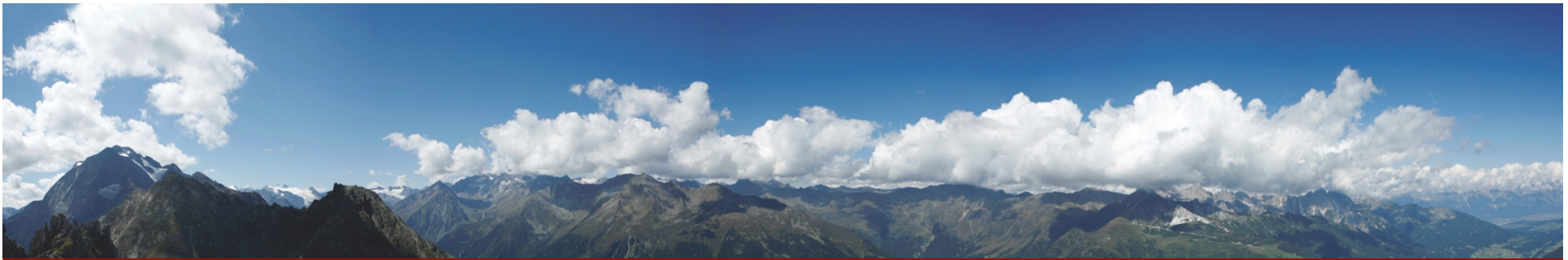



Intelligent Systems

Neural Networks



Where are we?

#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Theorem Proving, Description Logics and Logic Programming
5	Search Methods
6	CommonKADS
7	Problem Solving Methods
8	Planning
9	Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
 13	Neural Networks
14	Semantic Web and Exam Preparation

-
- Motivation
 - Technical Solution
 - (Artificial) Neural Networks
 - Neural Network Structures
 - Learning and Generalization
 - Expressiveness of Multi-Layer Perceptrons
 - Illustration by Larger Examples
 - Summary

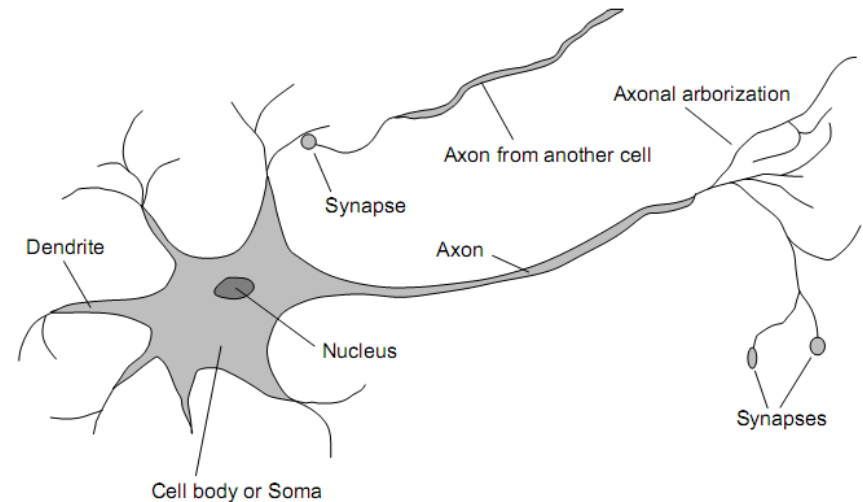
MOTIVATION

- A main motivation behind neural networks is the fact that symbolic rules do not reflect reasoning processes performed by humans.
- Biological neural systems can capture highly parallel computations based on representations that are distributed over many neurons.
- They learn and generalize from training data; no need for programming it all...
- They are very noise tolerant – better resistance than symbolic systems.
- In summary: neural networks can do whatever symbolic or logic systems can do, and more. In practice it is not that obvious however.

- Neural networks are strong in:
 - Learning from a set of examples
 - Optimizing solutions via constraints and cost functions
 - Classification: grouping elements in classes
 - Speech recognition, pattern matching
 - Non-parametric statistical analysis and regressions

TECHNICAL SOLUTIONS

- Neural networks are networks of neurons as in the real biological brain.
- **Neurons** are highly specialized cells that transmit impulses with animals to cause a change in a target cell such as a muscle effector cell or glandular cell.
- The **axon**, is the primary conduit through which the neuron transmits impulses to neurons downstream in the signal chain
- Humans: 10^{11} neurons of > 20 types, 10^{14} synapses, 1ms-10ms cycle time
- Signals are noisy “spike trains” of electrical potential

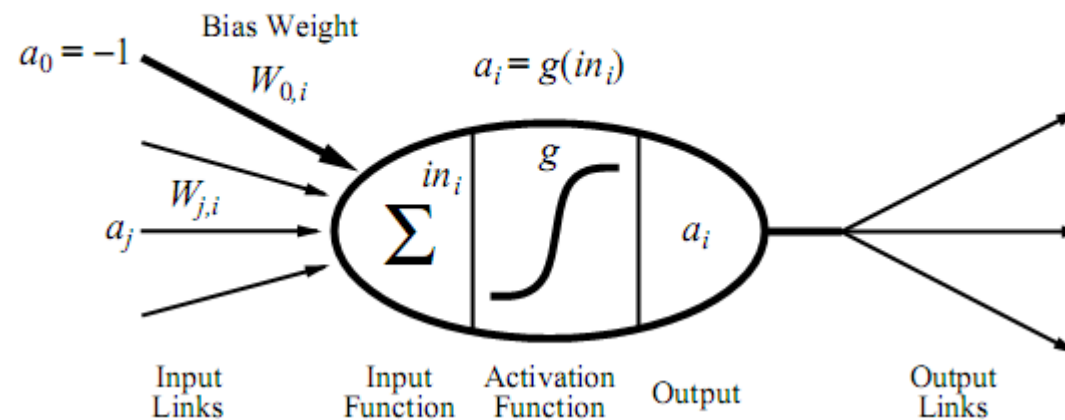


- What we refer to as Neural Networks in the course are mostly Artificial Neural Networks (ANN).
- ANN are approximation of biological neural networks and are built of physical devices, or simulated on computers.
- ANN are parallel computational entities that consist of multiple simple processing units that are connected in specific ways in order to perform the desired tasks.
- Remember: **ANN are computationally primitive approximations of the real biological brains.**

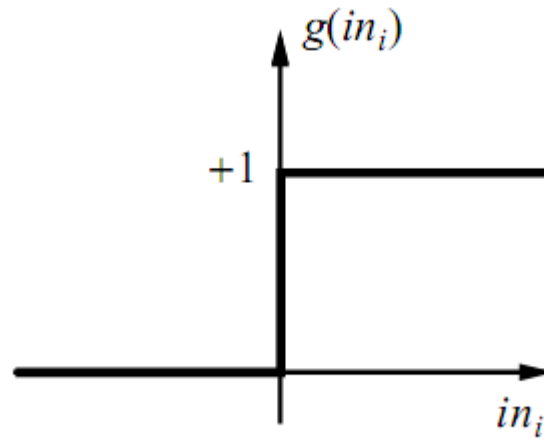
- Neural networks vs. classical symbolic computing
 1. Sub-symbolic vs. Symbolic
 2. Non-modular vs. Modular
 3. Distributed representation vs. Localist representation
 4. Bottom-up vs. Top-down (Evolution vs. Design)
 5. Parallel processing vs. Sequential processing
- In reality however, it can be observed that the distinctions become increasingly less obvious!

