

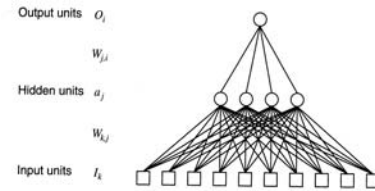
Knowledge-based systems in Bioinformatics, 1MB602

Lecture 11: Neural networks

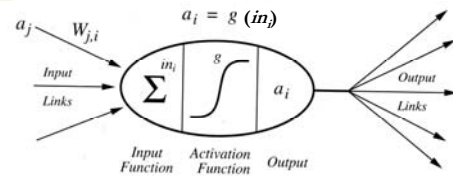
Artificial neural networks

- Inspired by how the brain works – a mathematical model for the operation of the brain
- Brain versus computers:
 - serial versus parallel computing
 - even though a computer is much faster than in raw switching speed, the brain is faster at what it does
- An ANN is a number of **nodes** (units) connected by **links**. Each link is associated with a numerical **weight**.
 - Training set: $(x_p/(y_p)), (x_p/(x_p)), \dots, (x_p/(x_p))$
 - Learning in an ANN is reduced to the process of using the training data to tune the weights so that the network represents the function f

Network structure

- Feed-forward network:** all units are connected to all units in the next layer
 - One (sufficiently large) hidden layer can represent any continuous function
 - More hidden layers can even represent discontinuous functions
- 
- Recurrent network:** feed back loops, internal states (memory):
 - E.g. The brain is clearly a recurrent network

Units

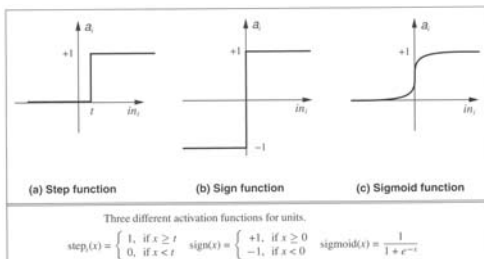


- Input function:** linear component, in_i , that compute the weighted sum of the units input values

$$in_i = \sum_j W_{j,i} \cdot a_j$$
- Activation function:** nonlinear transformation, g , of the input function into the unit's activation value

$$a_i = g(in_i)$$

Activation functions

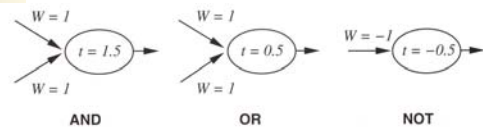


Typically, the threshold of the activation function is embedded in the input function:

$$a_i = \text{step}_t\left(\sum_{j=1}^n W_{j,i} \cdot a_j\right) = \text{step}_0\left(\sum_{j=0}^n W_{j,i} \cdot a_j\right), \text{ where } W_{0,j} = t \text{ and } a_0 = -1$$

Boolean functions

- Units can represent the basic logical gates
- Thus, units can build networks that can represent any Boolean function



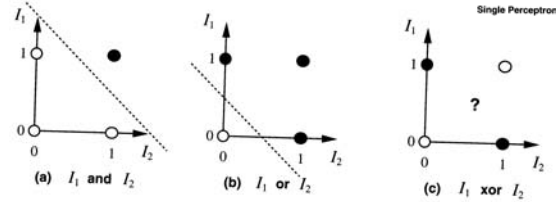
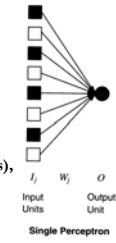
Optimal network structures

- Too small network: the network will be incapable of representing the desired function
- Too large network: the network can memorize all the examples by forming a lookup table
 - Overfitting!
- Finding the optimal network structure is itself a search problem
 - Potentially time consuming since very state in this search involves training and evaluating a neural network of a particular size
 - Genetic algorithms
 - Hill-climbing
 - Evaluation: e.g. cross validation

THE LINNAEUS CENTRE FOR BIOINFORMATICS
http://www.lcb.lu.se

Perceptrons

- Perceptrons: single-layer, feed-forward networks
 - Majority function: outputs 1 if a majority of the n inputs are 1 (would require a decision tree with $O(2^n)$ nodes)
- A perceptron can only represent a function if there is a line that separates all the white dots (0s) from the black dots (1s), i.e. **functions that are linearly separable**

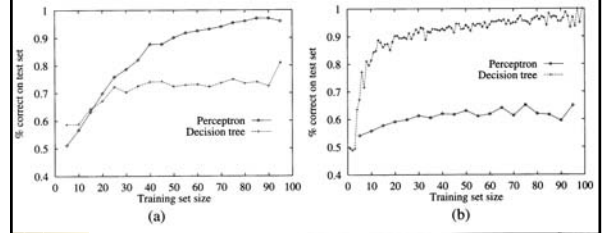


Learning linearly separable functions

- Only one unit: $O = \text{Step}_0(\sum_j W_j I_j)$
- $\text{Err} = T - O$, where T is the correct output
- **Perceptron learning rule:** $W_j \leftarrow W_j + \alpha \times I_j \times \text{Err}$
 - α is called the learning rate
- **Learning algorithm:**
 - Initiate weights, e.g. Random values between 0 and 1
 - For each example
 - Compute O
 - Update weights with the learning rule
 - Repeat until all examples correctly predicted
- **Epoch:** updating all weights for every example
- **Note:** the perceptron rule is guaranteed to learn a linearly separable function given enough examples!

