Evolutionary Algorithms

Alexander Asteroth, Adam Gaier, Alexander Hagg

Bonn Rhein Sieg University o.a.s., Deparment of Computer Science

Example Applications

- Behavioral patterns of a neurocontroller in robotics
- Algorithmic trader for financial markets
- Image analysis and computer vision: clustering, feature selection, categorization
- Data compression: compression rate vs reconstruction accuracy
- And many more

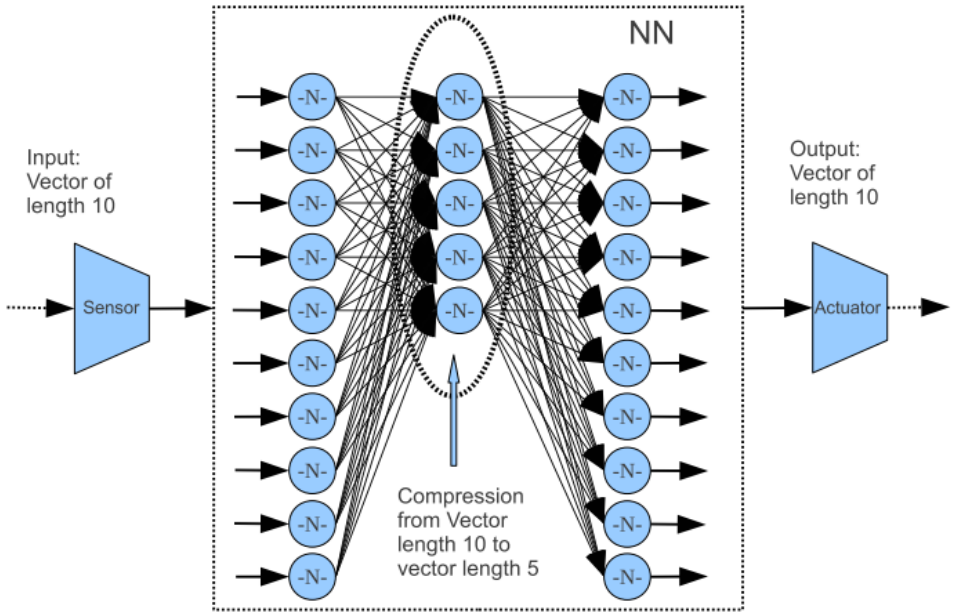
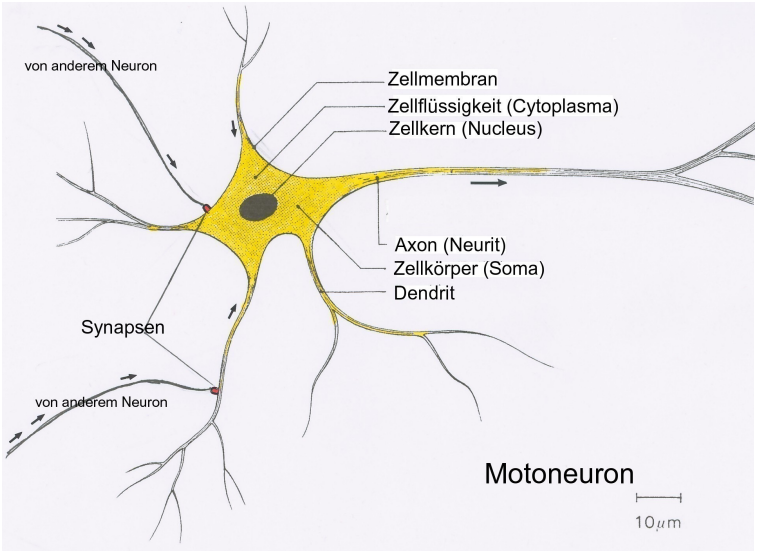
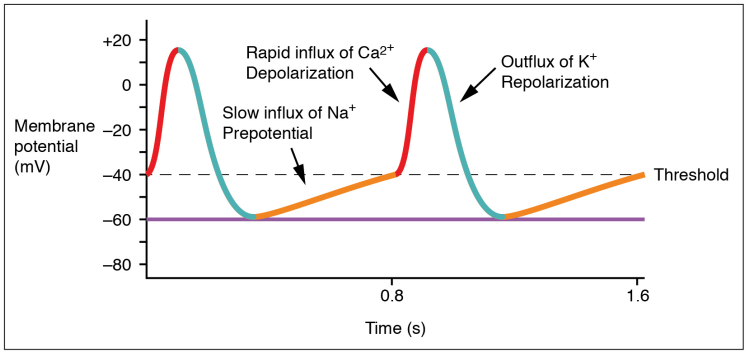
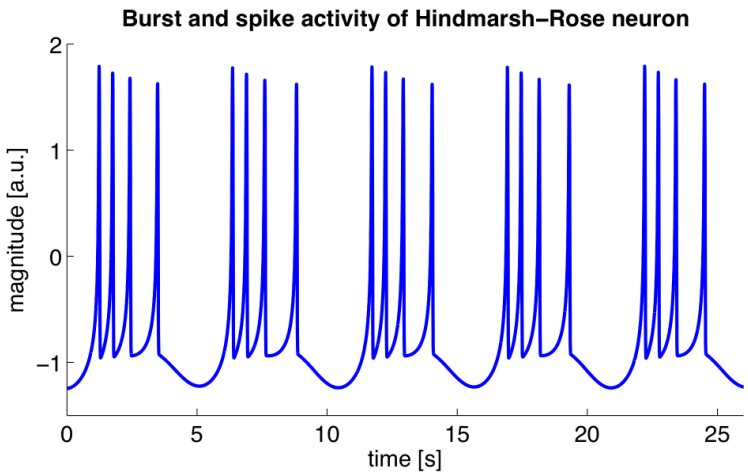


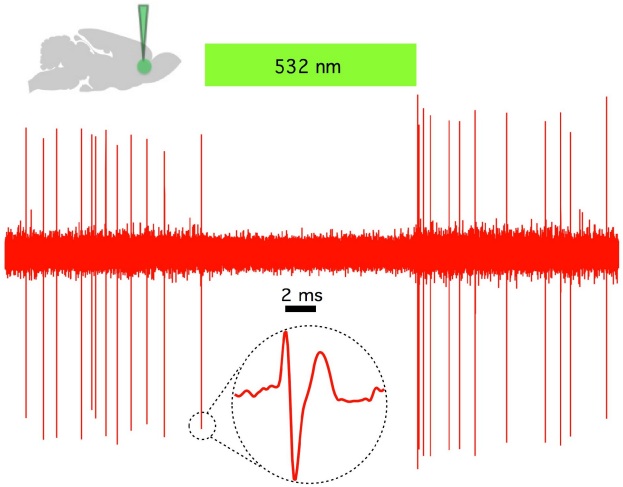
Figure: Compression using a NN

Biological neural networks

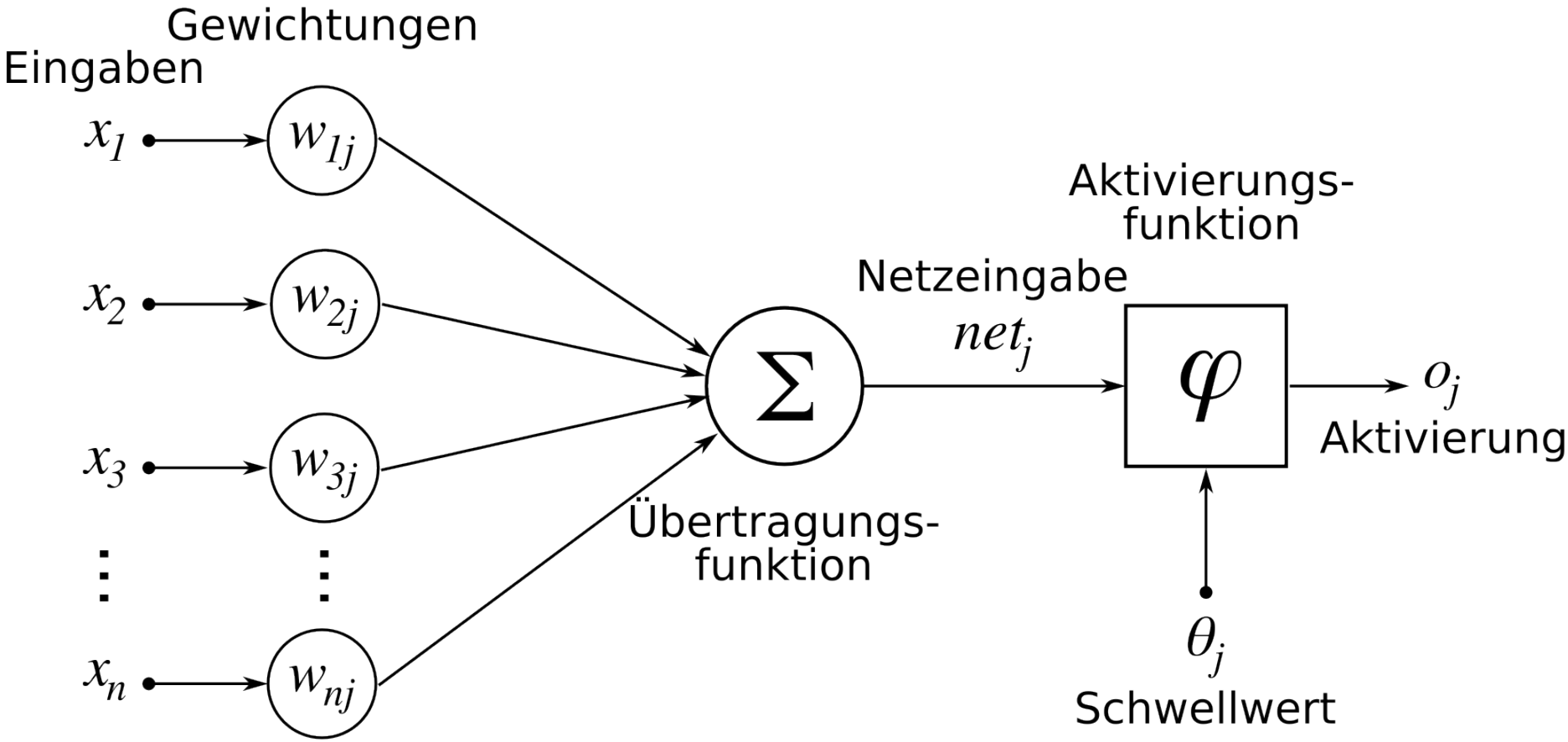








Model of a biological neuron



Artificial Neural Networks (ANN)

Definition (ANN Syntax)

An Artificial Neural Network N is a directed weighted annotated Graph $N = ((V, E), g)$. Nodes $v \in V$ are annotated with their activation function

$$g_v : \mathbb{R} \to \mathbb{R}$$

. Two subsets $I, O \subseteq V$ are marked as input and output nodes.

