**The Neurotron, a Basic Neural Computing Unit**

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

## *Abstract*

*We introduce the* Neurotron*, a basic neural computing unit to build hierarchical temporal sequence memory as a basis for machine intelligence. Sequence memory is capable of learning sequences, and to predict the next item, when only part of a sequence is presented.*

*The* basic Neurotron*, with the absence of any learning mechanism, is a purely digital computing unit. It can be augmented with a learning mechanism based on analog valued synaptic permanences, which represent the development state of the modeled synapses. Such quasi-digital approach for an elementary neural computing unit in combination with a strict application of the fundamental paradigm of sparse pattern representations leads to highly efficient algorithms with fast execution time and low memory needs. A cluster of Neurotrons can implement spatial pattern separation by* sparsifying *and orthogonalization of spatial input patterns, as well as self-learning temporal sequence memory, predicting the completion of a given partial sequence.*

*The Neurotron implements a micro-circuit built from atomic terminals, dynamic blocks and gate logic, while terminals are comprising decoders and permanence-units. The Neurotron follows truly local asynchronous principles, synchroneity is triggered by change of input patterns. Local means, that Neurotrons as basic functional units cannot directly access states of other Neurotrons, except the activation state (the Neurotron’s output) , which mimics an action potential of a biological neu­ron’s soma distributed via its axon.*

*As the concept of the Neurotron has been strongly influenced by Numenta’s architecture of Hierarchical Temporal Memory (HTM), which employs a self-learning spatial pooler feeding to sequence memory, the underlying HTM key concepts have been incorporated into the Neurotron.*

*Finally, we benchmarked the performance of a Neurotron based sequence memory layer with the performance of transformer based neural network alternatives, when both systems predict the continuation of text sequences from "Tiny Shakes­peare".*

# Introduction

The neuron model used in most artificial neural networks is known as the perceptron [1], 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 that the proximal synapses, those closest to the soma (cell body), have a large effect on the likelihood of the cell generating an *action potential,* which is bursting along the neuron’s axon to synapses of other neurons of the network. Notably the perceptron model is a pure mapping model without presence of 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 [2].

After development of advanced research methods [3], however, re­searchers could unlock the secret of internal dendritic NMDA (N-methyl-D-aspartate) spikes. Caused by synaptic excitation within spatial and temporal neigh­bor­hood such NMDA spikes lead to a depolarization of the soma, which further enables the neuron to fire earlier than neighbor neurons with comparable proximal excitation. This gives such neuron the benefit to inhibit “competing” neurons.

Ein Bild, das Zeichnung, Diagramm, Text, Entwurf enthält.

Automatisch generierte Beschreibung

Figure 1: comparison of neuron models

Since NMDA spikes have a much longer duration than the action potential spikes of soma and axon, this mechanism acts like memory (based on a depolarized soma state). In addition to the (perceptron like) mapping functionality of the neuron, the availability of memory in neural models offers capabilities like *prediction* and *bursting*, introduced in [4].

Such and other findings make it evident that biological neu­rons are very com­plex systems with complex sub-func­tionalities in the soma, axon, trunk and dendritic segments. A comprehensive characterization splits neurons in com­part­ments, where each has its own computing capabilities involving spatio/temporal behavior of various ion concen­trations as well as electrical quan­tities. Some of this approaches found in the literature are shown in figures 1,2,3.

Ein Bild, das Cartoon enthält.

Automatisch generierte Beschreibung

Figure 2: Multi compartment model respecting local computations in different dendritic segments [9].

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

Automatisch generierte Beschreibung

Figure 3: Cortical microcircuit showing different input

kinds applied to a principal (pyramidal) neuron [6]

Such detailed models without further abstraction (often described as analog spiking models [6,7]) make it, how­ever, hard to understand behavior in higher levels, like explanations how the brain might model sequence memory. Thus, understanding of any given neuron requires the right level of abstraction [2].

