Sparse Predictive Hierarchies

Eric Laukien, Ogma Corp

An alternative to Deep Learning

- Deep Learning (DL) has had many successes, but still has some fundamental drawbacks
- In particular, the reliance on the backpropagation algorithm prohibits:

  1 - Online/incremental learning: Learning from data streams without replay or other heavy decorrelation mechanisms

  2 - Remembering beyond a replay buffer horizon

1 – Online/Incremental learning

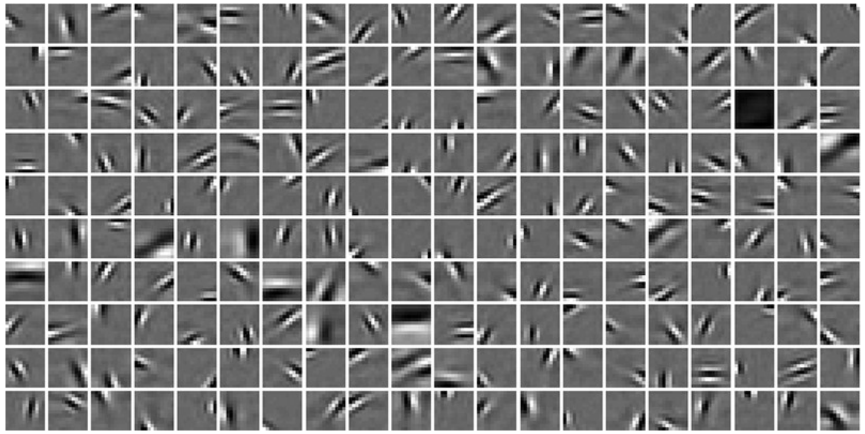
- Online (or incremental) learning allows construction of a model through observation of temporally correlated streams of data.
- Dense backpropagation has an i.i.d. assumption, which when ignored leads the model to chase the last values observed (in most cases)
- Backpropagation can therefore only learn in an incremental/online setting through the use of replay buffers or other decorrelation mechanisms.
  - These mechanisms are very slow and especially memory intensive

2 - Remembering and Catastrophic Interference

- Due to backpropagation’s need to use something like a replay buffer or similar expensive methods (such as multiple agents), Deep Learning algorithms have a tendency to forget beyond their replay buffer horizon
- This is known as catastrophic interference
  - Stems from the reliance on dense (not sparse) representations primarily, as these result in “hidden” units in a network interfering with each other (hence the name) – Dense backpropagation doesn’t generally learn orthogonal representations
- Replay buffers get extremely large and memory intensive. Iterating over them is also expensive

So what do we do?

- Age old answer: Turn to biology
- Biology doesn’t use backpropagation
  - Would require replay memory which biology doesn’t have
  - Would require hierarchical synchronization and way faster neurons
  - Animal (and human) brains are sparse – cannot backpropagate through sparse representations (non-differentiable)
- Biology seems to use sparse coding – Gabor filters are common in V1
- How do we perform credit assignment without backpropagation?

