**Understanding HTM Sequence Memory**

Hugo Pristauz, Walter Eder – Oct. 2023

## *Abstract*

*This paper targets to be a useful aid for people who want to get detailed understanding of 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 been presented to the sequence memory.*

*In this paper a mathematical model of the HTM sequence memory algorithm is presented, which complies to the standard representation of nonlinear discrete time state representation, as it is commonly used in system and control theory. Further a toy network comprising a total of 40 neurons is presented, which allows to investigate and study the HTM sequence memory algorithm in every detail. Besides a formal description of the algorithm, we add a complete implementation in MATLAB code, to support jump start into a playground for HTM sequence memory.*

# Introduction

In his revolutionary 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 the capability to leveraging four abilities, which can be definitely 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.
* Creating a model of the world by physical (or mental) movement through the world, with the ability to build an understanding of the world by sensing sequential information of the real word.
* Simultaneous creation of multiple models of the world in order to make accurate predictions. The human brain is mainly a prediction machine, which benefits from the possibility to select the best prediction model of a pool of simultaneously entertained prediction models.
* Utilization of Reference frames for any kind of knowledge storage. Reference frames can be dynamically assembled, which, in consequence allows to “assemble” knowledge which is stored with respect to reference frames.

Capability 2 of this list (world understanding based on sequential sensor information) emphasizes the importance of *dealing with sequential information* (with both spatial and temporal attributes). In this sense *sequence memory*, as it is under focus in this paper, might be a powerful building block for required skill sets, as described above. 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, 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 2: 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 2/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,* as 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 activation 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 2/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.

# 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.

# Mathematical Model of the 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, Schrift enthält.

Automatisch generierte Beschreibung

Figure 3: input/output model of a cell (HTM neuron)

All quantities shown in figure 3 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* ***v*** *t* Î {0,1}N.

Further has the HTM neuron a state comprising

* the (scalar) binary cell state *xt*Î{0,1}
* the (vectorial) dendritic weights ***d*** *t*Î [0,1] *M*

with [0,1] denoting the closed interval{*x*Î  *|* 0£ *x* £1}. Since the tuple (*xt,* ***d*** *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 dendritic weight vector ***d*** *0*Î [0,1] *M*

which is sketched in figure 3 by the two input signals at the top. Finally, we have

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

In addition to inputs and outputs we have some cell parameters ***k***Î *(1,2,…,N)* and threshold values hÎ[0,1] and QÎ*{0,1,…,N},* which will be explained in more detail later*.*

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*)

# HTM Neuron Layer

To implement HTM sequence memory we need many *cells*, and we arrange them in terms of an *mn* matrix with n columns, each containing *m* cells (figure 4), resulting in a layer of total *N := mn* cells.

Ein Bild, das Text, Screenshot, Diagramm enthält.

Automatisch generierte BeschreibungFigure 4: HTM layer with a matrix of *m* *n* neurons

In HTM terminology each column of the layer matrix is called a *minicolumn*, and each of them is attached with a scalar binary input *ujt*Î{0,1}. In total the HTM layer receives *n* scalar inputs *ujt*, which are combined as the (binary) *input vector*

***u****t =* [ *u1t, u1t, … , unt* ]*T* Î {0,1}*n*  *(1)*

(with *T* denoting matrix transpose). *A*s figure 4 suggests, each cell of a particular minicolumn *j* receives the *same* binary input *ujt*. In a similar way we provide quantities *x**t*, *y**t* and ***d*** *t* of figure 3 with cell index *k* Î {*1, ... ,N* } and combine them to (binary) vectors ***x*** *t*, ***y*** *t* and (non-binary) matrix ***D*** *t.*

***x*** *t =* [ *x1t, x2t, … , xNt* ]*T* Î {0,1}*N* *(2a)*

***y*** *t =* [ *y1t, y2t, … , yNt* ]*T* Î {0,1}*N* *(2b)*

***D*** *t =* [ ***d****1t,* ***d****2t, … ,* ***d****Nt* ]Î [0,1]*MxN* *(2c)*

Note that the binary context input ***v*** *t* collects all cell outputs ***y*** *t* and is applied as a common context input to all cells (thus needs no augmentation). Formally we write ***v*** *t* = ***y*** *t*, which introduces a contextual feedback loop to the cell layer, as it is shown in figure 5.

