**Understanding HTM Sequence Memory**

Hugo Pristauz, Walter Eder, Willy Ottinger – Oct. 2023 (Draft v0.2)

## *Abstract*

*This paper is for jump starters into a playground with Numenta’s HTM (Hierarchical Temporal Memory) sequence memory algorithm. Sequence memory is capable of learning sequences, and to predict the next item, when part of a sequence is presented to the se­­quence memory.*

*In this sense some pre-requisites are setup first: a formal framework for HTM sequence memory with a detailed formulation of the HTM algorithm, which can be efficiently execute in Matlab or Python. Focus is on modular formulation down to the HTM-neuron level with local state transition and learning rules. Algorithms are in harmony with the standard formulations of nonlinear discrete time state representations, commonly used in system and control theory.*

*Based on these pre-requisites the HTM-algorithm is studied by running it on a 40 neuron toy network to investigate the behavior when feeding the system with different "toy input sequences". Finally, the performance of HTM sequence memory is compared with transformer based neural network alternatives, when both systems have to predict the continuation of text sequences from "Tiny Shakes­peare".*

# Introduction

In his book *On Intelligence* [1] Jeff Hawkins, with the help of Sandra Blakeslee, describes “*how a new understanding of the brain will lead to the creation of truly intelligent machines*”, suggesting that current “machine intelligence” is still far away from human intelligence. In his later book *A Thousand Brains* [2] Jeff Hawkins suggests a new *definition of intelligence* for beings (humans, ani­mals, machines) on base of a skill set leveraging four abilities found in the human brain:

* *Continuous learning*: the being must have the ability of continuous (on-line) learning, which is important for continuous adopting to a continuously changing world. This contrasts with current neural networks, which must be pre-trained before being utilized in an operational mode.
* *Sequential world modelling*: Creating a world model by physical (or mental) movement through the world with sequential information sensing generated by interactions with the world.
* *Multi World Prediction*: The brain of humans and mammals is an accurate prediction machine, which benefits from the possibility to select the best prediction model of a pool of simultaneously created and entertained prediction models.
* *Reference framing*: Reference frames are used for any kind of knowledge storage. Reference frames can be dynamically assembled, which, in consequence allows hierarchical “assembly” of knowledge to be stored with respect to reference frames.

The challenges induced by this list emphasize the impor­tance of *dealing with sequential information* with both spatial and temporal attributes. In this sense *sequence me­mo­ry* under focus with its promising prediction properties seems to be a powerful building block for above listed skill set.

, which promises to solve the prediction In addition, capability 3 to predict is not of less importance.

# Biological Background

HTM (Hierarchical Temporal Memory) sequence memory is a computer model of a biological neuron layer, which is based on a modern neuro-physical understanding of cortical neuron functionality [3,4].

# Ein Bild, das Zeichnung, Diagramm, Text, Entwurf enthält. Automatisch generierte Beschreibung

Figure 1: Comparison of neuron models

The neuron model used in most artificial neural networks is known as the perceptron [5], proposed by Frank Rosenblatt in 1957. It has few synapses and no modelling of dendrites (figure 1/A). The perceptron attempts to model the proximal area of the neuron and represents de-facto a mapping of input information arriving at the proximal synapses to the neuron’s *output.* There is evidence the proximal synapses, those closest to the soma (cell body), have a large effect on the likelihood of a cell generating an *action potential,* which is bursting along the neuron’s axon to synapses of other neurons of the network. It is important to realize that the perceptron model is a pure mapping model without any memory.

In contrast, an excitation of a distal (non-proximal) synapse has little effect at the soma (cell body). For this reason, it was hard to understand how the thousands of distal synapses can play a role for the cell’s responses [4].

Powered by advanced research methods [3], researchers could, however, unlock the secrets, that an activation of neighbored distal synapses within a short time interval leads to a local dendritic NMDA (N-methyl-D-aspartate) spike which causes a depolarization of the soma. Such depolarization subsequently enables the neuron to fire earlier than neighbor neurons with comparable proximal excitation, which gives such neuron the benefit to inhibit neighbored neurons.

Since NMDA spikes have a much longer duration than the action potential spikes of the soma, this mechanism acts like a memory functionality related to the depolarized state, in addition to the (perceptron like) mapping functionality of the neuron. Availability of memory in a neural model offers both the ability of *prediction*, and to implement *sequence state*, as will be shown in further sections.

Thus, compared to the perceptron model of figure 1/A, which generates the output signal according to a mapping of input signals at the proximal synapses (feedforward information) an extended neural model (like the HTM neuron model) deals additionally with context information arriving at distal synapses of several dendritic segments, which causes a state change of the neural model, influ­encing the neuron’s behavior in the near future.

