**The Neurotron, a Basic Neural Computing Unit to Implement Sequence Memory**

Hugo Pristauz, Walter Eder, Willy Ottinger – Nov. 2023 (Draft v0.4)

## *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. We also demonstrate that a cluster of Neurotrons can process spatial pattern separation by* sparsifying *and orthogonalization of spatial input patterns. Further we construct a Neurotron cluster to operate as self-learning temporal sequence memory.*

*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.*

*The Neurotron implements a micro-circuit built from a set of elementary building blocks: terminals, dynamic blocks and gate logic, while terminals are comprising detectors and permanence-units. These building blocks are used in the Neurotron architecture but can be used to compose other abstractions of neural microcircuits. The introduction of universal digital pulse units to mimic spiking and plateau dynamics might be useful for future digital modelling of neural microcircuit abstractions.*

*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.*

*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 Text, Diagramm, Screenshot, Design 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 processing memory (based on a depolarized soma state), which is comparable to register memory in microprocessors. 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 into com­part­ments, where each has its own computing capabilities involving spatio/temporal behavior of various ion concen­trations as well as electrical quan­tities. As studies have shown the information flow between soma and other compart­ments can be bidirectional (figure 2, [9]). 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:

* *Detectors*: 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 (*detector, 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 *detector, 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 *detec­tor*.

# Ein Bild, das Diagramm, Text, Plan, Reihe 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*.

The Neurotron operates on changing patterns of two input vectors, *feedforward vector* ***f*** *∈ B M* (*B* denotes the binary set *{0,1}*), which encodes the sequence tokens to be pro­cessed, and *context vector* ***c*** *∈ B N,* which collects the outputs of all *Neurotrons* in a processing cluster, including the *Neurotron* itself and those of the collaborative groups. Formally ***f*** and ***c*** are composed to *output vector* ***y***, which represents the outputs of *all Neurotrons* being involved in the processing scheme. Since the Neurotron output *y* is part of *output vector* ***y***, it will update the corresponding element *yk* of ***y*** = (… , *yk*, …) continuously.

# 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 3 duty 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 detectors are floating.

Ein Bild, das Text, Screenshot, Diagramm, Zahl 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. Figure 5g also demonstrates, that re-triggering is ignored in case of a non-zero relax parameter.

Figures 5h,i,j demonstrate that a parameter choice of lag=0, duty=1 relax=0 leads to an identity block which passes an input sequence unchanged to the output. This means that dyna­mic blocks can be deactivated just by proper choice of para­meters.

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.

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

# Detectors

Detectors have the role of coincidence detection, which is very similar to the role of digital decoding in electronic circuits to select subunits (like a processor register or I/O port) by application of a proper address at the decoder’s input. If such subunit should be selected by the binary address ***a*** *= [1,0,1,0]* according to a given decoder input ***v*** *= [v1,v2,v3,v4]* , then the decoder operation is a mapping

s =  *v1* ∧ ¬ *v2* ∧ *v3* ∧ ¬ *v4 .*

This decoding scheme discriminates reliably an intended address pattern **a** from and any other possible pattern. Such approach can be adopted for the detection of sparse patterns. The 1-norm of a binary vector x *∈ B n* is defined as *(1)*

||***x***||1 = ||*(x1 , x2 ,… xn)*||1 := *Σ i |xi* | = *Σ i xi* = <***x****|****x***>, *(1)*

with <***a****|****b***> denoting scalar product. We write ***x*** *≤* ***w*** *iff* (if and only if) *||****x****||*1 *≤ ||****w****||*1, and we call <***w****|****x***>the overlap of the two vectors **w** and **x**. Given a weight vector ***w*** *∈ B n* and a threshold *Θ* we define a *detection* *target* as the set

*D(****w****, Θ )* := {***x*** *∈ B l* | ***x*** *≤* ***w***∧ <***w****|****x***>*≥* *Θ* } *(2)*

In other words, the *detection* *target D(****w****, Θ)* is the set compri­sing all binary vectors ***x*** *≤* ***w*** (number of non-zero bits of ***x*** must not exceed those of ***w***) with overlap <***w****|****x***> greater than or equal to threshold *Θ.* A *detection target D(****w,*** *Θ)* induces a map

*f: B l → B* with ***x*** *(****x*** *∈ D(w, Θ)), (3)*

i.e., a *detector* triggers (outputs a logical 1) *iff* the input pattern ***x*** *≤* ***w*** is part of its *detection target.* The smaller the threshold *Θ* is chosen, the bigger is the *detection target (set)*, and the bigger is the chance for a pattern close to the reference pattern of the *target* (given by weight vector ***w***) to trigger the detector. An important consequence is that all vectors of a *detection target* have similar semantic mea­ning. To complete the picture, there are also input vectors ***x*** >***w*** outside of a *detection target*, ***x*** *∉ D(w, Θ),*which can also trigger the detector*.* Under a paradigm of sparsity, however, such cases are considered as pathological and expec­ted to occur very rarely.

