Meta-Learning Backpropagation And Improving It

Anonymous Author(s)
Affiliation
Address
email

Abstract

In the past a large number of variable update rules have been proposed for meta learning such as fast weights, hyper networks, learned learning rules, and meta recurrent neural networks. We unify these architectures by demonstrating that a single weight-sharing and sparsity principle underlies them that can be used to express complex learning algorithms. We propose a simple implementation of this principle, the Variable Shared Meta RNN, and demonstrate that it allows implementing neuronal dynamics and backpropagation solely by running the recurrent neural network in forward-mode. This offers a direction for backpropagation that is biologically plausible. Then we show how backpropagation itself can be further improved through meta-learning. That is, we can use a human-engineered algorithm as an initialization for meta-learning better learning algorithms.

1 Introduction

The shift from standard machine learning to meta learning involves learning the learning algorithm itself, reducing the burden on the human designer to craft appropriate learning algorithms [Schmidhuber, 1987]. Modern meta learning has primarily focused on narrow task-distributions such as few-shot learning [Finn et al., 2017] or adaptation to slightly different environments or goals [Houthooft et al., 2018]. This is in stark contrast to human engineered algorithms that generalize across a wide range of datasets or environments. Very recently, MetaGenRL and related approaches [Kirsch et al., 2020, Alet et al., 2020, Oh et al., 2020] demonstrated that meta learning can also be successful in generating algorithms that generalize across significantly different environments, such as from toy environments to Mujoco and Atari. Unfortunately, the large number of still human-designed and unmodifiable inner-loop components in these algorithms (e.g., backpropagation) remains a serious limitation of such approaches.

Is it possible to implement modifiable versions of backpropagation or related algorithms as part of activation dynamics of a neural net (NN), instead of inserting them as fixed routines? Here we propose the principle of sparse variable-sharing in NNs to provide such a general mechanism. We generalize the arguably simplest meta learner, the meta recurrent NN [Hochreiter et al., 2001, Duan et al., 2016, Wang et al., 2016], and view end-to-end-differentiable fast weights [Schmidhuber, 1992, 1993, Ba et al., 2016, Schlag and Schmidhuber, 2017], hyper networks [Ha et al., 2016]), learned learning rules [Bengio et al., 1992, Gregor, 2020, Randazzo et al., 2020], and hebbian-like synaptic plasticity [Miconi et al., 2018, 2019, Najarro and Risi, 2020] as special cases. Our mechanism allows for implementing neuronal dynamics (neurons as an emerging phenomenon) and backpropagation solely in the forward-dynamics of a recurrent NN and thus allows for meta-optimization of backprop-like algorithms. Meta-learning experiments show additional improvements of the learned algorithm. This may also enable future research in biologically plausible versions of backpropagation.

Our computational perspective blurs the semantic distinction between activations of neurons, weights or synapses of connections, and learning algorithms. We collapse all of this into the concept of parallel interactions between variables linked through variable sharing.

Submitted to the 4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada. Do not distribute.

2 Background

Deep learning based meta learning that does not rely on fixed gradient descent in the inner loop has historically fallen into two categories, 1) Learnable weight update mechanisms that allow changing the parameters of a neural network to implement a learning rule (FWs / LLRs) and 2) Meta learning implemented in recurrent neural networks (Meta RNNs).

Learnable weight update mechanisms: Fast weights & Learned learning rules (FWs / LLRs) In a standard neural network the weights (and biases) are updated by a fixed learning algorithm. This framework can be extended to meta-learning by defining an explicit architecture that allows modifying these weights. This weight-update architecture augments a standard neural network architecture, similar to how a learning algorithm such as backpropagation would be defined external to the network architecture. Neural networks that generate or modify the weights of another or the same neural network are known as fast weights [Schmidhuber, 1992, 1993, Ba et al., 2016, Schlag and Schmidhuber, 2017], hypernetworks [Ha et al., 2016], synaptic plasticity [Miconi et al., 2018, 2019, Najarro and Risi, 2020], or learned learning rules [Bengio et al., 1992, Gregor, 2020, Randazzo et al., 2020]. Often these architectures make use of local Hebbian-like update rules and / or outer-products and we summarize this category as FWs / LLRs. In FWs / LLRs the variables $\bar{V}_L$ that are subject to learning are the weights of the network, whereas the meta-variables $\bar{V}_M$ that implement the learning algorithm are defined by the weight-update architecture. Note that the dimensionality of $\bar{V}_L$ and $\bar{V}_M$ can be defined independently from each other and often $\bar{V}_M$ are reused coordinate-wise for $\bar{V}_L$ resulting in $|\bar{V}_L| \gg |\bar{V}_M|$.

