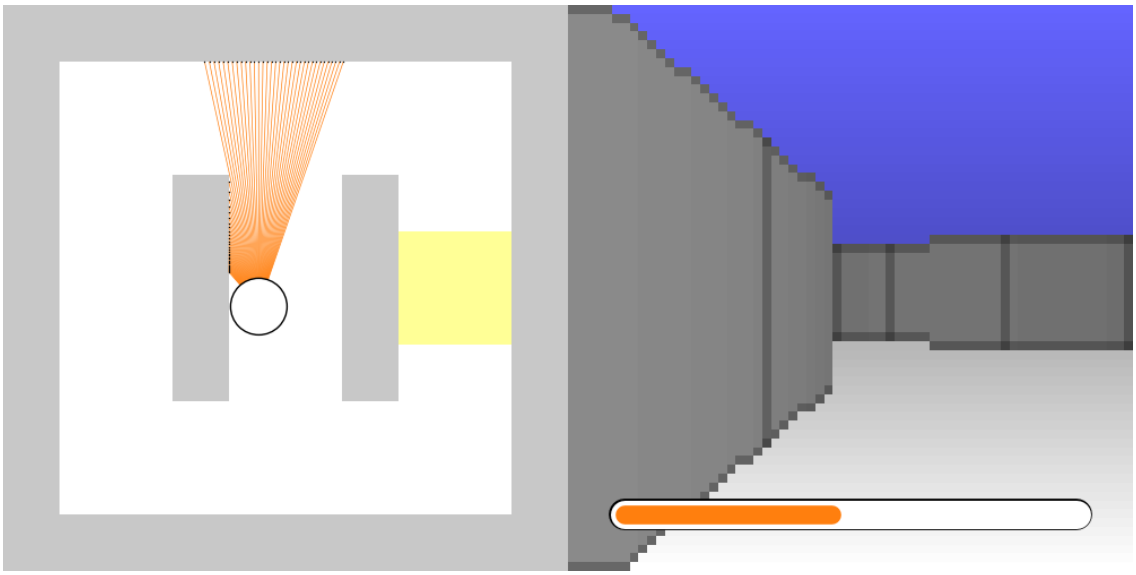
Contents


```
  Ongoing:
+ Bug dans softmax
+ Souci dans new variable
+ Valider programmatoid.sh
```

1 The braincraft challenge

Let us consider the following digital experimental setup, as described in braincraft challenge presentation[1].

1.1 Simplified problem statement



The braincraft challenge[2] bot moves at a constant with sensor inputs and one orientation output, it uses some energy and refill this energy on a given yellow location. The 2D space size is $[0, 1] \times [0, 1]$. The bot starts in the middle and oriented at 90, i.e., upward.

Input variables	
$p_l \in [0_{\text{wall-hit}}, 1_{\text{no-wall}}[$	Leftward proximity, max value of the [0, +30°] range 32 left sensors.
$p_r \in [0_{\text{wall-hit}}, 1_{\text{no-wall}}[$	Rightward proximity, max value of the [-30°, 0] range 32 right sensors.
$c_{l\bullet} \in \{0, 1\}, \bullet \in \{b_{\text{blue}}, r_{\text{red}}\}$	Leftward binary red and blue color detectors.
$c_{r\bullet} \in \{0, 1\}, \bullet \in \{b_{\text{blue}}, r_{\text{red}}\}$	Rightward binary red and blue color detectors.
$g_e \in [0_{\text{death}}, 1_{\text{full}}]$	Energy gauge value.
Output variable	
$d_o \in [-5, 5]$	Orientation difference, saturated at ± 5°.

With respect to the original braincraft challenge:
- we consider two leftward and rightward "average" sensors only,
- color is input as binary variables, and always available,
- the wall hit indicator is not used.

1.2 Input preprocessing

1.2.1 Proximity sensors

Motivation Simplifies the left-right navigation by compacting the leftward and rightward sensors as a simple pair of input.

[1] https://github.com/rougier/braincraft/blob/master/README.md#introduction
[2] https://github.com/rougier/braincraft

Implementation A simple sum or average could be used, thus using directly a linear combination of the input in afferent units.

We also can use a softmax mechanism as derived in Appendix 11, enjoying a neuronoid approximation, and allowing to balance between averaging and computing the maximum.

1.2.2 Color sensors

Motivations Again, color blob detection is to perform either on the left or on the right, allowing the color input to be compacted. For each color, a channel is specified, simplifying the programmatoid implementation and providing an input closer to biological colored vision. Since the setup color input is a discrete color index, the channel value is binary, accounting for the presence, or not, of a least one related color index.

Implementation For the distributed implementation, each camera color index value is mapped on each color channel input with the 0 value if the color is different and the 1 value if equal, e.g., using step unit:

$$\begin{array}{rcl} c_{\dagger\bullet} & \leftarrow & H(\sum_{k\in K}(1-D(i_\bullet - c[k]))), c[k]\in\mathcal{N} \\ & & \dagger\in\{l_{\text{left}}, r_{\text{right}}\}, \bullet\in\{b_{\text{blue}}, r_{\text{red}}\} \\ D(x) & \stackrel{\text{def}}{=} & H(x-1/2)+H(-x-1/2)) = \text{ if } |x|<1/2 \text{ then 0 else 1} \end{array}$$

where K stands for the left or right sensor related indexes, $i_\bullet$ stands for the color index, and $c[k]$ stands for the sensor color index value.

1.2.3 Energy measurement

Motivation The energy increase accumulation is to be pre-processed, since used in some task.

Processed Variables

$g_{c\flat}\in[0,1], \flat\in\{1,2\}$	$g_{c\flat}\vert_{t=0}=0$	Last and last-before-last cumulative energy increases.
$g_{e\flat}\in[0,1], \flat\in\{1,2\}$	$g_{e\flat}\vert_{t=0}=0$	Last and last-before-last energy value.

Here instantaneous energy increase is $(g_e - g_{e1})$, and we assume that the energy always changes so that $g_e \neq g_{e1}$.

Implementation

Cumulating energy increase starts, saving last increase in g_{c2}.

if $\underbrace{g_e > g_{e1} \text{ and } g_{e1} < g_{e2}}$ then $g_{c2}\leftarrow g_{c1},\ g_{c1}\leftarrow(g_e - g_{e1})$

increase after a decrease

Cumulating energy increase continues.

if $\underbrace{g_e > g_{e1} \text{ and } g_{e1} > g_{e2}}$ then $g_{c1}\leftarrow g_{c1}+(g_e - g_{e1})$

increase after an increase

Otherwise $g_e < g_{e1}$, thus cumulating energy increase stops, and g_{c1} is memorized.

$g_s \stackrel{\text{def}}{=} g_e < g_{e1}$ and $g_{e1} > g_{e2}$ indicates that cumulating energy increase has just stopped.

$g_n \stackrel{\text{def}}{=} g_e < g_{e1}$ and $g_{e1} < g_{e2}$ indicates that cumulating energy increase did stop before.

Pseudo programmatoid solution

$$\begin{array}{rcl} g_{c1} & \leftarrow & g_{c1} - H(g_{e2}-g_{e1})\, g_{c1} + H(g_e - g_{e1})\,(g_e - g_{e1}) \\ g_{c2} & \leftarrow & g_{c2} + H(H(g_e - g_{e1}) + H(g_{e2}-g_{e1}) - 3/2)\,(g_{c1} - g_{c2}) \end{array}$$

Then

$$\begin{array}{rcl} g_{e2} & \leftarrow & g_{e1} \\ g_{e1} & \leftarrow & g_e \end{array}$$

which is not a pure programmatoid solution because, because of term of the form $H(u)\,v$, thus with a product, but is implementable in the neuronoid framework discussed in the sequel. In brief:

$$H(x)\,y \simeq \omega'\,h(y/\omega' + \omega\,h(\omega\,x) - \omega)$$

where $h(\cdot)$ is the sigmoid function, and ω and ω' are sufficiently large numbers.

