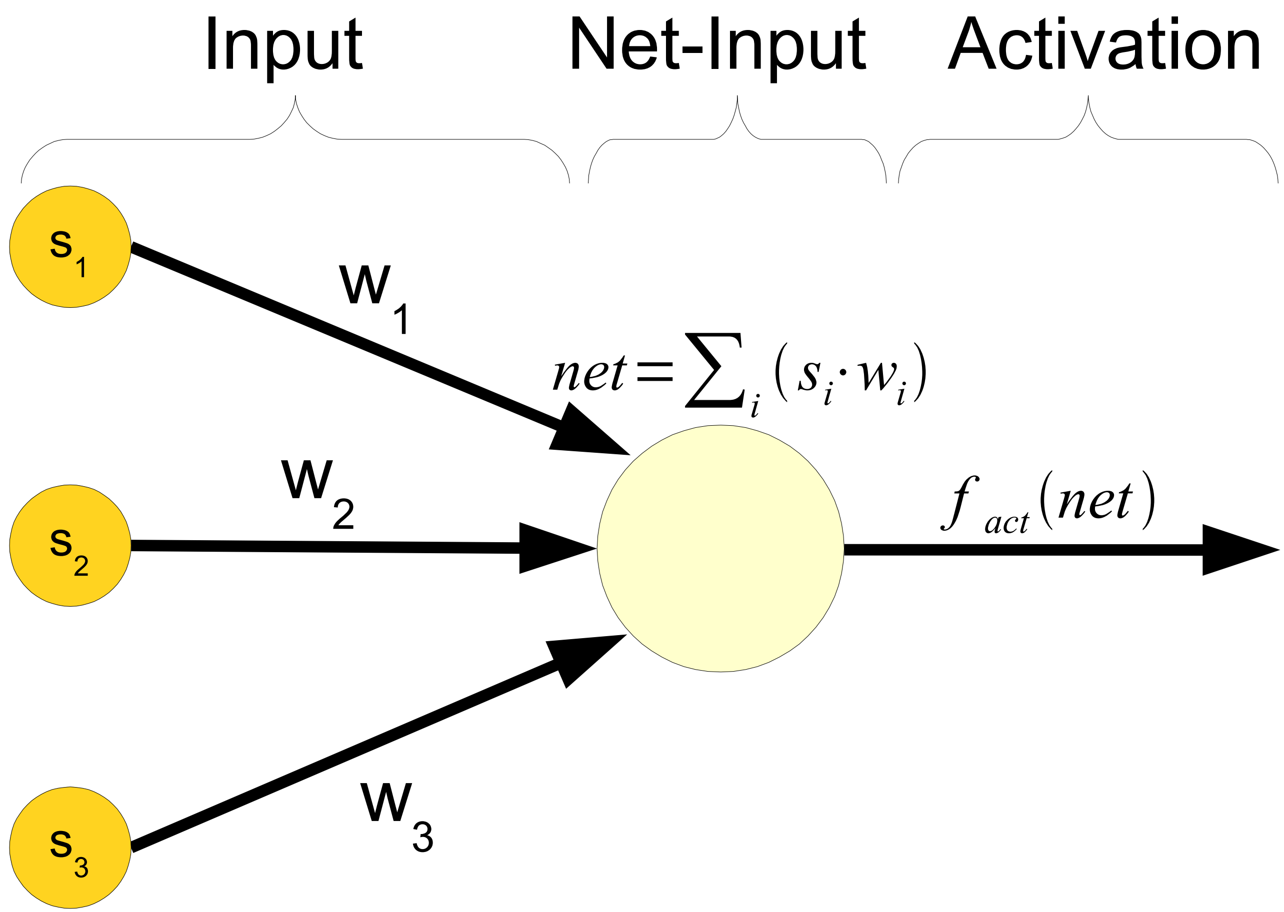
Artificial Neural Nets

- Preparation
  - Functional equations
  - Geometrical equations
    - Parametric equations
    - Hessian normal form
- Perceptron
- Neural Nets of Perceptrons
- Learning of weights
- How neural learning works

