Meta Learning Backpropagation And Improving It

Anonymous Author(s)
Affiliation
Address
email

Abstract

Many concepts have been proposed for meta learning with neural networks (NNs), e.g., NNs that learn to control fast weights, hyper networks, learned learning rules, and meta recurrent NNs. Our *Variable Shared Meta Learning (VSML)* unifies the above and demonstrates that simple weight-sharing and sparsity in an NN is sufficient to express powerful learning algorithms (LAs) in a reusable fashion. A simple implementation of VSML where the weights of a neural network are replaced with tiny LSTMs allows for implementing the backpropagation LA solely by running in forward-mode. It can even meta-learn new LAs alternative to online SGD and generalize to datasets outside of the meta training distribution without explicit gradient calculation. Introspection reveals that our meta-learned LAs learn qualitatively different from gradient descent through fast association. [1]

1 Introduction

The shift from standard machine learning to meta learning involves learning the learning algorithm (LA) itself, reducing the burden on the human designer to craft appropriate learning algorithms [40]. Recent meta learning has primarily focused on learning in similar tasks such as few-shot learning [10] or adaptation to related environments or goals [16]. This is in contrast to human-engineered LAs that generalize across a wide range of datasets or environments. If such generalization is not achieved, meta-learned LAs can not entirely replace human-engineered variants. Recently, it was demonstrated that meta learning can also successfully generate LAs that generalize across wide spectra of environments [18, 1, 29], e.g., from toy environments to Mujoco and Atari. Unfortunately, however, many recent approaches still rely on a large number of human-designed and unmodifiable inner-loop components such as backpropagation.

Is it possible to implement modifiable versions of backpropagation or related algorithms as part of the end-to-end differentiable activation dynamics of a neural net (NN), instead of inserting them as separate fixed routines? Here we propose the Variable Shared Meta Learning (VSML) principle for this purpose. It introduces a novel way of using sparsity and weight-sharing in NNs for meta learning. We build on the arguably simplest neural meta learner, the meta recurrent neural network (Meta RNN) [15, 9, 50] by replacing the weights of a neural network with tiny LSTMs. This can be viewed as RNNs passing messages, one big RNN, or learning the function each neuron performs and its LA. VSML generalizes certain end-to-end differentiable fast weights [42, 43, 3, 38], hyper networks [13]), learned learning rules [4, 12, 31], or hebbian-like synaptic plasticity [41, 43, 24, 25, 28]. Our mechanism, VSML, can implement backpropagation solely in the forward-dynamics of an RNN. Consequently, it enables meta-optimization of backprop-like algorithms. We envision a future where novel methods of credit assignment can be meta-learned while still generalizing across vastly different tasks. This may lead to improvements in sample efficiency, memory efficiency, continual learning, and others. As a first step, our system meta-learns LAs from scratch that frequently learn faster than

[1] Anonymized code can be found in the supplementaries

Submitted to 35th Conference on Neural Information Processing Systems (NeurIPS 2021). Do not distribute.

online gradient descent and generalize to datasets outside of the meta training distribution. Our VSML RNN is the first neural meta learner without hard-coded backpropagation that shows such strong generalization. Our results may also be of interest with regard to biologically plausible alternatives to backpropagation.

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 an NN to implement a learning rule (FWs / LLRs) and 2) Meta learning implemented in recurrent neural networks (Meta RNNs).

Fast weights & Learned learning rules (FWs / LLRs) In a standard NN the weights (and biases) are updated by a fixed LA. This framework can be extended to meta learning by defining an explicit architecture that allows for modifying these weights. This weight-update architecture augments a standard NN architecture. NNs that generate or modify the weights of another or the same NN are known as fast weight NNs [41, 42, 43, 3, 38], hypernetworks [13], NNs with synaptic plasticity [24, 25, 28] or learned learning rules [4, 12, 31]. 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 LA are defined by the weight-update architecture. Note that the dimensionality of V_L and V_M can be defined independently of each other and often V_M are reused in coordinate-wise fashion for V_L resulting in $|V_L| \gg |V_M|$ where $|\cdot|$ is the vector size.

Meta Learning in activations (Meta RNNs) It was shown that an RNN such as LSTM can learn to implement a LA [15] when the reward or error is given as an input [44]. After meta training, the LA is encoded in the weights of this RNN and determines learning during meta testing. The activations serve as the memory used for the LA solution. We refer to this as Meta RNNs [15, 9, 50]. [2] 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 LA is largely overparameterized whereas the available memory for learning is very small, making this approach prone to overfitting [18]. As a result, the RNN parameters often encode task-specific solutions instead of generic LAs. In order to meta-learn a simple and generalizing LA 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 [47, 27, 37, 26, 39].