1.3 A putative controller

1.3.1 Navigation

Heuristic : The bot runs at constant velocity,

(i) ahead by default, thanks to a linear correction, high enough to correct the direction, small

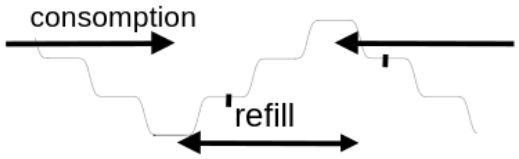


Figure 1: Representation of energy profile when cumulating energy increase starts (first tick), with $g_e > g_{e1} < g_{e2}$, and stops (last tick), with $g_e < g_{e1}$, while during refill $g_e > g_{e1} > g_{e2}$.

enough to avoid oscillations, and

(ii) attempts to perform a quarter-turn in the preferred orientation as soon as passing is detected.

With this navigation mechanism the bot performs either leftward or rightward half-loops, traversing the central corridor.

Internal variable

$$q_p \in \{0_{\text{leftward}}, 1_{\text{rightward}}\}, \quad q_p|_{t=0} = 0 \quad \text{Preferred quarter-turn direction.}$$

Programmatoid solution :

$$\begin{array}{rcl} d_o & \leftarrow & \underbrace{\gamma\,(p_l - p_r)}_{\substack{\text{linear correction to}\\ \text{maintain direction ahead}}} + \underbrace{\alpha\,(t_l - t_r)}_{\substack{\text{quarter- turn}\\ \text{left or right}}} \\ t_l & \leftarrow & (1-q_p)\,H(\beta - p_l) = H(H(\beta - p_l) - q_p - 1/2) \\ t_r & \leftarrow & q_p\,H(\beta - p_r) = H(H(\beta - p_r) + q_p - 3/2) \end{array}$$

where:

$$\begin{array}{rcll} w & \simeq & 1/4 & \text{Rough estimation of the path-width.} \\ \gamma & = & \frac{5}{w/2} & \text{Saturates the correction at 5° if the depth difference is half of the path-width.} \\ \alpha & = & 90 & \text{Saturates the correction at } \pm 90° \text{ to make the quarter-turn, since the linear } |\gamma\,(p_l - p_r)| < 40 \text{ is lower than the quarter-turn term, the latter submut the former.} \\ \beta & = & w & \text{Triggers the quarter-turn if the depth is higher than the path-width.} \\ \omega & = & 10 & \text{Constant to transform a boolean product to a step-function threshold.} \end{array}$$

while:

$$\begin{array}{lllll} t_l = 1 & \text{iff} & q_p = 0 & \text{and while} & \beta < p_l \\ t_r = 1 & \text{iff} & q_p = 1 & \text{and while} & \beta < p_r \end{array}$$

in words: we execute the quater-turn until another wall is detected[3].

We thus have a linear output unit for d_o and two step-unit for t_l and t_r.

Given this controler the design reduces to navigation direction choice, i.e., control the q_p variable.

1.3.2 Task 1: Simple decision

Strategy Restrict navigation to the half-loop that contains the energy source, while the other does not.

Heuristic If the energy is too low, thus looping in the wrong direction, the direction is changed once.

At start $q_p = 0$. Then, if the energy is too low it changes once to $q_p = 1$.

$$q_p \leftarrow \text{if } q_p = 1 \text{ or } \eta > g_e \text{then1 else 0}$$

Programmatoid solution

$$q_p \leftarrow H(\omega\,q_p + (\eta - g_e))$$

where:

[3]The indenties:

$$\begin{array}{rcl} (1-q_p)\,H(\beta - p_l) & = & H(H(\beta - p_l) - q_p - 1/2) \\ q_p\,H(\beta - p_r) & = & H(H(\beta - p_r) + q_p - 3/2) \end{array}$$

are easy to verify using a simple truth table.

c	$=$	$1/1000$	Energy consumption at each step.
s	$=$	$1/100$	Speed: location increment at each steps.
b	$=$	$3/2$	Distance bound between the starting point and the putative energy sources.
η	$\simeq$	$b\,c/s = 3/20$	Energy consumption threshold if no source on the path.

1.3.3 Task 1b: Simple decision but varying environment

Strategy Restrict navigation to the half-loop that contains the energy source, while the other does not, this may change with time.

Heuristic

- If the energy is too low, the direction is inverted.
-This is registered, avoiding multiple changes at low energy.
-When the energy is high enough, change registration is reset.

Internal variable

$g_c \in \{0,1\}$ $\quad g_c|_{t=0} = 0$ Registers if the low energy has been detected.

Programmatoid solution

Inverts the direction if to be changed.
$q_p \leftarrow$ if $\eta > g_e$ and $g_c = 0$ then $1 - q_p$ else q_p
Registers the inversion until the energy is high enough.
$g_c \leftarrow$ if $g_c = 0$ and $\eta > g_e$ then 1 elif $g_c = 1$ and $2\,\eta < g_e$ then 0 else g_c

In the sequel, we are going to derive a generic way to compile such an expression with binary values as a programmatoid .

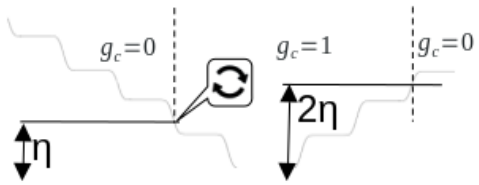


Figure 2: Representation of g_c value at different level of energy with the indication of the direction change.

1.3.4 Task 2: Cued environment decision

Strategy Restrict navigation to the half-loop without a closed path, as indicated by a color that has already been seen once.

Heuristic Detect and store the first color blob, and choose to turn in the direction it appears again. The detection is reset when the energy decreases.

Internal variable

$c_{c\bullet} \in \{0,1\}, \bullet \in \{b_{\text{blue}}, r_{\text{red}}\}$ $\quad c_{p\bullet i}|_{t=0} = 0$ Detected the color cue code, if any.

Programmatoid solution

Detect the cue, if not yet done, and reset below an energy threshold
$c_{cb} \leftarrow$ if $c_{cb} = 0$ and $c_{cr} = 0$ and ($c_{lb} = 1$ or $c_{rb} = 1$) then1 elif $g_e < \eta/2$ then 0 else c_{cb}
$c_{cr} \leftarrow$ if $c_{cb} = 0$ and $c_{cr} = 0$ and ($c_{lr} = 1$ or $c_{lr} = 1$) then1 elif $g_e < \eta/2$ then 0 else c_{cr}
Set the direction according to the cue
$q_p \leftarrow$ if $c_{lb} = c_{cb}$ or $c_{lr} = c_{cr}$ then 0 elif $c_{rb} = c_{cb}$ or $c_{rr} = c_{cr}$ then 1 else q_c

1.3.5 Task 3: Valued environment decision

Strategy and Heuristic Test energy sources and change direction if the latter yields less increse than the former.

Programmatoid solution

$$q_p \leftarrow \text{if } \underbrace{g_e < g_{e1} \text{ and } g_{e1} > g_{e2}}_{\substack{\text{energy increase} \\ \text{just stopped.}}} \text{ and } \underbrace{g_{c2} > g_{c1}}_{\substack{\text{previous energy} \\ \text{increase is higher.}}} \text{ then } 1 - q_p \text{ else } q_p$$

2 Programmatoid computation

We name "programmatoid" computation the conception of an input-output straight-line program[4] implementing an operator on numerical value expressions using an LN-system with the step-function as non-linearity, this writing:

$$o_n[t] = H(\textstyle\sum_{m\in\{1,M\}} w_{nm}\, i_m[t-1] + w_{n0}), n \in \{1, N\}$$

where $i_m[t]$ is the m-th input at a discrete t and $o_n[t]$ is the n-th output, including recurrent system with some output corresponding to inputs, while $w_{nm}, n \in \{1, N\}, m \in \{0, M\}$ are the systems parameters, or, weights for $m > 0$ and bias for $m = 0$.

