Foundations of Artificial Neural Networks: Linear Discriminants, Neuron and Perceptron

Brain – the best learning system

- Artificial Neural Networks were inspired by the functionality of the human brain
- The brain consists of neurons and synapses
- And their sheer number makes it the best learning system
- Artificial Neural Networks try to simulate it algorithmically with the help of numbers

Basics of machine learning

- machine learning algorithms learn from data
- data is usually spilt into training and test data

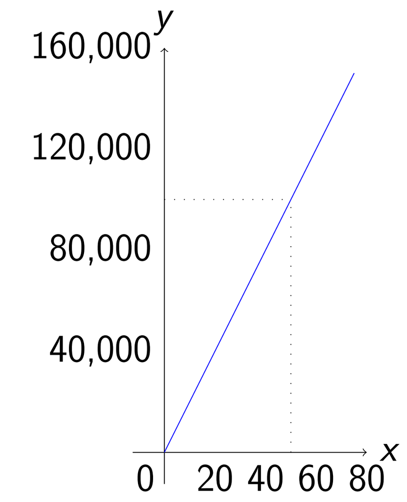
Guessing the price

House prizes in square meters and euro

square meter	prize
20	40 000
40	80 000
50	?
60	120 000
80	160 000

Can you find the prize for a house of 50 square meters?

Geometric solution

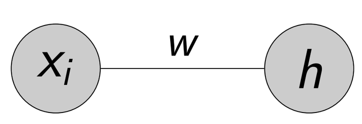


From calculation point of view

$y(x) = xw$
$y(20) = 20 * 2,000$
$y(20) = 40,000$
This means you can pay for a square meter 2,000 euro
So, learning is about finding the w in $y(x) = xw$ which is a linear function. If you have a house which is 50 square meters then the prize should be around $y(50) = 50 * 2,000 = 100,000$ euros. In real usage, also a bias value is added to the calculation

$$y(x) = wx + b. \tag{1}$$

if we call the prediction h instead of $y(x)$



An algorithm for training has

1. a set of inputs, in our case $x = [20, 40, 60, 80]$
2. a weight w
3. a set of targets or labels $t = [40, 80, 120, 160]$. Note that we omitted the last three zeros for readability reasons.
4. a learning rate which is usually set to a small random number $\eta = 0.00002$
5. A maximum number of iterations to update the weight. This number of iterations is often called epochs

The algorithm does for training

1. initialize w with a random number
2. compute a prediction with $w \cdot x_i$ The outcome will be put into h_i variable
3. compute the error the prediction made and see how different the output is from the desired target or label $\Delta w = \eta \cdot (t_i - h_i) \cdot x_i$
4. update the weight with $w \leftarrow w + \Delta w$

Python implementation of training with house prizes

```
inputs = [20, 40,  60, 80]
targets = [40, 80, 120, 160]
weight = 0.5
eta = 0.00002
epoch = 26
print "--------------------"
print inputs
print targets
for e in range(epoch):
        print "---->>>>>> eopch " + str(e)
        for i in range(4):
                h = weight * inputs[i]
                print "the h prediction is: " + str(h)
                print "but the target is: " + str(targets[i])
                diff = targets[i] - h
                print "the difference: " + str(diff)
                weight += eta * diff * inputs[i]
                print "new weight: " + str(weight)
```

Python implementation of training with house prizes

```
epoch 0
the h prediction is: 10.0
but the target is: 40
the difference: 30.0
new weight: 0.512
the h prediction is: 20.48
but the target is: 80
the difference: 59.52
new weight: 0.559616
the h prediction is: 33.57696
but the target is: 120
the difference: 86.42304
new weight: 0.663323648
the h prediction is: 53.06589184
but the target is: 160
the difference: 106.93410816
new weight: 0.834418221056
```

Python implementation of training with house prizes

```
epoch 24
the h prediction is: 39.9295376652
but the target is: 40
the difference: 0.0704623347734
new weight: 1.9965050682
the h prediction is: 79.8602027278
but the target is: 80
the difference: 0.13979727219
new weight: 1.99661690601
the h prediction is: 119.797014361
but the target is: 120
the difference: 0.202985639221
new weight: 1.99686048878
the h prediction is: 159.748839102
but the target is: 160
the difference: 0.251160897596
new weight: 1.99726234622
```

Python implementation of training with house prizes

```
epoch 25
the h prediction is: 39.9452469243
but the target is: 40
the difference: 0.0547530756758
new weight: 1.99728424745
the h prediction is: 79.8913698979
but the target is: 80
the difference: 0.108630102141
new weight: 1.99737115153
the h prediction is: 119.842269092
but the target is: 120
the difference: 0.157730908309
new weight: 1.99756042862
the h prediction is: 159.804834289
but the target is: 160
the difference: 0.195165710547
new weight: 1.99787269376
```

Vectorization of the process

- to get rid of for loop
- to speed up the running time
- you can use python library *numpy* or Java library *NDArray* or many other libraries in other programming languages
- scalar are denoted with lower case italic letters, vectors with lower case bold italic letters and the matrices with upper case bold italic letters
- so, our equation for prediction changes to $\boldsymbol{h} = w \cdot \boldsymbol{x}$
- so, our equation to train changes to $\Delta w = \eta \cdot (\boldsymbol{t} - \boldsymbol{h}) \cdot \boldsymbol{x}$
- have a look into linear algebra for more details