Meta Learning in RNNs (Meta RNNs) It was shown that recurrent neural networks can learn to encode learning algorithms in their activations when the reward is given as an input. We refer to this as Meta RNNs [Hochreiter et al., 2001, Duan et al., 2016, Wang et al., 2016]. They are conceptually simpler than FWs / LLRs as no additional weight-update rules with many degrees of freedom need to be defined. In Meta RNNs $\bar{V}_L$ are the RNN activations and $\bar{V}_M$ are the parameters for the RNN. Note that an RNN with N neurons will yield $|\bar{V}_L| \in O(N)$ and $|\bar{V}_M| \in O(N^2)$. This means that the learning algorithm is largely overparameterized whereas the available memory to perform learning with is very small making this approach prone to overfitting [Kirsch et al., 2020]. In order to meta-learn a simple and generalizing learning algorithm we would benefit from $|\bar{V}_L| \gg |\bar{V}_M|$. Previous approaches have tried to mend this issue by adding architectural complexity through additional memory mechanisms [Sun, 1991, Mozer and Das, 1993, Santoro et al., 2016, Mishra et al., 2018].

3 Variable Shared Meta RNNs (Var-Shared RNNs)

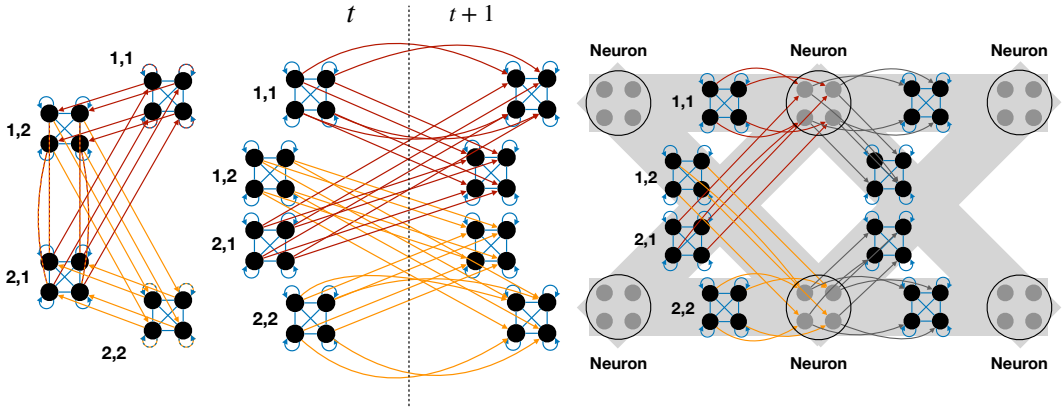
In Var-Shared RNNs we build on the simplicity of Meta RNNs while retaining $|\bar{V}_L| \gg |\bar{V}_M|$ from FWs / LLRs. We show that the principle of sparse-variable sharing in $\bar{V}_M$ generalizes FWs / LLRs and Meta RNNs and can thus still maintain the expressiveness of these approaches. Further, we discuss the emergence of neuronal dynamics and learning algorithms.

From Meta RNNs to Var-Shared RNNs We begin by formalizing Meta RNNs which are often implemented as gated variants such as the LSTM [Gers et al., 2000, Hochreiter and Schmidhuber, 1997] or the GRU [Cho et al., 2014]. For notational simplicity we consider a vanilla RNN. Let $s \in \mathbb{R}^N$ be the hidden state of a recurrent neural network. The update for an element $j \in \{1, \dots, N\}$ is given by

$$s_j \leftarrow \sigma(\mathfrak{b}_j + \sum_i s_i W_{ij}) \tag{1}$$

where σ is a non-linear activation function, $W \in \mathbb{R}^{N \times N}$, and $\mathfrak{b} \in \mathbb{R}^N$. For simplicity we omit inputs by assuming a subset of s to be given by the environment observations and rewards (or supervised data). Conventionally, we would refer to s as (hidden) neurons and to W as the weights. As we will see later, these semantics are not necessarily fitting.