Artificial Neural Networks (ANN)

Definition (ANN Semantics)

Each node v_i in the Network $N = ((V, E), g, I, O)$ are associated two values: net_i and o_i.
For

$$v_i \in I : net_i = x_i$$

where x_i is the i-th input to the network
For $v_i \in V \setminus I$

$$net_i = \sum_{j \in Pred(i)} w_{ij} o_j$$

where w_{ij} is the weight of the edge that connects node v_j and node v_i.

$$o_i = g_{v_i}(net_i)$$

Remark

This Semantics might not be sufficient for recurrent networks. We will deal with this later.

ANN Implementation

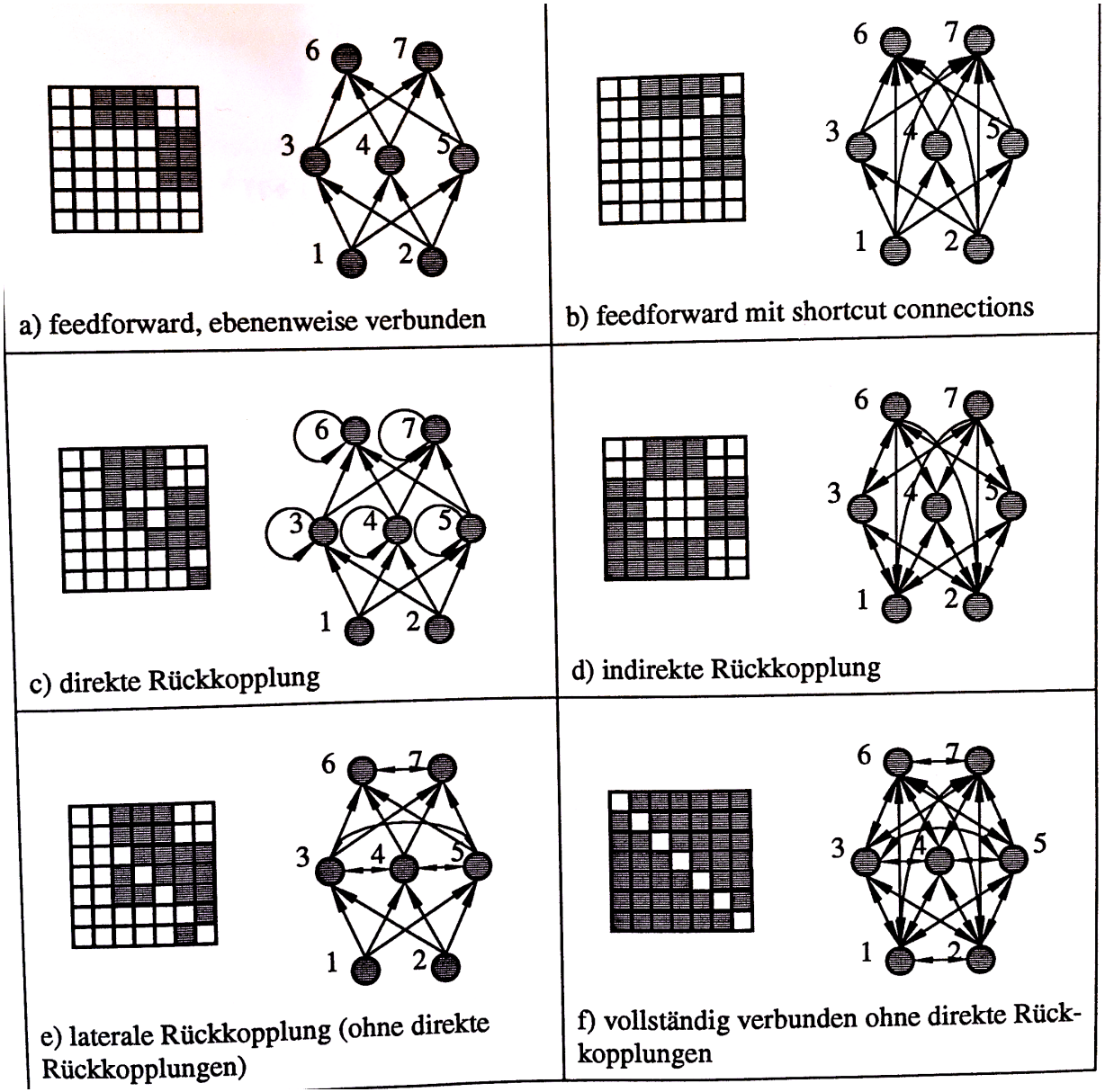
The weighted connections of N will be represented by a matrix

$$\mathbf{W} = (w_{ij})_{i,j=1,\ldots,n}$$

This way the vector of netinputs $\mathbf{net} = (net_i)_{i=1,\ldots,n}$ can be determined by multiplying the vector of activations $\mathbf{o} = (o_i)_{i=1,\ldots,n}$ with W:

$$\mathbf{net} = \mathbf{oW}$$

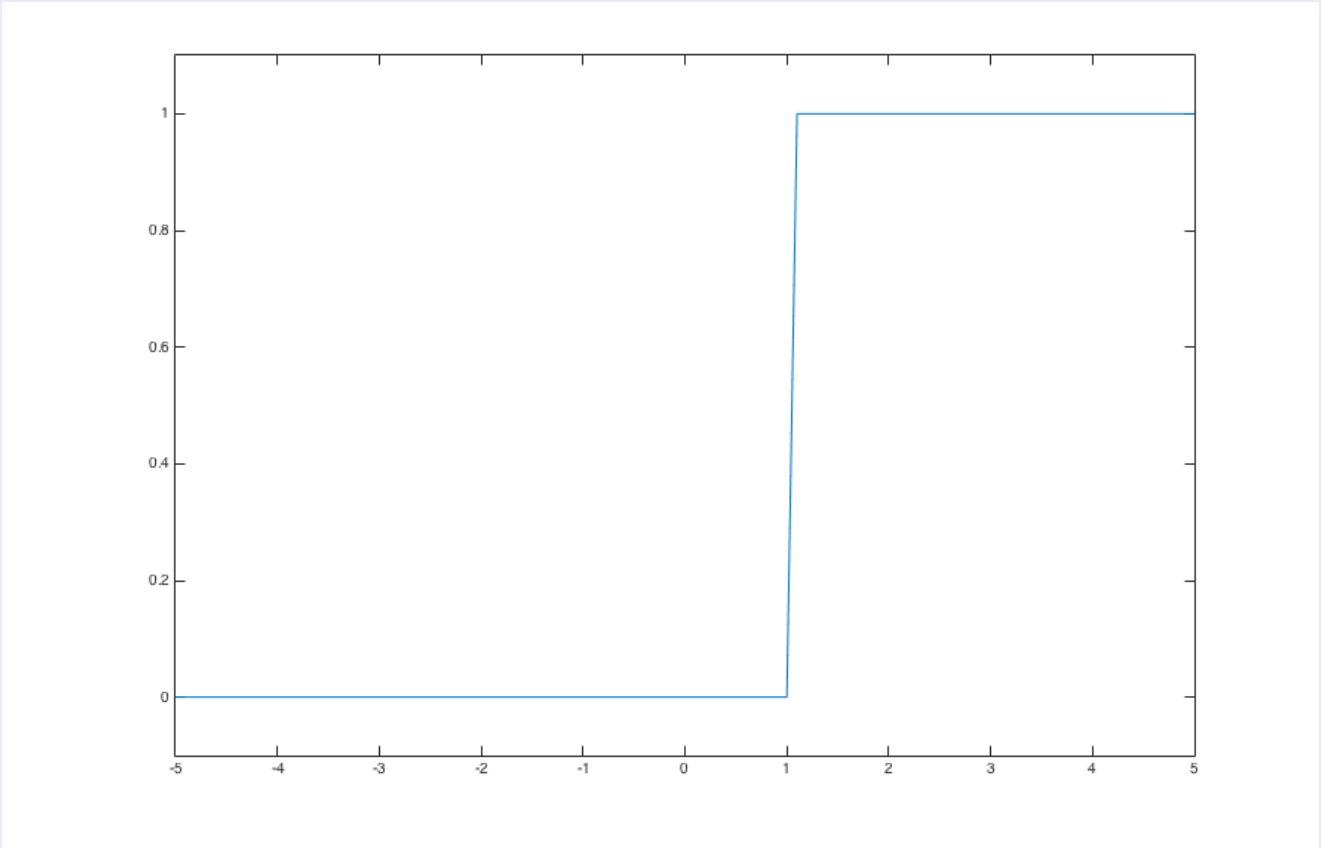
ANN Implementation (topologies)



