Towards a Deeper Understanding of the Hierarchical Temporal Memory Algorithm

DJ Passey

August 9, 2022

History of Hierarchical Temporal Memory

HTM, or Hierarchical Temporal Memory is an artificial intelligence architecture rooted in neuroscience and justified by biological plausibility. It is the outgrowth of funding by Jeff Hawkins, the former CEO of Palm Pilot and computational techniques developed by the research scientists Dileep George and Subatai Ahmad. In 2004, Jeff Hawkins published a book titled, "On Intelligence: How a New Understanding of the Brain Will Lead to the Creation of Truly Intelligent Machines". He founded the Redwood Institute for Neuroscience in 2005. The institute hired Dileep George, a computational scientist, and one year later in 2006, George gave a presentation titled "Hierarchical Temporal Memory: Theory and Applications" [5]. Three years later, in collaboration with their new company Numenta, George and Hawkins published a comprehensive theoretical paper: "Towards a Mathematical Theory of Cortical Micro-circuits" [6]. This paper describes a tree-like structure of nodes. Each node contains multiple Markov chains of which one is chosen to be "active". The state of the parent nodes is determined by which Markov chains are active in each child node. The model is optimized via Bayesian belief propagation so that probabilities of a particular Markov chain can be determined by the state of a node and it's siblings.

Dileep George left Numenta in 2010. This *YCombinator thread* suggests that George left due to a difference in technical direction. This *article* quotes George saying "HTM was an important effort, much like Poggio's foundational HMAX model, HTM had the right high level goals. But, when you dig into the algorithm level, you'll see that HTM implementations hadn't solved the problems of information representation in the hierarchy."

With the departure of George, leadership of the Numenta Research team switched to Subatai Ahmad and in 2016-2017, the company published a series of papers on their new HTM algorithm [2, 4, 3, 7].

This algorithm relies heavily on sparse distributed representations, a high dimensional method for encoding data with many useful properties [8]. The new hierarchical temporal memory algorithm divides the architecture into four components, an encoder component, a spacial pooler component, a temporal

memory component and a machine learning classifier. The first two components of the HTM architecture focus on translating raw data into an appropriate sparse high dimensional representation while the temporal memory component and classifier do the work of prediction.

This algorithm showed promise, out performing LSTM on time series prediction and showing excellent abilities at anomaly detection.

It was also a departure from the theoretical work completed by Dileep George in 2009. True to it's name, the temporal memory component no longer leverages a hierarchy of features at different granularity. Additionally, the temporal memory algorithm is optimized via a number of heuristic rules and does not rely on Bayesian optimization.

How does HTM compare to Deep Learning?

Time Series Prediction

There are a few papers compare HTM to deep learning algorithms directly [10, 9, 4, 2]. Of these, [10] is the most comprehensive. It suggests that it is the first paper to make such a comparison, and it shows definitively that with proper hyper parameter optimization, LSTM outperforms HTM, both in terms of accuracy *and speed*. It goes further than this, by using the same datasets as the original Numenta papers and showing that with proper hyper parameters, LSTM actually outperforms HTM [4, 2].

However, the future of HTM is not all bleak. The last section of [10] describes training the LSTM and HTM models built for the Numenta taxicab dataset (without changing any hyper parameters) on a non stationary dataset with a global trend. HTM was able to adapt to this dataset without any hyperparameter adjustment. With no hyper parameter optimization, LSTM failed. While this is more of a case study, it does illustrate that HTM is fundamentally different from LSTM and offers some comparative advantages.

Anomaly Detection

Struye et. al. claims that HTM has proved itself at anomaly detection [10]. Numenta has published a *a repository* for benchmarking real time streaming anomaly detection. It is assumed that the adaptive nature of HTM makes it well suited for this sort of problem. However futher investigation calls this claim into question. This is well illustrated by the title of a recent paper: "Current Time Series Anomaly Detection Benchmarks are Flawed and are Creating the Illusion of Progress" [11]. The paper examines many anomaly detection time-series datasets including the Numenta benchmark and finds that they contain mislabeled data, many trivial problems, too many anomalies and a run to failure bias (most anomalies occur at the end). The paper uses one of the Numenta benchmark datasets as an illustration of a poor choice. The paper goes into

depth on the use of anomalies in the Numenta taxi driver dataset and shows the existence of multiple false positives and false negatives in the labels.

Data hygine notwithstanding, HTM was beaten by another brain inspired algorithm called Adaptive Resonance on it's own benchmark [1].

Why Study Hierarchical Temporal Memory?

The question remains, if HTM is less accurate than other techniques, what is the value of studying HTM? The answer is as follows.

While HTM technically loses to LSTM in terms of speed and accuracy, it is not far behind. This is remarkable considering that the optimization step of HTM relies on heuristic rules that are not well understood. Additionally, LSTM optimization has been supported and explored by a massive army of machine learning engineers while HTM has seen no such investment.

In our work studying the HTM algorithm, we focused on uncovering the essential mechanisms of ther temporal memory algorithm: What is it *exactly* that leads to the success of the this approach?

