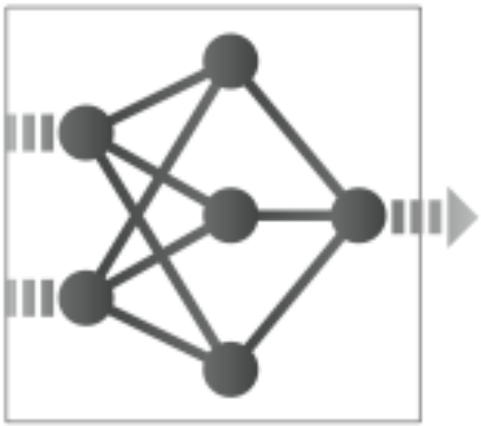
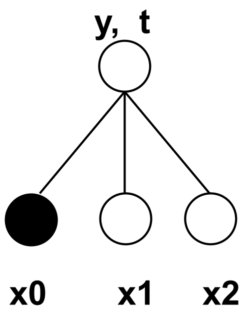
Neural Systems: Part 2



Supervised Learning

- **Teacher** provides desired responses for a set of training patterns
- Synaptic weights are modified in order to reduce the **error** between the output *y* and its desired output *t* (a.k.a. teaching input)

Widrow-Hoff defined the error with the symbol delta: $\delta_i = t_i - y_i$
(a.k.a. delta rule)



linear units

$$w_{ij} = rnd(\pm 0.1)$$ initialize weights to random values

$$y_i = \sum_{j=0} w_{ij} x_j$$ present input pattern and compute neuron output

$$\Delta w_{ij} = \eta (t_i - y_i) x_j$$ compute weight change using difference between desired output and neuron output

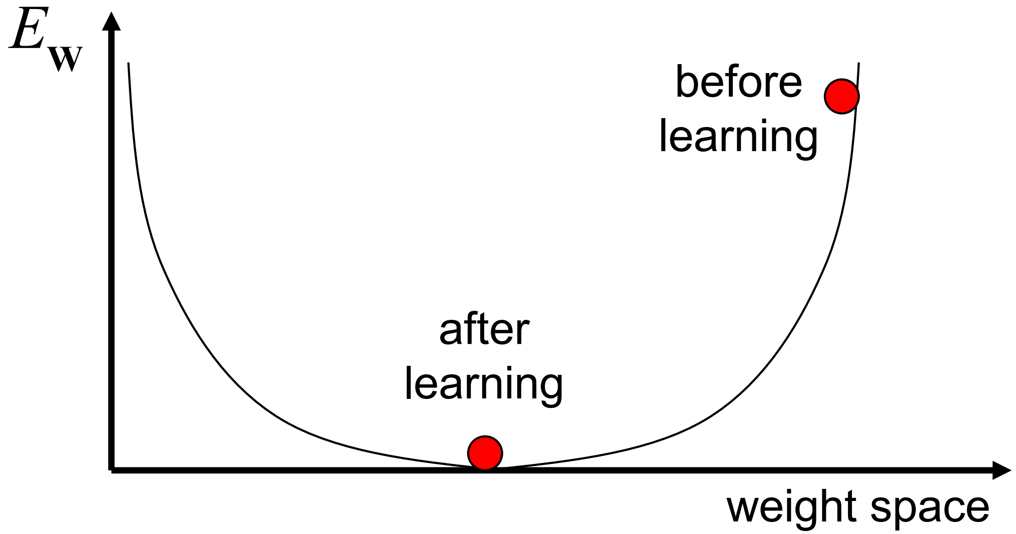
$$w_{ij} = w_{ij}^{t-1} + \Delta w_{ij}$$ get new weights by adding computed change to previous weight values

repeat for every input/output pair until error is 0

Error function

The delta rule modifies the weights to descend the gradient of the error function

$$E_{\mathbf{w}} = \frac{1}{2}\sum_{\mu}\sum_{i}\left(t_i^{\mu} - \sum_{j=0} w_{ij}x_j^{\mu}\right)^2$$

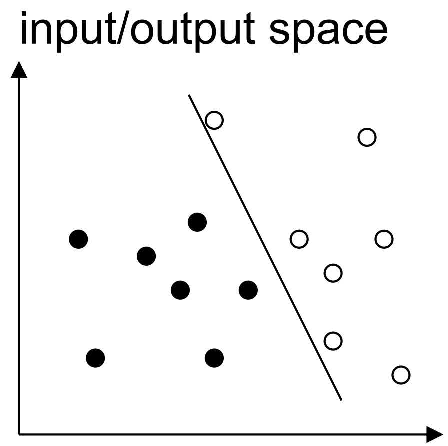


*Error space for a network with a single layer of synaptic weights (**perceptron**, Rosenblatt, 1962)*

Linear Separability

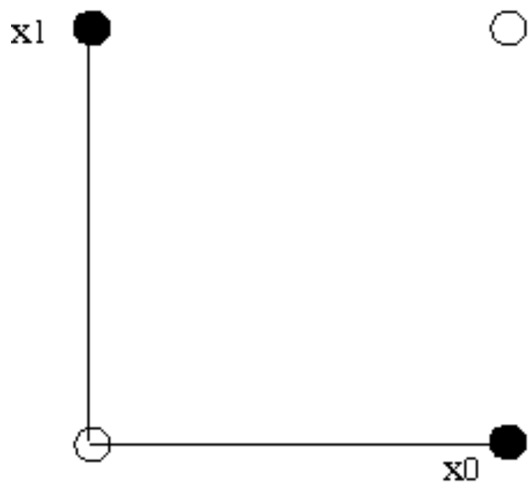
Perceptrons can solve only problems whose input/output space is **linearly separable**.

Several real world problems are not linearly separable.



● class A
○ class B

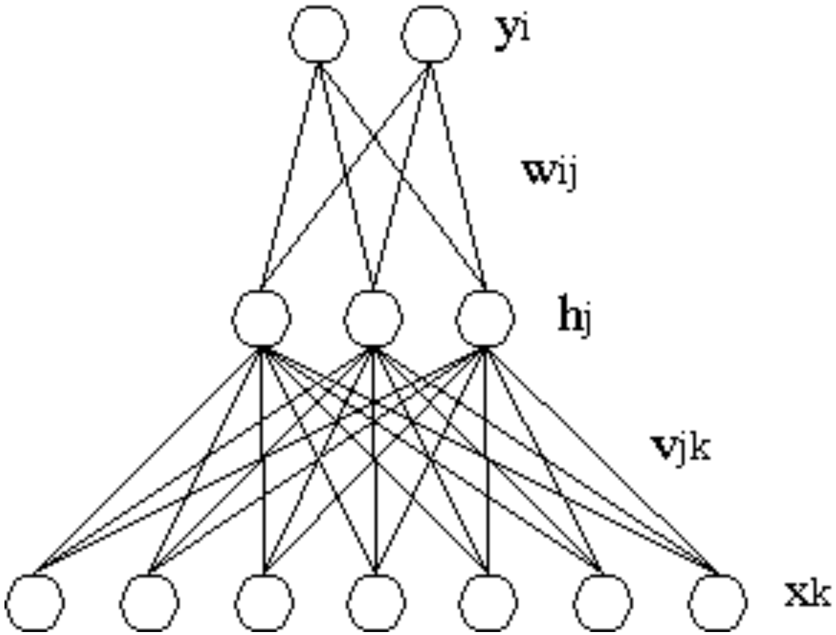
Example of XOR problem



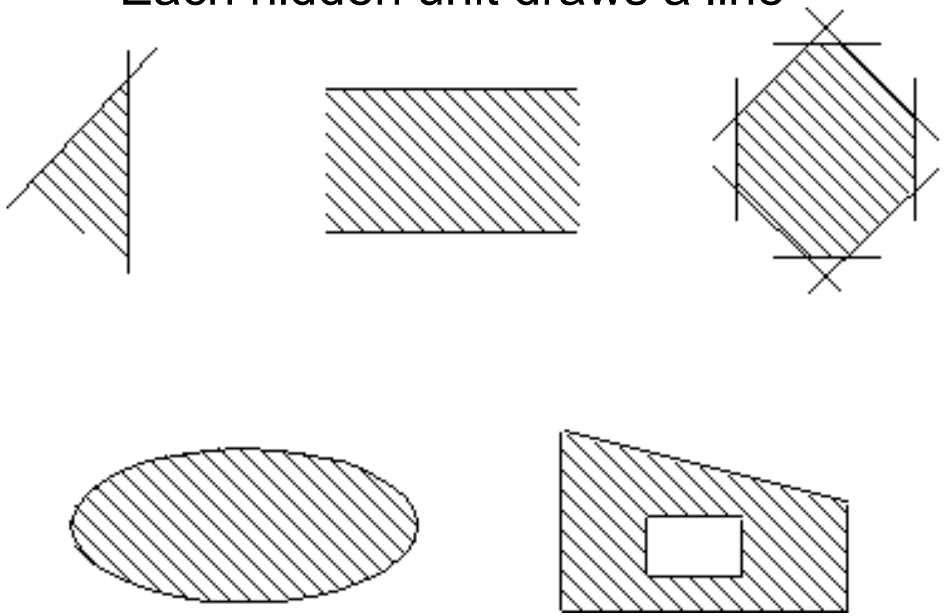
x0	x1	t	
0	0	0	○
1	1	0	○
1	0	1	●
0	1	1	●

Multi-layer Perceptron (MLP)

- Multi-layer neural networks can solve problems that are not linearly separable
- Hidden units re-map input space into a space which can be linearly separated by output units.