ANN Implementation (activation functions)

Binary threshold function

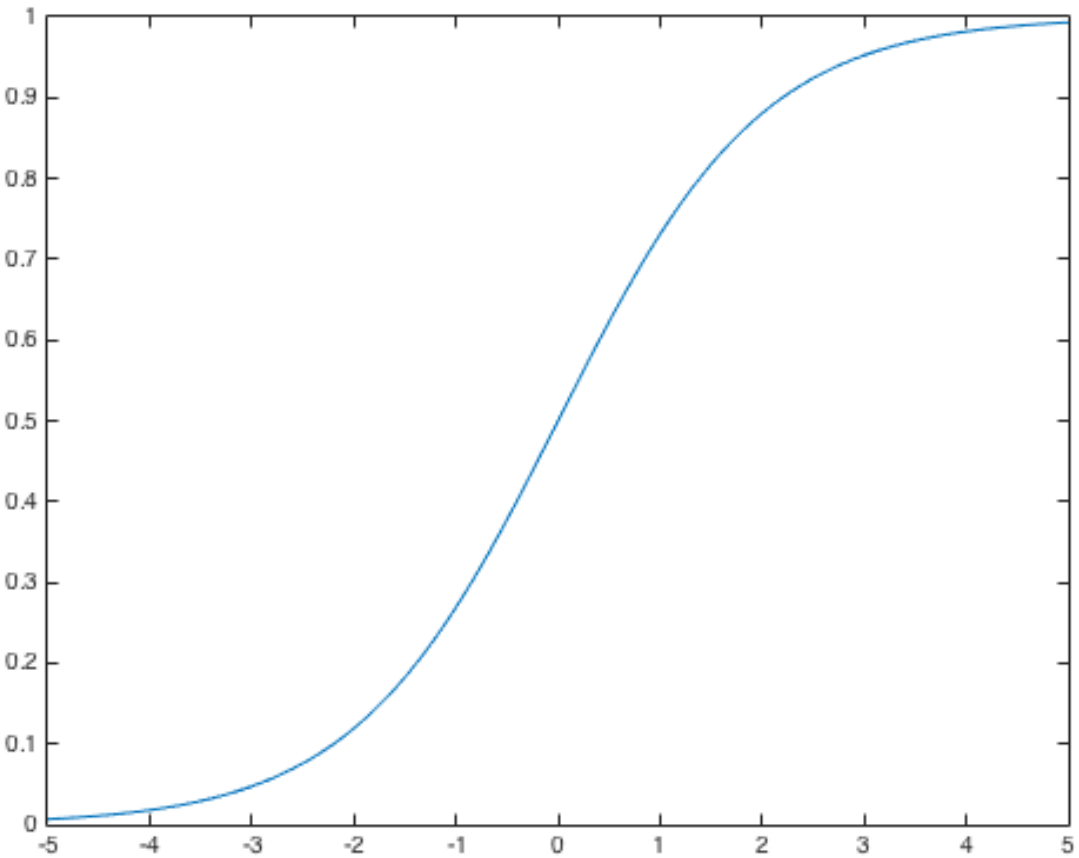
$$g(x) = \begin{cases} 0 & \text{if } x < \text{threshold} \\ 1 & \text{if } x \geq \text{threshold} \end{cases}$$



ANN Implementation (activation functions)

Logistic function

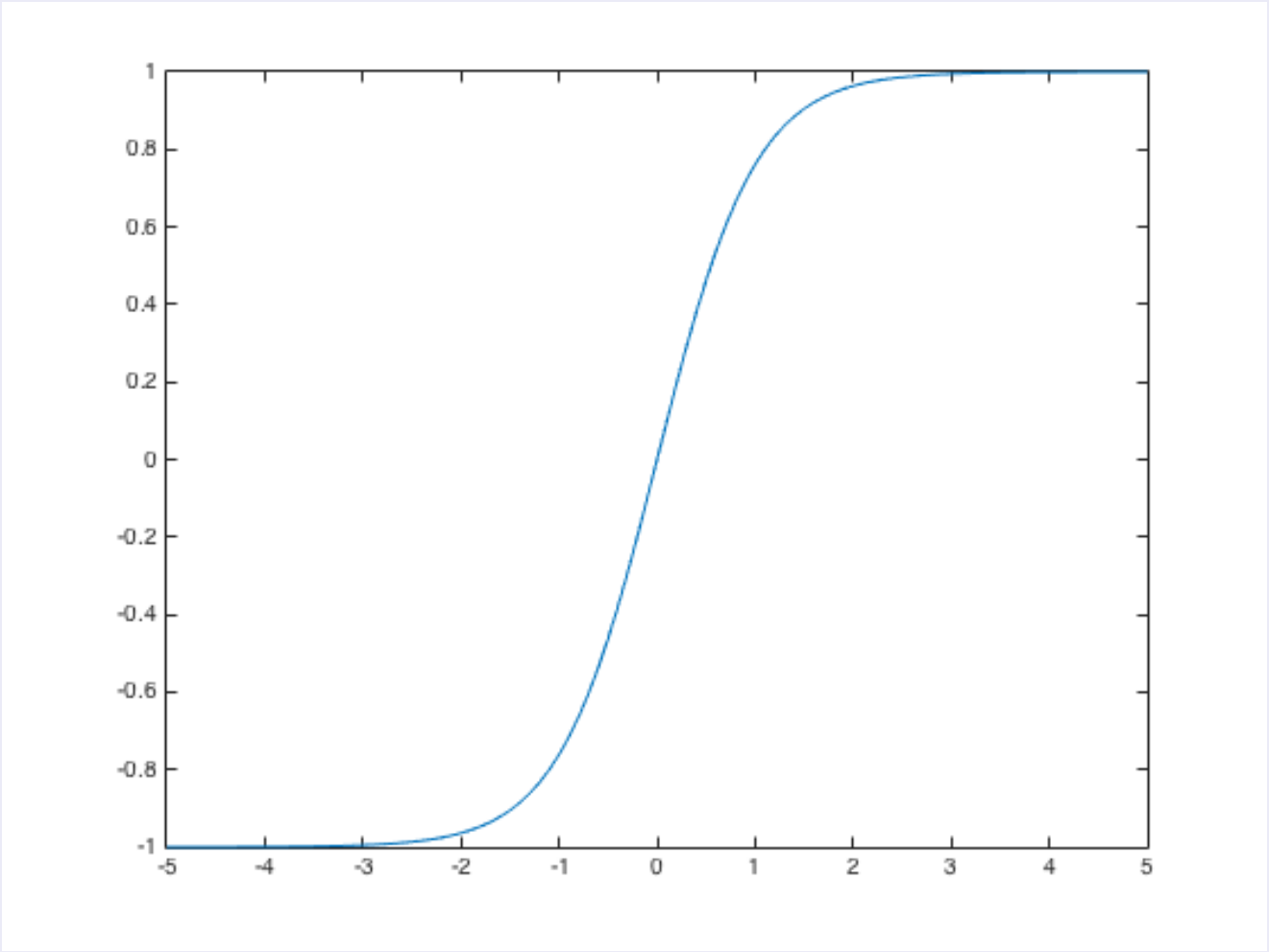
$$g(x) = \frac{1}{1 + e^{-x}}$$



ANN Implementation (activation functions)