For the sake of simplicity if the left hand size correspond to a value at time t and right-hand size to values at $t - 1$, which will be always here, and if not informative, temporal indexes are omitted, i.e. the previous equation will be written:

$$o_n \leftarrow H(\textstyle\sum_{m\in\{1,M\}} w_{nm}\, i_m + w_{n0}), n \in \{1, N\}$$

The step-function, also called Heaviside function, implements the computation of the sign of a value:

$$\begin{aligned} H(x) &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if} \quad x > 0 \\ 1/2 & \text{if} \quad x = 0 \\ 0 & \text{if} \quad x < 0 \end{cases} \\ &= \text{if } x > 0 \text{ then } 1 \text{ elif } x = 0 \text{ then } 1/2 \text{ else } 0. \end{aligned}$$

The design choice of $H(0) = 1/2$, instead for instance $H(0) = 0$, this latter simplifying some formula, is due the neuronoid approximation developed in the sequel.

By extension, we use, for any property $\mathcal{P}$, the notation:

$$H(\mathcal{P}) \stackrel{\text{def}}{=} \text{if } \mathcal{P} \text{ then } 1 \text{ else } 0$$

2.1 Comparison implementation

Given two numerical variables v_1 and v_2, using the notation , we have the equivalence[5], considering $true$ as 1 and $false$ as 0:

$H(v_1 > v_2)$	$H(v_1 \leq v_2)$	$H(v_1 = v_2)$	$H(v_1 \neq v_2)$	
not $H(v_1 \leq v_2)$	$H(H(v_1 - v_2))$	$H(v_1 \leq v_2)$ and $H(v_2 \leq v_1)$	not $H(v_1 = v_2)$	with $H(0) = 1/2$
$H(v_1 - v_2)$	not $H(v_1 - v_2)$	not $H(v_1 \neq v_2)$	$H(v_1 > v_2)$ or $H(v_2 > v_1)$	with $H(0) = 0$

while $H(v_1 < v_2) = H(v_2 > v_1)$ and $H(v_1 \geq v_2) = H(v_2 \leq v_1)$. We thus can implement any numerical comparison as a programmatoid, providing that we can implement boolean expressions, as given now.

2.2 Boolean expression implementation

Given binary variables $b_n \in \{0_{false}, 1_{true}\}, n \in \{1, N\}$ we have the obvious[6] correspondence:

[4] `https://en.wikipedia.org/wiki/Straight-line_program`

[5] Considering the pseudo truth table, with $H(0) = 1/2$:

	$H(v_1 > v_2)$	$H(v_1 = v_2)$	$H(v_1 < v_2)$
$H(v_1 - v_2)$	1	1/2	0
$H(H(v_1 - v_2))$	1	1	0
$H(H(v_2 - v_1))$	0	1	1
$H(H(v_1 - v_2)) + H(H(v_2 - v_1)) - 1$	0	1	0

we obtain the expected results.

[6] Easy to verify with, e..g., a truth table, and by induction for the generalized formula.

b_1	b_1 and b_2	b_1 or b_2	not b_1
$H(b_1 - 1/2)$	$b_1\, b_2 = H(b_1 + b_2 - 3/2)$	$H(b_1 + b_2 - 1/2)$	$1 - b_1 = H(1/2 - b_1)$

and more generally:

$$\begin{aligned} \text{and}_{n\in\{1,N\}} &= H\left(\textstyle\sum_{n\in\{1,N\}} b_n - N + 1/2\right) = \prod_{n\in\{1,N\}} H(b_n - 1/2) = \prod_{n\in\{1,N\}} b_n \\ \text{or}_{n\in\{1,N\}} &= H\left(\textstyle\sum_{n\in\{1,N\}} b_n - 1/2\right) \end{aligned}$$

An interesting consequence is that binary variable products can be translated to a programmatoid.
Let us also notice that all arguments x of the step function $H(\cdot)$ still verify $|x| \geq 1/2$.

2.3 Conditional expression implementation

A conditional expression on variables on any numerical type $v_n, n \in \{0,1\}$ with a binary variable $b_1 \in \{0,1\}$ writes:

$$\begin{aligned} v &\leftarrow \text{if } b_1 \text{ then } v_1 \text{ else } v_0 \\ &= (1 - b_1)\, v_0 + b_1\, v_1 \end{aligned}$$

while, for binary variable, i.e., if and only if $v_n \in \{0,1\}, n \in \{0,1\}$:

$$\begin{aligned} &= H(v_0 + (1 - b_1) - 3/2) + H(v_1 + b_1 - 3/2) \\ &= H(v_0 - b_1 - 1/2) + H(v_1 + b_1 - 3/2) \\ &= H(H(v_0 - b_1 - 1/2) + H(v_1 + b_1 - 3/2)) \end{aligned}$$

as easy to verify, using for instance a truth table. Here:
- products with $(1 - b_1)$ and b_1 behaves as switches between v_o and v_1
- when v_i are binary, we are left with a two layer computation, the first layer being built from two programmatoid, and the second layer from either another programmatoid or a simple linear unit, i.e., a linear combination of values.

This generalizes to conditional expressions on variables on any numerical type $v_n, n \in \{0, N\}$: [7]:

$$\begin{aligned} v &\leftarrow \text{if } b_1 \text{ then } v_1 \,[\text{ elif } b_n \text{ then } v_n]_{n\in\{2,N\}} \text{ else } v_0 \\ &= \textstyle\prod_{n'\in\{1,N\}} (1 - b_{n'})\, v_0 \\ &+ \textstyle\sum_{n\in\{1,N\}} b_n \prod_{\substack{n' \in \{1,N\},\\ n' \neq n}} (1 - b_{n'})\, v_n \\ &= \textstyle\sum_{n\in\{0,N\}} h_n\, v_n \\ \text{with } h_0 &\stackrel{\text{def}}{=} H(1/2 - \textstyle\sum_{n'\in\{1,N\}} b_{n'}) \in \{0,1\} \\ \text{and } h_n &\stackrel{\text{def}}{=} H(b_n - \textstyle\sum_{n'\in\{1,N\}, n'\neq n} b_{n'} - 1/2) \in \{0,1\}, n \in \{1,N\} \end{aligned}$$

while, for binary variable, i.e., if and only if $v_n \in \{0,1\}, n \in \{0,N\}$:

$$\begin{aligned} &= H(v_0 - \textstyle\sum_{n'\in\{1,N\}} b_{n'} - 1/2) \\ &+ \textstyle\sum_{n\in\{1,N\}} H(v_n + b_n - \sum_{n'\in\{1,N\}, n'\neq n} b_{n'} - 3/2). \end{aligned}$$

Let us also notice that all arguments x of the step function $H(\cdot)$ again verify $|x| \geq 1/2$.

As a consequence,

- A N-term conditional expression on binary variables reduces to a two layers programmatoid of N and 1 unit, the former unit being either linear of with a step-wise function.

- A N-term conditional expression on numerical variables reduces to a two layers system of N programmatoid and an output unit with h_n exclusive switches.

[7] By induction, considering the second line, one one hand, due to the products, if $b_n = 0, n \in \{1, N\}$ we obtain v_0. On the other hand, if $b_k = 0, k < K$ and $b_K = 1$, due the products, we obtain v_K, which is precisely the semantic of the conditional expression of the first line.
The 3rd line is dediced from the second line, transforming the binary products on the corresponding programmatoid while:

$$\begin{aligned} h_0 &\stackrel{\text{def}}{=} H(\textstyle\sum_{n'\in\{1,N\}} (1 - b_{n'}) - N + 1/2) \\ &= H(1/2 - \textstyle\sum_{n'\in\{1,N\}} b_{n'}) \\ h_n &\stackrel{\text{def}}{=} H(b_n + \textstyle\sum_{n'\in\{1,N\}, n'\neq n} (1 - b_{n'}) - N + 1/2) \\ &= H(b_n - \textstyle\sum_{n'\in\{1,N\}, n'\neq n} b_{n'} - 1/2) \end{aligned}$$

The 4rd line integrates the v_n variables in the programmatoid sum, since they are binary, and provides the same obvious algebraic reduction as for the 3rd line.