- Output is a „squashed“ linear function of the inputs:

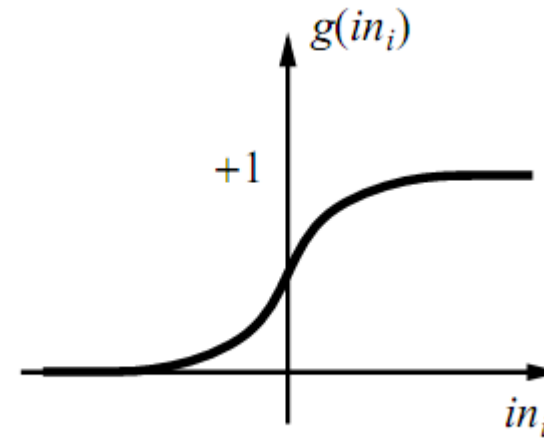
$$a_i \leftarrow g(in_i) = g(\sum_j W_{j,i} a_j)$$



- A clear oversimplification of real neurons, but its purpose is to develop understanding of what networks of simple units can do.



(a)

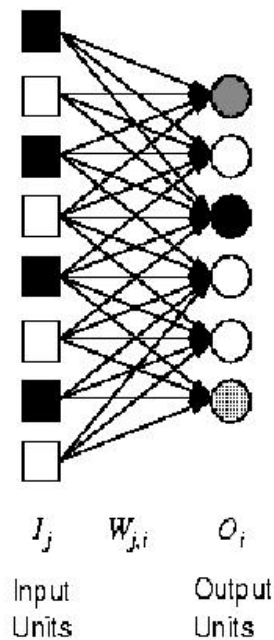


(b)

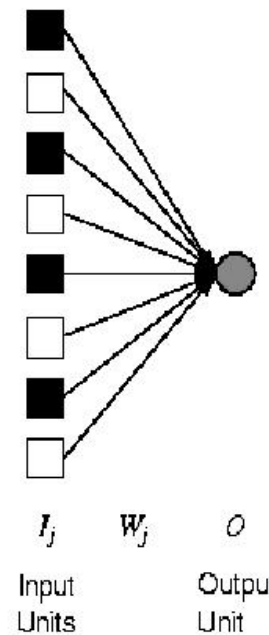
- (a) is a step function or threshold function
- (b) is a sigmoid function $1/(1+e^{-x})$

- Changing the bias weight $w_{0,i}$ moves the threshold location

- McCulloch-Pitts neurons can be connected together in any desired way to build an artificial neural network.
- A construct of one input layer of neurons that feed forward to one output layer of neurons is called **Perceptron**.



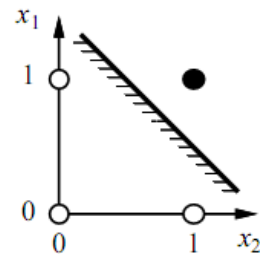
Perceptron Network



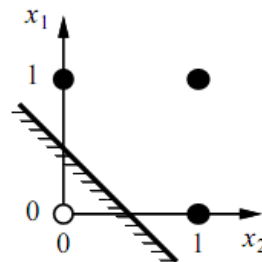
Single Perceptron

- A perceptron with $g = \text{step function}$ can model Boolean functions and linear classification:
 - As we will see, a perceptron can represent AND, OR, NOT, but not XOR
- A perceptron represents a linear separator for the input space

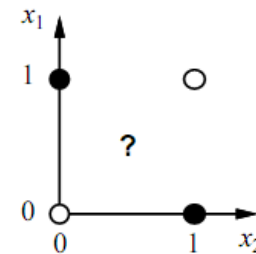
$$\sum_j W_j x_j > 0 \quad \text{or} \quad \mathbf{W} \cdot \mathbf{x} > 0$$



(a) x_1 and x_2

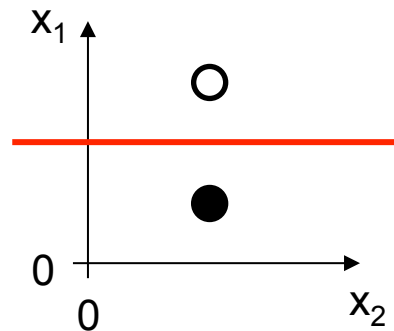


(b) x_1 or x_2



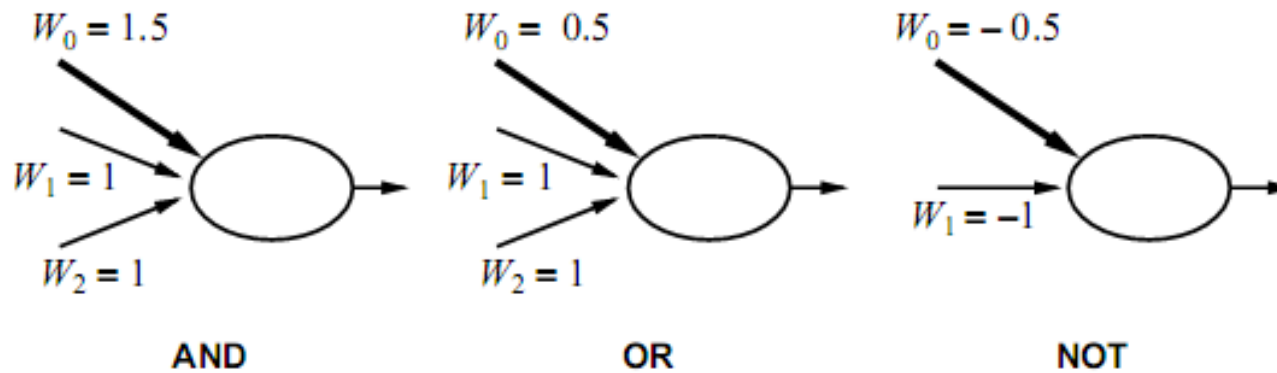
(c) x_1 xor x_2

- Threshold perceptrons can represent only linearly separable functions (i.e. functions for which such a separation hyperplane exists)



- Such perceptrons have limited expressivity, but there exists an algorithm that can fit a threshold perceptron to any linearly separable training set.

Example: Logical Functions



- McCulloch and Pitts: Boolean function can be implemented with a artificial neuron (not XOR).

