MACHINE INTELLIGENCE COMMUNITY

Decoupled Neural Interfaces using Synthetic Gradients

Max Jaderberg, Wojciech Marian Czarnecki, Simon Osindero, Oriol Vinyals, Alex Graves, Koray Kavukcuoglu

DeepMind

Summary by Justin Chen

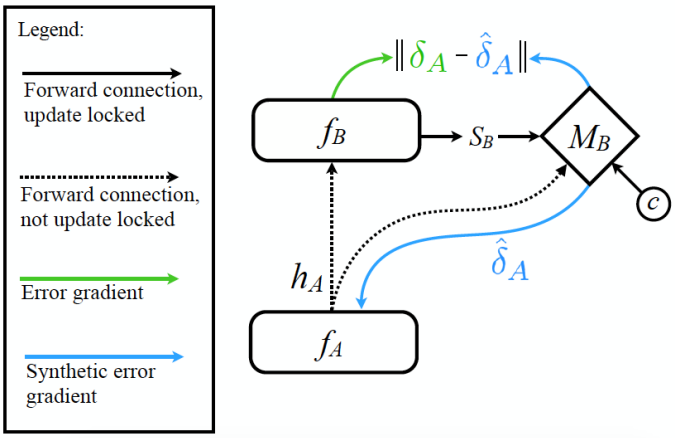
1 Introduction

- **Locking** - when a computation in the graph must wait for prerequisite computations
- **Forward Locking** - Parameters in current layer must wait for signals from previous layer/module before producing output
- **Update Locking** - Parameters cannot be updated until forward pass completed and cost function computed
- **Backwards Locking** - Parameters in current layer/module cannot be updated until entire forward pass and backwards pass for dependent layers are completed
- Locking delays parameter updates and is bottlenecked by slowest update
    - Disadvantageous if want to update models in multi-agent settings, or if agents are operating at different timescales, or if distributed training
- **Goal: Remove all locking and remove update locking for RNN by approximating backpropagation**
- Decomposing backprop:

$$\frac{\partial L}{\partial \theta_i} = f_{BProp}((h_i, x_i, y_i, \theta_i), (h_{i+1}, x_{i+1}, y_{i+1}, \theta_{i+1}), ...)\frac{\partial h_i}{\partial \theta_i} \simeq \hat{f}_{BProp}(h_i)\frac{\partial h_i}{\partial \theta_i}$$

- h = activations, x = input, y = supervision, L = loss function
- **Module** - logical component (e.g. layer or some subdivision) of computation graph
- **Neural Interface (NI)** - connection between two modules in computation graph
- **Decoupled Neural Interface (DNI)** - a neural network that mediates communication asynchronously between modules
    - Frames interaction between layers as a local communication problem
    - Removes dependence of global knowledge of system
    - **Synthetic Gradient Model** - immediately responds to previous layer's input with an approximate error gradient
        - Removes **backwards locking**
        - Final regression layer initialized with zero weight and biases
    - **Synthetic Input Model** - immediately responds to next layer with synthetic inputs
        - Removes **forwards locking**
- **Synthetic Gradient** - approximate error gradient w.r.t. input from previous layer
    - Acts as a regularizer and can cause underfitting, refer to cDNI

2 Decoupled Neural Interfaces



Decoupled Neural Interface using Synthetic Gradient

$$\hat{\delta}_A = M_B(h_A, s_B, c)$$

$\hat{\delta}_A$ Synthetic gradient for previous module f_A

M_B Synthetic Gradient Model for subnetwork B that predicts synthetic gradient from previous module f_A

