ways to introduce plasticity in this architecture. Here, for simplicity, we chose to restrict plasticity solely to the weights from the final layer to the softmax layer, leaving the rest of the convolutional embedding non-plastic. Thus, across many training episodes, we expect the convolutional architecture to learn an adequate discriminant between arbitrary handwritten characters. Meanwhile, the plastic weights between the convolutional network and the softmax should learn to memorize associations between observed patterns and outputs, which are directly influenced by the labels when these are present.

Following common practice, we augment the dataset with rotations by multiples of 90°. We divide the dataset into 1,523 classes for training and 100 classes (together with their augmentations) for testing. We train the networks with an Adam optimizer (learning rate 3×10^{-4}) over 1,000,000 episodes. For this experiment, we use Oja's rule (equation 3) and yoked α across connections in the plastic layer, which seemed to improve stability. To evaluate final model performance, we train 10 models with different random seeds, then test each of those on 100 episodes using the (previously

Table 1: Results for the 5-way, 1-shot omniglot tasks, including recent reported results and the new differentiable plasticity (DP) result (± indicates 95% CI). Note that these reports describe widely varying approaches and model sizes (see text).

VINYALS ET AL. (MATCHING NETWORKS) (VINYALS ET AL., 2016)	SNELL ET AL. (PROTONETS) (SNELL ET AL., 2017)	FINN ET AL. (MAML) (FINN ET AL., 2017)	MISHRA ET AL. (SNAIL) (MISHRA ET AL., 2017)	DP (OURS)
98.1 %	97.4%	98.7% ± 0.4%	99.07% ± 0.16	98.5% ± 0.57

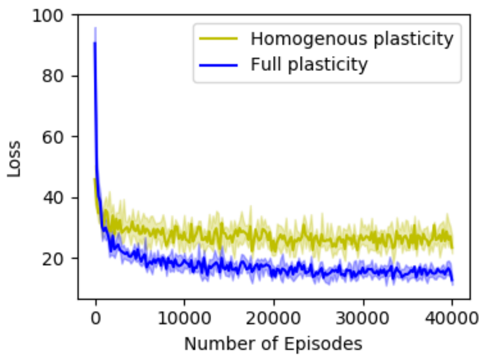
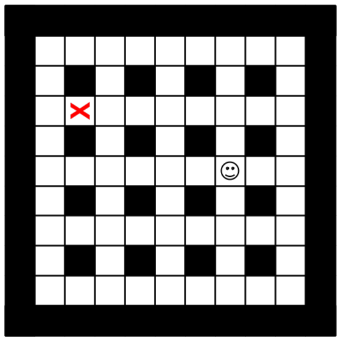


Figure 5: Median and inter-quartile range over 10 runs for a plastic network with independent α (blue curve) or yoked (homogenous) α, equivalent to an optimal fast-weights network (yellow curve). Independent α coefficients for each connection improve performance, by allowing useful structure in the plasticity matrix.

unseen) test classes.

The overall accuracy (i.e. proportion of episodes with correct classification, aggregated over all test episodes of all runs) is 98.5%, with a 95% confidence interval across runs of 0.57%. The median accuracy across each of the 10 runs is 99%, indicating consistency in learning. Table 1 compares our results with those reported in recent papers. All these reports made use of the simple convolutional embedding described above, which was introduced by Vinyals et al. (2016). However, these approaches differ widely in terms of computational cost and number of parameters. Our results are similar to those reported for the computationally intensive MAML method (Finn et al., 2017) and the classification-specialized Matching Networks method (Vinyals et al., 2016), and slightly below those reported for the SNAIL method (Mishra et al., 2017), which trains a whole additional temporal-convolution network on top of the convolutional architecture described above, thus adding many more parameters. We conclude that simply adding a few plastic connections in the output of the network (for a total addition of $64 \times 5 = 320$ parameters, out of 111,746 parameters in the overall network) allows for competitive one-shot learning over arbitrary man-made visual symbols.