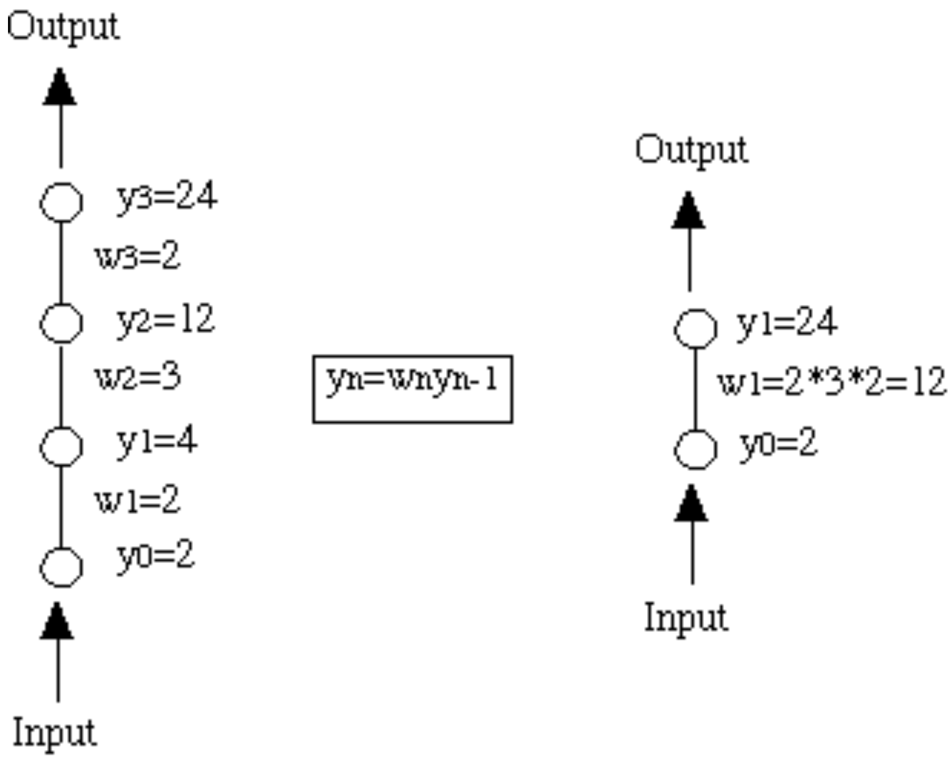
Each hidden unit draws a line



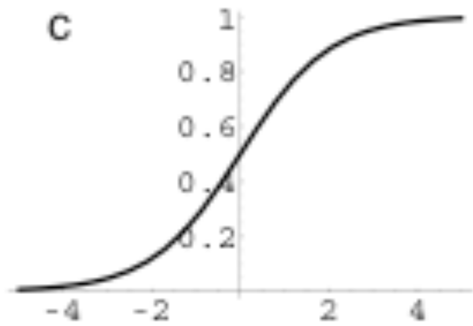
Output units “look” at regions (in/out)

Output Function in MLP

- Multi-layer networks should not use linear output functions because a linear transformation of a linear transformation remains a linear transformation.
- Therefore, such a network would be equivalent to a network with a single layer



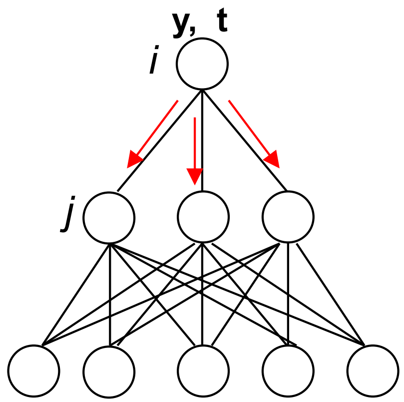
Sigmoid function is often used in MLP



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

Back-propagation of Error

In a simple perceptron, it is easy to change the weights to minimize the error between output of the network and desired output.



$$\delta_i = t_i - y_i$$

$$\Delta w_{ij} = \eta \delta_i x_j$$

$$\delta_i = (t_i - y_i)\dot{\Phi}(A_i)$$

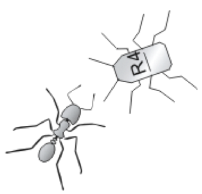
in the case of non-linear output functions, add derivative of output

In a multilayer network, what is the error of the hidden units? This information is needed to change the weights between input units and hidden units.

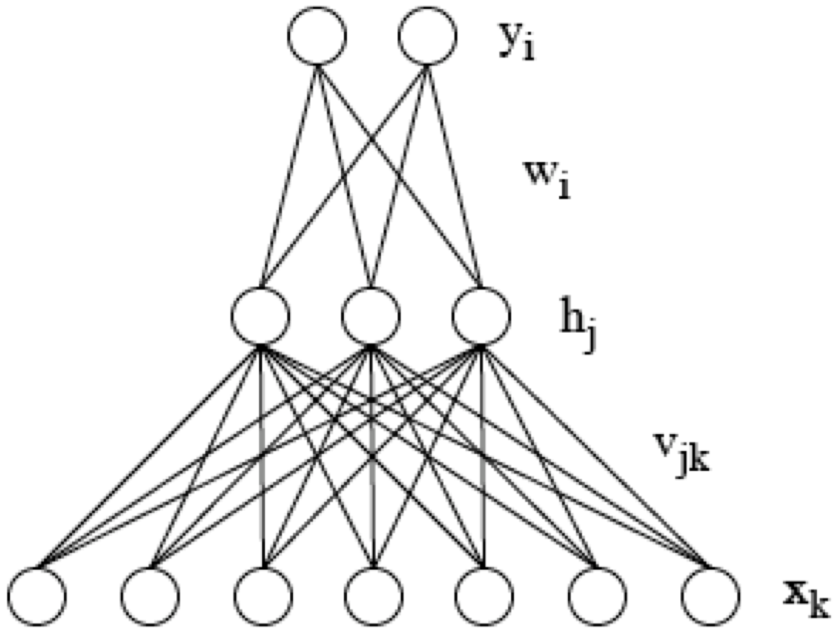
The idea suggested by Rumelhart et al. in 1986 is to propagate the error of the output units backward to the hidden units through the connection weights:

$$\delta_j = \dot{\Phi}(A_j)\sum_i w_{ij}\delta_i$$

Once we have the error for the hidden units, we can change the lower layer of connection weights with the same formula used for the upper layer.



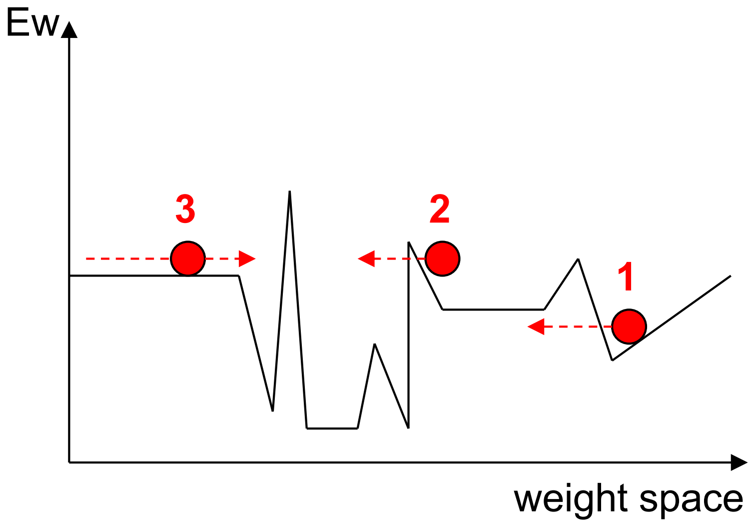
Algorithm



1. Initialize weights (random, around 0)
2. Present pattern $x_k^\mu = s_k^\mu$
3. Compute hidden $h_j^\mu = \Phi\left(\sum_k v_{jk} x_k^\mu\right)$
4. Compute output $y_i^\mu = \Phi\left(\sum_j w_{ij} h_j^\mu\right)$
5. Compute delta output $\delta_i^\mu = \dot{\Phi}\left(\sum_j w_{ij} h_j^\mu\right)\left(t_i^\mu - y_i^\mu\right)$ $\qquad \delta_i^\mu = y_i^\mu\left(1 - y_i^\mu\right)\left(t_i^\mu - y_i^\mu\right)$
6. Compute delta hidden $\delta_j^\mu = h_j^\mu\left(1 - h_j^\mu\right)\sum_i w_{ij}\delta_i^\mu$
7. Compute weight change $\Delta w_{ij}^\mu = \delta_i^\mu h_j^\mu, \quad \Delta v_{jk}^\mu = \delta_j^\mu x_k^\mu$
8. Update weights $w_{ij}^t = w_{ij}^{t-1} + \eta \Delta w_{ij}^\mu, \quad v_{jk}^t = v_{jk}^{t-1} + \eta \Delta v_{jk}^\mu$

