Supplementary material for "Meta-Learning Bidirectional Update Rules"

Mark Sandler [1] **Max Vladymyrov** [1] **Andrey Zhmoginov** [1] **Nolan Miller** [1] **Andrew Jackson** [1] **Tom Madams** [1] **Blaise Agüera y Arcas** [1]

1. Connection to Gradient Descent

Instead of verifying the equivalence of our update rule $\Delta \boldsymbol{w}(\boldsymbol{w})$ to GD (with $\boldsymbol{w}$ including both forward and backward weights), we may ask a more general question of equivalence to a generalized gradient descent satisfying:

$$\Delta w_j = -\gamma \sum_i g_{ij}^{-1} \frac{\partial L_{\text{equiv}}}{\partial w_i}, \tag{1}$$

where g_{ij} are components of a Riemannian metric tensor $\hat{\boldsymbol{g}}(\boldsymbol{w})$ and $\boldsymbol{w}$ are the model weights. Since Eq. (1) can be rewritten as $-\gamma dL_{\text{equiv}} = \hat{\boldsymbol{g}}\Delta\boldsymbol{w}$, where dL_{equiv} is the exterior derivative of L_{equiv}, it follows that $d(\hat{\boldsymbol{g}}\Delta\boldsymbol{w}) = 0$, or equivalently

$$\frac{\partial}{\partial w_r}\left(\sum_j g_{ij}\Delta w_j\right) = \frac{\partial}{\partial w_i}\left(\sum_j g_{rj}\Delta w_j\right). \tag{2}$$

For contractible parameter spaces, this condition is also sufficient for the existence of L_{equiv} (Lee, 2013).

Constant $\hat{\boldsymbol{g}}$. Consider a special case where $\hat{\boldsymbol{g}}$ is constant, i.e., independent of $\boldsymbol{w}$. Then we can rewrite Eq. (2) as

$$\sum_j g_{ij}\frac{\partial \Delta w_j}{\partial w_r} = \sum_j g_{rj}\frac{\partial \Delta w_j}{\partial w_i},$$

or introducing an $n \times n$ matrix $Q_{ij} := \partial \Delta w_i / \partial w_j$ with $n = \dim \boldsymbol{w}$ as:

$$\hat{\boldsymbol{Z}}[\hat{\boldsymbol{g}}, \hat{\boldsymbol{Q}}] := \hat{\boldsymbol{g}}\hat{\boldsymbol{Q}} - (\hat{\boldsymbol{g}}\hat{\boldsymbol{Q}})^\top = 0. \tag{3}$$

For $\hat{\boldsymbol{g}}$ to be a metric tensor, it has to be symmetric and positive definite and Eq. (3) should be satisfied for all possible $\hat{\boldsymbol{Q}}(\boldsymbol{w})$ calculated at all accessible states $\boldsymbol{w}$.

In our numerical experiments, we studied a pre-trained two-state model learning a 2-input Boolean task with a single hidden layer of size 5. Finding a null space of the matrix corresponding to a linear system (3) for some $\hat{\boldsymbol{Q}}(\boldsymbol{w}_*)$, we then checked the validity of Eq. (3) for the basis vectors of this null space and 100 other $\hat{\boldsymbol{Q}}(\boldsymbol{w})$ matrices calculated at different other random states $\boldsymbol{w}$. Out of 265 basis vectors[1], 4 solved Eq. (3) for all 100 of matrices $\hat{\boldsymbol{Q}}$, but as we verified later, none of those 4 vectors (or $\hat{\boldsymbol{g}}$ candidates) were positive-definite and actually had 0 eigenvalues. This shows that at least in this particular case, our update rules $\Delta\boldsymbol{w}(\boldsymbol{w})$ cannot be represented as a generalized GD as shown in Eq. (1).

Equivalence to SGD and closed trajectories. Solving Eq. (2) for a Riemannian metric $\hat{\boldsymbol{g}}$ in a more general case is difficult. There are certain special cases, however, where Eq. (2) does not have solutions. For example, if the vector field defined by $\Delta\boldsymbol{w}$ supports closed integral curves, there are trivially no $\hat{\boldsymbol{g}}$ and L_{equiv} that would satisfy Eq. (1). Indeed, if they existed, then parameterizing this closed integral curve Γ by $t \in [0, 1]$, we would obtain:

$$0 = \int_0^1 \frac{dL_{\text{equiv}}(\Gamma(t))}{dt}\,dt = \int_0^1 dL_{\text{equiv}}(\dot{\Gamma})\,dt = \\ = -\gamma \int_0^1 \hat{\boldsymbol{g}}^{-1} dL_{\text{equiv}} dL_{\text{equiv}}\,dt = 0,$$

which cannot hold for a Riemannian metric $\hat{\boldsymbol{g}}$.

