TechTalk on Artificial Intelligence

— A practical approach to Neural Networks —

Alexandre Bergel
University of Chile, Object Profile
http://bergel.eu

Goal of today

Show what can be done in plain Pharo related to neural network

These slides…

… are a support for the TechTalk

… are not meant to be understandable when read offline

… are a summary of a lecture given at the University of Chile

… incremental in their content

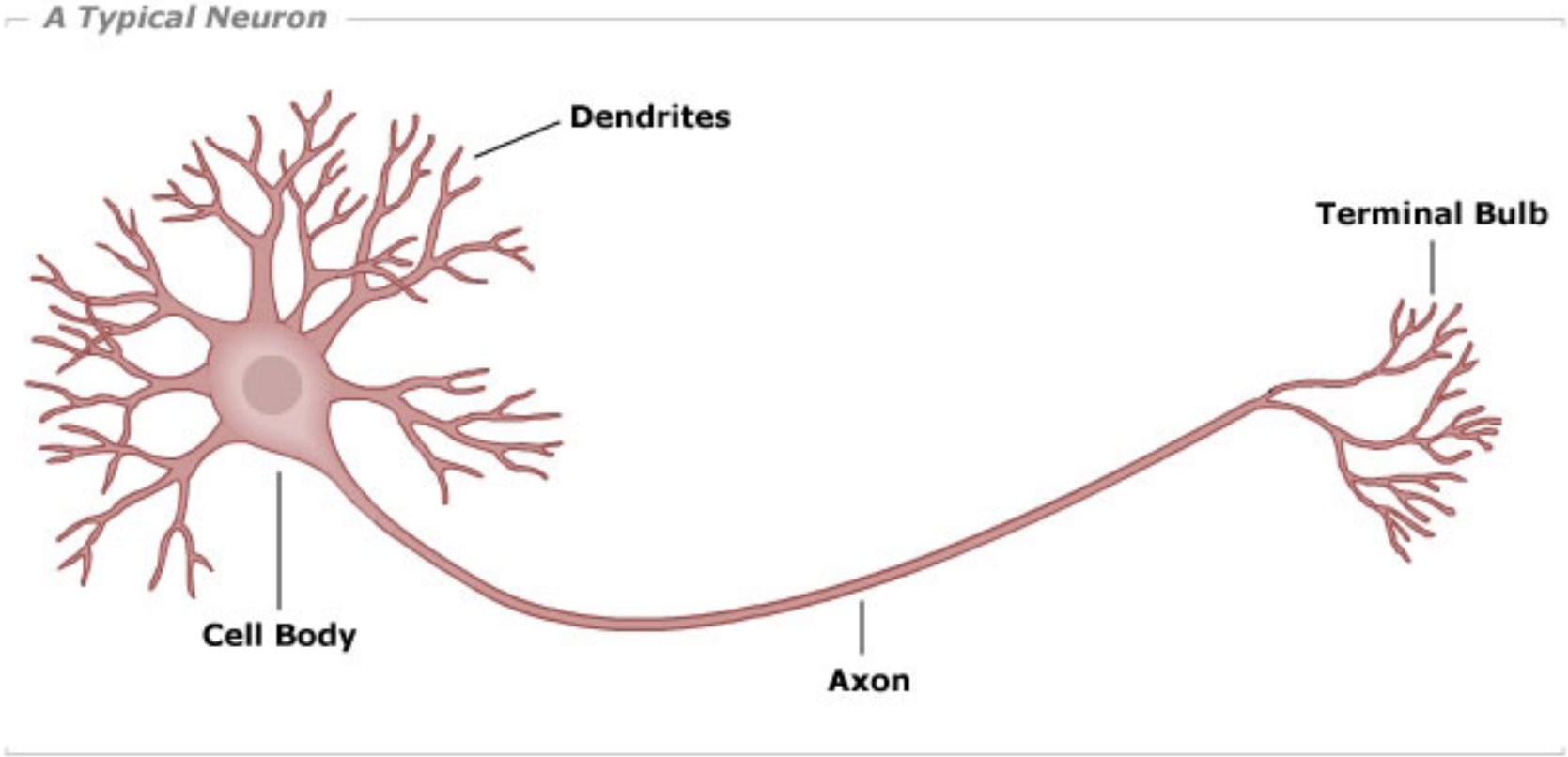
… do not assume any theoretical knowledge (even if some slides are scary)

```
Gofer it
    smalltalkhubUser: 'abergel' project: 'NeuralNetworks';
    configurationOf: 'NeuralNetworks';
    loadDevelopment
```

Outline

1. Perceptron
2. Learning Perceptron
3. Sigmoid Neuron
4. Neural Network
5. Classifying some data

Perceptron



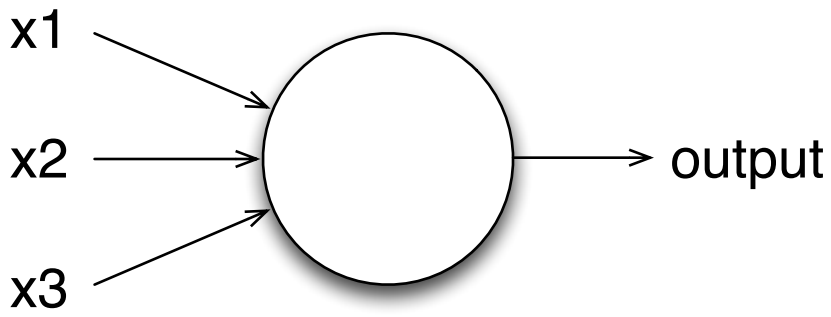
Dendrite: propagate electrochemical stimulation received from other neural cells

Axon: conducts electrical impulses away from the neuron

Perceptron

A *perceptron* is a kind of *artificial neuron*

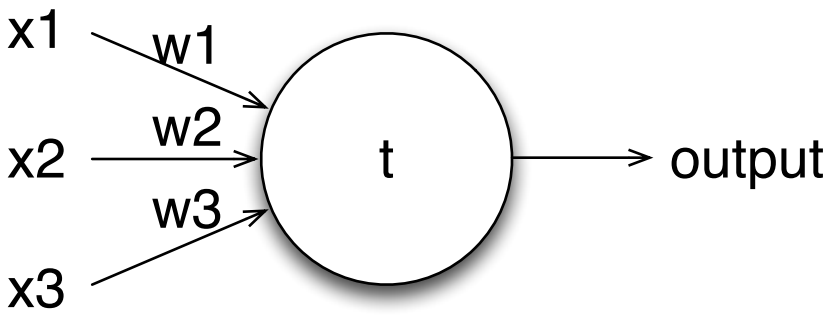
Developed in the 50s and 60s by Frank Rosenblatt, Warren McCulloch, Walter Pitts



Takes several binary inputs, x1, x2, … and produces a single binary output

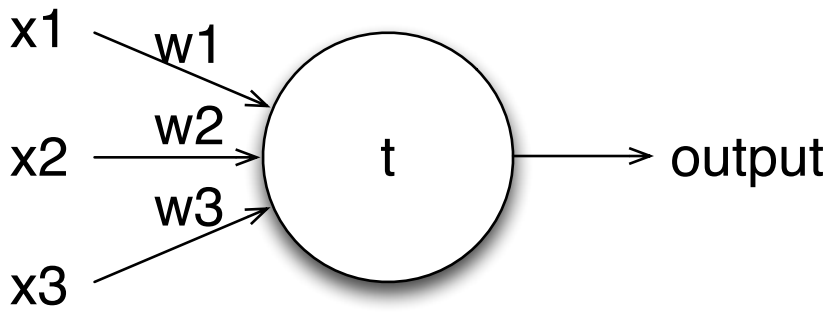
Perceptron

The output of a perceptron is the weighted sum of the input



if the weighted sum is greater than t, then we fire 1

Perceptron



$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

This is how a perceptron works. A perceptron is a device that makes decisions by weighing up evidence

