**The Neurotron, an Atomic Neural Computing Unit**

Hugo Pristauz, Walter Eder, Willy Ottinger – Nov. 2023 (Draft v0.1)

## *Abstract*

*We introduce the* Neurotron*, an atomic 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 Neurotron is a quasi-digital computing unit, which means that the most of signal processing is purely digital. Only the learning process is based on analog* permanences*, which represent the development state of the modeled synapses. The quasi-digital approach in combination with a strict application of the fundamental paradigm of sparse pattern representations lead to highly efficient algorithms with fast execution time and low memory needs.*

*As the concept of the Neurotron has been strongly influenced by Numenta’s architecture of Hierarchical Temporal Memory (HTM), 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 with absence of any memory.

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

Automatisch generierte Beschreibung

Figure 1: comparison of neuron models

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.

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 would have to involve spatial and temporal behavior of various ion concentrations as well as electrical quantities.

Such detailed models without further abstraction 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
* *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.
* *Minicolumn*: a kind of collaborative group of neurons 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 minicolumn to fire, which are expected to do so on base of the processed input pattern, while other neurons of the minicolumn stay quiet.
* *Burst:* a state being activated during presentation of unexpected patterns. Bursting causes all neurons of a minicolumn 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 [6].

The HTM algorithm presented in [4] is formulated for a neuron-layer comprising Minicolumns as an array of “HTM-neurons”. However, the description 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 “molecular” computing unit is the *minicolumn*, we are going to introduce an “atomic” computing unit, the *Neurotron*, which can be used to build *Minicolumns* on the next higher level, but also to build other next level building blocks like a *Spatial Pooler* (another functional unit consulted in the HTM approach).

# The McCulloch-Pitts Model

Most artificial neural networks (ANN) are based on Rosenblatt’s Perceptron [1], which work with analog inputs, outputs and weights. The advantage of analog signals based ANNs is that global learning can be em­ployed, using gradient based backpropagation algorithms.

The authors of [4], however, have demonstrated, that local learning mechanisms, as they occur in biological neurons [6], lead to very efficient learning algorithms, and allow a further abstraction of neuronal functionality by using quasi-digital signal representation. Such approach is compliant with the claim to describe the salient input/output properties of a neuron using a minimal des­crip­tion [2].

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]

Thus, we concluded that the 80-years-old computation model proposed by McCulloch and Pitts in 1943 (figure 4a) is well suited for the HTM approach presented in [4] and for the design of our Neurotron. However, the McCulloch-Pitts model is not applied to the entire neuron, rather to some compartment of a neuron, which is related to a dendritic segment. Without going to details, we refer to several proposals in the literature [2][6][9] how to split a pyramidal neuron into compart­ments representing mea­ning­ful sub-functionalities (figure 2,3).

With this background in mind, we come back to the McCulloch-Pitts model, which is consulted in this context for the des­cription of the input output functionality of some neuron’s compartment (not the entire neuron).

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

Automatisch generierte Beschreibung

Figure 4: augmented neuron model based on the proposal of McCulloch and Pitts in 1943

Figure 4a 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 neuron 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 an array of McCulloch-Pitts models (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)*

# 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* relationship

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) ° x(t) or u(t-T) ° not d(t) (4)*

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

Automatisch generierte Beschreibung

Figure 6: several examples demonstrating the logical dependency between states *u(t), = u(t-T), p(t) and y(t)*

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

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

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

[9] 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*.

# Algorithm

Appendix

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

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

input ***u*** *t*

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

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

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

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