Meta-Learning Backpropagation And Improving It

Louis Kirsch
The Swiss AI Lab IDSIA, USI, SUPSI
louis@idsia.ch

Jürgen Schmidhuber
The Swiss AI Lab IDSIA, USI, SUPSI
juergen@idsia.ch

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 simply introducing weight-sharing and sparsity to a neural network is sufficient to express powerful learning algorithms. We call this principle *Variable Shared Meta Learning (VS-ML)*. We propose a simple implementation of VS-ML, the *Variable Shared Meta RNN*, and demonstrate that it allows implementing backpropagation solely by running the recurrent neural network in forward-mode. Then we show how we can find new learning algorithms that improve upon backpropagation through meta-learning. The meta-learned learning algorithms do not require explicit gradient calculation and generalize to different datasets.

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. We call the search for such general-purpose learning algorithms General Meta Learning. Unfortunately, the large number of still human-designed and unmodifiable inner-loop components in these prior 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 Variable Shared Meta Learning (VS-ML) principle to provide such a general mechanism. It simply introduces sparsity and weight-sharing in NNs for meta learning. 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, VS-ML, allows for implementing neural forward computation (neurons and weights as an emerging phenomenon) and backpropagation solely in the forward-dynamics of an RNN. Consequently, it enables meta-optimization of backprop-like algorithms. We show that learning algorithms can be meta-learned that outperform online backpropagation and generalize to datasets outside of the meta training distribution.

4th Workshop on Meta-Learning at NeurIPS 2020, Vancouver, Canada.

Our computational perspective blurs the semantic distinction between activations, weights, and learning algorithms. We collapse these concepts into generic variables that parameterize functions and interact with each other. Some of the computation is replicated through variable sharing.

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 V_L that are subject to learning are the weights of the network, whereas the meta-variables V_M that implement the learning algorithm are defined by the weight-update architecture. Note that the dimensionality of V_L and V_M can be defined independently from each other and often V_M are reused coordinate-wise for V_L resulting in $|V_L| \gg |V_M|$.

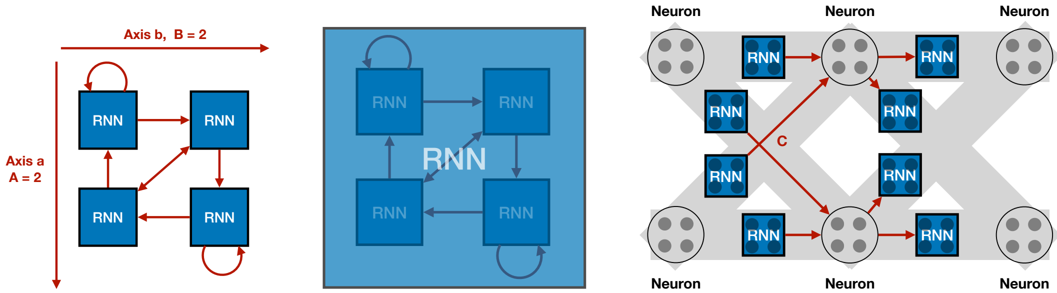
Meta Learning in activations (Meta RNNs) It was shown that recurrent neural networks can learn to encode learning algorithms in their activations when the reward is given as an input. We refer to this as Meta RNNs [Hochreiter et al., 2001, Duan et al., 2016, Wang et al., 2016]. They are conceptually simpler than FWs / LLRs as no additional weight-update rules with many degrees of freedom need to be defined. In Meta RNNs V_L are the RNN activations and V_M are the parameters for the RNN. Note that an RNN with N neurons will yield $|V_L| \in O(N)$ and $|V_M| \in O(N^2)$. 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 $|V_L| \gg |V_M|$. 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 Learning (VS-ML)

In VS-ML we build on the simplicity of Meta RNNs while retaining $|V_L| \gg |V_M|$ from FWs / LLRs. We do this by reusing the same few parameters V_M many times in a neural network (variable sharing) and introducing sparsity in the connectivity. Activations in this neural network V_L then can be interpreted as the weights of a conventional neural network, consequently blurring the distinction between a weight and an activation[1]. This enables V_M to both define how weights are used (forward computation) as well as how they are updated (learning algorithm). Further, we will show that VS-ML generalizes FWs / LLRs and Meta RNNs. We apply VS-ML to RNNs, resulting in the Variable Shared Meta RNN.

