HierarchicalTemporalMemory.jl

A short δ from paper to code

Konstantinos Samaras-Tsakiris

https://github.com/oblynx/HierarchicalTemporalMemory.jl

HierarchicalTemporalMemory

an algorithmic model to understand the human brain

HierarchicalTemporalMemory.jl

Julia package for model research and time series prediction

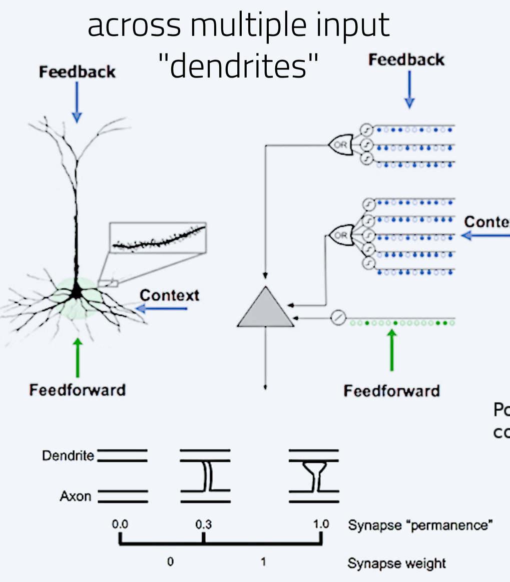
Make the HTM algorithms **accessible** with 475 lines of Julia

Experiments from the literature are reproduced to demonstrate correctness

A biologically *constrained* neural network...

HTM neuron

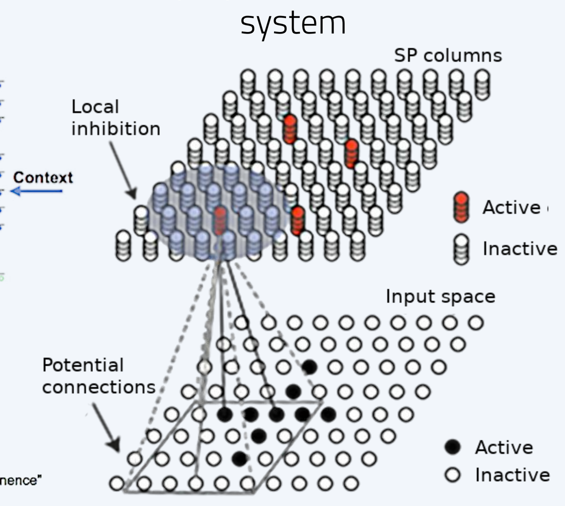
coincidence detection across multiple input "dendrites"