Hyperbolic tangent

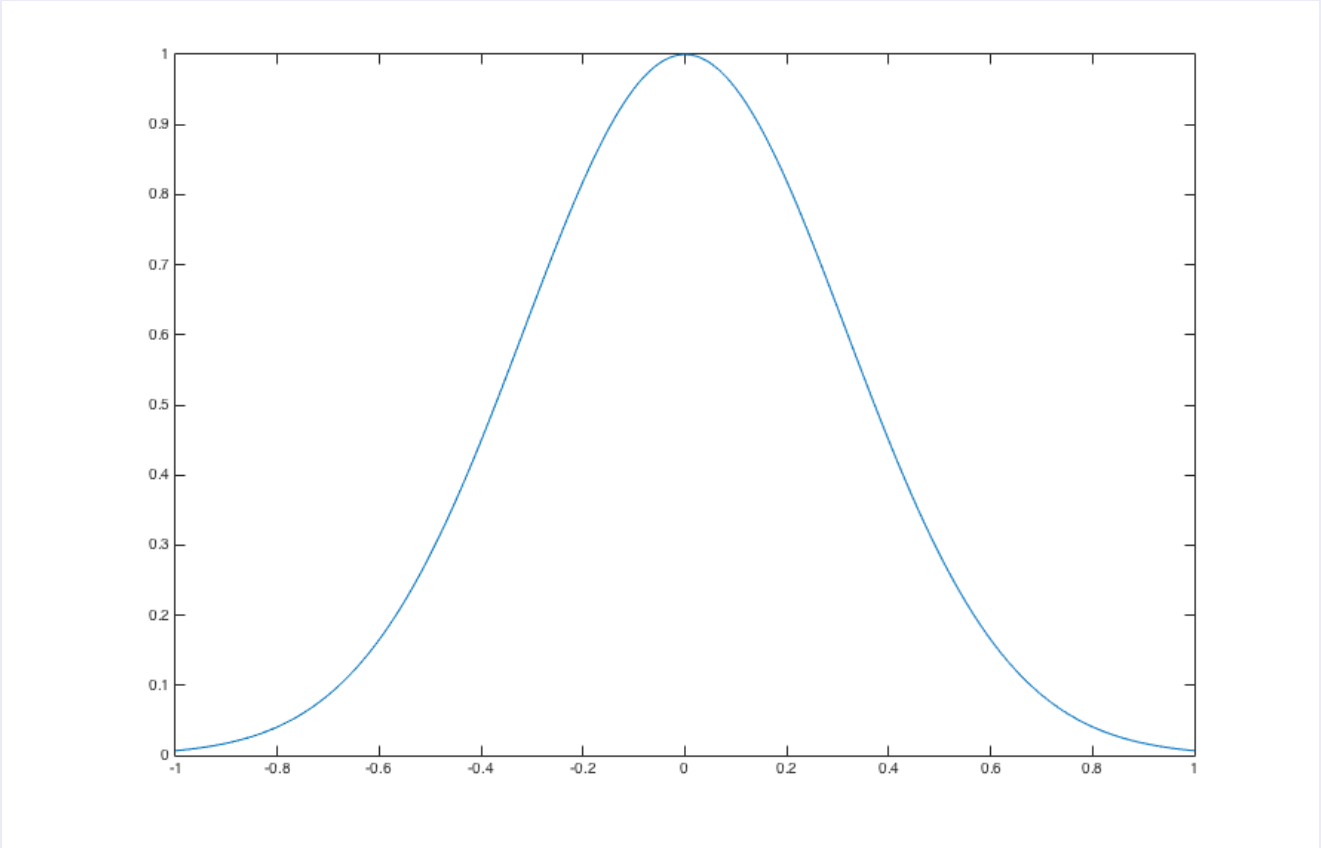
$$g(x) = \tanh(x)$$



ANN Implementation (activation functions)

Gaussian

$$g(x) = e^{-(x-\mu)/\sigma^2}$$



Feedforward networks (Perceptrons)

- Rosenblatt developed in the 1960's a family of ANNs that he called perceptrons.
- Perceptrons are strictly feed forward
- Learning was not based on least squares
- Can learn only "linearly separable" functions

Feedforward networks (single layer perceptron)

- most simple network model
- input layer connected with output layer
- binary threshold activation function

ANN Learning (perceptron learning rule)

problem statement

Given a vector of inputs $\mathbf{x}^{(t)} \in \mathbb{R}^n$ and a corresponding vector of outputs $\mathbf{y}^{(t)} \in \{0,1\}^m$ find the weight matrix $\mathbf{W}$, that minimizes the error: $\sum \|\mathbf{o}^{(t)} - \mathbf{y}^{(t)}\|$

idea

Increase netinput if output is too small, decrease if too high.

Learning rule:

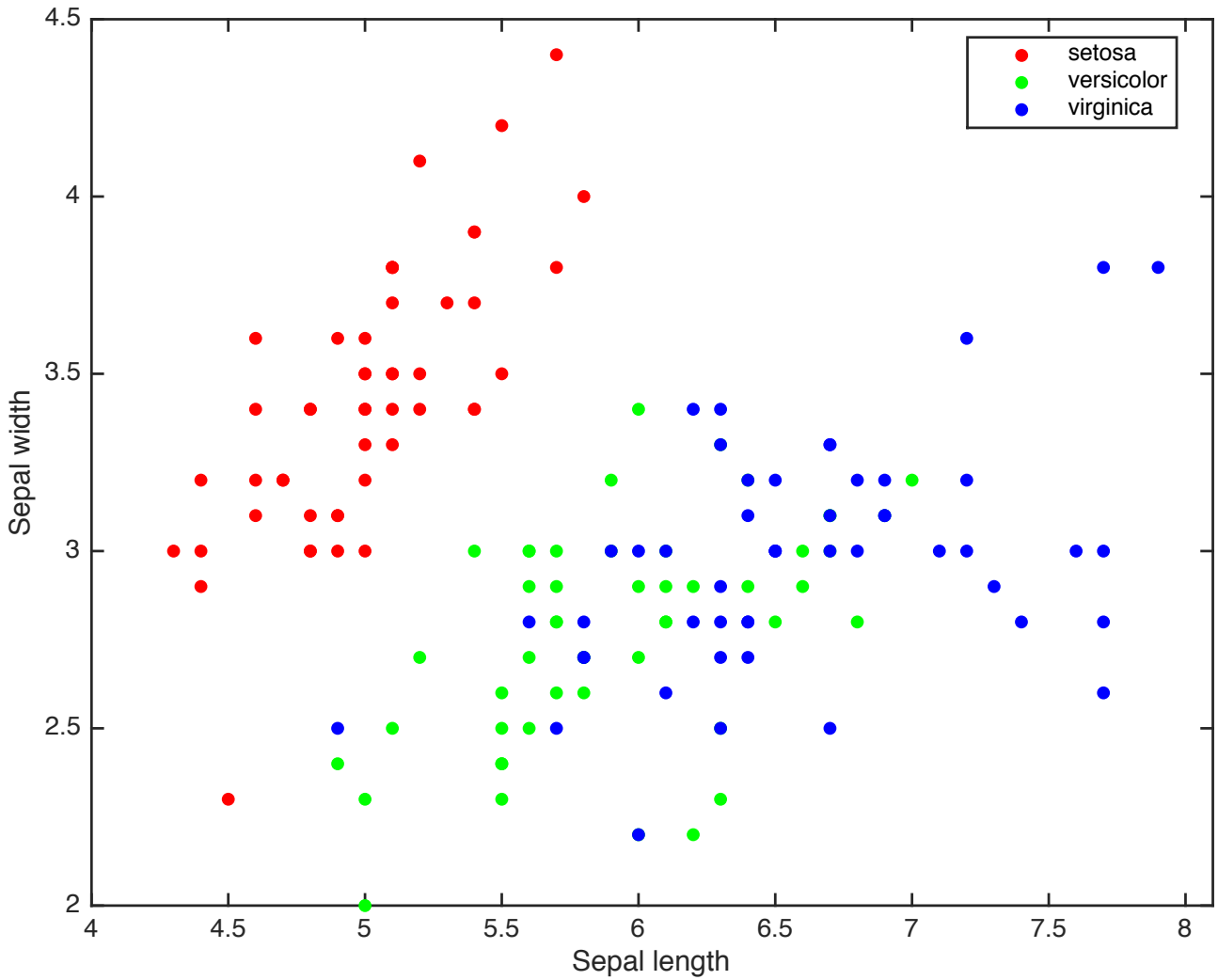
$$w_{ij} = w_{ij} + \eta(y_i^{(t)} - o_i^{(t)}) \cdot x_j^{(t)}$$

matrix notation

$$\mathbf{W} = \mathbf{W} + \eta(\mathbf{y}^{(t)} - \mathbf{o}^{(t)}) \cdot (\mathbf{x}^{(t)})^T$$

Example: Fisher's Iris dataset

Example: Fisher's Iris dataset



(see e.g. `https://en.wikipedia.org/wiki/Iris_flower_data_set`)

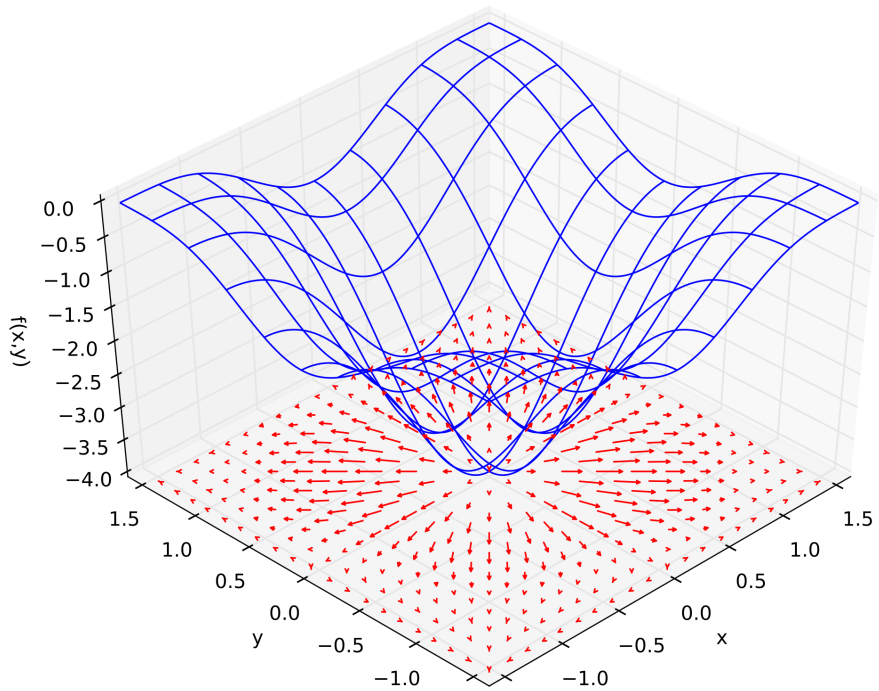
Gradient based optimization

Only on demand ▸ Skip

Gradient based optimization

Idea:

- gradient ∇f points direction of steepest ascent
- step in direction $-\nabla f$ should lower function value
- series of steps will lead to local minimum



Math

Notations

For a function $f : \mathbb{R}^n \to \mathbb{R}$ the derivative of f w.r.t. the variable x_i is denoted