3 Variable Shared Meta Learning (VSML)

In VSML 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 an NN (variable sharing) and introducing sparsity in the connectivity. Activations V_L in this NN then can be interpreted as the weights of a conventional NN, consequently blurring the distinction between a weight and an activation. [3]

A simple conceptual way of understanding VSML is taking a standard recurrent or feed-forward architecture and replacing all weights with small RNNs, as visualized in Figure 1. All these RNNs share the same parameters but have distinct states. Similar to how weights in a neural network are connected, these RNNs pass messages to each other, both in forward direction $\overrightarrow{m}$ and backward $\overleftarrow{m}$. This can be seen as a generalization of learned learning rules [4, 12, 31] and Hebbian-like differentiable mechanisms [41, 43, 24, 25] (more on this relationship in Appendix D).

The role of weights in a standard neural network is now assigned to the states of RNNs. This allows these RNNs to define both the neural forward computation as well as the learning algorithm that

[2] Also sometimes referred to as memory based meta learning.

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

determines how the network is updated. In the case of backpropagation, this would correspond to the forward and backward passes being implemented purely in the recurrent dynamics.

3.1 From Meta RNNs to VSML RNNs

We begin by formalizing Meta RNNs which often use multiplicative gates such as the LSTM [11, 14] or its variant GRU [6]. For notational simplicity, we consider a vanilla RNN. Let $s \in \mathbb{R}^N$ be the hidden state of an RNN. The update for an element $j \in \{1, \ldots, N\}$ is given by

$$s_j \leftarrow \sigma(\sum_i s_i W_{ij}) \quad (1)$$

where σ is a non-linear activation function, $W \in \mathbb{R}^{N \times N}$, and the bias is omitted for simplicity. We also omit inputs by assuming a subset of s to be externally provided. Each application of this iteration reflects a single time tick in the RNN.

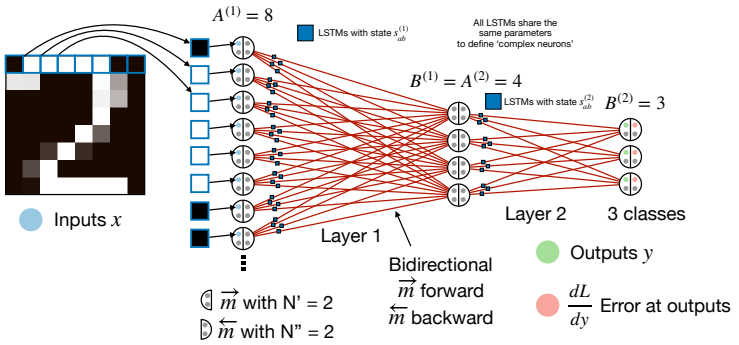

Figure 1: In VSML we replace all weights of a standard NN with tiny LSTMs using shared parameters (resembling complex neurons). This allows these LSTMs to define both the neural forward computation as well as the learning algorithm that determines how the network is updated. Information flows forward and backward in the network through multi-dimensional messages $\overrightarrow{m}$ and $\overleftarrow{m}$, generalizing the dynamics of a NN trained using backpropagation.

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

$$s_{abj} \leftarrow \sigma(\sum_i s_{abi} W_{ij}). \quad (2)$$

Here we follow the semantic interpretation of RNNs representing weights in a neural network and thus A corresponds to the dimensionality of the inputs and B of the outputs in a layer (see Figure 1). For this formal construction we describe a single layer with $A = B$ that is used recurrently. The computation so far describes $A \cdot B$ independent computation paths ($A \cdot B$ independent RNNs) which we connect using an interaction term

$$s_{abj} \leftarrow \sigma(\sum_i s_{abi} W_{ij} + \underbrace{\sum_{c,i} s_{cai} C_{ij}}_{\text{interactions}}) \quad (3)$$

where $C \in \mathbb{R}^{N \times N}$. This recursion constitutes the basic VSML RNN with $V_M := (W, C)$. The interaction term mimics the (forward) connectivity of weights in a NN.

We can also write Equation 3 as multiple RNNs passing messages $\overrightarrow{m} : \mathbb{R}^N \to \mathbb{R}^{N'}$ to each other to determine the next state $s_{ab} \leftarrow f_{\text{RNN}}(s_{ab}, \sum_c \overrightarrow{m}(s_{ca}))$. This is related to Graph Neural Networks [52]. Messages are essentially inputs and outputs of RNNs, thus the RNN state size N can be different from the message size N'. In this paper we investigate a specific implementation of VSML corresponding to replacing weights with RNNs. We highlight that other schemes based on different kinds of message passing and graph connectivity are possible.

In standard NNs weights and activations have multiplicative interactions. For VSML RNNs to mimic such computation we require multiplicative interactions between parts of the state s. Fortunately, LSTMs already incorporate this using gating. and Equation 3 becomes

$$z_{ab}, h_{ab} \leftarrow f_{\text{LSTM}}(z_{ab}, h_{ab}, \sum_c \overrightarrow{m}(h_{ca})) \quad (4)$$

where z_{ab} is the context vector and h_{ab} is the hidden state.