binary synapses

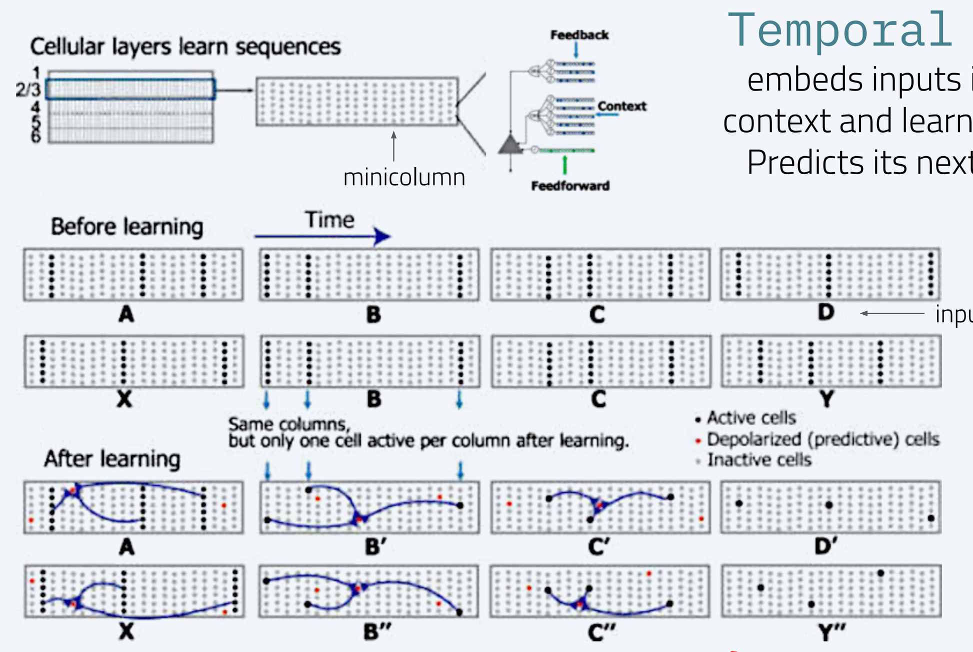
Spatial Pooler

normalizes feedforward input with useful properties for the HTM system



Temporal Memory

embeds inputs in temporal context and learns sequences. Predicts its next activation.



each input activates different sets of minicolumns (SP)

neurons in column encode the same symbol in different temporal context

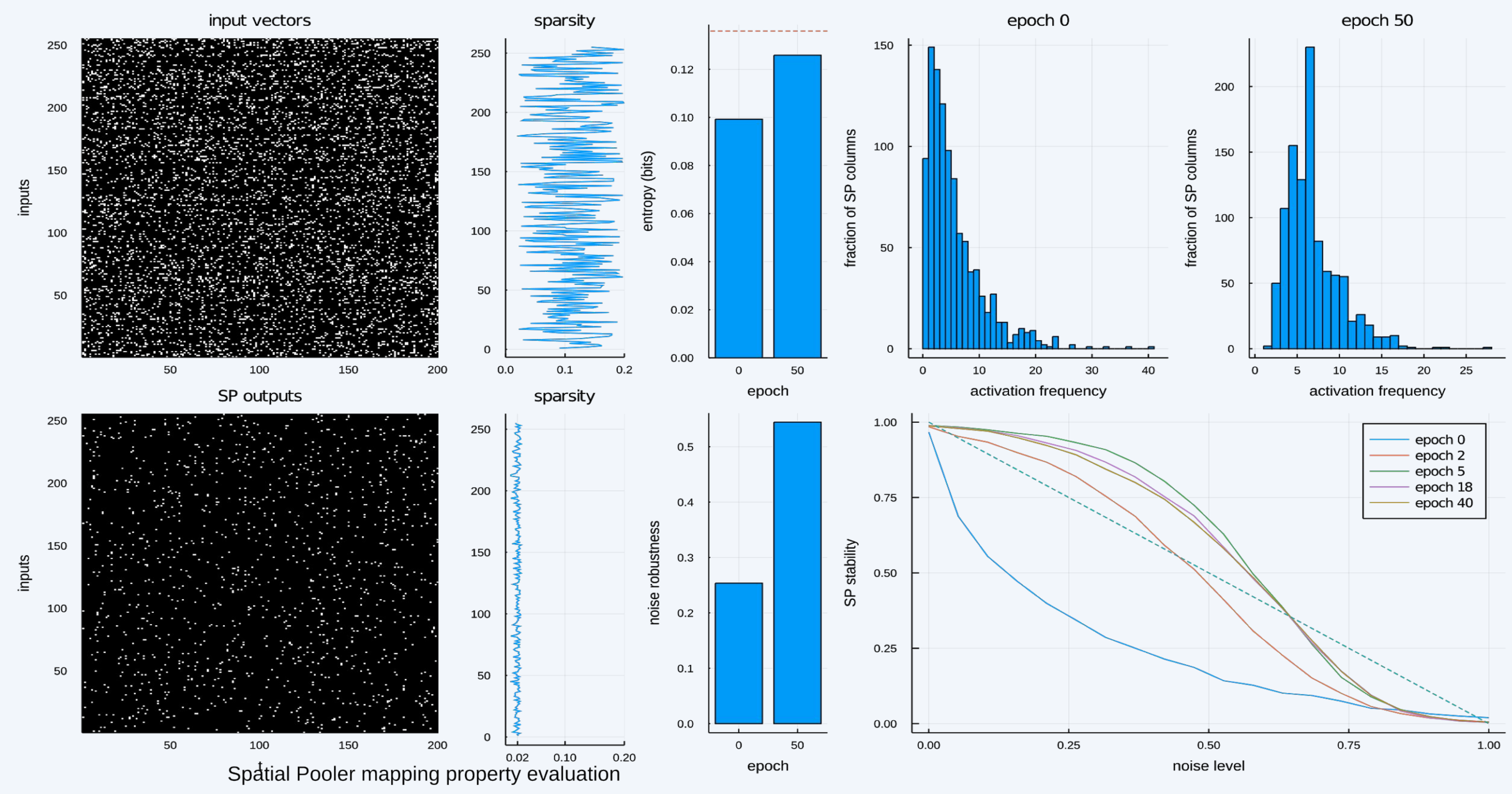
Figures from [2],[3]