We now introduce variable sharing (reusing W) into the recurrent neural network by duplicating the computation along two batch axes A, B (here $A = B$) giving $s \in \mathbb{R}^{A \times B \times N}$. For $a \in \{1, \dots, A\}, b \in$



(a) Connectivity (b) V unrolled (c) Neuronal interpretation

Figure 1: The connectivity of Var-Shared RNNs gives rise to a neuronal interpretation (all figures show the same computation): (a) shows $N = 4$ states s duplicated 4 times ($A = 2, B = 2$) with blue coloring for W and red / orange for interactions V, we have dropped the full connectivity of V for readability (4 connections instead of 16); (b) we unrolled the recurrence in V across two time steps; (c) intermediate computation results are drawn in gray circles which are then added to different state clusters (gray arrows). These intermediate results can be interpreted as multi-dimensional neurons while V describes synaptic connectivity and (a subset of) s defines the synaptic weight / state.

$\{1, \dots, B\}$ we have

$$s_{abj} \leftarrow \sigma(\mathfrak{b}_j + \sum_i s_{abi} W_{ij}). \tag{2}$$

This computation describes $A \cdot B$ independent computation paths ($A \cdot B$ independent RNNs) which we connect using an interaction term

$$s_{abj} \leftarrow \sigma(\mathfrak{b}_j + \sum_i s_{abi} W_{ij} + \underbrace{\sum_{c,i} s_{cai} V_{ij}}_{\text{interactions}}) = \sigma(\mathfrak{b}_j + \underbrace{\sum_{c,d,i} s_{cdi} \tilde{W}_{cdiabj}}_{\text{sparse-shared equivalent}}) \tag{3}$$

where $V \in \mathbb{R}^{N \times N}$. This recursion constitutes the Var-Shared RNN. It is trivial that if $A = 1$ and $B = 1$ Equation 3 recovers the original Meta RNN Equation 1. In the general case, we can derive the equivalent right hand-side term that corresponds to a standard recurrent neural network with a single matrix $\tilde{W}$ that has zero entries and shared entries (where the six axes can be flattened to the regular two axes). For both terms to be equivalent $\tilde{W}$ must satisfy (derivation in Appendix A)

$$\tilde{W}_{cdiabj} = \begin{cases} V_{ij}, & \text{if } d = a \wedge (d \neq b \vee c \neq a). \\ W_{ij}, & \text{if } d \neq a \wedge d = b \wedge c = a. \\ V_{ij} + W_{ij}, & \text{if } d = a \wedge d = b \wedge c = a. \\ 0, & \text{otherwise.} \end{cases} \tag{4}$$

Emerging neuronal dynamics embedded in LSTMs Besides variable-sharing, Equation 3 has a surprising alternative interpretation. The state variable s (or subsets thereof) can be interpreted as the multi-dimensional weights (synapses) of a neural network which are updated by some mechanism defined by the variables W and V. This is visualized in Figure 1. Both neural evaluation (forward pass in a regular neural network) and learning can be integrated in the recurrent dynamics. In the case of backpropagation, this would correspond to the forward and backward passes being implemented purely by unrolling the RNN. Information that needs to be stored for credit assignment, e.g. the activations from the forward pass in backpropagation are part of the state s. While computationally not strictly necessary (two time steps can accomplish the same), an additional interaction term can be added to make the mapping to a backward pass more straight-forward

$$s_{abj} \leftarrow \sigma(\mathfrak{b}_j + \sum_i s_{abi} W_{ij} + \underbrace{\sum_{c,i} s_{cai} V_{ij}}_{\text{forward interactions}} + \underbrace{\sum_{c,i} s_{bci} U_{ij}}_{\text{backward interactions}}) \tag{5}$$

and corresponds to interaction connections in the opposite direction in Figure 1c.

In a standard NN weights and activations have multiplicative interactions. For Var-Shared RNNs to mimic such neuronal dynamics these multiplicative interactions would have to exist between parts of the state s, e.g. by modifying Equation 3 to

$$s_{abj} \leftarrow \sigma(\mathfrak{b}_j + \sum_i s_{abi} W_{ij} \ \cdot\!/\!+ \sum_{c,i} s_{cai} V_{ij}). \tag{6}$$

Fortunately, LSTMs already incorporate such multiplicative interactions using gating in the context vector c recurrence

$$c \leftarrow f \cdot c + i \cdot g \tag{7}$$

where f, i, and g are a function of the input and hidden state. As we will show in Section 4.1 LSTMs can implement neuronal dynamics also in practice.