2. Modified Oja's Rule

Oja's rule is generally derived by augmenting weight update with the normalization multiplier (Oja, 1982). Writing a normalized weight update as:

$$w_{ij}^* = \frac{w_{ij} + \gamma\phi[w_{ij}, x_i, y_j]}{\left[\sum_i \left(w_{ij} + \gamma\phi[w_{ij}, x_i, y_j]\right)^2\right]^{1/2}}, \tag{4}$$

where x and y are pre-synaptic and post-synaptic activations, and w^* is the weight at the next iteration, we can see that in

[1]Google Research. Correspondence to: Mark Sandler <sandler@google.com>.

Proceedings of the 38th International Conference on Machine Learning, PMLR 139, 2021.

[1]Found by performing SVD and taking vectors corresponding to singular values below a 10^{-8} threshold.

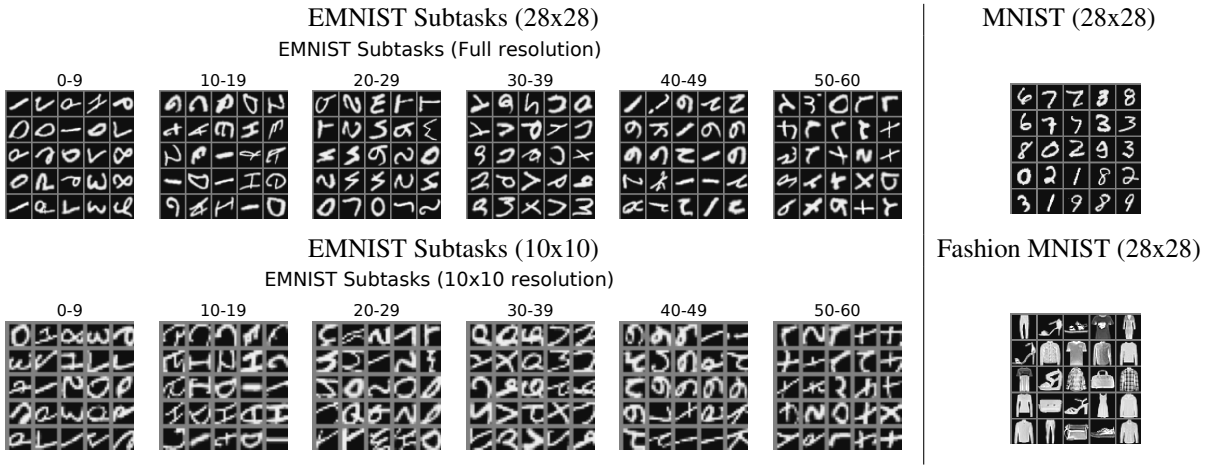

Figure 1. Dataset subtasks at 10x10 and 28x28 resolution

the limit of $\gamma \to 0$, this update rule can be rewritten as:

$$w_{ij}^* = w_{ij} + \gamma\phi[w_{ij}, x_i, y_j] - \\ - \gamma w_{ij} \sum_i w_{ij}\phi[w_{ij}, x_i, y_j] + O(\gamma^2). \quad (5)$$

For the Hebbian update $\phi = x_i y_j$, the last $O(\gamma)$ term in Eq. (5) becomes a familiar Oja's term:

$$-\gamma w_{ij} \left(\sum_r w_{rj} x_r \right) y_j = -\gamma w_{ij} y_j^2. \quad (6)$$