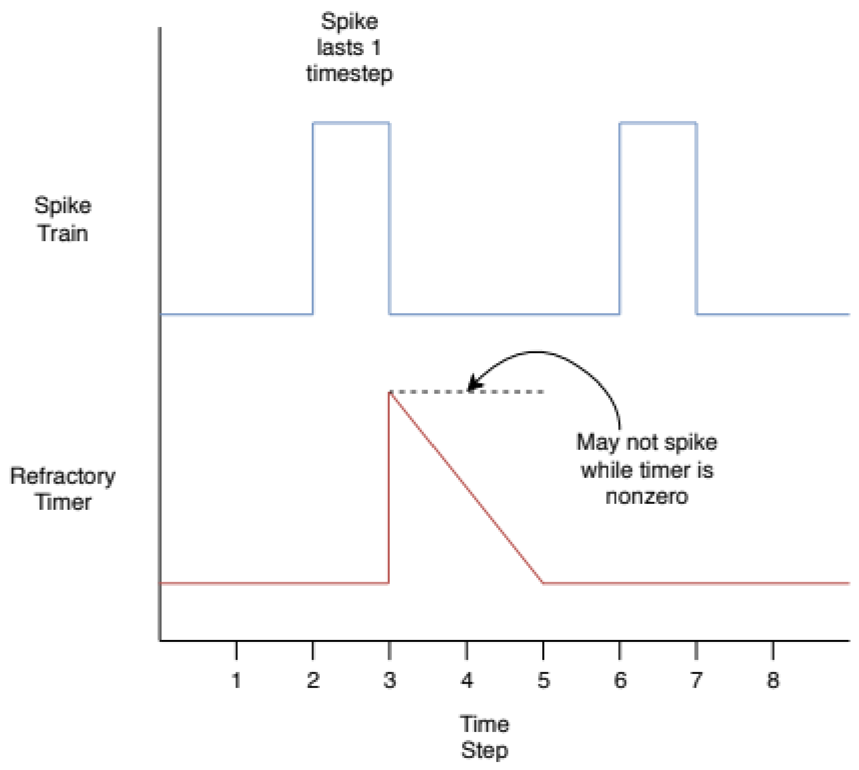

Olshausen and Field 1997

Sparse Coding

- Sparse coding seeks to find a sparse representation for a signal giving some codebook (dictionary)
- Coding can occur iteratively, and dictionaries can be updated with Hebbian rules
- There exist both biologically plausible and implausible variants, which achieve similar results with slightly different methods
- Typically requires some sort of iterative solving of the code, followed by a dictionary update (then repeat)
- Sparse coding performed on whitened natural image patches yields gabor filters
- Sparsity causes units to “compete”
- Can be made fully incremental relatively easily – since sparsity forces orthogonality in the parameters!
  - Essentially “bins” things into discrete representations with little to no overlap

Spiking Sparse Coding

- An optional improvement to the sparse coding is to not only make it sparse in space but also in time
- We can do this by switching from binary neurons to binary spiking neurons (they have a refractory period)
- Experimentally improves results depending on task

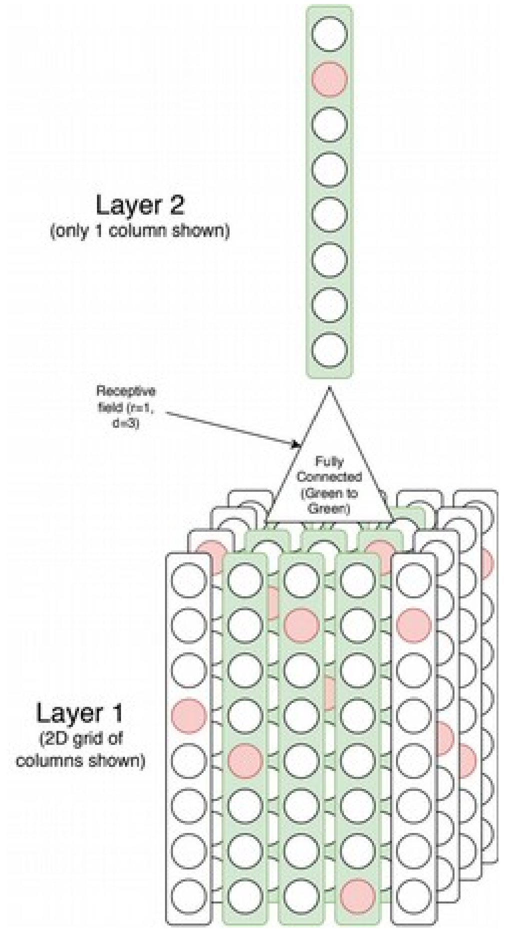


Sparse Representation Format

- We want to use sparse coding as an unsupervised learning method
- What format should our sparse code take?
  - Normally, it is unrestricted in structure – nonzero values seem uniformly randomly distributed
- Unrestricted structure works well when the dictionary is fully connected. However, when it is sparsely connected, units will compete on different input patterns, resulting in “unfair competition” that leads certain units to dominate the code
- Fix: Adopt a locally fully connected structure - “columns”
  - A column is a 1D array of “cells”
  - All cells in a column see the same input patch
  - No competition between columns, only within
  - Columns are typically one-hot (1 active unit at a time)
- In order to be able to handle visual input (2D) later on, we organize columns into a 2D grid of columns (therefore a 3D grid of cells)
- We call this structured representation a Columnar Sparse Distributed Representation (CSDR).
- Since each column is one-hot, we can represent a CSDR as a 2D array of integers, where the integers index the active unit in each column

