(E)OgmaNeo Overview

As of August 2018

Eric Laukien
Ogma Corp

What is (E)OgmaNeo?

- Two versions: EOgmaNeo and OgmaNeo
- Both implement the same algorithm, one intended for embedded (EOgmaNeo) CPU-only projects and the other (OgmaNeo) for GPU and other OpenCL devices.
- They implement Sparse Predictive Hierarchies (SPH)
- Support world-model building
- Fully online/incremental learning

(E)OgmaNeo vs Deep Learning

- Sparse predictive hierarchies differ substatially from Deep Learning
  - Do not use backpropagation
  - Bio-inspired
  - Local, sparse computation
  - Models dynamical systems through changing sparse distributed representations (SDRs) at multiple scales (both spatial and temporal)

(E)OgmaNeo vs HTM

- HTM is Hierarchical Temporal Memory
  - While we share motivations and problem domains, as well as using SDRs, pretty much everything else is different
  - We represent time through “Exponential Memory”, as opposed to HTM’s sequence memory
  - We use further simplified neuron models
  - We use iterative sparse coding to learn SDRs
  - We implement hierarchy (in fact, it is vital to representing time)
  - We are dense across columns, sparse within columns
  - We predict through decoding, not through lateral prediction
  - We use continuous synapses
  - Synapses are not grown at run-time, rather we start with all needed synapses (less memory allocations/deallocations)

A Motivating Example

- How can we understand a time-driven world?
  - Build a model of the world
  - If we can accurately predict the next timestep of information at some timescale (step size), we “understand” the world at that timescale *and all timescales above*
  - How do we perform this prediction when hidden causes (partial observability) mean we need to remember information from several timesteps prior?