Perceptron 

Suppose there is a great metal concert this weekend

You love metal, and you are wondering if you should go or not to the concert

You may want to make your decision by weighing up three factors:

- Is the weather good?
- Does your brother/sister/{boy,girl}friend want to accompany you?
- Is the festival near a metro stop? (You do not like driving)

Perceptron 

Suppose there is a great metal concert this weekend

You love metal, and you are wondering if you should go or not to the concert

You may want to make your decision by weighing up three factors:

x1 Is the weather good?

x2 Does your brother/sister/{boy,girl}friend want to accompany you?

x3 Is the festival near a metro stop? (You do not like driving)

Perceptron 

x1 Is the weather good?

x2 Does your brother/sister/{boy,girl}friend want to accompany you?

x3 Is the festival near a metro stop? (You do not like driving)

If you are a true-metal lover who love to share, then you may want to go even if the weather is bad and there is no stop near-by.

In that case, w1 = 2, w2 = 6, w3 = 2

Perceptron 

x1 Is the weather good?

x2 Does your brother/sister/{boy,girl}friend want to accompany you?

x3 Is the festival near a metro stop? (You do not like driving)

… or if you wish to not weak up your parents late in the evening: w1 = 1, w2 = 1, w3 = 8

Perceptron 

We are using the perceptron to model a simple decision-making.

If we pick 5 as our threshold, then we have the following condition:

$$(x1 * w1) + (x2 * w2) + (x3 * w3) >= 5$$

If the condition is true, then the perceptron outputs 1, else it output 0

Perceptron 

Varying the weights and the threshold produces a new model of decision-making

x1 Is the weather good?

x2 Does your brother/sister/{boy,girl}friend want to accompany you?

x3 Is the festival near a metro stop? (You do not like driving)

w1 = 2	w1 = 2
w2 = 6	w2 = 6
w3 = 2	w3 = 2
t = 5	t = 2

Decreasing t means that you are more willing to the metal party

Adapting the perceptron

We can move the threshold to the other side of the equation: *threshold* is now named *bias*

*(x1 * w1) + (x2 * w2) + (x3 * w3) >= 5*
*(x1 * w1) + (x2 * w2) + (x3 * w3) - 5 >= 0*

The perception can be rewritten

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

We want some code!

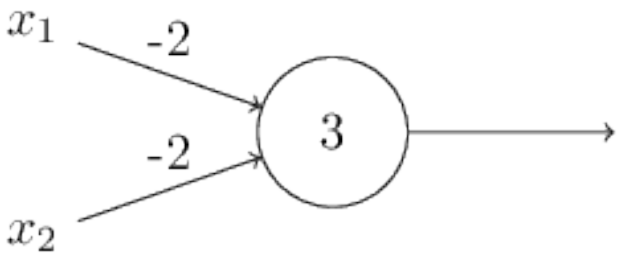
```
Object subclass: #MPPerceptron
    instanceVariableNames: 'weights bias'
    classVariableNames: ''
    package: 'NeuralNetworks-Core'
```

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

Formulating logical equation

AND, OR, NAND are elementary logical function

Consider the following perceptron:



With 00 (shortcut for x1 = 0, x2 = 0) produces the output 1, since (-2)*0 + (-2)*0 + 3 = 3, which is positive

The input 01 and 10 produces 1. But 11 produces -1, which is negative. Our perceptron implements a NAND gate.

$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

Learning Perceptron

Perceptron learning algorithm (1/3)

During the training period, a series of inputs are presented to the perceptron

Each input follows x = (x1, x2, …, xN)

For each input set, there is a *desired output (0 or 1)*

The actual output is determined by w*x + b

If the actual output is wrong, then two things could have happened:

Perceptron learning algorithm (2/3)

The designed output is 0, but the actual input is *above* the threshold. So the actual output becomes 1

In such a case, we should decrease the weights

The decrease in weight should be proportional to the input.

wN = wN - C*xN

b := b - C

C is a constant, let's pick 0.01

Perceptron learning algorithm (3/3)

The other case when the perceptron makes a mistake is when the desired output is 1, but the actual input is *below* the threshold

We should increase the weights

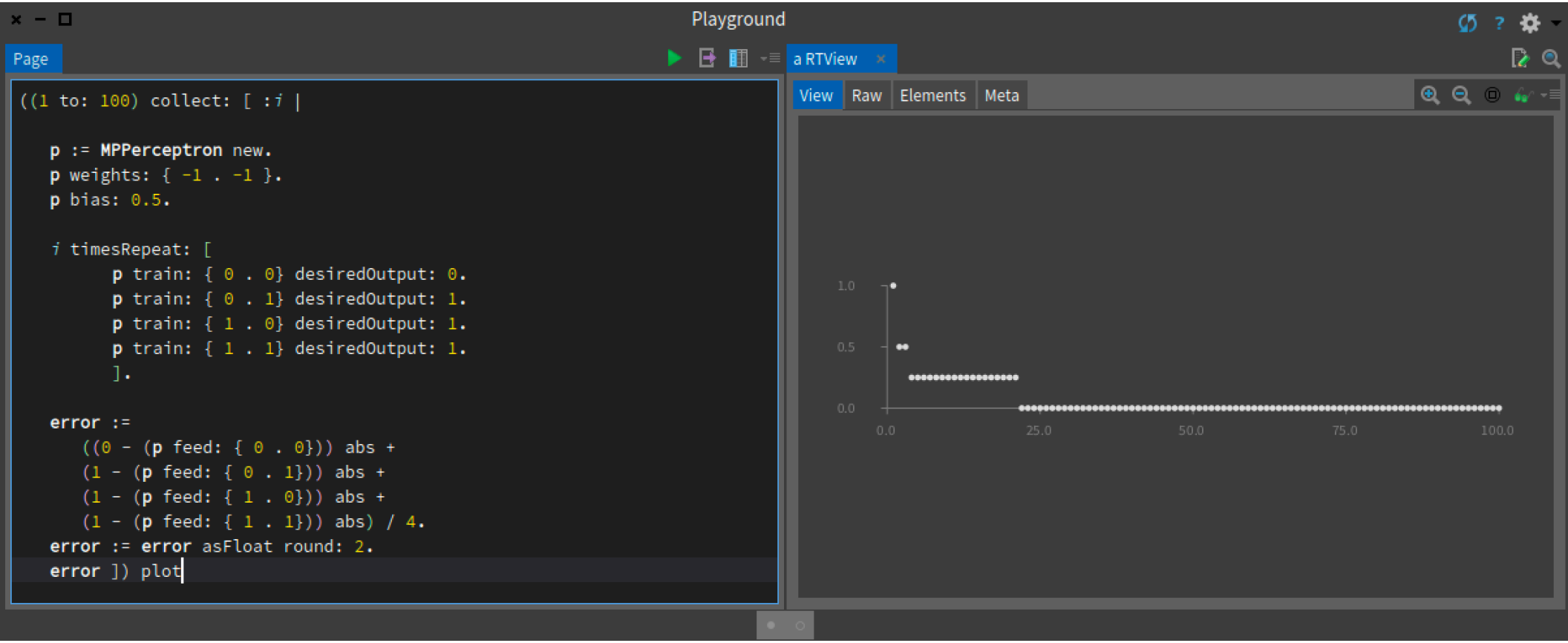
$wN = wN + C*xN$

$b := b + C$

```
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 * (inputs at: i)) ].
            bias := bias - c ]
        ifFalse: [ "We have: designedOutput = 1"
            newWeights := (1 to: weights size)
                 collect: [ :i | (weights at: i) + (c * (inputs at: i)) ].
            bias := bias + c ].
    weights := newWeights
```

```
testBasicLearning

    | p |
    p := MPPerceptron new.
    p weights: { -1 . -1 }.
    p bias: 0.5.

    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.
```

```
(1 to: 100) collect: [ :i |

    p := MPPerceptron new.
    p weights: { -1 . -1 }.
    p bias: 0.5.

    i 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.
            ].

    error :=
        ((0 - (p feed: { 0 . 0})) abs +
        (1 - (p feed: { 0 . 1})) abs +
        (1 - (p feed: { 1 . 0})) abs +
        (1 - (p feed: { 1 . 1})) abs) / 4.
    error := error asFloat round: 2.
    error ]
```


