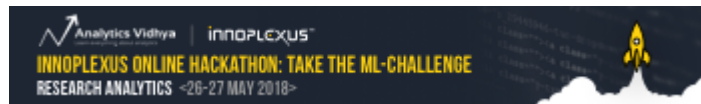


f (<https://www.facebook.com/AnalyticsVidhya>)

t (<https://twitter.com/analyticsvidhya>)

g+ (<https://plus.google.com/+Analyticsvidhya/posts>)

in (<https://www.linkedin.com/groups/Analytics-Vidhya-Learn-everything-about-5057165>)



(https://datahack.analyticsvidhya.com/contest/innoplexus-hiring-hackathon/?utm_source=AVhometop)

Home (<https://www.analyticsvidhya.com/>) > Data Science (<https://www.analyticsvidhya.com/blog/category/data-science/>) > An ...

An Alternative to Deep Learning? Guide to Hierarchical Temporal Memory (HTM) for Unsupervised Learning

DATA SCIENCE ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/DATA-SCIENCE/](https://www.analyticsvidhya.com/blog/category/data-science/)) DEEP LEARNING

([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/DEEP-LEARNING/](https://www.analyticsvidhya.com/blog/category/deep-learning/)) MACHINE LEARNING

([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/CATEGORY/MACHINE-LEARNING/](https://www.analyticsvidhya.com/blog/category/machine-learning/))

[https://www.facebook.com/sharer.php?u=https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?](https://www.facebook.com/sharer.php?u=https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?%20Hierarchical%20Temporal%20Memory%20(HTM)%20for%20Unsupervised%20Learning)

[https://twitter.com/home?](https://twitter.com/home?%20Hierarchical%20Temporal%20Memory%20(HTM)%20for%20Unsupervised%20Learning)

[https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?](https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?%20Hierarchical%20Temporal%20Memory%20(HTM)%20for%20Unsupervised%20Learning)

[https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?](https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?%20Hierarchical%20Temporal%20Memory%20(HTM)%20for%20Unsupervised%20Learning+https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?%20Hierarchical%20Temporal%20Memory%20(HTM)%20for%20Unsupervised%20Learning)

[https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?](https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?%20Hierarchical%20Temporal%20Memory%20(HTM)%20for%20Unsupervised%20Learning)

[https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?](https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?%20Hierarchical%20Temporal%20Memory%20(HTM)%20for%20Unsupervised%20Learning)

[https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?](https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-ning/&t=An%20Alternative%20to%20Deep%20Learning?%20Hierarchical%20Temporal%20Memory%20(HTM)%20for%20Unsupervised%20Learning)

Introduction

Deep learning has proved its supremacy in the world of supervised learning, where we clearly define the tasks that need to be accomplished. But, when it comes to unsupervised learning, research using deep learning has either stalled or not even gotten off the ground!

There are a few areas of intelligence which our brain executes flawlessly, but we still do not understand how it does so. Because we don't have an answer to the "how", we have not made a lot of progress in these areas.



If you liked my previous article (<https://www.analyticsvidhya.com/blog/2018/04/replicating-human-memory-structures-in-neural-networks-to-create-precise-nlu-algorithms/>) on the functioning of the human brain to create machine learning algorithms that solve complex real world problems, you will enjoy this introductory article on Hierarchical Temporal Memory (HTM). I believe this is the closest we have reached to replicating the underlying principles of the human brain.

In this article, we will first look at the areas where deep learning is yet to penetrate. Then we will look at the difference between deep learning and HTM before deep diving into the concept of HTM, its workings and applications.

Let's get into it.

Table of Contents

1. Progress areas of deep learning

2. Is deep learning really brain-like learning?
3. Crash course on Neocortex
4. How is HTM different from deep learning?
5. Applications of HTM implemented & commercially tested
6. Working of Hierarchical Temporal Memory (HTM)
7. Simple python implementation of HTM
8. So what's next for Numenta?

Progress areas of deep learning

Following is a list of the few areas where Deep Learning has a long way to go yet:

1. **Working on unsupervised data models** – Humans generally perform actions based on supervised models running in their brain
2. **Working with live streaming data** – Humans can work with live feeding information which we get from our sensory organs
3. **Combining information from many individual data sources** – Even though we might feel the apple, see the apple, taste the apple or smell the apple differently, our brain combines all this information into a single idea
4. **One shot learning** – Humans can learn with extremely less number of data points. If you see/taste a new fruit, say Kiwi, just a couple of times, you will start distinguishing this new fruit from others with high accuracy
5. **Contextual Understanding** – Humans combine a lot of past information/context on every new experience. For instance, if you taste a banana and it is very bitter, you immediately spit it out. How do you know it has gone bad without even tasting this fruit? Obviously, from your previous experience eating it
6. **Working with high velocity data** – Imagine you are talking to a friend and some object is coming towards you. Fortunately, you look in that direction and quickly change your position to save yourself. When you look in a different direction, the entire image that is sent to the brain changes, but the brain is still able to adapt quickly

Deep learning has made some progress in each of the above six directions, but we are far from the end state. Following is what we have achieved through deep learning in each of the six fields:

1. **Working on unsupervised data models** – We now have auto-encoders and RBMs that can work on unsupervised problems, but generally they won't return the final action but some embedding to process further
2. **Working with live streaming data** – Deep learning cannot work with live streaming data yet

3. **Combining information from many individual data sources** – We have seen some initial success in working with deep learning models on combined data sources, but we are yet to see some strong results
4. **One shot learning** – Few embedding based models like facial identification can do one shot learning but we only have a handful of such success stories
5. **Contextual Understanding** – Long Short Term Memory (LSTM) and Gated Recurrent Unit (GRU) are models that can work with contextual data but they can only carry context based on supervised tasks and not on unsupervised tasks
6. **Working with high velocity data** – GPUs have squeezed the processing time for deep learning models, but we are currently talking about extremely small number of neurons when compared to the human brain. The amount of time a brain-like model will take is still enormous

You might have already realized that an Artificial Neural Network does not provide a single solution for all the above skills. And yet our brain uses a common learning algorithm that can take care of all the above attributes. Which brings me to a fundamental question.

Is deep learning really brain-like learning?

The human brain has always been the ultimate motivation in the field of deep learning. But we have created a very mathematical formulation to replicate brain functions in form of ANNs and it has not changed over a period of decades. Are we saying that our brain works on such complex mathematical functions without even realizing it? We are pretty sure that our brain does not learn through backpropagation of gradients (which is the basic fundamental principle of deep learning/ANN).