3.2 Connection to Meta RNNs

It is trivial to see that with $A = 1$ and $B = 1$ we obtain a single RNN and Equation 3 recovers the original Meta RNN Equation 1. In the general case, we can derive an equivalent formulation that corresponds to a single standard RNN 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)

$$s_{abj} \leftarrow \sigma(\sum_{c,d,i} s_{cdi} \tilde{W}_{cdiabj}). \tag{5}$$

For Equation 3 and Equation 5 to be equivalent, $\tilde{W}$ must satisfy (derivation in Appendix A)

$$\tilde{W}_{cdiabj} = \begin{cases} C_{ij}, & \text{if } d = a \wedge (d \neq b \vee c \neq a). \\ W_{ij}, & \text{if } d \neq a \wedge d = b \wedge c = a. \\ C_{ij} + W_{ij}, & \text{if } d = a \wedge d = b \wedge c = a. \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

To distinguish between the single sparse-shared RNN and the small RNNs representing weights, we now call the latter sub-RNNs.

3.3 Stacking VSML RNNs and feeding inputs

In order to get a structure similar to the one of non-recurrent deep feed-forward architectures (FNNs), we stack multiple VSML RNNs where their states are untied and their parameters are tied. [4] This is visualized with two layers in Figure 1 where the states $s^{(2)}$ of the second column of sub-RNNs are distinct from the first column $s^{(1)}$. The parameters $A^{(k)}$ and $B^{(k)}$ describing layer sizes can then be varied for each layer $k \in \{1, \ldots, K\}$ constrained by $A^{(k)} = B^{(k-1)}$. Our updated Equation 3 with distinct layers k is defined as $s^{(k)}_{abj} \leftarrow \sigma(\sum_i s^{(k)}_{abi} W_{ij} + \sum_{c,i} s^{(k-1)}_{cai} C_{ij})$. In this case information would only flow into one direction which we can compensate for using an additional backward term

$$s^{(k)}_{abj} \leftarrow \sigma(\sum_i s^{(k)}_{abi} W_{ij} + \sum_{c,i} s^{(k-1)}_{cai} C_{ij} + \sum_{c,i} s^{(k+1)}_{bci} U_{ij}). \tag{7}$$

In the message passing view this is $s^{(k)}_{ab} \leftarrow f(s^{(k)}_{ab}, \sum_c \overrightarrow{m}(s^{(k-1)}_{ca}), \sum_c \overleftarrow{m}(s^{(k+1)}_{bc}))$ with a forward message function $\overrightarrow{m} : \mathbb{R}^N \to \mathbb{R}^{N'}$ and a backward message function $\overleftarrow{m} : \mathbb{R}^N \to \mathbb{R}^{N''}$. We denote the resulting summed incoming messages to sub-RNN ab at layer k by $\overrightarrow{\boldsymbol{m}}^{(k)} \in \mathbb{R}^{A(k) \times N'}$ where $\overrightarrow{\boldsymbol{m}}^{(k)}_a := \sum_c \overrightarrow{m}(s^{(k-1)}_{ca})$ and by $\overleftarrow{\boldsymbol{m}}^{(k)} \in \mathbb{R}^{B(k) \times N''}$ where $\overleftarrow{\boldsymbol{m}}^{(k)}_b := \sum_c \overleftarrow{m}(s^{(k+1)}_{bc})$. Thus, our state update equation can also be written as

$$s^{(k)}_{ab} \leftarrow f_{V_M}(s^{(k)}_{ab}, \overrightarrow{\boldsymbol{m}}^{(k)}_a, \overleftarrow{\boldsymbol{m}}^{(k)}_b). \tag{8}$$

Messages $\overrightarrow{\boldsymbol{m}}^{(1)}$ and $\overleftarrow{\boldsymbol{m}}^{(K)}$ are externally specified inputs.

In a similar fashion, other neural architectures can be explicitly enforced. Note that VSML in principle allows for learning some architectures implicitly if positional information is fed into each sub-RNN (Appendix C). We then update all states $s^{(k)}$ in sequence $1, \ldots, K$ to mimic sequential layer execution. Note that we can also apply multiple ticks for each layer k before moving to the next.

To provide the VSML RNN with data, each time we execute the first layer, a single new datum $x \in \mathbb{R}^{A(1)}$ (e.g. one flattened image) is distributed across all sub-RNNs. In our present experiments we match the axis $A(1)$ to the input datum dimensionality. Let $k = 1$ be the first layer, then we explicitly set the forward message $\overrightarrow{\boldsymbol{m}}^{(1)}_{a0} := x_a$ (padding $\overrightarrow{\boldsymbol{m}}$ with zeros if necessary). Similarly, we read the output $y \in \mathbb{R}^{B(K)}$ from $y_a := \overrightarrow{\boldsymbol{m}}^{(K+1)}_{a0}$. Finally, we feed the error $e \in \mathbb{R}^{B(K)}$ at the output such that $\overleftarrow{\boldsymbol{m}}^{(K)}_{b0} = e_b$. See Figure 1 for a visualization. Alternatively, multiple input or output dimensions could be patched together and fed into fewer sub-RNNs.

[4] The resultant architecture as a whole is still recurrent. Note that even standard FNNs are recurrent if the LA (backpropagation) is taken into account.