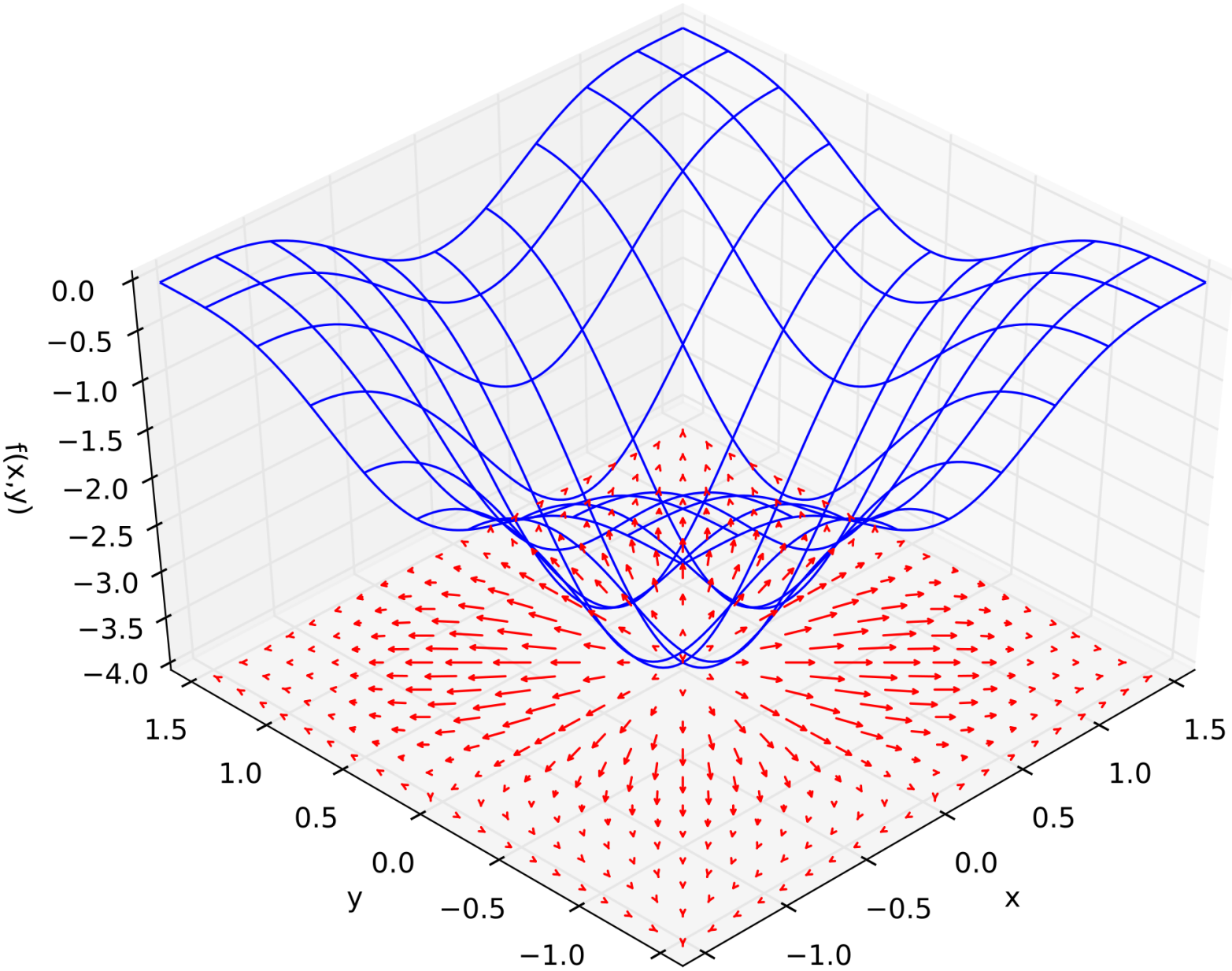
$$\frac{\partial f}{\partial x_i}$$

The vector of all partial derivatives is called the gradient

$$\nabla f = \left(\frac{\partial f}{\partial x_i}, \ldots, \frac{\partial f}{\partial x_n} \right)^T$$

While $\frac{\partial f}{\partial x_i}$ is the steepness of f in direction x_i, the vector ∇f points in the direction of the steepest ascent. it's length represents the steepness of the ascent.

2D gradient (revisited)



Math

Chain rules

From school (Analysis) you know that

$$f(g(x))' = f'(g(x)) \cdot g'(x), \qquad \left(\frac{\partial f}{\partial x} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x}\right)$$

If $f : \mathbb{R} \to \mathbb{R}, g : \mathbb{R}^n \to \mathbb{R}$, and $f = f(g(x_1, \ldots, x_n))$ then

$$\frac{\partial f}{\partial x_i} = \frac{\partial f}{\partial g}\frac{\partial g}{\partial x_i}$$

If $f : \mathbb{R} \to \mathbb{R}^n, g_i : \mathbb{R} \to \mathbb{R}, i = 1, \ldots, n$, and $f = f(g_1(x), \ldots, g_n(x))$ then

$$\frac{\partial f}{\partial x} = \sum_{i=1}^{n} \frac{\partial f}{\partial g_i}\frac{\partial g_i}{\partial x}$$

derivatives of sigmoidal activation functions

logistic

$$g(x) = \frac{1}{1 + e^{-x}}, \qquad g'(x) = g(x)(1 - g(x))$$

hyperbolic tangent

$$g(x) = tanh(x), \qquad g'(x) = 1 - g(x)^2$$

derivatives of netinput function

$$net_j = \sum_k w_{kj} o_k$$

therefore

$$\frac{\partial net_j}{\partial o_i} = \frac{\partial}{\partial o_i} \sum_k w_{kj} o_k = \sum_k \frac{\partial}{\partial o_i} w_{kj} o_k = w_{ij}$$

Error minimization in Feedforward networks

Let $y^{(p)}$ denote the target output vector p and $o^{(p)}$ the output of the network for input $x^{(p)}$. Then the squared error of the network on pattern p is defined as:

$$E^{(p)} = \frac{1}{2}\left(y^{(p)} - o^{(p)}\right)^2$$

The sum squared error over all pattern p is:

$$E = \sum_p E^{(p)}$$

The goal is to minimize E by making small steps in the direction of $-\nabla E$. in *online learning* steps will be made in direction $-\nabla E^{(p)}$ for each pattern presented for learning.

gradient of $E^{(p)}$

Since the gradient $\nabla E^{(p)} = \left(\frac{\partial E^{(p)}}{\partial w_{ij}}\right)$ cannot be determined directly, chain rules have to be used:

$$\frac{\partial E^{(p)}}{\partial w_{ij}} = \frac{\partial E^{(p)}}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}} = \frac{\partial E^{(p)}}{\partial net_j} o_i$$

The term $\frac{\partial E^{(p)}}{\partial net_j}$ plays a central role and therefore is given it's own name:

$$\delta_j^{(p)} = \frac{\partial E^{(p)}}{\partial net_j}$$

or δ_j for short (if p is known from context the superscript (p) will be ommited).

Remark

Online Learning

We will describe online learning only to avoid index confusion.
For non-online version every gradient has to be summed up over all pattern before update.

First steps

To furter avoid indices we will at first only consider two consecutive layers and generalize the results later on.
We also will only consider logistic activations.

negative error gradient

output layer

For any output neuron i:

$$-\frac{\partial E}{\partial o_i} = -\sum_{j \in Outputlayer} \frac{\partial}{\partial o_i} \frac{1}{2}(y_j - o_j)^2 = (y_i - o_i)$$

$$-\frac{\partial E}{\partial net_i} = -\frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial net_i} = (y_i - o_i)g'(net_i) = (y_i - o_i)o_i(1 - o_i)$$

vector notation:

$$\left(-\frac{\partial E}{\partial o_i}\right) = \mathbf{y} - \mathbf{o}, \qquad \boldsymbol{\delta} = (\mathbf{y} - \mathbf{o}) \circ \mathbf{o} \circ (\mathbf{1} - \mathbf{o})$$

where $\boldsymbol{\delta}$ denotes the vector of all $\delta_i = -\frac{\partial E}{\partial net_i}$ and $\circ$ denotes elementwise multiplication.

negative error gradient

intermediate layer

$$-\frac{\partial E}{\partial o_i} = \sum_{j \in Post(i)} -\frac{\partial E}{\partial net_j}\frac{\partial net_j}{\partial o_i} = \sum_{j \in Post(i)} \delta_j w_{ij}$$

$$-\frac{\partial E}{\partial net_i} = -\frac{\partial E}{\partial o_i}\frac{\partial o_i}{\partial net_i} = -\frac{\partial E}{\partial o_i} g'(net_i) = \left(\sum_{j \in Post(i)} \delta_j w_{ij}\right) o_i(1 - o_i)$$

vector notation:

$$\left(-\frac{\partial E}{\partial o_i}\right) = \boldsymbol{\delta} \cdot \mathbf{W}^T$$

where $\boldsymbol{\delta}$ is the vector of δ_i from the next layer. For the new $\boldsymbol{\delta}'$ of the current layer we get:

$$\boldsymbol{\delta}' = \left(-\frac{\partial E}{\partial net_i}\right) = \boldsymbol{\delta} \cdot \mathbf{W}^T \circ \mathbf{o} \circ (\mathbf{1} - \mathbf{o})$$