Population code

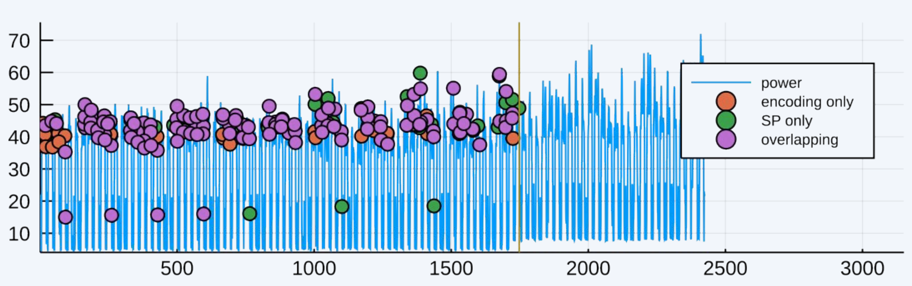
This *spatial pooler* is well-behaved...

Reproducing spatial pooler metrics from [1], figure 2.

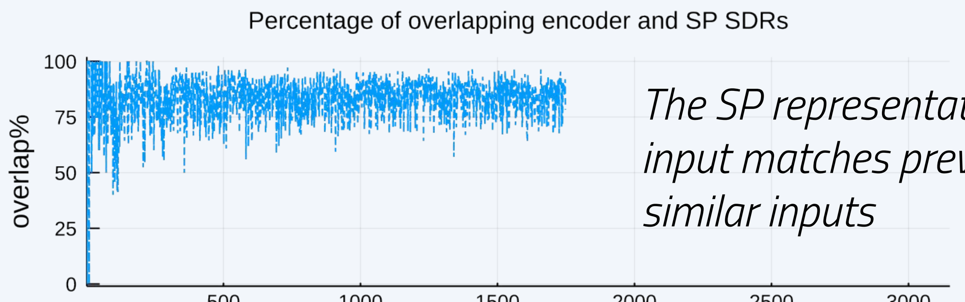
Properties: *sparsity control, high entropy, noise robustness, stable representations*



Spatial Pooler mapping property evaluation



Percentage of overlapping encoder and SP SDRs



A spatial pooler is trained on 250 random binary input vectors (left). The output sparsity is tightly controlled at 2%.

The SP representation of an input matches previous similar inputs

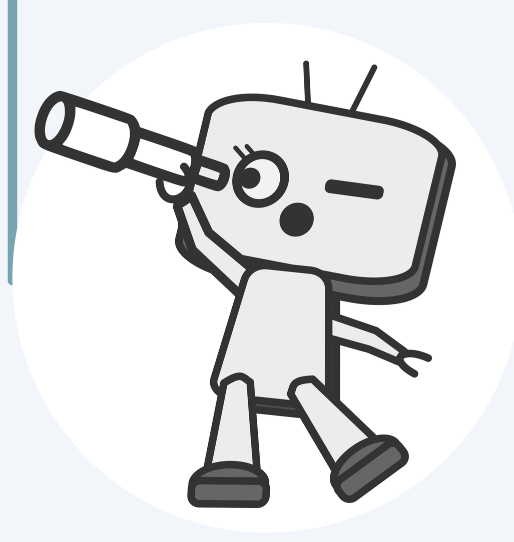
After training 50 times on the data all the minicolumns participate and the output remains stable under 30% bit flip on the input

Why all the fuss?

HTM aims first to model the brain and secondly to machine learning applications. This package similarly targets research on the model itself first and applications secondly.

HTM theory is not yet complete, lacking a definitive way to stabilize sequence representations and compose small models. Exactly for this reason, we believe that a concise and high level model can accelerate the research.