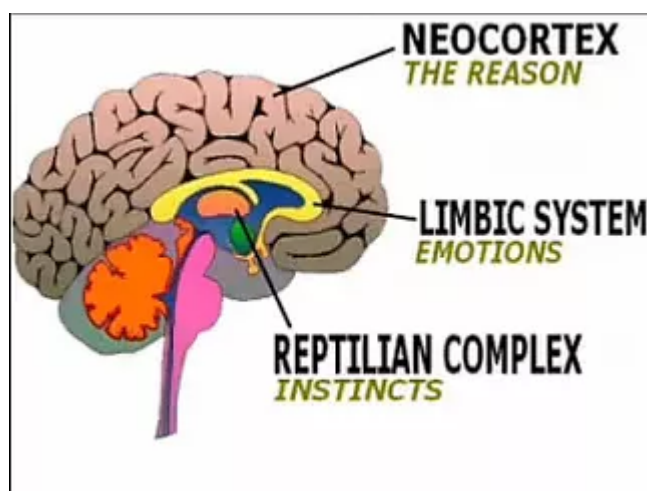
Recurrent Neural Network (RNN) architectures are the closest we have reached to our brain in the deep learning space. You can read one of my previous articles (<https://www.analyticsvidhya.com/blog/2018/04/replicating-human-memory-structures-in-neural-networks-to-create-precise-nlu-algorithms/>) on RNNs where I compare them to the human brain. But RNNs are supervised learning models, unlike the brain. So if deep learning is far from replicating the human brain structure, is there anyone trying to replicate brain structure and have they found any success yet?

The answer is yes! Numenta, a company founded in 2005, is solely dedicated to replicating the functioning of the human brain and using it in the space of artificial intelligence. Numenta was founded by Jeff Hawkins, the man behind Palm Pilot. Numenta has created a framework called

Hierarchical Temporal Memory (HTM) that replicates the functioning of the Neocortex, the component of our brain responsible for the real intelligence in humans. I will talk about HTM and its practical applications in this article, but first let's do a crash course on Neocortex.

Crash course on Neocortex

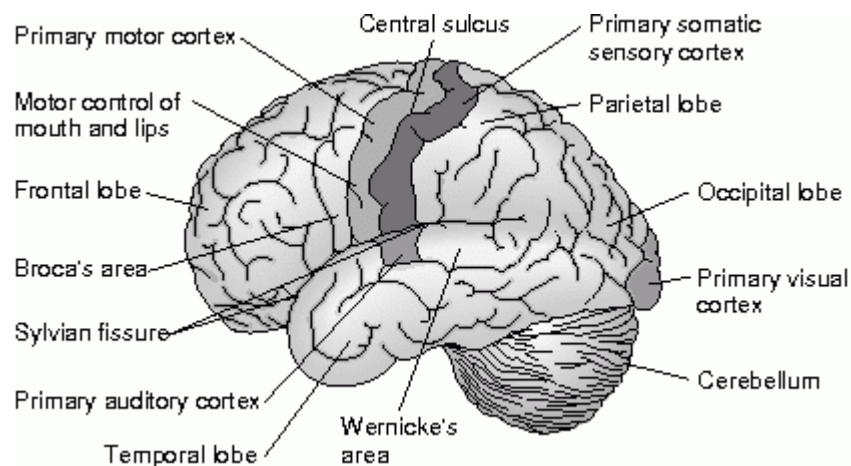
Our brain primarily has three parts – Neocortex, Limbic system and Reptilian complex.



Simplistic view of a brain

The limbic system supports most of the emotions linked functions, including behavior, motivation and emotional state. Reptilian complex is for all the survival instincts like eating, sleeping etc. Neocortex is the that part of the brain which gives us power to reason and other higher order brain functions like perception, cognition, spatial reasoning, language and generation of motor command.

Neocortex is a mammalian development and is almost like a dinner napkin squeezed in our skull. In general, whenever we talk about "brain" or "intelligence" in colloquial terms, we are almost always referring to the Neocortex. If we look at the detailed structure of the Neocortex (below diagram), you will see many sections responsible for different tasks.

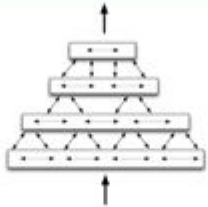
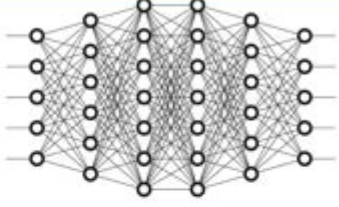


An interesting fact about Neocortex is that the cellular structure throughout all these regions is almost the same, whether it be from the visual processing region or the audio processing region. This finding is extremely important as this means that the brain is trying to solve similar problems to process any kind of sensory data – visual, audio etc. These regions are logically related to each other in a hierarchical structure. We will refer to this hierarchical structure later when we cover HTM.

The sensory data is represented as simple ideas in the lower level and the idea gets more abstract in the higher level. A parallel to this process in the deep learning space – the initial layers in neural networks detect simple ideas like edges, intermediate layers detect shapes, and final layers identify objects.

Enough of biology, let's now get down to business and talk about HTM models. The best way to initialize your brain with what you are about to learn is by contrasting against a known concept – Deep Learning.

How is HTM different from deep learning?

| Attribute | HTM (Hierarchical Temporal Memory) | Deep Learning |
|--------------------------|---|---|
| |  |  |
| Premise | Biological | Mathematical |
| Learning Mechanism | Hebbian Learning | Back Propagation |
| Learning type | Unsupervised | Supervised |
| Learning batches | Online learning | Batch wise learning |
| Neuron cell state | Active/Inactive/Predictive | Active/Inactive |
| Batch size need to learn | Very small data is sufficient | Required huge data volume |

As you can see in the image above, the differences between these two approaches are significant. If you have used deep/machine learning before, you will know how hard it is to imagine how a model can work without finding gradients. Hebbian learning is one of the oldest learning algorithms and works on an extremely simple principle – synapse between two neurons is strengthened when the neurons on either side of the synapse (input and output) have highly correlated outputs.

Before diving into how HTM works, I will give you a flavor of where we can use HTM to solve real world problems. This will give you the motivation to learn more about this novel technique.

Applications of HTM implemented & commercially tested

First, let's try to nail down a few pointers on "when can we expect HTM to outperform other learning techniques?":

1. **Is the input data temporal?** An easy way to find whether the data is temporal or not is to randomly shuffle the data and see if the semantics of the data changed

2. **Does the data have a high velocity?** High velocity data will have input coming very frequently and will need online learning rather than batch learning
3. **Do you have multiple data sources creating inputs?** These data sources can have completely different structures, for example – image data and audio data used together
4. **Do you need the model to learn continuously?**
5. **Is your data unlabeled?** As it is very expensive to label data, we have a lot of problems that are unsolved because of unlabeled data

If the answer to all the above questions is “yes”, HTM is the way to go. Anomaly detection is one such task as it needs action in real time and it is an unsupervised model. Here is the general framework for anomaly detection:

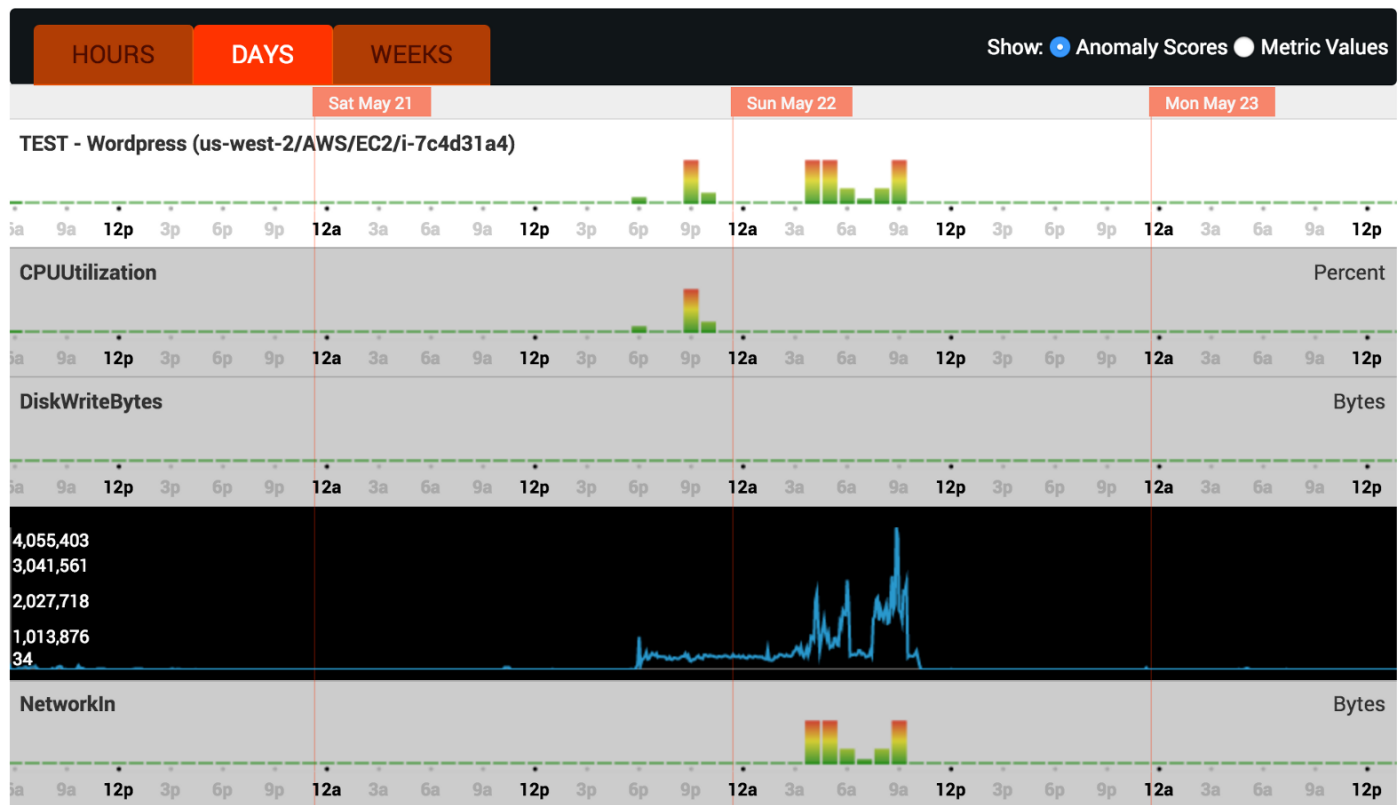


Figure 1 - Process for Modeling Movements and Identifying Anomalies

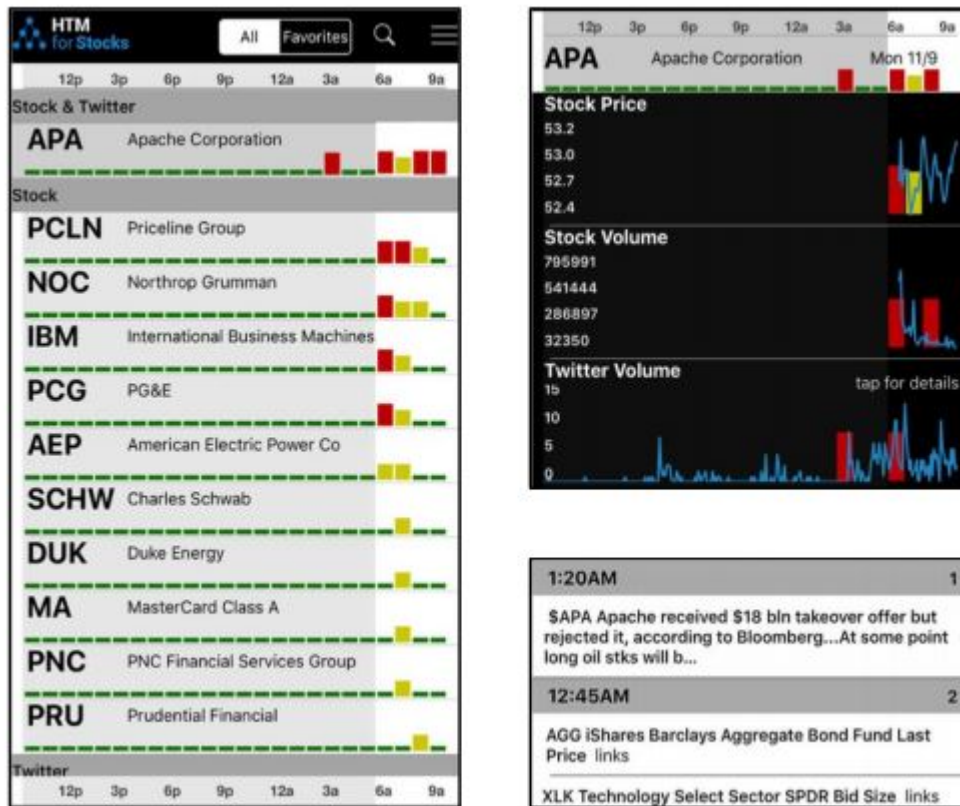
Below are few of the use cases that have already been commercially tested:

1. **Server Anomalies detection using Grok:** Grok was the first commercially available product from Numenta that is used for anomaly detection. One of the important use cases is anomalies in server monitoring. Servers generate a lot of metrics like CPU usage, GPU usage, etc. at a high frequency. Grok creates an SDR for individual metrics and feeds it into individual HTM models. The HTM model then sends a trigger to the user whenever it sees an anomaly for any of the metrics. This application has

already been tested with major cloud service providers and has proved to be extremely useful.



2. **Stock volume anomalies:** This application tracks the stock price and Twitter data of publicly traded companies and sends an alert whenever anomalies are detected to take timely buy/sell action. Here is a look at the interface (taken from Numenta's whitepaper):



3. **Rogue human behavior:** Companies spend a lot of money on data security. Rogue human behavior is an application based on HTM that can help automate discovering these data security issues.

