Hierarchical Temporal Memory
including
HTM Cortical Learning Algorithms

Jeff Hawkins (Numenta)

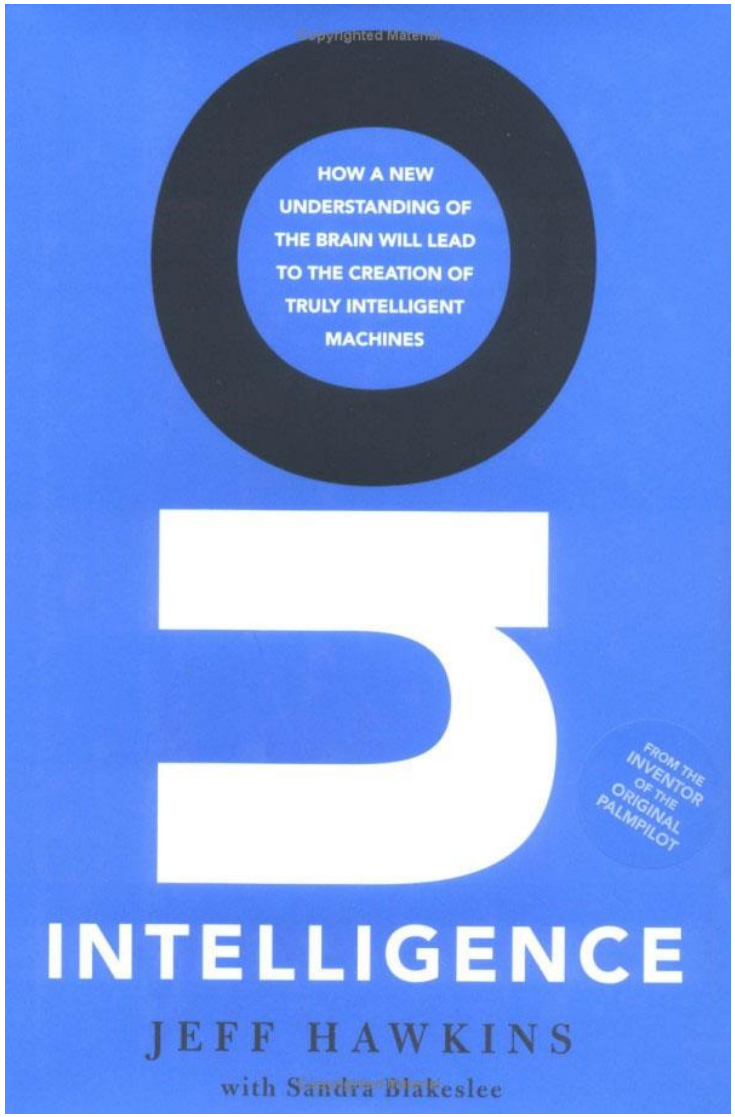
Presenter: Sungjoon Choi

Department of Electrical and Computer Engineering
Seoul National University

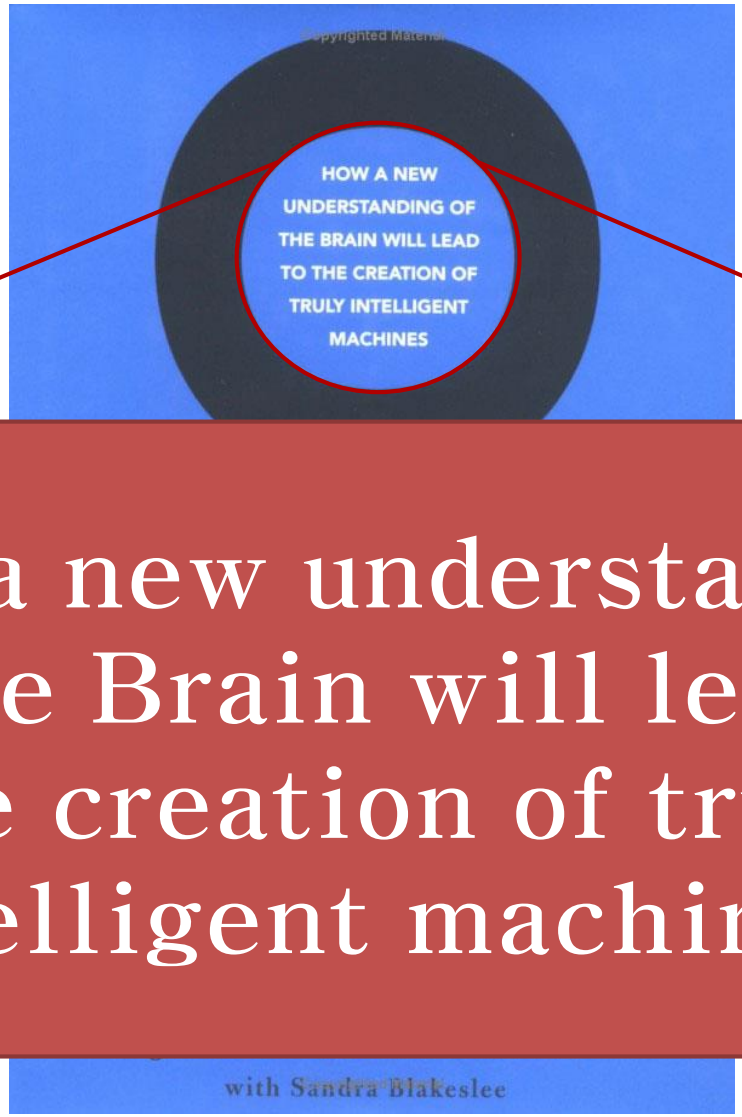
CONTENTS

- On Intelligence
- Truly Intelligent Machine
- Human Brain and Neocortex
- Hierarchical Temporal Memory
- Spatial Pooler implementation
- Temporal Pooler implementation
- Discussion

On Intelligence



On Intelligence



How a new understanding of the Brain will lead to the creation of truly intelligent machines.