Inspired by such findings regarding distal dendrite func­tion­ality a group around Jeff Hawkins developed the HTM approach [4,5], which we consider a powerful conceptual abstraction of an enhanced neural layer functionality serving as an abstract building block for truly intelligent algorithms, which can run efficiently on computers. It incorporates the following concepts:

* The fundamental paradigm of sparse pattern repre­sentation to implement pattern separation [4,5,6]
* *Quasi-digital processing*: most of the signals have binary values (0 or 1), except permanence values and synaptic thresholds (analog values in the interval [0,1]).
* *Binary synaptic weights*: causing a synapse to behave as unconnected (weight 0) or connected (weight 1). A weight is controlled by the analog *perma­nence state* of the synapse (range [0,1]), causing the weight to take value 1 if the *permanence* exceeds a synaptic threshold, and 0 otherwise.
* *Collaborations:* a collaborative group of neurons (in [4] called *minicolumn*) with a mechanism for “*voluntary neuron*” selection, which assigns a neuron with ownership to take over co-representation of an unexpected pattern.
* *Predictive neuron states:* enable those neurons of a collaboration to fire, which are expected to do so on base of the processed input pattern, while other neurons of the collaboration stay quiet.
* *Burst:* a state being activated during presentation of unexpected patterns. Bursting causes all neurons of a collaboration to vote for ownership of pattern co-representation.
* *Local learning:* the learning mechanism is Hebbian like. In contrast to Hebbian learning where weight adjustment is correlated with the product of pre-synaptic input and neuron output, HTM neurons are only empowered for learning, if they were able to make a correct prediction.

There are some notable remarks: The quasi-digital nature of the algorithms in combination with sparse pattern representations allows very fast and efficient processing on digital computing hardware. The capability of *bursting* in case of a new detected pattern increases the probability of finding voluntary neurons to take over co-representation ownership for unexpected patterns, which contributes to learning efficiency.

The HTM learning rule, where synapses are only credited for learning, when a neuron made a correct prediction, provides strong learning focus on neurons with actual involvement in the prediction process. It helps to avoid ruinous overwriting of well-established memory contents, which is a key challenge in the formulation of suitable synaptic plasticity hypothesis [8].

The HTM algorithm presented in [4] is formulated for a neuron-layer comprising separated *minicolumns* as an array of “HTM-neurons”. The concept implies that each neuron is part of exactly one *minicolumn*. We replaced the concept of *minicolumn* by the concept of *collaboration*, where Neurotrons belong to one or more collaborative groups, which, in the biological equivalent, is more likely than a strict organization in terms of separate *minicolumns*.

Also, the description in [4] leaves it fuzzy how the algo­rithm might be implemented on the granularity of neurons. Such approach is reasonable, when the intention is to provide building blocks for higher level cognitive functionality. When we designed the *Neurotron*, however, we wondered how the HTM algorithm could be broken down to the granularity of neurons.

While in [4] the basic computing unit is the *minicolumn*, we are going to introduce an even more elementary computing unit, the *Neurotron*, which can be used to build either *minicolumns* on the next higher level, or in a more general form to build *collaborations* as part of a cluster layer. Alternatively, we expect that *Neurotrons* can be the building blocks for higher level functional units like the *Spatial Pooler* consulted in the HTM approach, or an abstract model of the pattern separation functionality of the dentate gyrus modeled in [6], although both applications have not yet been studied in detail.

# The Neurotron in a Nutshell

Now we are going to introduce the *Neurotron* in a top-down approach. There are four atomic functional units employed by the Neurotron:

* *Decoders*: representing the dendritic capability of coincidence detection, which is a pure binary mapping functio­nality (without memory).
* *Dynamic Blocks:* digital abstractions of dynamic beha­vior of neuronal compartments (soma, dendritic seg­ments. We use a unified parametrizable *pulse unit* to implement a dynamic block.
* *Gate Logic:* a combination of elementary logic func­tions like AND, OR and NOT with pure mapping functionality (excluding memory functions).
* *Permanences:* analog valued, matrix organized memo­ry representing the development state of synapses. Each row of a permanence matrix corresponds to synapses of a dendritic segment.