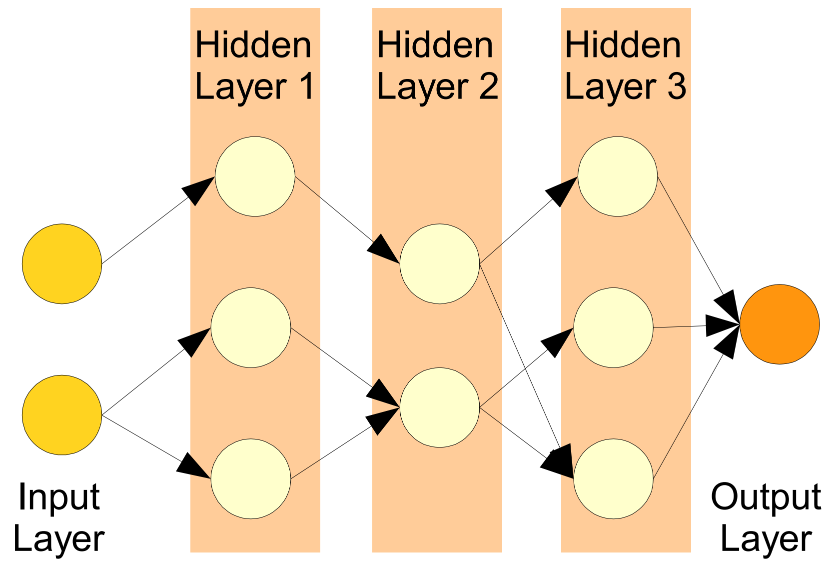
Connectionistic Neuron=Perceptron

- Perceptron
- Input s_i
- Weights w_i
- Net-Input
- activation

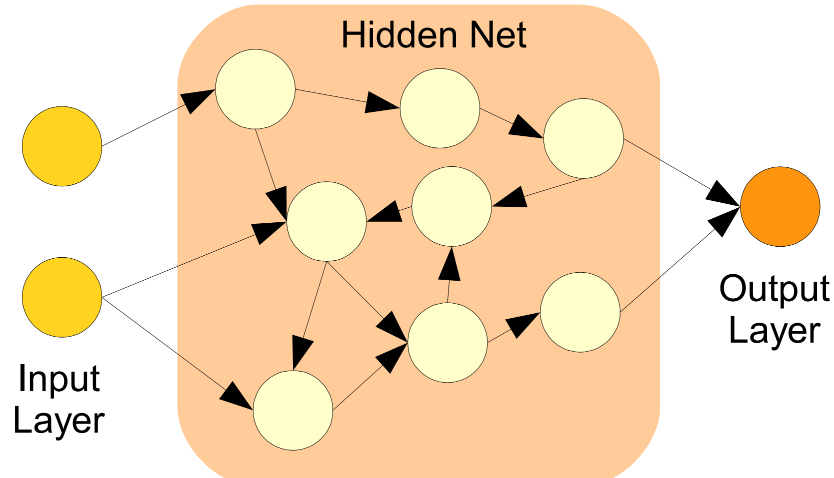


Connectionistic Nets

- Layered nets
  - No loops inside layer
  - Clear direction of updates



- Dynamic nets
  - Loops inside net
  - No clear update direction
  - Activation can explode!

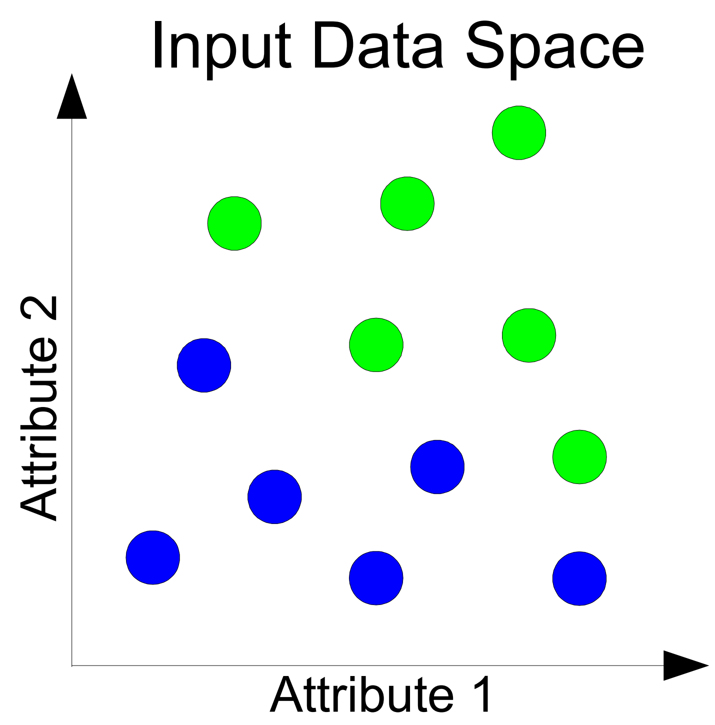


Difference to biology

- Clocked update (no spike trains)
- Not in parallel (in ordinary computers)
- Only one type of neuron/receptor
- No habituation
- Most connectionistic systems are stable in structure, only flexible in input/output/Weights

