Meta-Neural Cellular Automata

Meet Barot[1],
[1]Mythos Scientific, New York, USA
meetbarot@nyu.edu

Abstract

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. 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 (Mordvintsev et al., 2020). In this preliminary 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. We present experiments demonstrating the local rule's generalization across architectures, stability over many local rule updates and robustness to perturbations of setting random weights or entire layers to zero.

Method Description

Problem setup. We have a dataset $(x, y) \in D$ for a task $\mathcal{T}$, and a corresponding task neural network that makes predictions for this task $\hat{y} = T(\theta_T; x)$. The task network T has weights $w_{ij}^L \in \theta_T$, where L is a layer of the task network and i and j are indices of the weight matrix for layer L. In addition, for each weight w_{ij}, the task network also has a hidden state vector $\vec{h}_{ij}$.

The updates to $\theta_T^{(t)}$ and $H_T^{(t)}$ are given by a separate local rule network R, parametrized by θ_R. For a given weight w_{ij}, we take the neighboring weights and hidden state vectors of w_{ij} as the input to the local rule network R.

Local rule network inputs. We construct a "neighborhood perception vector" as the input to the local rule network R in the following way. We start with a particular weight w and its corresponding hidden state $\vec{h}$. To give a sense of direction for the neighborhood of w, we define the "forward" neighbors and "backward" neighbors as those weights connected to the forward and backward neurons of w; see Figure 1 for an illustration. In order for the local rule network to be able to output updates for any weight in any layer in the task network, we remove the dependence on any particular number of weights by averaging all the forward neighboring weights w_k of w and their hidden states:

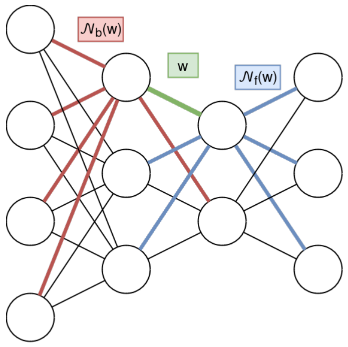

Figure 1: Illustration of the neighborhood of an example weight w.

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

w_f and $\vec{h}_f$ constitute our "forward signals". We do the same for the backward neighboring weights and hidden states, obtaining w_b and $\vec{h}_b$. We concatenate all these components in this way:

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

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

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

Training the local rule neural network. For our experiments on the Iris (Fisher, 1936) and MNIST (Deng, 2012)

	Epochs to train	Epochs within 10%
Non-sample pooled	5	13
Sample Pooled	5	496

Table 1: Stability experiment. ”Epochs to train” indicates number of local rule updates to arrive at the minimum 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.

datasets, we use a cross-entropy classification loss. We update the weights of the local rule neural network by backpropagating the gradient through the task neural network through to the local rule neural network over many epochs (i.e., local rule updates) of the task network. We only update the weights of the local rule network in this way; the task network is exclusively updated by the local rule network outputs. To simulate asynchronous updates, we apply the local rule updates stochastically to 80% of weights of the task net per local rule update for our experiments.

Experiments

We investigate MetaNCA in architecture generalization ability, stability of the task network over many local rule network updates, and robustness to perturbations of task network weights. We report our findings in architecture generalization experiments on the MNIST dataset, and stability and robustness experiments on the Iris dataset.

Architecture generalization. We measure the ability of a trained local rule network to train task networks of architectures not included in its training set of architectures. We show a heatmap of performances for various training and testing architectures for the MNIST dataset in Figures 2.

Stability. We measure the ability of a trained local rule network to continuously train a task network for as long as possible beyond the meta-training setting of updating for a fixed number of epochs, without breaking down the performance of the model. We measure the number of epochs after reaching minimum training loss before the task network deviates from the minimum significantly. Similar to the sample-pooling strategy in Mordvintsev et al. (2020), we take samples of previous models after a number of iterations to be the initial state for the next meta-epoch, with fully reinitialized models included as well. We measure this for both sample pooled and non-sample pooled settings in the Iris dataset, and as observed in Table , we see an increase in stability over many epochs with sample-pooling.

Robustness. Lastly we define one 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. We perform this experiment on the Iris dataset for both local rule networks trained in these conditions and those without these conditions present in training, where sample-pooled models have this perturbation applied. With no zero out conditions in training, the performance of the model quickly suffers and does not recover. However, we observe that when including zero out conditions in training, where we set one layer fully to zero and 30% of the other layers' weights to zero between metaepochs, the resulting trained local rule network is able to continuously recover the original accuracy from 20 zero out perturbations over the course of 100 epochs.

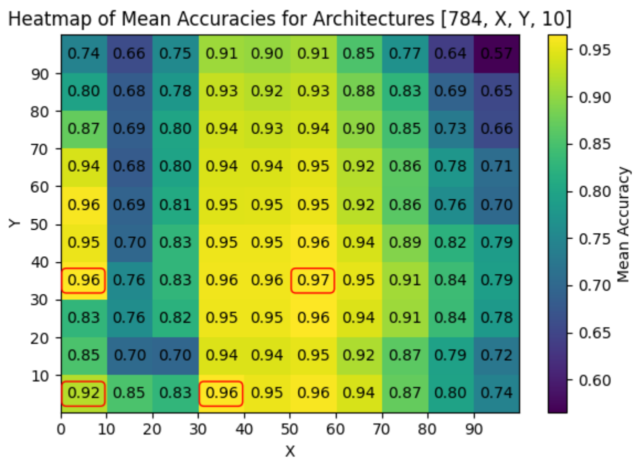

Figure 2: Heatmap of performances of task networks trained using a local rule network to test generalization across architectures. The only architectures that were included in the meta-training phase are indicated with red rectangles. These performances are on the MNIST dataset randomly split into 80% training and 20% test sets, averaged across 100 model samples per architecture.

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 Randazzo et al. (2020) except not explicitly relying on ordering of updates.

References

Deng, L. (2012). The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE signal processing magazine*, 29(6):141–142.

Fisher, R. A. (1936). The use of multiple measurements in taxonomic problems. *Annals of eugenics*, 7(2):179–188.

Mordvintsev, A., Randazzo, E., Niklasson, E., and Levin, M. (2020). Growing neural cellular automata. *Distill*, 5(2):e23.

Randazzo, E., Niklasson, E., and Mordvintsev, A. (2020). Mplp: Learning a message passing learning protocol. *arXiv preprint arXiv:2007.00970*.