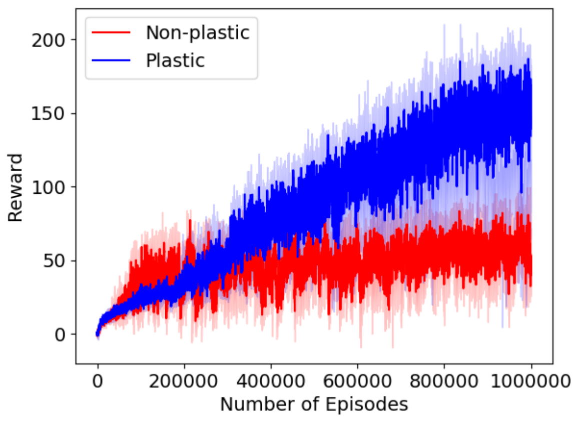


Figure 6: Top: schematic representation of the maze, indicating the agent location (smiley) and the reward location (red cross, for illustration only: the reward is not visible to the agent). Bottom: Training curves for the maze exploration task: median and inter-quartile range of reward over 15 runs for each episode.

4.5. Reinforcement learning: Maze exploration task

Simple recurrent neural networks can be trained to perform reinforcement learning tasks, in such a way that the network implements its own self-contained reinforcement learning algorithm during each episode (Wang et al., 2016; Duan et al., 2016). Differentiable plasticity might improve the learning abilities of these networks. To test this possibility, we devise a simple maze exploration task.

The maze is composed of 9×9 squares, surrounded by walls, in which every other square (in either direction) is occupied by a wall. Thus the maze contains 16 wall squares, arranged in a regular grid (Figure 6, top). The shape of the maze is fixed and unchanging over the whole task. At each episode, one non-wall square is randomly chosen as the reward location. When the agent hits this location, it receives a large reward (10.0) and is immediately transported to a *random* location in the maze (we also provide a small negative reward of -0.1 every time the agent tries to walk into a wall). Each episode lasts 250 time steps, during which the agent must accumulate as much reward as possible. The reward location is fixed within an episode and randomized across episodes. Note that the reward is invisible to the agent, and thus the agent only knows it has hit the reward location by the activation of the reward input at the next step (and possibly by the teleportation, if it can detect it).

Inputs to the agent consist of a binary vector describing the 3×3 neighborhood centered on the agent (each element being set to 1 or 0 if the corresponding square is or is not a wall), together with the reward at the previous time step, following common practice (Wang et al., 2016). The architecture is a simple recurrent network with 100 neurons, with a softmax layer on top of it to select between the 4 possible actions (up, right, left or down). As in (Wang et al., 2016), we use A2C (i.e. a single-thread, non-parallel variant of the A3C policy gradient search algorithm (Mnih et al., 2016)) to meta-train the network.

For each condition (plastic and non-plastic), we perform 15 runs with different random seeds. To alleviate the high noise in reward, we smooth the rewards for each run with a moving average over 10 episodes (this is only for post-hoc illustration and evaluation: no smoothing occurs during the learning). As shown in Figure 6 (bottom), plasticity strongly improves performance in this maze-learning task. The curves suggest that simple RNNs get "stuck" on a sub-optimal strategy, while plastic networks can learn higher-performance strategies. Thus, differentiable plasticity can successfully handle reinforcement learning problems.

5. Discussion

The idea that an optimized form of plasticity can play a key role in learning is entirely natural, which makes it interesting that it is so rarely exploited in modern neural networks. While supervised learning of fixed networks provide a powerful option for general learning, task-specific learning from experience may greatly improve performance on real-world problems. Plasticity is literally the natural choice for this kind of learning, and the results in this paper highlight that indeed simple plastic models can provide the scaffolding for learning to learn. Furthermore, they reveal for the first time that gradient descent itself (without the need for any evolutionary process) can optimize the plasticity coefficients of such a meta-learning system, raising the novel prospect that all the progress and power gained through recent years of deep learning research can be brought to bear on training and discovering novel plastic structures.

The experiments show that in fact this kind of meta-learning can vastly outperform alternative options on some tasks. Such plastic networks also exhibit the ability to fine-tune plasticity coefficients in ways unavailable to fast weight networks, and the potential advantage of this added capability was shown empirically. Differentiable plasticity also performed competitively in the Omniglot task, despite its relative simplicity and compactness (in terms of added parameters), which suggests a unique profile of trade-offs previously unexamined in the field.