For more complicated networks and families of update rules, the canonical form of the Oja's rule (6) is no longer applicable. Let us derive a saturating term for the family of update rules from Section 3.2, where the updated weight now has the form $\hat{\boldsymbol{w}}^* = \tilde{f}\hat{\boldsymbol{w}} + \tilde{\eta}\hat{\boldsymbol{\phi}}$. Expressing $\tilde{f}$ as $1 + \tilde{\eta}\beta$, we can rewrite the update rule as $\hat{\boldsymbol{w}}^* = \hat{\boldsymbol{w}} + \tilde{\eta}\hat{\boldsymbol{\phi}}_\circ$, where $\hat{\boldsymbol{\phi}}_\circ = \hat{\boldsymbol{\phi}} + \beta\hat{\boldsymbol{w}}$ and the corresponding saturating term in the update rule then becomes

$$-(\tilde{f} - 1) w_{ij}^c \sum_r (w_{rj}^c)^2 - \tilde{\eta} w_{ij}^c \sum_r w_{rj}^c \phi[\boldsymbol{w}_{rj}, \boldsymbol{a}_r, \boldsymbol{a}_j],$$

where both activations $\boldsymbol{a}$ and weights $\hat{\boldsymbol{w}}$ now also have an additional "state" dimension. Substituting $\phi = \sum_{e,d} a_i^e \tilde{\nu}^{ec} \tilde{\mu}^{cd} a_j^d$ here, we finally obtain the following inhibitory component of the update rule:

$$(\Delta w_{ij}^c)^{\text{Oja}} = -(\tilde{f} - 1) w_{ij}^c \sum_r (w_{rj}^c)^2 - \\ - \tilde{\eta} w_{ij}^c \sum_{r,e,d} w_{rj}^c a_r^e \tilde{\nu}^{ec} \tilde{\mu}^{cd} a_j^d.$$

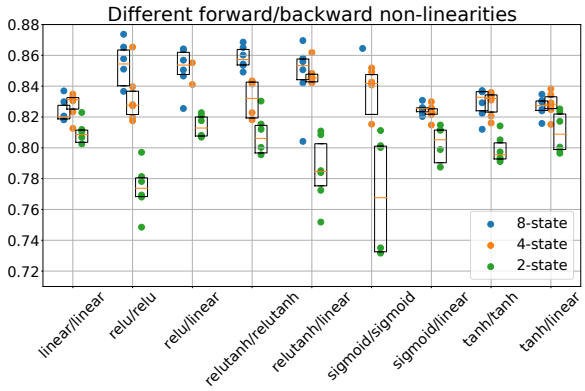

Figure 2. Performance of different non-linearities. All models were meta-trained with two hidden layers and use 2-8 states per neuron.

3. Additional Experiments

3.1. Parameters

Meta optimizer		Genome	
Optimizer	Adam	States per neuron	2-8
LR	0.0005	Batch size	128
Gradient clip	10	Unroll length	10-50

Fig. 1 shows a sample from datasets used in the experiments.

3.2. Non-linearities

On Fig. 2 we show the performance of our meta-trained while using different non-linearites. We explore using both identical non-linearities on forward/backward pass as well as not-using non-linearity in backward-ass at all at all. It can be seen that we successfully meta-learned a configuration for almost every combination of non-linearities. Also perhaps not-surprisingly fully linear network while correctly does not see any advantage of increasing number of states,

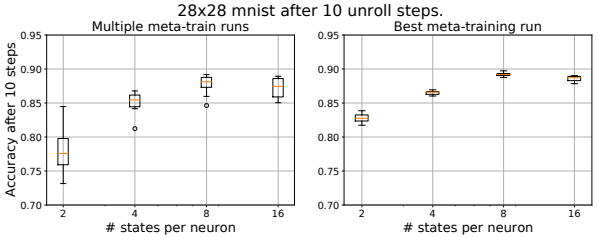

Figure 3. Performance after 10 unrolls with different number of channels per neuron. All meta-trained runs were trained with the same architecture.

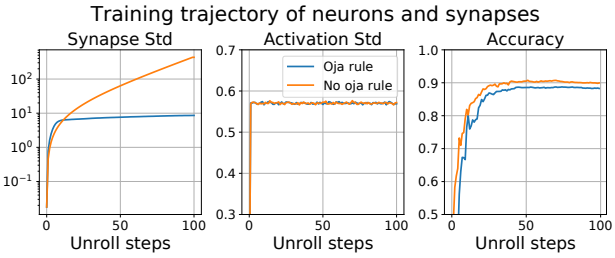

Figure 5. The impact of Oja's rule on synapse amplitude.