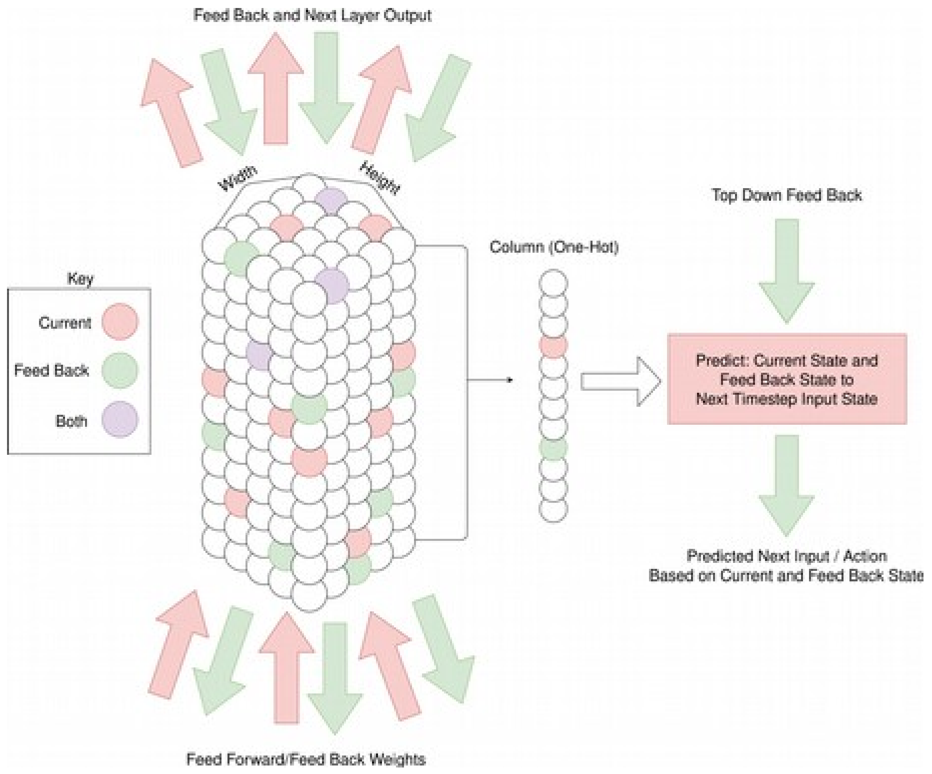
Connectivity between CSDRs ("layers")

- Connectivity is sparse, can be represented through a sparse matrix (we use Compressed Sparse Row (CSR) format)
- A column connects to other columns locally with some radius ("receptive field")
- This is NOT convolutional – connections ("synapses") between cells/columns are not shared. Biology doesn't use convolutions either, at least not ones that learn. This also has the benefit of making layer sizes very flexible with no padding needed


Layer 2
(only 1 column shown)
Receptive field (r=1, d=3)
Fully Connected (Green to Green)
Layer 1
(2D grid of columns shown)

Layers

- Layers consist of two superimposed CSDRs along with several sparse weight matrices
  - Current layer state, and predicted state
  - A layer is tasked with:
    - Encoding the input (bottom-up) into the current layer state
    - Predicting the next (t+1) input state (top-down), while taking its own state and predictions of its own state from above (feed-back) into account


Feed Back and Next Layer Output
Width
Height
Top Down Feed Back
Column (One-Hot)
Key
Current
Feed Back
Both
Predict: Current State and Feed Back State to Next Timestep Input State
Predicted Next Input / Action Based on Current and Feed Back State
Feed Forward/Feed Back Weights

Taking a step back

- We have layers now that can encode input into CSDRs and form some kind of prediction?
- Why do we predict, what do we predict, and how?