# Synaptic Model

In Neurobiology a synapse forms a connection between a neuron's dendrite and an axon of some other neuron.

Ein Bild, das Cartoon enthält.

Automatisch generierte Beschreibung

Figure 2: axon/dendrite connections via synapses

In formal neuron models synapses are usually assigned with a weight, influencing the transmission efficacy of the synapse, which is changed during the learning process. In the HTM approach there are two fundamental diffe­rences compared to the learning rules of most neural models.

* First, learning occurs by growing and removing synapses from a pool of "potential" synapses [7].
* Second, Hebbian learning and synaptic change occur at the level of a dendritic segment, not the entire Neuron [8].

Learning in an HTM neuron is modeled by the growth of new synapses from a set of potential synapses. A "permanence" value is assigned to each potential synapse and represents its growth. Learning occurs by incrementing or decrementing the permanence values. The synaptic weight is a binary value set to 1 if the permanence is above a threshold (figure 3).

Ein Bild, das Diagramm, Reihe, Screenshot, Text enthält.

Automatisch generierte Beschreibung

Figure 3: permanence p and weight of a synapse

Since the pre-synaptic signal is also binary, the synapse acts as a logical gate. For sufficient high permanence *p ≥ η*, the synapse is called *connected* and allows a pre-synaptic signal to cause a post-synaptic effect *(q=z)*, while otherwise the synapse behaves as *un­con­nec­ted* with no post-synaptic effect, regardless of the value of the pre-synaptic signal *(q=0)*.

Ein Bild, das Text, Screenshot, Diagramm, Reihe enthält.

Automatisch generierte Beschreibung

Figure 4: Block diagram for a single HTM synapse

This is shown in figure 4, where the (binary) pre-synaptic signal *z* is gated by the binary *synaptic weight* *w = (p ≥ η)*, the Boolean result of the comparison of permanence *pÎ*[0,1]with synaptic threshold *η* (a typical setting is *η = 0.5).* The other block input is context ***c****Î{0,1} L*, typically a high dimensional binary vector col­lect­ing neu­ron outputs from a large neuron context set. The *select* function block in figure 5 means to pick the *k*-th element of *context* ***c*** for the pre-synaptic signal *z*. In this sense index *k* contributes to the neural network topology definition. The for­mal mo­del for an HTM-synapse is as follows:

*w = (p ≥ η) w Î{0,1}, synaptic weight* (1a)

*z* = ***i****kT⋅* ***c*** *= ck z Î{0,1}, pre-synaptic signal* (1b)

*q* = *w⋅ z q Î{0,1}, post-synaptic effect* (1c)

with ***i****k*denoting the *k-th* unit vector *of {0,1}L,* and *T* used for trans­pose.

# Synaptic Regions

So far (1a-1c) tells us whether a given context ***c*** presented to a synapse will contribute to the synaptics q (will cause a post-synaptic effect*)* with respect to the current synaptic permanence *p*, which is a sort of synap­tic state.

A formal learning rule, however, is still missing, and we postulated for our model, that *learning* and *synaptic change* shall occur on the level of dendritic segments and not the entire neuron. This leads us to the concept of *synaptic regions* where groups of synapses collaborate with each other in order to generate some group effect.

As an example, the synapses of all proximal dendrites (receiving feedforward signals - see figure 1) collaborate with each other in order to cause occasionally the soma to generate an action potential. In this case we speak about the *proximal synaptic region*.

In contrast we have several *distal synaptic regions* formed by a collection of all synapses of distal dendritic segments. If those synapses generate sufficient collective post-synaptic effect, the distal dendritic segment can generate an NMDA spike, which can put the neuron into a so called *predictive state* by depolarization.

While we do not cover the proximal synaptic region in this paper, we will deal with the formal HTM-model for the distal synaptic regions related to distal dendritic segments.

We will start with a generic formal model for a synaptic region (figure 5).

Ein Bild, das Diagramm, Screenshot, Reihe, Plan enthält.