Applications of such a system include:

- detection of unusual employee access to intellectual property and internal systems
- identifying abnormal financial trading activities or asset allocations by individual traders
- sending alerts when employee behaviors or actions fall outside of typical patterns
- detect the installation, activation, or usage of unapproved software
- sending alerts when employee computers or devices are used by unauthorized individuals. The system sorts the employee name with an anomaly score as follows (image taken from Numenta's whitepaper):



4. **Natural Language prediction:** Deep learning has been very successful with NLP applications. Numenta claims that its HTM model can provide similar or better accuracy. Currently, they have tested simple NLP tasks like language models and have found the results to be promising. Numenta has partnered with Cortical.io to create word embedding and language models that have already shown strong results. Later in this article, I will show you how can you use the API interface of Cortical.io.
5. **Geospatial tracking:** Geospatial tracking takes the input of various locations (latitude or longitude) and the velocity of a vehicle or even humans. We convert these metrics into SDR and feed them into the HTM model. The model will send you a trigger if it sees any anomaly in the data. This tracking can be used by any transportation or cab service to track all their vehicles. Here is an example that is quoted in Numenta's whitepaper:

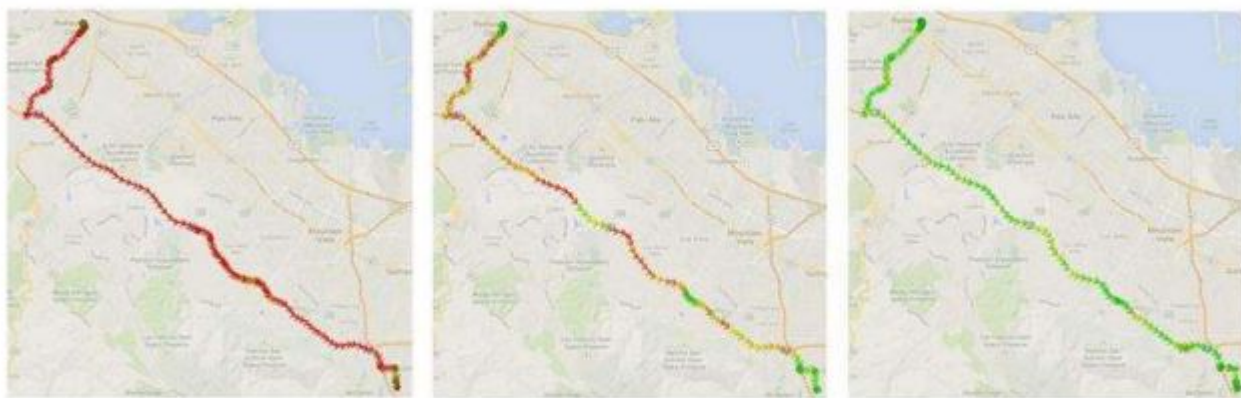


Figure 3 - Learning a New Route: (a) First trip on new route; (b) Second trip; (c) Third Trip, showing traffic anomalies

The algorithm initially tries to learn a route that is shown in all red. Gradually it learns each latitude, longitude and velocity with a particular sequence. Any change in route or speed or direction will be tracked as an anomaly and reported back.

Working of Hierarchical Temporal Memory (HTM)

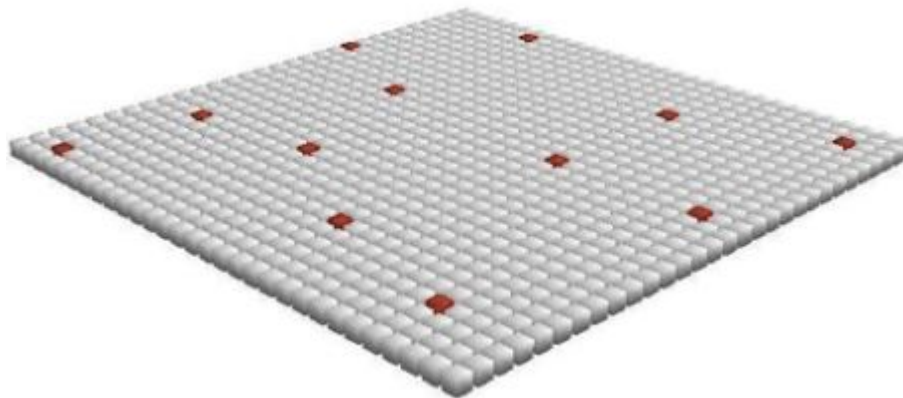
HTM works as follows (don't get scared):

*Input temporal data generated from various data sources is **semantically encoded** as a sparse array called as **sparse distributed representation (SDR)**. This encoded array goes through a processing called **spatial pooling** to normalize/standardize the input data from various sources into a sparse output vector or mini-columns (column of pyramidal neurons) of definitive size and fixed sparsity. The learning of this spatial pooling is done through **Hebbian learning** with **boosting** of prolonged inactive cells. The spatial pooling retains the context of the input data by an algorithm called **temporal memory**.*

For people who did not understand the above language at all, don't worry! I will break it down. The key words have been highlighted in bold and need to be understood first to completely grasp HTM.

Concept 1 : Sparse Distributed Representation

SDR is simply an array of 0's and 1's. If you take a snapshot of neurons in the brain, it is highly likely that you will only see less than 2% neurons in an active state. SDR is a mathematical representation of these sparse signals which will likely have less than 2% ones. We represent SDR as follows:



SDR has a few important properties :

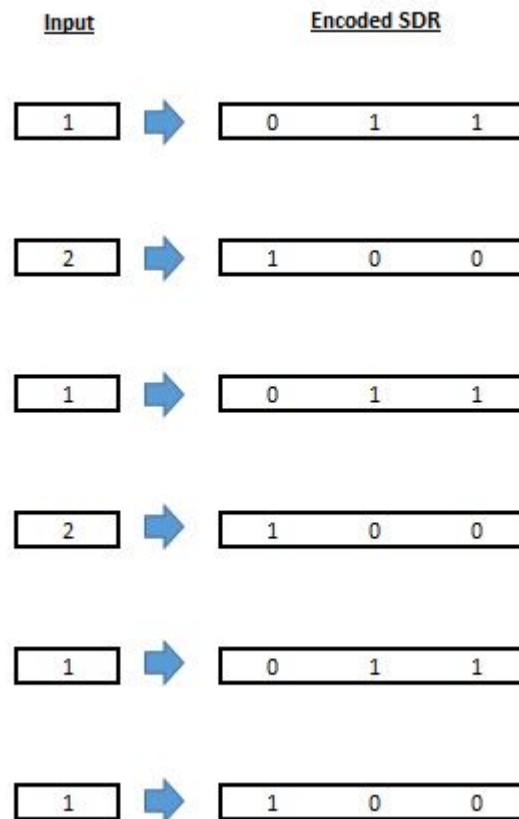
1. SDR is extremely noise resistant. Even if you introduce noise in magnitude of say 33%, highly likely that the SDR will only match with very like objects. You will realize this even for human memory. Try to remember a name of a professor from your graduation college and try remembering his/her face. It is possible that you might not remember him/her face clearly. Even then, if I show you 100 random picture which include your professor's image, you will definitely locate your professor with high accuracy.
2. SDR can be sub-sampled without losing much information. Say, your SDR has 20 ones and 10000 zeros. Even if you remove 5 zeros from this SDR, we can still use the sub-sampled SDR to match with new SDRs. You will realize this even for human memory. Even if we forget a lot of properties of a fruit which we tasted 10 years back, we can still classify this fruit accurately based on all other properties that we do remember.