A problem artificial NN can solve

- Supervised learning
- Domain: Reputation of customers for a loan
  - Attributes
    - Income (Attribute 1)
    - Place of living (Attribute 2)
- Prediction of ordinal attribute (class is given)
  - Pay loan back (green)
  - Did not pay credit back (blue)
- Task: find a model that can sort green from blue



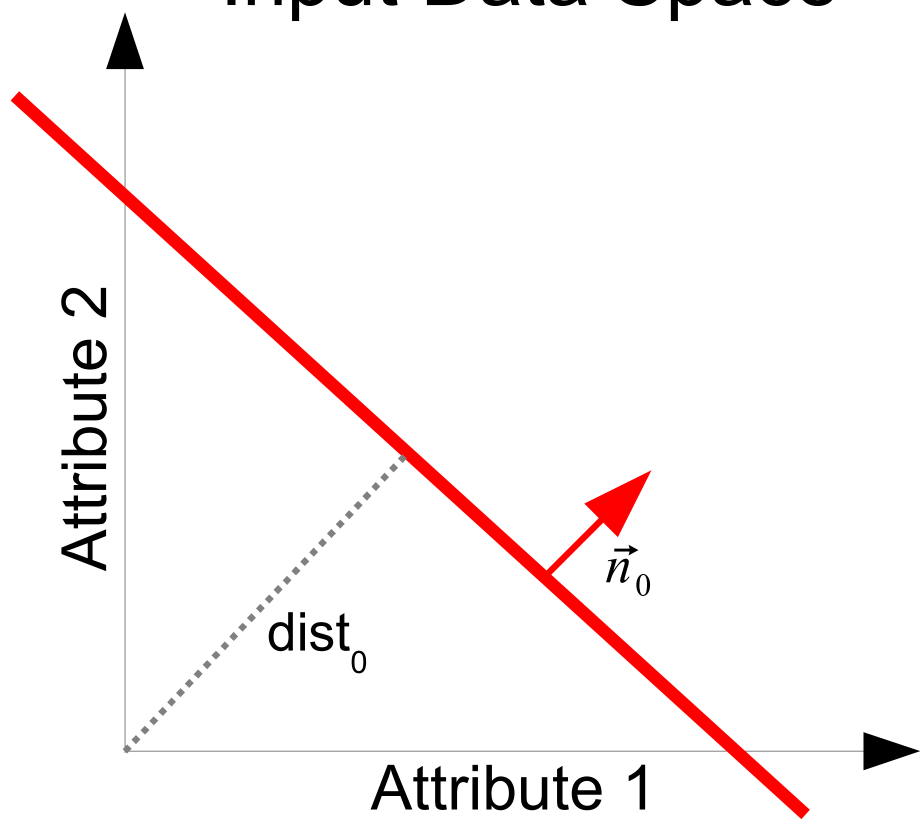
The model

- A geometrical model
- A straight line separating the input space
  - Hessian normal form

    $$\vec{r} \cdot \vec{n}_0 = dist_0$$

    - Normal vector n_0
    - Distance to origin
  - Here I calculate the normal vector from angle