Subnetwork - section of neural network that uses DNI to communicate with other portions of the network

h_A Activation of previous module f_A

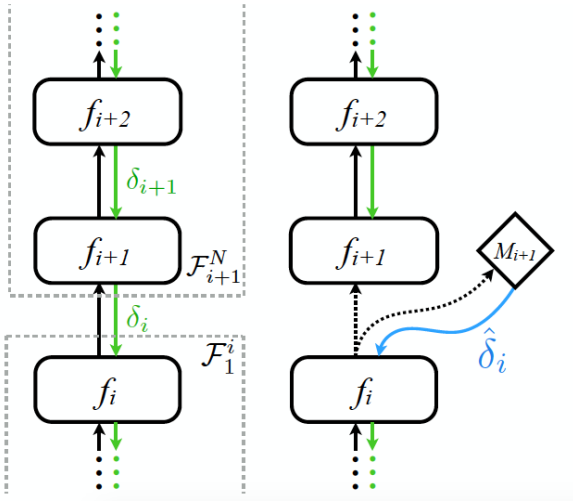
s_B Activation of next module f_B

c **Context** - Additional/privileged information that could help subnetwork learn e.g. label information

δ_A **True utility** - actual error gradient that backpropagation would have produced

- Overtime, M_B can learn to minimize difference between $\hat{\delta}_A$ and δ_A
- Removes update locking because f_A can update without waiting for it's true utility from f_B
    - Additionally, pressures f_A to learn to produce useful signals for f_B
    - M_B is **model agnostic**. Can also be used in target propagation, or reward in reinforcement learning, etc. Can be used with any optimization algorithm.

2.1 Synthetic Gradient for Feed-Forward Networks

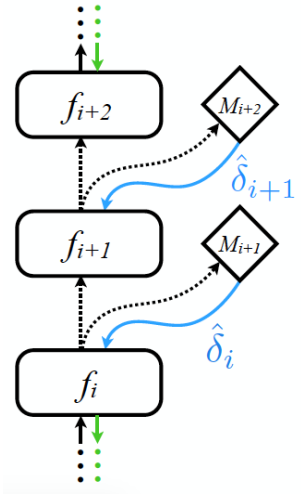


How to decouple neural interfaces in a feedforward network

$$\theta_i \leftarrow \theta_i - \alpha\delta_i \frac{\partial h_i}{\partial \theta_i}; \delta_i = \frac{\partial L}{\partial h_i}$$

Synthetic Gradient Model update rule

- $\delta_i = \frac{\partial L}{\partial h_i}$ computed with backpropagation



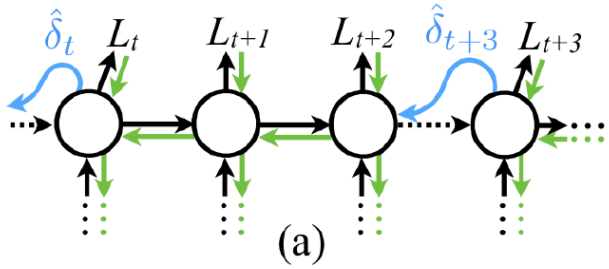
Decouple every layer

$$\theta_n \leftarrow \theta_n - \alpha\hat{\delta}_i \frac{\partial h_i}{\partial \theta_n}, n \in \{1, ..., i\}$$

Synthetic Gradient Model Update Rule for multiple DNI

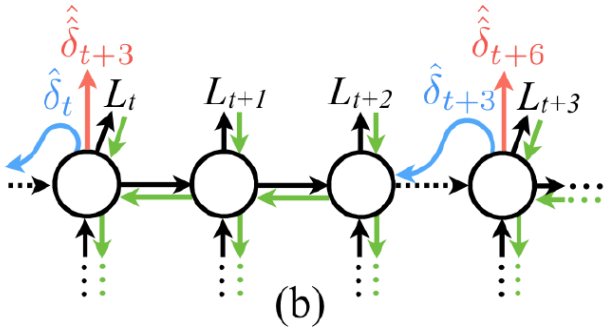
- Train modules M_i
  - Update locked
  - Trained to minimize $L_{\delta_i} = d(\hat{\delta}_i, \delta_i)$, where the loss is L_2