3.4 Meta learning general learning algorithms from scratch

Having formalized the VSML RNN we can now use end-to-end meta learning to create LAs from scratch. In this setting, we simply optimize $V_M := \{W, C\}$ to minimize the sum of prediction losses over many time steps starting with a random state $V_L := s$. Meta training may be performed using end-to-end gradient descent or gradient-free optimization such as evolutionary strategies [51, 35]. Crucially, during meta testing no explicit gradient descent is used. Layers (if used) are always executed in sequence $1, \ldots, K$. When the first layer is executed, a single new data point (e.g. one image) is fed into the VSML RNN distributed across all sub-RNNs. At the last layer, the prediction is read and the error is fed as an input.

3.5 Learning to implement backpropagation in RNNs

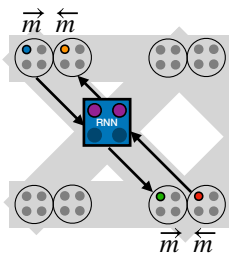

Figure 2: To implement backpropagation we optimize the VSML RNN to use and update weights w and biases b as part of the state s_{ab} in each sub-RNN. Inputs are pre-synaptic x and error e. Outputs are post-synaptic $\hat{y}$ and error $\hat{e}'$.

An alternative to end-to-end meta learning is to first investigate whether the VSML RNN can implement backpropagation. Due to the algorithm's ubiquitous use, it seems desirable to be able to meta-learn backpropagation-like algorithms. Here we investigate how VSML RNNs can learn to implement backpropagation purely in their recurrent dynamics. We do this by optimizing $V_M := \{W, C\}$ 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 [22]. We call this process *learning algorithm cloning* and it is visualized in Figure 2.

To optimize each sub-RNN to compute $y = \tanh(x)w + b$ or a related activation function, we designate an element of the summed input message $\overrightarrow{\boldsymbol{m}}_a^{(k)}$ to correspond to x, and an element of the output message $\overrightarrow{m}(s_{ab}^{(k)})$ to the predicted $\hat{y}$. Similarly, we set $w := s_{ab0}$ and $b := s_{ab1}$. We then use gradient descent with respect to V_M to regress $\|y - \hat{y}\|$ on a dataset of (x, w, b) tuples.

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. Here, we chose the latter. Having two interaction terms (Equation 7) allows for optimizing V_M such that $w := s_{ab0}$ and $b := s_{ab1}$ are updated according to $e \cdot \nabla_{w,b}[y]$ where e is the error that is passed backwards according to the backpropagation algorithm. An element of the summed input message $\overleftarrow{\boldsymbol{m}}_b^{(k)}$ is thus designated as the input error e. At the same time, we produce another output error $\hat{e}'$ that is an element of the output message $\overleftarrow{m}(s_{ab}^{(k)})$. This error is optimized to produce $e' = e \cdot \nabla_x[y]$. Multiple iterations in the VSML RNN would then correspond to the backward pass in backpropagation. We continuously run the VSML RNN forward, alternately running the layers in order $1, \ldots, K$ and in reverse order $K, \ldots, 1$.[5] In summary, we optimize V_M to regress $\hat{y}$, Δw, Δb, and $\hat{e}$ toward their respective targets. We can either generate the training dataset $D := \{(x, w, b, e)_i\}$ randomly and unroll the VSML RNN only for a single step or alternatively run a 'shadow' NN on some supervised problem and fit the VSML RNN to its activations and parameter updates. To mimic batched optimization we run multiple VSML RNNs in parallel and average their states.

4 Experiments

Now we demonstrate the capabilities of the VSML RNN by showing that it can implement neural forward computation and backpropagation in its recurrent dynamics on the *MNIST* [19] and *Fashion MNIST* [53] dataset. Furthermore, we show how we can meta-learn a LA from scratch on one set of datasets and then successfully apply it to another (out of distribution). Such generalization is enabled by extensive variable sharing where we have very few meta variables $|V_M| \approx 2,700$ and

[5] Executing layers in reverse order is not strictly necessary as information always also flows backward through $\overleftarrow{m}$ but makes LA cloning easier. We train the VSML RNN to perform backpropagation during reverse order execution.

many learned variables $|V_L| \approx 125,000$. Finally, we introspect the meta-learned LA and compare it to gradient descent.

Our implementation uses LSTMs and the message interpretation from Equation 8. Hyperparameters and training details can be found in the appendix.

4.1 VSML RNNs can implement backpropagation

As described in Section 3.5, we optimize the VSML RNN to implement backpropagation. We compose the VSML RNNs to mimic a feedforward NN with either one hidden layer or no hidden layers. To obtain training targets, we instantiate a standard NN, the shadow network, and feed it MNIST data. After cloning, we then run the LA on the MNIST and Fashion MNIST dataset and observe that it performs learning purely in its recurrent dynamics, making explicit gradient calculations unnecessary. Figure 3 shows the learning curve on these two datasets achieving around 87% (deep) and 90% (shallow) test accuracy on MNIST and 76% (deep) and 80% (shallow) on Fashion MNIST. Notably, learning works both on MNIST (within distribution) and on Fashion MNIST (out of distribution). We observe that the loss is decently minimized, albeit regular gradient descent still outperforms our cloned backpropagation. This may be due to non-zero errors during learning algorithm cloning, in particular when these errors accumulate in the deeper architecture. It is also possible that activations and parameters deviate too far from ranges seen during cloning, in particular in the deep case when the loss starts increasing.