Automatisch generierte BeschreibungFigure 5: block symbol for a) single synapse, b) synaptic region

While in figure 5a input/output signals of a single neuron's model are scalars (except context ***c***), we extend the single-synapse model in figure 5b to model a synaptic region with a total number of *s* synapses. The individual synapses are still considered to work independently from each other according to (1a-1c), except the *s* instances of signals *p*, *w*, *q*, *z*, and *k* are now grouped in terms of vectors ***p****r ,* ***w****r ,* ***q****r ,* ***z****r* and ***k****r* augmen­ted with index *r* to refer to synaptic region *rÎ{1, ... , d}*, *d* denoting the total number of *synaptic regions.*

The formal model is straight forward,

***w****r = (****p****r ≥ η)* ***w****rÎ{0,1}d,* synaptic weights (2a)

***z****r* = ***c*** *(* ***k****r )* ***z****r Î {0,1} d, pre-synaptic signals* (2b)

***q****r* = ***w****r* °***z****r* ***q****rÎ{0,1} d, synaptic* (2c)

with ° denoting elementwise product, and ***c*** *(* ***k****r )* mea­­ning to pick *d* elements from***c***relating to the indices given by***k****r .*

# Synaptic Bank

The formal model of an HTM-neuron requires typically more than one synaptic regions. A set of synaptic regions can be combined to a synaptic bank, which is a *formal generic building block* used to model the behavior of a set of *distal dendritic segments*. Those play a key role in the HTM neuron for modelling soma depolarization and synaptic learning when sufficient spatially and temporally neigh­bored synapse activity occurs.

Ein Bild, das Diagramm, Screenshot, Plan, technische Zeichnung enthält.

Automatisch generierte Beschreibung

Figure 6: schematics of a *synaptic bank: a) less detailed, b) more detailed*

Figure 6a and 6b symbolize a synaptic bank with synapses arranged in a *d×s* matrix to model, e.g., a set of *d* dendritic segments with s synapses each. Given d synaptic regions indexed by *r* with *permanences* ***p****r ,**connection indices* ***k****r* , *synaptic weights* ***w****r* , and output signals ***z****r* , ***q****r* , we arrange these vectors as matrices.

***P*** = [... , ***p****r* , ...] , ***K*** = [... , ***k****r* , ...] ,  ***W*** = [... , ***w****r* , ...]

***Z*** = [... , ***z****r* , ...] , ***Q*** = [... , ***k****r* , ...]

The representation of the listed vectors as matrix columns must not be confused with the horizontal arrangement of a synapse region in figure 6. Equations (2a-2c) give the for­mal description of the synapse bank functionality.

***W*** *= (****P*** *≥ η)* ***W****Î{0,1} d×s ,* synaptic weights (3a)

***Z*** = ***c*** *(* ***K****)* ***Z*** *Î {0,1} d×s, pre-synaptic signals* (3b)

***Q*** = ***W***°***Z******Q****Î{0,1} d×s , "synaptics"* (3c)

Matrix ***Q***, which holds the plurality of synaptic effects of each synapse, plays such an important role that we invented the artificial name *synaptics* for it, and we will recap the meaning of the *synaptics* ***Q***:

* If an element *qµν* of the *synaptics* ***Q*** equals 1, it means both, that synapse *µ* of the synaptic field ***qν*** is connec­ted (permanence *qµν* ≥ *η*), and that the neuron addressed by index *k µν* (found in matrix ***K*** in row *µ* and column *ν*) fires its output signal through this synapse.
* If an element *qµν* of the *synaptics* ***Q*** equals 0, it means that at least one of the conditions above is violated.

We use the 1-norm of a vector || **q** ||1 for the sum of absolute values of its elements, and the 1-norm of matrix ***Q*** as the maximum absolute column sum

|| ***Q*** ||*1 := max j* {|| ***q****j*||1} (4)

The 1-norm *||****Q****||1* builds internally in a first step all column sums of the *synaptics*, which are the sums of post synaptic effects of each synaptic region (or distal dendritic segments), and picks in a second step the maximum of these values.

In the next section, where the HTM cell algorithm is presented, we will see that a neuron gets predictive whenever the 1-norm of the *synaptics* ***Q*** exceeds a threshold *θ.*

***|| Q* ||***1 ≥ θ* (5)

Equation (5) means that at least one of the synaptic regions (distal dendritic segments) "sees" sufficient post-synaptic activities, i.e., can sum up the number of post-synaptic effects to a value exceeding threshold *θ.*

## Example 2

Consider the following *synaptics* of a neuron with threshold *θ = 2.*

***Q*** *=*

The post synaptic effects for synaptic fields 1,2,3 are in vector-representation ***q****1 = [0 1 1 0 0]T,* ***q****2 = [1 1 0 0 0]T,* ***q****3 = [0 0 1 0 0]T*. These model post-synaptic activity in the distal dendritic seg­ments 1,2,3. The related 1-norms of the ***q****j*-vectors are:

*||* ***q****1 ||1 = 2, ||* ***q****2 ||1 = 3, ||* ***q****3 ||1 = 2*

The 1-norm of the synaptics is

***|| Q* ||***1 = max*{*||* ***q****1 ||1,||* ***q****2 ||1 ,||* ***q****3 ||1*} *= max{2,3,2} = 3*

Since with *θ = 2* we have ***|| Q* ||***1 ≥ θ,* the neuron will enter a predictive state.