Playground
Page
((1 to: 100) collect: [:i |
p := MPPerceptron new.
p weights: { -1 . -1 }.
p bias: 0.5.
i 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.
].
error :=
((0 - (p feed: { 0 . 0})) abs +
(1 - (p feed: { 0 . 1})) abs +
(1 - (p feed: { 1 . 0})) abs +
(1 - (p feed: { 1 . 1})) abs) / 4.
error := error asFloat round: 2.
error]) plot
a RTView
View
Raw
Elements
Meta
1.0
0.5
0.0
0.0
25.0
50.0
75.0
100.0

Sigmoid Neuron

Problem with the perceptron

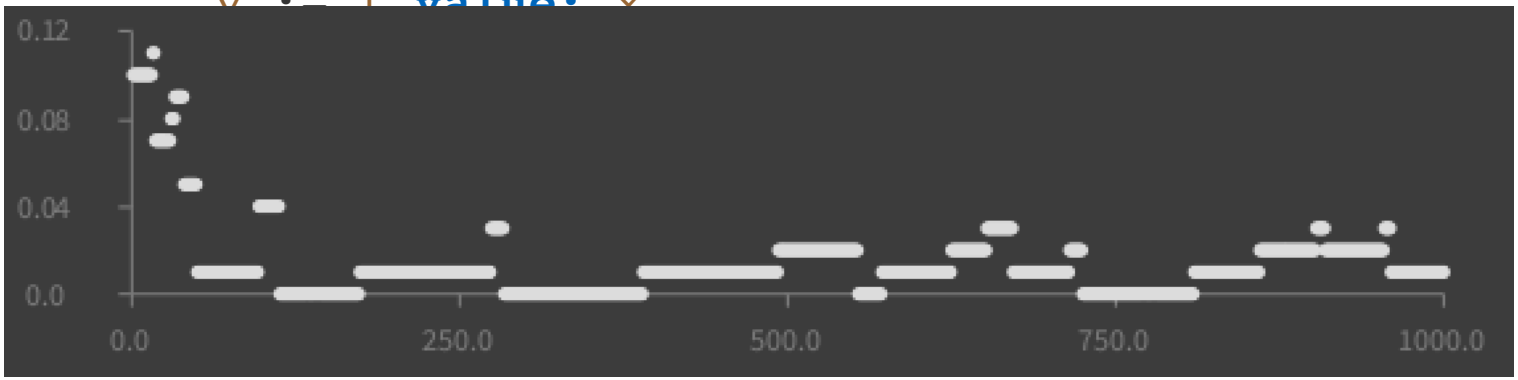
$$\text{output} = \begin{cases} 0 & \text{if } w \cdot x + b \leq 0 \\ 1 & \text{if } w \cdot x + b > 0 \end{cases}$$

When learning from a small error, a big change can occurs in the output

=> And that is a big problem

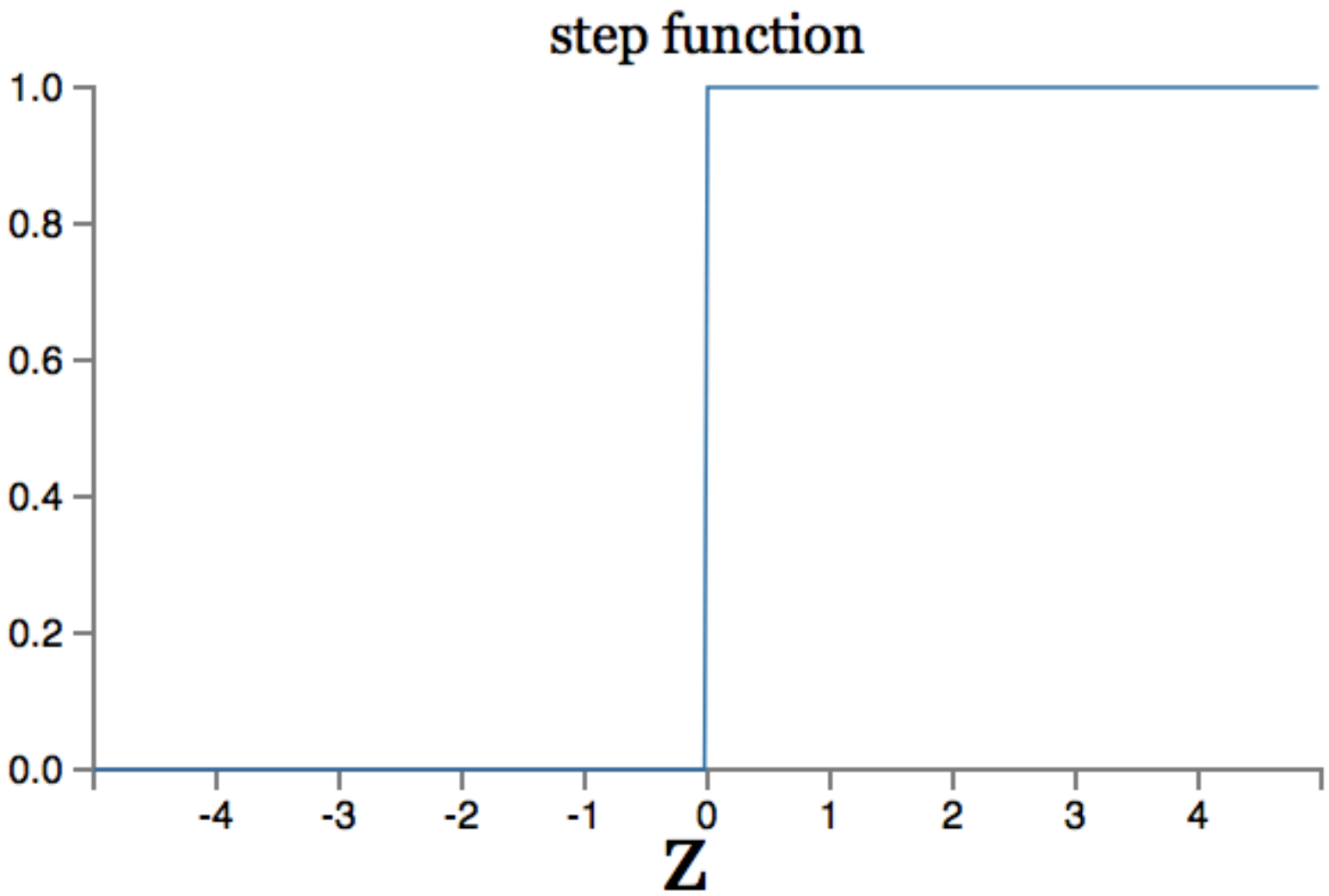
```
((1 to: 1000) collect: [ :i |
    f := [ :x | x * -2 - 40 ].
    r := Random new seed: 42.

    p := MPPerceptron  new.
    p weights: { -1 . -1 }.
    p bias: -0.5.

    i timesRepeat: [
          x := (r nextInt: 100) - 50.
          y := f value: x.
          o := (y >= 0) ifTrue: [ 1 ] ifFalse: [ 0 ].
          p train: { x . y } desiredOutput: o.
          ].

    error := 0.
    100 timesRepeat: [
          x := (r nextInt: 100) - 50.
          y := f value: x.
          o := (y >= 0) ifTrue: [ 1 ] ifFalse: [ 0 ].
          error := error + (o - (p feed: { x . y })) abs.
    ].
    error := error / 100.
    error := error asFloat round: 2.
    error ]) plot
```

```
((1 to: 1000) collect: [ :i |
    f := [ :x | x * -2 - 40 ].
    r := Random new seed: 42.

    p := MPPerceptron  new.
    p weights: { -1 . -1 }.
    p bias: -0.5.

    i timesRepeat: [
            x := (r nextInt: 100) - 50.
            y := f value: x.
```