Using Back-Propagation

Error space can be complex in multilayer networks: local minima and flat areas

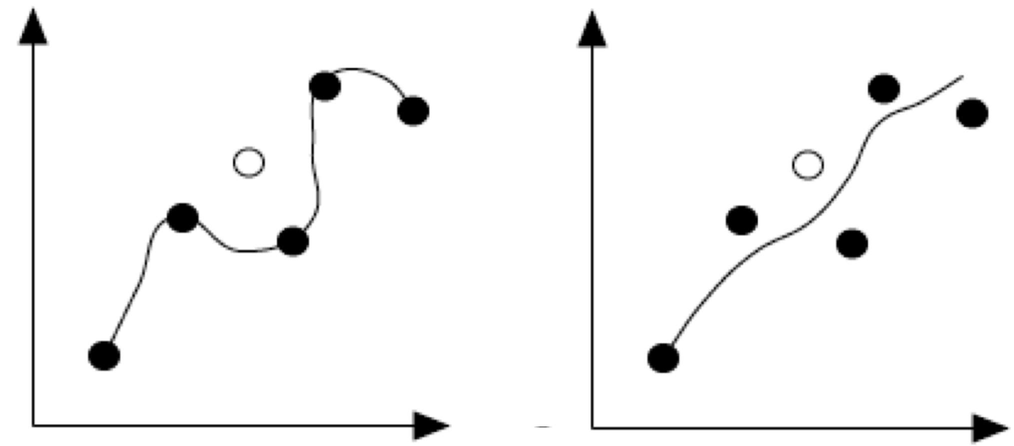


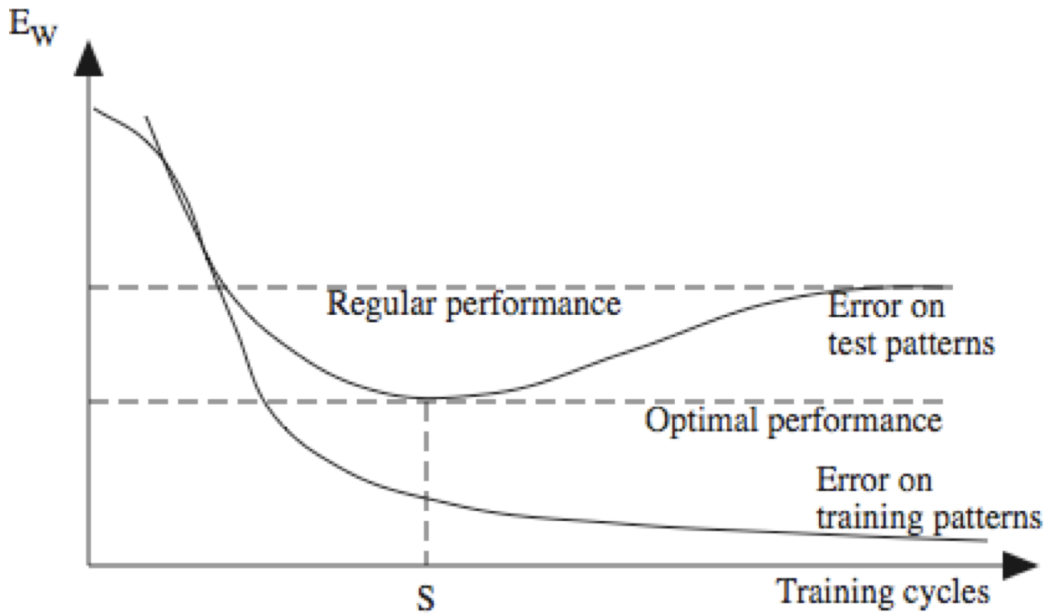
1. Large learning rate: take large steps in the direction of the gradient descent

2. Momentum: add direction component from last update $\Delta w_{ij}^{t} = \eta \delta_i + \alpha \Delta w_{ij}^{t-1}$

3. Additive constant: keep moving when no gradient $\delta_i^{\mu} = \left(\dot{\Phi} + k\right)\left(t_i^{\mu} - y_i^{\mu}\right)$

Over-fitting





Overfitting training data leads to poor generalisation

Overfitting can derive from too many weights and/or too long learning of training patterns

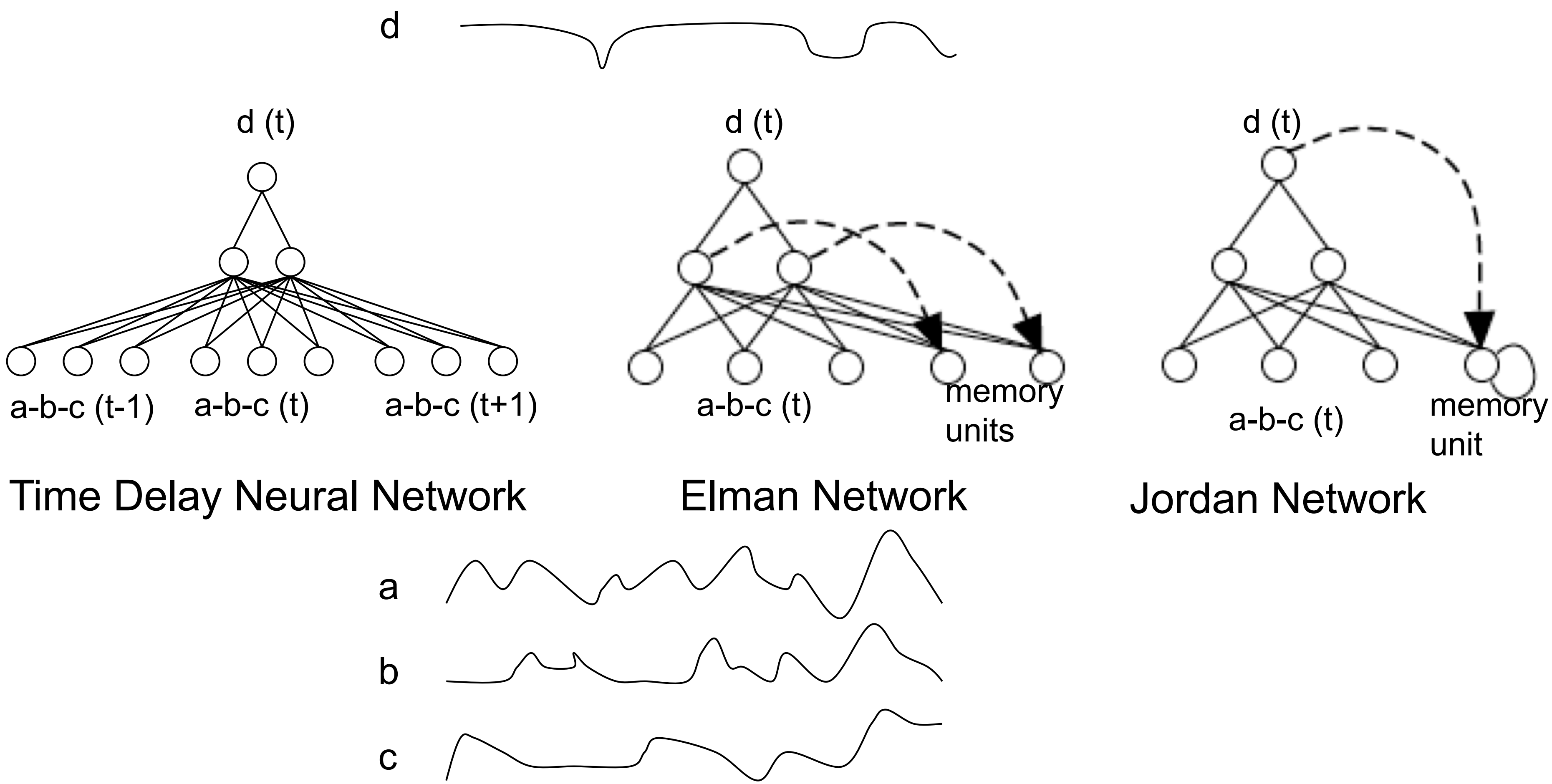
Solution: Use a Validation Set

Divide available data into:
- training set (for weight update)
- validation set (for error monitoring)

Stop training when error for validation set starts growing

Time Series

Extraction of time-dependent features is necessary for time-series analysis



NETtalk

A neural network that learns to read aloud written text:

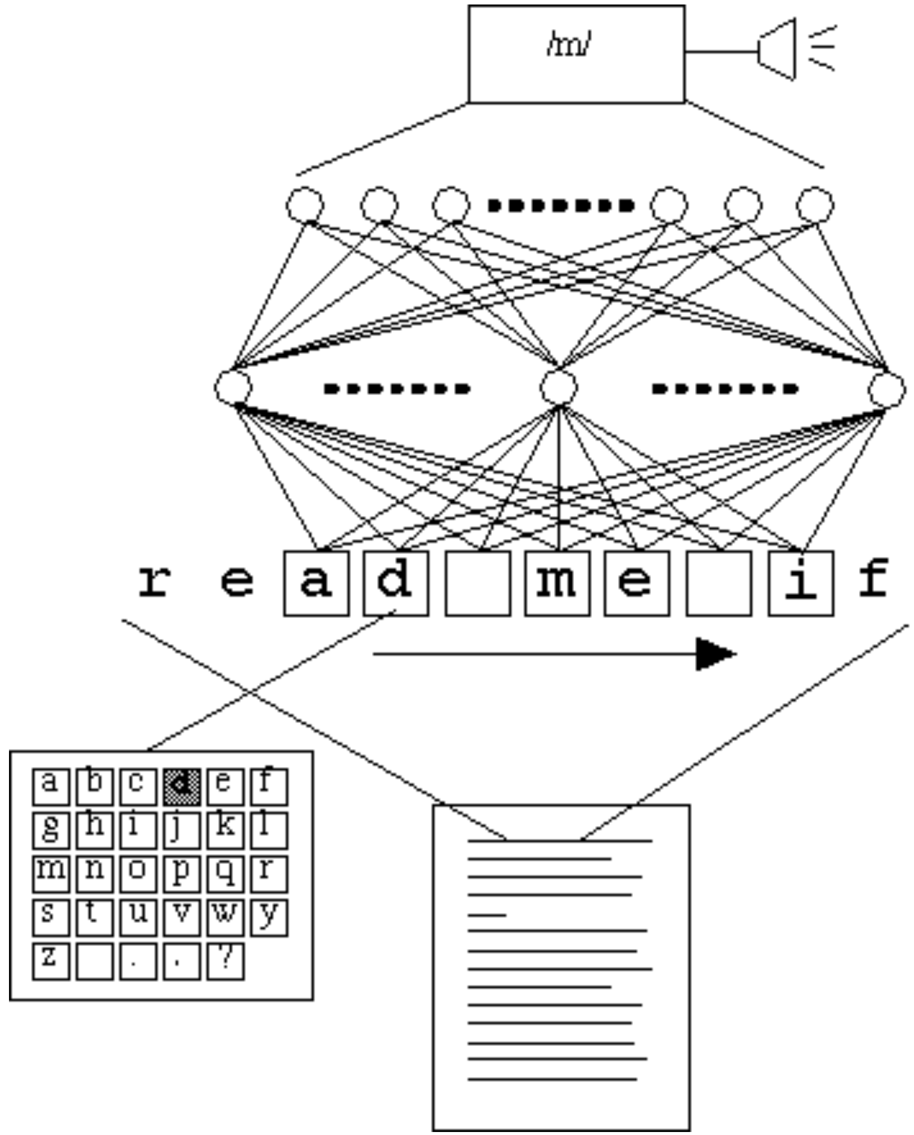
•7 x 29 input units encode characters within a 7-position window(TDNN)