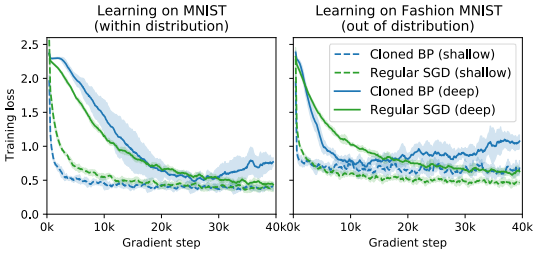

Figure 3: The VSML RNN is optimized to implement backpropagation in its recurrent dynamics on MNIST, then tested both on MNIST and Fashion MNIST where it performs learning purely by unrolling the LSTM. We test on shallow and deep architectures (single hidden layer of 32 units). Standard deviations are over 6 seeds.

4.2 Meta learning supervised learning from scratch

In the previous experiments we have established that VSML is expressive enough to meta-optimize backpropagation-like algorithms. Instead of cloning a LA, we now meta-learn from scratch as described in Section 3.4. This is an opportunity to drop mini-batching and return to an online setting. A single input image is fed to the VSML RNN at a time which then outputs logits corresponding to the label prediction. With the next input we also feed the error of the previous prediction. Here, we use a single layer from input to output dimension and run it for two ticks per image with $N = 16$ and $N' = 8$. First, the VSML RNN is meta-trained end-to-end using evolutionary strategies (ES) [35] on MNIST to minimize the sum of cross-entropies over 500 data points starting from random state initializations. As each image is unique and V_M can not memorize the data, we are implicitly optimizing the VSML RNN to generalize to future inputs given all inputs it has seen so far. We do not pre-train V_M with a human-engineered LA.

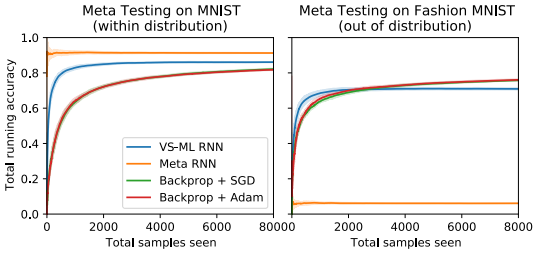

Figure 4: The VSML RNN with parameters V_M can be directly meta trained on MNIST to minimize the sum of errors when classifying online starting from a random state initialization. This allows for faster learning during meta testing compared to online gradient descent with Adam on the same dataset and even generalizes to a different dataset (Fashion MNIST). In comparison, a standard Meta RNN [15] strongly overfits in the same setting. Standard deviations are over 10 seeds.

During meta testing on MNIST (Figure 4) we plot the total running accuracy on all previous inputs on the y axis ($\frac{\sum_{t=1}^{T} c_t}{T}$ after example T with binary c_t indicating prediction correctness). For each example the prediction when this example was fed is used, thus measuring sample efficient learning. We observe that learning is considerably faster compared to the baseline of online gradient descent (no mini batching, learning rate tuned). One possibility is that VSML simply overfits to the training

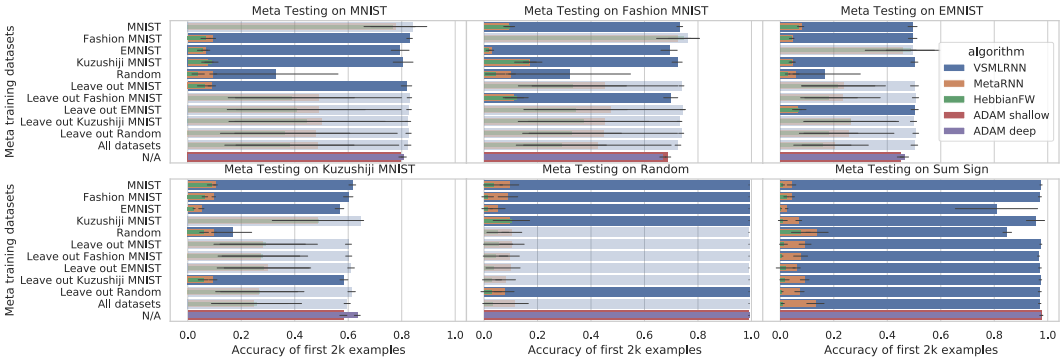

Figure 6: Online learning on various datasets. Total accuracy in % after having seen 2k training examples evaluated after each prediction starting with random states (VSML, Meta RNN, HebbianFW) or random parameters (SGD). Standard deviations are over 32 meta test training runs. Meta testing is done on the official test set of each dataset. Meta training is on subsets of datasets excluding the Sum Sign dataset. Unseen tasks, most relevant from a general-purpose LA perspective, are opaque.

distribution. We reject this possibility by meta testing the same unmodified RNN on a different dataset, here Fashion MNIST. Learning still works well, meaning we have meta-learned a fairly general LA (although performance at convergence still lacks behind a little). This generalization is achieved without using any hardcoded gradients during meta testing purely by running the LSTM forward. A Meta RNN in comparison to our method heavily overfits.