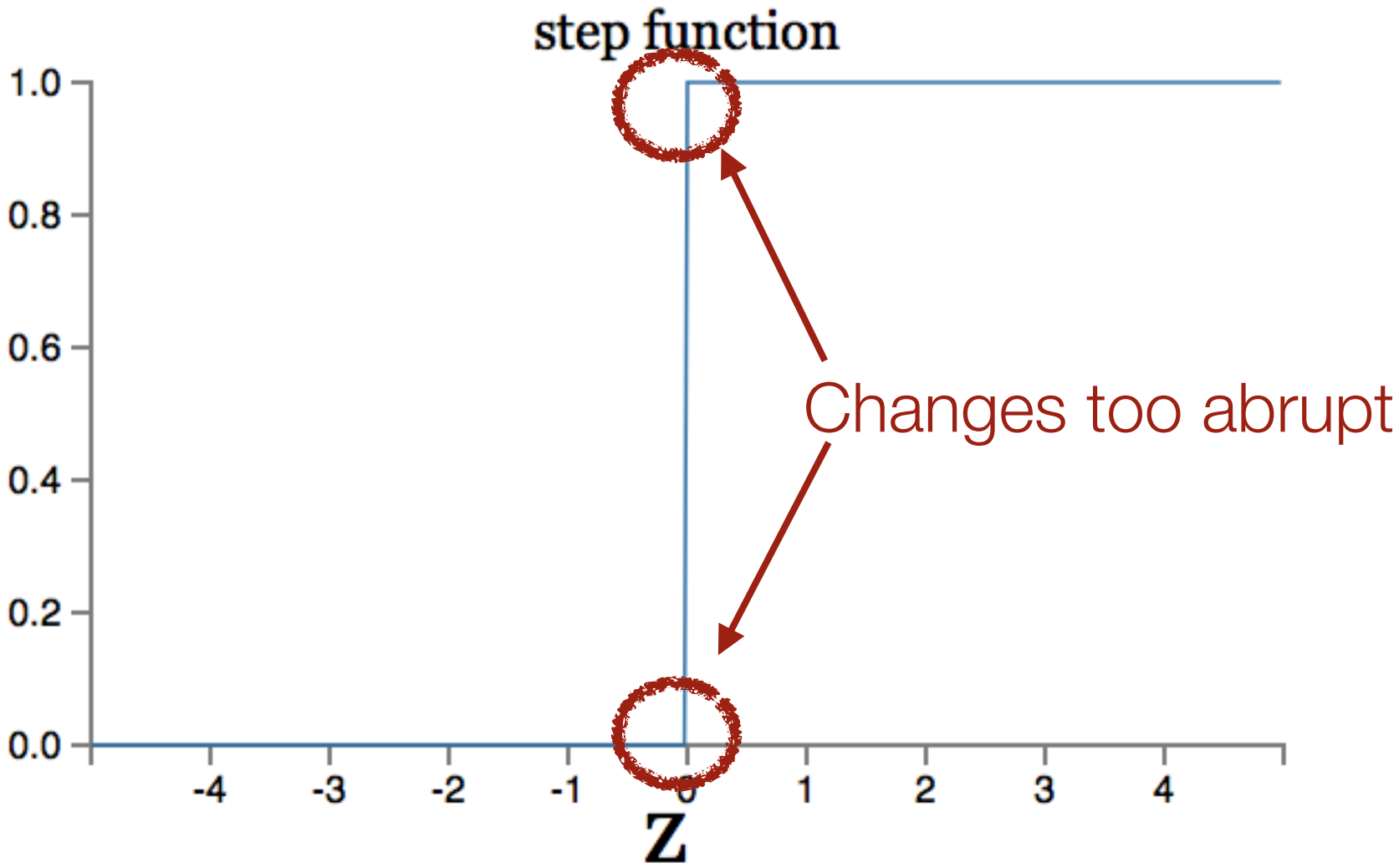



```
            x := (r nextInt: 100) - 50.
            y := f value: x.
            o := (y >= 0) ifTrue: [ 1 ] ifFalse: [ 0 ].
            error := error + (o - (p feed: { x . y })) abs.
    ].
    error := error / 100.
    error := error asFloat round: 2.
    error ]) plot
```