# Mathematical Model of HTM Neuron

In HTM terminology a single HTM neuron is called a *cell*, which can be treated in the first approach as a black box with the input/output model shown in figure 3.

Ein Bild, das Diagramm, Kreis, Reihe enthält.

Automatisch generierte Beschreibung

Figure 6: input/output model of a cell (HTM neuron)

All quantities shown in figure 6 are related to a discrete time index *t* (*t = 0, 1, 2, …* ). The HTM neuron presented in this paper has two kinds of inputs

* the (scalar) binary *feedforward input* *ut*Î{0,1}
* the (vectorial) binary *context input* ***c*** *t* Î {0,1}L.

and

* the (scalar) binary cell state *xt*Î{0,1}

Further the HTM neuron entertains an *s×d* synaptic bank, entertaining

* the permanence matrix ***P*** *t*Î [0,1] *s×d*

with [0,1] denoting the closed interval{*x*Î  *|* 0£ *x* £1}. Since the tuple (*xt,* ***P*** *t* ) represents a state of the cell, which will be sequentially updated at each time step *t*, we need to provide

* initial cell state *x 0*Î{0,1}
* initial permanence matrix ***P*** *0*Î [0,1] *M,*

which is sketched in figure 6 by the two input signals at the top. Finally, we have

* the (scalar) binary cell output *y t* Î{0,1}.

In addition, we have some static cell parameters ***G***Î *(1,2,…,N)m* and ***K***Î *(1,2,…,L) s×d*, which are index matrices, telling the cell who are the group-partners (other cells of a *minicolumn*), and about the con­nections with other cells to form the pre-synaptic signal ma­trix ***Z*** *t*, as it has been described in the formal model for sy­nap­tic fields. Finally, we have two other internal matrix quan­­tities, synaptic weight matrix ***W*** *t*, and the synaptics ***Q*** *t*.

# HTM Cell Algorithm

We have now all pre-requisites for a formal model of the HTM neuron (cell). Depending on the binary values of their input, state and output we will call a cell in the following as

* *excited*, when its input is activated (*u t = 1*)
* *predictive*, when its state is activated (*x t = 1*)
* *active*, when its output is activated (*y t = 1*)

The HTM algorithm, broken down to a single neuron, starts with

**A) Configuration and Initialization phase**

* Matrix ***G*** is configured with the indices of the other neurons, with which the neuron collaborates inside of a group (minicolumn).
* Matrix ***K*** is configured with the indices of the context vector ***c*** to tell the neuron, which signals of the context vector connect to corresponding synapses of the synaptic bank.
* State *x**0* is initialized as zero.
* The permanence matrix ***P*** *0* is initialized with normal distributed random numbers around mean value *η.*

**B) Iteration Phase**

Step1: input *u t*

Step 2: calculate output *y t* from state *x t* and input *u t*

*y t' = x t* · *u t*  (6)

Step 3: update *y t* in case of a *burst* condition, i.e., no other cell in the group is in a predictive state. This requires that before execution of step 3 all other cells have already executed step 2, so that outside the cell algorithm ***c*** *t'* can be updated for progressed time *t'.*

*y t = y t'* ∨ (||***c*** *t'(****g****)||1 = 0)*  (7)

Step 4: calculate the cell's *synaptics* ***Q*** *t*

***W*** *t* = *(****P*** *t* ≥ *η)* weight matrix(8a)

***Z*** *t* = ***c*** *t(****K****)* pre-synaptic signals (8b)

***Q*** *t* = ***S*** *t* ° ***Z*** *t* synaptics (8c)

Step 5: perform a state transition, i.e., calculate *x t* from the *synaptics* ***Q*** *t*

*x t+1 =* (*||* ***Q*** *t ||1 ≥ Θ* ) state transition (9)

Step 6: Hebbian like learning; all *active* cells change the permanence values of their synapses. Synapses with corresponding non-zero synaptics' elements are *reinforced*, all other synapses of an *active* cell are *penalized*.

*Δ****P*** *t* ***=***  *(δ +****Q*** *t - δ -)y t* (10a)

***P*** *t+1* ***=*** ***P*** *t* ***+*** *Δ****P*** *t* (10b)

# Example

# ... to be done ...

Text after here needs rework!