4.3 Robustness to varying inputs and outputs

A defining property of VSML is that the same parameters V_M can be used to learn on varying input and output sizes. Further, the architecture and thus the meta-learned LA is invariant to the order of inputs and outputs. In this experiment, we investigate how robust we are to such changes. We meta-train across MNIST with 3, 4, 6, and 7 classes. Likewise we train across rescaled versions with 14x14, 28x28, and 32x32 pixels. We also randomly project all inputs using a linear transformation, with the transformation fixed for all inner learning steps.

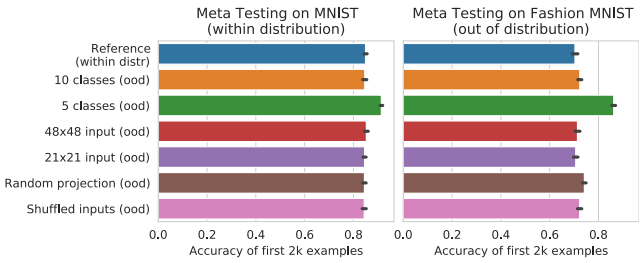

Figure 5: The meta-learned learning algorithm is robust to permutations and size changes in the inputs and outputs. All six configurations have not been seen during training and perform comparable to the unchanged reference. Standard deviations are over 32 seeds.

In Figure 5 we meta-test on 6 configurations that were not seen during meta-training. Performance on all of these configurations is comparable to the unchanged reference from the previous section. In particular the invariance to random projections suggests that we have meta-learned a learning algorithm beyond transferring learned representations [compare 10, 48, 49]

4.4 Varying datasets

To better understand how different meta training distributions and meta test datasets affect VSML RNNs and our baselines, we present several different combinations in Figure 6. The opaque bars represent tasks that were not seen during meta training, and are thus most relevant for this analysis. This includes four additional datasets: (1) *Kuzushiji MNIST* [7] with 10 classes, (2) *EMNIST* [8] with 62 classes, (3) A randomly generated classification dataset (*Random*) with 20 data points that changes with each step in the outer loop, and (4) *Sum Sign* which generates random inputs and requires classifying the sign of the sum of all inputs. Meta training is done over 500 randomly drawn samples per outer iteration. Each algorithm is meta-trained for 10k outer iterations. Inputs are randomly projected as in Section 4.3. We again report the accuracy on all data seen since the beginning of meta test training. We compare to SGD with a single layer, matching the architecture of VSML, and a

hidden layer, matching the number of weights to the size of V_L in VSML. We also have included a Hebbian fast weight baseline [24].

We observe that VSML RNNs generalize much better than Meta RNNs and Hebbian fast weights. Notably, we even generalize to the unseen task Random and Sum Sign which has no shared structure with the other datasets. The baselines don't perform as well even on seen datasets compared to Figure 4 due to the random projections. Without those, overfitting is increased (Appendix B). In many cases VSML's performance is similar to SGD but a little more sample efficient in the beginning of training (learning curves in Appendix B). This suggests that our meta-learned LAs are good at quickly associating new inputs with their labels. We further investigate this in the next Section 5.

5 Analysis

Given that VSML RNNs seems to learn faster than online gradient descent in many cases we would like to qualitatively investigate how learning differs. We first meta train on the full MNIST dataset as before. During meta testing, we plot the output probabilities against the number of samples seen in Figure 7. We highlight the input class □ as well as the predicted class ◯. In this case, our meta test dataset consists of MNIST digits with two examples of each type. The same digit is always repeated twice. This allows us to observe and visualize the effect with only few examples. We have done the same introspection with the full dataset with similar conclusions (see Appendix B).

We observe that in VSML almost all failed predictions are followed by the correct prediction with high certainty. In contrast, SGD makes many incorrect predictions and fails to adapt correctly in only 20 steps. It seems that SGD learns qualitatively different from VSML. The VSML RNN meta-learns the skill of fast association of new inputs whereas SGD fails to do so. We tried several different SGD learning rates but the phenomenon seems to be independent of this parameter. As evident from high accuracies in Figure 6, VSML does not only memorize inputs using this strategy, but the associations generalize to future unseen inputs.