World Modeling

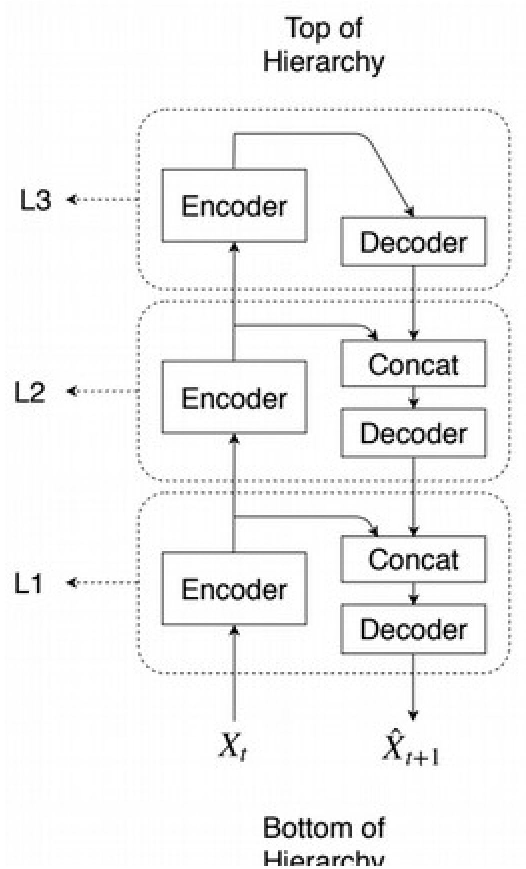
- In SPH, the goal is to predict the input one timestep ahead of time
- If we can do this successfully, we know that:
  - The model understands the world at that timescale
  - The model also understands the world at every timescale “above” (lower resolution) that timescale (since we can just chain together predictions at higher resolutions to get lower resolution ones)
- Given some vector X_t, we try to predict X_(t+1) at every t. If we can do this accurately enough, then we have successfully modeled the environment

Prediction over Layers

- As mentioned earlier, each layer predicts one timestep ahead of time
- The target of this prediction is simply the state of the layer directly below
- A layer predicts the state of the layer below (or input, if it’s the first layer) using:
  - The current layer state
  - The predictions from the layer above as context
    - Since these predictions are of the state of the current layer, they are useful context for predicting the state of the layer below as well
    - Avoids backpropagation!

Hierarchy

- We organize the layers into a bi-directional hierarchy
- Compress and extract features going up via sparse coding
- Predict next timestep of each layer going down, using next higher layer predictions as context. Predictions are formed through simple incremental regression (perceptron)
- Sparse codes provide the needed nonlinearity
- Implemented in two passes, but can also be done asynchronously



Working Memory

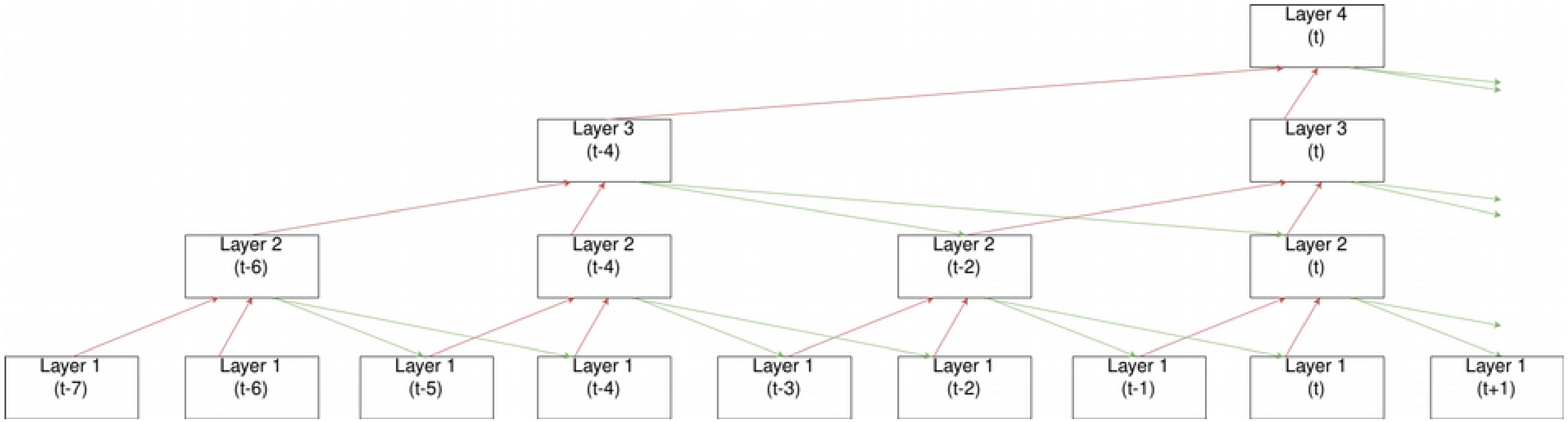
- If we keep the hierarchy as shown in the previous slide, each layer receives 1 timestep of data and produces 1 timestep of prediction
- It has no working memory, and cannot remember past 1 step
- It would seem natural to just add recurrent connections in the sparse coder
  - This works alright, but we can do better