From Meta RNNs to VS Meta 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

[1] This is the reason why we refer to V_M as the meta variables and not the weights and call VS-ML variable sharing instead of weight sharing. Likewise, V_L are the learned variables and not the activations.



(a) VS Meta RNNs (b) Viewed as a Meta RNN (c) Viewed as Neural Network

Figure 1: Different perspectives on VS Meta RNNs: (a) The VS Meta RNN consists of many connected RNNs with shared parameters $V_M := \{\hat{b}, W, C\}$. (b) The same VS Meta RNN can be viewed as a single Meta RNN [Hochreiter et al., 2001] where some entries in the weight matrix are shared or zero. (c) Another perspective is that VS Meta RNN implements a neural network where V_M determines the nature of weights and biases, how these are used in the forward computation and the learning algorithm by which those are updated. Neurons correspond to intermediate computation results of the interaction term which are fed back into the RNNs. Weights and neurons are thus implicitly multi-dimensional and a subset of the RNN state s.

$s \in \mathbb{R}^N$ be the hidden state of a recurrent neural network. The update for an element $j \in \{1, \dots, N\}$ is given by

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

where σ is a non-linear activation function, $W \in \mathbb{R}^{N \times N}$, and $\hat{b} \in \mathbb{R}^N$. 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.

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

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

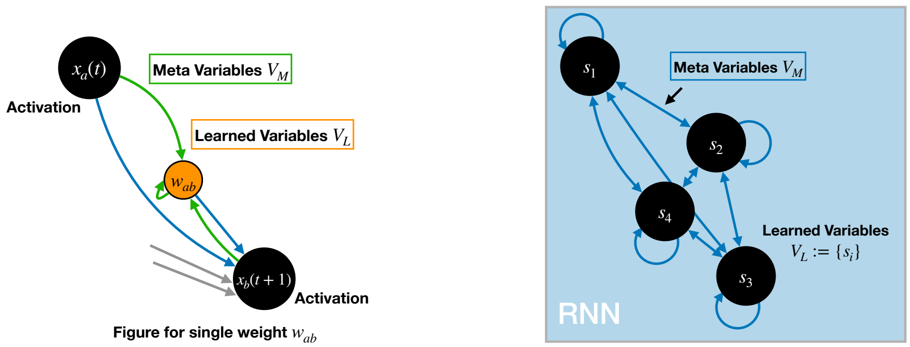
This computation describes $A \cdot B$ independent computation paths ($A \cdot B$ independent RNNs) which we connect using an interaction term

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

where $C \in \mathbb{R}^{N \times N}$. This recursion constitutes the VS Meta RNN and is visualized in Figure 1a. Here we chose the interaction term to mimic the connectivity of weights in a neural network but other interactions could be chosen. Likewise, the duplication along exactly two axes has been selected to facilitate semantic interpretation.

It is trivial that if $A = 1$ and $B = 1$ we yield a single RNN and 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). This corresponds to Figure 1b. For both terms to be equivalent $\tilde{W}$ must satisfy (derivation in Appendix A)

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



(a) FWs / LLRs

(b) FWs / LLRs generalized to an RNN

Figure 2: FWs / LLRs define a weight update rule that augments the standard neural network. (a) Shows the forward computation $x_b(t+1) = \sigma(\sum_a x_a(t) \cdot w_{ab}(t))$ (blue) and weight update rule (green) for a single weight w_{ab} (orange). The same update rule is used for all weights. (b) We can convert the weight to an activation such that the forward computation and update rule can be merged to the recurrent dynamics of an RNN. The size of the RNN can be adapted, here we add one state. If we then duplicate this RNN and connect them we yield the VS Meta RNN from Equation 3.

Emerging weights and learning algorithms in VS Meta RNNs 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 of a neural network which are updated by some mechanism defined by the variables W and C. This is visualized in Figure 1c. The difference between a weight and an activation then merely becomes the frequency with which it is updated and how it is processed by the RNN. Both how the weights are used during forward computation and how they are updated (the learning algorithm) can be integrated into the recurrent dynamics. In the case of backpropagation, this would correspond to the forward and backward passes being implemented purely by unrolling the RNN which we discuss in Section 3.1.