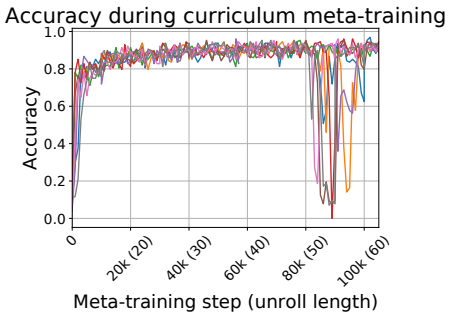

Figure 4. Trajectory of training evolution during curriculum training.

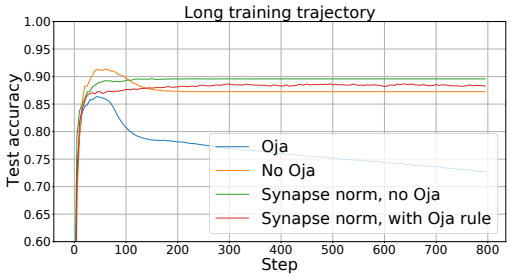

Figure 6. MNIST test error measured in the process of training models with and without additional synapse normalization. Synapse normalization ("Synapse norm" label) was only performed when computing neuron activations and did not affect the actual stored synapse values. Given stored synapse values w_{ij} with i being the input dimension and j being the output dimension, the effective weight used while computing the activations was chosen as $w'_{ij} = w_{ij}/(\sum_{i'} w^2_{i'j})^{1/2}$.

while networks with non-linearity improve with the number of states.

3.3. Channels per neuron

As we mentioned using only 2-states per neuron provides a slightly richer space than gradient descent. So what happens if we increase the number of states? Fig. 3 suggests that merely increasing the number of states improves performance, but fully capturing the power of multi-state neurons is subject of future work.

3.4. Curriculum metatraining

Fig. 4 shows the accuracy of 8-identical randomly initialized genomes trained for 10,000 steps with initial unroll length of 10 steps. We then increase the length of unroll by 5 steps for each consecutive 10,000 steps and synchronize genomes across all runs.

3.5. Importance of Oja Rule

In our experiments using additional regularization Oja's term on synapse weights helps preventing synapse explosion. For instance in Fig. 5 we show, that without modified Oja's rule the synapse weights tended to diverge, even though the overall accuracy was not affected until overflow occurred.

3.6. Synapse normalization

While reaching a high accuracy at unroll step n_{target} used as a target for meta-learning, the model performance frequently degrades when it is trained beyond that point. One technique that proved useful for solving this issue is to normalize synapse values during the computation of activations of individual layers. Fig. 6 shows evolution of the test accuracy for the best out of 10 runs in 4 different experiments: with and without synapse normalization (and with or without using Oja's rule). Notice that while the peak accuracy in experiments with normalization may be lower than in those without normalization, it continues to grow well beyond n_{target}. Furthermore, in models with Oja's rule, the synapses saturate during training and if we randomize the labels after step 500, the model performance degrades soon after, in other words, the model with synapse normalization and Oja's rule continue to learn. In contrast, synapses grow exponentially in models without the Oja's rule, which prevents them from learning after a certain stage and later their synapses overflow and they eventually diverge.

3.7. Ablation study

In Fig. 7 we show the ablation study for the following parameters:

- Type of the backward update: *additive*, *multiplicative* or *multiplicative second state only*. This refers to the states' update on the backward pass.
- Forward and backward synapses: *symmetric* vs *asymmetric*.
- Number of states in synapses: *single state* vs *multistate*.
- Initialization: *random* vs *backprop genome initialization*.

The backpropagation algorithm can be recovered as an initialization to *"Multiplicative second state only, symmetric, single state, backprop init"* variant (dashed purple line). The most general version of the algorithm that is used in most experiments in the paper is *"Additive, symmetric, multistate, random init"* variant (solid red line).

Depending on which backward update rule is chosen, different combinations of parameters dominate over others. While we couldn't find a clear pattern to order the parameters by their performance, surprisingly, most of the variants give reasonable results, suggesting that the space of potentially useful update rules is quite large.

References

Lee, J. M. *Introduction to Smooth Manifolds*. Springer, 2013.