Nomenclature

- We consider a network with $k-1$ layers of weights and k layers of neurons. Layers of neurons will be labeled with indices $1, \ldots, k$. The number of neurons per layer will be denoted as n_k.
- For $i = 1, \ldots, n_k$ with $o_{k,i}$ we denote the activation of neuron i in layer k. For easy threshold handling we add on additional constant activation $o_{k,n_k+1} = 1$ to each layer.
- For $i = 1, \ldots, n_k$ with $net_{k,i}$ we denote the netinput of neuron i in layer k.
- $\mathbf{W}_i$ denote the adjacency matrix connecting layer i with layer $i+1$.
- $\mathbf{o}_i, \mathbf{net}_i$ denotes the activation/netinput rowvector of layer i
- In matrix notation we can comute:

$$\mathbf{net}_{i+1} = \mathbf{net}_i \mathbf{W}_i, \qquad \mathbf{o}_i = [g(\mathbf{net}_i) \quad 1]$$

error back-propagation

for all layers $1 \leq i < k$ compute (forward):

$$\mathbf{o}_{i+1} = g(\mathbf{o}_i \mathbf{W}_i)$$

For output layer k:

$$\boldsymbol{\delta}_k = (\mathbf{y} - \mathbf{o}_k) \circ \mathbf{o}_k \circ (\mathbf{1} - \mathbf{o}_k)$$

for inner layers $i < k$ compute (backwards):

$$\boldsymbol{\delta}_i = \boldsymbol{\delta}_{i+1} \mathbf{W}_i^T \circ \mathbf{o}_i \circ (\mathbf{1} - \mathbf{o}_i)$$

adjust the weights

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial net_j} \frac{\partial net_j}{\partial w_{ij}} = \delta_j o_i$$

in vector notation:

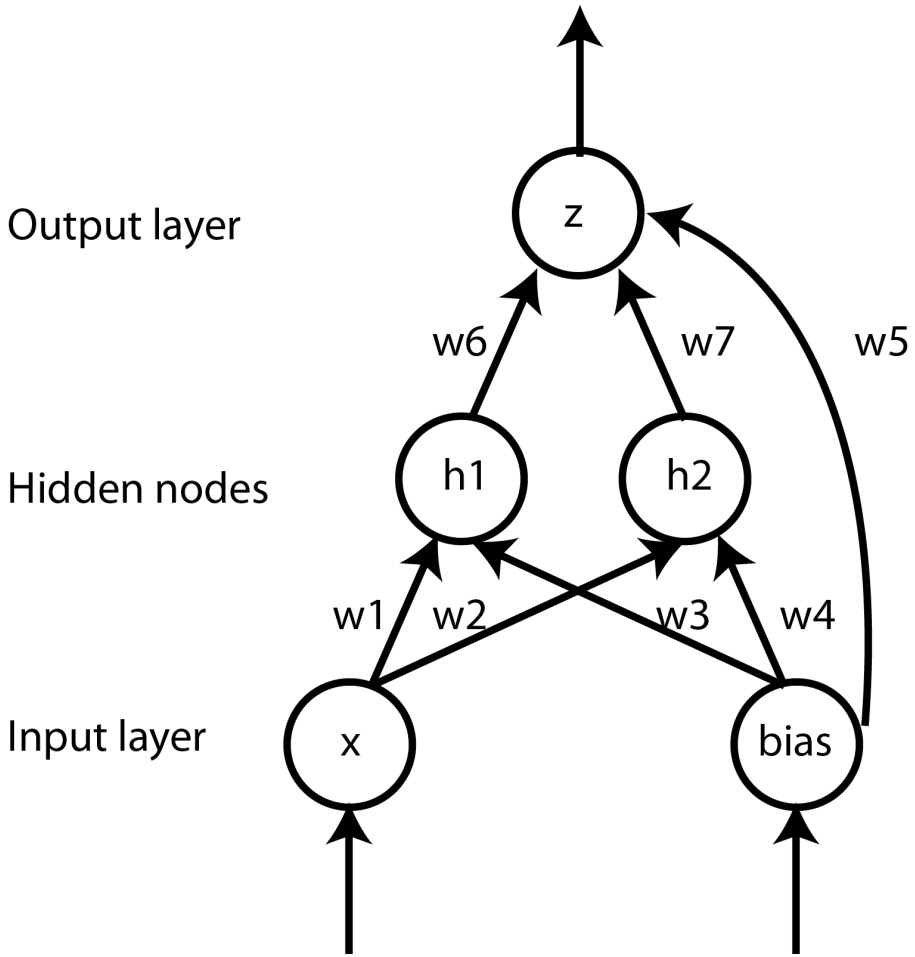
$$\Delta \mathbf{W}_i = \mathbf{o}_i^T \boldsymbol{\delta}_{i+1}$$

Exercise

- Implement back-propagation algorithm
- Solve iris classification task using different network topologies
- Compare error-(learning)-curves and performance of different topologies

Evolving Neural Networks with Fixed Topologies

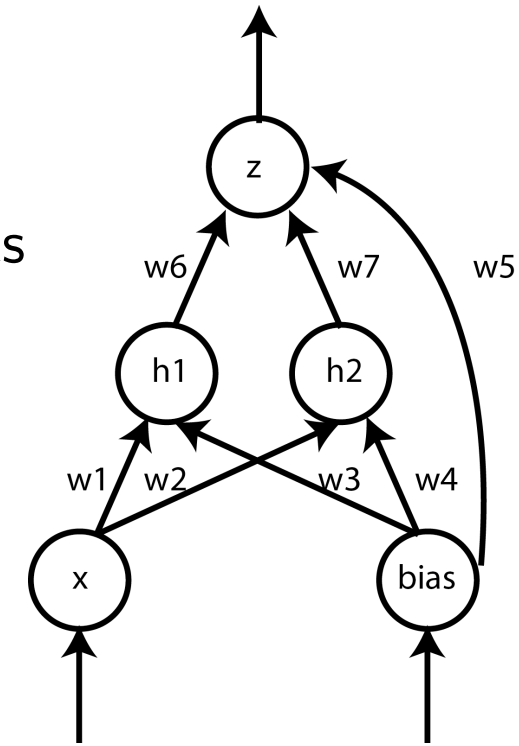
Let us start simple.



Evolving Neural Networks with Fixed Topologies

Fixed Topology Feed Forward Neural Network

- Define network topology
- Genotype: matrix of real-valued weights
- Feed Forward Network with 2 hidden nodes:
  genome length $= 7$
  ((1 input+bias) $*$ 2 hidden
  $+$ (2 hidden+bias) $*$ 1 output)