Concept 2 : Semantic Encoding

We use an encoding engine to take input from an input source and create an SDR. We need to make sure that the encoding algorithm gives us similar SDR for similar objects. This concept is very similar to embedding in the deep learning space. A lot of pre-built encoders are already available online that include numeric encoding, datetime encoding, English word encoding, etc.

Let's say we have a simple sequence – 1,2,1,2,1,1. The sixth element breaks the sequence, i.e., it should be 2 but the actual value is 1. We will try to understand how HTM pinpoints this anomaly. The first step is semantic encoding. For the purpose of this article, I will use a dense vector as encoded SDR. In real

world scenarios, these encoded vectors are extremely sparse.



How are encoders developed?

Even though we have a lot of built-in encoders, you might need to create your own encoder for specific problems. I will try to give you a brief introduction of how word encoders are developed.

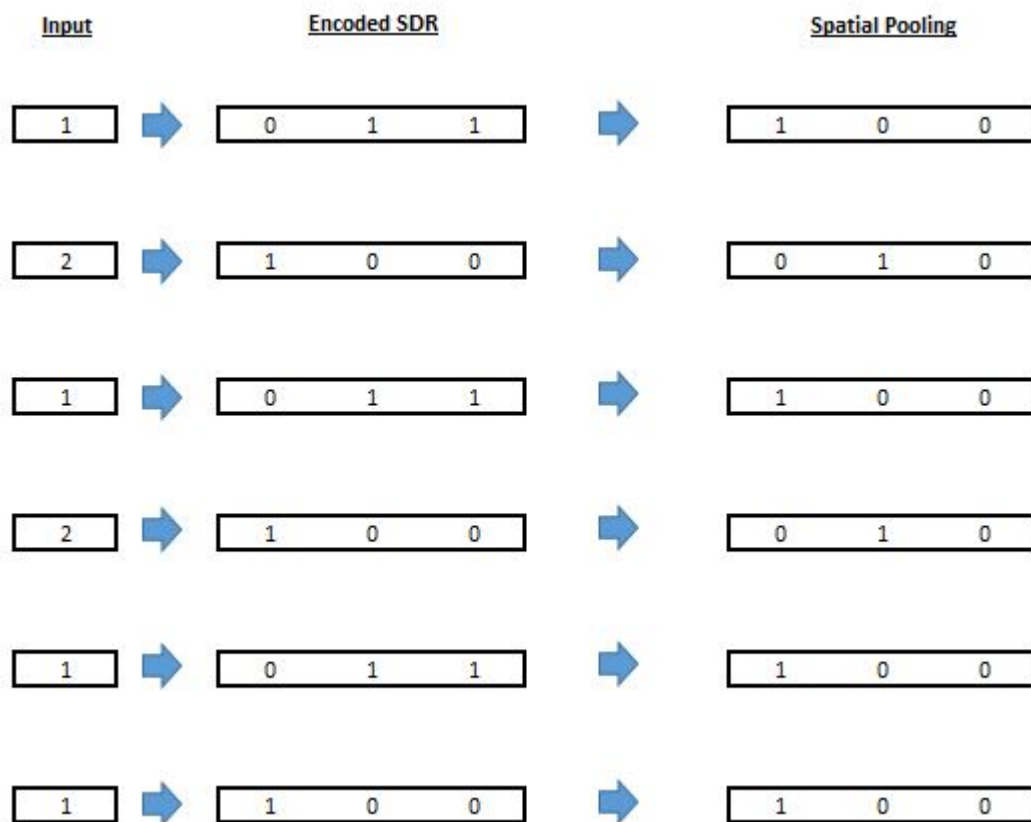
- **Step 1** – We choose a set of documents that we will use to find semantics of words
- **Step 2** – We will clean the documents and slice each document into snippets. We will then cluster these snippets so that similar snippets are kept together
- **Step 3** – We will now represent each snippet as a node in a SDR
- **Step 4** – Now we pick up individual (target) words and activate all the nodes (documents) in the SDR that contain our target word. This will create a word fingerprint in form of SDR
- **Step 5** – We will repeat the above four steps to get word fingerprints for all the words we are interested in

Concept 3: Spatial Pooling

Spatial pooling is the process of converting the encoded SDR into a sparse array complying with two basic principles:

- Make sure the sparsity of the output array is constant at all times, even if the input array is a very sparse SDR or a not so sparse SDR
- Make sure the overlap or semantic nature of the input is maintained

So the overlap of both input and output SDR of two similar objects need to be high. Let's try to understand this with our example.



The input vector had a sparsity varying from 33% to 67%, but the spatial pooling made sure the sparsity of the output array is 33%. Also the semantics of the two possible inputs in the series are completely different from each other, and the same was maintained in the output vector. How do we use this framework to pinpoint anomalies? We will come back to this question once we cover temporal memory.

Concept 4: Hebbian Learning

Learning in HTM is based on a very simple principle. The synapse between the active column in the spatially pooled output array, and active cells in encoded sequence, is strengthened. The synapses between the active column in the spatially pooled output array, and inactive cells in encoded input, is weakened. This process is repeated again and again to learn patterns.

