Perceptron

This chapters plays two roles. The first one, is to describes how and why a perceptron plays a role so important in the meaning of deep learning. The second role of this chapter, is to provide a gentle introduce to the Pharo programming language.

Biological Connection

The primary visual cortex contains 140 millions of neurons, with tens of billions of connections. A typical neuron propagates electrochemical stimulation received from other neural cells using *dendrite*. An *axon* conducts electrical impulses away from the neuron (Neuron).

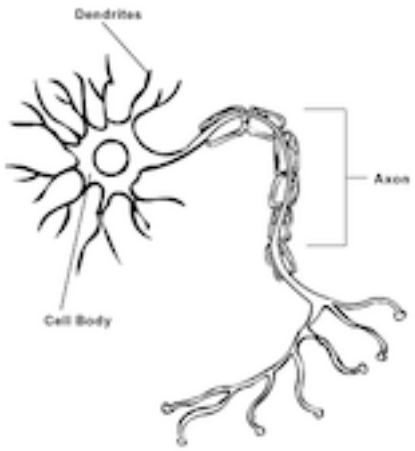

Figure 1: Neuron

Expressing a computation in terms of artificial neurons was first thought in 1943, by Warren S. Mcculloch and Walter Pitts in their seminal article *A logical calculus of the ideas immanent in nervous activity*. This paper has been cited more than 14 000 times.

Perceptron

A perceptron is a kind of miniature machine that produces an output for a provided input (Perceptron). A perceptron may accept 0, 1, or more input, and result in a small and simple computation. A perceptron operates on numerical values, which means that the inputs and the output are numbers (integer or float, as we will see later).

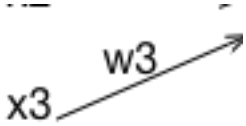

Figure 2: Perceptron

The figure depicts a perceptron with three inputs, noted *x1*, *x2*, and *x3*. Each input is indicated with an incoming arrow and the output with the outgoing arrow.

Not all inputs have the same importance for the perceptron. For example, an input may be more important than the others. Relevance of an input is expressed using a weight associated to that input. In our figure, the input *x1* has the weight *w1*, *x2* has the weight *w2*, and *x3* has *w3*.

Modeling boolean gates

A perceptron is a kind of artificial neuron, developed in the 50s and 60s by Frank Rosenblatt, Warren McCulloch, and Walter Pitts.

A Perceptron in action

```
Object subclass: #Perceptron
    instanceVariableNames: 'weights bias'
    classVariableNames: ''
    package: 'NeuralNetworks-Core'

Perceptron>>weights: someWeightsAsNumbers
    weights := someWeightsAsNumbers copy

Perceptron>>weights
    ^ weights

Perceptron>>bias: aNumber
    bias := aNumber

Perceptron>>bias
    ^ bias
```

```
Perceptron>>feed: inputs
    | r tmp |
    tmp := inputs with: weights collect: [ :x :w | x * w ].
    r := tmp sum + bias.
    ^ r > 0 ifTrue: [ 1 ] ifFalse: [ 0 ]
```

Formulating Logical expressions

```
TestCase subclass: #NNPerceptronTest
    instanceVariableNames: ''
    classVariableNames: ''
    package: 'NeuralNetworksTests'
```

```
NNPerceptronTest>>testAND
    | p |
    p := MPPerceptron new.
    p weights: { 1 . 1 }.
    p bias: -1.5.

    self assert: (p feed: { 0 . 0 }) equals: 0.
    self assert: (p feed: { 0 . 1 }) equals: 0.
    self assert: (p feed: { 1 . 0 }) equals: 0.
    self assert: (p feed: { 1 . 1 }) equals: 1
```

```
NNPerceptronTest>>testOR
    | p |
    p := MPPerceptron new.
    p weights: { 1 . 1 }.
    p bias: -0.5.

    self assert: (p feed: { 0 . 0 }) equals: 0.
    self assert: (p feed: { 0 . 1 }) equals: 1.
    self assert: (p feed: { 1 . 0 }) equals: 1.
    self assert: (p feed: { 1 . 1 }) equals: 1
```

```
Perceptron>>train: inputs desiredOutput: desiredOutput
    | c newWeights output |
    output := self feed: inputs.
    c := 0.1.
    "Works well"
    desiredOutput = output
        ifTrue: [ ^ self ].

    "Basic check"
    self assert: [ weights size = inputs size ] description: 'Wrong size'.
    desiredOutput = 0
```

```
        ifTrue: [ "we should decrease the weight"
            newWeights := (1 to: weights size) collect: [ :i | (weights at: i) - (c * (input
            bias := bias - c ]
        ifFalse: [ "We have: designedOutput = 1"
            newWeights := (1 to: weights size) collect: [ :i | (weights at: i) + (c * (input
            bias := bias + c ].
    weights := newWeights
```

```
NNPerceptronTest>>testTrainingOR
    | p |
    p := MPPerceptron new.
    p weights: { -1 . -1 }.
    p bias: 2.

    100 timesRepeat: [
        p train: { 0 . 0 } desiredOutput: 0.
        p train: { 0 . 1 } desiredOutput: 1.
        p train: { 1 . 0 } desiredOutput: 1.
        p train: { 1 . 1 } desiredOutput: 1.
    ].

    self assert: (p feed: { 0 . 0 }) equals: 0.
    self assert: (p feed: { 0 . 1 }) equals: 1.
    self assert: (p feed: { 1 . 0 }) equals: 1.
    self assert: (p feed: { 1 . 1 }) equals: 1
```

```
p := MPPerceptron new.
p weights: { -1 . -1 }.
p bias: 2.

100 timesRepeat: [
    p train: { 0 . 0 } desiredOutput: 0.
    p train: { 0 . 1 } desiredOutput: 1.
    p train: { 1 . 0 } desiredOutput: 1.
    p train: { 1 . 1 } desiredOutput: 1.
].
p feed: { 1 . 0 }
```

Exercises

The method `train:desiredOutput:` defines the learning rate `c` with the value `0.1`. Try using different values.