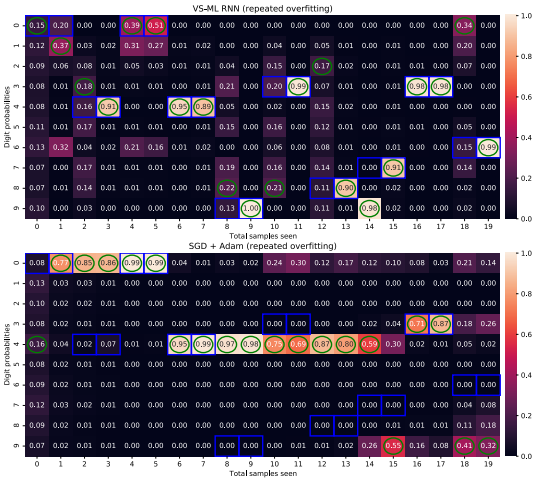

Figure 7: Introspection of how output probabilities change after observing an input and its error at the output units when meta testing on MNIST with two examples for each type. We highlight the input class □ as well as the predicted class ◯. The top plot shows the VSML RNN quickly associating the input images with the right label, almost always making the right prediction the second time with high confidence. The bottom plot shows the same dataset processed by SGD with Adam which fails to learn quickly.

6 Related Work

Memory based meta learning (Meta RNNs) Memory-based meta learning in RNNs [15, 9, 50] is a simple neural meta learner (see Section 2). Unfortunately, the LA encoded in the RNN parameters is largely overparameterized which leads to overfitting. VSML demonstrates that weight sharing can mend this issue leading to more general-purpose LAs.

Learned Learning Rules / Fast Weights NNs that generate or modify the weights of another or the same NN are known as fast weight systems [41], hypernetworks [13], NNs with synaptic plasticity [24] or learned learning rules [4] (see Section 2). In VSML we do not require explicit architectures for weight updates as weights are emergent from RNN state updates. Additionally, we implicitly learn how the neural forward computation is defined. Contemporary to this work, fast weights have also been used to meta-learn more general LAs [36] (our preprint f.a. in Dec 2020).

Learned gradient-based optimizers Meta learning has been used to find optimizers that update the parameters of a model by taking the loss and gradient with respect to these parameters as an

input [32, 2, 20, 21, 23]. In this work, we are interested in meta learning that does not rely on fixed gradient calculation in the inner loop.

Discrete program search An interesting alternative to distributed variable updates in VSML is meta learning via discrete program search [45, 33]. In this paradigm, a separate programming language needs to be defined that gives rise to neural computation when its instructions are combined. This led to the automated rediscovery of backpropagation [33]. Compared to our work, modifications of the program are usually highly non-smooth.

Multi-agent systems In the reinforcement learning setting multiple agents can be modeled with shared parameters [46, 30, 17], also in the context of meta learning [34]. This is related to the variable sharing in VSML depending on how the agent-environment boundary is drawn. Unlike these works, we demonstrate the advantage of variable sharing in meta learning more general-purpose LAs.

7 Discussion and Limitations

The research community has perfected the art of leveraging backpropagation for learning for a long time. At the same time there are unsolved issues such as memory requirements, online learning, continual learning, sample efficient learning, separate backward phases, and others. VSML suggests that instead of building on top of backpropagation as a fixed routine, meta learning offers an alternative to discover general purpose LAs. Nevertheless, this paper is only a proof of concept—until now we have only investigated small-scale problems and performance does not yet quite match the mini-batched setting with large quantities of data. In particular, we observed premature convergence of the solution at meta-test time which prompts future investigations. Scaling this to harder problems and larger meta task distributions is important future work.

The computational cost of the VSML RNN is also larger than the one of standard backpropagation. If we run a sub-RNN for each weight in a standard NN with W weights the cost is in $O(WN^2)$ where N is the state size of a sub-RNN. If N is small enough, and our experiments suggest small N may be feasible, this may be an acceptable cost.

Meta optimization is also prone to local minima. In particular, when the number of ticks between input and feedback increases (e.g. the with deeper architectures), credit assignment becomes harder. Early experiments suggest that diverse meta task distributions can help mitigate these issues. Additionally, learning horizons are limited when using backprop-based meta optimization. Using ES allowed training across longer horizons and more stable optimization.

VSML can also be viewed as regularizing the NN weights that encode the LA through a representational bottleneck. It is possible that LA generalization as found by VSML can also be achieved through other regularization techniques. Unlike most regularizers, VSML also introduces heavy reusal of the same learning principle and permutation invariance through variable sharing.

8 Conclusion

We have introduced Variable Shared Meta Learning, a simple principle of weight sharing and sparsity to meta-learn powerful learning algorithms (LAs). In our implementation we replace the weights of a neural network with tiny LSTMs of shared parameters. We show connections to meta recurrent neural networks, fast weight generators (hyper networks), and learned learning rules.

Using *learning algorithm cloning*, VSML RNNs can learn to implement the backpropagation algorithm and its parameter updates encoded implicitly in the recurrent dynamics. On MNIST it learns to predict well without any human-designed explicit computational graph for gradient calculation.

Finally, we have shown that Variable Shared Meta Learning can meta-learn supervised LAs from scratch that do not explicitly rely on gradient computation and that *generalize to unseen datasets*. Introspection revealed that VSML LAs learn qualitatively different from stochastic gradient descent by fast association. This leads to gains in sample efficiency. Future work will focus on reinforcement learning settings, improvements of meta learning, and larger task distributions.