In focusing on this question we uncovered theoretical concepts that were *fundamentally different* than the theoretical tools applied by standard deep learning approaches.

Our hope is that by reducing the key mechanisms of HTM to a theoretical framework, we can leverage existing theory from sparse distributed memory, branching processes and other tools to converge on clearer optimization pathways, lighter weight models and the potential to augment current machine learning techniques with fresh tools. Additionally, HTM appears to have a comparative advantage when it comes to adaptability and resistance to certain types of noise [2, 3, 4]. It is possible that a deeper investigation of the relevant computational mechanisms will provide a pathway to refine this comparative advantage.

Results

The main results of this investigation were as follows. First, a Julia implementation of the Nupic temporal memory algorithm, an exploration to determine if the main mechanism behind the temporal memory algorithm was random projection, and a study on higher order representation and synapse creation.

All of the resulting code is stored in the repository *here*. See the README file for a more granular description of each phase of investigation.

In this report we will focus on the most intriguing finding from our investigation. This is the idea that the temporal memory algorithm is essentially a Markov chain, but it is vanshingly sparse and near infinite dimensional. Instead of each state representing one event, multiple states correspond to the same event occuring in different contexts. For example, the classical Markov chain example is weather.

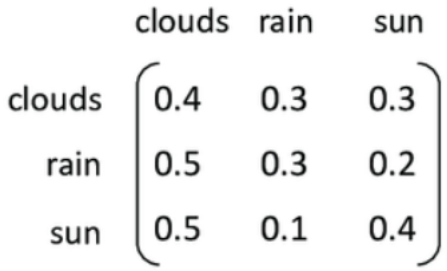

Figure 1: A simple Markov chain modeling weather as transitions between three states. Weather states that transition between each other with varying probability. It is immediately clear that this is too simple for weather, and that more that the current state of the world is needed to predict the future. For example, a third day of sun after two previous sunny days may be more likely than a second day of sun after one sunny day.

Our examination of the temporal memory algorithm led us to believe that it makes multiple representations for each state in order to capture context. We can extend our weather example to help explain this idea. Let's expand the dimensions of our Markov transition matrix so that there are many states corresponding to clouds, rain and sun. We will call these clouds_1, clouds_2, ..., clouds_n, rain_1, rain_2, ..., rain_n, and sun_1, sun_2, ..., sun_n. Each of these states correspond to clouds, rain or sun appearing in a different context. For example sun_1 might be used to represent a sunny data following one previous sunny day and sun_2 might be used to represent a sunny day preceded by two sunny days.

However, for real problems, where the number of states is large, this model would take up a large amount of memory and might memorize the training data and not be able to generalize.

How does the temporal memory algorithm accomplish this?

One can conceptualize the temporal memory model as a grid of cells. There are many columns and several rows. The original Numenta paper uses 2048 columns and 32 rows. Each sequence is encoded as a random subset of columns. (Let's use 16 for this example.)

Initially, the algorithm starts with no synapses associating the cells in the grid with each other. To train the model, it is exposed to the training sequence by activating columns corresponding to the encoding of the characters in the sequence.

When a column is activated, one cell from the column is chosen as a "winner cell". This is the cell that is the closest to activating in response to the previous

$$
\begin{array}{c}
 \\
\text{clouds}_1 \\
\text{clouds}_2 \\
\vdots \\
\text{rain}_1 \\
\text{rain}_2 \\
\vdots \\
\text{sun}_1 \\
\text{sun}_2 \\
\vdots
\end{array}
\begin{array}{c}
\begin{array}{ccccccccc}
\text{clouds}_1 & \text{clouds}_2 & \cdots & \text{rain}_1 & \text{rain}_2 & \cdots & \text{sun}_1 & \text{sun}_2 & \cdots
\end{array} \\
\begin{pmatrix}
0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\
0 & 0 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \\
0 & 1 & \cdots & 0 & 0 & \cdots & 0 & 0 & \cdots \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots \\
0 & 0 & \cdots & 0 & 1 & \cdots & 0 & 0 & \cdots \\
1 & 0 & \cdots & 0 & 0 & \cdots & 1 & 0 & \cdots \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots & \ddots
\end{pmatrix}
\end{array}
$$

Figure 2: A high dimensional sparse Markov chain with multiple representations for each state. The idea here is that clouds_1 represents the presence of clouds today, when they were preceded by rain yesterday. The state clouds_2 might represent clouds today preceded by two days of rain. We hypothesize that the temporal memory algorithm leverages a strategy like this one for time series prediction.

input, but initially there are no synapses and the cell is chosen randomly.

In the temporal memory algorithm, a column is considered "predictive" if *any* of the cells in that column are in a predictive state. Therefore, there are many different ways that a character can be correctly predicted by the algorithm. However, the specific collection of cells or representation used to predict the column carry with them information about past states, because that collection of cells activates only in response to specific previous patterns.

