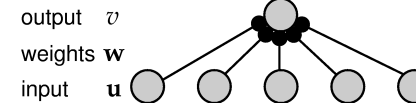


Neural Networks That Learn (Supervised Learning)

Readings: D&A, chapter 8.

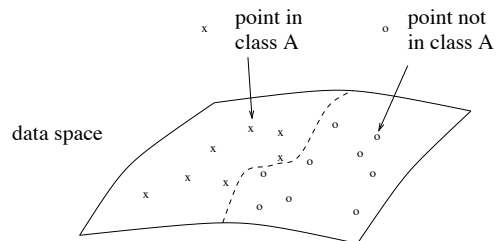
Supervised learning



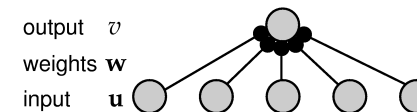
- we want to have a network which is able to do a **classification** of inputs into 2 categories (e.g. A or not A)
- we want to **optimize the parameters = weights \mathbf{w}** of the network so that the network makes as few errors as possible;
- we assume that there exist a data base which contains **examples (\mathbf{u}, v)** of the inputs to learn together with the correct (or 'target') output;
- Final task of the network: **generalize** to new data;

Supervised learning

- The task of learning and generalization corresponds to finding a surface -- a.k.a **discriminant function** -- which separates the elements of class A from the others.



The Perceptron (1) - definition



$$v = \begin{cases} +1 & \text{if } \mathbf{w} \cdot \mathbf{u} - \gamma \geq 0 \\ -1 & \text{if } \mathbf{w} \cdot \mathbf{u} - \gamma < 0 \end{cases}$$

$$\mathbf{w} \cdot \mathbf{u} = \sum_{i=1}^{i=N} w_i u_i$$

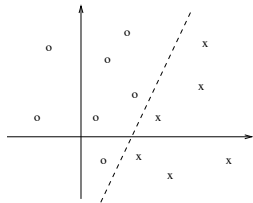
- simplest one-layer network (with binary output);
- the inputs are m **input patterns \mathbf{u}** ;
- \mathbf{w} are the **weights** of the perceptron;
- Task of the perceptron : place each input pattern into one of two classes designated by the desired output $v^m = -1$ or $v^m = +1$;
- The threshold determines the dividing line between values of $\mathbf{w} \cdot \mathbf{u}$ that generate +1 and -1 outputs

The Perceptron (2) - Linear Separability

- The boundary between the 2 classes is given by:

$$\sum_{i=1}^{i=N} w_i u_i - \gamma = 0$$

- This is a **linear equation** and defines a **hyper-plane** in the input space.
- A simple perceptron can only solve problems which are **linearly separable**.



The Perceptron (3): how do we learn the weights?

- Test one data point \mathbf{u}^m after the other, i.e. apply it at input layer and compare the output $v(\mathbf{u}^m)$ to the desired output v^m
- If output is correct, don't take any action;
- If output is incorrect, change \mathbf{w} .
- The **learning rule** is [Rosenblatt, 1958]:

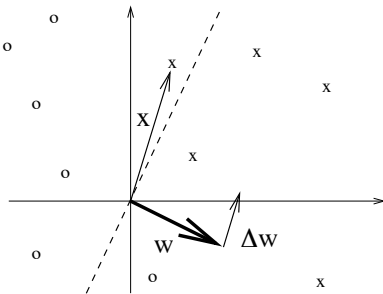
$$\Delta w_i = \eta (v^m - v(\mathbf{u}^m)) u_i^m$$

- where η is the learning rate -- a small parameter.

e.g. $v^m = +1, v(\mathbf{u}^m) = -1$ $w_i \rightarrow w_i + 2\eta u_i$

$$w_i \cdot u_i \rightarrow w_i \cdot u_i + 2\eta u_i^2$$

The Perceptron (4): Learning



- if a point x is misclassified, the weight vector is changed in direction of x . This rotates the separating line in the desired direction.

Batch Learning vs. Online Learning

- Two ways to apply the learning rule:
- Online**: change the weights after presentation of each input data:

$$\Delta w_i = \eta (v^m - v(\mathbf{u}^m)) u_i^m$$

- Batch**: present all the data then change the weights:

$$\Delta w_i = \eta \sum_{m=1}^{m=Ns} (v^m - v(\mathbf{u}^m)) u_i^m$$

- Batch learning is often more effective but a bit more prone to get stuck in **local minima**.
- Online learning is more **plausible biologically**, but error not guaranteed to go down at each step (optimizing for a new pattern can result in unlearning the previous pattern).