Input Data Space



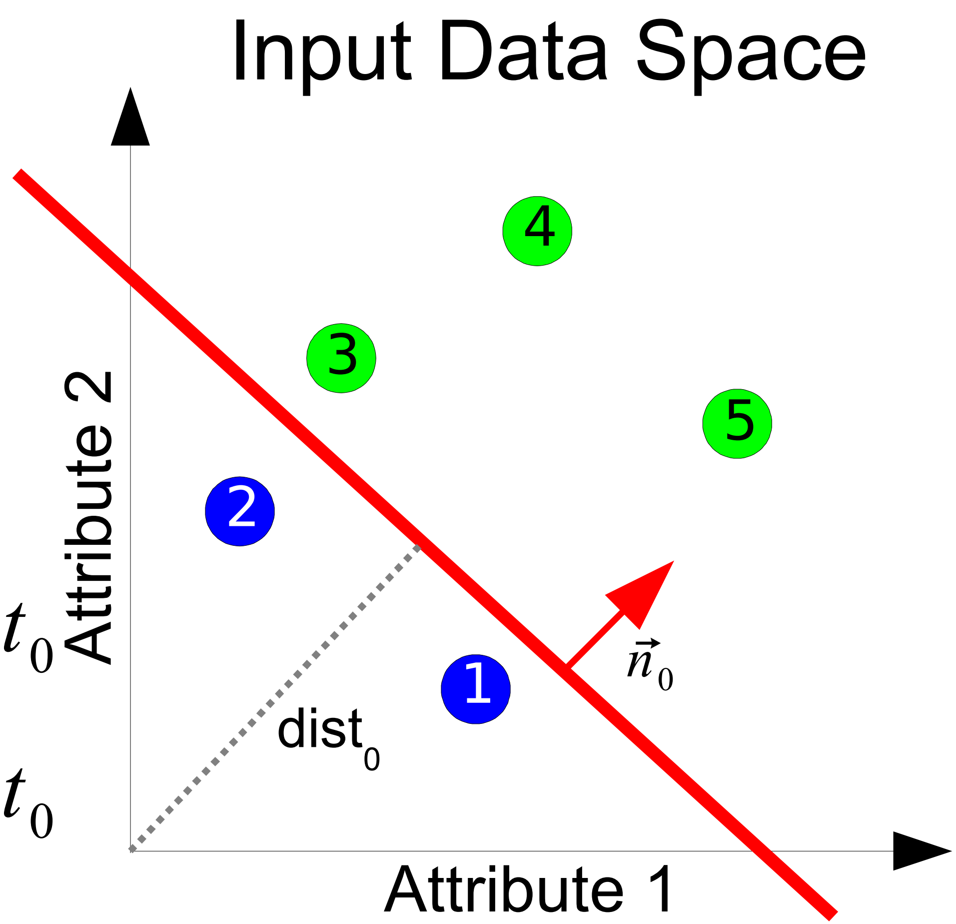
Linear separation of input space

- How prediction works
- Plane defined as

$$\vec{r} \cdot \vec{n}_0 = dist_0$$

- Predicted class:

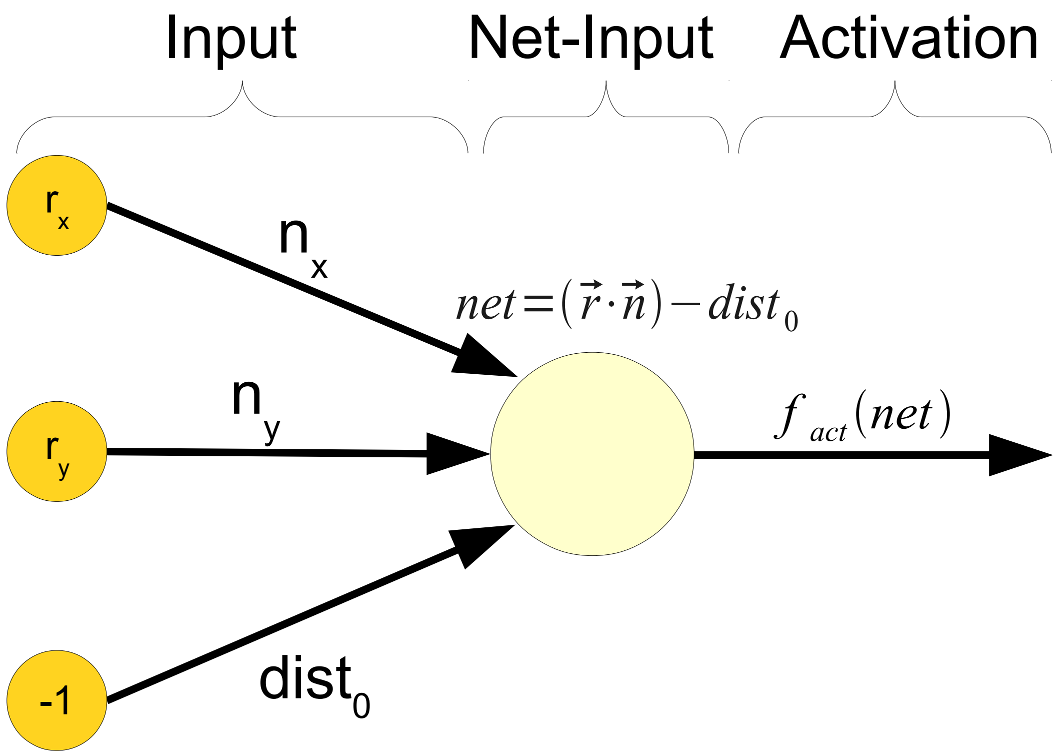
$$\begin{cases} \text{class}_1 & \text{if } \vec{r}_1 \cdot \vec{n}_0 < dist_0 \\ \text{class}_2 & \text{if } \vec{r}_1 \cdot \vec{n}_0 \geq dist_0 \end{cases}$$