Synapse creation in the temporal memory algorithm

The original temporal memory algorithm paper did not have a strong focus on synapse formation, and states that synapses are initialized randomly with a fixed density [7]. However, the actual Numenta algorithm initializes synapses on the fly. The mechanism for creating synapses is intricate and the consequences of the choices Numenta made are not clear.

Temporal Memory Synapse Creation
Temporal memory algorithm receives input.
for *each column* **do**
    Check if the column is in a predictive state;
    Check if the prediction is correct;
    if *False positive column* **then**
        Punish synapse permanences ;
        No synapse synapse formation occurs;
        Return;
    end
    if *False negative column* **then**
        Identify the cell that was closest to activating, randomly if no cells were close;
        if *None of the cells segments were close to activating* **then**
            Make one new segment on that cell and form synapses to all previous winner cells ;
        end
        else
            pick the segment closest to activating ;
            add synapses to some previous winner cells but don't break existing connections to previously active cells. ;
        end
        Return;
    end
    if *True negative column* **then**
        Reduce permanence on segments that were close to activating;
        No changes to synapses;
        Return;
    end
    if *True Positive column* **then**
        Increase permanence values;
        Form additional synapses but don't undo existing active synapses;
        Return;
    end
end
Algorithm 1: Synapse creation in the temporal memory algorithm.

Notes on the algorithm

A key piece of this process to notice is that new synapses appear only in the event of a false negative. Reenforcement happens during true positive, but this assumes that the needed synapses were in place already.

The creation of synapses only during false negatives represents an "as needed" approach. This way, the algorithm builds synapses when it is not able to make

a correct prediction.

However, another detail is that if synapses exist, and they would have been active if their permanence values were higher, the algorithm does not add new synapses and instead waits until their permanence values are increased to the proper value. This prevents the algorithm from growing more synapses than necessary.

Open questions

The temporal memory algorithm, in connection with the encoder, spatial pooler and a classifier is able to solve deep learning problems like the MNIST dataset of hand written digits. It is clear that there is some potential.

We seek a better understanding of how the temporal memory algorithm is able to produce correct predictions.

We have identified several questions of interest.

-
- To what extend does the high dimensional markov chain explain the success of the temporal memory algorithm?
- What are the limitations of the algorithm. How many different representations can is realistically remember? How long can it predict before a cycle appears? To what extend can it capture branching processes?
- What is the impact of initial conditions. Are stop and start initializations needed?
- Given a clearer theoretical understanding of the algorithm, how can it be optimized for better performance?

We look forward to gaining a deeper understanding and believe this could lead to advances that would benefit many other areas of machine learning.

References

[1] Leonardo Enzo Brito da Silva, Islam Elnabarawy, and Donald C. Wunsch. A survey of adaptive resonance theory neural network models for engineering applications. *Neural Networks*, 120:167–203, 2019. special Issue in Honor of the 80th Birthday of Stephen Grossberg.

[2] Yuwei Cui, Subutai Ahmad, and Jeff Hawkins. Continuous online sequence learning with an unsupervised neural network model. *Neural Computation*, 28(11):2474–2504, nov 2016.

[3] Yuwei Cui, Subutai Ahmad, and Jeff Hawkins. The htm spatial pooler—a neocortical algorithm for online sparse distributed coding. *Frontiers in Computational Neuroscience*, 11, 2017.

[4] Yuwei Cui, Chetan Surpur, Subutai Ahmad, and Jeff Hawkins. A comparative study of htm and other neural network models for online sequence learning with streaming data. In *2016 International Joint Conference on Neural Networks*, pages 1530–1538, 07 2016.

[5] Dileep George. Hierarchical temporal memory: Theory and applications. In *2006 International Conference of the IEEE Engineering in Medicine and Biology Society*, volume Supplement, pages 6643–6643, 2006.

[6] Dileep George and Jeff Hawkins. Towards a mathematical theory of cortical micro-circuits. *PLoS computational biology*, 5:e1000532, 10 2009.

[7] Jeff Hawkins and Subutai Ahmad. Why neurons have thousands of synapses, a theory of sequence memory in neocortex. *Frontiers in Neural Circuits*, 10, 2016.

[8] Pentti Kanerva. Hyperdimensional computing: An introduction to computing in distributed representation with high-dimensional random vectors. *Cognitive Computation*, 1(2):139–159, 2009.

[9] Jonathan Mackenzie, John F. Roddick, and Rocco Zito. An evaluation of htm and lstm for short-term arterial traffic flow prediction. *IEEE Transactions on Intelligent Transportation Systems*, 20(5):1847–1857, 2019.

[10] Jakob Struye and Steven Latré. Hierarchical temporal memory and recurrent neural networks for time series prediction: An empirical validation and reduction to multilayer perceptrons. *Neurocomputing*, 396:291–301, 2020.

[11] Renjie Wu and Eamonn Keogh. Current time series anomaly detection benchmarks are flawed and are creating the illusion of progress. *IEEE Transactions on Knowledge and Data Engineering*, pages 1–1, 2021.