•26 output units encode english phonemes

•approx. 80 hidden units

Training on 1000-word text, reads any text with 95% accuracy

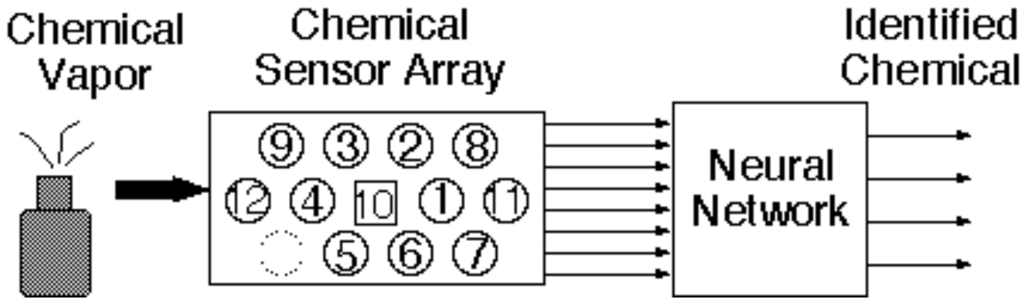
Learns like humans: segmentation, bla-bla, short words, long words



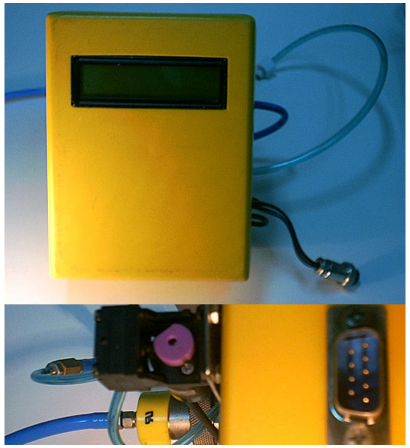
[Sejnowski & Rosenberg, 1987]

Artificial Nose

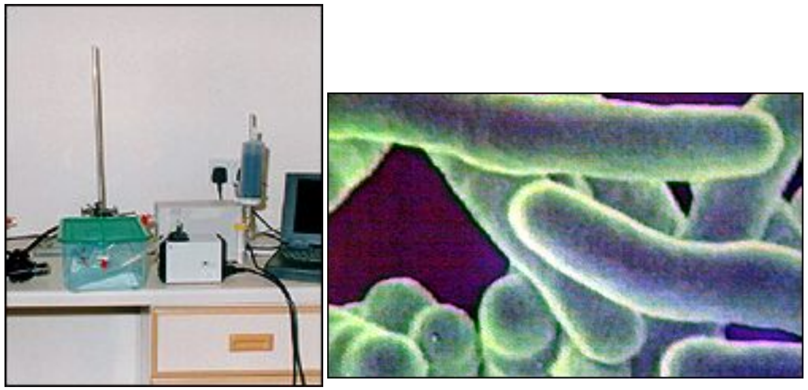
The human brain recognizes millions of smell types by combining responses of only 10,000 receptors. Smell detection is a multi-billion industry (food, cosmetics, medicine, environment monitoring...). Human detection: costly, fatigue, history, aging, subjective.



landmine detection
Tufts University

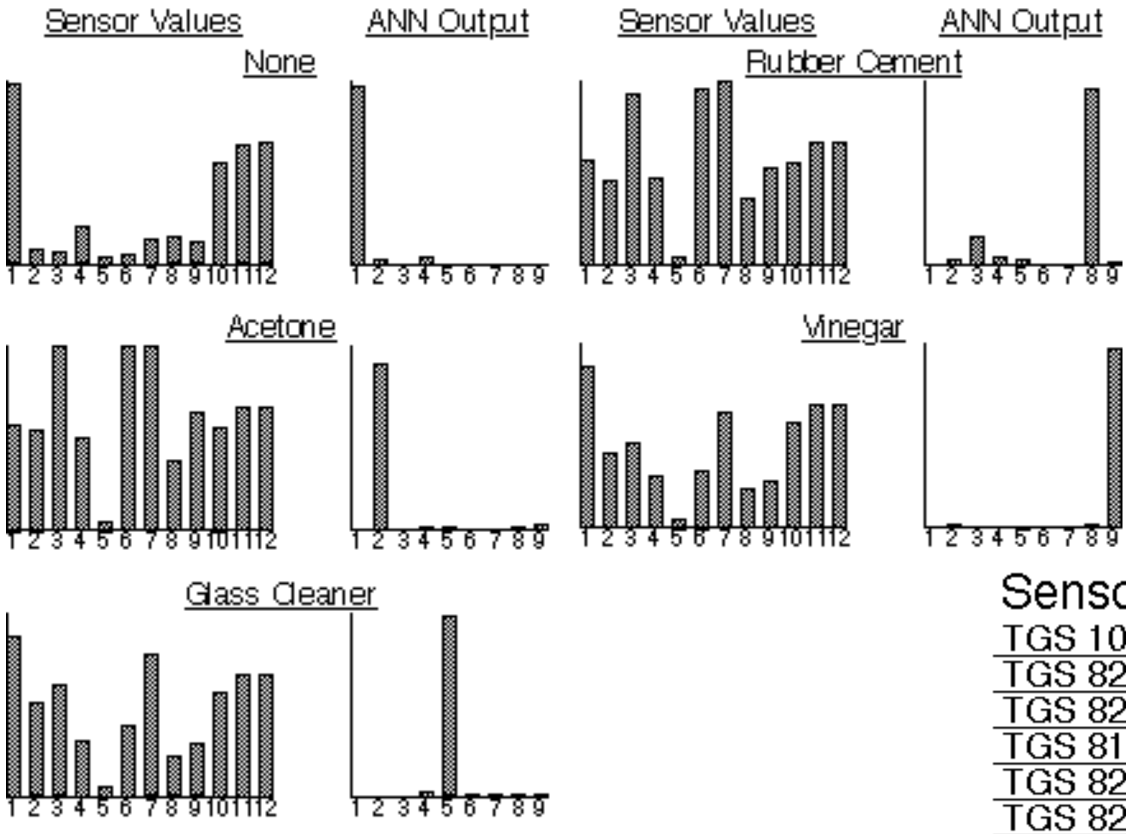


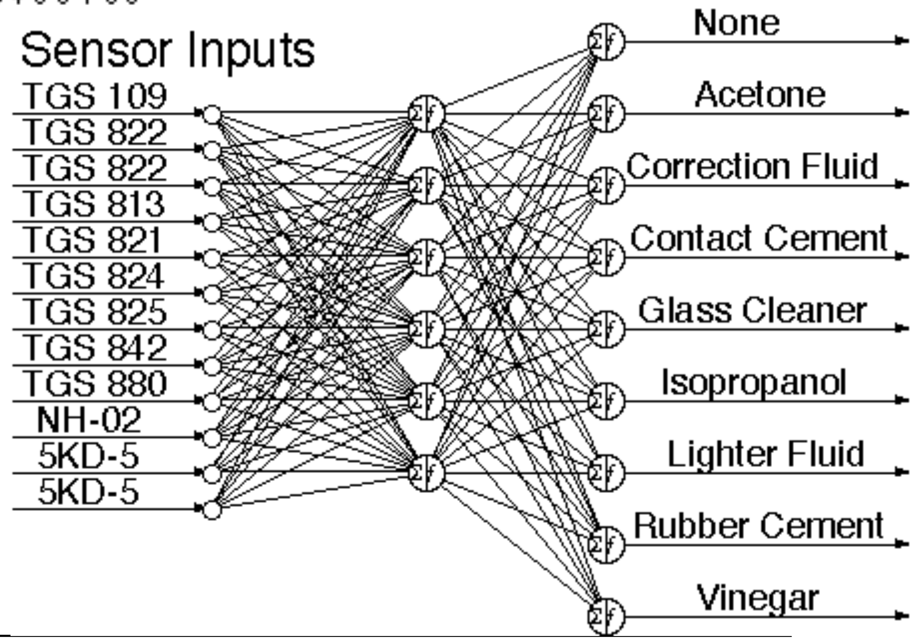
food quality
Pampa Inc.



tubercolosis diagnosis
Cranfield University

Neural Net Recognition



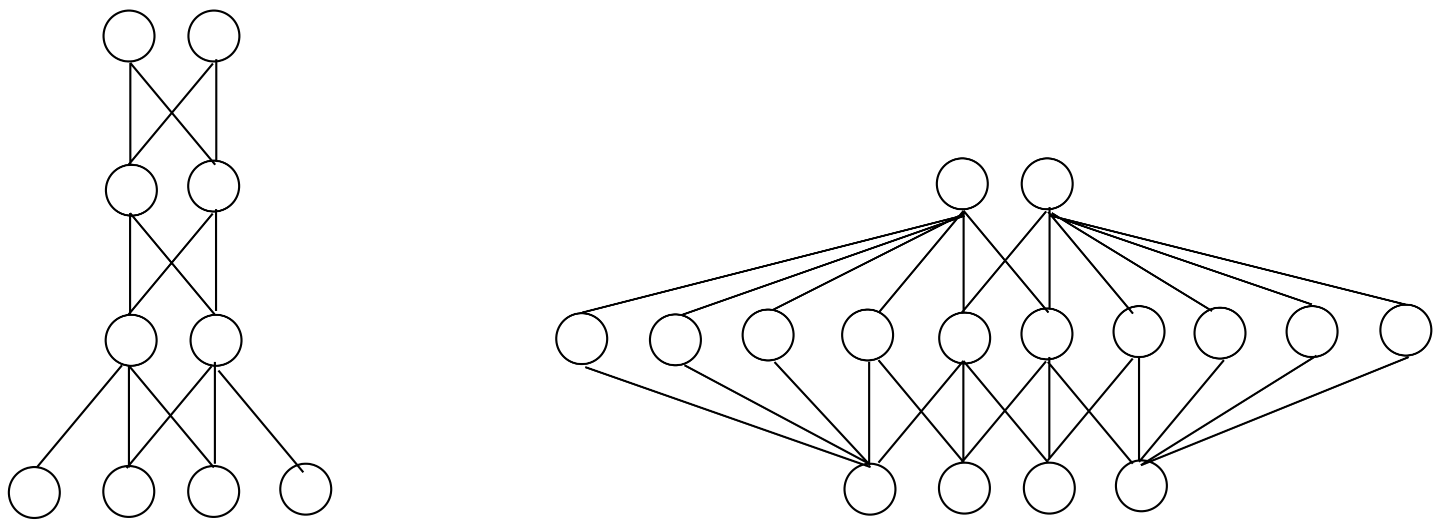


[Keller et al., 1994]