Interpreting Var-Shared RNNs as FWs / LLRs Let θ be the parameters of a neural network and h its activations, then in the most general case we can define FWs / LLRs by a variable update rule

$$\theta \leftarrow f_\phi(\theta, h, D, R) \tag{8}$$

parameterized by ϕ computing the updates on θ given inputs D and (multidimensional) feedback R[1]. In Var-Shared RNNs θ and h can be represented by subsets of s and ϕ is given by W and V, yielding a general RNN update

$$s \leftarrow f_{(W,V)}(s, D, R) \tag{9}$$

resembling one or multiple recursion steps in Equation 3. As long as the sparse-shared RNN is sufficiently large we can thus in principle represent any FWs / LLRs-like algorithm.

The neuronal interpretation from Figure 1 also suggests that Var-Shared RNNs are particularly well suited to implement coordinate-wise applied rules such as local learning rules [Bengio et al., 1992, Gregor, 2020, Randazzo et al., 2020], Hebbian-like differentiable mechanisms [Schmidhuber, 1993, Miconi et al., 2018, 2019], and backpropagation. Let us formally define a parameterized local update equation, the *learning rule*, for the weights w of a neural network,

$$x_b(t+1) = \sigma(\sum_a x_a(t) \cdot w_{ab}(t)) \qquad \text{The neuronal dynamics} \tag{10}$$

$$\Delta w_{ab}(t+1) = f_\theta(x_a(t), x_b(t+1), w_{ab}(t), m_{ab}(t)) \qquad \text{The learning rule} \tag{11}$$

$$w_{ab}(t+1) = w_{ab}(t) + \Delta w_{ab}(t+1) \tag{12}$$

where f_θ is a function such as a neural network parameterized by θ and m_{ab} corresponds to an additional modulation signal such as the rewards in the environment. If we map $w_{ab} := s_{ab0}$, $\theta := (W, V)$, $x_a(t) := \sum_{c,i} s_{cai}(t-1) V_{i0}$, and $x_b(t+1) := \sum_{c,i} s_{cbi}(t) V_{i0}$ then we can rewrite the learning rule to

$$\Delta s_{ab0}(t+l) = f_{(W,V)}(\sum_{c,i} s_{cai}(t-1) V_{i0}, \sum_{c,i} s_{cbi}(t) V_{i0}, s_{ab0}(t), m_{ab}(t)). \tag{13}$$

This is similar to the structure of Equation 3 when the recursion is applied at least twice, i.e. $l \geq 2$. As a universal function approximator, given enough time ticks l and parameters the RNN can implement any such learning rule f.

4 Experiments

In this section we demonstrate the capabilities of Var-Shared RNNs by showing that it can implement neuronal dynamics and backpropagation in their recurrent dynamics on the MNIST and FashionMNIST dataset. We introduce *learning algorithm cloning*, the implementation of a human-engineered algorithm in a neural network. Finally, *meta learning to replace human engineered algorithms* uses these cloned algorithms as an initialization for meta learning. Hyperparameters and training details can be found in the appendix.

[1] In the supervised setting R would correspond to the negative error on the network outputs, e.g. the gradient of the cross entropy with respect to the outputs. In the POMDP RL setting D and R can be summarized by the interaction history $\tau = (o_0, a_0, r_1, o_1, ..., o_{T-1}, a_{T-1}, r_T)$ of length T with observations $o \in O$, actions $a \in A$ and rewards $r \in \mathbb{R}$.

4.1 Var-Shared RNNs can implement neuronal dynamics

In Section 3 we argued that Var-Shared RNNs can implement neuronal dynamics with the help of multiplicative interactions in LSTMs. We experimentally show that an LSTM can compute $y = \tanh(x)w$ for scalars x, y, w by minimizing

$$\mathcal{L}(W, V) := (\tanh(h_1(t))h_0(t) - \sum_i h_i(t+1)V_{i1})^2 \tag{14}$$

using gradient descent on random data where h corresponds to the LSTM hidden vector which is analogous to $s_{a,b}$ for any a, b. The LSTMs parameters are denoted by W while V remain the interaction parameters, and h is updated by the LSTM recursion. The input x is represented by h_1 and w by h_0. That is, we would feed inputs as $x_i = s_{ib1}$ replicated across the second axis B. Equation 14 is trivially extended to a bias $\mathfrak{b}$. Note that h is bounded between $(-1, 1)$ and h_1 is only determined by the interaction term and the network inputs which we discuss in Appendix B. We unroll the LSTM for a single time step to minimize $\mathcal{L}$.

