Computational Intelligence WS1718
Introduction to Artificial Neural Networks

Max Springenberg, 177792

1 Abstraction And Models

A neural net tries to replicate the behaviour of a neuron.
A neuron consists of dendrites that carry the signal input, a nucleus/ cell body that processes the signal and a axon and synapse that carry out signals.

Now, there are different methods to replicate such behaviour.

1.1 McCulloch-Pitts Neuron 1943

We take input vaues $x_1, .., x_n$ with $x_i \in \mathbb{B}, 0 < i \leq n$, apply them to a function and use the output.

obviously such a "neuron" is either active or inactive.
the function "fires" a 1, whenever a threshold Θ is reached.
therefore gates like OR and AND, aswell as NOT can be replicated with such neurons.
In consequence all logical functions can be replicated, using such neurons.

Wheighted Neural Nets are NN, that multiply input values, before handing them to the function. Those are equivilent, since for every wheighted net a unweighted net, that takes the input values multiple times and adjust the threshold, can be constructed.

1.1.1 conclusion

PROS	CONS
feed forward (computes any logical /boolean function)	almost identical to logical circuits
recursive (can simulate any DFA)	difficulties in construction
	no good learning algorithm available

1.2 The Perceptron, Rosenblatt 1958 Minsky-Papert 1969

The perceptron by Rosenblatt is a rather complex model, that has been reduced by Minsky and Papert to it's core neccassities, resulting in the Minsky-Papert perceptron.

The essential difference of the MPP to the MCP is, that its input values are within $\mathbb{R}$, rather than $\mathbb{B}$

1.2.1 What can a single MPP do?

Let W be the set of weights applied to the input signals, with $W \subseteq \mathbb{R}$
The function of a MPP with binary output can be defined as

$$f(W, X) = (w_1 * x_1 + ... + w_n * x_n \geq \Theta)$$

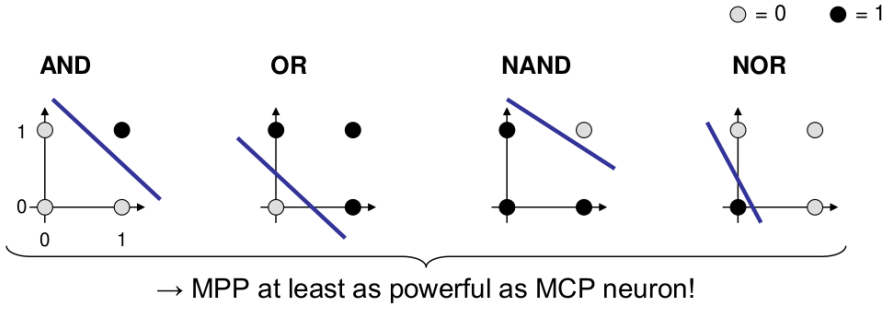
Let us isolate an x_i

$$x_i = \frac{\Theta - (w_1 * x_1 + ... + w_{i-1} * x_{i-1} + w_{i+1} * x_{i+1} + ... + w_n * x_n)}{w_i}$$

for $n = 2$ we get

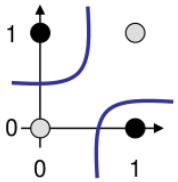
$$x_2 = \frac{\Theta - w_1 * x_1}{w_2}$$

leading to a linear curve seperating $\mathbb{R}$ for true and false output.



To solve the xor functions a nonlinear function and an additional "neuron" is required.

XOR $g(x_1, x_2) = 2x_1 + 2x_2 - 4x_1x_2 - 1$ with $\theta = 0$



$g(0,0) = -1$
$g(0,1) = +1$
$g(1,0) = +1$
$g(1,1) = -1$

So far all modules obtained their wheights and threshold by construction.
Now we want to have the neurons figure those out themselves, by training/ learning.

1.3 Perceptron learning

Let test examples with correct I/O - behaviour be given.
Following we apply the principle:

1. choose initial weights in arbitrary number
2. feed in test pattern
3. if outpu of perceptron is wrong. then change the wheights
4. goto (2.) until the output is correct for all test patterns

We emerge the subsets N and P of the original dataset, with
$N :=$ set of examples with negative output $(\forall x \in N.x = 0)$
$P :=$ set of examples with positive output $(\forall x \in P.x = 0)$

The algorithm Alg works by the following principle:

1. choose $w_{t=0}$ randomly
2. choose arbitrary $x \in P \cup N$
3. if $x \in P \wedge w_t * x > 0$ then goto (2.)
   if $x \in N \wedge w_t * x \leq 0$ then goto (2.)
4. if $x \in P \wedge w_t * x \leq 0$ then
   $w_{t+1} = w_t + x; ++t; goto(2.)$
5. if $x \in N \wedge w_t * x > 0$ then
   $w_{t+1} = w_t - x; ++t; goto(2.)$
6. (I/O correct for all examples) ? stop : run again

A major drawback with this algorithm is:
$Alg \in O(exp)$