Deep vs. shallow neural networks

Compact distributed encoding (smallest possible number of computing elements) = better generalization -> e.g., feature encoding

Compared to compact network of k layers, a network of k-1 layers requires exponentially larger number of computing elements and therefore has worse generalisation

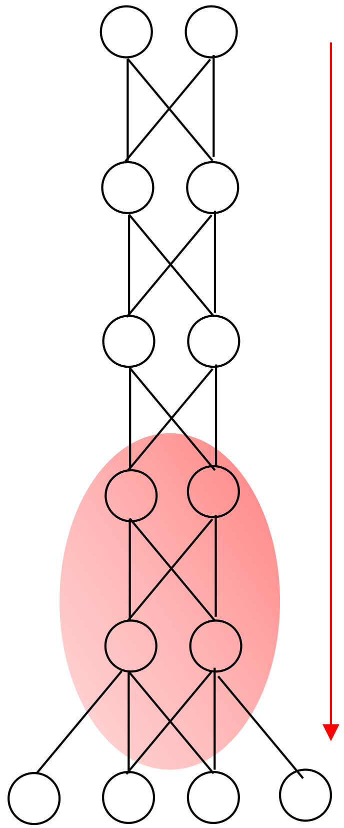


Not all connections are shown

Backpropagation in deep networks

Backpropagation yelds poor results when applied to networks of many layers (k>3)

The problem seems to lie in the poor gradient estimation in the lower layers of the neural network, equivalent to shallow gradients and thus small weight modifications

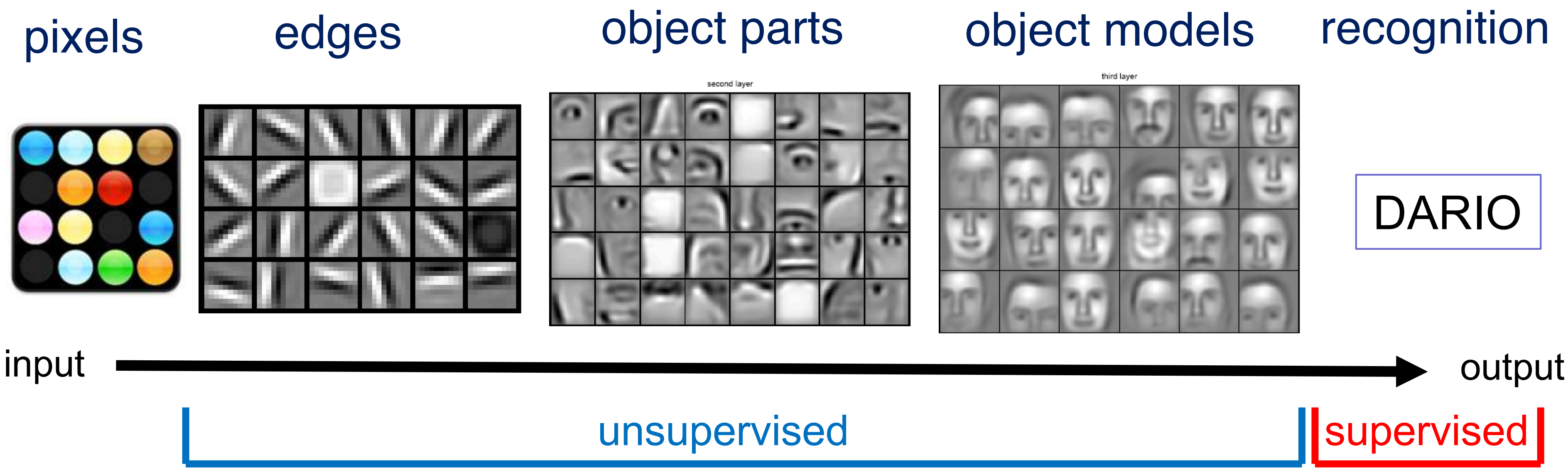


$$\delta_j = \dot{\Phi}(A_j)\sum_i w_{ij}\delta_i$$

Not all connections are shown

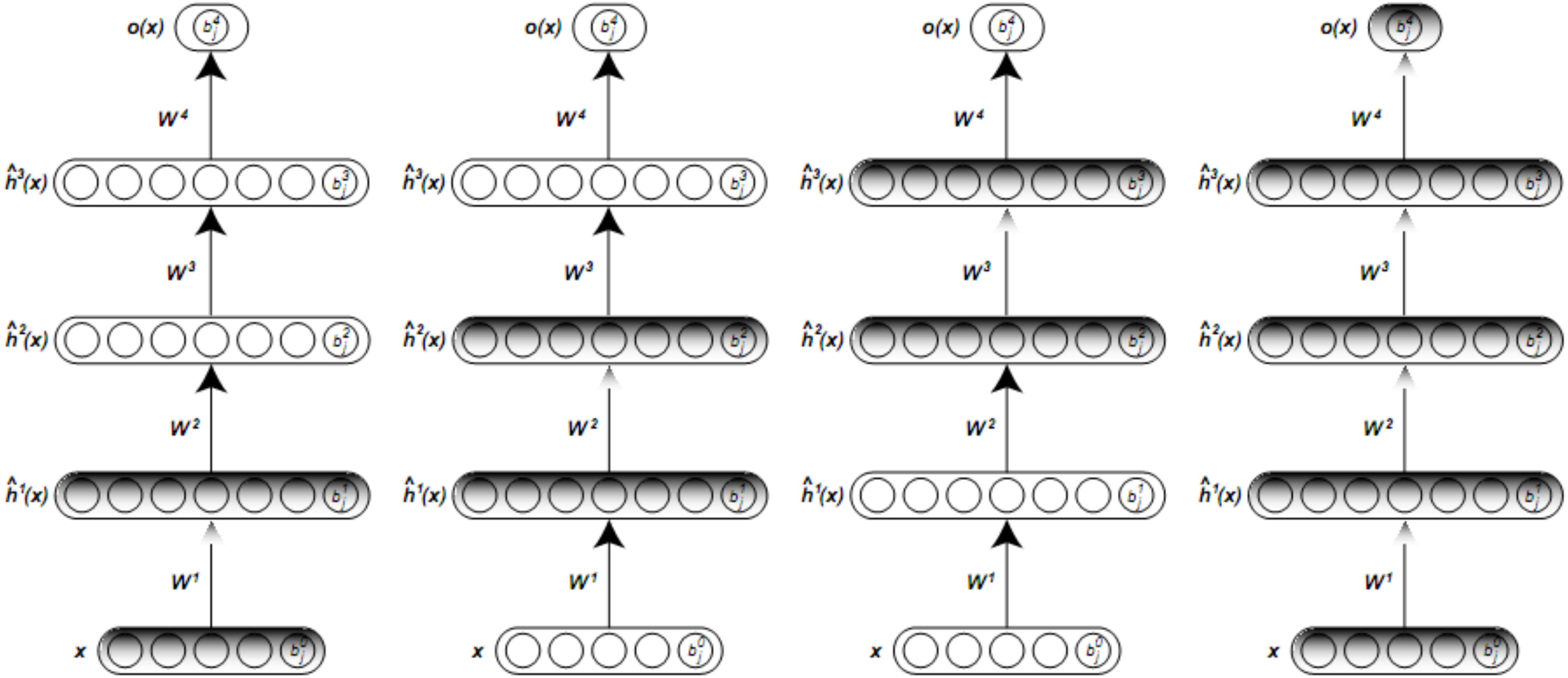
“Deep learning” method (2006)

Unsupervised training of low layers to generate structure of increasingly complex representations + Supervised training of top layer



Hinton, Osindero, Teh, 2006
Bengio, Lamblin, Popovici, Larochelle, 2007
Ranzato, Poultney, Chopra, LeCun, 2007
See online also *Learning Deep Architectures for AI* by Yoshua Bengio, 2008

Layer-wise training procedure

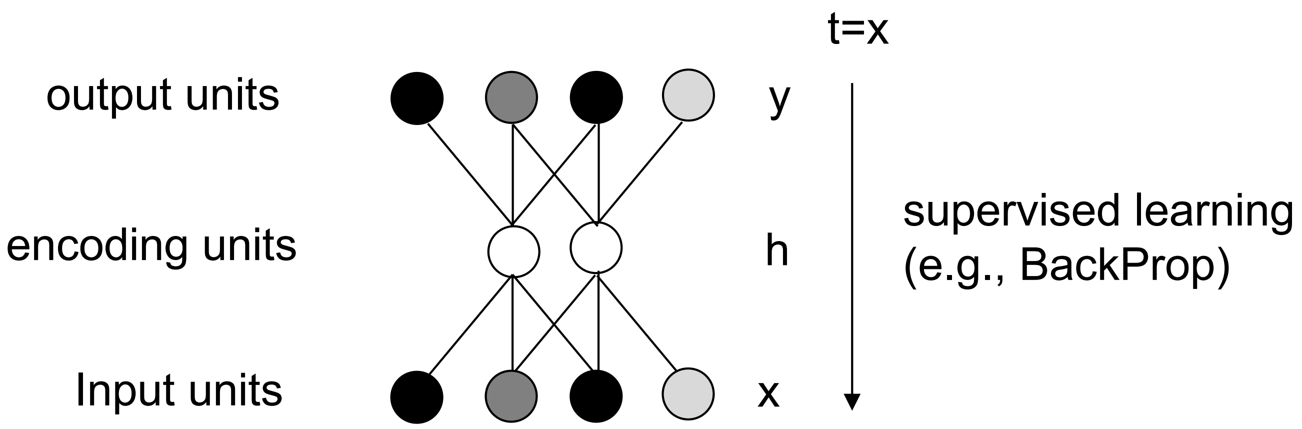


(a) First hidden layer pre-training (b) Second hidden layer pre-training (c) Third hidden layer pre-training (d) Fine-tuning of whole network

What type of unsupervised learning?