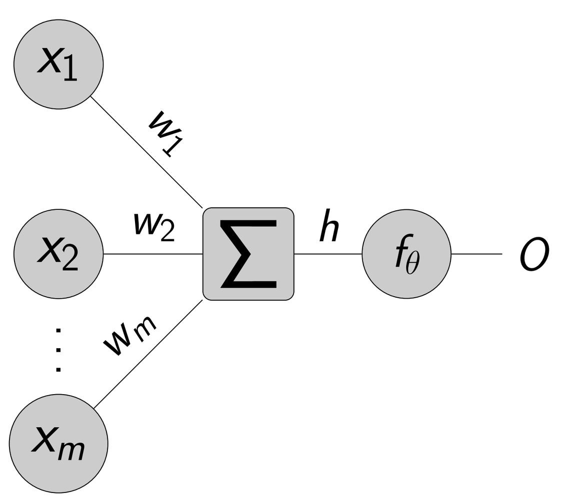
Vectorization in our case means

- to predict $\boldsymbol{h}$ you use $w \cdot \begin{pmatrix} 20 \\ 40 \\ 60 \\ 80 \end{pmatrix}$
- when $w = 1.5$ then $\boldsymbol{h} = 1.5 \cdot \begin{pmatrix} 20 \\ 40 \\ 60 \\ 80 \end{pmatrix} = \begin{pmatrix} 30 \\ 60 \\ 90 \\ 120 \end{pmatrix}$
- $\boldsymbol{t} - \boldsymbol{h}$ means $\begin{pmatrix} 40 \\ 80 \\ 120 \\ 160 \end{pmatrix} - \begin{pmatrix} 30 \\ 60 \\ 90 \\ 120 \end{pmatrix} = \begin{pmatrix} 10 \\ 20 \\ 30 \\ 40 \end{pmatrix}$
- $0.00002 \cdot \begin{pmatrix} 10 \\ 20 \\ 30 \\ 40 \end{pmatrix} \cdot \begin{pmatrix} 20 \\ 40 \\ 60 \\ 80 \end{pmatrix} = \begin{pmatrix} 0.0002 \\ 0.0004 \\ 0.0006 \\ 0.0008 \end{pmatrix} \cdot \begin{pmatrix} 20 \\ 40 \\ 60 \\ 80 \end{pmatrix} =$
  $0.004 + 0.016 + 0.036 + 0.064 = 0.12$
- updating the weight $1.5 + 0.12 = 1.62$

python implementation of training with house prizes with vectors

```
inputs = [20, 40,  60, 80]
targets = [40, 80, 120, 160]
weight = 0.5
eta = 0.00002
epoch = 26
print "--------------------"
print inputs
print targets
for e in range(epoch):
        print "---->>>>>> eopch " + str(e)
        h = dot(weight, inputs)
        print "the h prediction is: " + str(h)
        print "but the target is: " + str(targets)
        diff = targets - h
        print "the difference: " + str(diff)
        weight += eta * dot(diff, inputs)
        print "new weight: " + str(weight)
```

Brain – Let us look at a neuron



A neuron has

1. a set of inputs x_i
2. a set of weighted synapses w_i
3. an adder
4. an activation function

$$h = \sum_{i=1}^{m} w_i x_i \tag{2}$$

$$o = g(h) = \begin{cases} 1 & \text{if } h > \theta \\ 0 & \text{if } h \leq \theta \end{cases} \tag{3}$$

So, if we have 3 inputs [1,0,1]and 3 weights, then

A very basic python implementation

```
inputs = [1, 0, 1]
weights = [0.5, 0.7, 0.4]

# Iterate over inputs and weights to calculate h
h = 0
for input in inputs:
    for weight in weights:
        h += input*weight

# This is a very basic test for activation
if (h > 0.5):
    print "activated!"
if (h <= 0.5):
    print "not activated!"
```

A Perceptron can be trained and has

5. targets t(or so called labels)
6. an error function computes the difference between the target and the real output: $t_k - y_k$. In order to be able to fire even with a negative value we multiply this with x_i

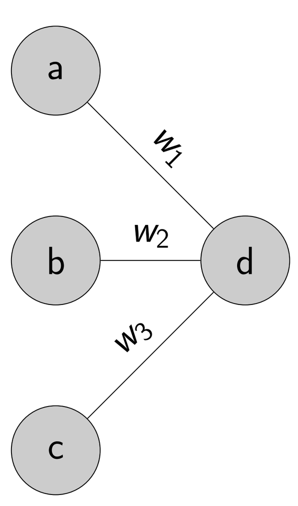
$$\Delta w_{ij} = \eta(t_j - y_j) \cdot x_i \tag{4}$$

7. a learning rate called η to determine how fast to change the weights

$$w_{ij} \leftarrow w_{ij} + \Delta w_{ij} \tag{5}$$

8. and a maximum number of iterations in order to train T

What is the input for the neuron d?



input neurons	connections
a	w1
b	w2
c	w3

If we change it to numbers:

input neurons	connections
1	0.5
0	0.7
1	0.4

Vector solution

Here we multiply two vectors to predict:

$$h = \boldsymbol{w} \cdot \boldsymbol{x} \tag{6}$$

In the real world scenario:

$$(w_1, w_2, w_3) \cdot \begin{pmatrix} a \\ b \\ c \end{pmatrix} = w_1 a + w_2 b + w_3 c. \tag{7}$$

$$(0.5, 0.7, 0.4) \cdot \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} = 1 \cdot 0.5 + 0 \cdot 0.7 + 1 \cdot 0.4 = 0.5 + 0 + 0.4 = 0.9. \tag{8}$$