In standard NNs weights and activations have multiplicative interactions. For VS Meta RNNs to mimic such computation these multiplicative interactions would have to exist between parts of the state s, e.g. by modifying Equation 3 to

$$s_{abj} \leftarrow \sigma(\hat{b}_j + \sum_i s_{abi} W_{ij} \;\; \cdot/+ \sum_{c,i} s_{cai} C_{ij}). \tag{5}$$

Fortunately, LSTMs already incorporate such multiplicative interactions using gating in the context vector $\hat{c}$ recurrence

$$\hat{c} \leftarrow \hat{f} \cdot \hat{c} + \hat{i} \cdot \hat{g} \tag{6}$$

where $\hat{f}$, $\hat{i}$, and $\hat{g}$ are a function of the input and hidden state. As we will show in our experiments, LSTMs can implement such multiplicative computation that mimics neural networks also in practice.

Interpreting VS Meta 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

$$\theta \leftarrow f_\phi(\theta, h, D, R) \tag{7}$$

parameterized by ϕ computing the updates on θ given inputs D and (multidimensional) feedback R[2]. In VS Meta RNNs θ and h can be represented by subsets of s, whereas ϕ is given by W and C, yielding a general RNN update

$$s \leftarrow f_{(W,C)}(s, D, R) \tag{8}$$

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

[2] In the supervised setting R would correspond to the negative error on the network 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, actions a and rewards r.

However, a similar argument would also apply to the regular Meta RNN. The variable sharing in our method tightens the relationship to FWs / LLRs. It makes VS Meta RNNs well suited to implement coordinate-wise applied rules such as local learning rules [Bengio et al., 1992, Gregor, 2020, Randazzo et al., 2020] and Hebbian-like differentiable mechanisms [Schmidhuber, 1993, Miconi et al., 2018, 2019]. This is done by merging the learned update rule into the dynamics of an RNN. This is visualized in Figure 2. Let us formally define a parameterized local update equation, the *learning rule*, for the weights w of a neural network,

$$x_b(t+1) = \sigma(\sum_a x_a(t) \cdot w_{ab}(t)) \qquad \text{The neural forward computation} \qquad (9)$$

$$\Delta w_{ab}(t+1) = f_\theta(x_a(t), x_b(t+1), w_{ab}(t), R(t)) \qquad \text{The learning rule} \qquad (10)$$

$$w_{ab}(t+1) = w_{ab}(t) + \Delta w_{ab}(t+1) \qquad (11)$$

where f_θ is a neural function approximator with parameters θ. If we map $w_{ab} := s_{ab0}$, $\theta := (W, C)$, $x_a(t) := \sum_{c,i} s_{cai}(t-1)C_{i0}$, and $x_b(t+1) := \sum_{c,i} s_{cbi}(t)C_{i0}$ then we can rewrite the learning rule to

$$\Delta s_{ab0}(t+l) = f_{(W,C)}(\sum_{c,i} s_{cai}(t-1)C_{i0}, \sum_{c,i} s_{cbi}(t)C_{i0}, s_{ab0}(t), R(t)). \qquad (12)$$

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 VS Meta RNN can implement any such learning rule f.

3.1 Learning to implement backpropagation in recurrent dynamics

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 investigate how VS Meta RNNs can learn to implement backpropagation purely in its recurrent dynamics. We do this by optimizing $V_M := \{\hat{b}, W, C\}$ to (1) store a weight w and bias b as a subset of each state s_{ab}, (2) compute $y = tanh(x)w + b$ to implement neural forward computation, and (3) update w and b according to the backpropagation algorithm. We call this process *learning algorithm cloning*. The optimization is performed on random normally distributed input data. This is visualized in Figure 3.

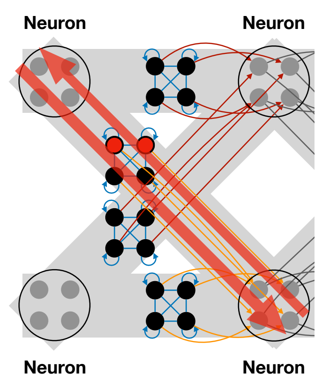


Figure 3: To implement backpropagation we optimize the VS Meta RNNs to use and update weights w and biases b (red dots) as part of the state s_{ab} in each RNN. An additional backwards interaction term (not displayed here) can be added to make learning backpropagation more straightforward.