Example: Finding Weights for AND Operation

- There are two input weights $W1$ and $W2$ and a threshold $W0$. For each training pattern the perceptron needs to satisfy the following equation:

$$\text{out} = \text{sgn}(W1 \cdot \text{in1} + W2 \cdot \text{in2} - W0)$$

- For a binary AND there are four training data items available that lead to four inequalities:

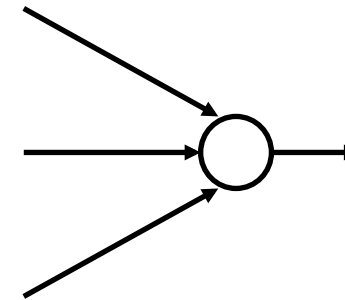
– $W1 \cdot 0 + W2 \cdot 0 - W0 < 0$	$\Rightarrow W0 > 0$
– $W1 \cdot 0 + W2 \cdot 1 - W0 < 0$	$\Rightarrow W2 < 0$
– $W1 \cdot 1 + W2 \cdot 0 - W0 < 0$	$\Rightarrow W1 < 0$
– $W1 \cdot 1 + W2 \cdot 1 - W0 \geq 0$	$\Rightarrow W1 + W2 \geq W0$

- There is an obvious infinite number of solutions that realize a logical AND; e.g. $W1 = 1$, $W2 = 1$ and $W0 = 1.5$.

- XOR:
 - $W1*0 + W2*0 - W0 < 0$ $\Rightarrow W0 > 0$
 - $W1*0 + W2*1 - W0 \geq 0$ $\Rightarrow W2 \geq 0$
 - $W1*1 + W2*0 - W0 \geq 0$ $\Rightarrow W1 \geq 0$
 - $W1*1 + W2*1 - W0 < 0$ $\Rightarrow W1 + W2 < W0$
- The 2nd and 3rd inequalities are not compatible with inequality 4, and there is no solution to the XOR problem.
- XOR requires two separation hyperplanes!
- There is thus a need for more complex networks that combine simple perceptrons to address more sophisticated classification tasks.

- Mathematically artificial neural networks are represented by weighted directed graphs.
- In more practical terms, a neural network has activations flowing between processing units via one-way connections.
- There are three common artificial neural network architectures known:
 - Single-Layer Feed-Forward (Perceptron)
 - Multi-Layer Feed-Forward
 - Recurrent Neural Network

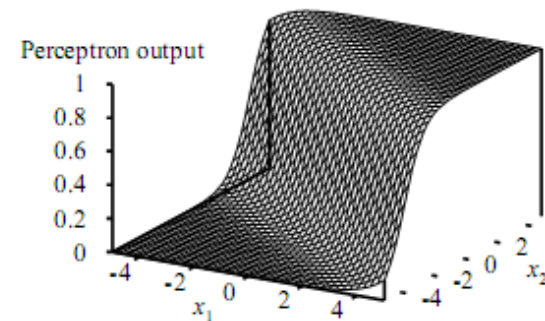
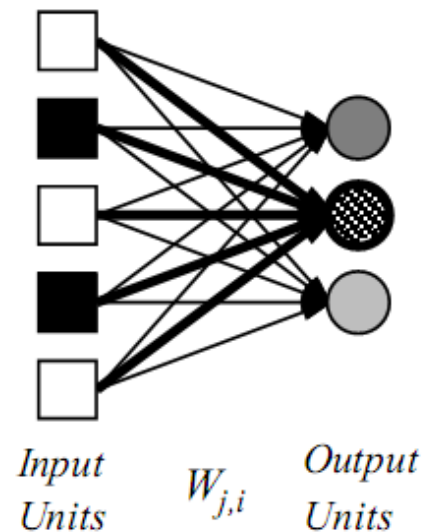
- A Single-Layer Feed-Forward Structure is a simple perceptron, and has thus
 - one input layer
 - one output layer
 - NO feed-back connections



- Feed-forward networks implement functions, have no internal state (of course also valid for multi-layer perceptrons).

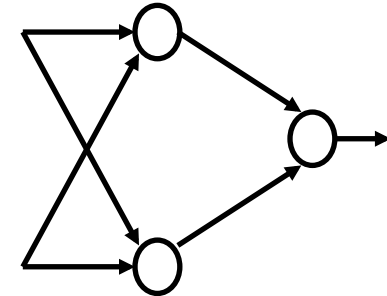
Single-Layer Feed-Forward: Example

- Output units all operate separately – no shared weights
- Adjusting weights moves the location, orientation, and steepness of cliff (i.e., the separation hyperplane).

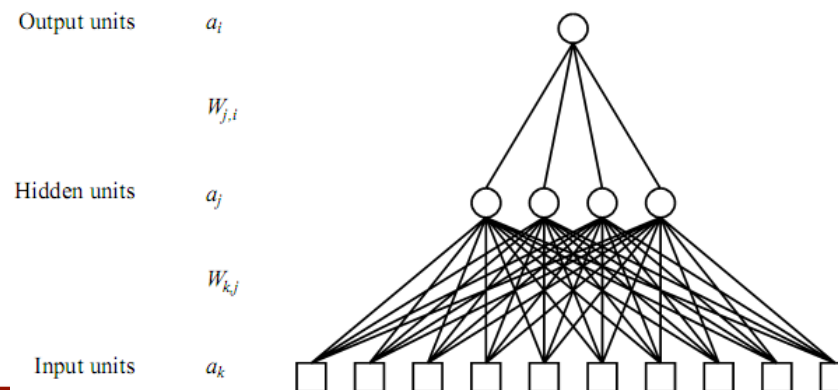


- Multi-Layer Feed-Forward Structures have:
 - one input layer
 - one output layer
 - one or many hidden layers of processing units

- The hidden layers are between the input and the output layer, and thus hidden from the outside world: no input from the world, not output to the world.