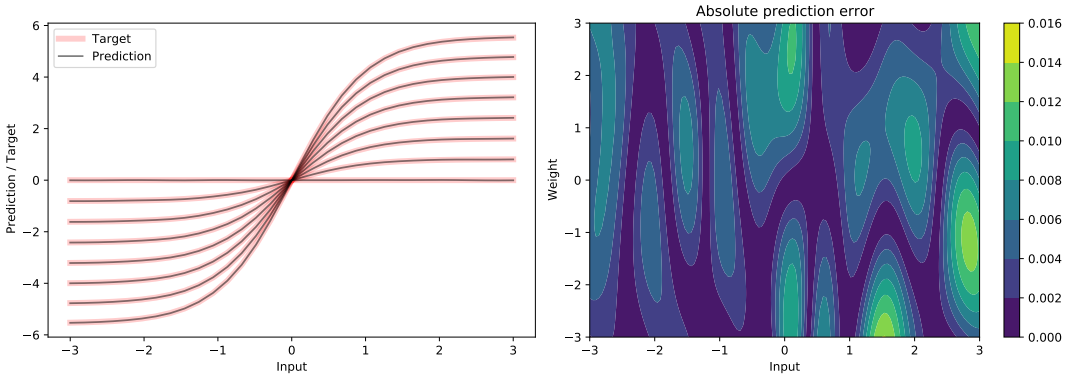


Figure 2: We are optimizing Var-Shared RNNs to implement neuronal dynamics such that for different inputs and weights a tanh-activated multiplicative interaction is produced (left), with different lines for different w. These neuronal dynamics are not exactly matched everywhere (right), but the error is relatively small.

Figure 2 (left) shows how for different inputs x and weights w the LSTM produces the correct target value, including the multiplicative interaction. The heat-map (right) shows that low prediction errors are produced but the target dynamics are not perfectly matched. A perfect fit is not strictly necessary due to the whole network still being a universal function approximator. We repeat these LSTMs in line with Equation 5 to obtain a 'neural network' within a computational network.

4.2 Var-Shared RNNs can implement backpropagation

Backpropagation is often used in supervised learning, unsupervised learning and as a component in many RL algorithms. Thus, it seems desirable to be able to meta-learn backpropagation-like algorithms. Here we demonstrate that Var-Shared RNNs can learn to implement backpropagation. We perform *learning algorithm cloning* in addition to optimizing for neuronal dynamics. Similar to Equation 14 we optimize W and V to update its parameters $w, \mathfrak{b}$ in state s according to $e \cdot \nabla_{w,\mathfrak{b}}[y]$ where e is the error as passed by the backward interaction term from Equation 5. We also optimize the backward interaction term to produce a new error $e' = e \cdot \nabla_x[y]$ such that the composition of these LSTMs in the form of Var-Shared RNNs resembles the backpropagation algorithm according to the chain rule. This requires previous inputs and activations to be fed in the respective time step or the Var-Shared RNN to memorize previous inputs in its state s. For the present experiments we feed the correct inputs.

We now run this Var-Shared RNN with the cloned backpropagation algorithm on the MNIST and Fashion MNIST dataset and observe that it performs learning purely in its forward dynamics, making any explicit gradient calculations obsolete. Figure 3 shows the learning curve on these two datasets without having ever seen any training data achieving around 78% test accuracy on MNIST and 63% on Fashion MNIST. We observe that the loss is decently minimized, albeit regular gradient descent