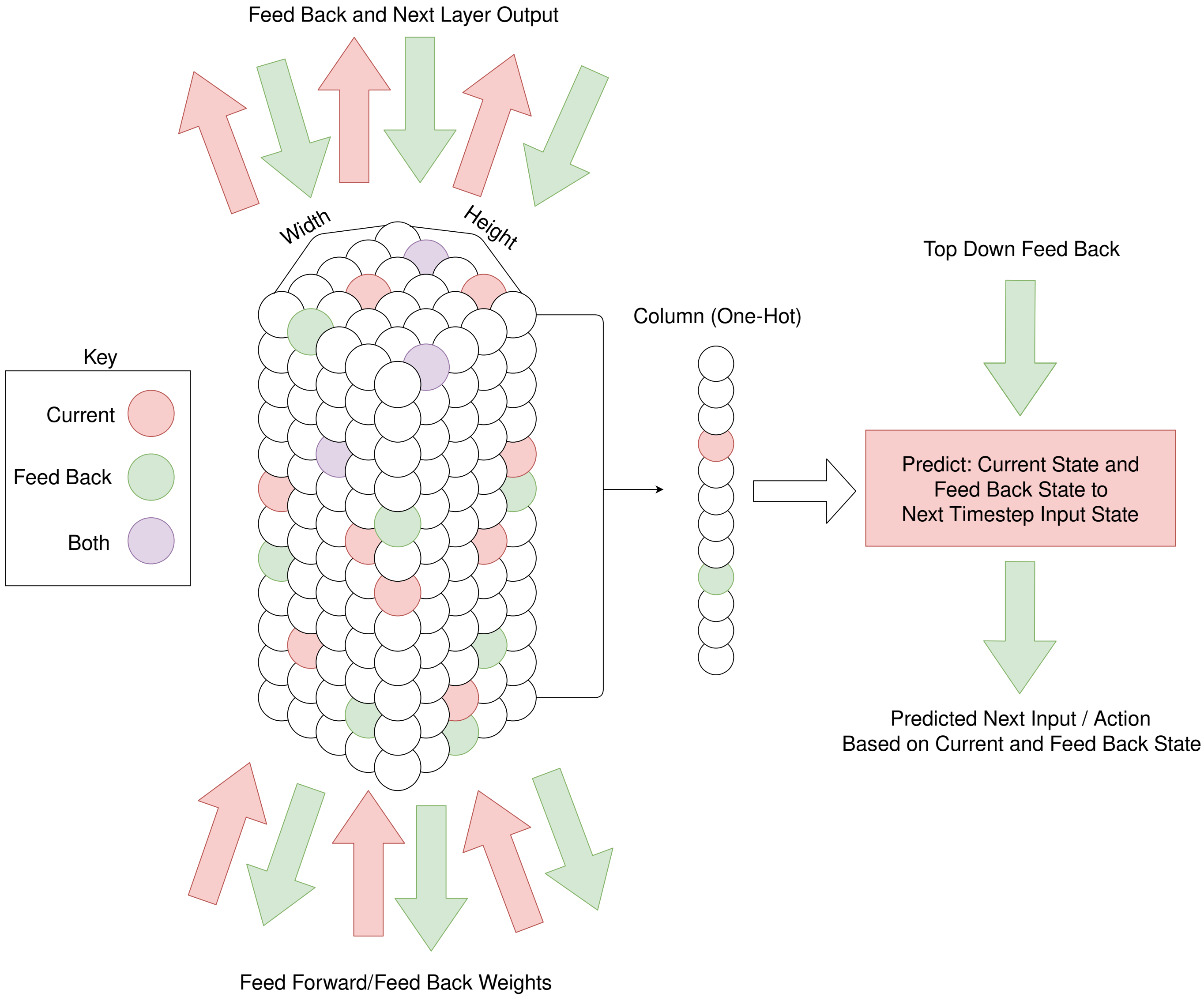
Online Learning

- We further want our system to learn online, or incrementally
- Deep Learning especially suffers from catastrophic interference in the representations of the layers
  - This is due to the use of dense representations instead of sparse ones – they are not orthogonal!
- Biological systems do not need the tens of thousands of timesteps of replay buffer data at a time in order to learn – instead, they organize their representations so that constant retraining isn't necessary

How a Single (E)OgmaNeo Layer Works

- Our layers are 3D grids of cells
- A layer has a width, height, and a column size
- It is therefore a 2D grid of columns
- A single layer takes some input, produces a hidden representation H (in the 3D grid of columns), and then maps to the input again but one timestep ahead of time
- A layer performs a $X_t \rightarrow H \rightarrow X_{(t+1)}$ mapping, where H is the hidden representation of columns
- A layer also has feed back from higher layers (used to refine predictions)
- Each layer therefore has information flowing in both directions, up and down
- Input and predictions are at the “bottom”, hidden state and feed back are at the “top”
- There are feed forward weights ($X_t \rightarrow H$)
- There are feed back weights ($H, F \rightarrow X_{(t+1)}$) where F is the feed back state

A 5x5x12 (WxHxColSize) Layer



How is H Learned?

- We have tried several algorithms. Among the most effective are Hebbian/Anti-Hebbian sparse coding, Adaptive Resonance Theory (ART), and BISTA (Binary version of the ISTA algorithm)
- H is binary, so most sparse coding algorithms require an amount of modification to work
- Our current best algorithm, Columnar Binary Sparse Coding (CBSC):

input : input states I, weight matrix W
output: sparse binary columnar cell states O
$A \leftarrow 0$ // For each column cell, initialize activation to 0

$R \leftarrow 0$ // For each input, initialize reconstruction to 0

for *numIters* **do**
  // Determine stimulus S
  $S \leftarrow W^T \cdot (I - R)$
  // Add on to activations
  $A \leftarrow A + S$
  // Inhibit activations to determine states
  $O \leftarrow ColumnWiseOneHot(A)$ // One active unit per column
  // Reconstruct
  $R \leftarrow W \cdot O$
end

return O

Algorithm 1: Columnar Binary Sparse Coding

How is the Prediction Formed?

- Simple perceptron or logistic regression
  - The H representation already has nonlinearity built in
- Reinforcement learning (experimental) version uses Q or SARSA here instead
  - Can optionally be of finite horizon due to exponential memory, which we will get to in a bit

What are the connectivity patterns like?