The *Basic Neurotron*, which does not support a learning mechanism, comprises only three of the listed atomic units, which have pure digital character (*decoder, dynamic block and gate logic*). The *augmented* or *general Neurotron* employs also the *permanence* unit, which has analog operation. Thus, we call a (*general*) *Neurotron* network *quasi-digital*.

A *decoder, optionally* augmented with *permanences* is called a *terminal* of the *Neurotron*. A terminal represents a compartment of a biological neuron where axons of other neurons arrive at neighbored synapses. Considering the permanence state as constant in a given time slice, a *terminal*, notably, has pure mapping functio­nality, i.e., the coincidence detection functionality of the underlying *deco­der*.

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

Figure 4: schematics of the *Basic* *Neurotron*

Before explaining the details of the employed atomic building blocks let us have a look on the overall schematics of the *Basic Neurotron* (figure 4). Nota­bly, the aim is not to present the *Neurotron* as a proven model of a given neu­­ral micro­circuit, rather the *Neurotron* is formulated as a *generic computation unit inspired by neural mechanisms*, which is capable to implement high level functions like *pooling* and self-learning sequence memory.

More specifically we claim that a cluster of *N* properly parametrized *Basic* *Neuro­trons* according to figure 2

* are able to perform a *pooling* function, i.e., a mapping of non-sparse binary patterns into sparse binary pat­terns maintaining sematic relationships
* are able to implement a sequence memory, where each *Neurotron* operates *asynchronously* and *locally* with only access to the outputs of other Neurons in the collection, but no access to other internal state variables of other neurons.
* Are able to perform a state observation from the Neurotron outputs, i.e., are able to reconstruct if any Neurotron in a given group is in a prediction state in order to suppress a *burst condition* (as it is described in [4]).

Furthermore, the *Basic Neurotron* algorithm can be aug­mented with a true local (Hebbian style) learning mecha­nism, operating in a quasi-digital mode, which implies to augment some of the terminals with adaptable *perma­nences*.

# Dynamic Blocks (Pulse Units)

*Dynamic blocks* are used in the Neurotron model to represent (binary) state. In neurobiology it has been observed that action potentials are aggregated by neuronal compartment to potentially exhibit spikes or plateau potentials, which have leading delays, plateau effects and relaxation phases, during which a reactivation is temporarily suppressed. We introduce a parametrizable time discretized digital *pulse unit* (see section “materials and methods” for a detailed description) as a universal dynamic block denoted by

*output sequence = Pulse(input sequence, lag, duty, relax)*

where the integer parameter *lag* models the leading delay (in discrete time units) during which an integration of the (binary) input signals happen. Once the integrated value reaches a threshold, which equals the lag parameter, the *lag state* of the pulse unit transitions into the *duty state,* with activated output (binary 1) over a number of discrete time units corresponding to the *duty* parameter. In the simple case (third parameter *relax* is zero), the output falls back to zero after the duty period (figure 3b,d), except the pulse unit is being retriggered (figure 3c). In the absence of bouncing input, the pulse unit behaves like a delayed re-triggerable mono flop.

Figure 5e shows the behavior of input bouncing (lag=2, duty=3, relax=0). The requirement of 3 (lag+1) conse­cutive one-inputs is initially not achieved, thus the integrator falls back from intermediate level 2 to level 1 (not shown), which needs two additional one-inputs to reach the integration threshold, causing finally a duty state transition to hold the output at level one (over 2duty steps), before flipping back to zero (figure 5e). Thus, the integrator at the input side of the pulse unit performs a debouncing function, which is important for the Neurotron during periods where the patterns applied to the Neuron’s decoders are floating.

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

Automatisch generierte Beschreibung

Figure 5: dynamic block represented as pulse unit in action

In figure 5f,g a pulse unit with activated relaxation (delay=1, duty=2 and relax=3) is shown in action. After each duty period (length 2) a relaxation period of length 3 must follow, during which the output is kept zero.