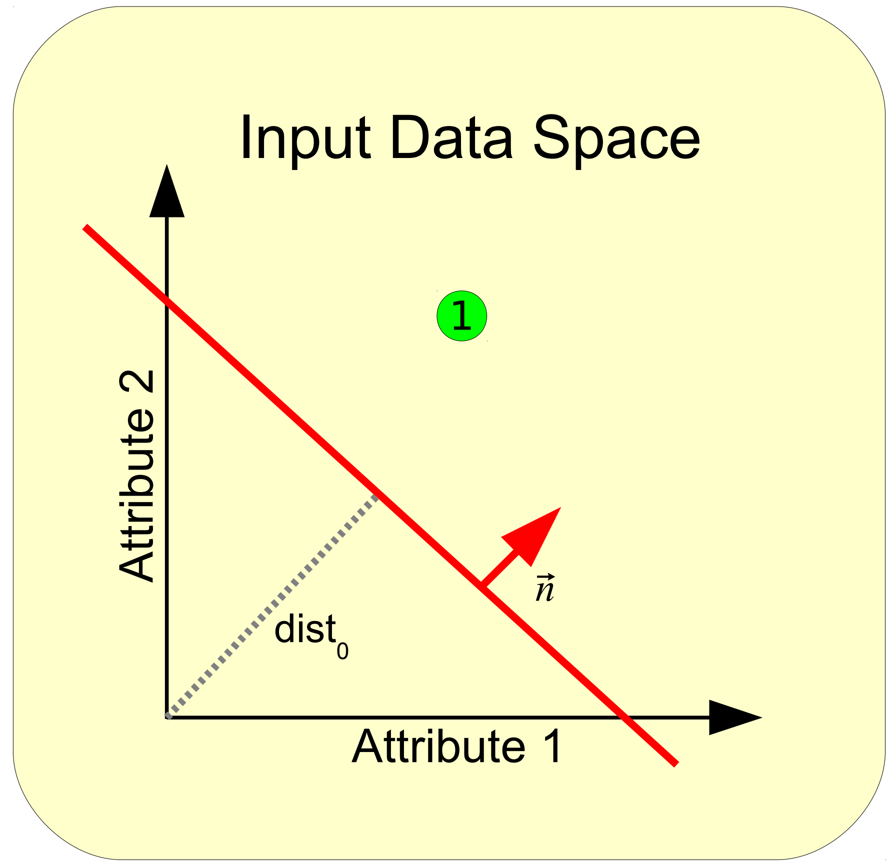


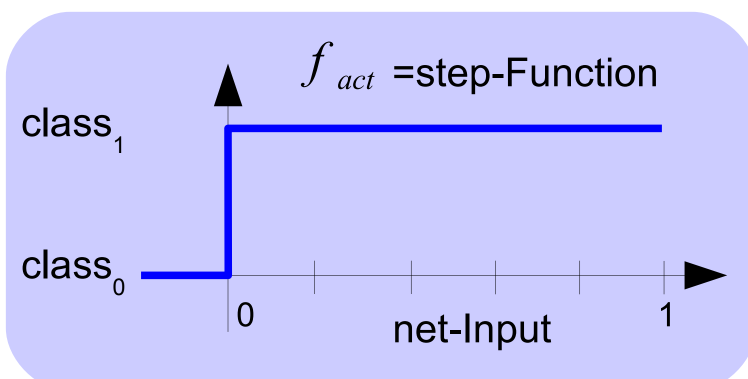
Sample 1 represented by

$$\vec{r}_1 = (attribute_1, attribute_2)$$

Graphical Interpretation of weights







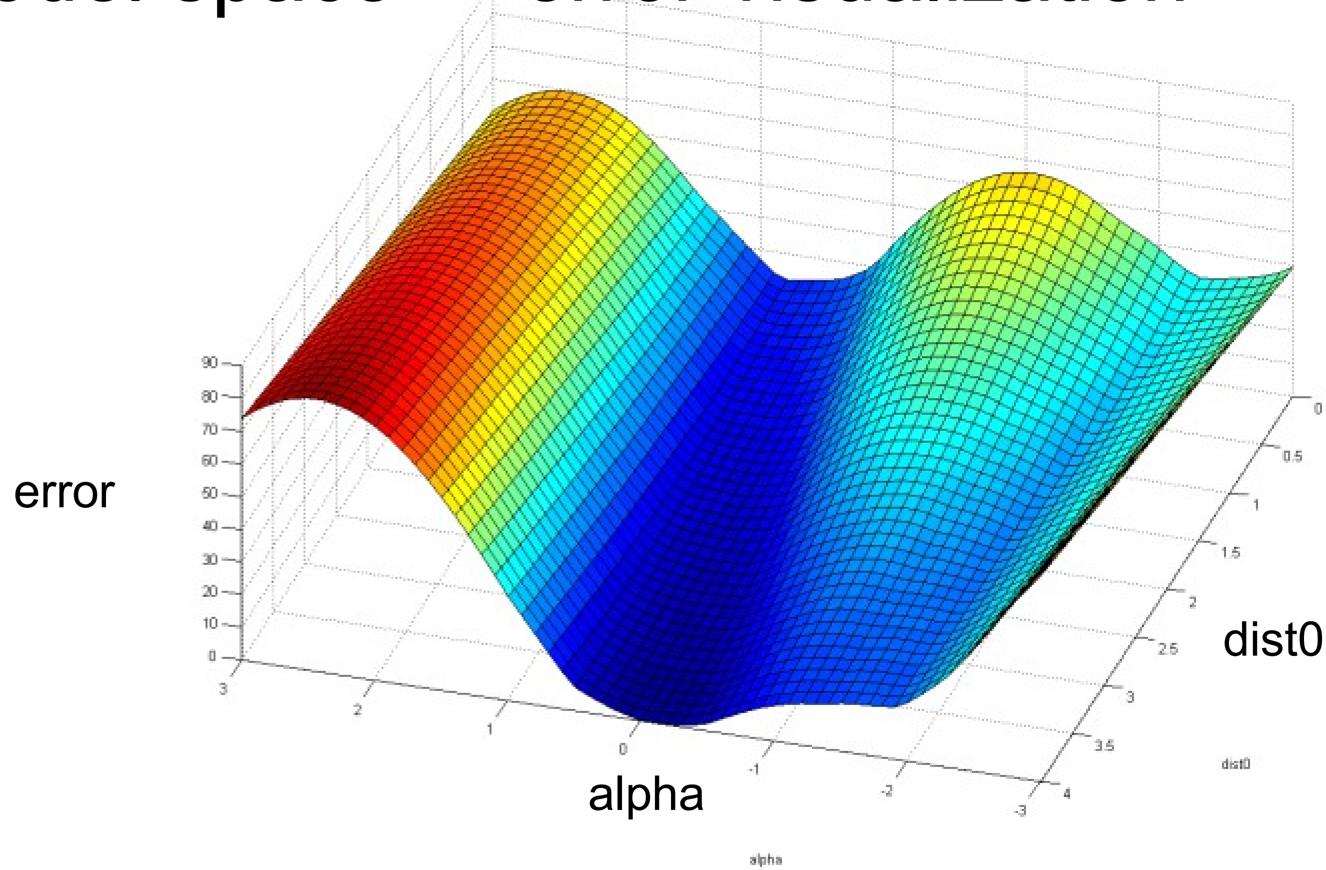
Weights n are in range (-1,+1) with length(n)=1

Practise

- Load the Material.zip for this lecture
- data=generateData;
- showData(data,alpha,dist)
  - Alpha is the angle in RAD in [-pi/2 .. +pi/2] for $(\vec{n}_0)$
  - dist is the distance from the origin
  - Show two classes in colors blue and green
  - Correctly classified samples as point „.“ missed samples as circle „o“
- gradientDescent(data,0.01,-pi/4,2)
  - Starts a gradient Descent learning process

Gradient Descent (delta rule)