# Input Sequences

Before studying this algorithm in an example, we need another pre-requisite: covering a formal description of input sequen­ces. Let us consider two sentences “Mary likes to sing.” and “John likes to dance.” , for which we define a vocabulary

*V* = {’Mary’, ’John’, ’likes’, ’to’, ’sing’, ’dance’, ’.’}

and related sequences

*S1* = (’Mary’, ’likes’, ’to’, ’sing’, ’.’)

*S2* = (’John’, ’likes’, ’to’, ’dance’, ’.’) .

Since each input at time instance *t* to our cell layer must be a binary vector ***u****t*Î{0,1}*N*, we must map each symbol of sequence *S1* and *S2* to the binary vector space {0,1}*n*, and we call this process tokenizing, and talk about the binary vectors ***u****0,* ***u****1,* ***u****2, …* as *tokens.* These are sequentially presented as input vectors ***u****t* to the cell layer at time instances *t = 0, 1, 2, …* .

In this sense we can define a tokenizer as a function F*: V* {*1,0*}*n*, mapping from a vocabulary *V* to the binary input space {*1,0*}*n*.

# Example 1

Let us define a 40 cell layer (*m=4, n=10*) where each cell supports 2 dendritic segments (*d=2*) with 5 synapses per segment (*s=5*). Thus, the layer has *n = 10* minicolumns with a total of *N = m·n = 40* cells*,* where each cell entertains *M = s·d = 10* synapses in total. Referring to vocabulary

*V* = {’Mary’, ’John’, ’likes’, ’to’, ’sing’, ’dance’, ’.’}

we could define a tokenizer F*: V* {*1,0*}*10* with the mapping

’Mary’ [0 0 1 0 0 1 1 1 0 0]*T*

’John’ [1 0 0 1 0 1 1 0 0 0]*T*

’likes’ [0 1 0 0 0 0 0 1 1 1]*T*

’to’ [0 1 0 1 1 0 0 0 0 1]*T*

’sing’ [1 1 0 0 1 0 1 0 0 0]*T*

’dance’ [1 0 0 1 1 0 0 0 0 1]*T*

In this sense the tokenizer maps the sequence

*S1* = (’Mary’, ’likes’, ’to’, ’sing’, ’.’)

to the input token sequence

*U1 =* {***u****0,* ***u****1,* ***u****2,* ***u****3,* ***u****4*} = { },

and sequence

*S2* = (’John’, ’likes’, ’to’, ’dance’, ’.’) .

To the input token sequence

*U2 =* {***u****0,* ***u****1,* ***u****2,* ***u****3,* ***u****4*} = { }.

# Distal Dendritic Segments

Let us next investigate the distal dendritic segments of a cell and consult figure 5, which shows a neuron with two (*d=2*) distal dendritic segments, each of them entertaining five synapses (*s=5*).

Figure 5: neuron with *2* distal dendritic segments and *5* synapses per segment (s = 5, d = 2, time index has been omitted)

According to figure 3 the model for cell *k* (HTM neuron *k*) provides both a *parameter vector* ***k****k* as well as a *distal*

# The HTM Algorithm

We are ready now to formulate the coarse form of the HTM

algorithm for sequence memory. The original form of this algorithm has been presented in [6]. The form which is used here makes more use of vector/matrix operations and less use of if-branches, which ends up in an algorithm with sequential, straight forward flow.

The computational scheme of the HTM algorithm for a sequence memory layer (as shown in figure 5) is outlined in figure 6.

Ein Bild, das Diagramm, Kreis, Screenshot, Reihe enthält.

Automatisch generierte Beschreibung

Figure 6: computational scheme for the HTM algorithm

Besides of the quantities ***u*** *t*, ***x*** *t*, ***D*** *t* and  ***y*** *t*, which are shown in figure 5, the *coincidence matrix* ***Q*** *t* introduced in the previous section plays an important computational role as an intermediate quantity. Figure 6 can be interpreted as follows: After initializing the layer state (***x*** *0*, ***D*** *0*) at time *t=0* a sequence of updating iterations is performed re­peated­ly for the discrete time sequence t = 0, 1, 2 ,...

initialize ***x*** *0*, ***P*** *0*

for *t = 0, 1, 2, ... :*

input ***u*** *t*

calculate ***y*** *t* from ***x*** *t*, ***u*** *t* (output activation)

calculate ***Q*** *t* from ***P*** *t*, ***y*** *t* (synaptic effects)

update ***x*** *t+1* from ***Q*** *t* (cell state update)

update ***P*** *t+1* from ***P*** *t,* ***Q*** *t* (cell state update)