TEN ML PEOPLE YOU NEED TO KNOW

4. Jeff Hawkins

Up until the nineties, Hawkin's name was primarily associated with the Palm Pilot, a device of his invention. In 2002, he dedicated himself to neuroscience and artificial learning processes focused around human cortical function, and established the Redwood Center for Theoretical Neuroscience. In 2005, he published a seminal work on the memory-prediction framework theory of the brain, entitled "On Intelligence: How a New Understanding of the Brain will Lead to the Creation of Truly Intelligent Machines". 2005 was also the year that Hawkins, along with former Palm Pilot CEO Donna Dublinsky and Dileep George founded Numenta, a company dedicated to theorising about brain function, and finding algorithms which can implement these theories in AI. Chief among their findings are the algorithmic frameworks for Hierarchical Temporal Memory and Fixed-Sparsity Distributed Representations.

helicopter developed through reinforcement learning, one of the most sophisticated of its kind in the world). Ng also founded the Google "Brain" project in 2011, and is currently working on the Baidu Brain project (expected to be the largest neural network in the world upon its completion).

10 living scientists to have been elected to all three national academies (Medicine, Science and Engineering). He currently an advisor to Obama's $100 million BRAIN initiative, which develops new tools for mapping neural circuits.

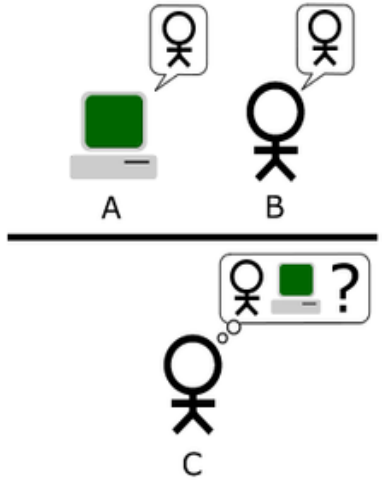
http://dataconomy.com/10-machine-learning-experts-you-need-to-know/

TRULY INTELLIGENT MACHINE (1/5)

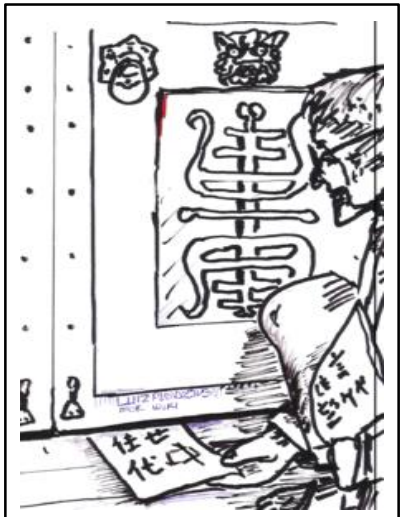
- **What is Intelligence?**
  - A person or animal that is intelligent has the ability to think, understand, and learn things quickly and well.
  - But still **ambiguous**, especially for programmers like us!

- **What about Turing Test?**
  - A test of a machine's ability to exhibit intelligent behavior equivalent to, or **indistinguishable** from, that of actual human.
  - Turing machine is particularly useful in explaining a CPU.

- **What do you think about Chinese room argument?**
  - It argues that just passing Turing Test is **not enough** by pointing out that the person inside the Chinese room cannot **understand** Chinese.



Turing test


Chinese room

http://en.wikipedia.org/wiki/Turing_test
http://en.wikipedia.org/wiki/Chinese_room

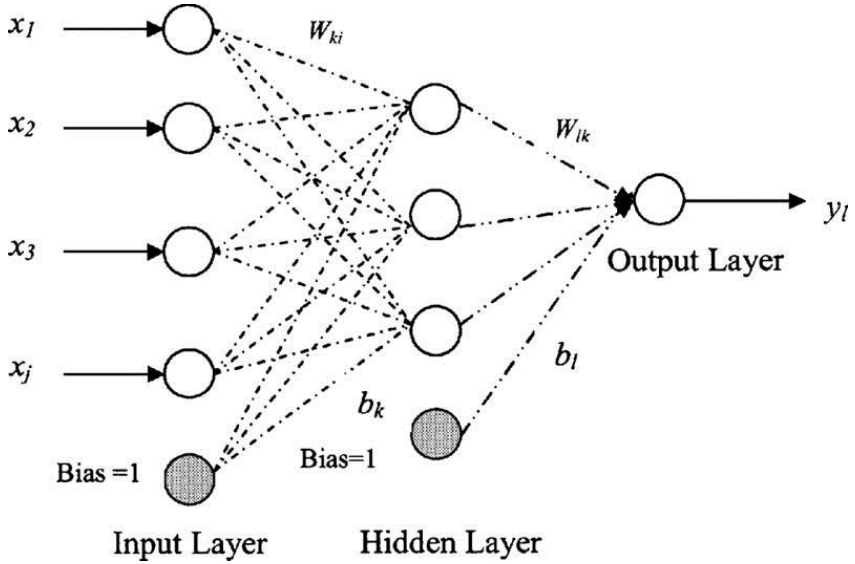
TRULY INTELLIGENT MACHINE (2/5)

- The argument between “Turing test” and “Chinese room” comes from the definition of Intelligence. While “Turing test” focused on **behavior**, “Chinese room argument” focused on **understanding**.

- In this presentation, I will follow the Jeff Hawkins’ definition of intelligence which is to **mimic the property of neocortex**. This does not mean that I completely agree this idea.