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

Automatisch generierte Beschreibung

Figure 6: finite state machine implementing a dynamic block represented by a pulse unit

Figure 5g also demonstrates, that re-triggering is ignored in case of a non-zero relax parameter.

A finite state machine implementing a pulse unit to serve as dynamic block is shown in figure 6. At the input side (input *u*) it has a discrete integrator which increments counter x for u = 1, and decrements counter x for u = 0, maintaining the limitation *0 ≤ x ≤ l.* This integrator also debounces the input for lag time *l > 0*. Besides of counter x, which determines the end of the LAG phase, there is another counter c, to control the duration of the DUTY or RELAX phase. Depending on the values of u, x and c two auxiliary variables y’ and c’ are calculated, which, together with machine state s ∈ {LAG, DUTY, RELAX}, control the ac­tions of the state machine according to the state transition diagram in figure 6, which also set the output *y* properly.

The state machine of figure 6 is a synthetical construct to meet given input/output behavior shown in figure 5 and has no direct inspiration from neurobiological structures, except to mimic neurobiological pulse dynamics with initial delay (lag), duty phase (plateau) and an optional relaxation phase during which input signals are temporarily ignored.

# Decoders

Digital decoders are functional units used in digital electronic circuits to select subunits in case of dedicated binary patterns applied to the decoder input. If a subunit should be selected by the binary pattern ***w*** *= [1,0,1,0]* and the decoder input is given by the binary vector ***v*** *= [v1,v2,v3,v4]* , then the decoder operation is a mapping

s =  *v1* ∧ ¬ *v2* ∧ *v3* ∧ ¬ *v4*

This decoding scheme discriminates reliably between the intended address pattern **a**, and any other possible pattern. If, however, the set of possible patterns contains sparse representations with big hamming distance (number of differing bits) between patterns, then the following relational expression (**.** denoting scalar product)

*s = ( [v1,v2,v3,v4]* ***.*** *[w1,w2,w3,w4] ≥ Θ ) =*

*= ( v1w1 + v2w2 + v3w3 + v4w4  ≥ Θ )*

does the same job. E.g., consider ***w*** *= [1,0,1,0]* as before and the set of sparse patterns comprising all vectors v with two bits at maximum set. Then, given threshold *Θ=2,* the only vector causing s = 1 is ***v*** *= [1,0,1,0] =* ***w****.* The scalar product of binary vectors calculates the *overlap* of two vectors, the number of matching bits. If we characterize a decoder by a binary weight vector **w**, then the decoder sets its output to 1 for all input patterns which have sufficient overlap *≥ Θ* with the decoder’s weight vector **w**.In other words, the patterns must be *sufficient similar* to the decoder’s weight vector ***w*** in terms of sufficient small Hamming distance. The smaller *threshold* *Θ* is chosen the more patterns will be included in the set of similar patterns.

For our decoders a symbol used for a McCulloch-Pitts neuron is employed, which is aug­mented with a parameter input for the weight vector (figure 6b). A set of decoders operating on a set of weight vectors ***w****k* can be combined to a decoder set parametrized with weight matrix ***W*** and operating on a set of patterns represented by matrix ***V*** (fi­gure 6c). Such decoder set is used for modelling a set of dendritic segments.

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

Automatisch generierte Beschreibung

Figure 6: Decoder symbols based on the digital neuron model of McCulloch and Pitts in 1943

The detailed mathematical description of a decoder or decoder set can be found in section “Materials and Methods”.

# The Neurotron

Equipped with these pre-requisites we can tackle now the design of the *Neurotron*. The *Neurotron* is a basic neural computing unit, serving as an atomic building block for HTM sequence memory (in the sense of [4]), for a *spatial pooler (*which is an autonomous learning map to maintain given sparsity by preserving semantics), and possibly for other higher-level functionality which has not yet been defined. The *Neuro­tron* is a well-defined abstraction of a pyramidal neuron, assigned with a well-defined algorithm deter­mining its computing process.