2.2 Synthetic Gradient for Recurrent Networks



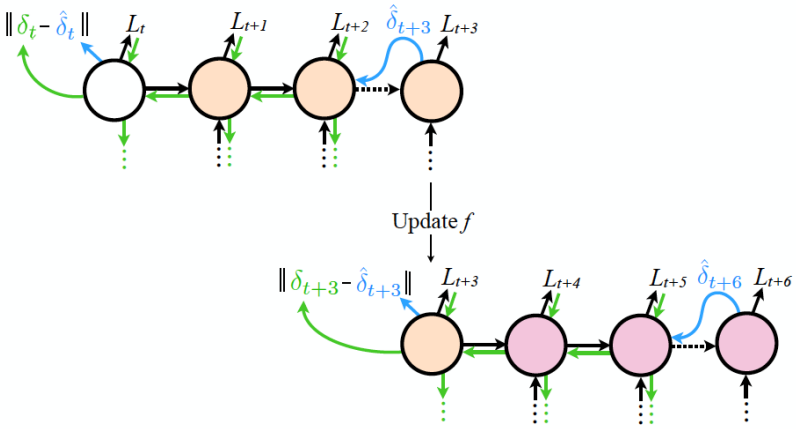
Decoupled Neural Interface for RNN

- DNI at unrolled RNN **time boundary** to predict synthetic gradients
  - Normal gradients cannot cross time boundary via **backpropagation through time (BPTT)**
  - Estimates **infinite time horizon**
  - Synthetic gradient produced at each timestep



Auxiliary task: predict synthetic gradients for future synthetic gradients

- **Future Synthetic Gradient** $\hat{\hat{\delta}}_{t+T}$ - a prediction of a synthetic gradient for a synthetic gradient at a future timestep



Training RNN with DNI

1. Save final RNN core from current unroll (first row, white circle)
2. Unroll T fresh RNN cores (first row, orange circles)
3. Compute $L_{t+3} = L_2(\delta_{t+3}, \hat{\delta}_{t+3}) = \|\delta_{t+3} - \hat{\delta}_{t+3}\|$
4. Update DNI and backprop to RNN core at time t (white)
5. Delete T RNN cores in memory except last RNN core (last orange) and repeat
    a. Use synthetic gradient $\hat{\delta}_{t+T}$, here $\hat{\delta}_{t+3}$, for next set of fresh RNN cores (second row, purple circles)

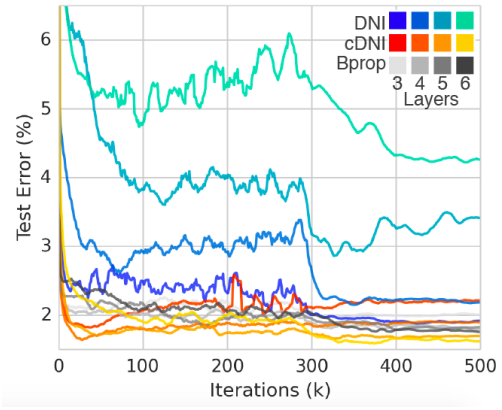
Arbitrary Network Graphs

- DNI can be used for any architecture and even communication between ensembles

3 Experiments

3.1 Feed-Forward networks

	Layers	MNIST (% Error)				CIFAR-10 (% Error)			
		No Bprop	Bprop	DNI	cDNI	No Bprop	Bprop	DNI	cDNI
FCN	3	9.3	2.0	1.9	2.2	54.9	43.5	42.5	48.5
	4	12.6	1.8	2.2	1.9	57.2	43.0	45.0	45.1
	5	16.2	1.8	3.4	1.7	59.6	41.7	46.9	43.5
	6	21.4	1.8	4.3	1.6	61.9	42.0	49.7	46.8
CNN	3	0.9	0.8	0.9	1.0	28.7	17.9	19.5	19.0
	4	2.8	0.6	0.7	0.8	38.1	15.7	19.5	16.4