Evolving Neural Networks with Fixed Topologies

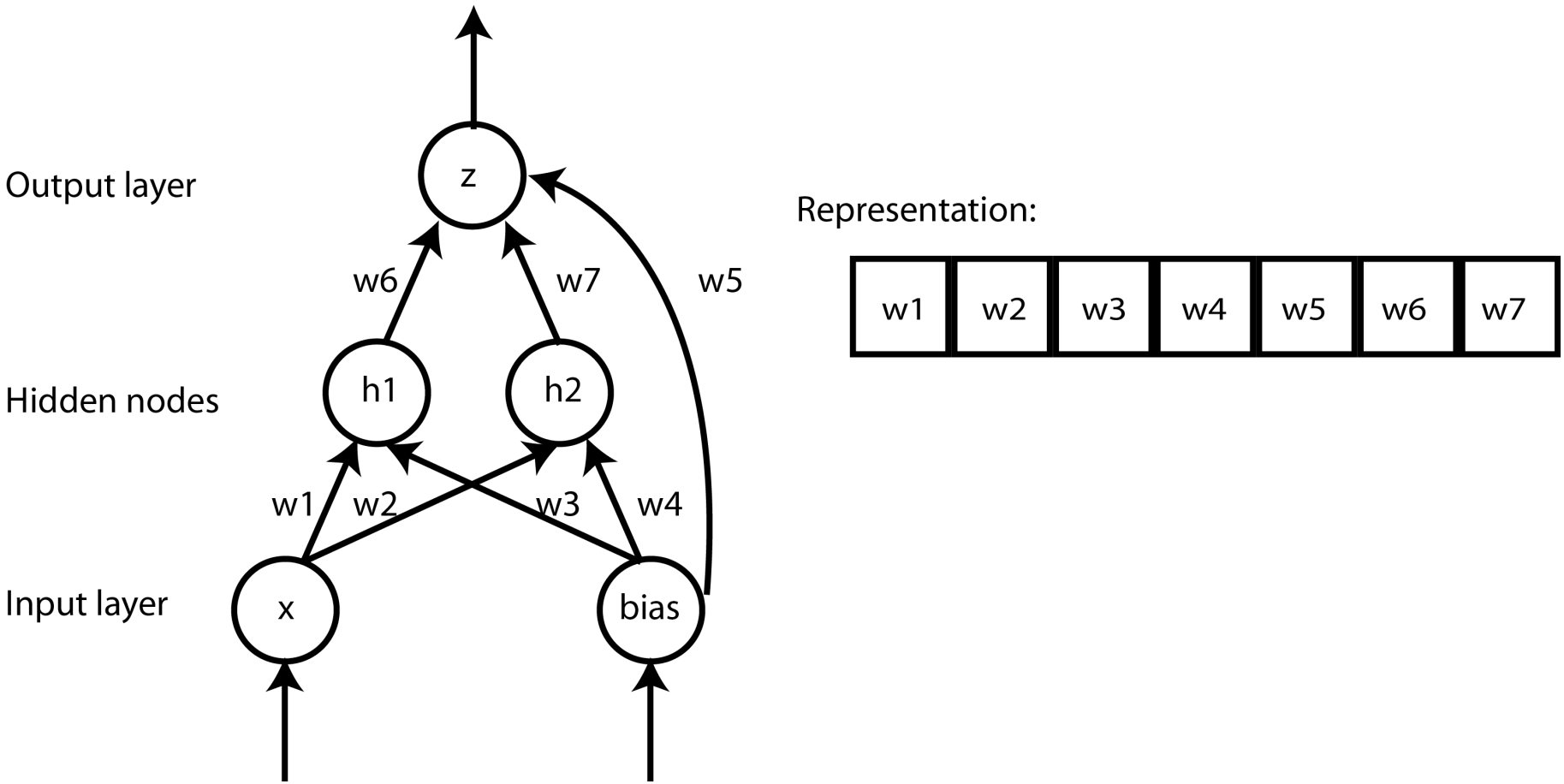


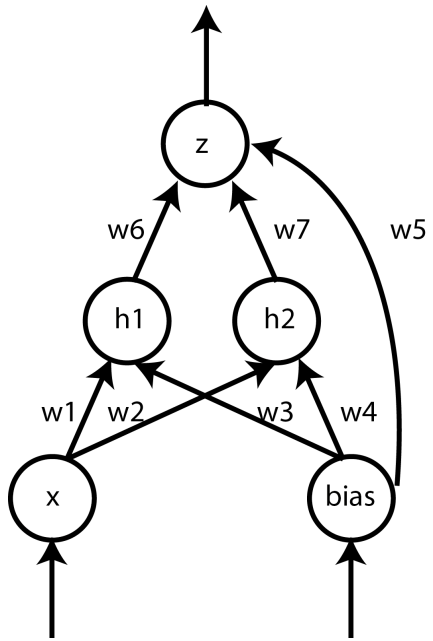
Figure: Representation of fixed-topology FFNN

Evolving Neural Networks with Fixed Topologies

- Each weight is a gene, the entire network is an individual
- Evaluate FFNN output against training sequence
- Fitness $\rightarrow$ MSE, SSE, **smoothness of path** a robot takes, **user feedback**, etc.
- Let us talk about implementing this a bit more.

Evolving Neural Networks with Fixed Topologies

	h_1	h_2	z
x	w_1	w_2	0
bias	w_2	w_4	w_5
h_1	0	0	w_6
h_2	0	0	w_7
z	0	0	0



Evolving Neural Networks with Fixed Topologies

$$(x_1, 1, 0, 0, 0) \cdot \begin{pmatrix} w_1 & w_2 & 0 \\ w_2 & w_4 & w_5 \\ 0 & 0 & w_6 \\ 0 & 0 & w_7 \\ 0 & 0 & 0 \end{pmatrix} = (w_1 x_1 + w_2, w_2 x + w_4, w_5)$$

- Weight matrix W, Activation matrix A, Resulting matrix R
- $(3 \times 5) * (5 \times 1) = (1 \times 3)$
- $(3 \times 5) * (5 \times N) = (N \times 3)$

Evolving Neural Networks with Fixed Topologies

Naive weight evolution:

- Representation: string of real numbers
- Crossover: use standard GA methods
- Mutation: use standard GA methods

Evolving Neural Networks with Fixed Topologies

Determine fitness:

- Instantiate all inputs and accompanying outputs
- Feed inputs to FFNN and save outcome
- Error based on outcome and (expected) outputs.

Evolving Neural Networks with Fixed Topologies

Recurrent Neural Networks (RNN)

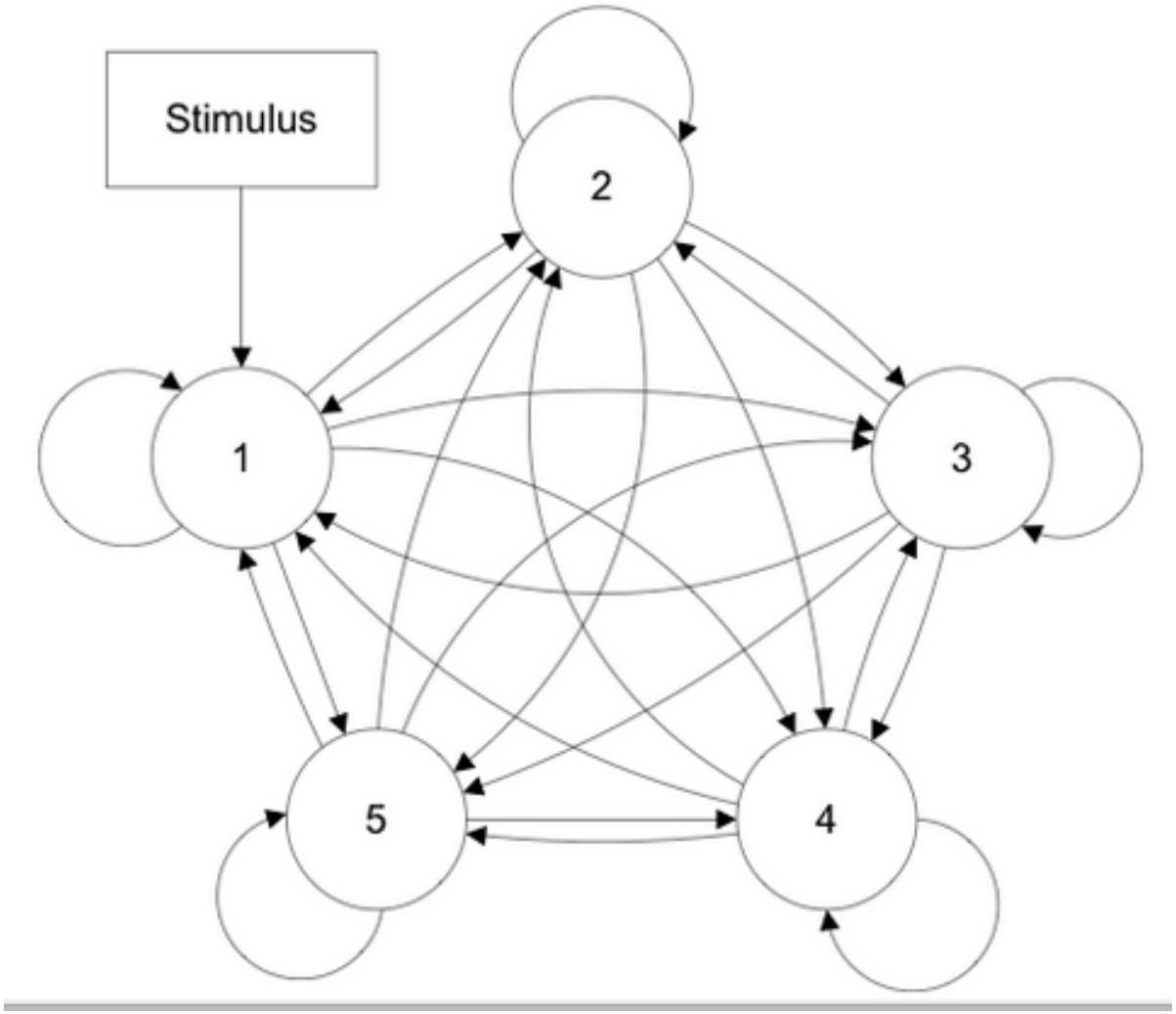
- Directed *cyclic* graph
- Nodes connected to each other and themselves
- Activation computed every timestep, using activations from input and previous state
- Internal memory (in the form of previous state activations) allows processing of sequences of inputs
- Allows for dynamic *temporal* behavior

Evolving Neural Networks with Fixed Topologies

Recurrent Neural Networks (RNN)

- Best known results for tasks such as speech recognition, handwriting recognition
- Training difficult with standard neural network training methods like back propagation
- Hard to learn correlation between temporally distant events
- Genetic algorithms the most common global optimization method for training RNNs

Recurrent Neural Network



ESP – enforced sub-populations

- Introduced in 1996 by Gomez and Miikkulainen (U-Texas, Austin)
- Based on SANE by Moriarty and Miikkulainen (Symbiotic Adaptive Neuro-Evolution)
- GA strategy suited for training recurrent neural networks
- Originally presented in combination with delta-coding

ESP

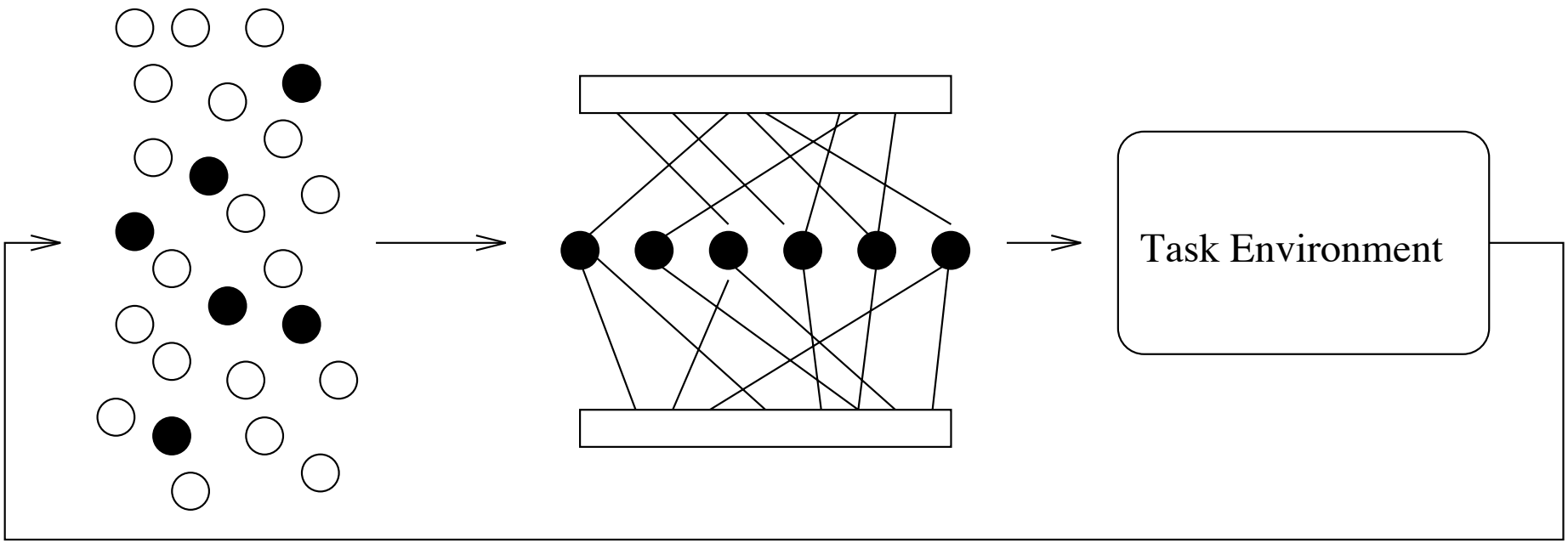
Idea

- Individuals represent nodes rather than networks (SANE)
- One sub-population per node
- Ranking, recombination, elitism on sub-population (node) level