Optimizing for neural forward computation We optimize any sub-LSTM in VS Meta RNN to compute $y = \tanh(x)w$ for scalars x, y, w by minimizing

$$\mathcal{L}(W, V) := (\tanh(h_1(t))h_0(t) - \sum_i h_i(t+1)C_{i1})^2 \qquad (13)$$

using gradient descent 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 C 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 each input dimension $\bar{i}$ as $x_{\bar{i}} = s_{\bar{i}b1}$ replicated across the second axis B. Equation 13 is trivially extended to a bias. Note that h is bounded between $(-1, 1)$ which poses additional challenges we discuss in Appendix B. We unroll the LSTM for a single time step to minimize $\mathcal{L}$.

Optimizing for backpropagation The activations from the forward pass necessary for credit assignment could be memorized as part of the state s or alternatively be explicitly stored and fed back. 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

$$s_{abj} \leftarrow \sigma(\hat{b}_j + \sum_i s_{abi}W_{ij} + \underbrace{\sum_{c,i} s_{cai}C_{ij}}_{\text{forward interactions}} + \underbrace{\sum_{c,i} s_{bci}U_{ij}}_{\text{backward interactions}}). \qquad (14)$$

This corresponds to connections in the opposite direction compared to Figure 1c. This interaction term allows us to optimize for w and b being updated according to $e \cdot \nabla_{w,b}[y]$ where e is the error that is passed backwards. This error is also optimized for such that $e' = e \cdot \nabla_x[y]$ and multiple iterations in the VS Meta RNN would correspond to the backward pass in backpropagation. To mimic batched optimization we run multiple VS Meta RNNs in parallel and average their states.

3.2 Meta Learning general learning algorithms from scratch

An alternative to *learning algorithm cloning* is end-to-end meta-learning. In this setting, we simply optimize $V_M := \{\hat{b}, W, C\}$ to minimize the sum of losses over many time steps starting with a random state $V_L := s$. This meta-training is done using gradient descent. Crucially, during meta-testing no explicit gradient descent is used. Each time step a single new data point (e.g. one image) is fed into the VS Meta RNN distributed across all sub-RNNs. In the following time step, the prediction error is fed as an input. Thus, we meta-learn a supervised learning algorithm entirely from scratch.

4 Experiments

In this section, we demonstrate the capabilities of VS Meta RNNs by showing that it can implement neural forward computation and backpropagation in their recurrent dynamics on the MNIST and FashionMNIST dataset. Further, we show how we can meta-learn a general learning algorithm from scratch on MNIST that can be successfully applied to Fashion MNIST. Such generalization is enabled by extensive variable sharing where we have very few meta variables $|V_M| \approx 2,800$ and many learned variables $|V_L| \approx 251,000$. Hyperparameters and training details can be found in the appendix.

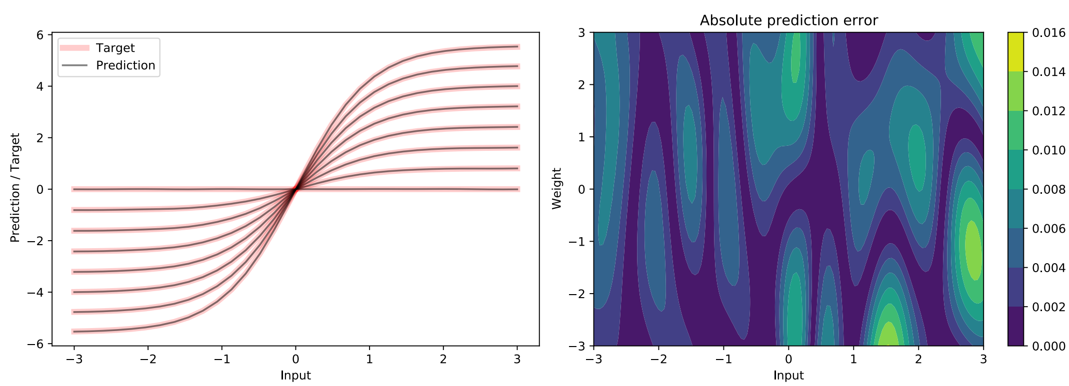


Figure 4: We are optimizing VS Meta RNNs to implement neural forward computation such that for different inputs and weights a tanh-activated multiplicative interaction is produced (left), with different lines for different w. These neural dynamics are not exactly matched everywhere (right), but the error is relatively small.