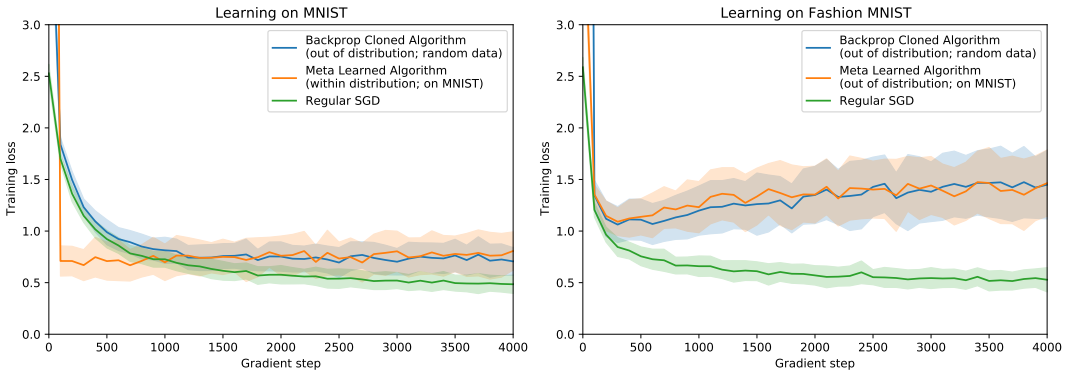


Figure 3: The Var-Shared RNN is optimized to implement backpropagation in its recurrent dynamics on random data, then tested (blue) both on MNIST and Fashion MNIST where it performs learning purely by unrolling the LSTM. We then meta-learn to minimize the loss on MNIST starting from the cloned backpropagation which improves learning speed in the beginning of training (orange) and does not worsen performance when meta-testing out of distribution on Fashion MNIST. Standard deviations are over 60 seeds.

still outperforms our cloned backpropagation. This may be due to difficulties of normalization / scaling of gradients, activations, and implicit learning rates. Similarly, we use shallow networks for the present experiments and are working on the extension to deep networks which we already train for using the implicit chain rule as described above.

As a second step, we use this algorithm encoded in the Var-Shared RNNs parameters as an initialization to perform meta learning, minimizing the loss after 10,000 steps of unrolling with respect to these parameters. We employ CMA-ES for this optimization which is suitable due to few parameters $|\bar{V}_M|$ and large $|\bar{V}_L|$. In Figure 3 we observe that meta learning on MNIST and meta testing on MNIST drastically increases learning speed in the first few gradient steps but then achieves similar performance to the backpropagation cloned algorithm later on. Further investigations are required to outperform standard SGD using meta-learned backpropagation. Surprisingly, when meta testing on a different dataset, Fashion MNIST, the meta-learned solution still performs similar to cloned backpropagation without additional overfitting after meta training on MNIST.

5 Conclusion

Our Variable Shared Meta RNNs use a simple principle of variable sharing and sparsity to express complex learning algorithms. They generalize meta recurrent neural networks, fast weight generators (also used in hyper networks), and learned learning rules, by producing emergent neural dynamics and learning algorithms solely in the forward activation spreading phase of their recurrent dynamics. We show how Variable Shared Meta RNNs can learn to implement the backpropagation algorithm on a small problem of supervised learning. On MNIST it learns to predict well without any human-designed explicit computational graph for gradient calculation. The backpropagation algorithm and its parameter updates are implicitly encoded in the recurrent dynamics. To achieve that, we used the novel technique of *learning algorithm cloning* to (1) implement neuronal dynamics in the form of multiplicative interactions and activation functions and (2) pre-train the net on a separate training set to compute correct gradients and the chain rule. Preliminary experiments show that such cloned learning algorithms can be used as an initialization for *meta learning to replace human-engineered algorithms*. Future experiments will focus on deeper emergent neural networks, reinforcement learning settings, and improvements of meta learning.

References

F. Alet, M. F. Schneider, T. Lozano-Perez, and L. P. Kaelbling. Meta-learning curiosity algorithms. In *International Conference on Learning Representations*, 2020.

J. Ba, G. Hinton, V. Mnih, J. Z. Leibo, and C. Ionescu. Using Fast Weights to Attend to the Recent Past. In *Advances in Neural Information Processing Systems*, pages 4331–4339, 2016.

S. Bengio, Y. Bengio, J. Cloutier, and J. Gecsei. On the optimization of a synaptic learning rule. In *Preprints Conf. Optimality in Artificial and Biological Neural Networks*, pages 6–8, 1992.

K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 6 2014. URL `http://arxiv.org/abs/1406.1078`.

Y. Duan, J. Schulman, X. Chen, P. L. Bartlett, I. Sutskever, and P. Abbeel. RLˆ2: Fast Reinforcement Learning via Slow Reinforcement Learning. *arXiv preprint arXiv:1611.02779*, 2016.