Exponential Memory

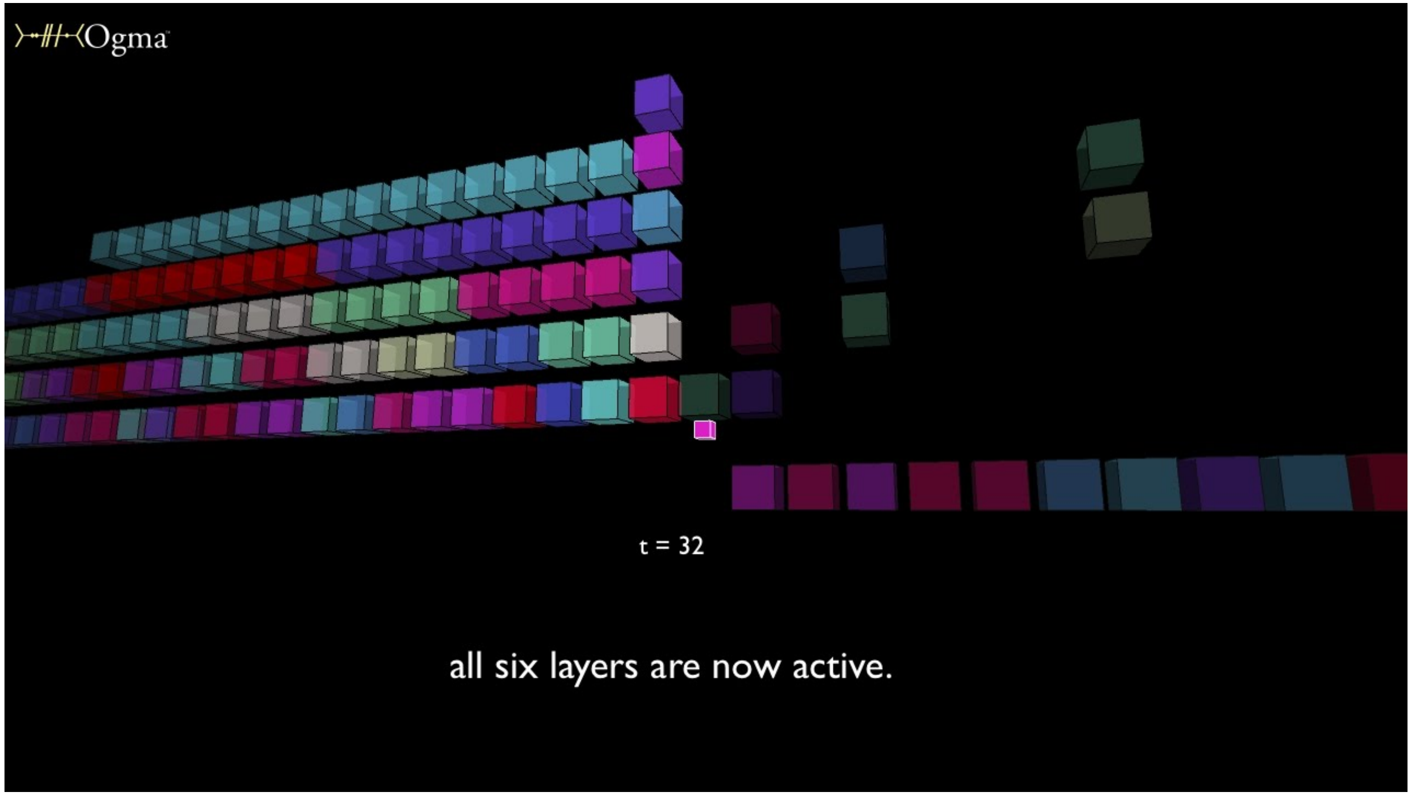
- To address the working memory issue, we make each layer take N timesteps of data and produce N timesteps as well
  - A layer therefore has both a sliding “input window” and a sliding “output window”
- We “clock” each layer N times slower than the layer below
  - Kind of like a bi-directional Clockwork RNN
- We therefore get N^(number of layers) timesteps of memory
  - Exponential growth with respect to layer count!
- So, when doing the “upward/encoding” pass, we only encode after a layer has received N inputs from the layer below
- When doing the “downward/prediction” pass, we select the particular prediction of the N predictions a layer produces to serve as context for the layer below
  - We select this “slice” of prediction based on the current layer clock, so each layer still receives a t+1 prediction from the layer above as context
  - We call this “de-striding”
- N is typically 2. If we want less memory, we can make N=1 for some layers and 2 for others.