References

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

[2] 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.

[3] 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.

[4] 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.

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

[6] 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.

[7] T. Clanuwat, M. Bober-Irizar, A. Kitamoto, A. Lamb, K. Yamamoto, and D. Ha. Deep Learning for Classical Japanese Literature, 2018.

[8] G. Cohen, S. Afshar, J. Tapson, and A. V. Schaik. EMNIST: Extending MNIST to handwritten letters. *2017 International Joint Conference on Neural Networks (IJCNN)*, 2017. doi: 10.1109/ijcnn.2017.7966217.

[9] 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.

[10] 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.

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

[12] K. Gregor. Finding online neural update rules by learning to remember. *arXiv preprint arXiv:2003.03124*, 3 2020.

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

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

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

[16] 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.

[17] 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.

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

[19] Y. LeCun, C. Cortes, and C. J. Burges. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

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

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

[22] S. Linnainmaa. *The representation of the cumulative rounding error of an algorithm as a Taylor expansion of the local rounding errors*. PhD thesis, Univ. Helsinki, 1970.

[23] L. Metz, N. Maheswaranathan, C. D. Freeman, B. Poole, and J. Sohl-Dickstein. Tasks, stability, architecture, and compute: Training more effective learned optimizers, and using them to train themselves. *arXiv preprint arXiv:2009.11243*, 9 2020.

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

[25] 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.

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

[27] 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.

[28] E. Najarro and S. Risi. Meta-Learning through Hebbian Plasticity in Random Networks. *Advances in Neural Information Processing Systems*, 2020.

[29] 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.

[30] D. Pathak, C. Lu, T. Darrell, P. Isola, and A. A. Efros. Learning to Control Self-Assembling Morphologies: A Study of Generalization via Modularity. In *Advances in Neural Information Processing Systems*, 2019.

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

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

[33] E. Real, C. Liang, D. R. So, and Q. V. Le. AutoML-Zero: Evolving Machine Learning Algorithms From Scratch. In *International Conference on Machine Learning*, pages 8007–8019, 2020.

[34] 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). *arXiv preprint arXiv:1912.01513*, 2019.

[35] T. Salimans, J. Ho, X. Chen, S. Sidor, and I. Sutskever. Evolution Strategies as a Scalable Alternative to Reinforcement Learning. *arXiv preprint arXiv:1703.03864*, 3 2017. ISSN ISSN 0302-9743. doi: 10.1.1.51.6328.

[36] M. Sandler, M. Vladymyrov, A. Zhmoginov, N. Miller, A. Jackson, T. Madams, and others. Meta-Learning Bidirectional Update Rules. *arXiv preprint arXiv:2104.04657*, 2021.

[37] 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.

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

[39] I. Schlag, T. Munkhdalai, and J. Schmidhuber. Learning Associative Inference Using Fast Weight Memory. In *International Conference on Learning Representations*, 2021.

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

[41] J. Schmidhuber. Learning to Control Fast-Weight Memories: An Alternative to Recurrent Nets. Technical Report FKI-147-91, Institut für Informatik, Technische Universität München, 3 1991.

[42] 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.

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

[44] J. Schmidhuber. A self-referential weight matrix. In *Proceedings of the International Conference on Artificial Neural Networks, Amsterdam*, pages 446–451. Springer, 1993.

[45] 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.

[46] Sims, Karl. *Evolving Virtual Creatures*. ACM SIGGRAPH, 1994.

[47] 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.

[48] E. Triantafillou, T. Zhu, V. Dumoulin, P. Lamblin, U. Evci, K. Xu, R. Goroshin, C. Gelada, K. Swersky, P.-A. Manzagol, and H. Larochelle. Meta-Dataset: A Dataset of Datasets for Learning to Learn from Few Examples. 3 2019.

[49] H.-Y. Tseng, H.-Y. Lee, J.-B. Huang, and M.-H. Yang. Cross-domain few-shot classification via learned feature-wise transformation. *arXiv preprint arXiv:2001.08735*, 2020.

[50] 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.

[51] D. Wierstra, T. Schaul, J. Peters, and J. Schmidhuber. Natural Evolution Strategies. In *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pages 3381–3387, 2008.

[52] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[53] H. Xiao, K. Rasul, and R. Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. *CoRR*, abs/1708.0, 2017.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [Yes] Our paper is focused on general-purpose methodologies for learning and is not directly concerned with an application. Thus, societal impact highly depends on its downstream use. Nevertheless, meta learning the learning algorithm itself instead of using well-known human engineered variants leads to loss of guarantees of their behavior. This may lead to additional challenges long-term with regards to AI safety.
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [Yes]
   (b) Did you include complete proofs of all theoretical results? [Yes]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes] Code can be found in the supplemental material.
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] Also see supplemental material.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [Yes]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] During the course of research we have used an estimated 100'000 node hs of NVIDIA Tesla P100 GPUs.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes] Datasets are publicly accessible and licenses can be found at `https://www.tensorflow.org/datasets/catalog/overview`.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes] Code for additional toy datasets can be found in the supplemental material.
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]