C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 1126–1135, 2017.

F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to Forget: Continual Prediction with LSTM. *Neural Computation*, 12(10):2451–2471, 2000.

K. Gregor. Finding online neural update rules by learning to remember. *arXiv preprint arXiv:2003.03124*, 3 2020. URL `http://arxiv.org/abs/2003.03124`.

D. Ha, A. Dai, and Q. V. Le. HyperNetworks. In *International Conference on Learning Representations*, 2016.

S. Hochreiter and J. Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

S. Hochreiter, A. S. Younger, and P. R. Conwell. Learning to learn using gradient descent. In *International Conference on Artificial Neural Networks*, 2001.

R. Houthooft, R. Y. Chen, P. Isola, B. C. Stadie, F. Wolski, J. Ho, and P. Abbeel. Evolved Policy Gradients. In *Advances in Neural Information Processing Systems*, pages 5400–5409, 2018.

L. Kirsch, S. van Steenkiste, and J. Schmidhuber. Improving Generalization in Meta Reinforcement Learning using Neural Objectives. In *International Conference on Learning Representations*, 2020.

T. Miconi, J. Clune, and K. O. Stanley. Differentiable plasticity: training plastic neural networks with backpropagation. In *International Conference on Machine Learning*, 4 2018.

T. Miconi, A. Rawal, J. Clune, and K. O. Stanley. Backpropamine: training self-modifying neural networks with differentiable neuromodulated plasticity. In *International Conference on Learning Representations*, 2019.

N. Mishra, M. Rohaninejad, and X. U. Chen Pieter Abbeel Berkeley. A Simple Neural Attentive Meta-Learner. In *International Conference on Learning Representations*, 2018.

M. C. Mozer and S. Das. A connectionist symbol manipulator that discovers the structure of context-free languages. In *Advances in neural information processing systems*, pages 863–870, 1993.

E. Najarro and S. Risi. Meta-Learning through Hebbian Plasticity in Random Networks. 2020.

J. Oh, M. Hessel, W. M. Czarnecki, Z. Xu, H. van Hasselt, S. Singh, and D. Silver. Discovering Reinforcement Learning Algorithms. *arXiv preprint arXiv:2007.08794*, 2020.

E. Randazzo, E. Niklasson, and A. Mordvintsev. MPLP: Learning a Message Passing Learning Protocol. 2020.

A. Santoro, S. Bartunov, M. Botvinick, D. Wierstra, and T. Lillicrap. Meta-Learning with Memory-Augmented Neural Networks. In *International conference on machine learning*, pages 1842–1850, 2016.

I. Schlag and J. Schmidhuber. Gated Fast Weights for On-The-Fly Neural Program Generation. In *NIPS Metalearning Workshop*, 2017.

J. Schmidhuber. Evolutionary principles in self-referential learning. Diploma thesis, Institut für Informatik, Technische Universität München, 1987.

J. Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Dynamic Recurrent Networks. *Neural Computation*, 4(1):131–139, 1992. ISSN 0899-7667. doi: 10.1162/neco.1992.4.1.131.

J. Schmidhuber. Reducing the ratio between learning complexity and number of variables in fully recurrent nets. Technical report, 1993.

G. Z. Sun. Neural networks with external memeory stack that learn context-free grammars from examples. In *Proceedings of the Conference on Information Science and Systems, 1991*, pages 649–653. Princeton University, 1991.

J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick. Learning to reinforcement learn. *arXiv preprint arXiv:1611.05763*, 2016.

A Derivations

Theorem 1. *The weight matrices W and V used to compute Var-Shared RNNs from Equation 3 (left side) can be expressed as a standard recurrent neural network with weight matrix $\tilde{W}$ (right side).*

$$s_{abj} \leftarrow \sigma(\mathfrak{b}_j + \sum_i s_{abi} W_{ij} + \underbrace{\sum_{c,i} s_{cai} V_{ij}}_{\textit{interactions}}) = \sigma(\mathfrak{b}_j + \underbrace{\sum_{c,d,i} s_{cdi} \tilde{W}_{cdiabj}}_{\textit{sparse-shared equivalent}}) \tag{3 revisited}$$

The weight matrix $\tilde{W}$ has zero entries and shared entries given by Equation 4.