Exponential Memory (Cont.)

Since we only need to update every layer half as often as the previous (if N=2), we cannot exceed twice the cost of the first layer in terms of processing!



Exponential Memory (Cont.)



Pre-Encoders and Pre-Decoders

- Our system assumes that both the input as well as the final prediction of the model is itself a CSDR
- CSDRs are a very specific data format, so how do we train on e.g. audio waveforms or video data?
- We use Pre-Encoders and Pre-Decoders
  - We often refer to both as just “Pre-Encoder” since the Pre-Decoder is often just a reverse transformation built into the Pre-Encoder

Example: Bounded Uniform Scalars

- We can easily pre-encode scalar signals if they are bounded and uniformly distributed
- Just one-hot encode the scalar into a single input column (1x1xColSize input)
- Since we represent one-hot columns with integers, the integer value of a column is simply:

  index = round((scalar - lowerBound) / (upperBound - lowerBound) * (ColSize - 1))

- Then to get the scalar back from the prediction:

  scalar = index / (ColSize - 1) * (upperBound - lowerBound) + lowerBound

- Useful for modeling e.g. scalar time series, since this is typically bounded and mostly uniform

Example: Video Data

- For video data, we need to use sparse coding again, but this time we are encode an image instead of another CSDR (one frame is entered into the hierarchy at a time)
- Since frames are dense, we need to be careful about how we design this pre-encoder to not have forgetting
- Biology uses far more complex algorithms that we may try to model more accurately in the future, but for now simple methods such as decaying self-organizing maps (SOMs) work fine
- Depending on the task, it is often desirable to filter the frames with e.g. a Difference of Gaussian filter or similar to extract edges. This mimics the center-surround receptive fields in the brain

Beyond Prediction

- Up until now we have only described SPH in terms of model-building and learning to predict the next timestep of data
- This is useful for many tasks, but we also want to build intelligent agents
- What about reinforcement learning?

Aside: Reinforcement Learning with SPH

- Let us assume that our input data contains both sensory (state) information as well as the action last taken by the agent
- If we predict this and act upon the action portion of the prediction, we are essentially learning how to act
- In order to maximize reward, we modify the decoders (down pass) to perform TD-learning
- This works because a prediction can be interpreted as an action
- When using exponential memory, each layer receives the average of the rewards experienced before the layer clocked

Implementation Concerns

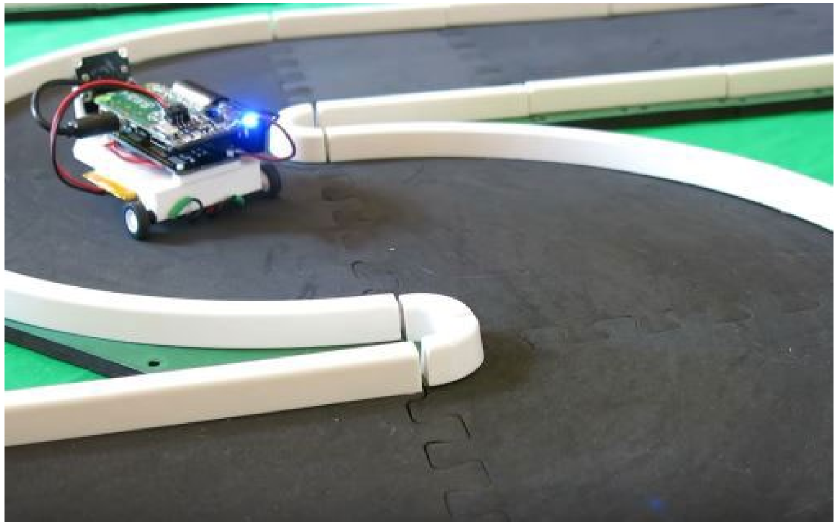
- Since the system is online/incremental and due to exponential memory, this is already a very fast system
- But we can do better
  - What about all the sparsity? Why process all the zero elements?