PCA are not suitable because 1) they makes sense only when output units are equal or less than input units; 2) PCA is a linear transformation of its input, thus not suitable to multi-layer networks

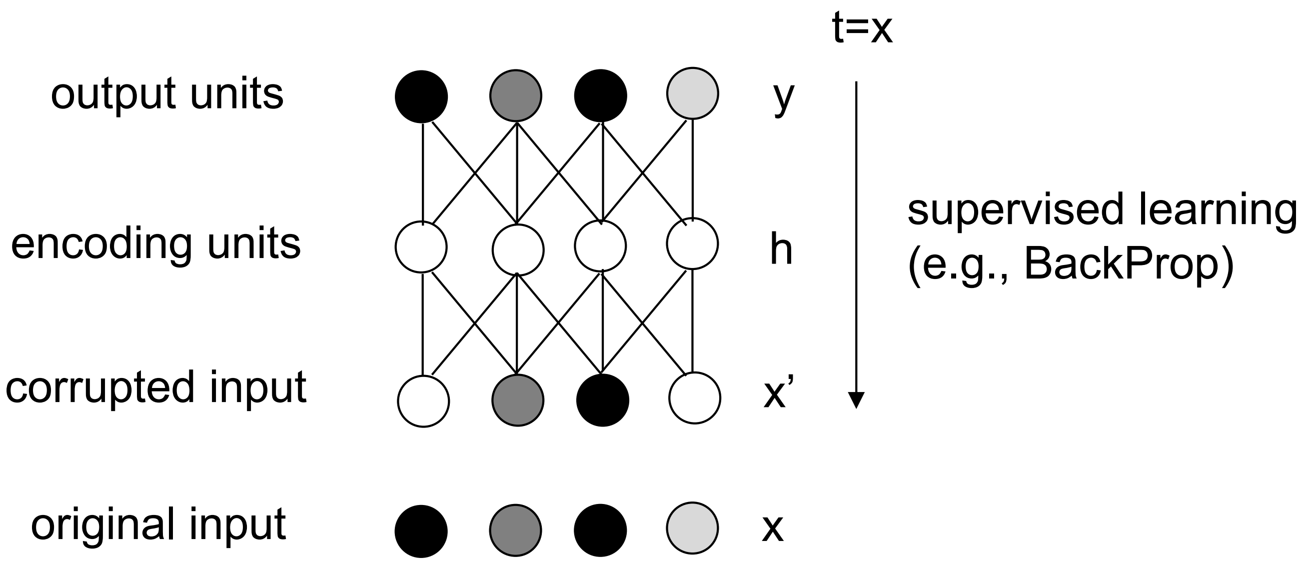
Autoencoders are supervised networks (e.g., Back-prop) that learn to reproduce the input pattern on the output layer. Usually, they have smaller set of hidden units (*encoding units*) to generate a compressed representation, which spans the same space of PCA representation, but can use non-linear units.



Not all connections are shown

Autoencoders in Deep Learning

In deep learning, encoding units should be equal or larger than input units in order to allow for large representation capabilities: however, this may lead to *identity coding problem*



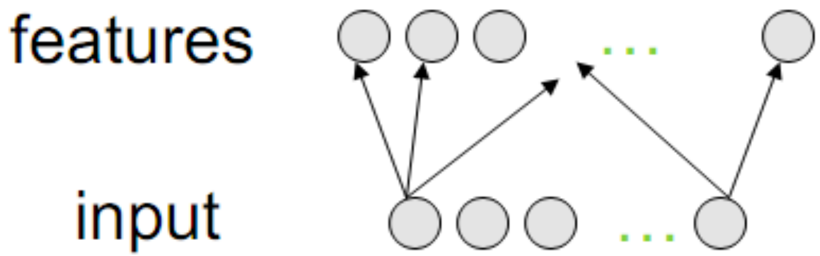
To prevent identity encoding, use *denoising autoencoders* (Vincent et al. 2008): corrupt input by randomly switching off 50% of units while keeping teaching output equal to uncorrupted input