*Ein Bild, das Diagramm, Screenshot, Reihe, Rechteck enthält.

Automatisch generierte Beschreibung*

Figure 5: input/output model of cell layer with contextual feedback loop

# Input Sequences

Let us now talk in more detail about how input sequences are presented to the cell layer of figure 5. Let us consider two sentences “Mary likes to sing.” and “John likes to dance.” To formally deal with these sequences, 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,* which 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: 4 x 10 Toy Layer with 5 x 2 synapses per cell

Let us define a 410 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*).

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

Automatisch generierte Beschreibung

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 permanence vector* ***d****kt* for the total set of distal segments. Broken down to a single distal dendritic segment these vectors can be split up into partial parameter vectors ***k****k1,* ***k****k2, …,* ***k****kd* and partial distal permanence vectors ***d****k1t,* ***d****k2t, …,* ***d****kdt* according to

***k****k =* [***k****k1T,* ***k****k2T, …,* ***k****kdT* ] *T* Î {*i* | 0 ≤ i ≤ N ∧ i ≠ *k*}*N* *(3a)*

***d****kt =* [***d****k1tT,* ***d****k2tT, …,* ***d****kdtT* ] *T* Î {*i* | 0 ≤ i ≤ N ∧ i ≠ *k*}*N (3b)*

Referring to figure 5 we have for example

***k****k =* [ *2 5 7 8 11 | 3 4 7 9 10* ]*T*

***d****kt =* [*0.1 0.5 0.8 0.3 0.7 | 0.2 0.8 0.7 0.3 0.9*]*T*

with partial vectors

***k****k1 =* [ *2 5 7 8 11* ]*T,* ***k****k2 =* [ *3 4 7 9 10* ]*T*

***d****k1t =* [*0.1 0.5 0.8 0.3 0.7*]*T,* ***d****k2t =* [*0.2 0.8 0.7 0.3 0.9*]*T*

The connection index vectors ***k****kµ* are telling us which other *cell outputs yi* are connected to synapses *of the distal dendritic segment µ*. As figure 5 demonstrates, segment 1 has *5* synapses which connect to cell outputs *y2*,  *y5*, *y7*,  *y8*, and *y11*, according to the index values contained in ***k****k1.* In a similar way the *5* synapses of segment 2 connect to cell outputs *y3*,  *y4*, *y7*,  *y9*, and *y10*, related to the index values contained in ***k****k2.*

Formally the connection vector***k****kµ* defines a projection *Pkµ*: {0,1}N {0,1}s given ***v*** *t* ***v*** *t(****k****kµ) =:* ***z****kµt.* In other words, from the total context ***v*** *t* a distal dendritic segment *µ* of cell *k* “sees” only the *s* values of the projected vector ***z****kµt*, which are picked according to the s index values defined by ***k****kµ*. Since the context vector ***v*** *t* is identical with layer output vector ***y*** *t*, we have also:

***z****kµ* = ***y*** *t(****k****kµ)* for all segments *µ* in cell *k*

If we continue the example shown in figure 5 by assuming

***y*** *t* = [*0 0 1 0 1 0 1 1 1 0 0 ...*]

then we get

***z****k1**t* = ***y*** *t (****k****k1) =* ***y*** *t* ([*2, 5, 7, 8, 11*]) = [*0 1 1 1 0*]*T*

***z****k2**t* = ***y*** *t (****k****k2) =* ***y*** *t* ([*3, 4, 7, 9, 10*]) = [*1 0 1 1 0*]*T*

for the projected context. Whether these projected cell outputs will all contribute to the excitation of the distal dendritic segment, depends on the following *synaptic rule:*

* A synapse contributes to the excitation of a distal dendritic segment, if, and only if

(i) the synapse is active (a binary value), which is given if permanence value of the synapse is greater than or equal the synaptic threshold

(ii) the synapse is connected to an active cell output, i.e.,  *ylt* = 1

Formally this can be modelled by introducing the binary synaptic vectors

***s****kµ**t =* (***d****kµ**t*) ***s****kµ**t* Î {0,1}*s (4)*

which has components equal to 1, if the relating component of permanence vector ***d****kµ* has a value greater or equal to synaptic threshold , and zero otherwise.