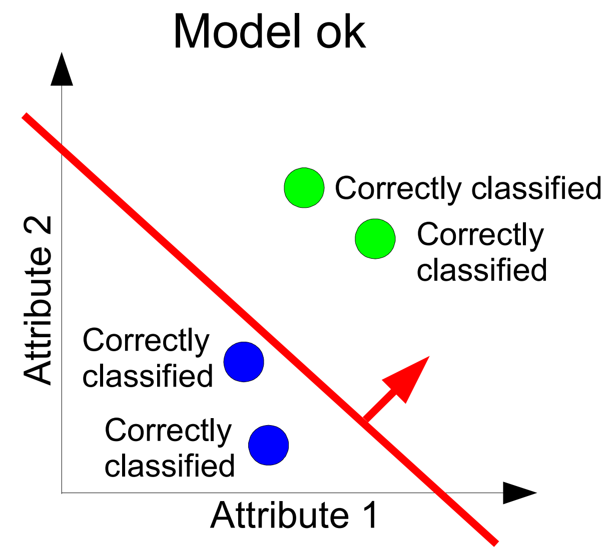
- Model space -> error visualization



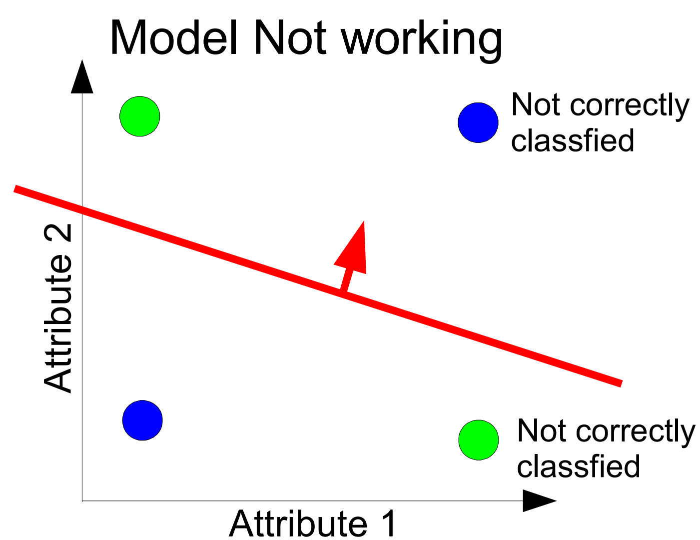
- gradientDescent(data,stepSize,alpha,dist0)
  - StepSize=0.01, alpha=-pi/4, dist0=2

Limits of the one-perceptron model

- Inspect the uglyData.mat

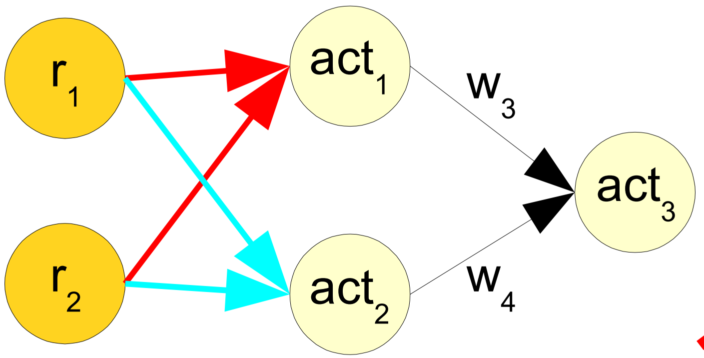


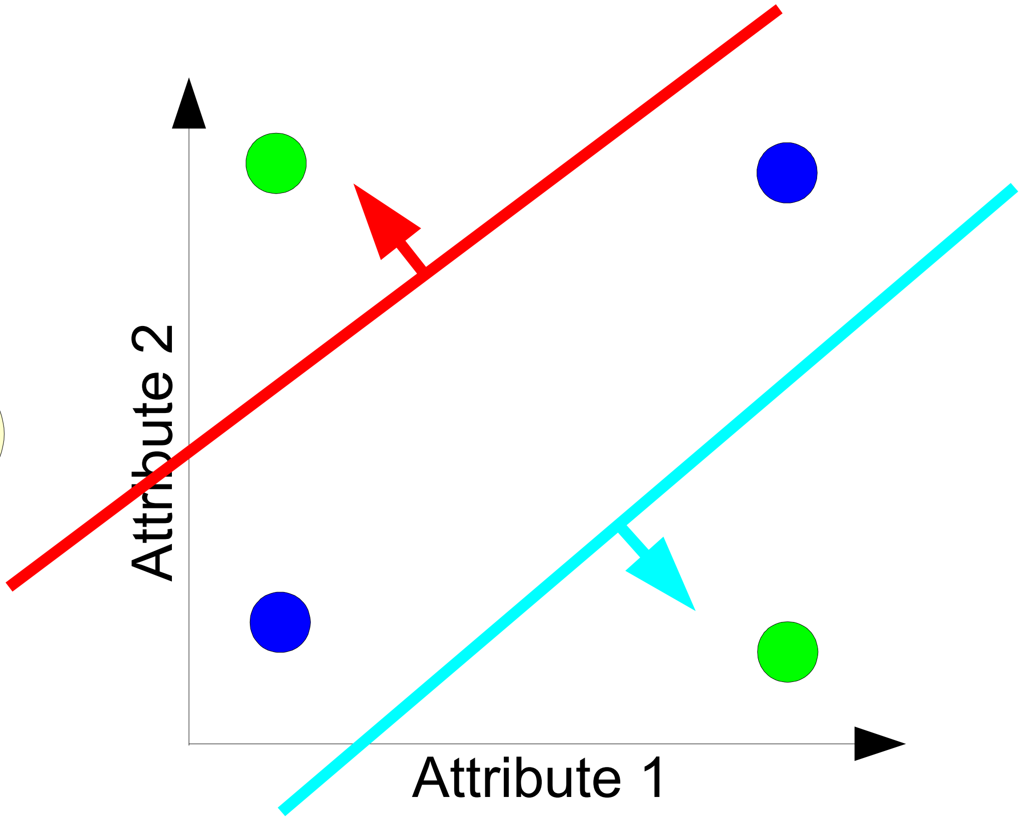
- The perceptron model can separate the green from the blue class