The Sparsity Optimization

- We can indeed ignore the zero-elements. Since CSDRs are stored as 2D arrays of integers, we already know where the active units (1’s) are located. All the rest are zero, so we only need to iterate through the nonzero elements
- Provides large speed boost depending on the level of sparsity. If the sparsity ratio is 2%, then we theoretically get a 50X speed boost!
  - Works very well in practice, too
- Dense networks require one to update all elements all the time, while with sparse networks we only need to update a small fraction at a time

Is it really that fast?

- We can perform both learning and inference for a toy vision-based self-driving car on-board a Raspberry Pi Zero (1 CPU core, 1 GHz)
- We can perform tasks such as e.g. learn sequence copy more accurately and many times faster than an LSTM can even do inference
- Can compress audio data in real-time (44000 Hz)

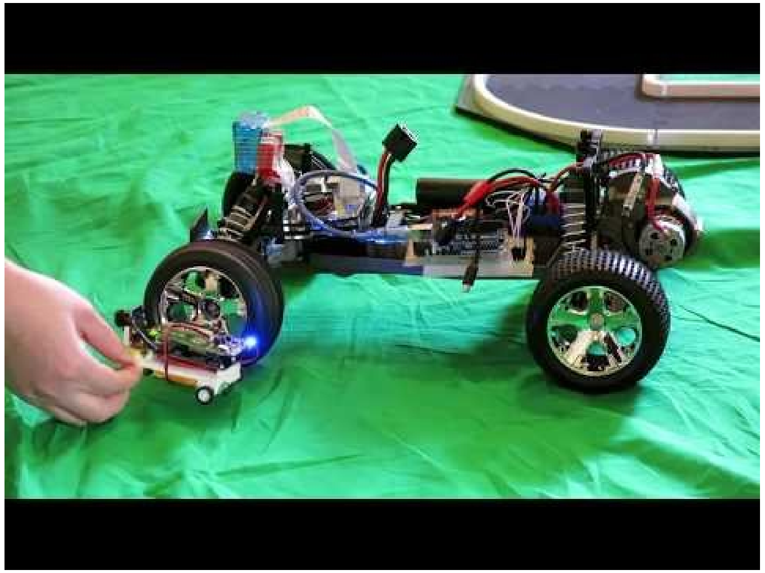


OgmaNeo

- OgmaNeo is our implementation of SPH
- Available on GitHub: https://github.com/ogmacorp/OgmaNeo2
- Python bindings: https://github.com/ogmacorp/PyOgmaNeo2

Self-Driving Micro-Car Demo

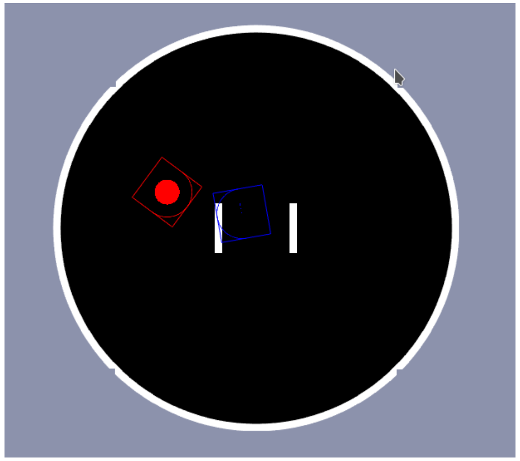
- Version 1 used a Raspberry Pi Zero
- Version 2 uses a NanoPi Duo (smaller, more powerful)
  - SOM Image Pre-Encoder
  - Learns by passively observing a human teacher
  - Drives by predicting the next steering angle
    - And loops it back in to complete the next input

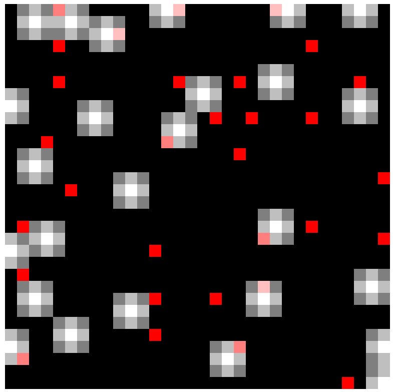


Micro-SDC vs original modified RC car SDC

Future Directions?

- Reinforcement learning is one of our main focuses now
- Also experimenting with a variant of OgmaNeo, Topo-OgmaNeo
- Exploits topology to better generalize
  - Based on multi-winner self-organizing maps
  - Looks a bit like grid cells
- FPGA version





End

- Thank you for your attention!