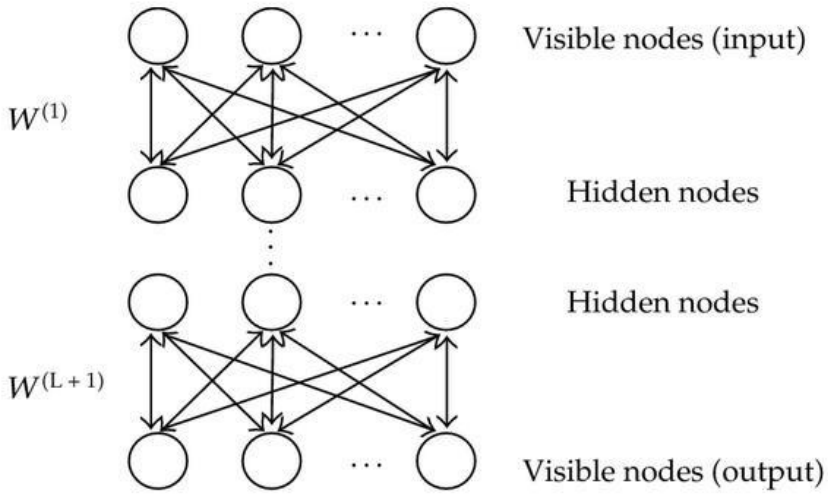
- If you are interested in the details of algorithms and neocortex, I highly recommend you to read the book, “On intelligence”.

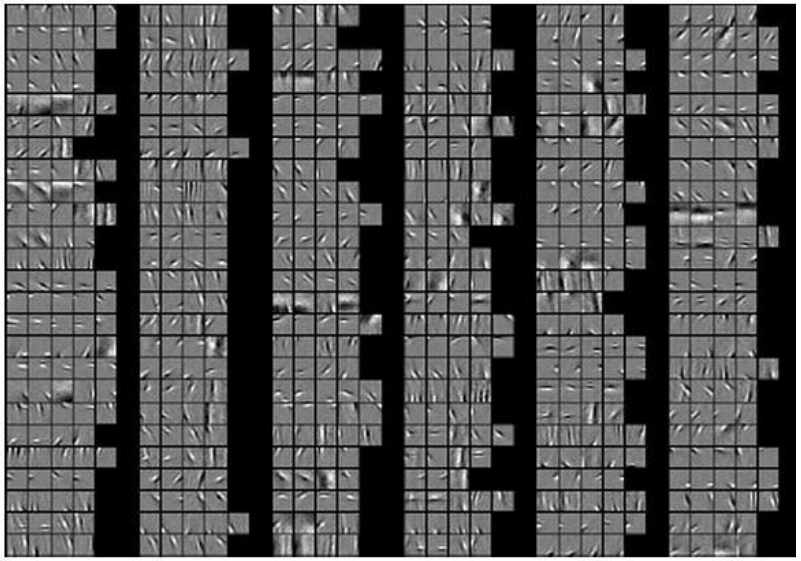
Artificial Neural Network (ANN)



- The main difference between ANN and other classifiers is that **information is distributed in the network**.
- Additionally, it is proven that ANN with the infinite number of hidden nodes is equivalent to Gaussian Process (single output).
- However, its structure is **too simplified**.
  - No feedback routine.
  - Cannot incorporate time.

Deep Belief Network (DBN)



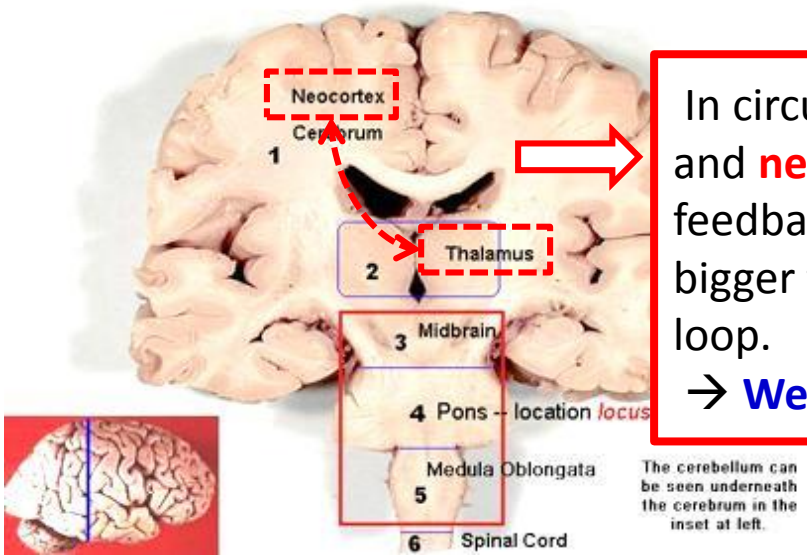


- The main difference between DBN and ANN is that it is basically **unsupervised learning architecture**.
- It is known that a DBN can mimic certain properties of visual area V1 and V2 [1].
- However, it is hard handle temporal data.

[1] Honglak Lee , Chaitanya Ekanadham and Andrew Y. Ng, "Sparse deep belief net model for visual area V2", Advances in Neural Information Processing Systems 2008

Truly Intelligent Machine (5/5)