- The perceptron model is not able to separate the input space linearly with the given data

Solution: use two perceptrons

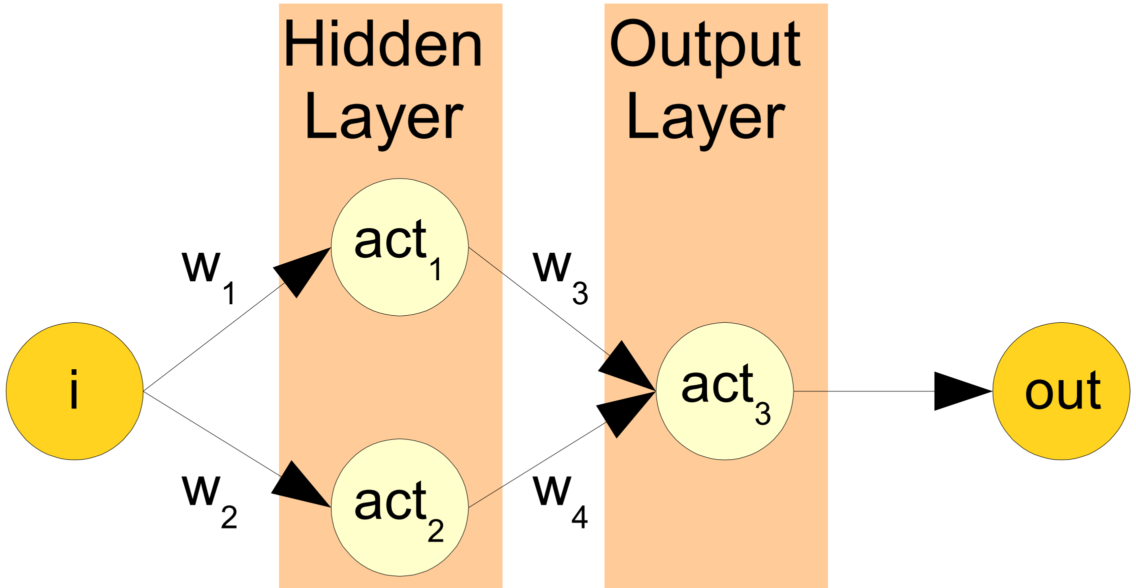




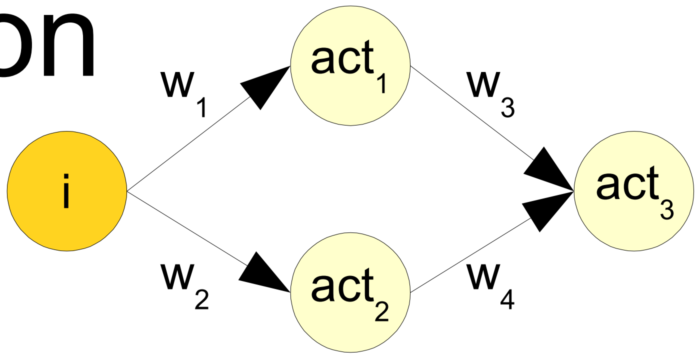
- Backpropagatoin is used to learn the weights
  - Is a gradient descent subType

Backpropagation

- Only for layered nets, a variant of gradient descent
- Learning requires supervised data
- Phases
- Compute output for a sample
- Compute the error
- Propagate the error backwards and adjust weights



Backpropagation



- What we know
  - Input: sample (attribute,class)
  - Error $e = \frac{1}{2}(class - net_3)^2$ Activation function is identity! $act_i = net_i$
  - Activity $net_1 = w_1 \cdot i$

    $net_2 = w_2 \cdot i$

    $net_3 = w_3 \cdot net_1 + w_4 \cdot net_2$

- Compute the gradient $\frac{d\ e}{d\ w_i}$ using the chain rule

for output layer

$$\frac{d\ e}{d\ w_3} = (class - net_3) \cdot (-net_1)$$

for hidden layer

$$\frac{d\ e}{d\ w_1} = (class - net_3) \cdot (-w_3) \cdot i$$