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

Automatisch generierte Beschreibung

Figure 5: basic states of the *Neurotron* and fundamental *Neurotron* relation

For the *Neurotron* four basic states are defined. We use the color codes in figure 5 (top row) to indicate the *Neurotron* to be in a specific state. Different states can be set con­currently in combination, which is demonstrated figure 6.

* *Excited (u = 1)*: The *Neurotron* mimics a biological neuron’s condition, where the aggregation of feed­forward signals arriving at proximal synapses is strong enough to enable firing an action potential. Whether this actually happens, depends on additional conditions.
* *Active (y = 1)*: The *Neurotron* mimics a biological neuron’s state in firing condition, i.e., the soma fires an action potential propagating along the axon.
* *Predictive (p = 1):* The *Neurotron* mimics a biological neuron’s condition where the sufficient aggregation of action potentials arriving at distal synapses at previous time caused some dendritic segment to evoke an NMDA spike, which depolarized the soma. In such condition the neuron “predicts” an upcoming (feed-forward) excitement
* *Depressed (d = 1)*: The Neurotron mimics a biologi­cal neuron’s condition where some collaborating neurons are trying to prevent the neuron from firing.

The logical dependency between states *u, y, p* and d is formulated as the *fundamental Neurotron relation* (see also figure 5) and defined as follows:

* The *Neurotron* gets active when it is

1. either both excited and predictive
2. or depressed after certain time of excitement

The formal description of the *fundamental Neurotron relation* is formulated in (4)

*y t = u t ° p t or u t-T ° not d t  (4)*

See figure 6 for the application of the *fundamental Neurotron relation* to several specific scenarios.

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

Automatisch generierte Beschreibung

Figure 6: several examples demonstrating the application of the *fundamental Neurotron relation*

The next step in the development of the *Neurotron* architecture is to add three so- called *terminals* to feed state variables *p*, *u*, and *d* which are implemented as McCulloch-Pitts models. We call these *terminals*:

* *prediction terminal* (related to *prediction state p*)
* *excitation terminal* (related to *excitation state u*)
* *depression terminal* (related to *depression state d*)

This leads us to the Neurotron core, which is shown in figure 7.

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

Automatisch generierte Beschreibung

Figure 7: *Neurotron core*

Our claim is that the *Neurotron core* is already capable of sequence prediction, when all its terminals are provided with proper synaptic weights. This is demonstrated in the following example.

## Example 1: “Sarah loves music”

This “toy” example demonstrates how a 3-*Neu­rotron* network can predict the next item, when parts of the sequence <Sarah loves music> are presented. Our initial assumption is that the words Sarah, loves and music are re­presented by *none-sparse* feed-forward patterns ***f****i* *∈ B 10*

Sarah  ***f****1 =* [*1, 1, 0, 1, 1, 1, 0, 1, 0, 1*]

loves  ***f****2 =* [*0, 1, 1, 1, 0, 1, 1, 0, 1, 1*]

music  ***f****3 =* [*1, 1, 1, 0, 0, 1, 0, 1, 1, 1*]

and the task for our three *Neurotrons* is to map these feed-forward patterns into sparse patterns ***u****i* *∈ B* *3.* The follo­wing map assigning the basis vectors of *B* *3* will do such job.