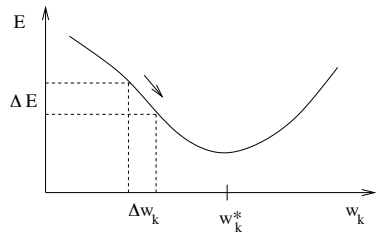
Gradient descent (1)

• The perceptron is a simple case. More generally, we consider a **continuous** output function: $v(\mathbf{u}^m) = g(\mathbf{w} \cdot \mathbf{u}^m - \gamma)$

• The total **quadratic error** is: $E(\mathbf{w}) = \frac{1}{2} \sum_m (v^m - v(\mathbf{u}^m))^2$
= a function of \mathbf{w}

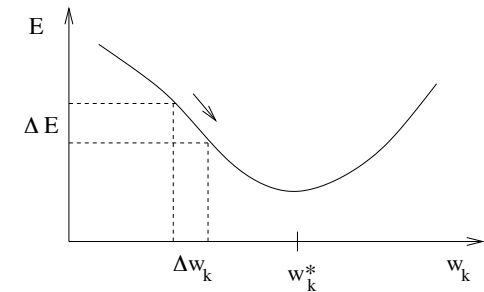
• We want to change the weights such that the error decreases = in **direction of the negative gradient**:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$



Gradient descent (2)

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i}$$



- if we are in a region where the slope of $E(w_k)$ is negative, we want to increase w_k ;
- if we are in a region where the slope is positive, we want to decrease w_k ;
- the steeper the slope, the more we want to change the weights.

Gradient descent (3)

• The gradient can be easily calculated, we get:

$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} = \eta \sum_{m=1}^{m=N_s} g'(\mathbf{w} \cdot \mathbf{u}^m) (v^m - v(\mathbf{u}^m)) \cdot u_i^m$$

- This is known as the **delta rule** [Widrow & Hoff, 1960].
- The perceptron rule is a particular case of this, where $g'=1$.
- 'delta' refers to:

$$\delta_m = g'(\mathbf{w} \cdot \mathbf{u}^m) (v^m - v(\mathbf{u}^m))$$

• Online rule:

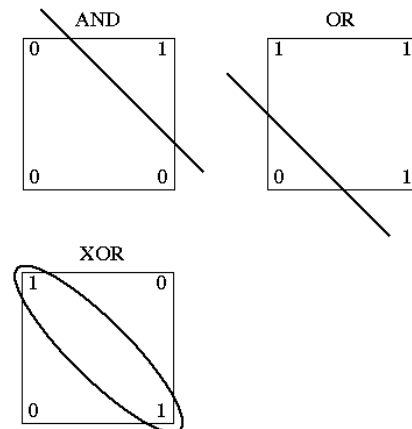
$$\Delta w_i = \eta \delta_m u_i^m$$

Perceptron cannot solve XOR

• perceptron can compute AND and OR but not XOR -- since XOR is not linearly separable.

X	Y	O
0	0	0
0	1	1
1	0	1
1	1	0

XOR



History of AI -- Perceptrons and the dark age of connectionism

- A perceptron was introduced in 1958 by **Frank Rosenblatt** -- a schoolmate of Marvin Minsky.
- He predicted that "**perceptron may eventually be able to learn, make decisions, and translate languages.**"
- An active research program into the paradigm was carried out throughout the 60s but came to a sudden halt with the publication **Minsky and Papert's 1969 book Perceptrons**. They showed that there were severe limitations to what perceptrons could do and that Frank Rosenblatt's claims had been grossly exaggerated.
- The effect of the book was devastating: virtually no research was done in connectionism for 10 years. Rosenblatt died in a boating accident shortly after the book was published.

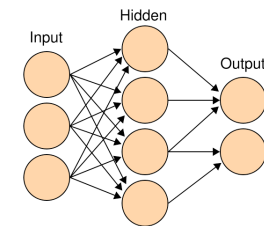
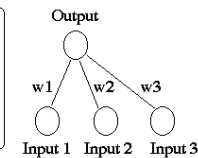


http://en.wikipedia.org/wiki/History_of_artificial_intelligence

Towards Multi-Layer networks

- What to do when the problem is not linearly separable?
- 1) **preprocess** to make the problem separable (e.g. by mapping to a higher dimension space) -- cf Support Vector Machines ; or
- 2) use a **multi-layer network**.
- The most important learning rule for multi-layer networks is the (error) **back-propagation** algorithm. = Generalization of the delta rule [Chauvin & Rumelhart, 1985].