3 Neuronoid computation

3.1 Neuronoid unit

By "neuronoid" we name the not very biologically plausible[8] simplest biological neuron or neuron small ensemble inspired by mean-field modelisation[9] of the Hodgkin–Huxley neuronal axon model[10].

We define the equation[11]:

$$\begin{array}{rl} \tau \frac{\partial v_i}{\partial t}(t) + v_i(t) = z_i(t), & z_i(t) \stackrel{\text{def}}{=} h\left(\sum_j w_{ij}\, v_j(t) + w_{i0}\right), \\ h(x) \stackrel{\text{def}}{=} \frac{1}{1+e^{-4\,x}} & = h(x_0) + sech(2\,x_0)^2\,(x-x_0) + O((x-x_0)^2), \\ & = 1 - e^{-4\,x} + O(e^{-8\,x}) \end{array}$$

where v_i is the membrane potential, so that:

$$\begin{array}{rcll} v(t) & = & 1/\tau \int_0^t e^{-(t-s)/\tau}\, z(s)\, ds + v(0)\, e^{-t/\tau} & \\ & = & z(0) + e^{-t/\tau}\,(v(0) - z(0))\big|_{z(t)=z(0)} & \text{(constant input)} \\ & = & z(t)|_{\tau=0} & \text{(no leak),} \end{array}$$

and the corresponding discrete approximation using an Euler schema writes:

$$\begin{array}{rcll} & \simeq & (1-\gamma)\, v(t-\Delta t) + \gamma\, z(t-\Delta t)) & \\ & = & \sum_{s=0}^{t-1} \gamma\,(1-\gamma)^{t-s+1}\, z(s) + v(0)\,(1-\gamma)^t & \\ & = & z(0) + (1-\gamma)^t\,(v(0) - z(0))|_{z(t)=z(0)} & \text{(constant input)} \\ & = & z(t)|_{\gamma=1} & \text{(no leak).} \end{array}$$

with $0 < \gamma < 1$ and $0 < \tau$ in correspondence:

$$\gamma \stackrel{\text{def}}{=} 1 - e^{\frac{1}{\tau}} \Leftrightarrow \tau = -\frac{1}{\log(1-\gamma)}, \text{while} \lim_{\gamma \to 0} \tau = +\infty, \lim_{\gamma \to 1} \tau = 0.$$

Here, $h(\cdot)$ is the normalized sigmoid with

$$h(-\infty) = 0, h(0) = 1/2, h'(0) = 1, h(+\infty) = 1, h(x) = 1 - h(-x)$$

All this is just very standard derivations, available as `Maple` code[12], of the vanilla neuron model, as already proposed by [Lapicque, 1907].

We call "neuronoid computation" the conception of an input-output transform based on feed-forward and recurrent combination of neuronoids, as defined previously.

3.2 Step-function mollification

The step-function approximates sigmoid with a huge slope at zero, i.e.:

$$\forall x \neq 0, H(x) = \lim_{\omega \to +\infty} h(\omega\, x),\, h'(\omega\, x)|_{x=0} = \omega$$

while the convergence is also obtained for $v = 0$ in the distribution sense with $H(0) = h(0) = 1/2$. More precisely, the $\mathcal{L}_1$ error magnitude, on $]-\infty, +\infty[$, writes:

$$|\epsilon_{H,\omega}(x)|_{\mathcal{L}_1} = \frac{\log(2)}{2}\,\frac{1}{\omega}, \epsilon_{H,\omega}(x) \stackrel{\text{def}}{=} H(x) - h(\omega\, x)$$

while:

$$|\epsilon_{H,\omega}(x)| = e^{-4\,\omega} + O\left(e^{-8\,\omega}\right)$$

It has been noticed that, except for numerical comparisons, all arguments x of the step function $H(\cdot)$ verify $|x| \geq 1/2$, and the related error is neglictible as soon as, say, $\omega \geq 10$:

ω	1	2	5	10	20	50
$\epsilon_\omega(\pm 1/2)$	0.119	0.0180	$0.455\,10^{-4}$	$0.206\,10^{-10}$	$0.426\,10^{-19}$	$0.372\,10^{-45}$

Any programmatoid computation involving the $H(\cdot)$ function can thus be approximated by a neuronoid, with $\tau = 0$, and sufficiently large ω.

Derivations are available as `Maple` code[13].

[8] `https://en.wikipedia.org/wiki/Biological_neuron_model#Relation_between_artificial_and_biological_neuron_models`

[9] `https://inria.hal.science/cel-01095603v1`

[10] `https://en.wikipedia.org/wiki/Hodgkin-Huxley_model`

[11] We also have:

$$h(x) \stackrel{\text{def}}{=} \frac{1}{1+e^{-4\,x}} = \frac{1+\tanh(2\,x)}{2} = \text{sech}(2\,x)\, e^{2\,x}/2,$$

[12] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/neuronoid.mpl.out.txt`

[13] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/mollificatio.mpl.out.txt`

3.3 Approximation of the identify function

We also have to use a sigmoid to approximate the identity function defining:

$$\begin{array}{rcl} l_{\omega'}(x) & \stackrel{\text{def}}{=} & \omega' \left(h(x/\omega') - 1/2\right) \\ & = & x - \frac{4}{3}\frac{1}{\omega'^2}\, x^3 + O\left(x^5\right) \end{array}$$

and writing $\epsilon_{l,\omega'}(x) \stackrel{\text{def}}{=} x - l_{\omega'}(x)$, the $\mathcal{L}_1$ error magnitude on $[-M, +M]$ writes:

$$\int_{x=-M}^{x=+M} |\epsilon_{l,\omega'}(x)|\, dx = \frac{2}{3}\,\frac{M^4}{\omega'^2} + O\left(\frac{1}{\omega'^4}\right)$$

and the related $\mathcal{L}_0$ error magnitude:

$$\max(|\epsilon_{l,\omega'}(x)|), x \in [-M, M] = \epsilon_{l,\omega'}(M) = \frac{4}{3}\,\frac{M^3}{\omega'^2} + O\left(\frac{1}{\omega'^4}\right)$$

The identity function is thus impaired by a bias $\pm\frac{4}{3}\,\frac{M^3}{\omega'^2}$ reducing the output value with respect to the input value.

Assuming values are normalized, in the $[-1, +1]$ we obtain, for the $\mathcal{L}_0$ error magnitude:

ω'	10	50	100	200	500	1000
$\epsilon_{l,\omega'}(1)$	0.0131	$0.533\,10^{-3}$	$0.133\,10^{-3}$	$0.333\,10^{-4}$	$0.54\,10^{-7}$	$0.12\,10^{-7}$

with a negligible error for, say, $\omega' \geq 100$.

Derivations are available as `Maple` code[14].

3.4 Approximation of switch mechanisms

Given normalized floating point variables $v_n \in [-1, 1], n \in \{1, N\}$ and binary variables $b_n \in \{0, 1\}, n \in \{1, N\}$, conditional expressions and related mechanisms require the implementation of formula of the form[15]:

$$\begin{array}{rcl} v & = & \sum_{n\in\{0,N\}} b_n\, v_n \\ & = & \sum_{n\in\{0,N\}} \omega'\, h(v_n/\omega' + \omega\, b_n - \omega) + O\left(\frac{1}{\omega'^2}\right) + O\left(e^{-4\,w}\right) \end{array}$$

The implementation as neuronoid, thus allows to implement more formula than only using programmatoid.

4 Examples of neuronoid computation

A step ahead, we considering neuronoid with $\tau > 0$ it seems obvious that we can designed temporizing mechanisms, oscillators and sequence generator, sleep sort mechanism, etc.