VS Meta RNNs can implement neural forward computation In this experiment, we optimize the VS Meta RNN to compute $y = tanh(x)w$ on random data using Equation 13. Figure 4 (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 14 to obtain a 'neural network' within a computational network.

VS Meta RNNs can implement backpropagation Similarly, as described in Section 3.1, we optimize the VS Meta RNN to implement backpropagation on random data. We now run the cloned learning algorithm on the MNIST and Fashion MNIST dataset and observe that it performs learning purely in its forward dynamics, making any explicit gradient calculations unnecessary. Figure 5 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

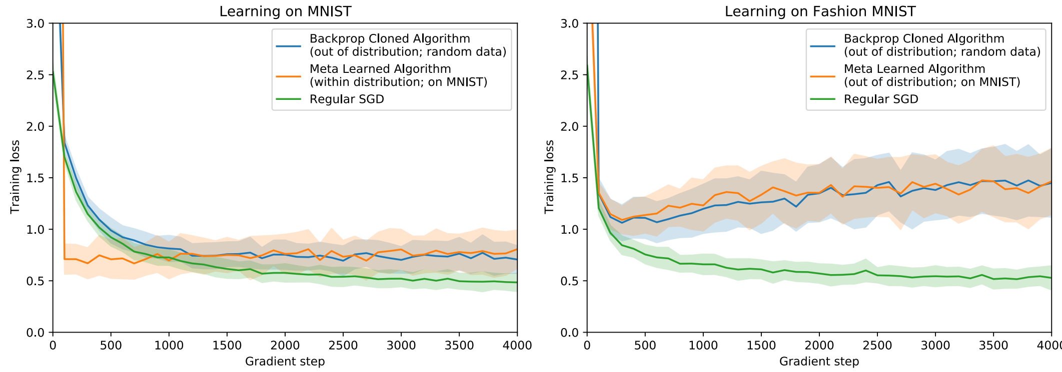


Figure 5: The VS Meta RNN is optimized to implement backpropagation in its recurrent dynamics on random data, then tested (blue) both on MNIST and Fashion MNIST where it performs learning purely by unrolling the LSTM. We then meta-learn to minimize the loss on MNIST starting from the cloned backpropagation 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.

minimized, albeit regular gradient descent 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 as we have observed instabilities with multiple iterations of the meta-trained VS Meta RNN. We hope to obtain stability for deep networks in the near future.

Meta-Learning to improve human-engineered learning algorithms As a second step, we use the cloned backpropagation algorithm encoded in V_M 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 $|V_M|$ and large $|V_L|$. In Figure 5 we observe that meta learning on MNIST and meta testing on MNIST 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 by improving upon a cloned learning algorithm.

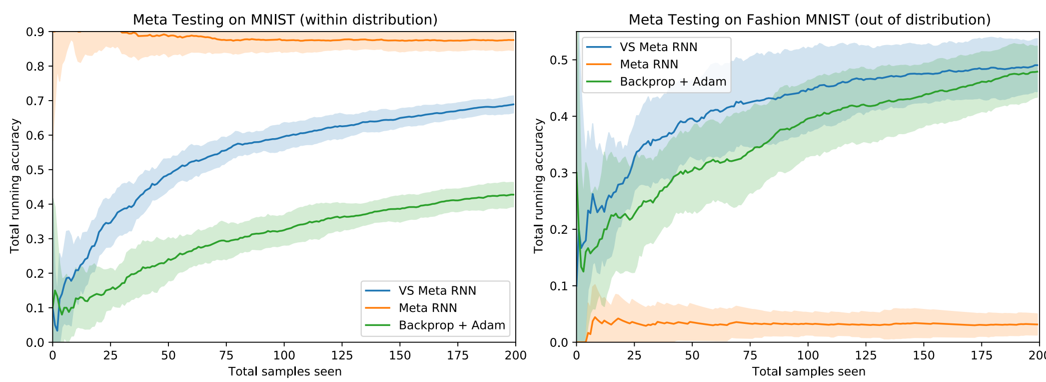


Figure 6: The VS Meta RNN with parameters V_M can be directly meta trained on MNIST to minimize the sum of errors when classifying online data starting from a random state initialization V_L. This allows for much faster learning during meta-testing compared to backpropagation with Adam on the same dataset and even generalizes to a different dataset (Fashion MNIST). Variable sharing in our approach allows for this generalization compared to a standard Meta RNN [Hochreiter et al., 2001] which strongly overfits in the same setting. Standard deviations are over 10 seeds.