To gain a comprehensive understanding of the implications of this new tool will require a broad program of research – almost any meta-learning problem is at least eligible for its application, and even outside meta-learning it could prove helpful for certain challenges. For example, it might complement recurrent structures in arbitrary temporal or sequential domains. Plastic LSTMs are conceivable future models. Furthermore, the breadth of possible plasticity models is wide even on its own. For example, neuromodulation (the control of moment-to-moment plasticity by network activity) has been shown to improve the performance of plastic neural networks designed by evolutionary algorithms (Soltoggio et al., 2008). Because our framework explicitly parametrizes plasticity, it is well-suited to exploring neuromodulatory approaches (for example by making η or α depend in part on the activity of some neurons). It is known that simple recurrent neural networks can be trained to perform reinforcement learning (Wang et al., 2016), and the new results herein show that simple plasticity already improves learning performance on a simple such task; however, the highly elaborate system of neuromodulation implemented in animal brains (Frank et al., 2004) is unlikely to be accidental, and incorporating neuromodulation in the design of neural networks may be a key step towards flexible decision-making. The prospects for this and other such ambitious enterprises are at least more conceivable now with the evidence that meta-learning through plasticity is achievable through gradient descent.

6. Conclusion

In this paper we introduced a framework for endowing neural networks with trainable, differentiable Hebbian plasticity. This framework enables the training of networks that can keep learning autonomously from their ongoing experience, storing high-dimensional information in their plastic connection weights. The framework is flexible, allowing for multiple specific implementations of Hebbian plasticity. The results demonstrate the feasibility of training structural plasticity in million-parameter recurrent networks, greatly improving performance over non-plastic networks on some tasks, while remaining competitive with more complex algorithms and architectures on others. This largely unexplored paradigm, with strong inspiration from biological learning mechanisms, offers a new direction for time-dependent learning in general, and meta-learning in particular.

References

Ba, Jimmy, Hinton, Geoffrey E, Mnih, Volodymyr, Leibo, Joel Z, and Ionescu, Catalin. Using fast weights to attend to the recent past. In Lee, D. D., Sugiyama, M., Luxburg, U. V., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29*, pp. 4331–4339. Curran Associates, Inc., 2016. URL `http://papers.nips.cc/paper/6057-using-fast-weights-to-attend-to-the-recent-past.pdf`.

Dayan, Peter and Abbott, Laurence F. *Theoretical neuroscience*, volume 806. Cambridge, MA: MIT Press, 2001.

Duan, Yan, Schulman, John, Chen, Xi, Bartlett, Peter L., Sutskever, Ilya, and Abbeel, Pieter. Rl^2: Fast reinforcement learning via slow reinforcement learning. 2016. URL `http://arxiv.org/abs/1611.02779`.

Finn, Chelsea, Abbeel, Pieter, and Levine, Sergey. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pp. 1126–1135, 2017.

Frank, Michael J, Seeberger, Lauren C, and O'reilly, Randall C. By carrot or by stick: cognitive reinforcement learning in parkinsonism. *Science*, 306(5703):1940–1943, 2004.

Graves, Alex, Wayne, Greg, and Danihelka, Ivo. Neural turing machines. October 2014.

Hebb, Donald O. The organization of behavior: a neuropsychological theory. 1949.

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

Hochreiter, Sepp, Younger, A, and Conwell, Peter. Learning to learn using gradient descent. *Artificial Neural Networks—ICANN 2001*, pp. 87–94, 2001.

Hopfield, John J. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

Kingma, Diederik P. and Ba, Jimmy. Adam: A method for stochastic optimization. In *3rd International Conference for Learning Representations*. 2015. URL `https://arxiv.org/abs/1412.6980`.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pp. 1097–1105, 2012.

Lake, Brenden M, Salakhutdinov, Ruslan, and Tenenbaum, Joshua B. Human-level concept learning through probabilistic program induction. *Science*, 350(6266):1332–1338, 2015.

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(7394): 381, 2012.