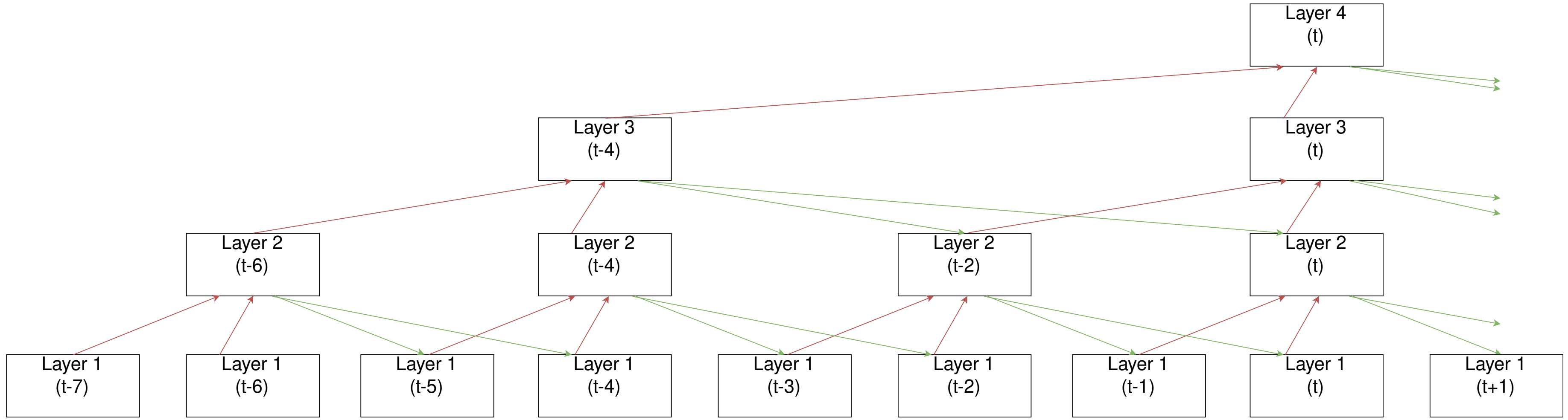
- The feed forward and feed back weight matrices are sparse
- Local receptive fields
- Not convolutional – weights are not shared, they are unique in each location. Convolution isn’t only biolgically implausible, but also computationally slower and not really needed (due to things like saccading)
- Competition between cells in H is only within a column, so all cells in the column have the same receptive field so that competition is “fair”

How is Hierarchy Implemented?

- Naive version: Just stack the layers, have each layer predict the next timestep of the H of the layer below. The lower layer then receives this prediction as feedback
  - Works, but still no sense of time! Cannot remember past 1 timestep
- Improved version: Same stacking, but instead of predicting 1 timestep of the H of the layer below, we:
  - Predict the next N timesteps
  - We clock the layer N times slower than the layer below
  - Lower layer receives slice of prediction made by the higher layer as feed back according to the current clock tick
- This is called “Exponential Memory” - a kind of bidirectional clockwork RNN
- Both versions: Extract features going up, predict going down!

Exponential Memory

- Each layer typically runs half as fast as the next layer below (N=2)
- We extract features through spacetime going up the hierarchy, and can still predict at fine timescales going down
- Results in an exponential memory horizon with respect to layer count (hence the name)
- Unlike e.g. wavenet, training is just as fast as inference!
- Each layer doubles the memory horizon, so as few as 12 layers would result in 4096 timesteps of memory!



Optimization

- With online learning and exponential memory, this is already a very fast system
  - Only need to update once per layer clock cycle
  - Layers clock exponentially less often, so less updates are needed
  - But we can take this further!
- Use the “Sparsity Optimization”
  - Since the representations are sparse, we can filter out the vast majority of cells in each timestep update since they are “0” anyways
  - Can often achieve a 10-100X speedup depending on the level of sparsity

(E)OgmaNeo details

- EOgmaNeo:
  - Multithreaded CPU implementation in C++
  - Python bindings
  - Runs on regular desktop as well as embedded devices (e.g. Raspberry Pi)
- OgmaNeo:
  - OpenCL implementation (can use GPU or CPU)
  - C++
- Both: simple but powerful interface for dealing with sparse predictive hierarchies

(E)OgmaNeo Demos

- “World’s Smallest” self-driving car
- Sidewalk following car
- Original LSTM paper tasks comparison
- Video Prediction
- OpenAI Gym (reinforcement learning)
- Level generation from pixels
- Anomaly detection

The End

- Try (E)OgmaNeo!