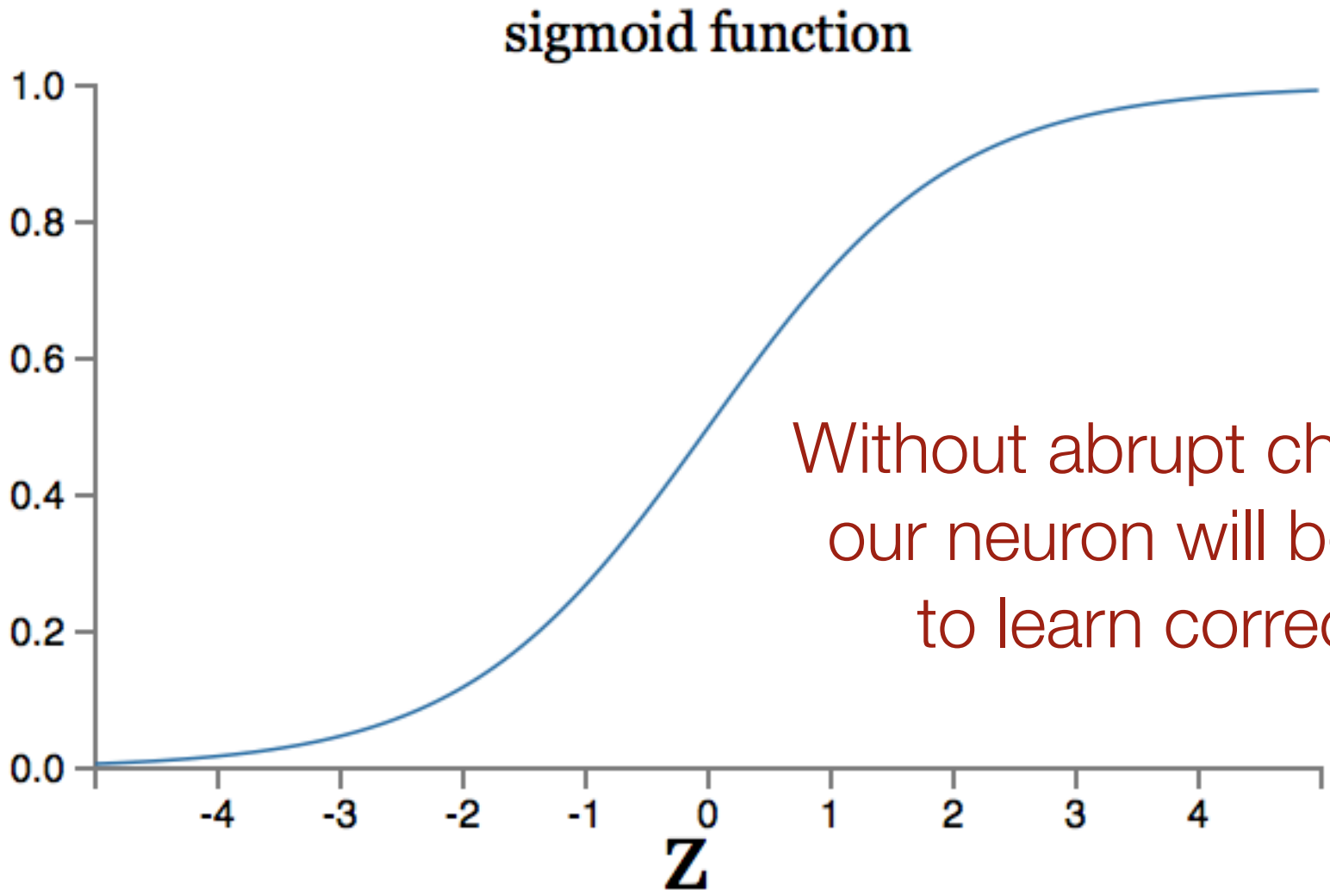
Perceptron



Perceptron


step function
1.0
0.8
0.6
0.4
0.2
0.0
-4
-3
-2
-1
0
1
2
3
4
Z
Changes too abrupt

Sigmoid neuron



Without abrupt changes, our neuron will be able to learn correctly

Sigmoid Neuron

To summarize, the output of a sigmoid neuron is

$$\frac{1}{1 + \exp(-\sum_j w_j x_j - b)}$$

```
MPNeuron>>feed: inputs
    | z |
    self assert: [ inputs isCollection ] description: 'Should be a
collection'.
    self assert: [ inputs size = weights size ] description: 'Input
should have the same size then the weights'.

    z := (inputs with: weights collect: [ :x :w | x * w ]) sum + bias .
    output := 1 / (1 + z negated exp).
    ^ output
```

```
MPNeuron>>train: inputs desiredOutput: desiredOutput
   | learningRate theError output delta |
   output := self feed: inputs.
   learningRate := 0.5.

   theError := desiredOutput - output.
   delta := (theError * (output * (1.0 - output))).

   inputs withIndexDo: [ :anInput :index |
      weights at: index put:
        ((weights at: index) + (learningRate * delta * anInput))

   bias := bias + (learningRate * delta)
```

Neural Network

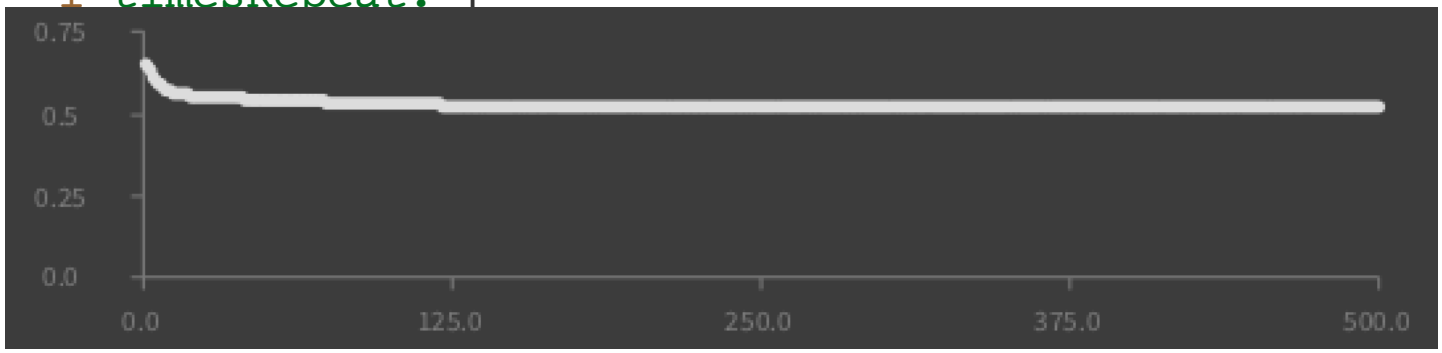
Limitation of a neuron

We have seen how to train a neuron and perceptron with a NAND, OR, AND logical gate

Let's try with a XOR

```
((1 to: 500) collect: [ :i |

    p := MPNeuron new.
    p weights: { -1 . -1 }.
    p bias: 0.5.

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

    error :=
        ((0 - (p feed: { 0 . 0})) abs +
        (1 - (p feed: { 0 . 1})) abs +
        (1 - (p feed: { 1 . 0})) abs +
        (1 - (p feed: { 1 . 1})) abs) / 4.
    error := error asFloat round: 2.
    error ]) plot
```

```
((1 to: 500) collect: [ :i |

    p := MPNeuron new.
    p weights: { -1 . -1 }.
    p bias: 0.5.

    i timesRepeat: [
```




```
    error :=
        ((0 - (p feed: { 0 . 0})) abs +
        (1 - (p feed: { 0 . 1})) abs +
        (1 - (p feed: { 1 . 0})) abs +
        (1 - (p feed: { 1 . 1})) abs) / 4.
    error := error asFloat round: 2.
    error ]) plot
```