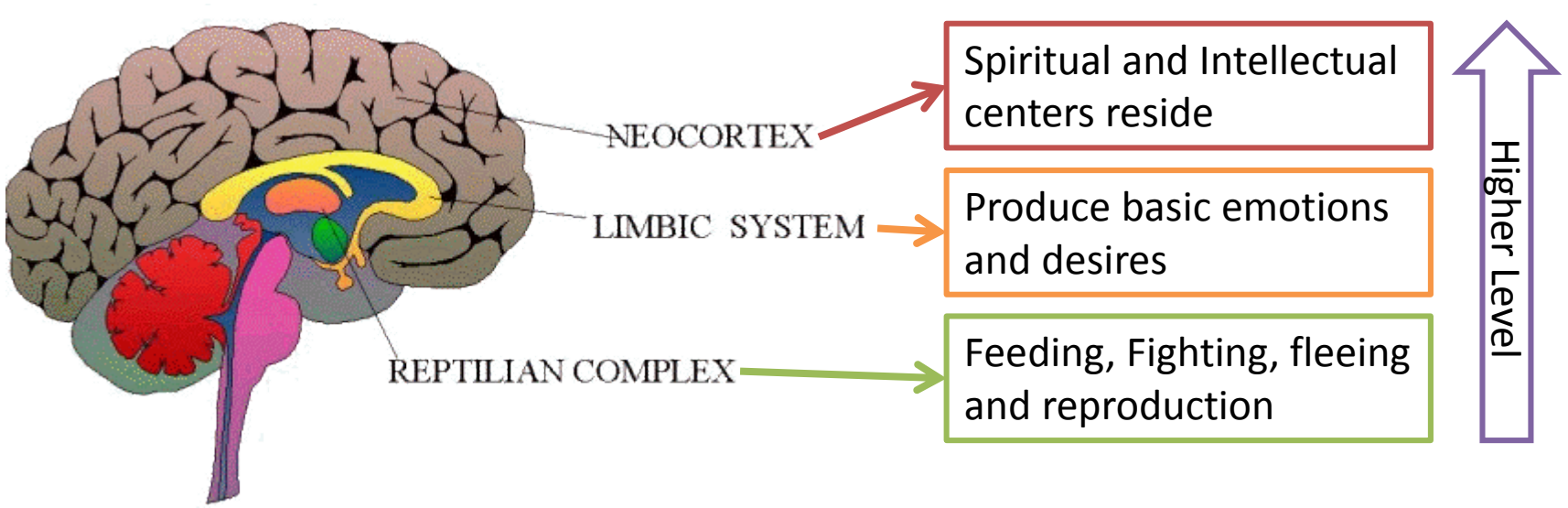
- The central pillar of understanding human's brain.
  1. Time must be considered.
  2. Feedback system must be implemented.
  3. Physical structure of actual brain must be reflected.
- Neither ANN nor DBN satisfies these properties.



In circuit connecting **thalamus** and **neocortex**, the portion of feedback loop is about 10 times bigger than that of feedforward loop.

→ **We see what we want to see.**

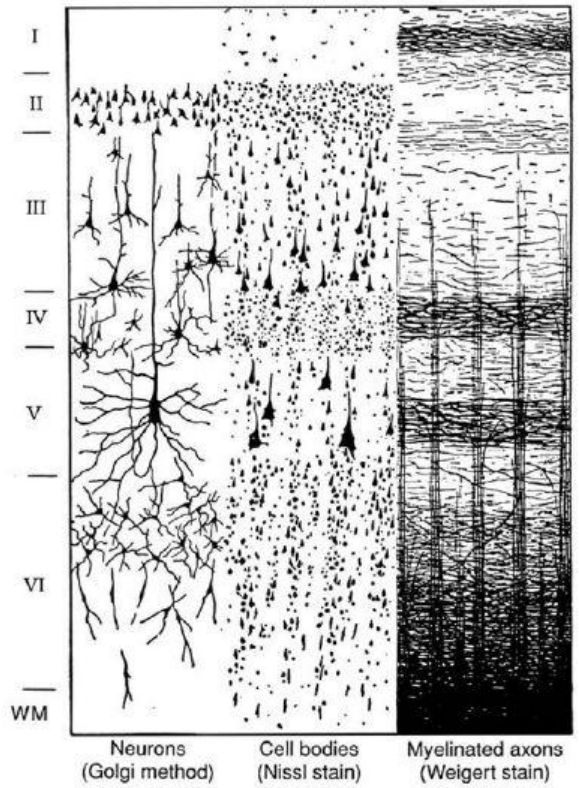
HUMAN BRAIN AND NEOCORTEX (1/2)



- Only vertebrate mammals (mammals which have backbone like a human or a monkey) have **Neocortex**.
- It is where **our intelligence** such sensing, guessing, reasoning, and thinking takes place.

Human Brain and Neocortex (2/2)

- The human **neocortex** is a sheet of neural tissue approximately 1,000 cm2 in area and 2mm thick. To visualize this sheet, think of a **cloth dinner napkin**, which is a reasonable approximation of the area and thickness of the neocortex.
- The **neocortex** is divided into dozens of functional regions, some related to vision, others to audition, and others to language, etc. Viewed under a microscope, the physical characteristics of the different regions **look remarkably similar**.
- From this **similarity**, we can infer that there could be **common thinking and learning mechanism**.