$$\tilde{W}_{cdiabj} = \begin{cases} V_{ij}, & \textit{if } d = a \wedge (d \neq b \vee c \neq a). \\ W_{ij}, & \textit{if } d \neq a \wedge d = b \wedge c = a. \\ V_{ij} + W_{ij}, & \textit{if } d = a \wedge d = b \wedge c = a. \\ 0, & \textit{otherwise.} \end{cases} \tag{4 revisited}$$

Proof. We rearrange $\tilde{W}$ into two separate weight matrices

$$\sum_{c,d,i} s_{cdi} \tilde{W}_{cdiabj} \tag{15}$$

$$= \sum_{c,d,i} s_{cdi} A_{cdiabj} + \sum_{c,d,i} s_{cdi} (\tilde{W} - A)_{cdiabj}. \tag{16}$$

Then assuming $A_{cdiabj} = (d \equiv b)(c \equiv a) W_{ij}$, where $x \equiv y$ equals 1 iff x and y are equal and 0 otherwise, it holds that

$$\sum_{c,d,i} s_{cdi} A_{cdiabj} = \sum_i s_{abi} W_{ij}. \tag{17}$$

Further, assuming $(\tilde{W} - A)_{cdiabj} = (d \equiv a) V_{ij}$ we yield

$$\sum_{c,d,i} s_{cdi} (\tilde{W} - A)_{cdiabj} = \sum_{c,i} s_{cai} V_{ij}. \tag{18}$$

Finally, solving both conditions for $\tilde{W}$ gives

$$\tilde{W}_{cdiabj} = (d \equiv a) V_{ij} + (d \equiv b)(c \equiv a) W_{ij}, \tag{19}$$

which we rewrite in tabular notation:

$$\tilde{W}_{cdiabj} = \begin{cases} V_{ij}, & \text{if } d = a \wedge (d \neq b \vee c \neq a). \\ W_{ij}, & \text{if } d \neq a \wedge d = b \wedge c = a. \\ V_{ij} + W_{ij}, & \text{if } d = a \wedge d = b \wedge c = a. \\ 0, & \text{otherwise.} \end{cases} \tag{20}$$

Thus, Equation 3 holds and any weight matrices W and V can be expressed by a single weight matrix $\tilde{W}$. □

B Alternative neuronal dynamics

Bounded states in LSTMs In Section 4.1 we noted that in LSTMs h is bounded between $(-1, 1)$. This means that with standard tanh neural dynamics $\tanh(x)w + \mathfrak{b}$ both w and $\mathfrak{b}$ would be limited to magnitude 1. This can be circumvented by choosing a large magnitude, here 20, by which we scale our training target Equation 14 while scaling down w and $\mathfrak{b}$ in the LSTM. Another method we have considered is encoding both parameters using a linear encoder to an arbitrary representation such that the target error can be minimized effectively.

LSTM states and LSTM inputs. In Section 4.1 h_1 denoted the part of the state that corresponds to the 'neuron' input. In practice we use make all interaction term outputs including h_1 an LSTM input, meaning the value is only determined by the interaction term and not the standard dynamics of the LSTM itself. This does not affect the generality of the derived Var-Shared RNNs but simplifies learning.

Parameter	Value
LSTM cell size	32
Learning rate	$5 \cdot 10^{-3}$
Batch size	$32,768$
Iterations	10^6
Random data std	2
Gradient update target norm	10^{-3}
Optimizer	Adam
Loss	Smooth L1

Table 1: Hyper-parameters for learning neural dynamics and backprop cloning

Parameter	Value
Batch size	64
Optimizer	CMA-ES
Iterations	200
Inner steps	10^4
Initial noise std	0.1

Table 2: Hyper-parameters for meta learning

C Other training details

Source code Source code will be made available with the publication of this work.

Batching for Var-Shared RNNs supervised experiments In Section 4.2 we optimize a Var-Shared RNN to implement backpropagation. To stabilize learning at meta-test time we run the RNN on multiple data points (batch size 64) and then average their states as an analogue to batching in standard gradient descent.

Stability during meta-testing In order to prevent exploding states during meta-testing we also clip the LSTM state between -4 and 4. Further, we clamp state-updates with a magnitude larger than 1.

Stacking Var-Shared RNNs In order to get a similar effect to non-recurrent deep feed-forward architectures one can stack multiple Var-Shared RNNs where their states are untied and their parameters are tied.