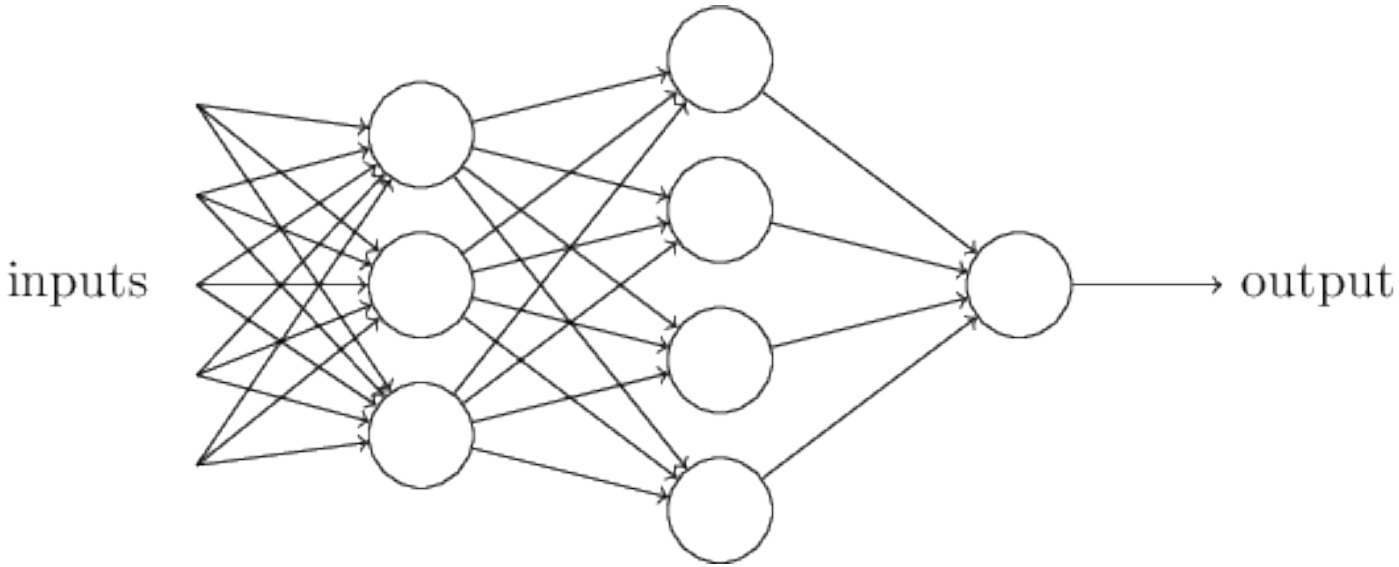
Limitation of a neuron

Our neuron cannot learn something as complex as a XOR logical gate

This is an example of why we need something more powerful than a single neuron

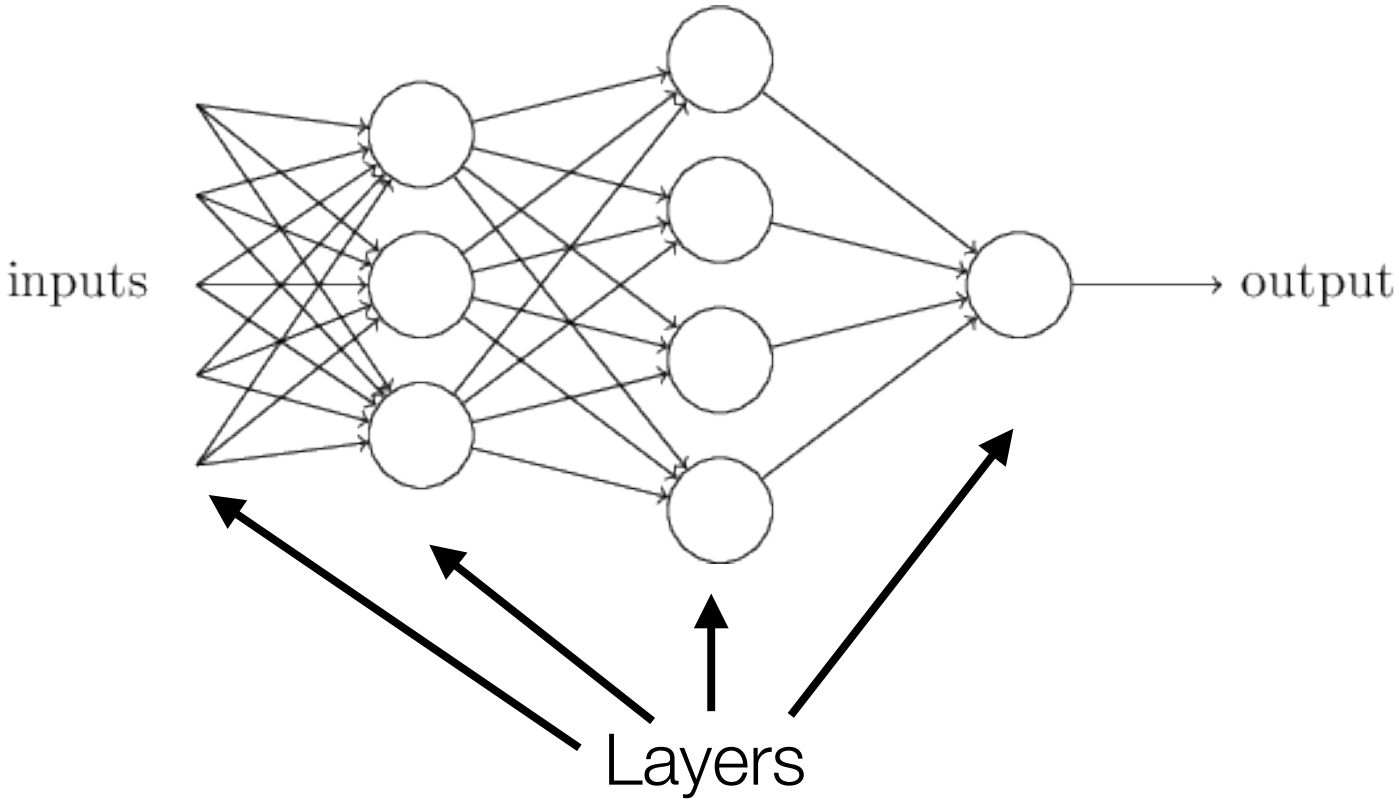
Network of neuron

A network has the following structure



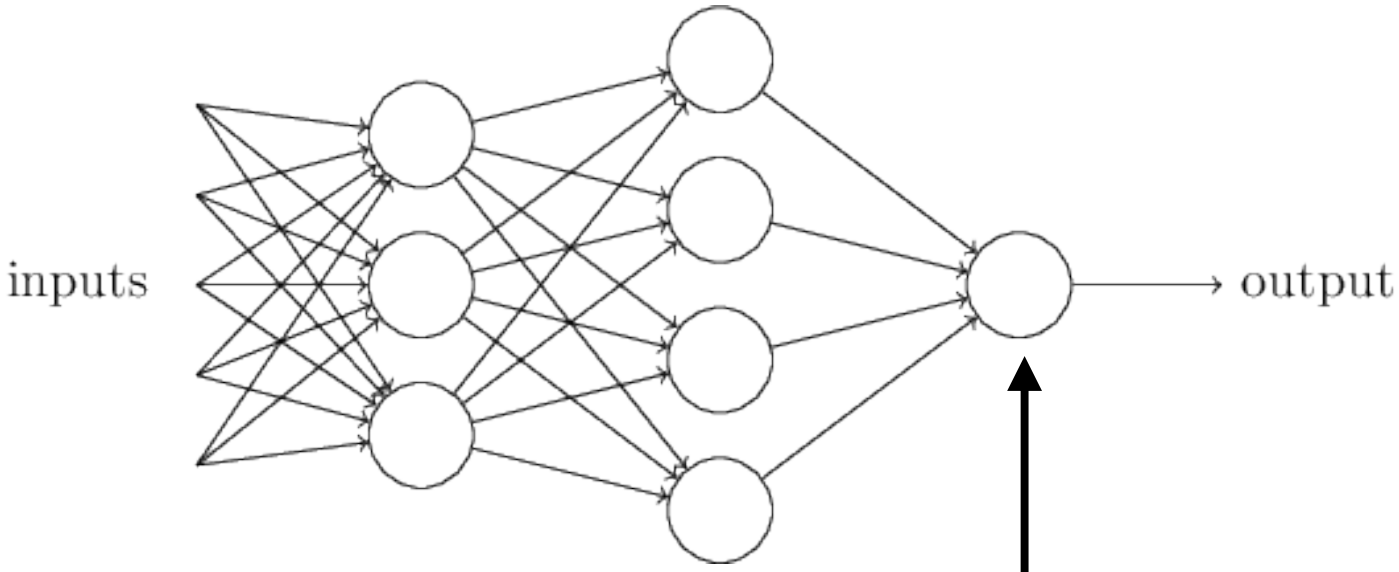
Network of neuron

A network has the following structure



Network of neuron

A network has the following structure

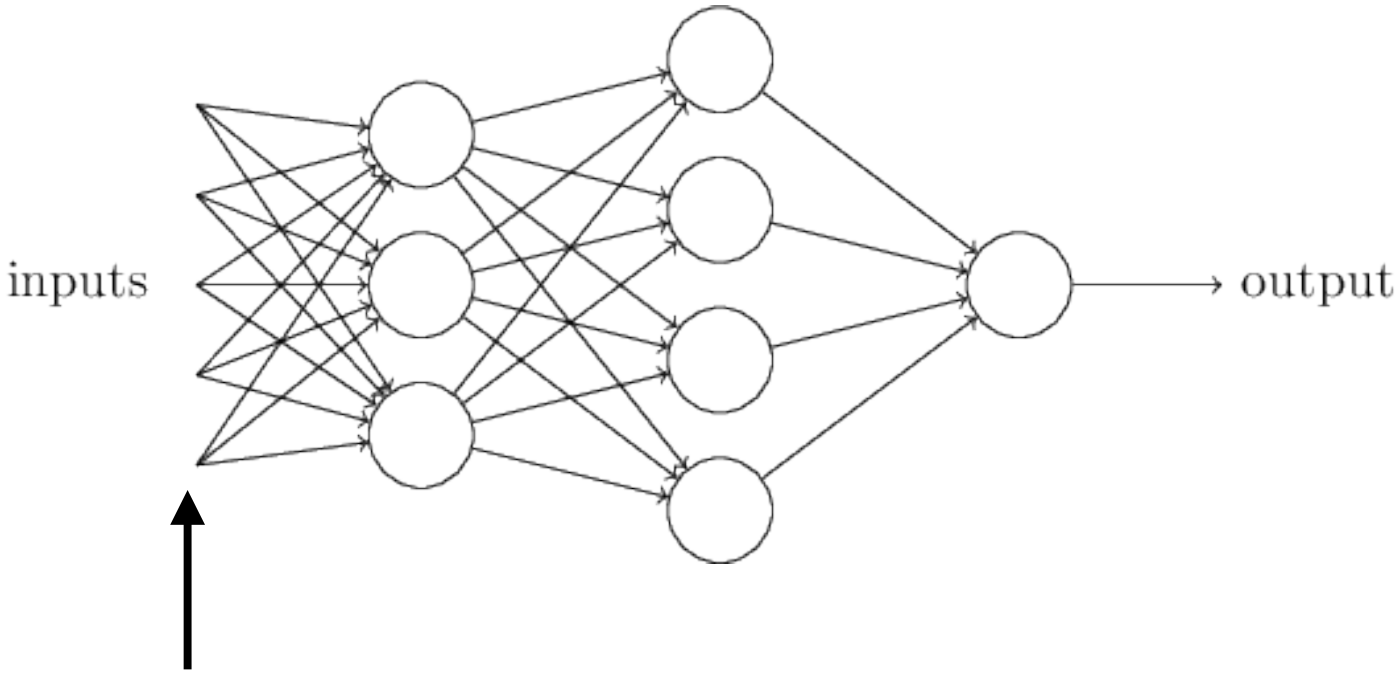


Output layer

(This example shows only one output neuron, but we could have many)