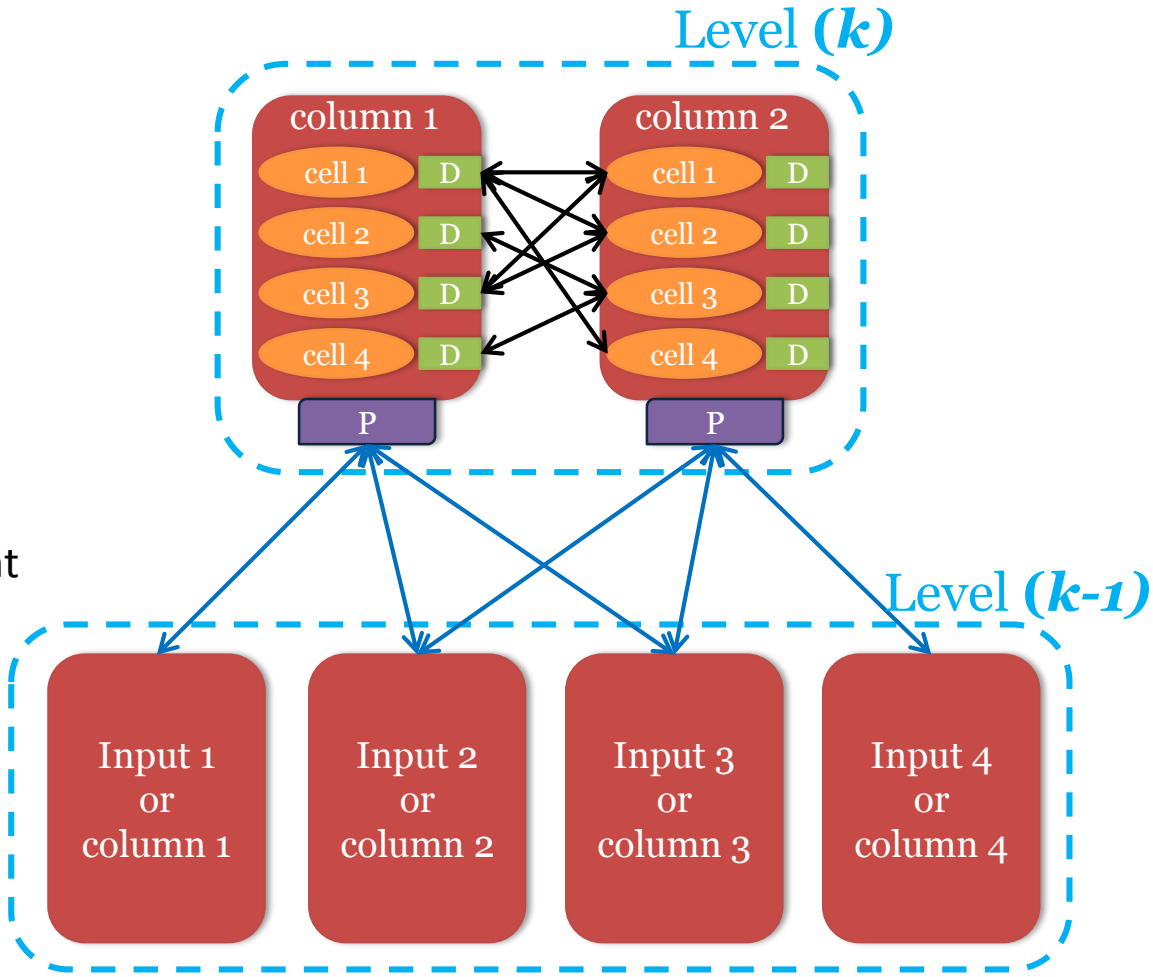
HIERARCHICAL TEMPORAL MEMORY

- **Hierarchical Temporal Memory** (HTM) is a machine learning technology that aims to capture the structural and algorithmic properties of the neocortex.
- HTM stores sequences of patterns. Successfully matching new inputs to previously stored sequences is the essence of inference and pattern matching. By matching stored sequences with current input, HTM forms a **prediction** about what inputs will likely arrive next.

1. Prediction is continuous.
2. Prediction occurs in every region at every level of the hierarchy.
3. Predictions are context-sensitive.
4. Prediction leads to stability.
5. Prediction tells us if a new input is expected or unexpected.

- How we **predict** in our daily lives.
  1. When listening to a song, we are continuously predict next note (or song, if played in a fixed playlist).
  2. When walking down the stairs, we are predicting when our foot will touch the next step.
  3. Our mouth is watering at the sight of lemon.
- How **prediction** is connected to a **Learning**.
  - We only **focus (learn)** when we encounter unpredicted situation. HTM looks for combinations of input bits that occur together often, which we call **spatial patterns**. It then looks for how these spatial patterns appears in sequence over time, which we call **temporal patterns** or sequences.

- **Terminology**
  - Region (Level)
  - Column
  - Cell
  - Dendrite segment
    - Distal dendrite segment
      - D
    - Proximal dendrite segment
      - P
  - Synapses
    - Distal synapse
    - Proximal synapse
  - Permanence

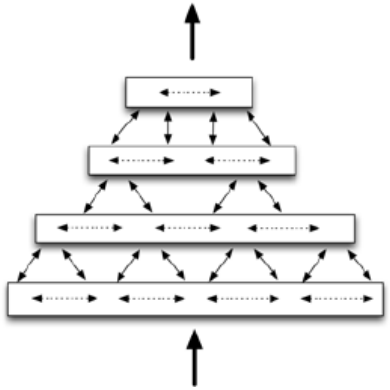


HIERARCHICAL TEMPORAL MEMORY (SUMMARY)

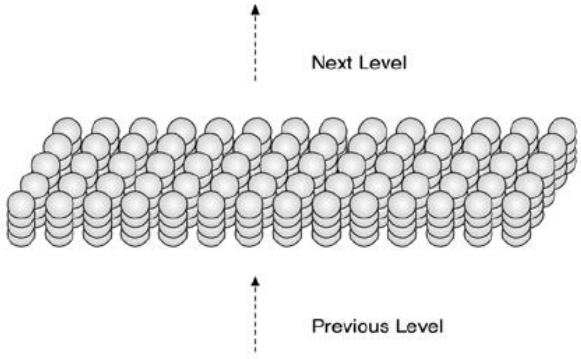
- Hierarchical Temporal Memory (HTM) is a machine learning technology that aims to **capture the structural and algorithmic properties of the neocortex**.
- The neocortex displays a remarkably uniform pattern of neural circuitry. The biological evidence suggests that the neocortex implements a **common set of algorithms** to perform many different intelligence functions.
- HTMs can be viewed as a type of neural network. By definition, any system that tries to model the architectural details of the neocortex is a neural network.
- However, unlike conventional neural network, HTMs model **neurons** (called **cells** when referring to HTM), which are arranged in **columns**, in **layers**, in **regions**, and in a **hierarchy**.
- As the name implies, HTM is fundamentally a **memory based system**.
- HTM memory has a hierarchical organization and is inherently **time based**. Information is always stored in a **distributed fashion.**

- **Key properties of HTM**
  - Hierarchy
  - Regions
  - Sparse Distributed Representations
  - The role of time
  - Learning
  - Inference
  - Prediction

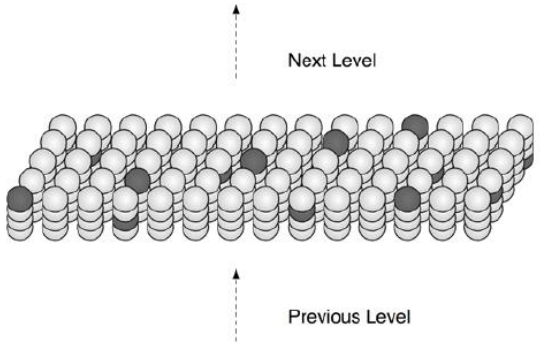
- **Hierarchy**
  - The region is the **main unit of memory and prediction** in an HTM and each HTM region represents one level in the hierarchy.
  - The benefit of hierarchical organization is **efficiency**. For an illustration, let's consider vision. At the lowest level of the hierarchy, your brain stores information about tiny sections of the visual field such as edges and corners. These low-level patterns are recombined at mid-levels into more complex components such as curves and textures. These mid-level patterns are further combined to represent high-level object features, such as heads, cars or houses.