Martin, Stephen J, Grimwood, Paul D, and Morris, Richard GM. Synaptic plasticity and memory: an evaluation of the hypothesis. *Annual review of neuroscience*, 23 (1):649–711, 2000.

Miconi, T. Backpropagation of hebbian plasticity for continual learning. In *NIPS Workshop on Continual Learning*, 2016.

Mishra, Nikhil, Rohaninejad, Mostafa, Chen, Xi, and Abbeel, Pieter. A simple neural attentive meta-learner. In *NIPS Workshop on Meta-Learning*, 2017.

Mnih, Volodymyr, Kavukcuoglu, Koray, Silver, David, Rusu, Andrei A, Veness, Joel, Bellemare, Marc G, Graves, Alex, Riedmiller, Martin, Fidjeland, Andreas K, Ostrovski, Georg, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

Mnih, Volodymyr, Badia, Adria Puigdomenech, Mirza, Mehdi, Graves, Alex, Lillicrap, Timothy, Harley, Tim, Silver, David, and Kavukcuoglu, Koray. Asynchronous methods for deep reinforcement learning. In *International Conference on Machine Learning*, pp. 1928–1937, 2016.

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

Santoro, Adam, Bartunov, Sergey, Botvinick, Matthew, Wierstra, Daan, and Lillicrap, Timothy. One-shot learning with Memory-Augmented neural networks. 19 May 2016.

Schmidhuber, J. Reducing the ratio between learning complexity and number of time varying variables in fully recurrent nets. In Gielen, Stan and Kappen, Bert (eds.), *ICANN '93: Proceedings of the International Conference on Artificial Neural Networks Amsterdam, The Netherlands 13–16 September 1993*, pp. 460–463. Springer London, London, 1993. ISBN 978-1-4471-2063-6. doi: 10.1007/978-1-4471-2063-6_110. URL `https://doi.org/10.1007/978-1-4471-2063-6_110`.

Silver, David, Huang, Aja, Maddison, Chris J, Guez, Arthur, Sifre, Laurent, Van Den Driessche, George, Schrittwieser, Julian, Antonoglou, Ioannis, Panneershelvam, Veda, Lanctot, Marc, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587): 484–489, 2016.

Snell, Jake, Swersky, Kevin, and Zemel, Richard. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 4080–4090, 2017.

Soltoggio, Andrea, Bullinaria, John A, Mattiussi, Claudio, Dürr, Peter, and Floreano, Dario. Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In *Proceedings of the 11th international conference on artificial life (Alife XI)*, number LIS-CONF-2008-012, pp. 569–576. MIT Press, 2008.

Soltoggio, Andrea, Stanley, Kenneth O., and Risi, Sebastian. Born to learn: the inspiration, progress, and future of evolved plastic artificial neural networks. 2017. URL `http://arxiv.org/abs/1703.10371`.

Sukhbaatar, Sainbayar, Szlam, Arthur, Weston, Jason, and Fergus, Rob. End-To-End memory networks. In Cortes, C, Lawrence, N D, Lee, D D, Sugiyama, M, and Garnett, R (eds.), *Advances in Neural Information Processing Systems 28*, pp. 2440–2448. Curran Associates, Inc., 2015.

Thrun, Sebastian and Pratt, Lorien. Learning to learn. In Thrun, Sebastian and Pratt, Lorien (eds.), *Learning to Learn*, chapter Learning to Learn: Introduction and Overview, pp. 3–17. Kluwer Academic Publishers, Norwell, MA, USA, 1998. ISBN 0-7923-8047-9. URL `http://dl.acm.org/citation.cfm?id=296635.296639`.

Vinyals, Oriol, Blundell, Charles, Lillicrap, Tim, Wierstra, Daan, et al. Matching networks for one shot learning. In *Advances in Neural Information Processing Systems (NIPS)*, pp. 3630–3638, 2016.

Wang, Jane X., Kurth-Nelson, Zeb, Tirumala, Dhruva, Soyer, Hubert, Leibo, Joel Z., Munos, Rémi, Blundell, Charles, Kumaran, Dharshan, and Botvinick, Matt. Learning to reinforcement learn. 2016.