Coming back to the electronic example: if we deal only with sparse patterns *||****v****||*1 *≤ 2,* then choosing ***w*** *=* (1, 0, 1, 0) and *Θ = 2* defines a *detection target* *D((1,0,1,0), 2)* = {(1,0,10)} for a detector. Such detector triggers only for one single input pattern ***v*** *=* ***w***, thus it is equivalent to the electronic decoder above. Lowering the threshold to *Θ = 1* induces a much bigger *detection target*:

*D((1,0,1,0), 1)*  = {*(1,0,1,0), (1,0,0,0), (0,0,1,0),*

*(1,1,0,0), (1,0,0,1), (0,1,1,0), (0,0,1,1)*} ,

implying to trigger on additional patterns, which are similar to the reference pattern determined by weight vector ***w***.

The functionality introduced by our *detectors* has some similarity with the classic McCulloch-Pitts model of a neuron, introduced in 1943 (figure 6a), when only those inputs are connected to the neuron model which relate to a one in the weight vector (for ***w*** = (1,0,1,0) => inputs *v1* and *v3*). Our detector be considered as an augmented Mc­Culloch-Pitts model with weight vector ***w*** as an additional parameter (figure 6b).

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

Automatisch generierte Beschreibung

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

While the mapping of a McCulloch-Pitts model is des­cribed by (1a-b),

*e = (v1, v2, …, vl) = Σi vi (1a)*

*s = (e) = (e ≥ Θ) (1b)*

the detector employs the augmented in­put/output mapping (2a-d) with ° denoting elementwise multiplication.

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

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

***e*** *= (****v****) =* ***w*** ° ***v*** *∈ B l (2c)*

*s = (****e****) = (Σei ≥ Θ) ∈ B l (2d)*

Weight vectors are a convenient way to describe a given connection topology. As an example, let us assume a clu­ster with 10.000 Neurotrons, each having a binary output *yk* (k = 1 … 10000), represented as output vector

***y*** *= (y1, y2, … y10000) ∈ B 10000*.

If we want to model a detector with 3 inputs v1, v2, v3 with the following connection topology

y2 → v1, y6 → v2, y8 → v3

we would just define a detector mapping from *f:B 10000* → *B* and a weight vector

***w*** *= ( 0, 1, 0, 0, 0, 1, 0, 1, 0, 0, 0, …) ∈ B 10000.*

A set of *detectors* operating on a set of weight vectors ***w****k* can be combined to a *detector set* parametrized with weight matrix ***W*** and operating on a set of patterns represented by matrix ***V*** (fi­gure 6c). Such *detector set* is useful for model­ling a set of dendritic segments. It implements the map­ping (3a-d).

***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)*

We call vector e in (2c) the empowerment vector with *(4)* defining the overlap of vectors **v** and ***w***,

||***e***||1 = ||***v***°***w*** ||1 = <***v****|****w***> *= Σ i |ei* | = *Σ i ei* *(4)*

and call *s* the spiking signal, which equals 1 *iff* the overlap

|| ***e*** ||1 exceeds threshold *Θ.* We say thata *detector spikes,* if it outputs a non-zero spiking signal, which has the equivalent meaning that the decoder has detected an input pattern *coincidence* with the *detection target*.

In a similar way we call ***E*** in (3c) the *empowerment matrix* of the *detector*, and vector ***s*** in *(3d)* the *spiking vector*, collecting the *spiking signals* of each *detector* of the set.

# Neurotron States and Core Logic

We are going to explain now the *basic states* and the *core logic* of the *Neurotron* (figure 7).

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

Automatisch generierte Beschreibung

Figure 7: basic states and core logic of the *Neurotron*

The meaning of the *basic states* is as follows.