- **Regions**
  - The notion of regions wired in a hierarchy comes from biology. The neocortex is a large sheet of neural tissue about 2mm thick. Biologists divide the neocortex into different areas or “regions” primarily based on how the regions connect to each other.



- **Sparse Distributed Representations**
  - Although neurons in the neocortex are highly interconnected, inhibitory neurons guarantee that only a small percentage of the neurons are active at one time. Thus, information in the brain is always represented by a small percentage of active neurons within a large population of neurons. (Thus sparse)



- **The role of time**
  - Time plays a crucial role in **learning**, **inference**, and **prediction**.
  - A static sound conveys little meaning. A word like “apple,” or the crunching sounds of someone biting into an apple, can only be recognized from the dozens or hundreds of rapid, sequential changes over time of the sound spectrum.
- **Learning**
  - An HTM region learns about its world by finding patterns and then **sequences of patterns** in sensory data.
  - It looks for combinations of input bits that occur together often, which we call spatial patterns. It then looks for how these spatial patterns appear in sequence over time, which we call temporal patterns or sequences.
  - Like a biological system, the learning algorithms in an HTM region are capable of “**on-line learning**”, i.e. they continually learn from each new input.

- **Inference**
  - When an HTM receives input, it will **match it to previously learned spatial and temporal patterns**. Successfully matching new inputs to previously stored sequences is the essence of inference and pattern matching.
  - Think about how you recognize a melody. Hearing the first note in a melody tells you little. The second note narrows down the possibilities significantly but it may still not be enough. Usually it takes three, four, or more notes before you recognize the melody.
- **Prediction**
  - Every region of an HTM stores **sequences of patterns**. By matching stored sequences with current input, a region forms a **prediction** about what inputs will likely arrive next. **HTM regions actually store transitions between sparse distributed representations**.

- **Overview of HTM architecture.**
  - Imagine your input consists of thousands or tens of thousands of bits. These input bits may represent sensory data or they may come from another region lower in the hierarchy. What are you supposed to do with this input?
  - Here is the simplest answer.
    1. Form a **sparse distributed representation** of the input
    2. Form a **representation** of the input in the context of previous inputs
    3. Form a **prediction** based on the current input in the context of previous inputs
  - Step 1 and 2 will be called "**Spatial pooler**" and Step 3 will be called "**Temporal pooler**".

- **Spatial pooler details (1/2)**
  - 1) Start with **an input consisting of a fixed number of bits**. These bits might represent sensory data or they might come from another region lower in the hierarchy.
  - 2) **Assign a fixed number of columns to the region receiving this input**. Each column has an associated dendrite segment. Each dendrite segment has a set of potential synapses representing a subset of the input bits. Each potential synapse has a permanence value. Based on their permanence values, some of the potential synapses will be valid.
  - 3) For any given input, **determine how many valid synapses on each column are connected to active input bits**.

- **Spatial pooler details (2/2)**
  - 4) The number of active synapses is multiplied by a “boosting” factor which is dynamically determined by how often a column is active relative to its neighbors.
  - 5) The columns with the highest activations after boosting disable all but a fixed percentage of the columns within an inhibition radius. There is now a **sparse set of active columns**.
  - 6) For each of the active columns, we **adjust the permanence values of all the potential synapses**. The permanence values of synapses aligned with active input bits are increased, and vice-versa. The changes made to permanence values may change some synapses from being valid to not valid, and vice-versa.

- **Temporal pooler details (1/3)**
  - 1) For each active column, check for cells in the column that are in a **predictive state**, and **activate them**. If no cells are in a predictive state, activate all the cells in the column. The resulting set of active cells is the representation of the input in the context of prior input.
  - 2) For every dendrite segment on every cell in the region, **count how many established synapses are connected to active cells**. If the number exceeds a threshold, that dendrite segment is marked as **active**. Cells with active dendrite segments are put in the predictive state unless they are already active due to feed-forward input.

- **Temporal pooler details (2/3)**
  - 3) When a dendrite segment becomes active, **modify the permanence values of all the synapses associated with the segment**. For every potential synapse on the active dendrite segment, increase the permanence of those synapses that are connected to active cells, and vice-versa.
  - Also, we choose the one that best matches the state of the system in the previous time step. For this segment, using the state of the system in the previous time step, **increase the permanence of those synapses that are connected to active cells** and decrement the permanence of those synapses connected to inactive cells. These changes to synapse permanence are marked as **temporary**.

- **Temporal pooler details (3/3)**
  - 4) Whenever a cell switches from being inactive to active due to feed-forward input, we traverse each potential synapse associated with the cell and remove any temporary marks. Thus **we update the permanence of synapses only if they correctly predicted the feed-forward activation of the cell**.
  - 5) When a cell switches from either active state to inactive, undo any permanence changes marked as temporary for each potential synapse on this cell. We don't want to strengthen the permanence of synapses that incorrectly predicted the feed-forward activation of a cell.