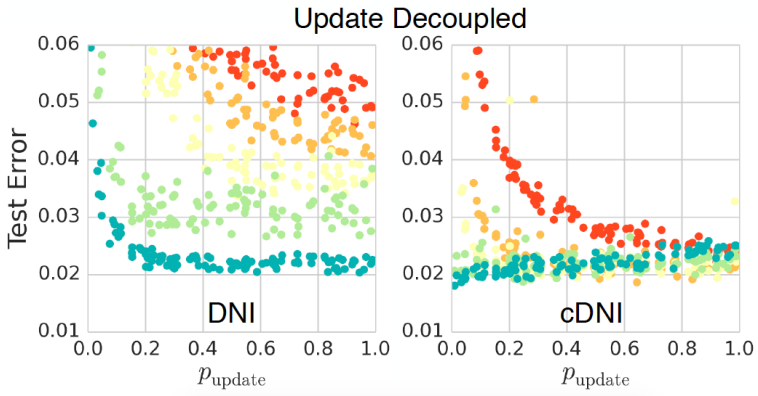


Every layer DNI

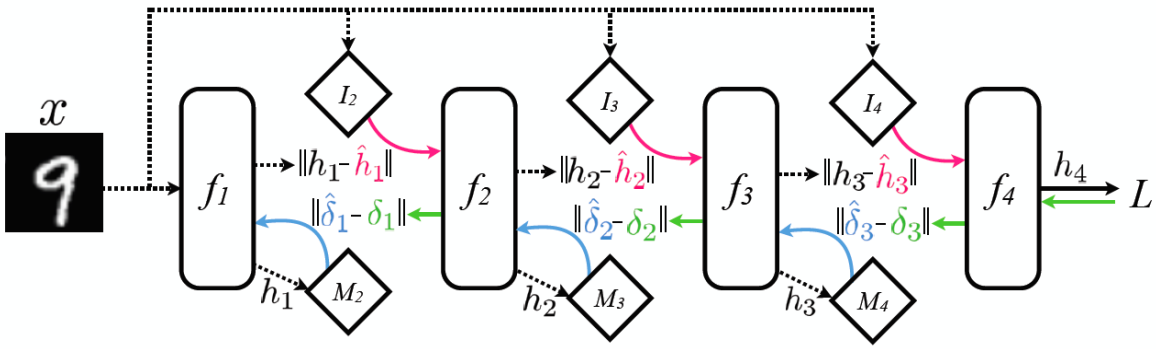
- DNI do not perform as well as training with BP on either MNIST or CIFAR-10, but difference in accuracy is small
- **cDNI** - DNI that condition on label information
    - Concatenate one-hot vector label to input to synthetic gradient model
        - For CNN, use one-hot encoded channel masks
    - Improves generalization over DNI without label information and address underfitting
- **Stale gradients** - gradients computed w.r.t. to a the model E epochs ago
    - Exponential average of stale gradients and experimented with various decay rates
    - Generalization: DNI < Stale Gradients < cDNI
    - Only used gradients that were stale by **at most one epoch**
- Tried CNN as the synthetic gradient model for predicting gradients for CNN layers
    - CNN synthetic gradient model did not use pooling and did not use zero-padding

Sparse Updates

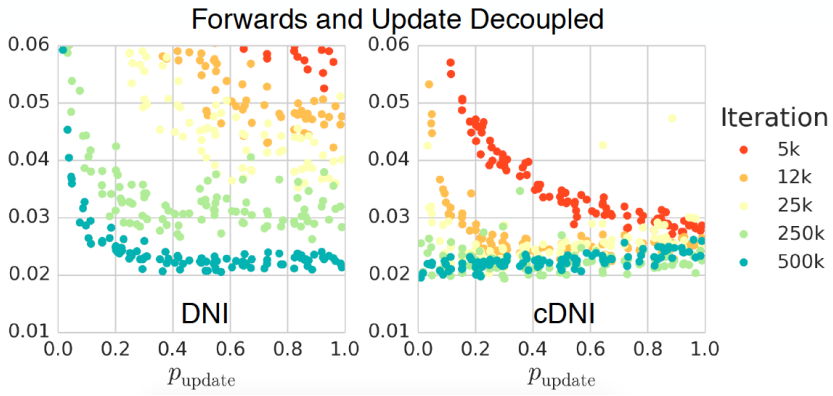
- Experimented with a 4 layer FCN
- Updates and backwards pass occurred randomly with probability p_{update}
- Ran 100 experiments with different randomly chosen probabilities
- Could still train to competitive accuracies