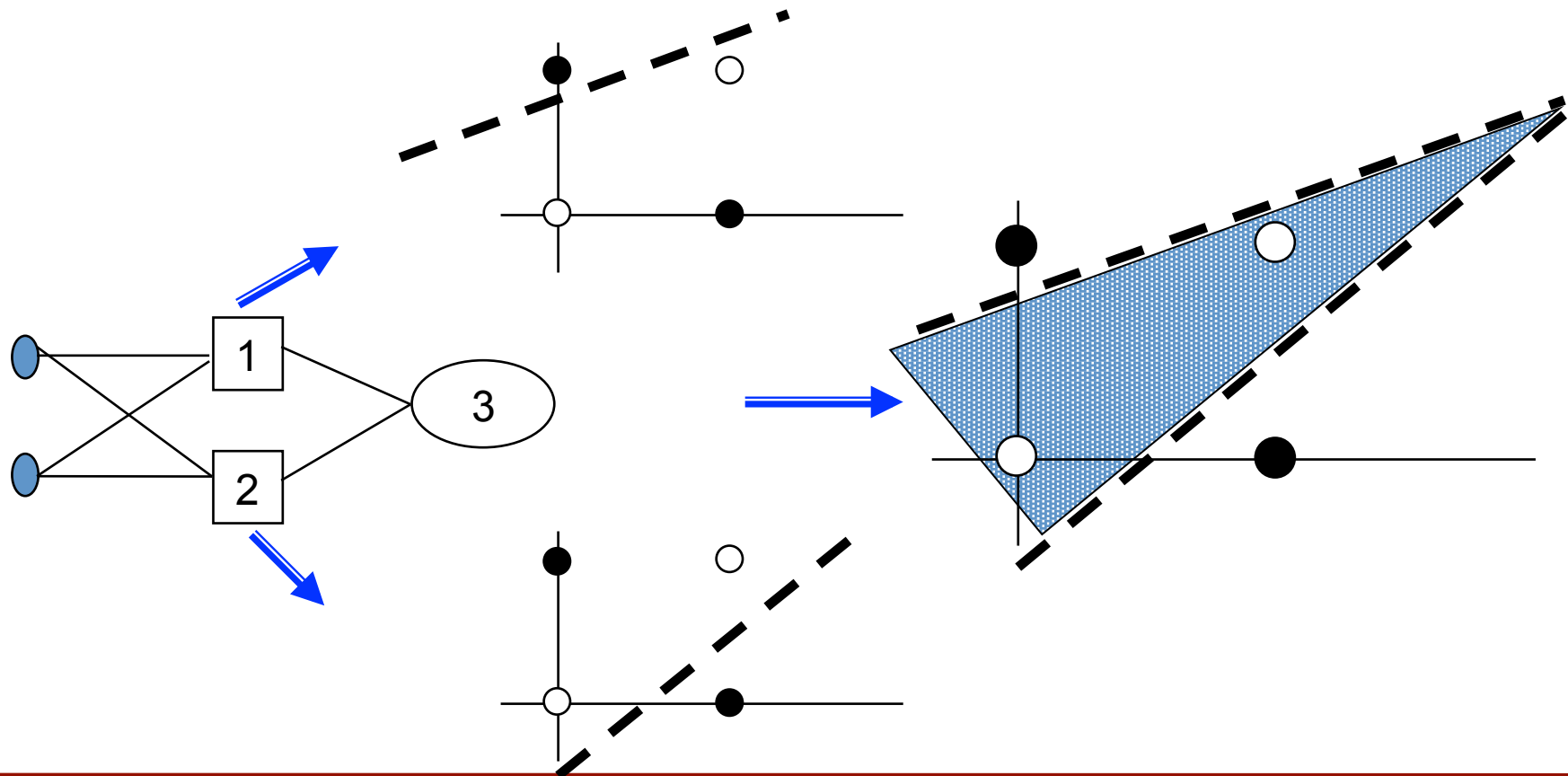


- Multi-Layer Perceptrons (MLP) have fully connected layers.
- The numbers of hidden units is typically chosen by hand; the more layers, the more complex the network (see Step 2 of Building Neural Networks)
- Hidden layers enlarge the space of hypotheses that the network can represent.
- Learning done by back-propagation algorithm → errors are back-propagated from the output layer to the hidden layers.

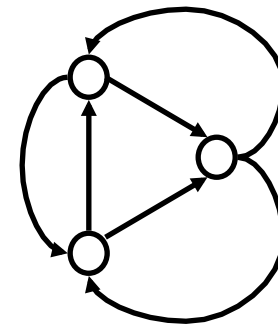


Simple MLP Example

- XOR Problem: Recall that XOR cannot be modeled with a Single-Layer Feed-Forward perceptron.



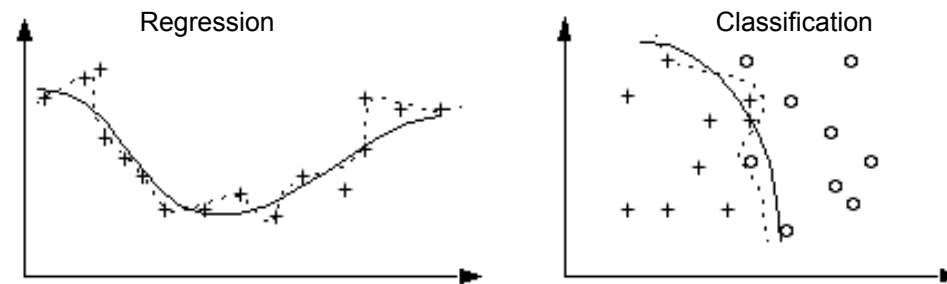
- Recurrent networks have at least one feedback connection:
 - They have directed cycles with delays: they have internal states (like flip flops), can oscillate, etc.
 - The response to an input depends on the initial state which may depend on previous inputs.
 - This creates an internal state of the network which allows it to exhibit dynamic temporal behaviour; offers means to model short-time memory
 - Boltzmann machines use stochastic activation functions, \approx *Markov Chain Monte Carlo* in Bayes nets



- Building a neural network for particular problems requires multiple steps:
 1. Determine the input and outputs of the problem;
 2. Start from the simplest imaginable network, e.g. a single feed-forward perceptron;
 3. Find the connection weights to produce the required output from the given training data input;
 4. Ensure that the training data passes successfully, and test the network with other training/testing data;
 5. Go back to Step 3 if performance is not good enough;
 6. Repeat from Step 2 if Step 5 still lacks performance; or
 7. Repeat from Step 1 if the network does still not perform well enough.

- Neural networks have two important aspects to fulfill:
 - They must learn decision surfaces from training data, so that training data (and test data) are classified correctly;
 - They must be able to generalize based on the learning process, in order to classify data sets it has never seen before.
- Note that there is an important trade-off between the learning behavior and the generalization of a neural network: The better a network learns to successfully classify a training sequence (that might contain errors) the less flexible it is with respect to arbitrary data.

- Noise in the actual data is never a good thing, since it limits the accuracy of generalization that can be achieved no matter how extensive the training set is.
- Non-perfect learning is better in this case!



„Perfect“ learning achieves the dotted separation, while the desired one is in fact given by the solid line.

- However, injecting artificial noise (so-called **jitter**) into the inputs during training is one of several ways to improve generalization

- There are many methods for estimating generalization error.
- Single-sample statistics
 - In linear models, statistical theory provides estimators that can be used as crude estimates of the generalization error in nonlinear models with a "large" training set.
- Split-sample or hold-out validation.
 - The most commonly used method for estimating the generalization error in ANN is to reserve some data as a "test set", which must not be used during training.
 - The test set must represent the cases that the ANN should generalize to. A re-run with the test set provides an unbiased estimate of the generalization error, provided that the test set was chosen randomly.
 - The disadvantage of split-sample validation is that it reduces the amount of data available for both training and validation.

- Cross-validation (e.g., leave one out).
 - Cross-validation is an improvement on split-sample validation that allows the use of all of the data for training.
 - The disadvantage of cross-validation is that the net must be retrained many times.
- Bootstrapping.
 - Bootstrapping is an improvement on cross-validation that often provides better estimates of generalization error at the cost of even more computing time.
- No matter which method is applied, the estimate of the generalization error of the best network will be optimistic.
- If several networks are trained using one data set, and a second (validation set) is used to decide which network is best, a third test set is required to obtain an unbiased estimate of the generalization error of the chosen network.