4.1 Memory gate

For a data input i, a control $i_l \in \{0, 1\}$, and an output o, the equation:

$$o \leftarrow \text{ if } i_l = 1 \text{ then } o \text{ else } i$$

implements, if $i \in \{0, 1\}$, $o \in \{0, 1\}$, a 1 bit memory, i.e., also called a RS gate, and reusing the previous conditional instruction parameters.

The key point is that it is now a recurrent system, which stability is obvious at the programmatoid level, but not necessarily at the neuronoid level, since we have an recurrent equation for $i_l = 1$.

For $i \in \{0, 1\}$, $o \in \{0, 1\}$ at the programmatoid level:

$$o \leftarrow H(i - i_l - 1/2) + H(o + i_l - 3/2)$$

For $i \in \{0, 1\}$, $o \in \{0, 1\}$:

$$\begin{array}{rcl} o & \leftarrow & h(\omega\,(i - i_l - 1/2)) + h(\omega\,(o + i_l - 3/2)) \\ & = & O\left(e^{-4\,\omega}\right) + h(\omega\,(o - 1/2))\big|_{i_l=1} \end{array}$$

For $i \in [-1, 1]$, $o \in [-1, 1]$:

$$\begin{array}{rcl} o & \leftarrow & \omega'\,(h(i/\omega' - \omega\, i_l) + h(o/\omega' - \omega\,(1 - i_l)) \\ & = & O\left(\omega'\, e^{-4\,\omega}\right) + \omega'\, h(o/\omega')\big|_{i_l=1} \end{array}$$

- The former equation is exact, so entirely stable during iterations.

[14] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/linearapproximation.mpl.out.txt`

[15] We easily verifies that:
- if $b_n = 0$, while $|v_n/\omega'| \leq 1/\omega' \ll 1$, the term $\omega'\, h(v_n/\omega' + \omega\, b_n - \omega) \simeq h(-\omega) \simeq 0$ up to $O\left(e^{-4\,w}\right)$ as derived previously, while
- if $b_n = 1$,the term $\omega'\, h(v_n/\omega' + \omega\, b_n - \omega) = \omega'\, h(v_n/\omega')$ corresponds to the approximation of the identity function, up to $O\left(\frac{1}{\omega'}\right)$ as derived previously.

- For the second equation, memorizing $o|_{t=0} \in \{0,1\}$, thus with $i_l = 1$, for

$$\epsilon \stackrel{\text{def}}{=} \left\{ \begin{array}{ll} o & |_{o|_{t=0}=0} \\ 1-o & |_{o|_{t=0}=1} \end{array} \right.$$

we obtain[16]:

$$\begin{array}{rcl} \epsilon & \leftarrow & (1-2\ o|_{t=0})\, h(-(3/2-i)\,\omega) + h(\omega\,(\epsilon - 1/2)) \\ & = & \left\{ \begin{array}{llcll} 0 & & \epsilon & = & 0 \quad \text{if } o|_{t=0} = 1 \\ e^{-2\,\omega} + O\left(e^{-4\,\omega}\right) & & \epsilon & = & \varepsilon\, e^{-2\,\omega} \quad \text{otherwise} \end{array} \right. \end{array}$$

In words : the iteration remains stable, and the error at the order of magnitude of $O\left(e^{-2\,\omega}\right)$. In particular the previous bias ε only impacts the $O\left(e^{-4\,\omega}\right)$ terms. Furthermore, if $o|_{t=0} = 1$, the positive bias yields a convergence towards 1 without any bias if $i = 1$, and with a exponentially decreasing bias if $i = 0$.

- For the third equation, memorizing $o|_{t=0} \in [-1,1]$, thus with $i_l = 1$, while $i \in [-1,1]$, we obtain:

$$\begin{array}{rcl} o & \leftarrow & \omega'\,(h(i/\omega' - \omega\, i_l) + h(o/\omega' - \omega\,(1-i_l)) \\ \text{while, since } i_l = 1\text{:} & & \\ & = & \omega'\,(h(i/\omega' - \omega) + h(o/\omega') \\ \text{while, when neglecting } |i/\omega'| < 1/\omega' \ll \omega\text{:} & & \\ & = & \omega'\,(h(-\omega) + h(o/\omega') \\ \text{while, using previous series formula:} & & \\ & = & o \pm \frac{4}{3}\,\frac{1}{\omega'^2} + e^{-4\omega} + O\left(\frac{1}{\omega'^4}\right) + O\left(e^{-8\omega}\right) \\ \text{while, considering only the larger term:} & & \\ & \simeq & o \pm \frac{4}{3}\,\frac{1}{\omega'^2} = o|_{t=0} \pm t\,\frac{4}{3}\,\frac{1}{\omega'^2} \end{array}$$

The memorization is thus impaired by a linear bias only, thanks to the fact that linear unit has been normalized. Because $O\left(e^{-4\omega}\right) \ll O\left(\frac{1}{\omega'^2}\right)$, fro the chosen values, the bias due the use of a neuronoid approximation of a programmatoid is negligible with respect to the bias due to the use of a neuronoid approximation of a linear unit.

Derivations are available as `Maple` code[17].

4.2 Multistable mechanisms

The previous developments leads to solutions for several temporal systems, as briefly given now.

4.2.1 Bistable mechanisms

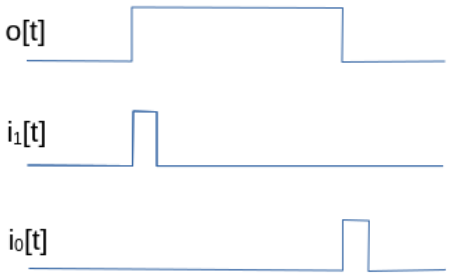


Figure 3: With:

$$\begin{array}{rcl} o & \leftarrow & \text{if } o = 0 \text{ and } i_1 = 1 \text{ then } 1 \\ & & \text{elif } o = 1 \text{ and } i_0 = 1 \text{ then } 0 \\ & & \text{else } o \end{array}$$

a bistable system with two 1 and 0 inputs, is implemented.

Figure 4: With:

$$\begin{array}{rcl} o & \leftarrow & \text{if } o = 0 \text{ and } i_1 = 1 \text{ then } 1 \\ & & \text{elif } o = 1 \text{ and } i_0 = 1 \text{ then } 0 \\ & & \text{else } o \\ i_1 & \leftarrow & \text{if } i_1 = 0 \text{ and } i = 1 \text{ then } 1 \\ & & \text{elif } o = 0 \text{ then } 0 \\ & & \text{else } i_1 \\ i_0 & \leftarrow & \text{if } i_0 = 0 \text{ and } i = 0 \text{ then } 1 \\ & & \text{elif } o = 1 \text{ then } 0 \\ & & \text{else } i_0 \end{array}$$

a bistable system with one two-way switch is implemented, using internal variables i_0 and i_1. This corresponds to a frequency divider by 2.

Figure 5: With:

$$\begin{array}{rcl} o & \leftarrow & \text{if } r = 0 \text{ and } i = 1 \text{ then } 1 \\ & & \text{else } 0 \\ r & \leftarrow & \text{if } o = 1 \text{ then } 1 \\ & & \text{elif } i = 0 \text{ then } 0 \\ & & \text{else } r \end{array}$$

a spike generation detecting signal rising, with an internal variable r registering if already detected.

4.2.2 Spike generation

4.2.3 Delay

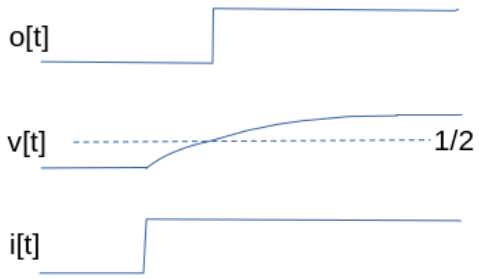


Figure 6: With:

$$\begin{array}{rcl} o & \leftarrow & \text{if } v > 1/2 \text{ then } 1 \text{ else } 0 \\ v & \leftarrow & (1-\gamma)\, v + \gamma\, i \end{array}$$

a delayed output is implemented. Assuming that $v = 0$, when $i = 1$ and remains at this value, we obtain for the delay T:

$$T = -\frac{\log(2)}{\log(1-\gamma)} > 0 \Leftrightarrow \gamma = 1 - 2^{-\frac{1}{T}}.$$

Here we assume that $i = 1, t \in [0, T]$ but it is very easy to avoid this constraint with a bistable mechanism as input.