Complete Unlock



- **Synthetic Input Model** - given actual input vector, approximates the activations from the previous layer for the next layer
    - Removes forward locking making the network fully asynchronous
    - Each layer can be trained in parallel but together **co-learn** a single function
    - Achieves about 98% accuracy on MNIST
    - Slower convergence



3.2 Recurrent Neural Networks

- Tasks: Copy, Repeat Copy, character-level language modelling
- Synthetic Gradient Models are shared across timesteps

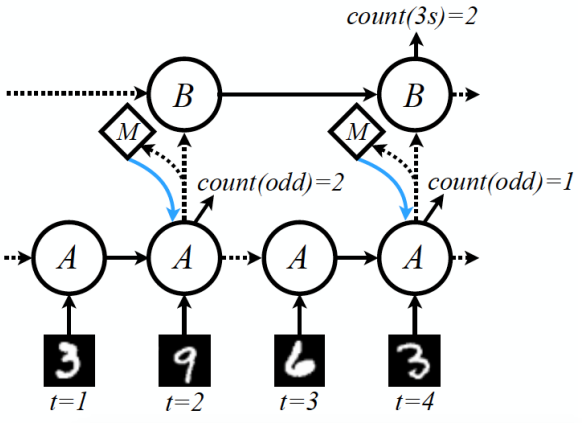
Copy and Repeat Copy

- **Copy task**: Train to read N characters and to output that exact sequence
    - $T_{task} = N + 3$
- **Repeat Copy task**: Train to read N character and an int R, and output the exact sequence R times
    - $T_{task} = NR + 3$
- Each instance of the task is an episode e.g. reading and copying entire sequence
- Test length of time RNN learns to model with and without DNI
- DNI allows modeling **longer sequences beyond the BPTT boundary** using a **smaller than typical T** and using **less data** allowing it to also **learn faster**
- Training on auxiliary task of predicting **future synthetic gradients** allows even better generalization
    - Repeat Copy Results: T = 3
        - With DNI: increases generalization from 33 timesteps (DNI) to 59 timesteps (future synthetic gradients)
        - Without DNI: only 5 timesteps

Language Modelling

- Penn Treebank dataset
- LSTM with 1024 units
- LSTM unrolled for 5 timesteps using DNI reaches same performance as vanilla RNN unrolled for 20 timesteps
    - DNI LSTM only uses 58% of data and 2x faster (wall clock time)

3.3 Multi-Network System



Collective training of RNN using DNI

- DNI allow communication between networks
- Both networks are treated as a single network
- Task: 2 RNNs A and B
    - A counts odd MNIST digits
    - B counts number of 3s
        - Events occur less frequently and thus learns at a slower rate than A
        - B must communicate with A to complete its task
    - Trained with full BP, which locks A as B updates requiring T^2 updates
    - Trained with DNI to unlocks A requiring on T updates

Comments/Questions

- In the implementation details they mention that
  "Every hidden layer consists of fully-connected layers with 256 units, followed by batch-normalization and ReLU non-linearity. The synthetic gradient models consists of two (DNI) or zero (cDNI) hidden layers and with 1024 units (linear, batch-normalization, ReLU) followed by a final linear layer with 256 units." Are they saying that when they used cDNI between FCNs on MNIST they just used a single 256x256 linear layer? That seems weird to me. Doesn't this violate the Universal Approximation Theorem?