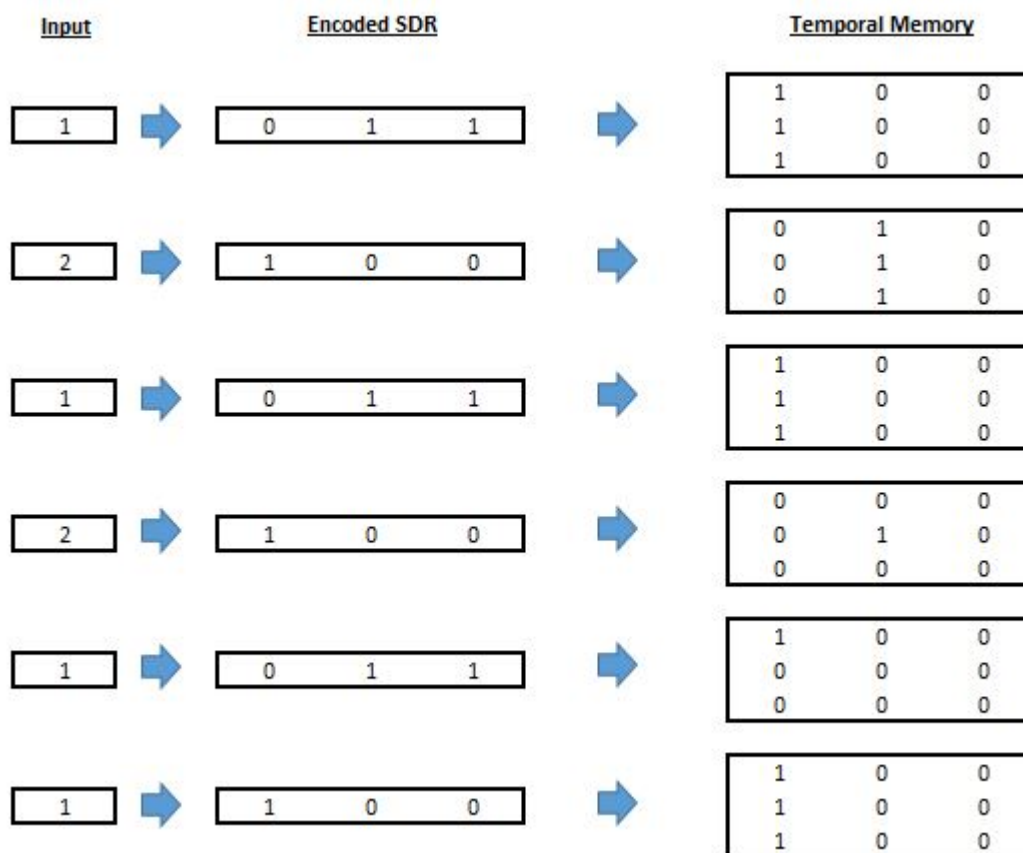
Concept 5 : Boosting

Most of the spatial pooling processes will create exceptionally strong columns in the output array which will suppress many columns from contributing at all. In such cases, we can multiply the strength of these weak columns to encoded sequence by a boosting factor. This process of boosting makes sure that we are using a high capacity of the spatially pooled output.

Concept 6 : Temporal Memory

Spatial pooling maintains the context of the input sequence by a method called temporal memory. The concept of temporal memory is based on the fact that each neuron not only gets information from lower level neurons, but also gets contextual information from neurons at the same level. In the spatial pooling section, we had shown each column in the output vector by a single number. However, each column in the output column is comprised of multiple cells that can individually be in active, inactive, or predictive state.

This mechanism might be a bit complex, so let's go back to our example. Instead of a single number per column in the spatial pooling step, I will now show all cells in the columns of the output vector.



Now let me break down the above figure for you.

At step 1, our HTM model gets an input "1" for the first time which activates the first column of the output sequence. Because none of the cells in the first column were in predictive mode, **we say column 1 goes "burst"** and we assign an active value to each of the cells in column 1. We will come back on how a cell is placed to a predictive state.

At step 2, our HTM model gets an input "2" again for the first time in the context of "1", and hence, none of its cells are in predictive state so column 2 goes burst.

Same thing happens at step 3, as the model is seeing "1" in context of "2" for the first time. Note that our model has seen "1" before, but it has never seen "1" in context of "2".

At step 4, something interesting happens. Our HTM model has seen "2" in context of "1" before, so it tries to make a prediction. *(Here I have ignored the cascading context complexity to keep this article simple. Cascading context means "2" in context of "1" in context of "2" and so on. For now, just assume our model has a 2 degree memory that it is able to remember one last step).*

The method it uses to make this prediction is as follows: It checks with all the cells that are currently active, i.e., column 1, to tell which of the 9 cells do they predict will turn active in the next time step. Say, the synapse between (2,2) cell is stronger with column 1 among (2,1),(2,2) and (2,3), so column 1 unanimously replies (2,2). Now (2,2) is put into a predictive state before consuming our next element of the sequence. Once our next element arrives, which is actually a "2", the prediction goes right and none of the columns burst this time.

At step 5, again none of the columns burst and only (1,1) is put in active state as (1,1) had a strong synapse with (2,2).

At step 6, the HTM model is expecting a value of "2" but it gets "1". Hence, our first column goes burst and our anomaly is detected in this sequence.

The entire algorithm can be overwhelming without visual simulations. So I strongly recommend that you check out the free online videos (<https://www.youtube.com/watch?v=XMBori4qgwc>) published by Numenta that have some very cool simulations of the process I mentioned above.

Simple python implementation of HTM

Numenta Platform for Intelligent Computing (*NuPIC*) is a machine intelligence platform that implements the HTM learning algorithms. We have NuPIC as one of the importable libraries in Python. *The library is not supported by Anaconda yet.* **A simple implementation of HTM can be found on this link**

(<http://nbviewer.jupyter.org/github/numenta/nupic/blob/master/examples/NuPIC%20Walkthro>)

This is a very well documented code by Numenta . The code starts with Encoding, where you can see how numbers/date/categories can be encoded in HTMs. It then gives examples of spatial pooling and temporal memory with a working example of a predictive model. The code is self-explanatory so I will skip this part to avoid replication of content.

Trying out API implementation of HTM

One cool way to experience what HTM is capable of doing is to use an API provided by [cortical.io](http://www.cortical.io/) (<http://www.cortical.io/>). To use this API, go to this link (<http://api.cortical.io/>). Here, I will show you a simple example of how can you use the API. When you go to the link, you will see the following screen:



You can try any of the tabs as each implementation gives very clear instructions of what kind of input it is expecting. I will show you one tab to help you get going – “Term”. Once you click on the “Term” tab, you will see the instructions of using this tab and the output format:

GET /terms/similar_terms Get the similar terms of a given term

Implementation Notes

This method returns a listing of similar terms for the specified input **term**.
If any valid **context_id** is specified the method returns similar terms for the term in this specific context.
If the **start_index** parameter for this method is not specified, the default of 0 will be assumed.
If the **max_results** parameter for this method is not specified, then the default value of 10 will be assumed. For this method the maximum number of results per page is limited to 1000.
If the **context_id** parameter is not specified, this method returns a list of similar terms over all contexts.
The **pos_type** parameter enables filtering of the results by parts of speech (one of: NOUN, VERB, ADJECTIVE). If this parameter is unspecified, (null), no filtering will occur.
More detailed information on the use of this method can be found in the [tutorial](#).

Response Class

Model Model Schema

```
Term {  
  term (string, optional): The term as a string.,  
  fingerprint (Fingerprint, optional): The Fingerprint of this term.,  
  pos_types (array[string], optional): The pos types of the term.,  
  score (number, optional): The score of this term.,  
  df (number, optional): The df value of this term.  
}
```

All we need to enter is a term, and the API will return terms that are synonyms or associated (you can choose either) to the term. You can also choose to get the fingerprints of all these words. Here are my inputs to get synonyms of “cricket”:

| Parameters | | | | |
|-----------------|-------------------|--|----------------|-----------|
| Parameter | Value | Description | Parameter Type | Data Type |
| retina_name | en_synonymous ▼ | The retina name | query | string |
| term | cricket | A term in the retina | query | string |
| context_id | | The identifier of a context (optional) | query | integer |
| start_index | 0 | The start-index for pagination (optional) | query | integer |
| max_results | 5 | Max results per page (optional) | query | integer |
| pos_type | NOUN ▼ | Part of speech (optional) | query | string |
| get_fingerprint | false (default) ▼ | Configure if the fingerprint should be returned as part of the results | query | boolean |