Derivations are available as `Maple` code[18].

[16] For $o|_{t=0} = 0$ the derivation is straightforward. For $o|_{t=0} = 1$ and $i = 0$:

$$\begin{array}{rcll} \epsilon & \leftarrow & 1 - o & \\ & = & 1 - h(-3/2\,\omega) - h(\omega\,(1 - \epsilon + 1 - 3/2)) & \\ & = & 1 - h(-3/2\,\omega) - h(\omega\,(-\epsilon + 1/2)) & \\ & = & -h(-3/2\,\omega) + h(\omega\,(\epsilon - 1/2)) & \text{since } h(x) = 1 - h(-x) \end{array}$$

with a similar derivation for $o|_{t=0} = 1$ and $i = 1$.

[17] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/memorygate.mpl.out.txt`

[18] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/delayastable.mpl.out.txt`

4.2.4 Oscillation

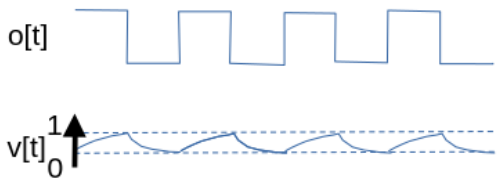


Figure 7: With:

$$\begin{array}{rcl} o & \leftarrow & \text{if } v < 1/3 \text{ then } 0 \\ & & \text{elif } v > 2/3 \text{ then } 1 \\ & & \text{else } o \\ v & \leftarrow & (1-\gamma)\, v + \gamma\, (1-o)\, i \end{array}$$

an binary oscillator is implemented. It is stopped if $i = 0$ and running for $i = 1$. We obtain for the period T:

$$T = -\frac{2\log(2)}{\log(1-\gamma)} > 0 \Leftrightarrow \gamma = 2 - 2^{1-\frac{1}{T}}.$$

Derivations are available as `Maple` code[19].

5 Softmax mechanisms

5.1 The explog function

The maximal and the average operators can be easily combined. For instance, given K inputs $i_k \in [-1,1], k \in \{1,K\}$, let us define:

$$\begin{array}{rcll} o & \stackrel{\text{def}}{=} & \frac{1}{\mu}\log\left(\frac{1}{K}\sum_{k\in\{1\cdots K\}}\exp(\mu\, i_k)\right) & \\ & = & \frac{1}{K}\sum_k i_k + O(\mu) & \text{(average)} \\ & = & \max_k(i_k) - \frac{\log(K)}{\mu} + \frac{O\left(e^{-\nu\,\mu}\right)}{\mu}, \nu > 0 & \text{(max)} \\ & = & i_1|_{i_i=i_2\cdots=i_K} & \\ & \in & [-1,1] & \end{array}$$

with $\mu > 0$ parameterizing the balance between:

- the average, for small μ, line 2 above,
- the maximun, for large μ, line 3 above, up to an offset $-\log(K)/\mu$,

as shown in Figure 8.

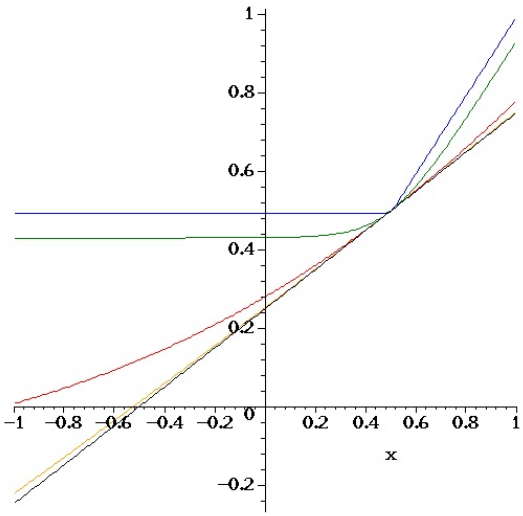


Figure 8: Representation of the exp-log function for $K = 2$, $i_2 = 1/2$, with $i_1 \in [0,1]$ and $\mu \in \{0.01, 0.1, 1, 10, 100\}$, from top to bottom, in black, ornage, red, green and blue, respectively. With small μ it is closed to linear average, whereas for $\mu = 100$ it is close to $\max(x, 1/2)$.

From Figure 8, we observe that for $i_k \in [-1,1]$, $\mu \in [0.01, 100]$ is a reasonable range to approximate the different profiles, so that:

[19] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/delayastable.mpl.out.txt`

- The $\exp(\cdot)$ function range is in $[-100, 100]$.

- The $\log(\cdot)$ function range is in $[e^{-100} \simeq 1, e^{100}]$. These are huge ranges but we may consider the following renormalization:

$$\begin{array}{rcl} o & = & \frac{N_i}{\mu}\left(\log\left(\frac{1}{K\,N_o}\sum_{k\in\{1\cdots K\}}\exp(\mu\,\frac{i_k}{N_i})\right)+\log(N_o)\right) \\ & & N_i=\max_\mu, N_o=\frac{e^{\max_\mu}}{N_l} \end{array}$$

so that the $\exp(\cdot)$ range is now $[-1, 1]$ and the $\log(\cdot)$ range is now $[1, N_l]$, with a small arbitrary value of N_l, allowing an implementation with not so big function's range. This is fine, providing that we can implement the $\exp(\cdot)$ and the $\log(\cdot)$ functions with a sufficient numerical precision.

Derivations of this section are available as `Maple` code[20].

5.2 Neuronoid approximation of `exp` and `log`

Taking benefit of the fact that sigmoid is convex as the exponential function in some range, and concave as the logarithm function in some range, we can easily design the following approximations, obtained by some manual intuitive choice of the sigmoid combination and numerical adjustment of the parameters. Two examples are given in Figure 9 and Figure 10. We share these examples to illustrate fact we really can approximate several bounded non-linear function with neuronoid, but we will use another track, to

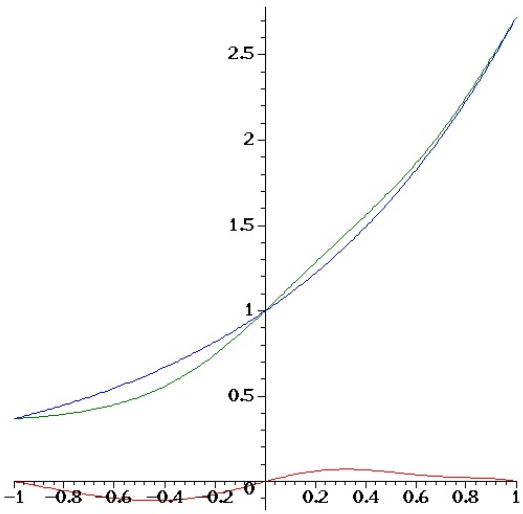


Figure 9: Approximation in the $[-1, 1]$ interval of the exponential function, in blue, by a sigmoid combination, in green:

$$0.345 + 2.34\,h(x-1) + 1.226\,h(x):$$

yielding a $\mathcal{L}_1$ error of 0.05, the error being drawn in red.

Given these two approximations, we can implement an approximation of the renormalized `explog` function proposed previously, with $N_l = 1$ in this case.

Derivations of this section are available as `Maple` code[21].

