Late Breaking Abstract

Meta-Neural Cellular Automata

Meet Barot

Mythos Scientific, New York, USA

Objectives

Training deep neural networks using backpropagation and gradient-based methods is computationally expensive. Biological neurons, on the other hand, quite efficiently learn through signals transmitted locally via synapses from other neighboring neurons. They are also able to grow new connections and can be robust to damage. Our general aim is to learn a local learning rule that demonstrates these properties for artificial neural networks.

Introduction

Neural cellular automata (NCA) models have been successful at efficient image generation soley through local update rules, demonstrating stability over many updates and robustness to perturbations [1]. In this work, we introduce a meta-neural cellular automata (MetaNCA) model to evolve neural network weights to solve a task using local rules defined by another neural network. Once trained, the local rule network allows us to efficiently sample many task networks of arbitrary feedforward architectures without backpropagation.

Method Description

Problem setup.

- Dataset $(x, y) \in D$ for a task $\mathcal{T}$.
- Task neural network that makes predictions for this task $\hat{y} = T(\theta_T; x)$ with weights $w \in \theta_T$.
- Each weight w has a hidden state vector $\vec{h} \in H_T$.
- Updates to $\theta_T^{(t)}$ and $H_T^{(t)}$ are given by a separate local rule network R, parametrized by θ_R.

Local rule network inputs.

We construct a "neighborhood perception vector" as the input to the local rule network R for a given weight w. We remove any dependence on the number of weights by averaging all the forward neighboring weights w_k of w and their hidden states:

$$w_f = \frac{1}{|\mathcal{N}_f(w)|} \sum_{w_k \in \mathcal{N}_f} w_k \qquad \vec{h}_f = \frac{1}{|\mathcal{N}_f(\vec{h})|} \sum_{\vec{h}_k \in \mathcal{N}_f} \vec{h}_k$$

w_f and $\vec{h}_f$ constitute our "forward signals", and w_b and $\vec{h}_b$ are "backward signals" which we concatenate:

$$v_{perception} = [w, \vec{h}, w_f, \vec{h}_f, w_b, \vec{h}_b]$$

The updates to w and $\vec{h}$ for the next timestep $t+1$ are finally given by the local rule network R:

$$\Delta w^{(t+1)}, \Delta \vec{h}^{(t+1)} = R(\theta_R; v_{perception})$$

The local rule network R is trained via backpropagation over many updates to the task network; the task network is only changed via local rule update.

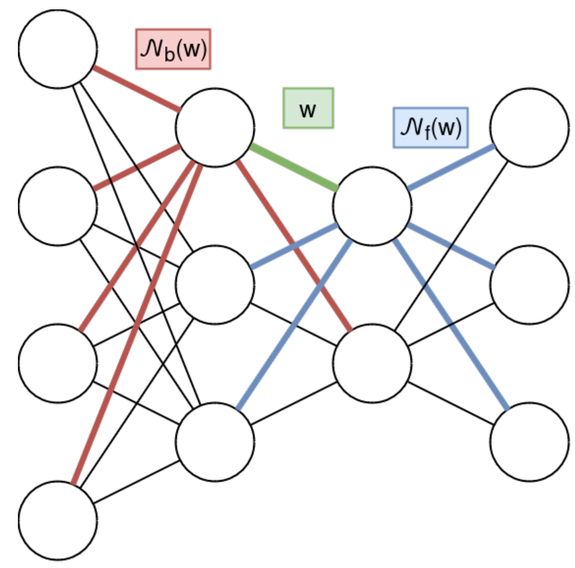

Figure 1:Illustration of the forward and backward neighborhoods $\mathcal{N}_f$ and $\mathcal{N}_b$ of an example weight w.