Here is a sample of the response output I get:

| Response Body |
|---|
| <pre>{ "term": "wickets", "df": 0.0007344738855576646, "score": 79, "pos_types": ["NOUN"], "fingerprint": { "positions": [] } }, { "term": "cricketers", "df": 0.0001348679659959262, "score": 76, "pos_types": ["NOUN"], "fingerprint": { "positions": [] } }, { "term": "cricket", "df": 0.0001348679659959262, "score": 76, "pos_types": ["NOUN"], "fingerprint": { "positions": [] } }, { "term": "bowling", "df": 0.0001348679659959262, "score": 76, "pos_types": ["NOUN"], "fingerprint": { "positions": [] } }, { "term": "wicket", "df": 0.0001348679659959262, "score": 76, "pos_types": ["NOUN"], "fingerprint": { "positions": [] } }</pre> |

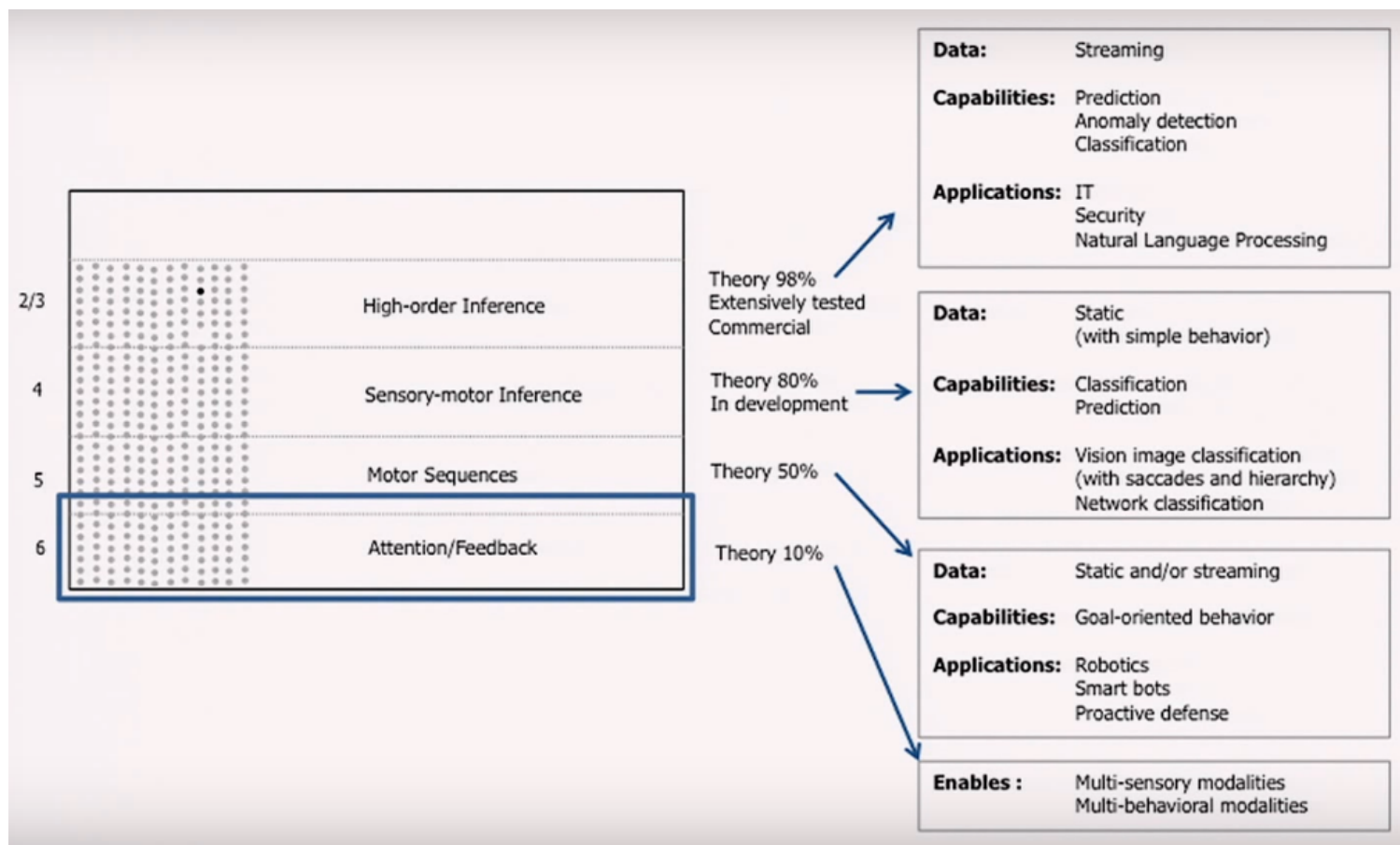
The words are sorted by the similarity score. The top 5 words that were found similar to "cricket" were cricket, wickets, cricketers, bowling, wicket. We can also choose to get fingerprints of each of these words. Let's pull the fingerprints of "wicket" and "wickets" and see if they are more similar to each other or to the word "cricket".

| | Wicket (A) | Wickets(B) | Similar flag (A intersection B) |
|---------------------|------------|------------|---------------------------------|
| 1st active | 6 | 593 | 0 |
| 2nd active | 7 | 2604 | 0 |
| 3rd active | 901 | 3271 | 1 |
| 4th active | 1016 | 3272 | 1 |
| 5th active | 2477 | 4156 | 0 |
| | ***** | ***** | ***** |
| Total active | 164 | 164 | 96 |

In the above table, column 2 and 3 are fingerprints (indices) of the word "wicket" and "wickets". The last column is when the active index of "wickets" is also found in "wicket". The overlap score comes out to be 96, which is far better than the best match of any word with the word "cricket" (obviously except the word itself). Hence, this API does a good job of mapping these words semantically as SDR.

So what's next for Numenta?

Here is a snapshot of the slide from Jeff Hawkins, showing the pipeline of research:



The layers are showing the hierarchy of the cortical tissue. Most of the current research efforts have been focused on the high-order inference layer. Everything covered in this article was related to the high-order inference layer. The second layer in the diagram (labeled as 4) mainly works on sensory-motor inference. This is an important function of the brain where it collaborates between the signals from sensory organs and motor cells to create concepts.

For instance, if you move your eyes, the image they capture changes rapidly. If the brain doesn't know what was the cause of this drastic change (which only motor cells can tell), it will fail to simplify the environment around us. However, if we combine the signals from sensory organs and motor cells, the brain can map a stable understanding of the surroundings. If we can master this skill of the brain, we can apply this skill on complex problems like image classification where we have to move our eyes across the picture to understand it in its entirety. This task is similar to what we do in Convolutional Neural Networks.

The third layer in the diagram is the capability of the brain that makes it goal oriented, which is something similar to reinforcement learning. With this new skill you can work on complex robotics problems. The last layer is the most complex part where we are talking about putting the entire

hierarchy of concept understanding in a place that can be used for multi-sensory modalities that can combine, say, a visual data with an audio data.