Input 1	Input 2	Input 3	Output
1	1	1	0
1	0	0	1
0	1	0	1
0	0	0	0



Back-propagation algorithm

- **initialize weights** to small random values
- **apply a sample input pattern** r^{in} to the input nodes
- **propagate input** through the network by calculating the rate of nodes in successive layers l

$$r_i^l = g(h_i^{l-1}) = g\left(\sum_j w_{ij}^l r_j^{l-1}\right)$$

- **Compute the delta term** for the output layer

$$\delta_i^{out} = g'(h_i^{out})(v_i^{out} - r_i^{out})$$

- **Back-propagate delta terms** through the network

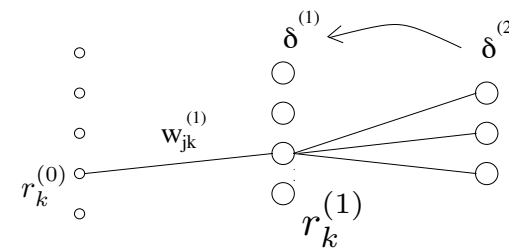
$$\delta_i^{l-1} = g'(h_i^{l-1}) \sum_j w_{ij}^l \delta_j^l$$

- Update weight matrix by adding the term

$$\Delta w_{ij}^l = k \delta_i^l r_j^{l-1}$$

- Repeat until error is sufficiently small.

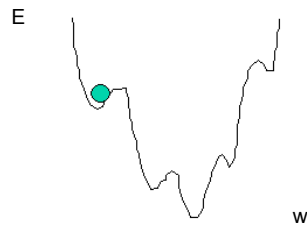
Back-propagation algorithm



- **At output layer**, same as delta rule
- **At hidden layer**, each unit receives the weighted sum of the delta-terms of the units it connects to (in output layer). This serves as delta-term in this layer.

$$\delta_i^{l-1} = g'(h_i^{l-1}) \sum_j w_{ij}^l \delta_j^l$$

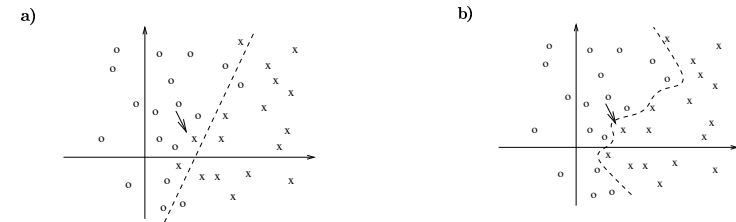
Stuck in a Local Minimum ?



- a general limitation of pure gradient descent methods is the possibility that the network gets **trapped in a local minimum of the Error surface**.
- Solution: include some **stochastic process** that enable random search
- **simulated annealing**: add some noise to the weights values. the noise level is then gradually reduced to ensure convergence.

Over-fitting and Generalization

- is it so good to have a very **flexible** network?
- in some cases it is better to have a network which doesn't perform perfectly on the training data set
- learning the noise in the data = **over-fitting**. This happens when the number of free parameters (weights) in the model is too large.
- stopping the training when the error on the testing data set increases is one way to prevent over-fitting (regularization by **early-stopping**)
- having lots of data is another.



History of AI: revival of connectionism

The introduction of the **Hopfield nets** (1982) by John Hopfield and of the **backpropagation** algorithm by David Rumelhart revived the field of connectionism which had been abandoned since 1970.

The new field was unified and inspired by the appearance of **Parallel Distributed Processing in 1986**—a two volume collection of papers edited by Rumelhart and psychologist James McClelland.

Neural networks would become **commercially successful** in the 1990s, when they began to be used as the engines driving programs like optical character recognition and speech recognition.



Models of the brain?

- controversial.
- **Supervised** learning is a better model of learning for some systems (e.g. motor learning -- visual feedback) than for others (e.g. development)
- **Back-propagation** of error signals is the most problematic feature;
- Inclusion of **derivative** terms;
- Different authors have proposed more biologically plausible implementations of back-propagation (O Reilly (1996), Roelfsema & Van Ooyen (2005))