Not all connections are shown

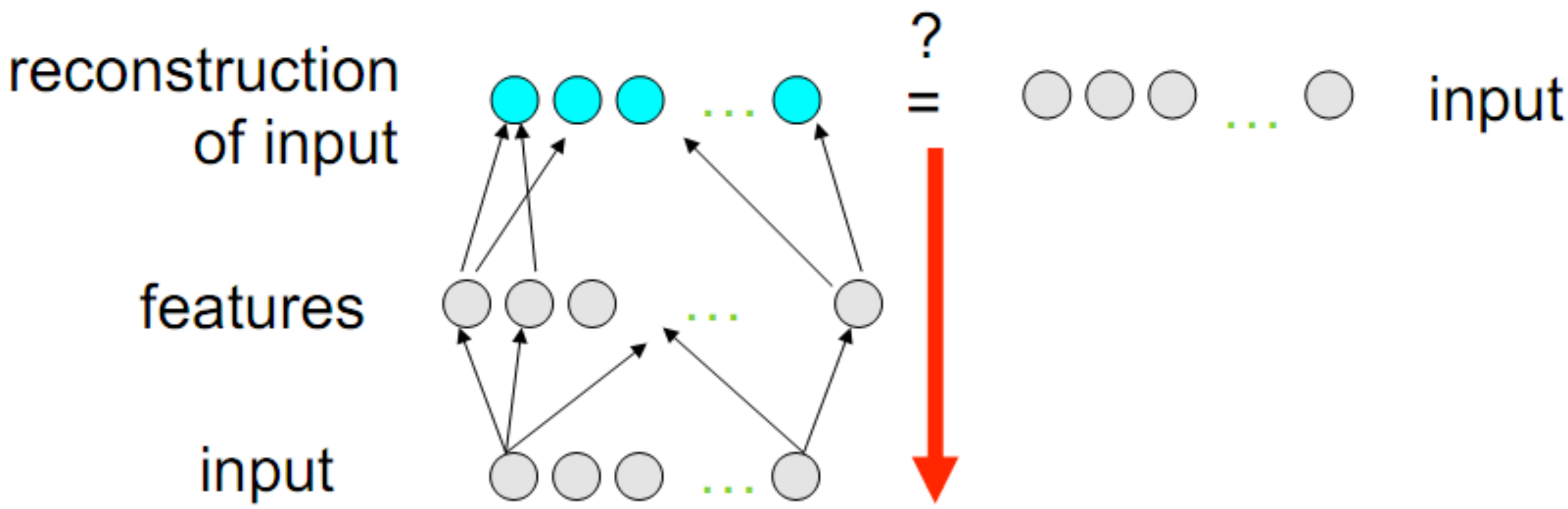
Deep training

input

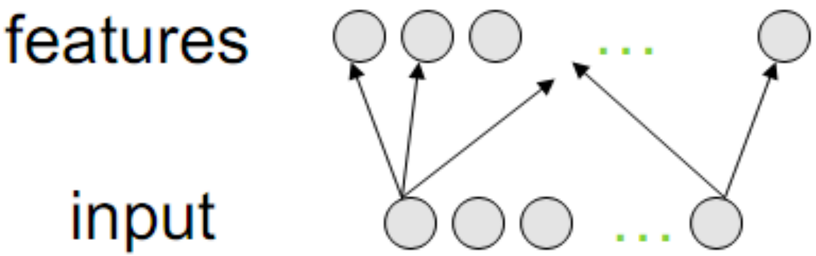
Layer-Wise Unsupervised Pre-training



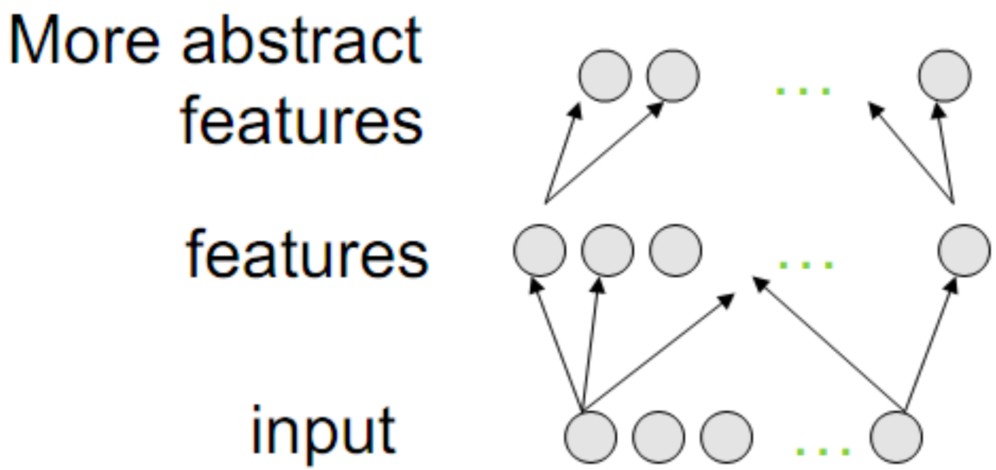
Layer-Wise Unsupervised Pre-training



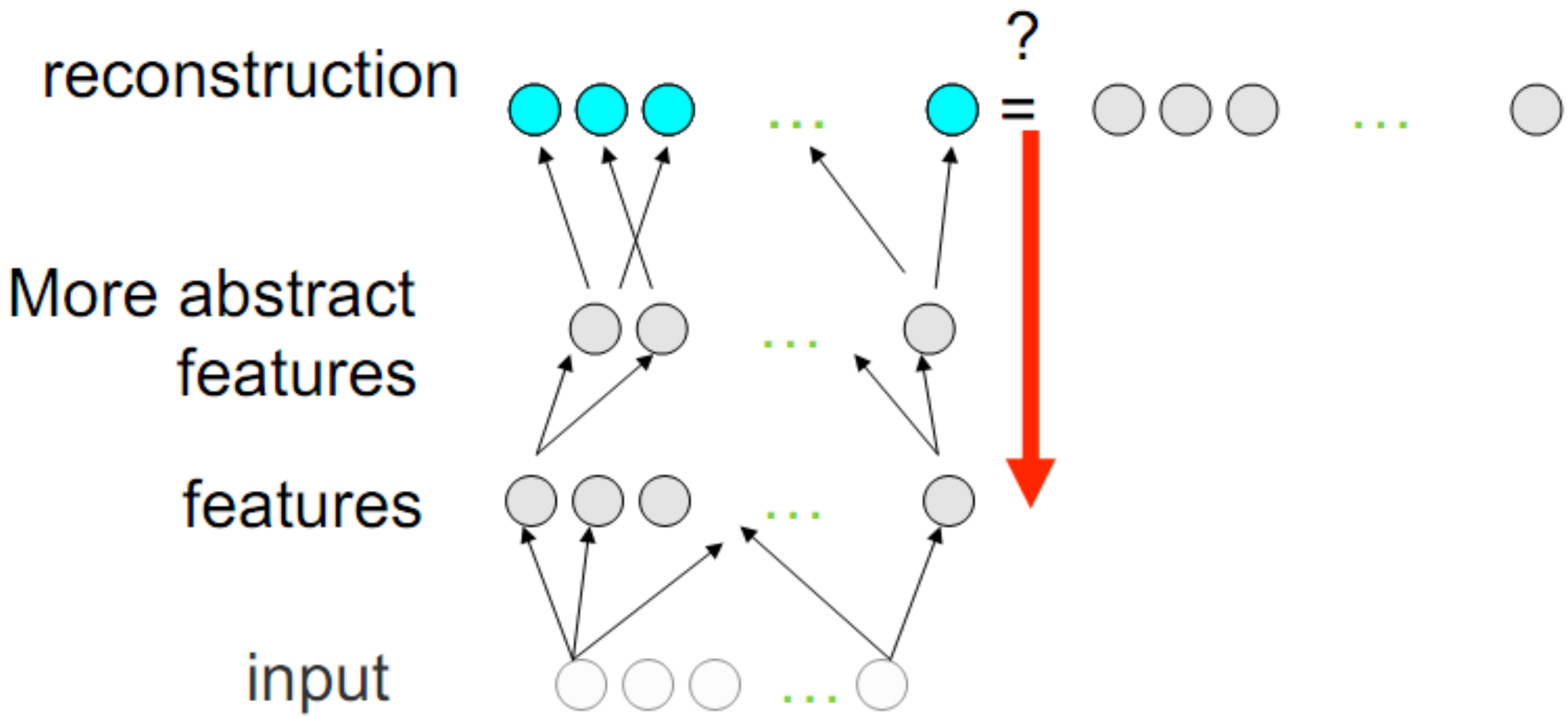
Layer-Wise Unsupervised Pre-training



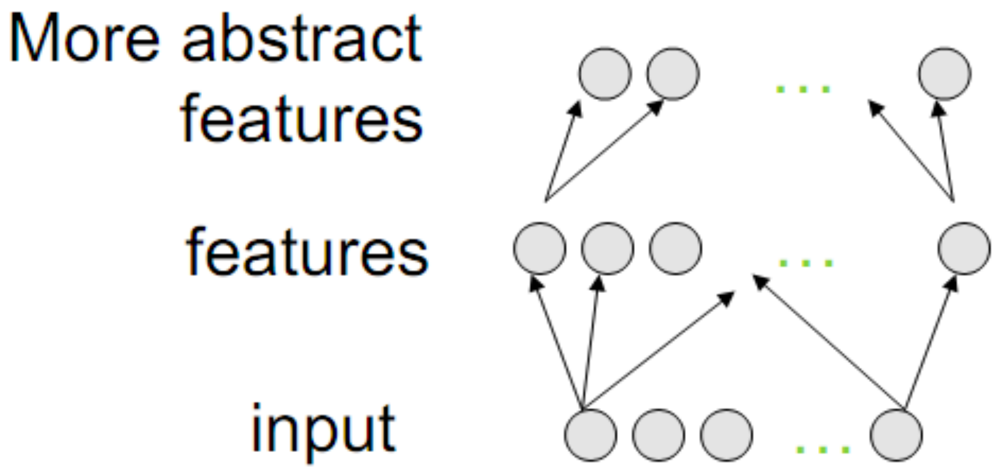
Layer-Wise Unsupervised Pre-training



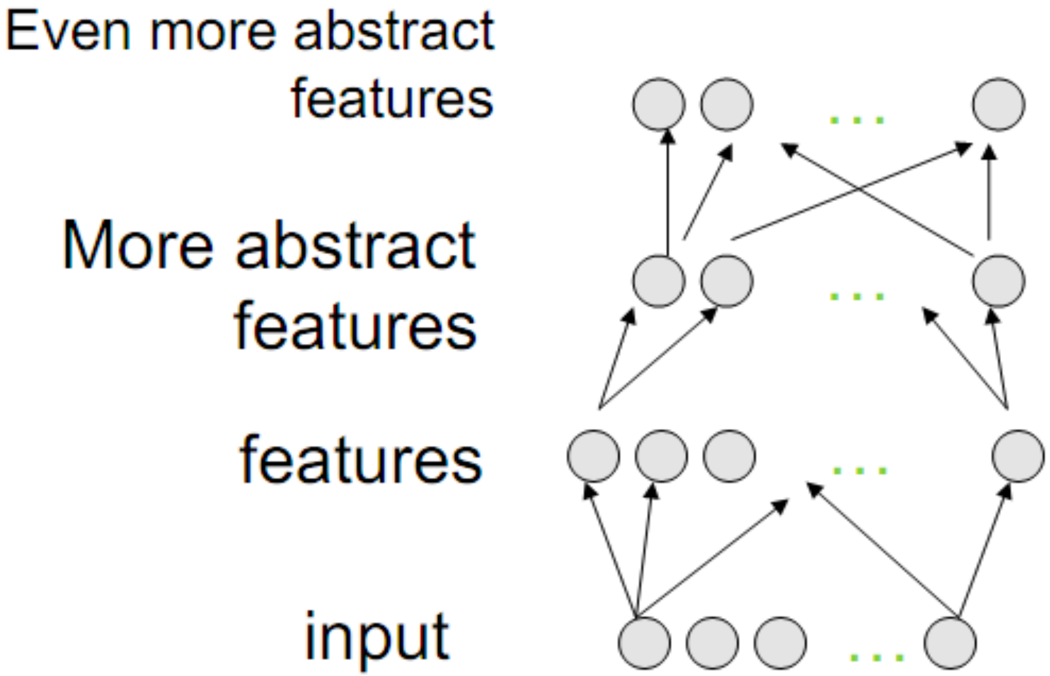
Layer-Wise Unsupervised Pre-training



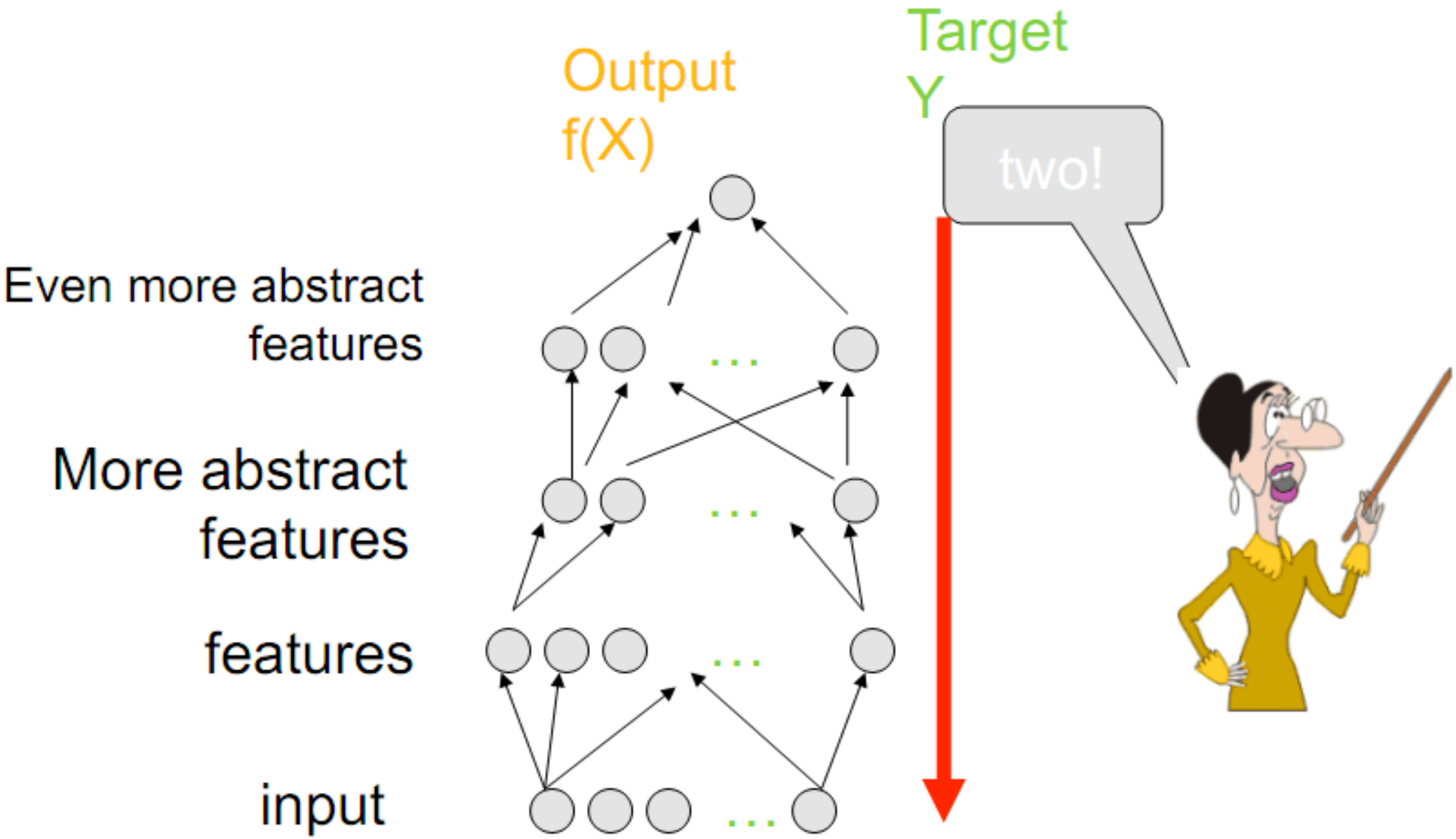
Layer-Wise Unsupervised Pre-training