Before diving into the algorithmic details let us recap that we call a cell *k* with activated input *excited* (*ukt = 1*). In case of activated output (*xkt = 1*) we call it *active*, and we talk about a *predictive* cell when its cell state is set (*xkt = 1*). Also keep in mind that all cells belonging to the same minicolumn *j* share the same input *ujt*. Thus, an excitation of one cell in a minicolumn implies an excitation of all cells in the minicolumn, which suggests calling the whole minicolumn *excited*, whenever one cell (which implies: all cells) of the minicolumn is excited.

In addition, we call a minicolumn predictive, if it contains at least one predictive cell, otherwise the minicolumn is called non-predictive. An excited minicolumn with no predictive cells is called a *bursting* *minicolumn*.

### Step 1) Output Activation

The rules for output activation, where calculate ***y*** *t* is calcu­lated from ***x*** *t*, ***u*** *t*, are as follows:

* excited predictive cells get active
* cells of bursting minicolumns get active
* any other cell gets inactive

The first rule is biologically inspired by the capability of pyramidal cells to transition into a mid-term (> 100ms) depolarized state caused by distal dendritic NMDA spikes [3,4]. Such depolarized neurons, which play a predictive role, are easier to activate and will start to fire earlier than non-depolarized neurons [6].

The second rule covers just the default, where there are no depolarized neurons in a minicolumn. In this case no neuron has an activation advantage against other neurons in the minicolumn, and all cells of the bursting minicolumn (which is by definition *excited*) become *active* [6].

The calculations are as follows:

***B*** *t = 1**-* ***1****m****max****(****X*** *t)* burst matrix ***B*** *t*Î{0,1}*mxn* (8a)

***C*** *t =* ***X*** *t +* ***B*** *t* candidates ***C*** *t*Î{0,1}*mxn* (8b)

***Y*** *t* = ***C*** *t* ° *(****1****m* ***⋅ u*** *tT)* output activation (8c)

Even this kind of formulation looks a bit unusual, the chosen form has the advantage of generating a straight-forward sequence of intermediate matrix results

***x*** *t* ***X*** *t* ***B*** *t* ***C*** *t* ***y*** *t*

with helpful interpretations, as will be demonstrated in the subsequent example. In the consolidated form (9) we clearly see the dependency of ***y*** *t* from ***x*** *t* and ***u*** *t*.

***Y*** *t = (****X*** *t + (1**-* ***1****m****max****(****X*** *t))° (****1****m* ***⋅ u*** *tT)* (9)

### Step 2) Calculate Coincidence Matrix

In this step the *layer's coincidence matrix* ***Q*** *t* is calculated from the permanence state ***D*** *t* and the layer output ***y*** *t.* The rationale behind is that each cell entertains d distal dendritic segments which act as coincidence detectors. Any element of the binary matrix ***Q*** *t*, which equals 1, tells about a coincidence of a related *enabled* distal synapse with a connected active cell. In contrast, a value of 0 means that either the synapse is disabled, or the connected cell is inactive, or both.

The details for the calculation of the coincidence matrix have been explained previously in detail. Thus, we only have to summarize the required calculations of this step.

***W*** *t* = *σ(****D*** *t - η)* weight matrix (10a)

***Z*** *t* = ***c*** *t(****K****)* pre-synaptic signals (10b)

***Q*** *t* = ***S*** *t* ° ***Z*** *t* synaptics (10c)

Again (10a-c) can be consolidated to a single operation:

***Q*** *t* = *σ(****D*** *t - η)* ° ***y*** *t(****K****)* (11)

Each Boolean element of the *synaptic matrix* ***S*** *t* in (10a) tells us, whether the related synapse is enabled or disabled, depen­ding on whether its permanence value exceeds threshold *η* or not. In (10b) the layer output ***y*** *t* is projected into a subspace to yield ***Z*** *t* according to the index values of **K** in order to achieve compatible matching with the synaptic matrix ***S*** *t*. Finally ***S*** *t* and ***Z*** *t* are element-wise multiplied to establish the coincidence values.

### Step 3) Cell State Transition

The aim of this step is to update *the cell state* by calculating***x*** *t+1* from the *coincidence matrix* ***Q*** *t*.