If I want to put the above paragraph in simple deep learning terms,


- layer 1 in the picture is like a getting simple neural network functionality with HTM (obviously with the added benefits of HTM)
- layer 2 in the picture is like a getting convolution neural network functionality with HTM
- layer 3 in the picture is like a getting reinforcement learning functionality with HTM
- layer 4 in the picture is like getting multiple CNNs to work with reinforcement learning and HTM


End Notes


So who wins between ANN/deep learning and HTM? As of now they are solving very different problems. Deep learning is very specialized for classification problems and HTM are specialized for real time anomaly detection problems. HTM still needs a lot of research to solve problems like image classification etc. that deep learning can solve pretty easily. However, the underlying theory behind HTM looks promising and you should keep this field of research in your radar.


If you have any ideas, suggestions or feedback regarding the article, do let me know in the comments below!


Share this:


 (<https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-unsupervised-learning/?share=linkedin&nb=1>)

 (<https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-unsupervised-learning/?share=facebook&nb=1>)

 (<https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-unsupervised-learning/?share=google-plus-1&nb=1>)

 (<https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-unsupervised-learning/?share=twitter&nb=1>)

 (<https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-unsupervised-learning/?share=pocket&nb=1>)

 (<https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-unsupervised-learning/?share=reddit&nb=1>)

TAGS: HIERARCHICAL TEMPORAL MEMORY ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/HIERARCHICAL-TEMPORAL-MEMORY/](https://www.analyticsvidhya.com/blog/tag/hierarchical-temporal-memory/)), HTM ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/HTM/](https://www.analyticsvidhya.com/blog/tag/htm/)), NUMENTA ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/NUMENTA/](https://www.analyticsvidhya.com/blog/tag/numenta/)), REAL TIME ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/REAL-TIME/](https://www.analyticsvidhya.com/blog/tag/real-time/)), UNSUPERVISED LEARNING ([HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/TAG/UNSUPERVISED-LEARNING/](https://www.analyticsvidhya.com/blog/tag/unsupervised-learning/))

Next Article

Qure.ai uses Machine Learning to Detect Brain Anomalies in less than 10 seconds

(<https://www.analyticsvidhya.com/blog/2018/05/quire-ai-uses-machine-learning-detect-brain-anomalies-less-10-seconds/>)

Previous Article

Move Over Photoshop – This Python Script Works like Magic on Low Light Photos (GitHub link included)

(<https://www.analyticsvidhya.com/blog/2018/05/move-over-photoshop-this-python-library-works-like-magic-on-low-light-photos-github-link-included/>)



(<https://www.analyticsvidhya.com/blog/author/tavish1/>)

Author

Tavish Srivastava (<https://www.analyticsvidhya.com/blog/author/tavish1/>)

Tavish is an IIT post graduate, a results-driven analytics professional and a motivated leader with 7+ years of experience in data science industry. He has led various high performing data scientists teams in financial domain. His work range from creating high level business strategy for customer engagement and acquisition to developing Next-Gen cognitive Deep/Machine Learning capabilities aligned to these high level strategies for multiple domains including Retail Banking, Credit Cards and Insurance. Tavish is fascinated by the idea of artificial intelligence inspired by human intelligence and enjoys every discussion, theory or even movie related to this idea.

ONE COMMENT

Niket says
REPLY (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2018/05/ALTERNATIVE-DEEP-LEARNING-HIERARCHICAL-TEMPORAL-MEMORY-HTM-UNSUPERVISED-LEARNING/?REPLYTOCOM=153242#RESPOND) AT 1:31 PM (HTTPS://WWW.ANALYTICSVIDHYA.COM/BLOG/2018/05/ALTERNATIVE-DEEP-LEARNING-HIERARCHICAL-TEMPORAL-MEMORY-HTM-UNSUPERVISED-LEARNING/#COMMENT-153242)

Very nice work,Tavish.

LEAVE A REPLY

Your email address will not be published.

Comment


Name (required)





Email (required)

Website

SUBMIT COMMENT

TOP ANALYTICS VIDHYA USERS

| Rank | Name | Points |
|------|--|--------|
| 1 |  vopani (https://datahack.analyticsvidhya.com/user/profile/Rohan_Rao) | 8714 |

| | | | |
|---|---|--|------|
| 2 |  | SRK (https://datahack.analyticsvidhya.com/user/profile/SRK) | 8287 |
| 3 |  | aayushmnit (https://datahack.analyticsvidhya.com/user/profile/aayushmnit) | 7439 |
| 4 |  | mark12 (https://datahack.analyticsvidhya.com/user/profile/mark12) | 6269 |
| 5 |  | sonny (https://datahack.analyticsvidhya.com/user/profile/sonny) | 5937 |

[More Rankings \(http://datahack.analyticsvidhya.com/users\)](http://datahack.analyticsvidhya.com/users)



(<http://www.greatlearning.education/analytics/>?)

utm_source=avm&utm_medium=avmbanner&utm_campaign=pgpba+bda)

POPULAR POSTS

- A Complete Tutorial to Learn Data Science with Python from Scratch
(<https://www.analyticsvidhya.com/blog/2016/01/complete-tutorial-learn-data-science-python-scratch-2/>)

- Essentials of Machine Learning Algorithms (with Python and R Codes)
(<https://www.analyticsvidhya.com/blog/2017/09/common-machine-learning-algorithms/>)
- 7 Types of Regression Techniques you should know!
(<https://www.analyticsvidhya.com/blog/2015/08/comprehensive-guide-regression/>)
- Understanding Support Vector Machine algorithm from examples (along with code)
(<https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/>)
- 6 Easy Steps to Learn Naive Bayes Algorithm (with codes in Python and R)
(<https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/>)
- A comprehensive beginner's guide to create a Time Series Forecast (with Codes in Python)
(<https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/>)
- A Complete Tutorial on Tree Based Modeling from Scratch (in R & Python)
(<https://www.analyticsvidhya.com/blog/2016/04/complete-tutorial-tree-based-modeling-scratch-in-python/>)
- A Complete Tutorial on Time Series Modeling in R
(<https://www.analyticsvidhya.com/blog/2015/12/complete-tutorial-time-series-modeling/>)

RECENT POSTS



(<https://www.analyticsvidhya.com/blog/2018/05/launching-student-datafest-2018-largest-student-machine-learning-festival/>)

Launching Student DataFest 2018 – The Largest Student Machine Learning Festival

(<https://www.analyticsvidhya.com/blog/2018/05/launching-student-datafest-2018-largest-student-machine-learning-festival/>)

KUNAL JAIN , MAY 15, 2018



(<https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-unsupervised-learning/>)

An Alternative to Deep Learning? Guide to Hierarchical Temporal Memory (HTM) for Unsupervised Learning

(<https://www.analyticsvidhya.com/blog/2018/05/alternative-deep-learning-hierarchical-temporal-memory-htm-unsupervised-learning/>)

TAVISH SRIVASTAVA , MAY 14, 2018