5.3 Neuronoid approximation a softmax operator

Another track is to consider the programatoid implementation of the maximium operator, given K inputs $i_k \in [-1, 1], k \in \{1, K\}$:

$$\begin{array}{rcll} o & \stackrel{\text{def}}{=} & \max_k(i_k) & \\ & = & \forall_k \text{if } {}_{\forall l\in\{1,K\},l\neq k} i_k > i_l \text{ then } i_k \text{ else } 0 & \text{Max symmetric definition} \\ & = & \sum_k And_{l\in\{1,K\},l\neq k}\,(H(i_k > i_l),\dots)\;i_k & \text{Logical implementation} \\ & = & \sum_k b_k\,i_k & \\ & & b_k \stackrel{\text{def}}{=} H\left(\sum_{l\in\{1,K\},l\neq k} H(i_k - i_l) - K - 1/2\right) & \text{Programmatoid definition} \\ & \simeq & \sum_k \omega'\,h\left(\frac{i_k}{\omega'} + \omega - \omega\,b_k\right) & \text{Switch mechanism} \\ & \simeq & \sum_k \omega'\,h\left(\frac{i_k}{\omega'} + \omega - \omega\,h_k\right) & \\ & & h_k \stackrel{\text{def}}{=} h\left(\omega\,\left(\sum_{l\in\{1,K\},l\neq k} h(\omega * (i_k - i_l)) - K - 1/2\right)\right) & b_k \text{ mollification} \end{array}$$

Let us now consider the modified definition, with $g \in [0, 1]$:

[20] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/explog.mpl.out.txt`

[21] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/explog.mpl.out.txt`

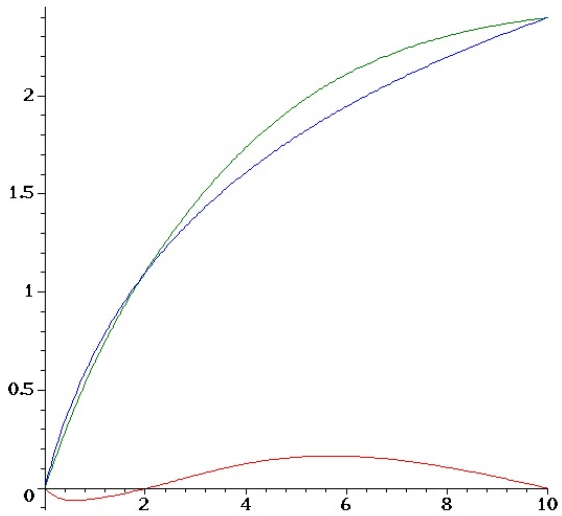


Figure 10: Approximation in the $[0, 10]$ interval of the logarithm $\log(1 + x)$ function, in blue, by a sigmoid combination, in green:

$$-2.48 + 4.42\, h(x/10) + 0.54\, h(x/2) :$$

yielding a $\mathcal{L}_1$ error of 0.05, the error being drawn in red.

$$\begin{array}{rcll} o & \stackrel{\text{def}}{=} & \frac{K\, g-g+1}{K} \sum_k \omega'\, h\left(\frac{i_k}{\omega'} + g\,\omega - g\,\omega\, h_k\right) & \\ & = & \sum_k \omega'\, h\left(\frac{i_k}{\omega'} + \omega - \omega\, h_k\right)\big|_{g=1} & \simeq \max_k(i_k) \\ & = & \frac{1}{K} \sum_k \omega'\, h\left(\frac{i_k}{\omega'}\right)\big|_{g=0} & \simeq \frac{\sum_k i_k}{K} \end{array}$$

With this definition we have constructed a softmax mechanism, directly based on the previous developmemts, without introducing numerical approximations of other non-linear functions, as shown in Figure. 11.

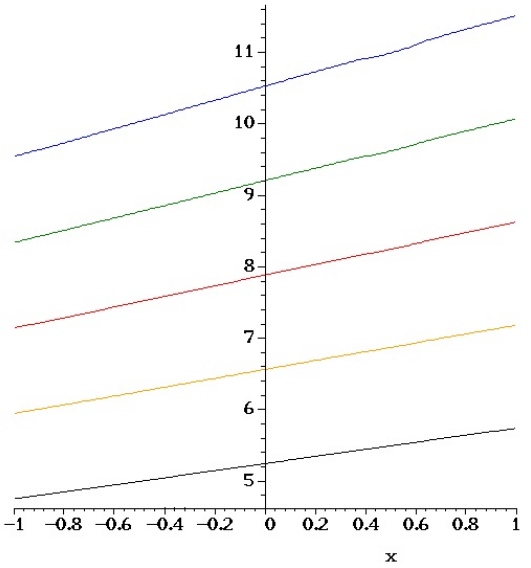


Figure 11: Representation of the exp-log function for $K = 2$, $i_2 = 1/2$, with $i_1 \in [0, 1]$ and $g \in \{0, 1/4, 1/2, 3/4, 1\}$, from top to bottom, in black, ornage, red, green and blue, respectively. With small $g = 0$ it is closed to linear average, whereas for $g = 1$ it is close to $\max(x, 1/2)$.

6 Comparing `tanh` with other non linearities

We have $h(x) = \frac{1+\tanh(2\,x)}{2}$ thus based on the `tanh` non linearity. This corresponds most common activation function when deriving a rate-based reduced neural network from bio-physical elements [Cessac and Samuelides, 2007]. This is far from being the only choice, as reviewed in [Szandała, 2020], and we could have considered, for instance $\bar{h}$ functions of the form:

Name:	Tanh	Erf	Arctan	Softsign
Formula:	$\frac{1}{2}+\frac{\tanh(2\,x)}{2}$	$\frac{1}{2}+\frac{\arctan(\pi\,x)}{\pi}$	$\frac{1}{2}+\frac{\mathrm{erf}(\sqrt{\pi}\,x)}{2}$	$\frac{1}{2}+\frac{x}{1+\lvert 2\,x\rvert}$
Sharpness:		$\lvert h(x)\rvert < \lvert h(x)\rvert$	$\lvert h(x)\rvert > \lvert h(x)\rvert$	$\lvert h(x)\rvert > \lvert h(x)\rvert$
$\mathcal{L}_1 =$	0	$-\frac{\pi\,\log(2)-2}{2\,\pi}$	$+\infty$	$+\infty$
$\mathcal{L}_0 \simeq$	0	0.018	0.082	0.81
κ_{max}	1.53	1.52	2.04	4

while all profiles are normalized and symmetric, i.e.:

$$\bar{h}(-\infty)=0, \bar{h}(0)=1/2, \bar{h}'(0)=1, \bar{h}(+\infty)=1, \bar{h}(x)=1-\bar{h}(-x)$$

and are drawn in the Figure 12.

- The Arctan and Softsign profiles are smoother than the Tanh profile, whereas we are looking here for sharper profiles in order to better approxximate the step-function. They also have higher maximal curvatures κ_{max} which means that the curvature is more inhomogeneous than for the Tanh profile, and they do not correspond to biologically plausible activation functions [Cessac and Samuelides, 2007], despite the fact they could be of great interest in machine learning [Szandała, 2020].
- The error function[22], Erf, based profile is very closed numerically from the Tanh profile, with a finite $\mathcal{L}_1 \simeq -0.028$ small distance magnitude, and a $\mathcal{L}_0 < 2\%$ small distance magnitude, with similar maximal curvatures κ_{max} up to 1%, and is also quoted as a biologically plausible activation functions [Cessac and Samuelides, 2007]. Though its sharpness is a bit better than fro teh Tanh profile, with a better repartition of the curvature, all algebraic derivations made in this paper would not have as easy as its is now.