- **Phase 1: Overlap**
  - Given an input vector, the first phase **calculates the overlap of each column** with that vector. The overlap for each column is simply the number of connected synapses with active inputs, multiplied by its boost. If this value is below minOverlap, we set the overlap score to zero.

```
1.  for c in columns
2.
3.          overlap(c) = 0
4.          for s in connectedSynapses(c)
5.                  overlap(c) = overlap(c) + input(t, s.sourceInput)
6.
7.          if overlap(c) < minOverlap then
8.                  overlap(c) = 0
9.          else
10.         overlap(c) = overlap(c) * boost(c)
```

- **Phase 2: Inhibition**
  - The second phase **calculates which columns remain as winners after the inhibition step**. desiredLocalActivity is a parameter that controls the number of columns that end up winning. For example, if desiredLocalActivity is 10, a column will be a winner if its overlap score is greater than the score of the 10'th highest column within its inhibition radius.

```
11. for c in columns
12.
13.       minLocalActivity = kthScore(neighbors(c), desiredLocalActivity)
14.
15.       if overlap(c) > 0 and overlap(c) ≥ minLocalActivity then
16.             activeColumns(t).append(c)
17.
```

- **Phase 3: Learning**
  - The third phase performs learning**; it updates the permanence values** of all synapses as necessary, as well as the boost and inhibition radius.

```
18. for c in activeColumns(t)
19.
20.       for s in potentialSynapses(c)
21.             if active(s) then
22.                   s.permanence += permanenceInc
23.                   s.permanence = min(1.0, s.permanence)
24.             else
25.                   s.permanence -= permanenceDec
26.                   s.permanence = max(0.0, s.permanence)
27.
28. for c in columns:
29.
30.       minDutyCycle(c) = 0.01 * maxDutyCycle(neighbors(c))
31.       activeDutyCycle(c) = updateActiveDutyCycle(c)
32.       boost(c) = boostFunction(activeDutyCycle(c), minDutyCycle(c))
33.
34.       overlapDutyCycle(c) = updateOverlapDutyCycle(c)
35.       if overlapDutyCycle(c) < minDutyCycle(c) then
36.             increasePermanences(c, 0.1*connectedPerm)
37.
38. inhibitionRadius = averageReceptiveFieldSize()
39.
```

- **Phase 1**
  - The first phase **calculates the activeState for each cell** that is in a winning column. For those columns, the code further selects one cell per column as the learning cell (learnState).

```
18. for c in activeColumns(t)
19.
20.     buPredicted = false
21.     lcChosen = false
22.     for i = 0 to cellsPerColumn - 1
23.         if predictiveState(c, i, t-1) == true then
24.             s = getActiveSegment(c, i, t-1, activeState)
25.             if s.sequenceSegment == true then
26.                 buPredicted = true
27.                 activeState(c, i, t) = 1
28.                 if segmentActive(s, t-1, learnState) then
29.                     lcChosen = true
30.                     learnState(c, i, t) = 1
31.
32.     if buPredicted == false then
33.         for i = 0 to cellsPerColumn - 1
34.             activeState(c, i, t) = 1
35.
36.     if lcChosen == false then
37.         l,s = getBestMatchingCell(c, t-1)
38.         learnState(c, i, t) = 1
39.         sUpdate = getSegmentActiveSynapses (c, i, s, t-1, true)
40.         sUpdate.sequenceSegment = true
41.         segmentUpdateList.add(sUpdate)
```

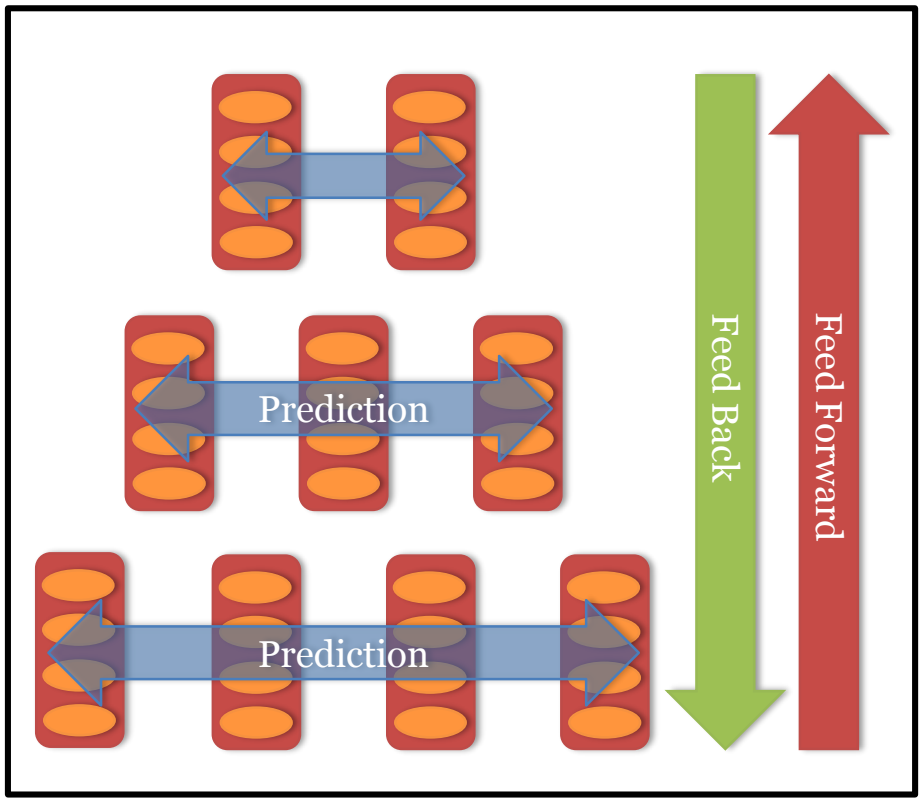
- **Phase 2**
  - The second phase **calculates the predictive state for each cell**. A cell will turn on its predictive state output if one of its segments becomes active, i.e. if enough of its lateral inputs are currently active due to feed-forward input.