- Learning is based on training data, and aims at appropriate weights for the perceptrons in a network.
- Direct computation is in the general case not feasible.
- An initial random assignment of weights simplifies the learning process that becomes an iterative adjustment process.
- In the case of single perceptrons, learning becomes the process of moving hyperplanes around; parametrized over time t : $\mathbf{W}_i(t+1) = \mathbf{W}_i(t) + \Delta \mathbf{W}_i(t)$

- The squared error for an example with input \mathbf{x} and true output y is

$$E = \frac{1}{2}Err^2 \equiv \frac{1}{2}(y - h_{\mathbf{W}}(\mathbf{x}))^2 ,$$

- Perform optimization search by gradient descent

$$\begin{aligned} \frac{\partial E}{\partial W_j} &= Err \times \frac{\partial Err}{\partial W_j} = Err \times \frac{\partial}{\partial W_j} (y - g(\sum_{j=0}^n W_j x_j)) \\ &= -Err \times g'(in) \times x_j \end{aligned}$$

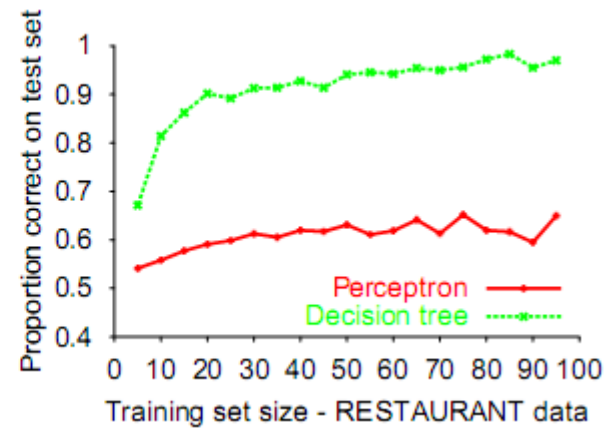
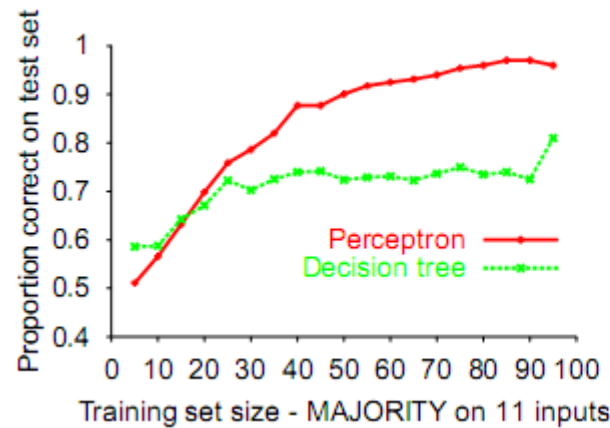
- Simple weight update rule

$$W_j \leftarrow W_j + \alpha \times Err \times g'(in) \times x_j$$

- positive error \Rightarrow increase network output:
 - increase weights on positive inputs,
 - decrease on negative inputs

- The weight updates need to be applied repeatedly for each weight W_i in the network, and for each training suite in the training set.
- One such cycle through all weights is called an **epoch** of training.
- Eventually, mostly after many epochs, the weight changes converge towards zero and the training process terminates.
- The perceptron learning process always finds a set of weights for a perceptron that solves a problem correctly with a finite number of epochs, if such a set of weights exists.
- If a problem can be solved with a separation hyperplane, then the set of weights is found in finite iterations and solves the problem correctly.

- Perceptron learning rule converges to a consistent function for any linearly separable data set



- Perceptron learns majority function easily, Decision-Tree is hopeless
- Decision-Tree learns restaurant function easily, perceptron cannot represent it.

- Output layer: same as for single-layer perceptron,

$$W_{j,i} \leftarrow W_{j,i} + \alpha \times a_j \times \Delta_i$$

where $\Delta_i = Err_i \times g'(in_i)$

- Hidden layer: back-propagate the error from the output layer:

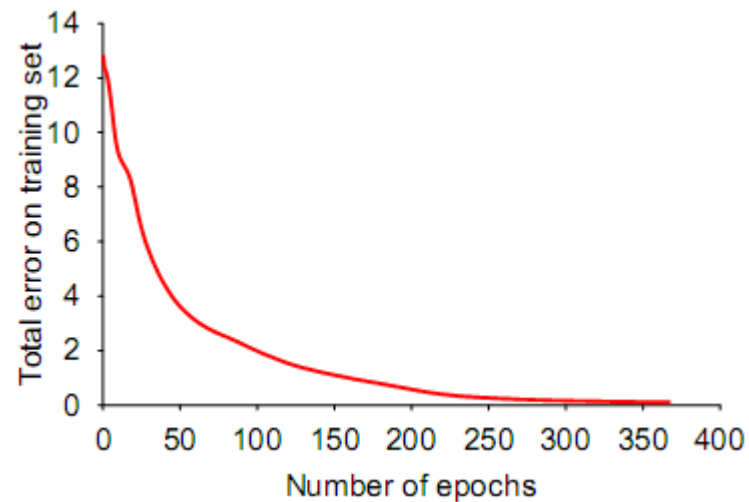
$$\Delta_j = g'(in_j) \sum_i W_{j,i} \Delta_i .$$

- Update rule for weights in hidden layer:

$$W_{k,j} \leftarrow W_{k,j} + \alpha \times a_k \times \Delta_j .$$

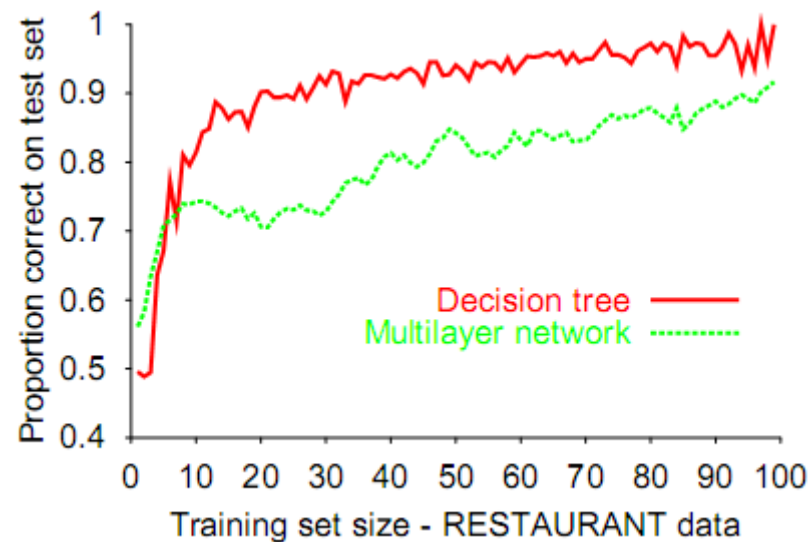
- *Most neuroscientists deny that back-propagation occurs in the brain.*