THE LINNAEUS CENTRE FOR BIOINFORMATICS
http://www.lcb.lu.se

Perceptrons versus decision trees: Example



(a) Majority function
(b) Waiting problem

Example	Attributes										Goal
	All	Bar	Fin	Flan	Flur	Price	Rank	Res	Type	Year	
X1	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	Yes
X2	Yes	No	No	Yes	Full	5	No	No	Thai	30-40	No
X3	No	Yes	No	No	Some	5	No	No	Berger	0-10	Yes
X4	Yes	No	Yes	Yes	Full	5	No	No	Thai	10-20	Yes
X5	Yes	No	No	No	Full	\$\$\$	Yes	Yes	French	>50	No
X6	Yes	Yes	No	No	Some	\$\$	Yes	Yes	Italian	0-10	Yes
X7	No	Yes	No	No	None	5	Yes	No	Berger	0-10	No
X8	No	No	No	Yes	Some	\$\$	Yes	Yes	Thai	0-10	Yes
X9	No	Yes	Yes	No	Full	5	Yes	No	Berger	>50	No
X10	Yes	Yes	Yes	Yes	Full	\$\$\$	Yes	No	Italian	10-20	No
X11	No	No	No	No	None	5	No	No	Thai	0-10	No
X12	Yes	Yes	Yes	Yes	Full	5	No	No	Berger	30-40	Yes

How to encode the examples in ANN

Example	Attributes										Goal
	All	Bar	Fin	Flan	Flur	Price	Rank	Res	Type	Year	
X1	Yes	No	No	Yes	Some	\$\$\$	No	Yes	French	0-10	Yes
X2	Yes	No	No	Yes	Full	5	No	No	Thai	30-40	No
X3	No	Yes	No	No	Some	5	No	No	Berger	0-10	Yes
X4	Yes	No	Yes	Yes	Full	5	No	No	Thai	10-20	Yes
X5	Yes	No	No	No	Full	\$\$\$	Yes	Yes	French	>50	No
X6	Yes	Yes	No	No	Some	\$\$	Yes	Yes	Italian	0-10	Yes
X7	No	Yes	No	No	None	5	Yes	No	Berger	0-10	No
X8	No	No	No	Yes	Some	\$\$	Yes	Yes	Thai	0-10	Yes
X9	No	Yes	Yes	No	Full	5	Yes	No	Berger	>50	No
X10	Yes	Yes	Yes	Yes	Full	\$\$\$	Yes	No	Italian	10-20	No
X11	No	No	No	No	None	5	No	No	Thai	0-10	No
X12	Yes	Yes	Yes	Yes	Full	5	No	No	Berger	30-40	Yes

- **Local encoding:** use single input units for each example attribute
 - E.g. $None = 0.0$, $Some = 0.5$, $Full = 1.0$
- **Distributed encoding:** use one input unit for every attribute value
- The local encoding approach is normally applied to the output

THE LINNAEUS CENTRE FOR BIOINFORMATICS
http://www.lcb.lu.se

Multilayer feed-forward networks

- Problem: assess the blame for the error and divide it among the contributing weights in all layers
- For derivation, we need a continuous activation function: the sigmoid
- Weight update rule:

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

- At the output node(s)

$$\Delta_i = (T_i - O_i) \times g'(in_i)$$

- The propagation rule for:

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

THE LINNAEUS CENTRE FOR BIOINFORMATICS
http://www.lcb.lu.se

The backpropagation algorithm

1. Initialize the weights in the network
2. Compute the Δ values for the output units using the observed error
3. Start with the output layer and repeat for each layer in the network until the last hidden layer
 1. Propagate the Δ values back to the previous layer
 2. Update the weights between the layers
4. Repeat 2-3 for each example
5. Repeat 2-4 until convergence

ANN discussion

- Time complexity for one epoch: $O(m|W|)$, where m is the number of examples and $|W|$ is the number of weights
- Very insensitive to noise
- ANNs are basically black box approach – unlike decision trees they do not provide a clue to how a prediction is made
- Difficult to incorporate prior biological data
- Can also be used for clustering (unsupervised learning): self-organizing maps

Applications (in Bioinformatics)

- Very popular for predicting various protein properties from sliding sequence windows:
 - Secondary structure
 - Disorder
 - Finding sequence motifs
 - ...
- But is also used for more general prediction applications like medical diagnosis, etc
- Classical AI applications
 - Speech recognition
 - Handwriting recognition
 - Image recognition
 - Driving a car!

References

- E. Keedwell, A. Narayanan, Intelligent bioinformatics: the application of artificial intelligence techniques to bioinformatics problems. Chichester : John Wiley, cop. 2005
- S. Russell, P. Norvig, Artificial intelligence: a modern approach, Prentice-Hall, Upper Saddle River, New Jersey, 1995