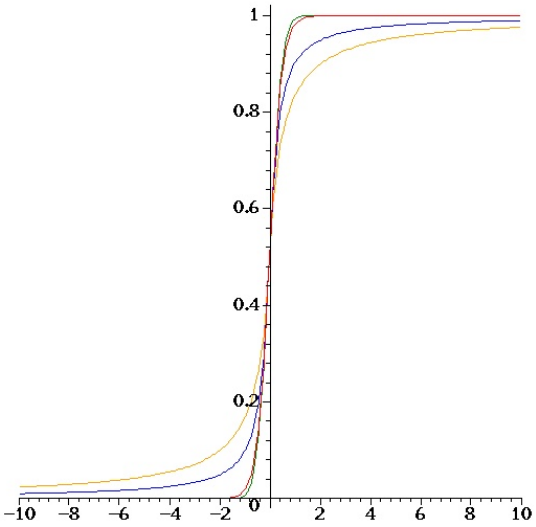


Figure 12: Comparision between the normalized Tanh profile in red, the Erf profile in green, the Arctan profile in blue and the Softsign in orange, i.e., from the sharpest to the smoothest.

Derivations are available as `Maple` code[23].

7 Using the braincraft challenge setup

This documentation allows to use the braincraft challenge for a programmatic implementation, or for a programmatoid/neuronoid implementation using code translator (not yet available).

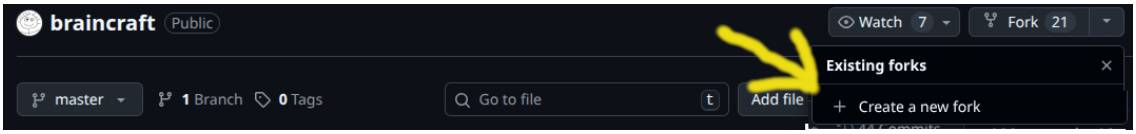
Here is braincraft challenge original documentation original documentation[24].

You must be familiar with basic `git` usage and basic `python` programmation.

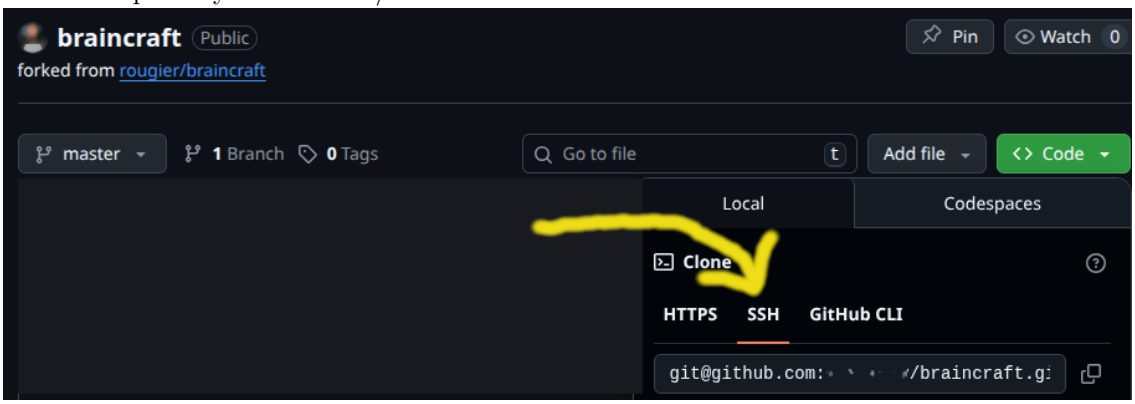
7.1 Installation of the setup

1. Connect to `https://github.com` with your login.
2. Go to the braincraft challenge[25] and create a new fork:

[22] `https://en.wikipedia.org/wiki/Error_function`
[23] `https://raw.githubusercontent.com/vthierry/braincraft/master/doc/tex/sigmoidatan.mpl.out.txt`
[24] `https://github.com/rougier/braincraft/blob/master/README.md`
[25] `https://github.com/vthierry/braincraft`



3. Download the repository in SSH read/write mode:



4. In the braincraft local git directory, run
   `make demo`

Note: You may have to install:
    `sudo apt install python3-tqdm`
while using a virtual environment, is advised, but not mandatory.

7.2 Running at the programmatic level

When running at the programmatic level,
- the `next_output_from_network(context)`[26] callback is to be implemented, and
- to be called by the "callback" version of the `evaluate(···)`[27] method.

The `challenge_callback_1.py`[28] source file includes all documented methods.

7.3 Running at the programmatoid level

At the programmatoid level, the system is defined by a set of equations of the form:

```
subs({
    T = 10, # The delay
    constantName = constantValue
},
[
  prgm_options = { omega = 10 },
  a = H(H(b)),
  Delay(b, , T)
])
```

Available options:

Name	Type	Default value	
`omega`	large positive float	1000	The ω used for both Id() function and H() mollification
`all_neuronoid`	Boolean	false	Wether H() functions are mollified to remain with neuronoid only

For binary variables `b_•` $\in \{0, 1\}$, continuous variables `v_•` $\in [-1, 1]$, and the sigmoid $h(\cdot)$ the following functions, defined previously, are implemented:

[26] `https://html-preview.github.io/?url=https://github.com/vthierry/braincraft/blob/master/doc/api/challenge_callback.html#next_output_from_network`
[27] `https://html-preview.github.io/?url=https://github.com/vthierry/braincraft/blob/master/doc/api/challenge_callback.html#evaluate`
[28] `https://github.com/vthierry/braincraft/blob/master/braincraft/challenge_callback.py`

`v_o = h(v_i)`	The normalized sigmoid.
`v_o = H(v)`	Generalized step-function. `v_o` $\in \{0_{<0}, 1/2_{1/2}, 1_{>0}\}$ Converted to `h()` when the option `prgm["neuronoid"] = true`.
`b_ o = If_b(b_1, b'_1, ..., b'_0)`	Binary conditional expression, Implements `if b_1 then b'_1 ...`.
`v_o = If_v(b_1, v_1, ..., v_0)`	Valued conditional expression, Implements `if b_1 then v_1 ...`.
`b_o = And(b_1, ...)`	Conjunction with a variable number of binary arguments. Implements `b_1 and b_2 ...`
`b_o = Or(b_1, ...)`	Disjunction with a variable number of binary arguments. Implements `b_1 or b_2 ...`
`b_o = Not(b)`	Negation of a binary value. Implements `not b`
`v_o = Bprod(b_1, v_1, ...)`	Sum of binary products Implements `b_1  v_1 + ...`. Default missing value is 0.
`v_o = Softmax(v_1, ..., G)`	Mean-max operator. `G` $\in [0_{average}, 1_{maximum}]$ controls the mean-max balance.
`Latch_b(b_o, b_i, b_c)`	Binary memory latch. `b_o` is the output, `b_i` is the input, `b_c` $\in \{0_{open}, 1_{latched}\}$ is the control.
`Latch_v(b_o, v_i, b_c)`	Valued memory latch. `b_o` is the output, `v_i` is the input, `b_c` $\in \{0_{open}, 1_{latched}\}$ is the control.
`Bistable(b_o, b_1, b_0)`	Two inputs bistable latch. `b_o` is the output, `b_0` resets to 0, `b_1` sets to 1.
`Bistable(b_o, b_i)`	One input bistable latch. `b_o` is the output, `b_i` is the two-way input.
`Spikeup(b_o, b_i)`	Spike generation on input rising front. `b_i` is the input, which rising front is detected.
`Delay(b_o, b_i, T)`	Delayed output. `T` > 0 is the delay, in number of global clock event, `b_o` is the output, `b_i` is the delayed input.
`Oscillator(b_o, b_c, T)`	Oscillatory output. `T` > 0 is the period, in number of global clock event. `b_o` is the output, `b_c` $\in \{0_{stop}, 1_{run}\}$ is the control.

References

[Cessac and Samuelides, 2007] Cessac, B. and Samuelides, M. (2007). From neuron to neural networks dynamics. *The European Physical Journal Special Topics*, 142(1):7–88.

[Lapicque, 1907] Lapicque, L. (1907). Recherches quantitatives sur l'excitation electrique des nerfs traitee comme une polarization. *Journal de physiologie et de pathologie générale*, 9:620–635.

[Szandała, 2020] Szandała, T. (2020). *Review and Comparison of Commonly Used Activation Functions for Deep Neural Networks.*