- At each epoch, sum gradient updated for all examples
- Training curve for 100 restaurant examples converges to a perfect fit to the training data



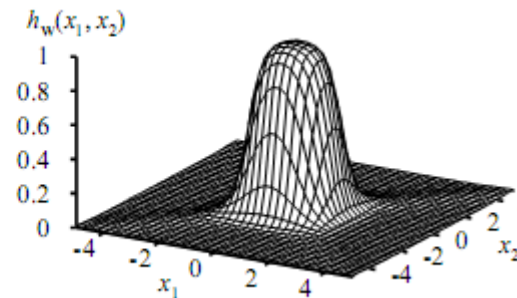
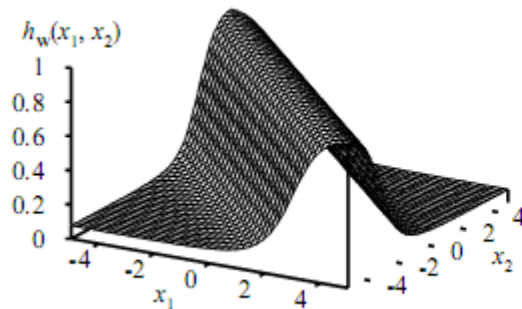
- Typical problems: slow convergence, local minima

- Learning curve for MLP with 4 hidden units (as in our restaurant example):

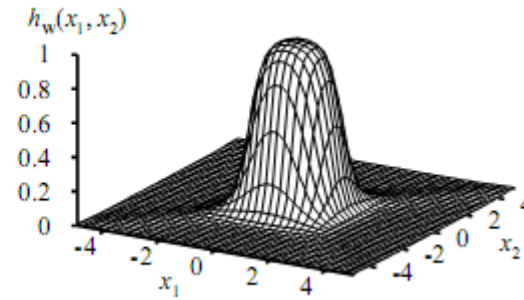
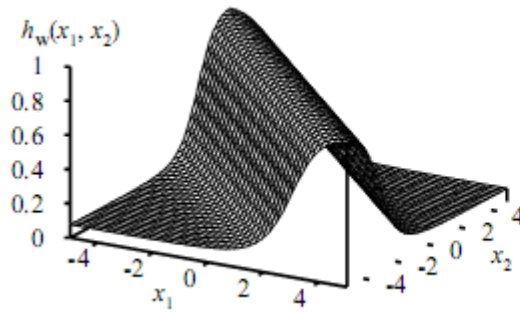


- MLPs are quite good for complex pattern recognition tasks, but resulting hypotheses cannot be understood easily

- 2 layers can represent all **continuous functions**
- 3 layers can represent **all functions**



- Combine two opposite-facing threshold functions to make a ridge.
- Combine two perpendicular ridges to make a bump.
- Add bumps of various sizes and locations to fit any surface
- The required number of hidden units grows **exponentially** with the number of inputs ($2^n/n$ for all boolean functions)

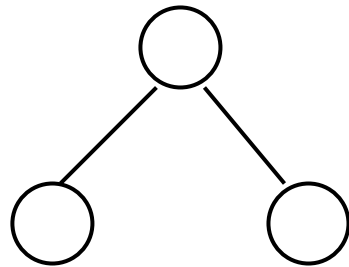
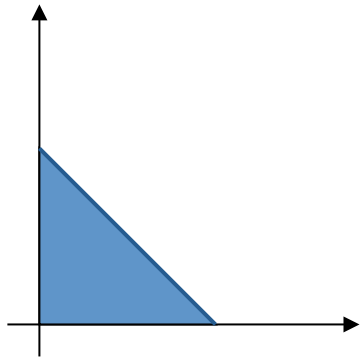


- The more hidden units, the more bumps
- Single, sufficiently large hidden layer can represent any continuous function of the inputs with arbitrary accuracy
- Two layers are necessary for discontinuous functions
- For any particular network structure, it becomes harder to characterize exactly which functions can be represented and which ones cannot.

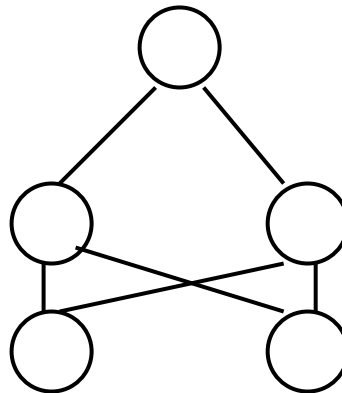
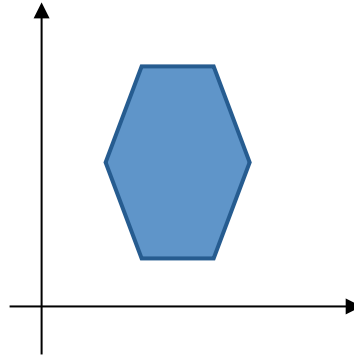
- Rule of Thumb 1: even if the function to learn is slightly non-linear, the generalization may be better with a simple linear model than with a complicated non-linear model; if there is too little data or too much noise to estimate the non-linearities accurately.
- In MLPs with threshold activation functions, two hidden layers are needed for full generality.
- In MLPs with any continuous non-linear hidden-layer activation functions, one hidden layer with an arbitrarily large number of units suffices for the "universal approximation" property. However, there is no theory yet that tells how many hidden units are needed to approximate any given function.

- Rule of Thumb 2: If there is only one input, there seems to be no advantage to using more than one hidden layer; things get much more complicated when there are two or more inputs.
- Using two hidden layers complicates the problem of local minima, and it is important to use lots of random initializations or other methods for global optimization. Local minima with two hidden layers can have extreme spikes or blades even when the number of weights is much smaller than the number of training cases.
- More than two hidden layers can be useful in certain architectures such as cascade correlation, and in special applications, such as the two-spirals problem and ZIP code recognition.

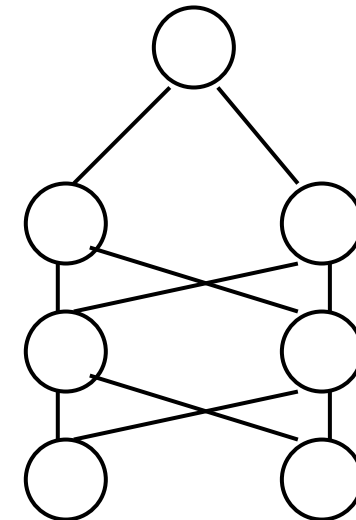
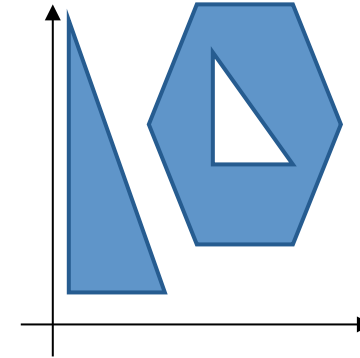
Example: Number of Hidden Layers



1st layer draws linear boundaries



2nd layer combines the boundaries.