* *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.
* *Stimulated (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 activation under a predictive condition. If the Neurotron is not predictive, stimu­lation is not sufficient for Neurotron activation.
* *Excited (q = 1)*: If stimulation is sufficiently long, the *Neurotron* gets excited. After some time in excited state a Neurotron, which is not depressed, starts bursting
* *Depressed (d = 1)*: The Neurotron mimics a biologi­cal neuron’s condition where some collaborating neurons are trying to prevent the neuron from firing. In a *depressed* state a Neurotron is not able to *burst*.
* *Bursting (b = 1)*: A non-depressed Neurotron in excited state will start bursting after some delay. This in turn causes *activation*, i.e., setting the output signal active.
* *Predictive (p = 1):* The *Neurotron* mimics a biologi­cal neuron’s condition where 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) stimulation with subsequent excite­ment and activation.

The logical dependency between states *u, d, x, y, q, b* and y is formulated as the *Neurotron core logic* (see also figure 7), defined as follows:

* A *Neurotron* gets active when it is

1. either both stimulated and predictive
2. or excited and not depressed

The formal description of the *Neurotron core logic* is for­mu­lated in *(5a-c)*.

*q = Q(u) (5a)*

*b = B(q ° not d) (5b)*

*y = Y(u ° x or b)  (5c)*

See figure 8 for different scenarios following the *Neurotron core logic*

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

Automatisch generierte Beschreibung

Figure 8: several examples demonstrating the application of the *Neurotron core logic*

# Sequence Memory

The following “toy” example demonstrates how a 3-*Neu­rotron* network can predict the next item, when parts of the sequence <Sarah loves music> are presented. In this example the words Sarah, loves and music are re­presented by *none-sparse* feed-forward patterns ***f****i* *∈ B 10* (called *tokens*).

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*]

The first tasks for our 3 *Neurotrons* is to map these feed-forward patterns into sparse orthogonal *stimuli* ***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)*

From the definition of a *detector*, we know that an input pattern must belong to its *focus set* in order to trigger. Thus, the goal is to find parameters of the excitation terminal (weight vectors ***w****u1,* ***w****u1,* ***w****u3* and thre­sholds *Θu1, Θu2, Θu3*) to satisfy the following equations:

***f****1 ∈ F(****w****u1, Θu1),* ***f****1 ∉ F(****w****u2, Θu2),* ***f****1 ∈ F(****w****u3, Θu3)*

***f****2 ∉ F(****w****u1, Θu1),* ***f****2 ∈ F(****w****u2, Θu2),* ***f****2 ∉ F(****w****u3, Θu3)*

***f****3 ∉ F(****w****u1, Θu1),* ***f****3 ∉ F(****w****u2, Θu2),* ***f****3 ∈ F(****w****u3, Θu3)*

An easy check proves that this works for ***w****ui =* ***f****i* and *Θui ∈ {6,7}, i = 1..3.* With such parameters the excitation terminals implement the following map, which performs both sparsifying and orthogonalization [6].

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

In the next step we define the framework conditions, under which our 3 *Neurotrons* are operating: All *Neurotron* outputs *yk* are collected by the *context vector* ***c****,*

***c*** *= ( y1, y2, y3 )*

which, together with *feedforward vector* ***f***, forms the *network output vector* ***y***.

***y*** *= (****c****,* ***f****) = ( y1, y2, y3, f1, f2, ... , f10 )*

According to figure 4 each Neurotron k decomposes the *network output vector* ***y*** into ***c*** and ***f*** and feeds them accordingly to the Neurotron's *terminals*. After processing the *detection* operation in the *terminals* and running the attached *pulse dynamics*, the *Neurotron* runs its *core logic* in order to processes the new output signal *yk*, which is finally updated in the *network output vector* ***y*** *= (..., yk, ...).*

In the next step we define the network structure, including the definition of *collaborations* (collaborative Neurotron groups). For the demonstrations in the current example, we do not need collaborative groups. Thus, we define the *weight vectors* ***w****ci ∈ B* 3 of the *collabo­ration terminal* as zero vectors.

***w****c1 =* ***w****c2 =* ***w****c3 = (0,0,0)*

For the prediction terminal we choose a *detector set* with two *detectors*, requiring the *prediction weight matri­ces* to be ***W****pi* *∈ B* 2 x 3, which we setup as follows:

1) when the first item Sarah is presented to the network, the network's *excitation terminals* map token *f1 to stimulus* ***u*** *= (1,0,0) which causes Neurotron 1 to fire, causing* ***c*** *=* ***u****1 = (1,0,0).* However, there is nothing for *Neurotron 1* to predict, since Sarah is the first item of the input sequence. Thus, we set

***W****p1 =* [ 0, 0, 0; 0, 0, 0].

2) On the other hand, *Neurotron 2* detects itself to become *predictive* by stimulus ***u****1 = (1,0,0)* (encoding Sarah) causing ***c*** = ***u****1*, since *Neurotron 2* has to fire when stimulus ***u****2 = (0,1,0)* (en­coding loves)is presented to the network, , causing ***c*** = ***u****2*. This is achieved by setting one of the rows of ***W****p2* to match ***u****2*, e.g.:

***W****p2 =* [ ***u****1 ;* ***0*** ] = [ 1, 0, 0; 0, 0, 0]

3) In a similar way Neurotron 3, encoding music repre­sen­ted by ***u****3 = (0,0,1)*, has to get predictive when ***u****2 = (0,1,0)* (en­coding loves) is detected, causing ***c*** = ***u****2*. Thus, we can choose

***W****p3 =* [ ***u****2 ;* ***0*** ] = [ 0, 1, 0; 0, 0, 0].

Remark: It does not matter which *detector* of the *prediction terminal's* *decoder set* does the prediction. Thus, the choices ***W****p2 =* [ ***0****;* ***u****1*], ***W****p3 =* [ ***0****;* ***u****2*], or even ***W****p2 =* [ ***u****1;* ***u****1*], ***W****p3 =* [ ***u****2;* ***u****2*] would also do the job.

Figure 9 shows the network in action when item Sarah is presented to the initialized network (all states zero). Each column of figure 9 corresponds to a certain time slice (discrete time progresses from left to right), while the *i-*th row shows state progression of *Neurotron i*. Only *Neuro­tron 1* decodes a *stimulation* (blue) when Sarah is presen­ted, which leads after two steps to *excitation* (violet) and *bursting* (orange), finally to *activation* (red). Since the other two *Neurotrons* detect no stimulus, a stimulus pattern ***u****1 = (1,0,0)* ends with a context ***c*** *=* ***u****1 = (1,0,0)* in phase "burst".

Ein Bild, das Screenshot, Kreis, Schrift, Farbigkeit enthält.

Automatisch generierte Beschreibung

Figure 9: network state transition after presentation of first sequence item Sarah

The second row of figure 9 shows the state transition of *Neu­ro­tron 2*. Without stimulus there is neither excitation, bursting or activation. Once *Neurotron 1*, however, acti­vates its output (column "burst") the *prediction terminal* of *Neurotron 2* detects context ***c*** *=* ***u****1* (Sarah) and becomes predictive (green). The white vertical bar symbolizes a *predictive spike*, which mimics an NMDA-spike in neurobiology, depolarizing the soma [2,3,4]. Since the pulse unit of the predictive state is parametrized with a long duty time (figure 4, duty = 8), the predictive state will last until the next sequence item is presented. Neurotron 3 stays passive all the time, since it neither decodes a stimulus, nor does its prediction terminal detect a pattern to trigger.

Figure 10 shows the state transition progress when item loves (token ***f****1*) is presented to the network. In this case only Neurotron 2 decodes a stimulus. Since *Neurotron 2* has been set predictive (green) in the previous phase, the stimulus will immediately set the output active (red) in phase "react". A combination of *prediction* and *activation* can be interpreted as a correct prediction, which (in case of an *augmented Neurotron*) would trigger synaptic *learning* in the *prediction* terminal, which is indicated by the yellow vertical bar*.*

Ein Bild, das Screenshot, Kreis, Schrift, Farbigkeit enthält.

Automatisch generierte Beschreibung

Figure 10: network state transition after presentation of second sequence item loves

In subsequent iterations (phases "excite", "burst") *Neuro­tron 2* gets excited and bursting, which, however, has no consequence for the network output since the output of *Neurotron 2* has been already activated in phase "react". *Neu­rotron 1* stays passive (no decoded stimulus, no detec­ted prediction pattern), while in phase "react" (when *Neu­ro­tron 2* activates its output) *Neurotron 3* gets predictive (predicting item music), when it detects context ***c*** = ***u***2 = (0,1,0) encoding loves.

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

Automatisch generierte Beschreibung

Figure 11: network state transition after presentation of third (and last) sequence item music

Finally, figure 11 shows the state transitions of the network after presentation of music, the last item of the sequence. While Neurotrons 1 and 2 stay passive Neurotron 3 runs a similar state transition pattern like Neurotron 2 in the previous cycle.

# 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).

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

[13] Major G., et all: Spatiotemporally Graded NMDA Spike/Plateau Potentials in Basal Dendrites of Neocortical Pyramidal Neurons. *Journal of Neurophysiology (Junie, 2008)*

[14] Brette R., Gerstner W.: Adaptive Exponential Integrate-and-Fire Model as an Effective Description of Neural Activity.

[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*.