Introduction

History

Humans have always been fascinated by an intelligence emerging from something else than Human. Vikings have myth of monsters made of stones and metal.

Modern Artificial Intelligence began with the invention of programmable digital computer in the 40s. In 1948, Alan Turing designed “Machine Intelligence” as machine automatically solving problems.

Arthur Samuel, in his 1983 talk titled “AI: Where it Has Been and Where It is Going” said that the main goal of machine learning and artificial intelligence is *to get machines to exhibit behavior, which if done by humans, would be assumed to involve the use of intelligence.*

Artificial Intelligence

“One of the central challenges of computer science is to get a computer to do what needs to be done, without telling it how to do it.” – John Koza

Agile Artificial Intelligence

About bringing agility in the way techniques related to artificial intelligence are designed, implemented, and evaluated. This implies that artificial intelligence systems is built in an iterative way and using an incremental design.

Our code is often accompanied with unit tests and visualizations, and written in a programming environment supporting …

This book focus on implementing effective techniques that are commonly associated to Artificial Intelligence. Only a very small portion of what Artificial Intelligence is is covered in this book.

Why this book?

This book is meant to detail some easy-to-use recipes to solve punctual problems. This book highlights some technical details using the Pharo programming language.

What if I am not a Pharo programmer?

The only way to salve you is to learn Pharo :) Pharo has a very simple syntax, which means that even for a Java programmer, code should be understandable.

Chapter 2 also presents the Pharo programming language and environment.

Additional Readings

- http://www.gp-field-guide.org.uk by Riccardo Poli, Bill Langdon, and Nic McPhee
- http://natureofcode.com by Daniel Shiffman
- http://neuralnetworksanddeeplearning.com

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).

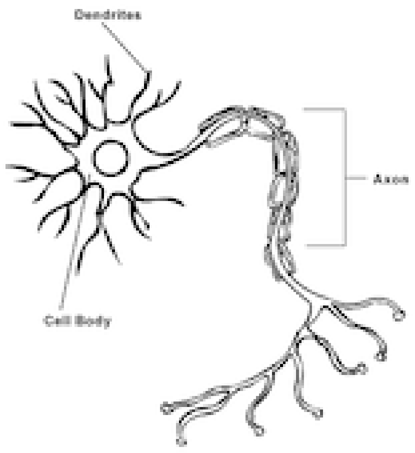

Figure 1: Neuron

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 an entity that accept some inputs and has exactly one output (Perceptron).

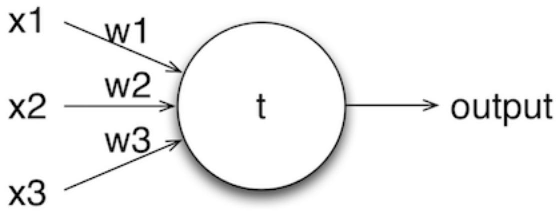

Figure 2: Perceptron

Model for Decision Making

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.