3rd layer can generate arbitrarily boundaries.

Some application examples

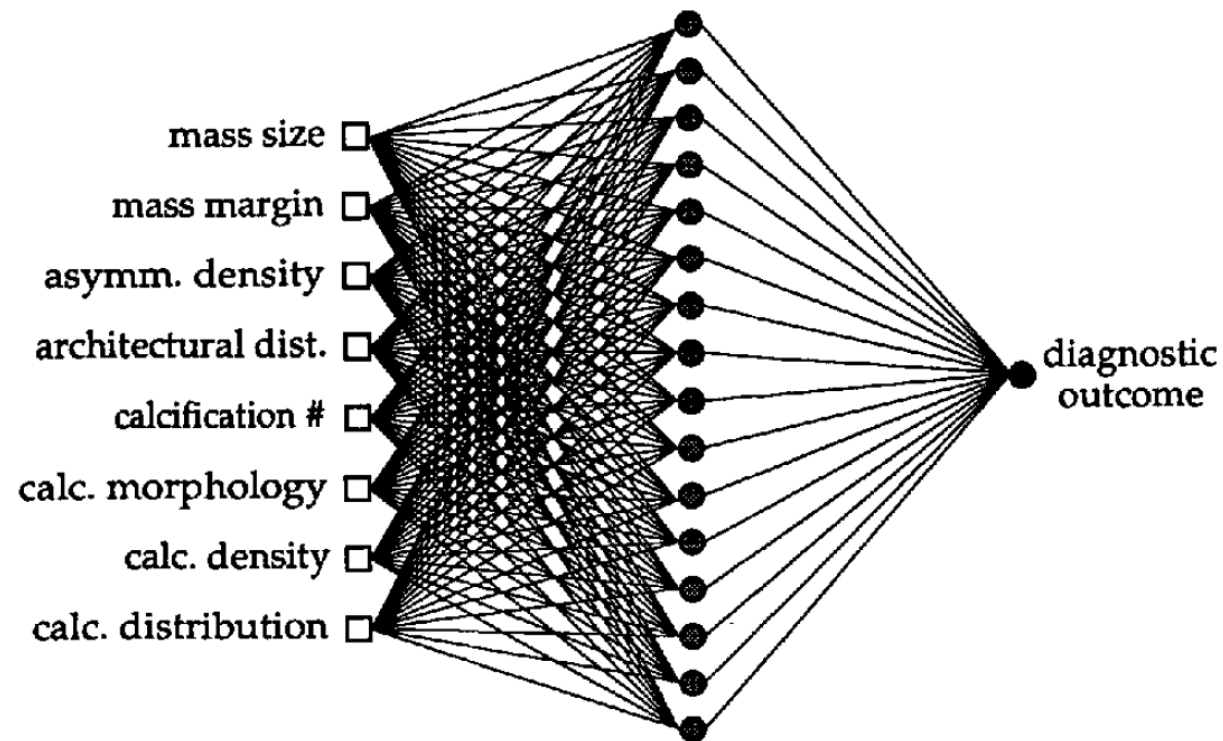
ILLUSTRATION BY A LARGER EXAMPLE

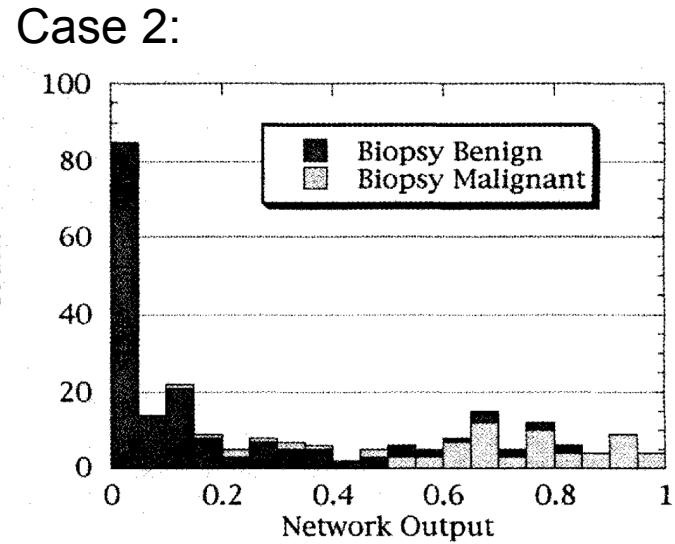
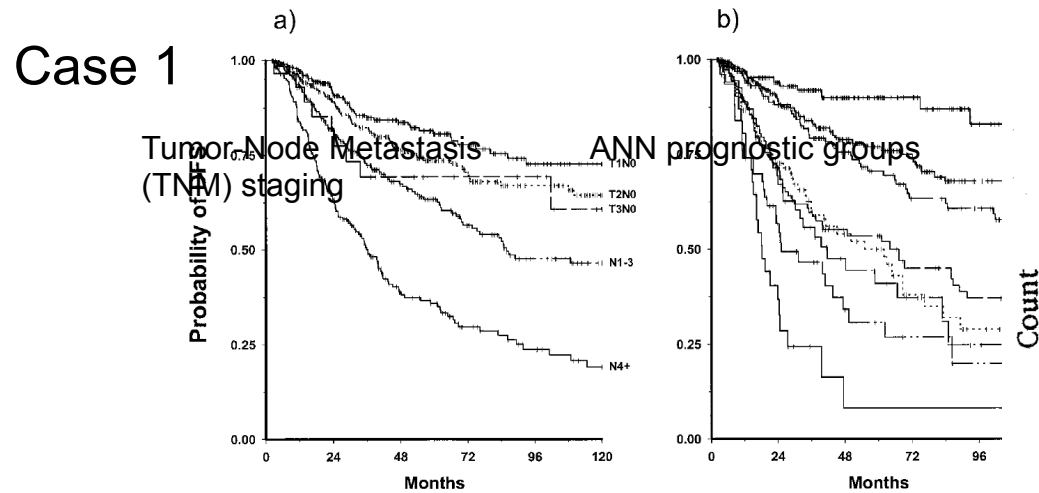
- Traditionally statistical techniques are employed, but ANN are argued to be more promising, when
 - Proportionality of hazard assumption cannot be applied to data
 - Relationship between variables is complex and unknown
 - Dependencies between variables
- ANN have application to cancer recurrence prediction in
 - Classification of risk group
 - Risk of recurrence
 - Time to relapse estimation
- Important for treatment decision-making
 - Patients with excellent prediction should be spared any treatment-induced toxicity
 - Patients with high risks should be given very aggressive regimes