***p*** *t* = [ *||* ***Q****1 t ||1*,*||* ***Q****2 t ||1*, ...,*||* ***Q****n t ||1*]*T* Î{0 *... s*}*n* (12a)

***x*** *t+1 = (****p*** *t* *≥ θ)* (12b)

Equation (12) means that the given *coincidence matrix* ***Q*** *t* has to be decomposed according to (7) into its parts ***Q****k t*. Then, according to (6), for each ***Q****kt* we calculate the coincidence number

*pkt* = *||* ***Q****k t ||1* = *max*(*||****q****k1t||*1*, ||****q****k2t||*1*, ..., ||****q****kdt||*1)

as the largest column sum, and form vector

***p*** *t* = [*p1t, p2t, ..., pnt* ]*T.* Î{0 *... s*}*n*

In (12b), finally, we set cell state *xkt+1* = 1 when the related co­incidence number exceeds a threshold *(pkt* *≥ θ)* , other­wise we set *xkt+1* = 0. Again (12a,12b) can be replaced by the consolidated form (13), which expresses the dependency of ***x*** *t+1* from***Q*** *t*

***X*** *t+1* = ([*||****Q****1 t ||1*, *||****Q****2 t ||1*, ... , *||****Q****n t||1*]*T ≥ θ)(m×n)* (13)

### Step 4) Learning

In the last state the permanence values of the distal dendritic segments are updated, which is called learning. The following rules are applied:

* synapses of active cells with coincidence increase their permanence values
* synapses of active cells with no coincidence decrease their permanence values
* inactive cells do not change their permanence values

Formally we have

*Δ****D*** *t* ***=***  *(δ +****Q*** *t - δ -)* ° ***Y*** *t* (14a)

***D*** *t+1* ***=*** ***D*** *t* ***+*** *Δ****D*** *t* (14b)

with *δ +* > *δ -*. The consolidated form is:

***D*** *t+1* ***=*** ***D*** *t* ***+*** *(δ +****Q*** *t - δ -)* ° ***Y*** *t* (15)

### Summary: HTM Algorithm

Taking (9,) the HTM algorithm looks in its compact form:

Ein Bild, das Text, Schrift, Screenshot, weiß enthält.

Automatisch generierte Beschreibung

# References

[1] Hawkins J., Blakeslee S.: “On Intelligence”; Owl Books (2005).

[2] Hawkins J.: “A Thousand Brains”; Basic Books, New York (2022).

[3] Antic, S. D.,et al.: “The decade of the dendritic NMDA spike”; *J. Neurosci. Res.* 88, 2991–3001. doi: 10.1002/jnr.22444 (2010).

[4] Major, G., Larkum, M. E., and Schiller, J.: “Active properties of neocortical pyramidal neuron dendrites”; *Annu. Rev. Neurosci.* 36, 1–24. doi: 10.1146/annurev-neuro-062111-150343 (2013)

[5] Rosenblatt, F.: „The perceptron. A probabilistic model for information storage and organization in the brain“; *Psychological Reviews*, 65 (1958): S. 386–408.

[6] Hawkins J., Subutai A.: „Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex“; *frontiers in Neural Circuits, 2016*.

[7] Hawkins J., Subutai A., Cui Y.: „A Theory of How Columns in the Neocortex Enable Learning the Structure of the World“; *frontiers in Neural Circuits, 2017*.

[8] Chklovskii et al.: Cortical rewiring and information storage. *Nature* 431, 782–788. doi: 10.1038/nature03012 (2004).

[9] Stuart, G.J., Häusser, M.:. Dendritic coincidence detection of EPSPs and action potentials. *Nat. Neurosci.* 4, 63–71. doi: 10.1038/82910 (2001)

[10] Berlyand L., Jabin P.E.: „Mathematics of Deep Learning; An Introduction“; *de Gruyter Textbook, 2023*.

# Appendix A: Notations

In the following sections we need some notational definitions, which will simplify our life. Given an arbitrary set *S* and a sequence of matrices

***A****1,* ***A****2, ...,* ***A****l Î S m×n*

we define ***A*** *=* [***A****1,* ***A****2, ...,* ***A****l*] = [***A****k ,*] *Î S m×nl* as a matrix constructed by horizontal line-up of partial matrices ***A***k. Conversely we define ***A*** *=* [***A****1;* ***A****2; ...;* ***A****l*] = [***A****k ;*] *Î S ml×n* as a matrix constructed by *vertical* line-up of matrices ***A***k, i.e., ***A*** *=* [***A****1T,* ***A****2T, ...,* ***A****lT* ]*T,* where *T* denotes *transpose*.