***f****1* ***u****1 =* [*1,0,0*]*,* ***f****2* ***u****2 =* [*0,1,0*]*,* ***f****3* ***u****1 =* [*0,0,1*]

Next we need to find weight vectors ***w****u1,* ***w****u1,* ***w****u3* and thre­sholds *Θu1, Θu1, Θu3* which implement the proposed map­pings. This will work for ***w****ui =* ***f****i* and *Θui ∈ {6,7}, i = 1..3.* For a cross-check let the feedforward pattern be ***f*** *=* ***f****1,* which empowers the excitation terminals of our 3 *Neuro­trons* with *eu1 = Σ(****f****1°* ***w****1)* = 7, *eu2 = Σ(****f****1°* ***w****2)* = 4, *eu3 = Σ(****f****1°* ***w****3) = 5,* implying u1 = 1, u2 = 0, u3 = 0 by choice of threshold *Θui = 6 (or Θui = 7).* If we let***f*** *=* ***f****2,* the *excitation terminals* are empowered with *eu1 = Σ(****f****2°* ***w****1)* = 4, *eu2 = Σ(****f****2°* ***w****2)* = 7, *eu3 = Σ(****f****2°* ***w****3) = 5,* implying u1 = 0, u2 = 1, u3 = 0 as desired. Finally with ***f*** *=* ***f****3,* we have *eu1 = Σ(****f****3°* ***w****1)* = 5, *eu2 = Σ(****f****3°* ***w****2)* = 5, *eu3 = Σ(****f****3°* ***w****3) = 7,* which sets our input states to u1 = 0, u2 = 0, u3 = 1, which finally proofs our assertion that our feedforward sequence items map sparsely into our excitation space *B 3*

Sarah [*1, 0, 0*]*,* loves [*0, 1, 0*]*,* music [*0, 0, 1*] o

In the next step we choose the dimensions of the prediction (***W****pi* *∈ B* 2 x 3) and depression terminal (***w****di* *∈ B* 3) and define the network wiring. Our approach is to collect all *Neurotron* outputs *yk* to the *context vector*

*c =* [ *y1, y2, y3* ]

and feed the context vector back to all terminals (the prediction terminal requires two times feed of the context vector, as the matrices ***V****pi* *∈ B* 2 x 3 have two rows). This procedure sets up a generic connection topology assuring that each terminal segment has connected synapses with all *Neurotron* outputs (in our case 3), thus the effective connection topology is left to the choice of weight vec­tor/matrix settings (figure 8).

Define now the following network states at given time *t*,

***u****t =* [*u1t, u2t, u3t*] *∈ B l network excitation (state)*

***y****t =* [*y1t, y2t, y3t*] *∈ B l network activation (state)*

***p****t =* [*p1t, p2t, p3t*] *∈ B l network prediction (state)*

***d****t =* [*d1t, d2t, d3t*] *∈ B l network depression (state)*

initialize all network states *(****u****0=****y****0=****p****0=****d****0=****0****)*, clear also all weights of the *prediction terminal* (***W****p1=****W****p2=****W****p3=****0***) and *depression terminal* (***w****d1=****w****d2==****w****d3=****0***), and watch the *Neurotron* network doing its job by presenting item by item of the sequence <Sarah loves music>.

Ein Bild, das Diagramm enthält.

Automatisch generierte Beschreibung

Figure 8: Connection topology of the 3-*Neurotron network*

When ***f****1* (the feedforward pattern for the first sequence item Sarah) is presented to the network, the excitation terminals encode ***f****1* into the *network excitation*

***u****1 =* [*u10, u20, u30*] *=* [*1, 0, 0*]

Since initially all values of the network prediction are zero and the zero weights of the prediction terminal prevent any change the network activation ***y***will remain in the zero state. Also, the network depression will remain in the zero state, caused by the zero weights of the depression terminal.

***d****1 =* [*0, 0, 0*]

***p****1 =* [*0, 0, 0*]

***y****1 =* ***u****1 °* ***p****1 or* ***u****0 ° not* ***d*** *1 =* [*0, 0, 0*]

Ein Bild, das Screenshot, Text, Kreis, Design enthält.

Automatisch generierte Beschreibung

Figure 9: “Network state movie”: Sarah loves music

# Materials and Methods

Figure 6a shows a McCulloch-Pitts neuron model with sca­lar binary inputs *v1, v2, …, vl ∈ B* , *B* *:={0,1},* and a scalar output *s ∈ B*. Notably the input/output functionality is a pure (memory-less) mapping with­out any presence of state. As will figure out, it is useful to split the input/output mapping according to (1a) into a composition of two partial functions *( ° )*.