Advantages

- Species of nodes (evolving in SANE) circumscribed by approach
- Evolution of individuals (nodes) takes place in context (place in the network)
- Particularly suited for recurrent networks

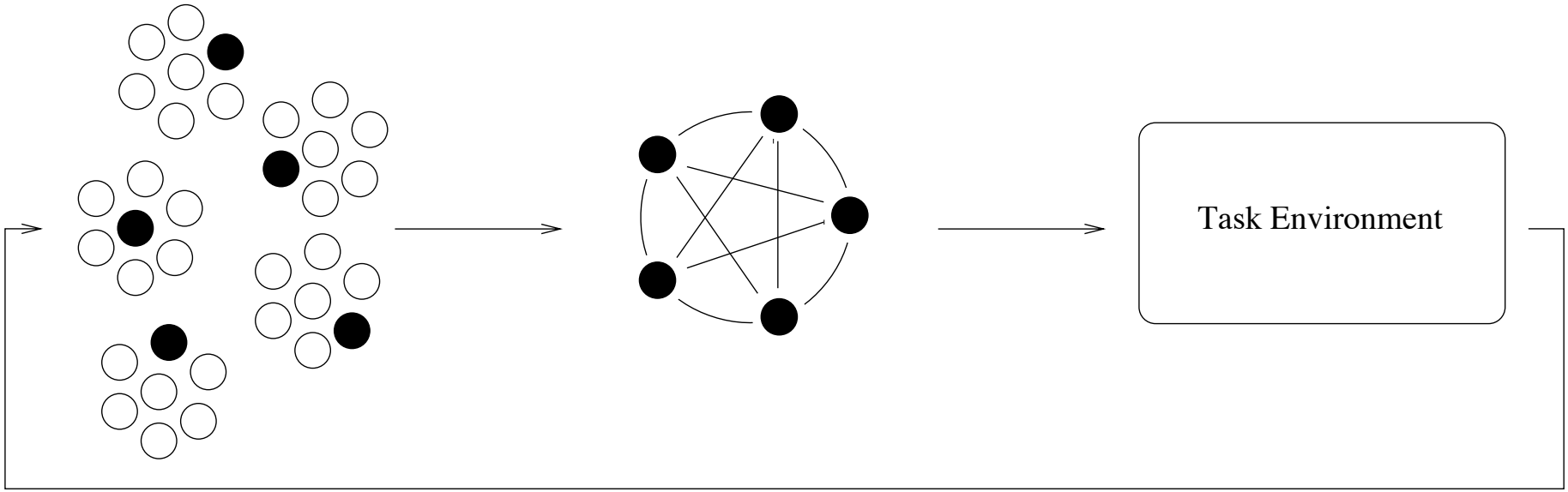
SANE – Symbiotic, Adaptive, Neuro-Evolution



- Population of nodes
- group of n nodes formes feedforward network
- evaluation is based on average fitness of the networks the individual is part of

(figure taken from: Gomez, Miikkulainen: *Incremental Evolution Of Complex General Behavior*, Adaptive Behavior(5):317-342, 1997)

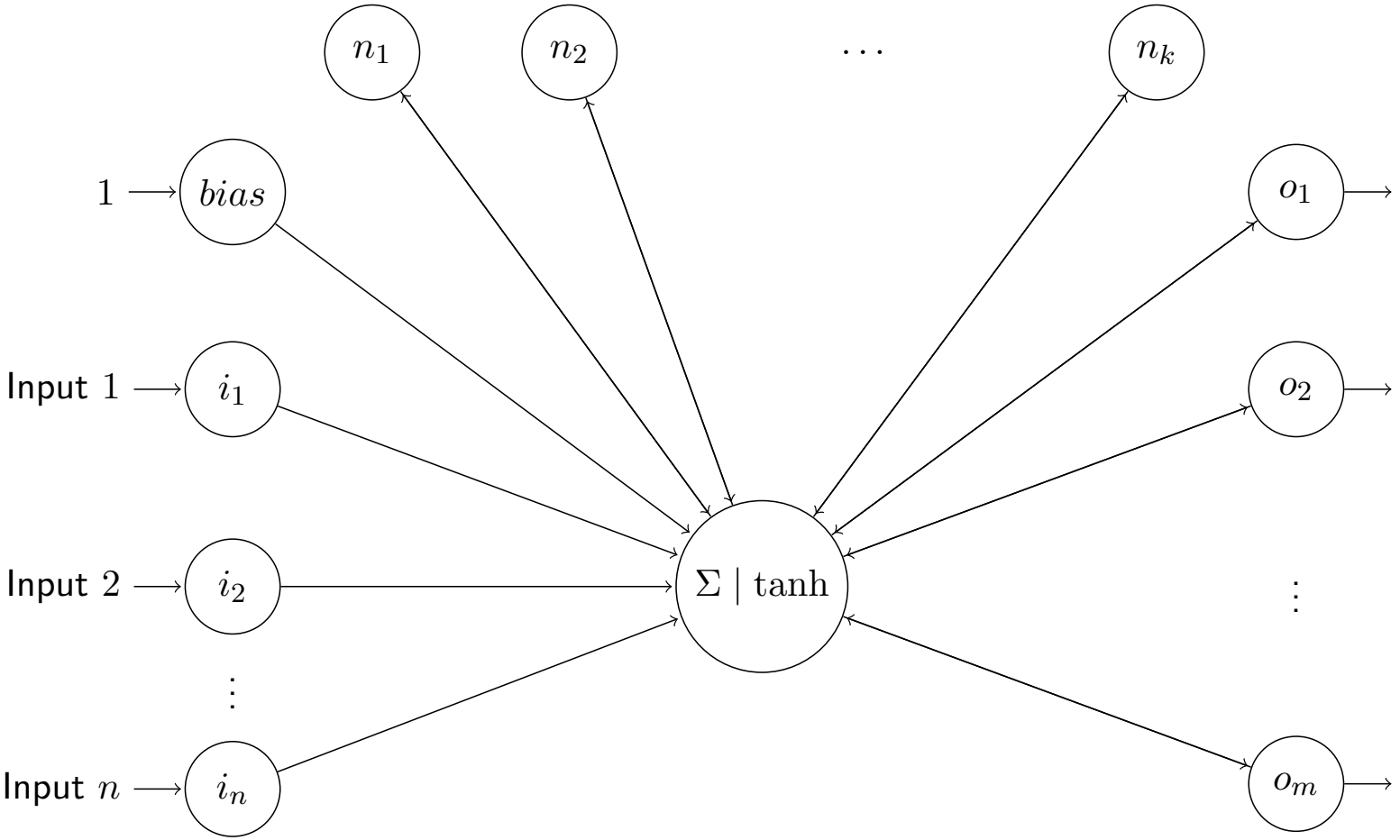
ESP – Enforced Sub-Populations



- One sub-population per node
- one node from n sub-populations formes recurrent network

(figure taken from: Gomez, Miikkulainen: *Incremental Evolution Of Complex General Behavior*, Adaptive Behavior(5):317-342, 1997)

ESP – node structure



- Input from $i_1, \ldots, i_n$, I/O from $n_1, \ldots, n_k, o_i, \ldots, o_m$, Output to $o_i, \ldots, n_m$

ESP – representation of nodes

- Each node has connection from input node but input nodes
- Output nodes are regular (as hidden) nodes

If we structure our activation vector as follows:

$$\mathbf{act} = (n_1, \ldots, n_k, o_1, \ldots, o_m), \quad \mathbf{in} = (1, i_1, \ldots, i_n)$$

The connection weight matrix has the following structure:

$$\mathbf{W} = \begin{pmatrix} \mathbf{W}_{i_i n_j} & \mathbf{W}_{i_i o_j} \\ \mathbf{W}_{n_i n_j} & \mathbf{W}_{n_i o_j} \\ \mathbf{W}_{o_i n_j} & \mathbf{W}_{o_i o_j} \end{pmatrix}$$

Actication can be calculated as:

$$\mathbf{act} = \tanh([\mathbf{in}\ \mathbf{act}] \cdot \mathbf{W})$$

ESP – recombination and reproduction

- recombination and reproduction within sub-populations
- based on fitness matrix `fitness(j,k)`
- We suggest
  - elitism
  - uniform crossover
  - gaussian mutation
  - tournament selection with selection pressure 2
- Anything else might work as well (even better)