Based on the concept of the projected context ***z****kµ* and the synaptic vector ***s****kµ* we can define now the *coincidence* vector for distal dendritic segment *µ*

***q****kµt =* ***z****kµt*° ***s****kµt ,* ***q****kµ**t* Î {0,1}*s (5)*

with ° denoting element wise multiplication. If an element of the segment *coincidence* vector is 1, we know that both the permanence value of the related synapse is greater than or equal to synaptic threshold , and we also know that the specific (contextual) cell output to the related synapse is also active (having value 1). Otherwise, one of the two conditions is violated.

In the next section, where we present the full HTM algorithm for the cell layer in figure 5, we will see, that the segment activation vectors { ***q****k1,* ***q****k2, ... ,****q****kd*}, which can be summarized as the *segment* *coincidence* matrix of cell *k.*

***Q****k**t* = [ ***q****k1**t,* ***q****k2**t, ... ,****q****kd**t*] , ***Q****k**t*Î {0,1}*sxd (6)*

It will play a key role for both state updates of a cell and reinforced learning, i.e., adaption of the synaptic permanence values (figure 6). For a compact formulation of the HTM algorithm we also define the *layer coincidence matrix* as

***Q*** *t* = [ ***Q****1**t(:),* ***Q****2**t(:), ... ,****Q****d**t(:)*] , ***Q*** *t*Î {0,1}*MxN (7)*

Before we tackle the HTM algorithm, let us complete the sample description of figure 5. First we calculate the synaptic vectors for each segment µ, assuming the common setting of  *= 0.5.*

***s****k1**t =* (***d****k1**t*) = ([*0.1 0.5 0.8 0.3 0.7*] *0.5*) = [*0 1 1 0 1*]

***s****k2**t =* (***d****k2**t*) = ([*0.2 0.8 0.7 0.3 0.9*] *0.5*) = [*0 1 1 0 1*]

Knowing now which synapses are currently in action, we can calculate the segment excitation.

***q****k1t =* ***z****k1t*° ***s****k1t =* [0 1 1 1 0]*T* °[0 1 1 0 1]*T =* [0 1 1 0 0]*T*

***q****k2t =* ***z****k2t*° ***s****k2t =* [1 0 1 1 0]*T* °[0 1 1 0 1]*T =* [0 0 1 0 0]*T*

***Q****k**t* = [ ***q****k1**t,* ***q****k2*] =

# The HTM Algorithm

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

algorithm for sequence memory. The original form is 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*, ***D*** *0*

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

input ***u*** *t*

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

calculate ***Q*** *t* from ***D*** *t*, ***y*** *t* (coincidence matrix)

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

update ***D*** *t+1* from ***D*** *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 the *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*.

Finally keep in mind that *cell excitation* (referring to input *ukt*) and *segment excitation* (referring to distal dendritic segment excitation ***Q****kt*) are completely different concepts, and neither *cell excitation* implies necessarily *segment excitation*, or vice versa.

### 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:

***X*** *t =* ***x*** *t(mxn)*  reshape to ***X*** *t*Î{0,1}*mxn*  (8a)

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

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

***y*** *t* = ***C*** *t(:)* °***u*** *t* output activation (8d)

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(mxn)t + (1**-* ***1****m****max****(****x*** *t(mxn))))(:)°* ***u*** *t* (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.

***S*** *t* = *σ(****D*** *t - η)* synaptic matrix (10a)

***Z*** *t* = ***y*** *t(****K****)* projected context (10b)

***Q*** *t* = ***S*** *t* ° ***Z*** *t* coincidence matrix (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* (13)

### Step 4) Cell State Transition

In the last state the permanence values of the distal dentritic 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

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

# Appendix: Mathematical 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).