Oja, E. Simplified neuron model as a principal component analyzer. *Journal of mathematical biology*, 15(3):267–273, 1982.

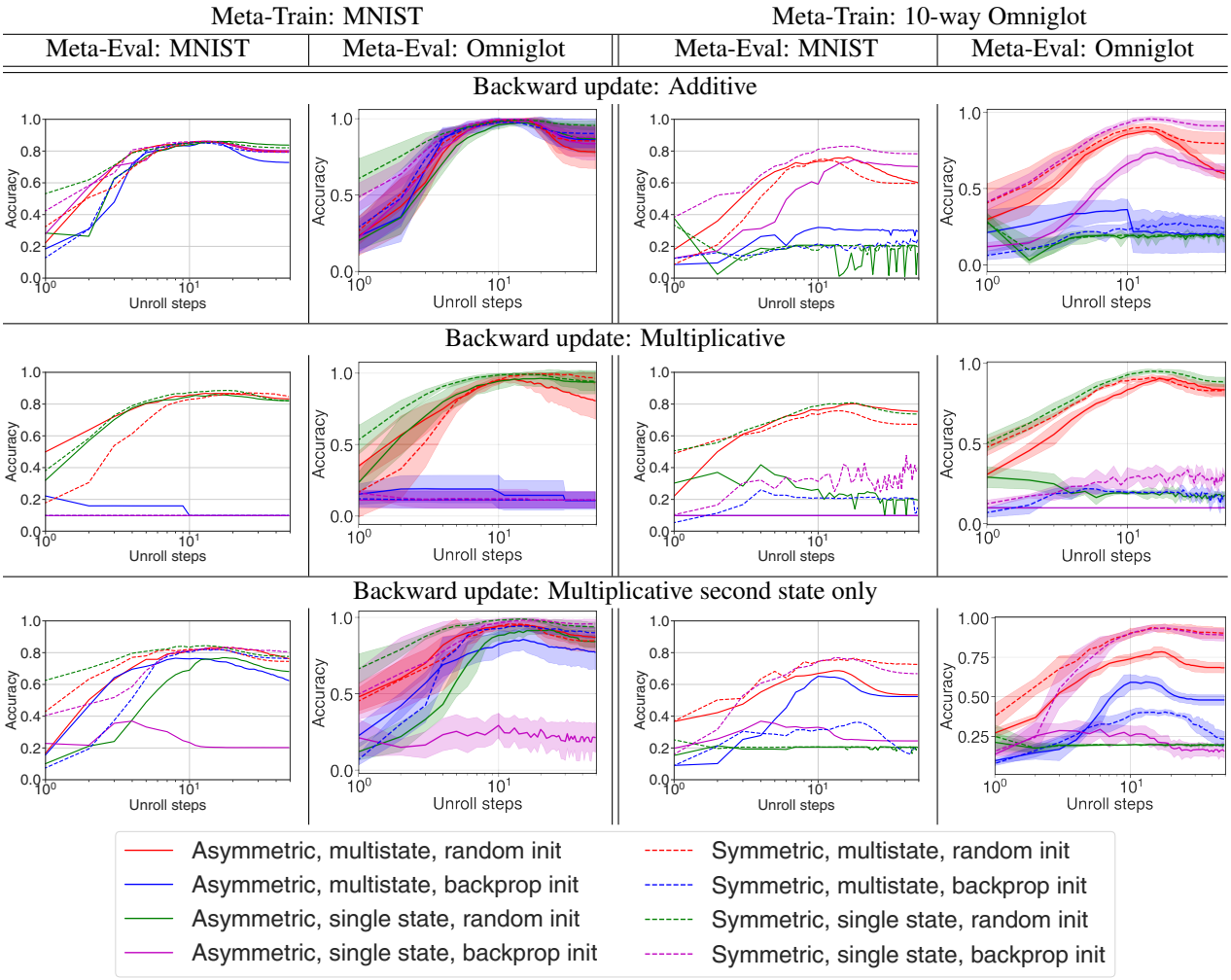

Figure 7. Testing different variants of our proposed learning method. See text for the description of each parameter. All variants were trained on MNIST or Omniglot to 10 unrolls and evaluated on MNIST and 10 classes from Omniglot. Errorbars show standard deviation of the accuracy over 10 different subsets of Omniglot. Only the best result of 8 runs is plotted.