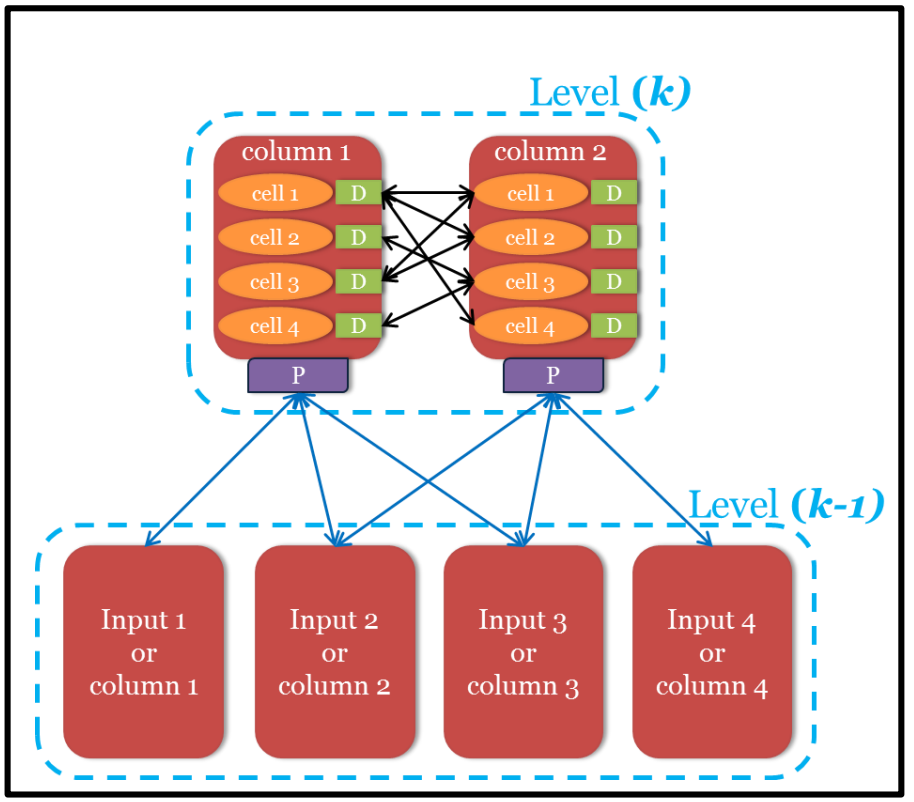
```
42. for c, i in cells
43.       for s in segments(c, i)
44.             if segmentActive(s, t, activeState) then
45.                   predictiveState(c, i, t) = 1
46.
47.                   activeUpdate = getSegmentActiveSynapses (c, i, s, t, false)
48.                   segmentUpdateList.add(activeUpdate)
49.
50.                   predSegment = getBestMatchingSegment(c, i, t-1)
51.                   predUpdate = getSegmentActiveSynapses(
52.                                         c, i, predSegment, t-1, true)
53.                   segmentUpdateList.add(predUpdate)
```

- **Phase 3**
  - The third and last phase actually **carries out learning**. In this phase segment updates that have been queued up are actually implemented once we get feed-forward input and the cell is chosen as a learning cell.

```
54. for c, i in cells
55.       if learnState(s, i, t) == 1 then
56.             adaptSegments (segmentUpdateList(c, i), true)
57.             segmentUpdateList(c, i).delete()
58.       else if predictiveState(c, i, t) == 0 and predictiveState(c, i, t-1)==1 then
59.             adaptSegments (segmentUpdateList(c,i), false)
60.             segmentUpdateList(c, i).delete()
61.
```

SUMMARY


Prediction
Prediction
Feed Back
Feed Forward

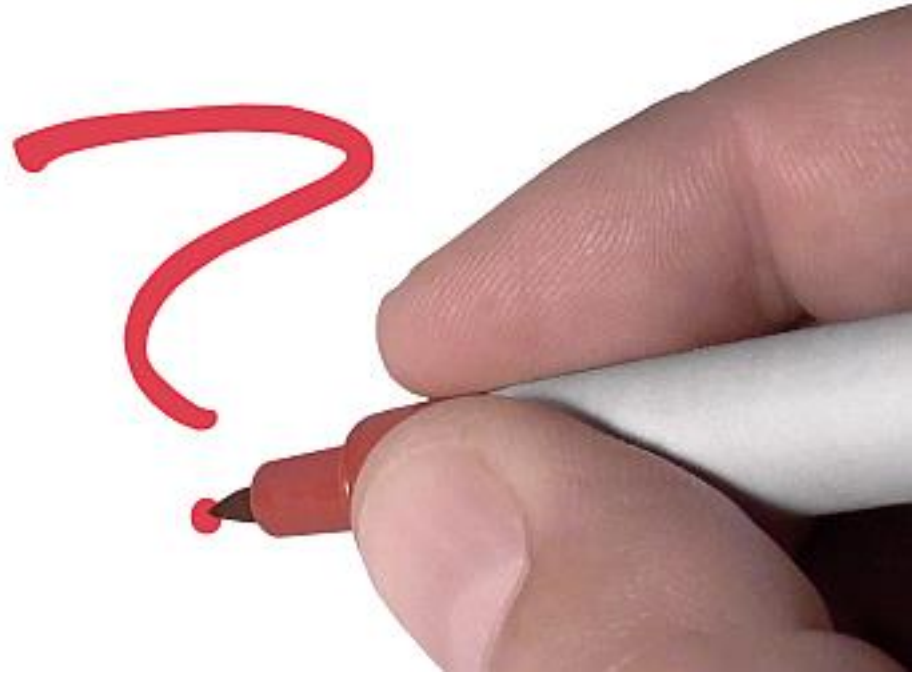

Level (k)
column 1
column 2
cell 1
cell 2
cell 3
cell 4
D
P
Level (k-1)
Input 1 or column 1
Input 2 or column 2
Input 3 or column 3
Input 4 or column 4

DISCUSSION

- **What makes HTM different from HMM?**
  - The main difference between HTM and HMM is the ability to incorporate **context**. Also the number of required node decreases significantly.
    - Same input does not necessarily mean same activation due to differently predicted cell.
- **What is the drawback of this algorithm?**
  - In my experience, it takes a lot of time for learning and the performance is sensitive to many thresholds.
  - It has many heuristic (perhaps natural) concepts and parameters, such as inhibition.
  - Feedback operation is not explained.
- **Can we implement this using other machine learning technique?**
  - There is a algorithm with same name implemented by Dileep George [1] (founder of Vicarious which is similar to Numenta).
  - It uses Markov chains to model coincidence patterns (temporal grouping).

[1] D. George and J. Hawkins, "Towards a Mathematical Theory of Cortical Micro-circuits", PLoS Computational Biology, 2009.

Q & A



Thank you for listening.