We understand ***A*** ° ***B*** as element-wise matrix multipli­cation and use symbol ***1****m×n* for an *m×n* matrix with all elements equal to 1. Notation *s* + ***A*** (s *Î S)* means to add a scalar value to each element of a matrix, i.e., *s*+***A*** = *s****1****m×n* +***A****.*

We write ***a****T* = ***max****(****A****)Î S1× n* for a row vector with elements representing the maximum values of the columns of ***A***, i.e.,

***max***([***a****1,* ***a****2, ...,* ***a****n*]) = [max(***a****1*), max(***a****2*), ..., max(***a****n*)]. Con­versely ***a*** = ***max****(****A****T) T* results in a column vector*Î S m* containing the maximum values of the rows of ***A***.

We use the 1-norm || ***a*** ||1 or ∞-norm || ***a*** ||∞ of a vector to denote the sum of absolute values or maximum absolute value of its elements.

# Appendix B: Math Preliminaries

Most of today’s artificial neural networks (ANNs) are based on *neuron functions*, which are defined [1] as functions *f* : of the form

*f(****x****) = (****w****·****u*** *+* b*)* (1)

where : is a continuous non-linear function called the activation function, ***u*** Î is called the *input vector*, while ***w*** Î is a vector of weights. Scalar b Î is called the bias, while ***w****·****u*** is the inner product on .

Such neuron function is a full description of a single *artificial neuron*. It is motivated by one of the first formal (approximate) descriptions of a biological neuron, the *perceptron*, introduced in 1957 by Frank Rosenblatt [2].

Artificial neurons can be grouped into layers comprising *m* artificial neurons. Such *artificial neuron layer* maps an input vector ***u*** Î (comprising *n* scalar inputs) to an output vector ***y*** Î (comprising *m* scalar outputs). The mapping is given by the so-called layer function ***g***: with

***g****(****x****) = (f1(****x****), f2(****x****),…, fm(****x****))* (2)

with *f1*, … , *fm*employing weight vectors ***w****1*,…,***w****m* and biases b*1,…,* b*m.* By using the weight vectors ***w****i* as the columns of a weight matrix ***W*** = *[****w****1,…,****w****m]*, and using the biases b*i* as the elements of a bias vector b *=(*b*1,…,* b*m)*, we can write (2) in compact form

***g****(****x****) = (****W****·****u*** *+* b*)* (3)

where *(x1,…,xm) = ((x1),…,(xm))* is the *vectorial activation function* derived from the scalar activation function *()* used in (1). Based on such kind of layer function (2) an ANN can be defined as a composite function

***g****(****x****) =* ***g****M* °***g****M-1* ° *…* °***g****1 , M* ³ *1* (4)

where each ***g****i* is a layer function. For large *M* such an ANN is called a deep neural network (DNN).

Ein Bild, das Screenshot, Reihe, Muster, Diagramm enthält.

Automatisch generierte Beschreibung

Figure 2: Simple ANN and Deep Neural Network (DNN)

DNNs are now widely used as powerful building blocks, used for many different tasks to mimic “machine intelligence”. At this point we should, however, recognize, that DNNs can just perform a mapping operation, without the possibility of storing information.

For sequence memory, which requires permanent update of a *contextual state*, while the elements of a sequence are presented one after each other to the sequence memory, we need an extended mathematical model (5ab), which is commonly employed in control engineering and system theory.

***x****t+1 =* ***f****(****x****t,****u****t) (5a)*

***y****t =* ***g****(****x****t,****u****t) (5b)*

The model operates on a state ***x****t* Î , which follows a sequence of state transitions ***x****0* ***x****1* ***x****2 …* according to the *transition function* ***f***: (5a) depending on the input sequence***u****0* ***u****1* ***u****2 … (****u****t* Î *).* For given time *t,*  output ***y****t* Î depends then both on the combination of state ***x****t* and the presented input pattern ***u****t* according to the *output function* ***g***: (5b).

# Algorithm

Appendix

configure ***K,*** init ***X*** *0,* ***P*** *0*

repeat for *t = 0,1,2,3*

input ***u*** *t*

***Y*** *t = (****X*** *t + (1**-* ***1****m****max****(****X*** *t))° (****1****m* ***⋅ u*** *tT)*

***Q*** *t* = *σ(****D*** *t - η)* ° ***y*** *t(****K****)*

***X*** *t+1* = ([*||****Q****1 t ||1*, *||****Q****2 t ||1*, ... , *||****Q****n t||1*]*T ≥ θ)(m×n)*

***D*** *t+1* ***=*** ***D*** *t* ***+*** *(δ +****Q*** *t - δ -)* ° ***Y*** *t*