Meta Learning supervised learning from scratch Alternatively, a learning algorithm can be meta-learned entirely from scratch as described in Section 3.2. In this setup learning is online, a single input image is fed per step. We do not pre-train V_M with a human-engineered learning algorithm. First, the Variable Shared Meta RNN is meta-trained on MNIST to minimize the sum of cross-entropies over 200 data points starting from random initializations in $V_L := s$. During meta-testing on MNIST (Figure 6) we plot the total accuracy computed on all previous inputs on the y axis, thus starting with low values at the beginning of learning but then rising quickly. We observe that learning is considerably faster compared to the baseline of online backpropagation with the Adam optimizer. One possibility is that the VS Meta RNN simply overfit to the training distribution. We reject this possibility by meta-testing the same unmodified RNN on a different dataset, here Fashion MNIST. Learning still performs fairly well, meaning we have meta-learned a fairly general learning algorithm. If we compare this to a standard Meta RNN we see that in contrast to our method this baseline overfits.

5 Conclusion

We have introduced Variable Shared Meta Learning, a simple principle of variable sharing and sparsity to express powerful learning algorithms. Our Variable Shared Meta RNNs generalize meta recurrent neural networks, fast weight generators (also used in hyper networks), and learned learning rules, by producing emergent neural forward computation 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 neural forward computation in the form of multiplicative interactions and activation functions and (2) pre-train the net on random data to compute correct gradients and the chain rule.

Finally, we have shown that using Variable Shared Meta Learning we can meta-learn supervised learning algorithms from scratch that do not explicitly rely on gradient computation and that display generalization to other datasets. Future experiments will focus on deeper models that learn with cloned backpropagation, reinforcement learning settings, and improvements of meta learning.

Acknowledgements

We thank the anonymous reviewers for their feedback. This work was supported by the ERC Advanced Grant (no: 742870) and computational resources by the Swiss National Supercomputing Centre (CSCS, projects s978 and s1041). We also thank NVIDIA Corporation for donating a DGX-1 as part of the Pioneers of AI Research Award, IBM for donating a Minsky machine, and weights & biases [Biewald, 2020] for their great experiment tracking software and support.

References

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

M. Andrychowicz, M. Denil, S. G. Colmenarejo, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas. Learning to learn by gradient descent by gradient descent. In *Advances in Neural Information Processing Systems*, pages 3988–3996, 6 2016.

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.

L. Biewald. Experiment Tracking with Weights and Biases, 2020. URL `https://www.wandb.com/`.

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.

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1724–1734, 6 2014.

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.

W. Huang, I. Mordatch, and D. Pathak. One Policy to Control Them All: Shared Modular Policies for Agent-Agnostic Control. In *International Conference on Machine Learning*, 2020.

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

K. Li and J. Malik. Learning to Optimize. *arXiv preprint arXiv:1606.01885*, 2016.

K. Li and J. Malik. Learning to Optimize Neural Nets. *arXiv preprint arXiv:1703.00441*, 2017.

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.

D. Pathak, C. Lu, T. Darrell, P. Isola, and A. A. Efros. Learning to Control Self-Assembling Morphologies: A Study of Generalization via Modularity. 2019.

E. Randazzo, E. Niklasson, and A. Mordvintsev. MPLP: Learning a Message Passing Learning Protocol. *arXiv preprint arXiv:2007.00970*, 2020.

S. Ravi and H. Larochelle. Optimization as a model for few-shot learning. In *International Conference on Learning Representations*, 2016.

E. Real, C. Liang, D. R. So, and Q. V. Le. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. 2020.

M. Rosa, O. Afanasjeva, S. Andersson, J. Davidson, N. Guttenberg, P. Hlubuček, M. Poliak, J. Vítku, and J. Feyereisl. BADGER: Learning to (Learn [Learning Algorithms] through Multi-Agent Communication). 2019.

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.

J. Schmidhuber. Discovering Problem Solutions with Low {Kolmogorov} Complexity and High Generalization Capability. Technical Report FKI-194-94, Fakultät für Informatik, Technische Universität München, 1994.

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.