Network of neuron

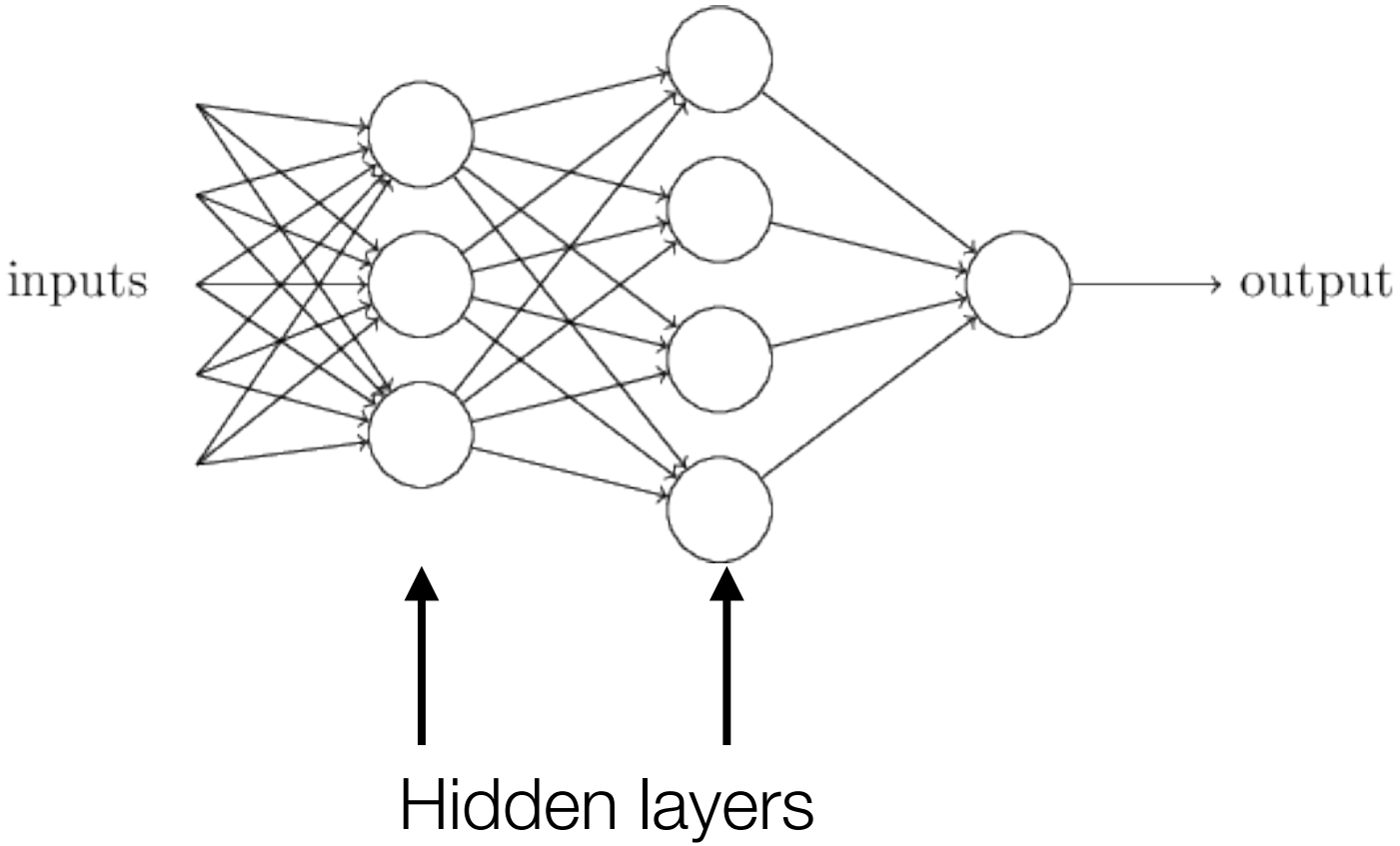
A network has the following structure



Input layer

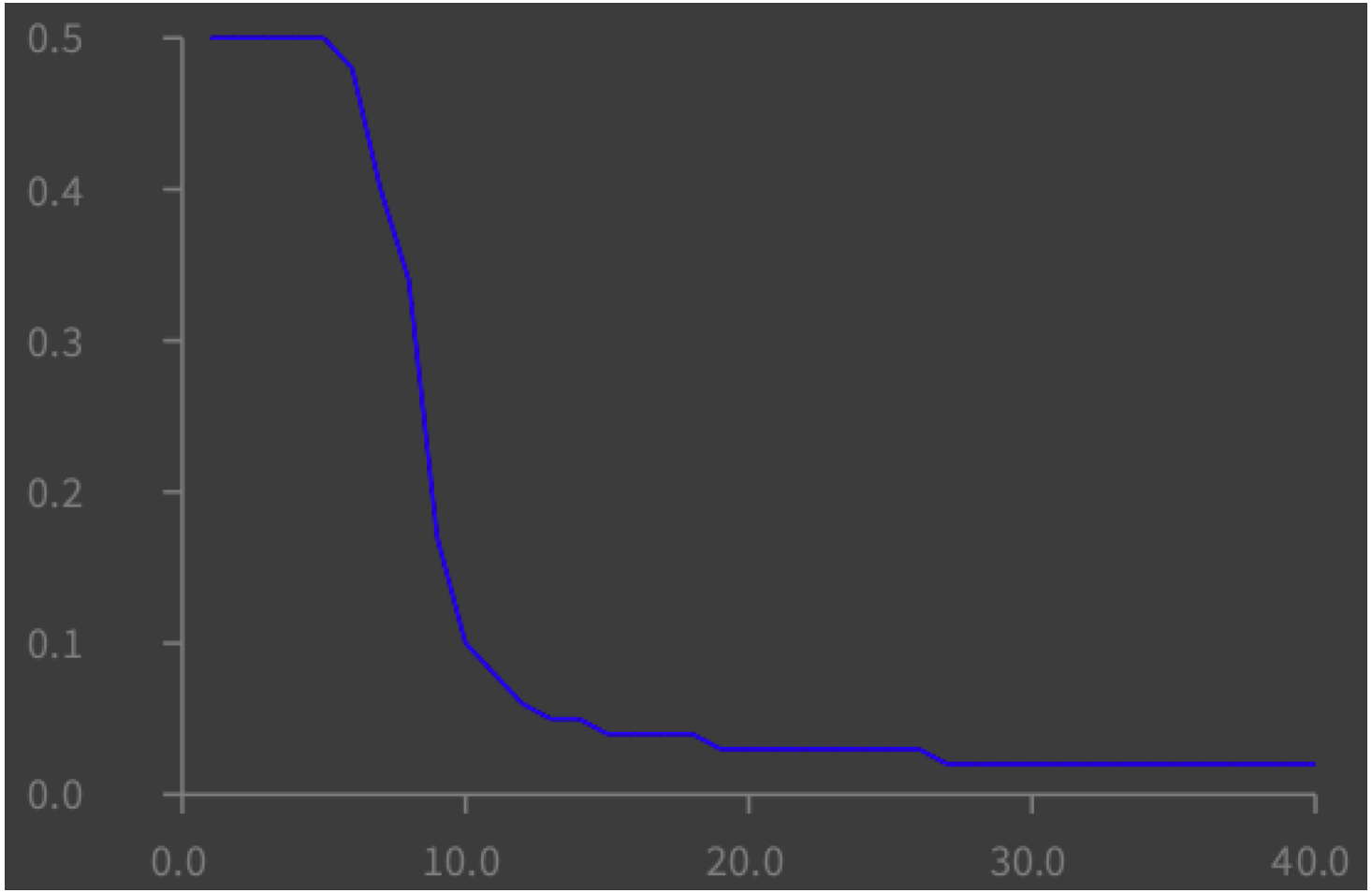
Network of neuron

A network has the following structure



```
((1 to: 10000 by: 250) collect: [ :i |

    p := NeuralNetwork new.
    p configure: 2 numberOfHidden: 2 nbOfOutput: 1.

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

    error :=
        ((0 - (p feed: { 0 . 0}) first) abs +
        (1 - (p feed: { 0 . 1}) first) abs +
        (1 - (p feed: { 1 . 0}) first ) abs +
        (0 - (p feed: { 1 . 1}) first) abs) / 4.
    error := error asFloat round: 2.
    error ]) plot
```

```
((1 to: 10000 by: 250) collect: [ :i |
```



Classifying some data

Iris

1. Title: Iris Plants Database
   Updated Sept 21 by C.Blake - Added discrepency information

2. Sources:
   (a) Creator: R.A. Fisher
   (b) Donor: Michael Marshall (MARSHALL%PLU@io.arc.nasa.gov)
   (c) Date: July, 1988

3. Past Usage:
   - Publications: too many to mention!!! Here are a few.
   1. Fisher,R.A. "The use of multiple measurements in taxonomic problems" Annual Eugenics, 7, Part II, 179-188 (1936); also in "Contributions to Mathematical Statistics" (John Wiley, NY, 1950).
   2. Duda,R.O., & Hart,P.E. (1973) Pattern Classification and Scene Analysis. (Q327.D83) John Wiley & Sons. ISBN 0-471-22361-1. See page 218.
   3. Dasarathy, B.V. (1980) "Nosing Around the Neighborhood: A New System Structure and Classification Rule for Recognition in Partially Exposed Environments". IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. PAMI-2, No. 1, 67-71.
      -- Results:
         -- very low misclassification rates (0% for the setosa class)
   4. Gates, G.W. (1972) "The Reduced Nearest Neighbor Rule". IEEE Transactions on Information Theory, May 1972, 431-433.
      -- Results:
         -- very low misclassification rates again
   5. See also: 1988 MLC Proceedings, 54-64. Cheeseman et al's AUTOCLASS II conceptual clustering system finds 3 classes in the data.

4. Relevant Information:
   --- This is perhaps the best known database to be found in the pattern recognition literature. Fisher's paper is a classic in the field and is referenced frequently to this day. (See Duda & Hart, for example.) The data set contains 3 classes of 50 instances each, where each class refers to a type of iris plant. One class is linearly separable from the other 2; the latter are NOT linearly separable from each other.
   --- Predicted attribute: class of iris plant.

Iris dataset

Obtained from:

https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data

Description of the values:

https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.names

```
g := RTGrapher new.

d := RTData new.
d dotShape
    if: [ :row | row last = 'Iris-setosa' ] fillColor: Color blue;
    if: [ :row | row last = 'Iris-versicolor' ] fillColor: Color white;
    if: [ :row | row last = 'Iris-virginica' ] fillColor: Color green.

d points: NNDataset new irisDataset.
d x: #first.
d y: #third.
g add: d.

g
```

```
| b data block d lb |
data := RTTabTable new input: (TRPlatform current downloadContent: 'http://mbostock.github.io/d3/
talk/20111116/iris.csv') usingDelimiter: $,.
data
    removeFirstRow;
    convertColumns: #( 2 3 4 5) to: #asNumber.

b := RTScatterplotMatrix new.

b objects: data values.

block := [ :n | n == n asInteger
    ifTrue: [ n asInteger ]
    ifFalse: [ n asFloat ] ].
b axisX
    numberOfTicks: 5;
    rotateLabels;
    labelConversion: block.
b axisY
    numberOfTicks: 5;
    labelConversion: block.

b lineShape: (RTStyledMultiLine new
    dashedLine;
    width: 0.5; yourself
    ).

d := Dictionary new
    at: 'setosa' put: Color red;
    at: 'versicolor' put: Color green;
    at: 'virginica' put: Color blue; yourself.
b shape circle
    size: 3.5;
    color: [ :a | d at: a first ].
```

```
dataSet := NNDataset new irisDatasetWithNumericalLast.
dataSet := dataSet shuffled.
cut := (dataSet size * 0.5) asInteger.
dataSetTraining := dataSet first: cut.
dataSetGuessing := dataSet last: (dataSet size - cut).


p := NeuralNetwork new.
p configure: 4 numberOfHidden: 5 nbOfOutput: 3.

p train: dataSetTraining nbEpoch: 2000.


precision := (dataSetGuessing select: [ :row |
    (p predict: row allButLast) = row last ]) size / dataSetGuessing size.

'Prediction of precision = ', (((precision round: 4) * 100) asString), '%'
```

Prediction of precision = 96.0%