*s = ( ° )(v1, v2, …, vl) (1a)*

*e = (v1, v2, …, vl) = Σi vi (1b)*

*s = (e) = (e ≥ Θ) (1c)*

In the first part the inputs *v1, v2, …, vl ∈ B,* aresummed up to the *empowerment* signal *e*, an internal quantity (1b). In the second step the *empowerment e* is mapped to output *s* by taking the logical result of comparing the *empowerment e* against the *spiking-threshold* *Θ (1c)*, suggesting the advice: “if the unit is sufficiently empowered, then it shall spike”. The basic McCulloch-Pitts model can be augmented by combining *v1, v2, …, vl* to *input vector* ***v*** *(2a)*, intro­ducing the digital weight vector ***w****∈ B,* and modifying *(1b)* to let the empowerment *e* be now a weighted sum of the input vector’s elements *(2c)*, where ***w*** ° ***v*** denotes elementwise product.

***v*** = (*v1, v2, …, vl )∈ B l (2a)*

***w*** := (*w1, w2, …, wl )∈ B l (2b)*

***e*** *= (****v****) =* ***w*** ° ***v*** *(2c)*

*s = (****e****) = (Σei ≥ Θ) (2d)*

The motivation for such augmentation (see also figure 4b) is to be prepared for the introduction of a learning mecha­nism, which allows to change an unconnected synapse (*wi* *= 0*) to a connected synapse (*wi* *= 1*) and vice versa.

Notably all signals and synaptic weights introduced so far are binary. Further, the augmented McCulloch-Pitts model (2a-d) is not intended to be applied to an entire neuron model. Rather we will use the augmented McCulloch-Pitts model neuron compartments, which are related to ta dendritic segment or a group of dendritic segments.

Since we have to model multiple neuron compartments (related to different dendritic segments), it makes sense to introduce a set of decoders (according to figure 4c) to represent a multitude of synaptic regions. The related equations are (3a-d), where *d* scalar outputs *si* are combined to output vector ***s***, and *d* input vectors ***v***i are combined to form the rows of input matrix ***V***. In a similar way this scheme is applied to *l* vectors ***e****i ,* ***w****i ,* which form the rows of matrices ***E****,* ***W****.*

***V*** = (***v****1,* ***v****2, …,* ***v****d )T ∈ B  d x l (3a)*

***W*** := (***w****1,* ***w****2, …,* ***w****d )T ∈ B  d x l (3b)*

***E*** *= (****V****) =* ***W*** ° ***V*** *∈ B  d x l (3c)*

***s*** *= (****E****) =* [*Σ* ***e****i ≥ Θ* ] *∈ B  d (3d)*

# References

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

[2] Major, G., Larkum, M. E., and Schiller, J.: “Active properties of neocortical pyramidal neuron dendrites”; *Annual Review of Neuroscience* 36, 1–24 (2013).

[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] Hawkins J., Ahmad S.: „Why Neurons Have Thousands of Synapses, a Theory of Sequence Memory in Neocortex“; *frontiers in Neural Circuits (2016)*.

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

[6] Chavlis S., et all: Dendrites of Dentate Gyrus Granule Cells Contribute to Pattern Separation by Controlling Sparsity. *Hippocampus 27:89-110* *(2017).*

[7] Gidon A., et all: Dendritic Action Potentials and Computation in Human Layer 2/3 Cortical Neurons. Science 367, 83-87 (2020).

[8] Magee J.C., Grienberger C.: Synaptic Plasticity Forms and Functions. *Annual Review of Neuroscience 46, 95-117 (2020).*

[9] McCulloch W.S., Pitts W.: A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics,* 5, 115-133 (1943).

[10] Winding M., et all: The connectome of an insect brain. *Science 379, eadd9330* (2023).

[11] Larkum M.E., et all: Synaptic integration in tuft dendrites of layer 5 pyramidal neurons: a new unifying principle. *Science 325:756-60* (2009).

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

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

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

[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*.