- Different ANN proposals use different settings
 - Observations as input variables
 1. *patient age, tumor size, number of axillary metastases, estrogen and progesterone receptor levels, S-phase fraction, and tumor ploidy (7 inputs)*
 2. *Radiographic features: mass size and margin, asymmetric density, architectural distortion, calcification number and morphology... (8 inputs)*
 3. *Mean, extreme and standard error of perimeter, area, smoothness, compactness, fractal dimension, symmetry etc. of cell nuclei (30 inputs)*
 - Three-layer MLP with back-propagation
 1. *One hidden layer with 5 processing elements*
 2. *One hidden layer with 16 nodes*
 3. *One layer with 20 hidden nodes*
 - Output layer categorizes the prediction
 1. *8 risk groups in terms of disease-free survival (DSF) of a 60 months period*
 2. *Single output: 0.0 for benign \Rightarrow 1.0 for malignant*
 3. *Vector with 20 entries (each entry represents DSF probability for six month period of up to 10 years)*

1) De Laurentiis et al. Clinical Cancer Research 5, 1999; 2) Floyd et al. Cancer 74(11), 1994; 3) Chi et al. American Medical Informatics Association Symposium 2007

- Architecture of neural network for predicting biopsy results of radiographic findings (case 2)





Over 72% of random pairs of patients with one relapsed and one disease-free were ranked correctly.

	Az^a (SD)
ANN	0.726 (0.02)
TNM	0.677 (0.02)

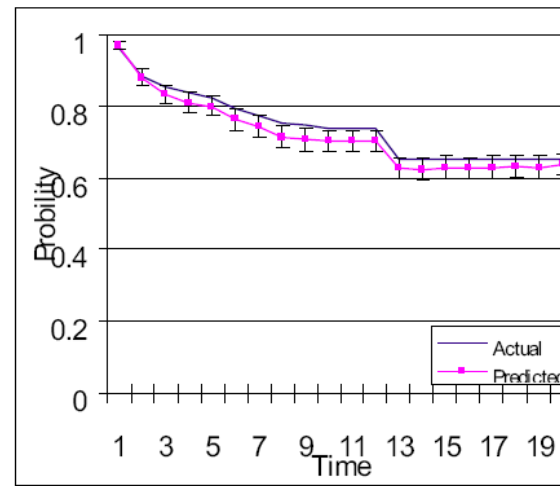
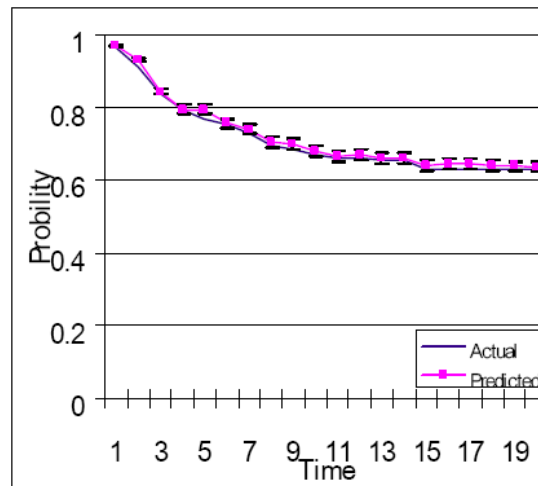
^a Az , area-under-the-ROC-curve

Comparing predictions and observations are highly accurate.

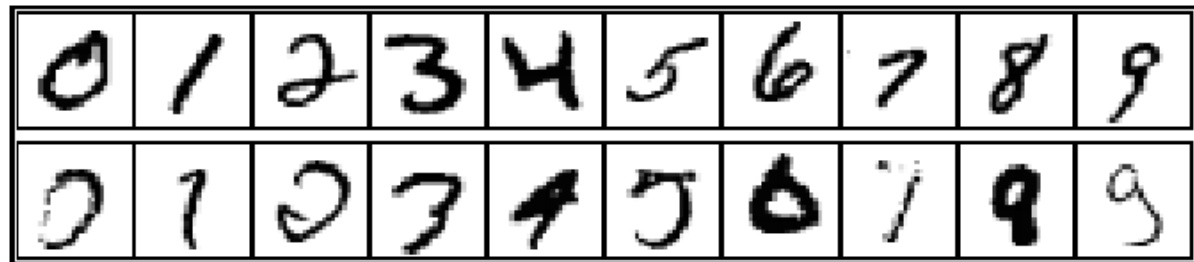
With the cases of output < 0.1 not sent to biopsy, 99 of 168 benign and 100% of the malignant cases are classified correctly

Case 3:

Actual survival compared to predicted survival curve for two different datasets (error bars mark 95% confidence intervals).



- There are endless further examples: :
 - Handwriting Recognition
 - Time Series Prediction
 - Kernel Machines (Support Vectore Machines)
 - Data Compression
 - Financial Predication
 - Speech Recognition
 - Computer Vision
 - Protein Structures
 - ...




SUMMARY

- Most brains have lots of neurons, each neuron approximates a linear-threshold unit.
- Perceptrons (one-layer networks) approximate neurons, but are as such insufficiently expressive.
- Multi-layer networks are sufficiently expressive; can be trained to deal with generalized data sets, i.e. via error back-propagation.
- Multi-layer networks allow for the modeling of arbitrary separation boundaries, while single-layer perceptrons only provide linear boundaries.
- Endless number of applications: Handwriting Recognition, Time Series Prediction, Bioinformatics, Kernel Machines (Support Vector Machines), Data Compression, Financial Prediction, Speech Recognition, Computer Vision, Protein Structures...

REFERENCES

- De Luarentiis, M., De Placido, S., Bianco, A.R., Clark, G.M. & Ravdin, P.M. (1999): A Prognostic Model That Makes Quantitative Estimates of Probability of Relapse for Breast Cancer Patients. *Clinical Cancer Research* 5, pp. 4133 – 4139.
- Elman, J. L. (1990): Finding structure in time. *Cognitive Science* 14, pp. 179–211.
- Gallant, S. I. (1990): Perceptron-based learning algorithms. *IEEE Transactions on Neural Networks* 1 (2), pp. 179-191.
- McCulloch, W.S. & Pitts, W. (1943): A Logical Calculus of the Ideas Immanent in Nervous Activity. *Bulletin of Mathematical Biophysics* 5, pp. 115-133.
- Rosenblatt, F. (1958): The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Reviews* 65, pp. 386-408.
- Rumelhart, D.E., Hinton, G. E. & Williams, R. J. (1986): Learning representations by back-propagating errors. *Nature* 323, pp. 533-536.
- Supervised learning demo (perceptron learning rule) at <http://lcn.epfl.ch/tutorial/english/perceptron/html/>

Next Lecture

#	Title
1	Introduction
2	Propositional Logic
3	Predicate Logic
4	Theorem Proving, Description Logics and Logic Programming
5	Search Methods
6	CommonKADS
7	Problem Solving Methods
8	Planning
9	Agents
10	Rule Learning
11	Inductive Logic Programming
12	Formal Concept Analysis
13	Neural Networks
 14	Semantic Web and Exam Preparation

Questions?