Architecture generalization

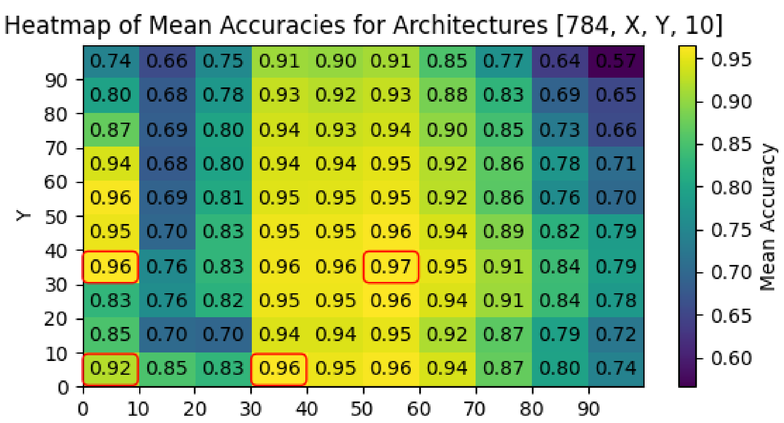

Figure 2:Performances of task networks trained using a local rule network to test generalization across architectures. Architectures that were included in the meta-training phase are indicated with red rectangles. Performances are on 20% randomly split test set from MNIST, averaged across 100 model samples per architecture.

Stability

We measure the ability of a trained local rule network to update a task network as many times as possible without breaking down the performance. Similar to the sample-pooling strategy in [1], we sample models from the prior metaepoch to be the initial state for the next meta-epoch along with reinitialized models. In Table 1 we see an increase in stability over many epochs with sample-pooling.

	Epochs to train	Epochs within 10%
Non-SP	5	13
SP	5	496

Table 1:Stability experiment. "SP" = "Sample Pooled". "Epochs to train" indicates number of local rule updates to arrive at the minimum training loss, "Epochs within 10%" indicates how many updates are done after the minimum is reached, where the loss is still within 10% of the minimum. Experiment done on iris dataset 80% training split.

Contact Information

- Web: https://www.meetbarot.com
- Web: https://www.mythos.science
- Email: meetbarot@nyu.edu

Robustness

Lastly we define a type of perturbation, **zero-out**, as setting a set of random weights as well as an entire layer of the task network to zero, to see if the local rule network grows back the weights to recover the performance.

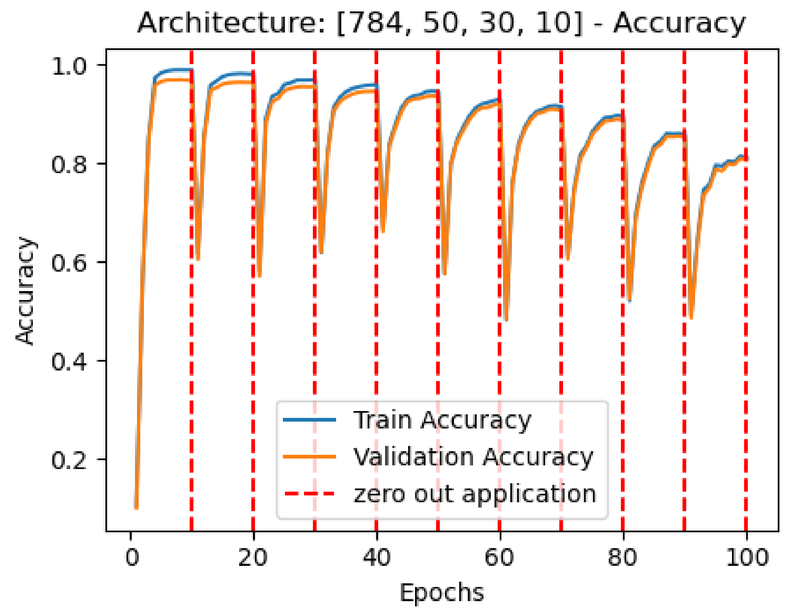

Figure 3:Plot of the effects of zero-out while updating a task network with a local rule network trained with zero-out conditions, where a random layer is set to 0 and 30% of the rest of the weights are set to 0. Performance is on the MNIST dataset.

Conclusion

We have shown the ability of the local rules learned by MetaNCA to generalize across architectures, to be stable for many local updates, and able to recover from setting much of the network to zero after reaching maximum performance.
This method allows for fast sampling of models of architectures not seen during training. However, the local rule updates are not conditioned directly on the data, and therefore any generalization of a local rule to new tasks would be limited at best. In the future we wish to add a dependence of the local rule on the data, incorporating signals from activations and loss values, similar to [2] except not explicitly relying on ordering of updates.

References

[1] Alexander Mordvintsev, Ettore Randazzo, Eyvind Niklasson, and Michael Levin.
Growing neural cellular automata.
Distill, 5(2):e23, 2020.

[2] Ettore Randazzo, Eyvind Niklasson, and Alexander Mordvintsev.
Mplp: Learning a message passing learning protocol.
arXiv preprint arXiv:2007.00970, 2020.