As a computational neuroscience research direction, the path should be explored between lower-level brain models (like [4]) that make fewer assumptions than HTM.



```
*(z::BitVector,W::SparseMatrixCSC)= Vector(z)*W
*(z::Adjoint{Bool,BitVector},W::SparseMatrixCSC)= Vector(z.parent)'W
*(W::Adjoint{<:Any,<:SparseMatrixCSC},z::BitVector)= W*Vector(z)
*(W::SparseMatrixCSC,z::BitVector)= W*Vector(z)
```

A short δ...

How do the proximal and distal synapses activate the neurons?

The definitions from [1],[3], made in Julia:

$$o_i = b_i \sum_j W_{ij} z_j$$
$$a_i = ((o_i \geq Z(V_i, k)) \wedge (o_i \geq \theta_{stim}))$$

```
o(z)= @> (b(sp) .* (W□(sp)'*(z|>vec)))
reshape(sz□□)
α(o)= o .+ tiebreaker(o,Z(o)) .>=
        @> Z(o) max.(θ_stimulus_activate)
z|> o|> α
```

$$a_{ij}^t = (j \in C^t) \wedge (\pi_{ij}^{t-1} = 1 \vee \sum_i \pi_{ij}^{t-1} = 0)$$

```
predicted(c,Π)= @percolumn(&,Π,c, k)     # (k × Nc)
burst(c,Π)= c .& .!@percolumn(any, Π, k) # Nc
activate(c,Π)= (predicted(c,Π) .| burst(c,Π)')|> vec

macro percolumn(f,a,b,k)
  esc(:( $f.(reshape($a,$k,:), $b') ))
end
macro percolumn(reduce,a,k)
  esc(:( $reduce(reshape($a,$k,:),dims=1)|> vec ))
end
```

A short δ...

For each active mini-column, increase the synapse to active inputs by p⁺, and decrease the synapse to inactive inputs by p⁻. Clip synapse at the boundaries of 0 and 1.

Dense/Sparse synapses

```
adapt!(D□::DenseSynapses, z,a, params)= begin
  @unpack p⁺,p⁻= params
  D□act= @view D□[:,a]
  actConn=   (D□act .> 0) .&   z
  inactConn= (D□act .> 0) .& .!z
  @inbounds D□act.= actConn .* (D□act .⊕ p⁺) .+
                    inactConn .* (D□act .⊖ p⁻)
end

adapt!(D□::SparseSynapses, z,a, params)= sparse_foreach(
    (scol,i)-> (@views adapt_synapses!(scol, z[i],
                .!z[i], params.p⁺,params.p⁻)), D□, a)

sparse_foreach(f, s::SparseMatrixCSC,columnIdx)=
  foreach(Truesof(columnIdx)) do c
    ci= nzrange(s,c)
    f((@view nonzeros(s)[ci]),rowvals(s)[ci])
  end
```

