A Derivations

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

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

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

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

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 $(\tilde{W} - A)_{cdiabj} = (d \equiv a) C_{ij}$ we yield

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

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

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

which we rewrite in tabular notation:

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

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

B Alternative neural forward computation

Bounded states in LSTMs In Section 4 we noted that in LSTMs h is bounded between $(-1, 1)$. This means that with standard tanh neural forward computation $\tanh(x)w + b$ both w and 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 13 while scaling down w and b in the LSTM.

LSTM states and LSTM inputs. In Section 4, h_1 denoted the part of the state that corresponds to the 'neuron' input. In practice, we 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 VS Meta 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 forward computation 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 to improve backpropagation

C Other training details

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

Batching for VS Meta RNNs backpropagation experiments In Section 3.1 we optimize a VS Meta 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 analog to batching in standard gradient descent. In our meta learning from scratch experiments, we did not use this technique which allows for a greatly increased sample efficiency during learning.

Stability during meta-testing In order to prevent exploding states during meta-testing we also clip the LSTM state between -4 and 4.

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

How input data is fed and output is read In our presented experiments we match the axis A to the input datum dimensionality such that for each dimension $\bar{i}$ we set $s_{\bar{i}b1} = x_{\bar{i}}$ for any b. Similarly, we read the output from $s_{a\bar{j}1}$ for each output dimension $\bar{j}$. Alternatively, multiple input or output dimensions could be bundled together which we will investigate in the future.

Specialization through RNN coordinates In addition to the recurrent inputs and inputs from the interaction term, each sub-RNN can be fed its coordinates a, b, position in time, or position in the layer stack. This may allow for specialization, akin to the specialization of biological neurons. In our experiments, we have not yet observed any benefits of this approach and leave this to future work.

D Additional related work

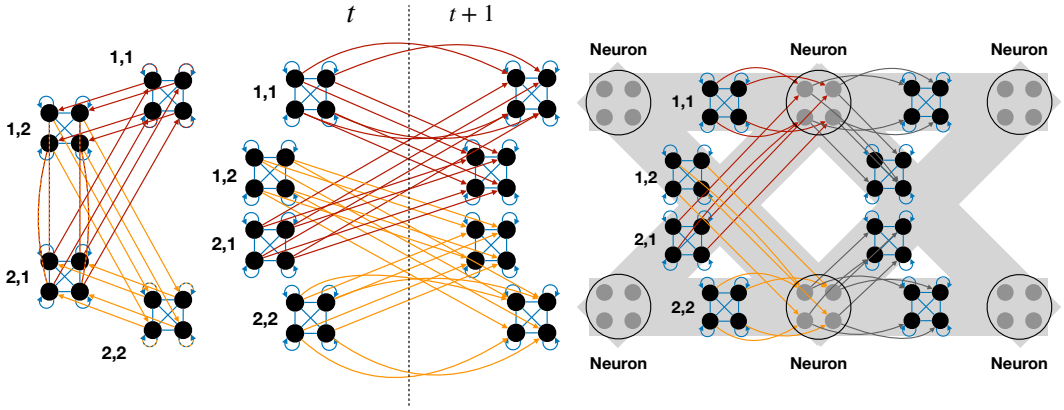
Meta learning has been used to find optimizers that update the parameters of a model by taking the loss and gradient with respect to these parameters as an input [Ravi and Larochelle, 2016, Andrychowicz et al., 2016, Li and Malik, 2016, 2017]. In this work, we are interested in meta learning that does not rely on fixed gradient calculation in the inner loop.

An interesting alternative to distributed variable updates in VS Meta RNNs is meta-learning via discrete program search [Schmidhuber, 1994, Real et al., 2020]. In this paradigm, a separate programming language needs to be defined that gives rise to neural computation when its instructions are combined. Compared to our work, this assumes global memory and modifications of the program are usually highly non-smooth making optimization expensive.

In the reinforcement learning setting multiple agents can be modeled with shared parameters [Pathak et al., 2019, Rosa et al., 2019, Huang et al., 2020]. This is related to the variable sharing in VS Meta RNNs depending on how the agent-environment boundary is drawn. Different from these works we

avoid the complexity of multiple agents. Furthermore, we demonstrate the advantage of variable sharing in meta-learning general-purpose learning algorithms.

E Visualizations of VS Meta RNNs



(a) Connectivity (b) C unrolled (c) Neural interpretation

Figure 7: The connectivity of VS Meta RNNs gives rise to a neural 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 C, we have dropped the full connectivity of V for readability (4 connections instead of 16); (b) we unrolled the recurrence in C 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 C describes synaptic connectivity and (a subset of) s defines the synaptic weight / state.