Formule RMS

$$rms\ proportion_e = \frac{rms_e = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(o_{i,e} - d_i)^2}}{max(rms_e),\ \forall e \in epochs}$$

$$with \begin{cases} n : number\ of\ neurons\ on\ the\ output\ layer \\ o_{i,e} : value\ obtained\ for\ the\ i^{th}\ neuron\ at\ the\ e^{th}\ epoch \\ d_i : value\ desired\ for\ the\ i^{th}\ neuron \end{cases}$$

```
function DISCRETIZE(hiddenNeuron[], piece)
    result ← 0
    for i = 0 → hiddenNeuron.length do
        result ← result + piece^i × cutting(hiddenNeuron[i], piece)
        i ← i + 1
    end for
    return result
end function
```

-

```
first_order.calc_hidden_layer(samples.inputs)
high_order.calc_output_layer(first_order.hidden_layer)
first_order.calc_output_layer(first_order.hidden_layer, [0, ..., 0])

h_output ← ampli(high_order.output_layer)
right_houtput ← [0, 0]
if good_answer(first_order) then
    right_houtput[1] ← 1
else
    right_houtput[0] ← 1
end if
first_order.calc_output_layer(first_order.hidden_layer, h_output)

calc_stats()

high_order.train(first_order.hidden_layer, right_houtput)
first_order.train(samples.inputs, samples.outputs, h_output)
```

```
function train(inputs, outputs, add)
    for i = 0 → output_neurons.length do
        y_output[i] ← g'(output_neurons[i].a) × (outputs[i] − output_neurons.state)
    end for

    for i = 0 → hidden_neurons.length do
        w_sum ← Σ_{j=0}^{output_neurons.length} output_neurons[j].weights[i] × y_output[j]
        y_hidden[i] ← g'(hidden_neurons[i].a) × w_sum
    end for
    update_weights_hidden_layer(y_hidden)

    for i = 0 → output_neurons.length do
        output_neurons[i].update_weights_gradient(y_output[i], hidden_neurons, add)
        output_neurons[i].update_weights_perceptron(outputs[i], hidden_neurons, add)
    end for
end function
function update_weights_gradient(error, intputs, add)
    calc_output(inputs + add)

    for j = 0 → inputs.length do
        dw ← weights[j] − last_weights[j]
        p ← error × inputs[j]
        weights[j] ← weights[j] + learning_rate × p + momentum × dw
    end for
end function
function update_weights_perceptron(goal, intputs, add)
    calc_output(inputs + add)

    for j = inputs.length → inputs.length + add.length do
        dw ← weights[j] − last_weights[j]
        p ← (goal − state) × add[inputs.length − j]
        weights[j] ← weights[j] + (learning_rate×p+momentum×dw) / add.length
    end for
end function
```