Custom iterator

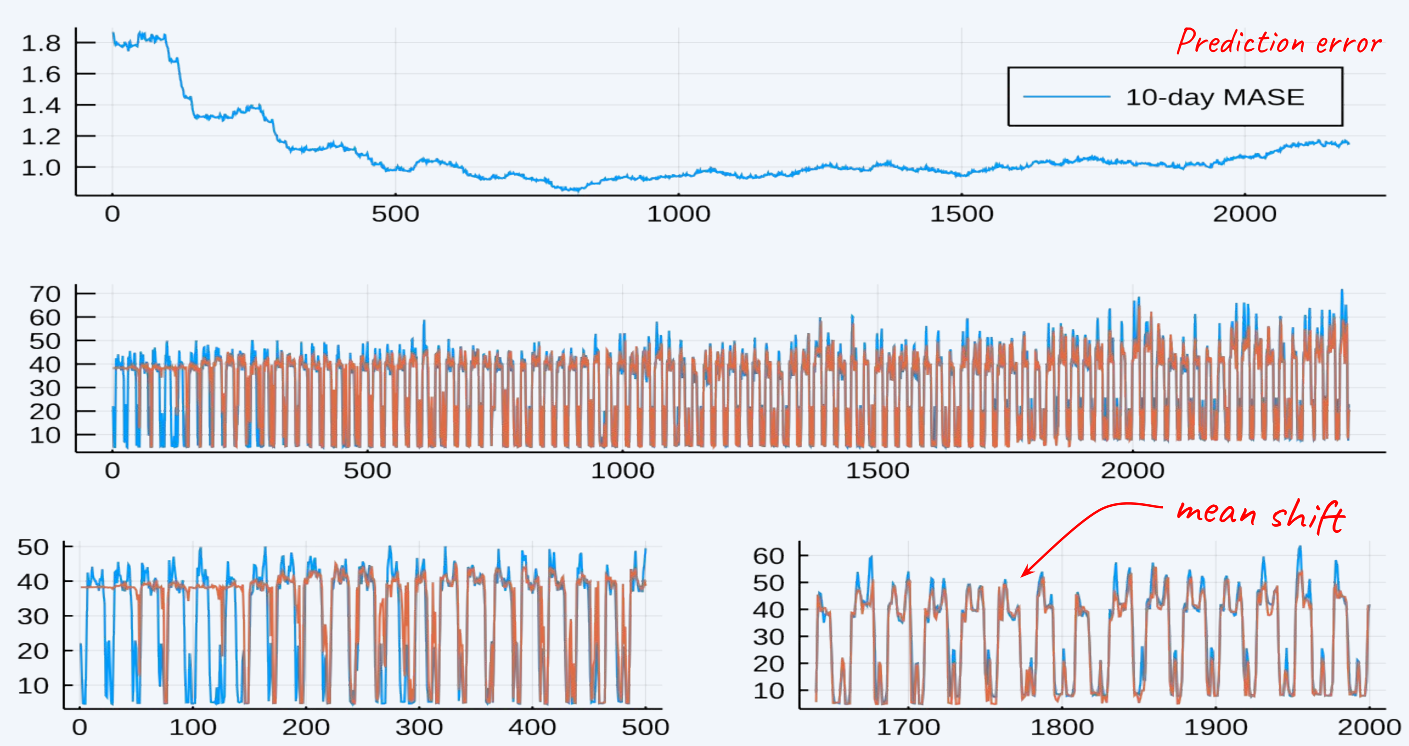
Unicode makes the equation look more natural.

@views and ***.Broadcasting*** make Spatial Pooler learning ×2.6 faster than the original

Multiple dispatch lets us extend Base.* to efficiently do SparseMatrix * BitVector

Predicting a building's power consumption...

Hourly power is encoded to binary input vectors, then the HTM pipeline predicts the power consumption 1 hour ahead.

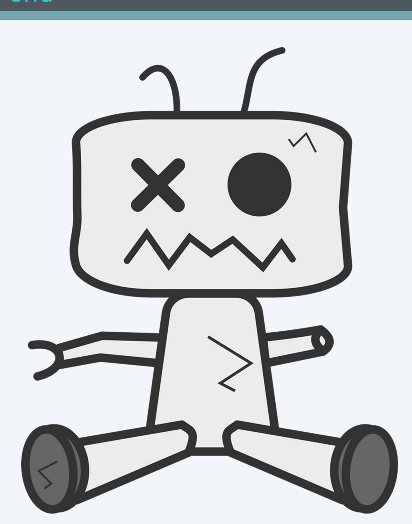


Characteristics

- daily and weekly seasonaility
- mean shift at t=1780

This prediction error is overall not competitive with other ways to model such a time series. However, HTM shows some interesting properties: it learns the periodic patterns without parameters tuned to the specific periods (eg. a window size) and follows the mean shift.

The TM if not biased towards the sequence it's currently predicting is sensitive to sequence noise. Applications typically use a slight variant of the TM algorithm, the "backtracking TM", which improves on this until temporal pooling is introduced.



Next steps...

- Reproduce time series prediction experiments
- Test & better docs
- Implement sensorimotor inference
- Explore temporal pooling and model composition

Contributions welcome!

References

[1] C Yuwei, A Subutai, H Jeff. "The HTM Spatial Pooler—A Neocortical Algorithm for Online Sparse Distributed Coding"

[2] J Hawkins, S Ahmad. "Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex"

[3] Y Cui, S Ahmad, J Hawkins. "Continuous Online Sequence Learning with an Unsupervised Neural Network Model"

[4] H Markram et al. "Reconstruction and Simulation of Neocortical Microcircuitry".