Layer-Wise Unsupervised Pre-training



Supervised Fine-Tuning

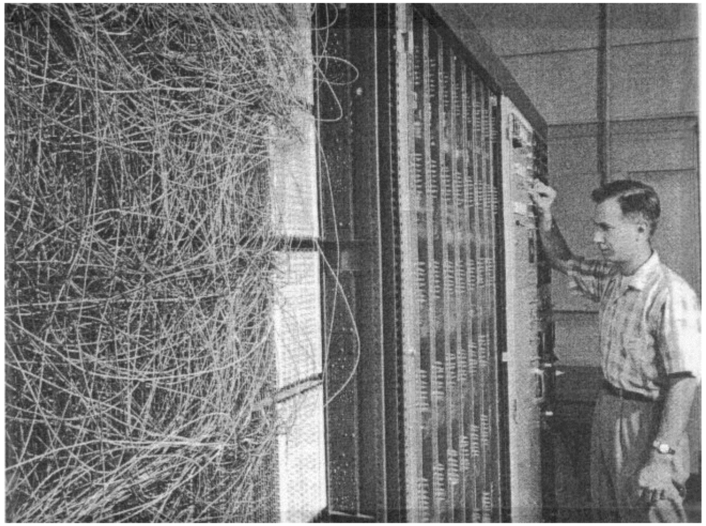


Deepmind Technologies bought by Google for >500 million USD, January 2014

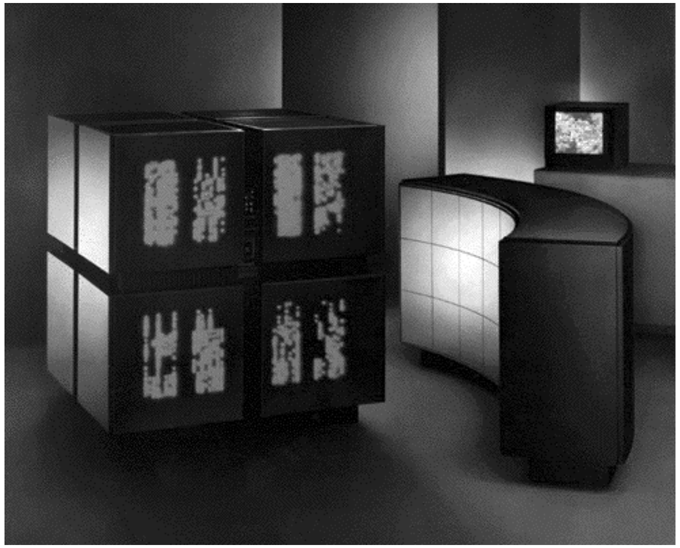
Deep learning networks beat human GO player, January 2016

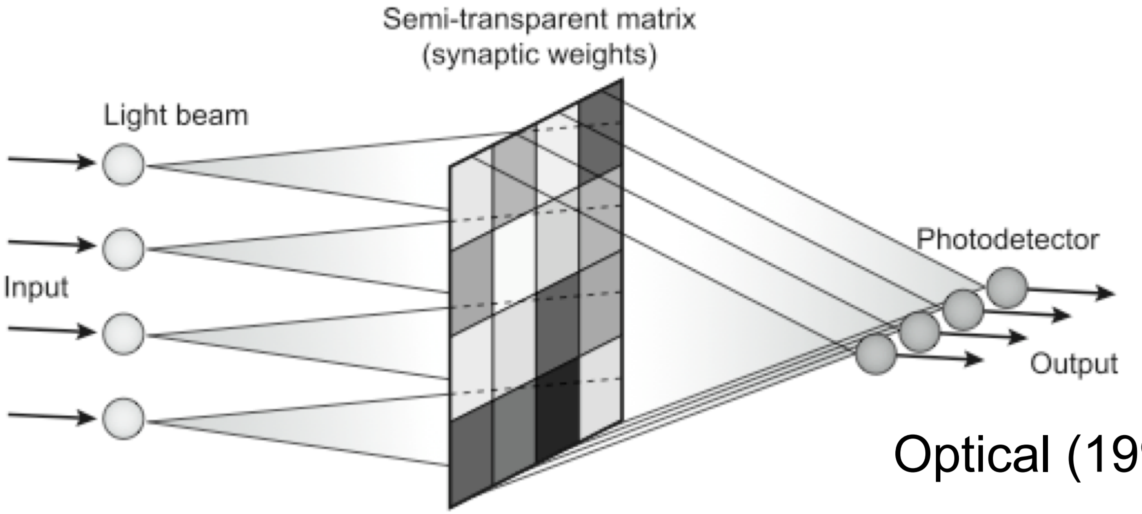
Neural Hardware Implementations

Mark I Perceptron (1960)



Connection Machine (1990)

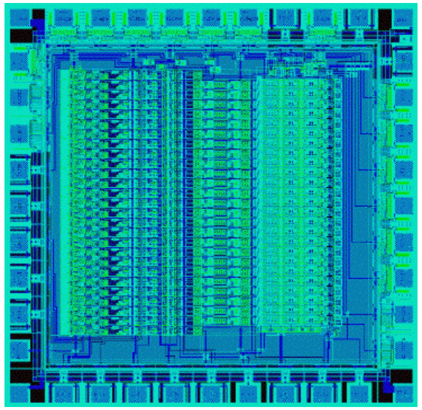




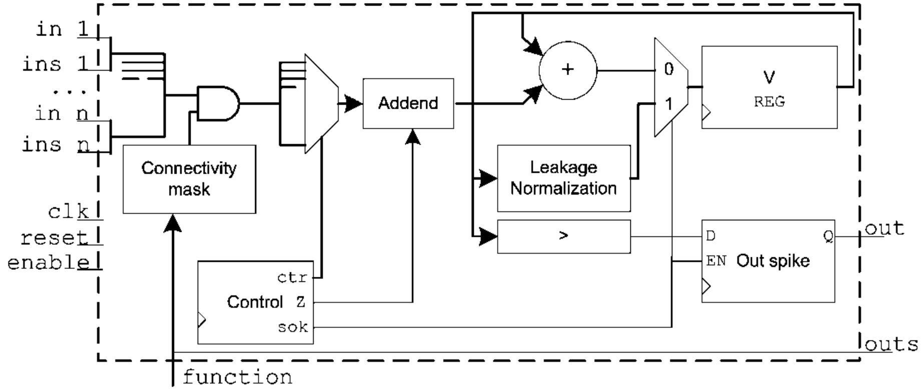
Optical (1990)

Neural Hardware Implementations

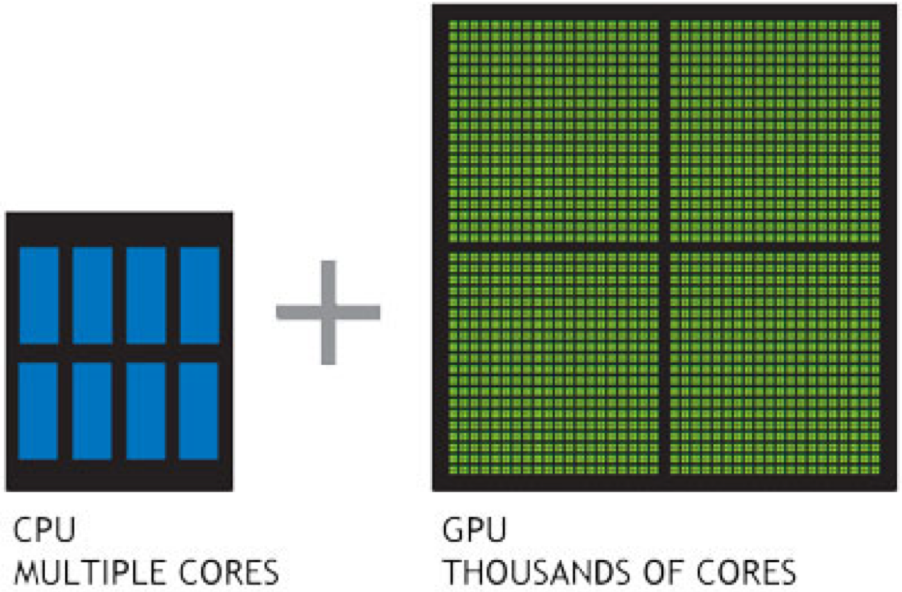
aVLSI (2000)



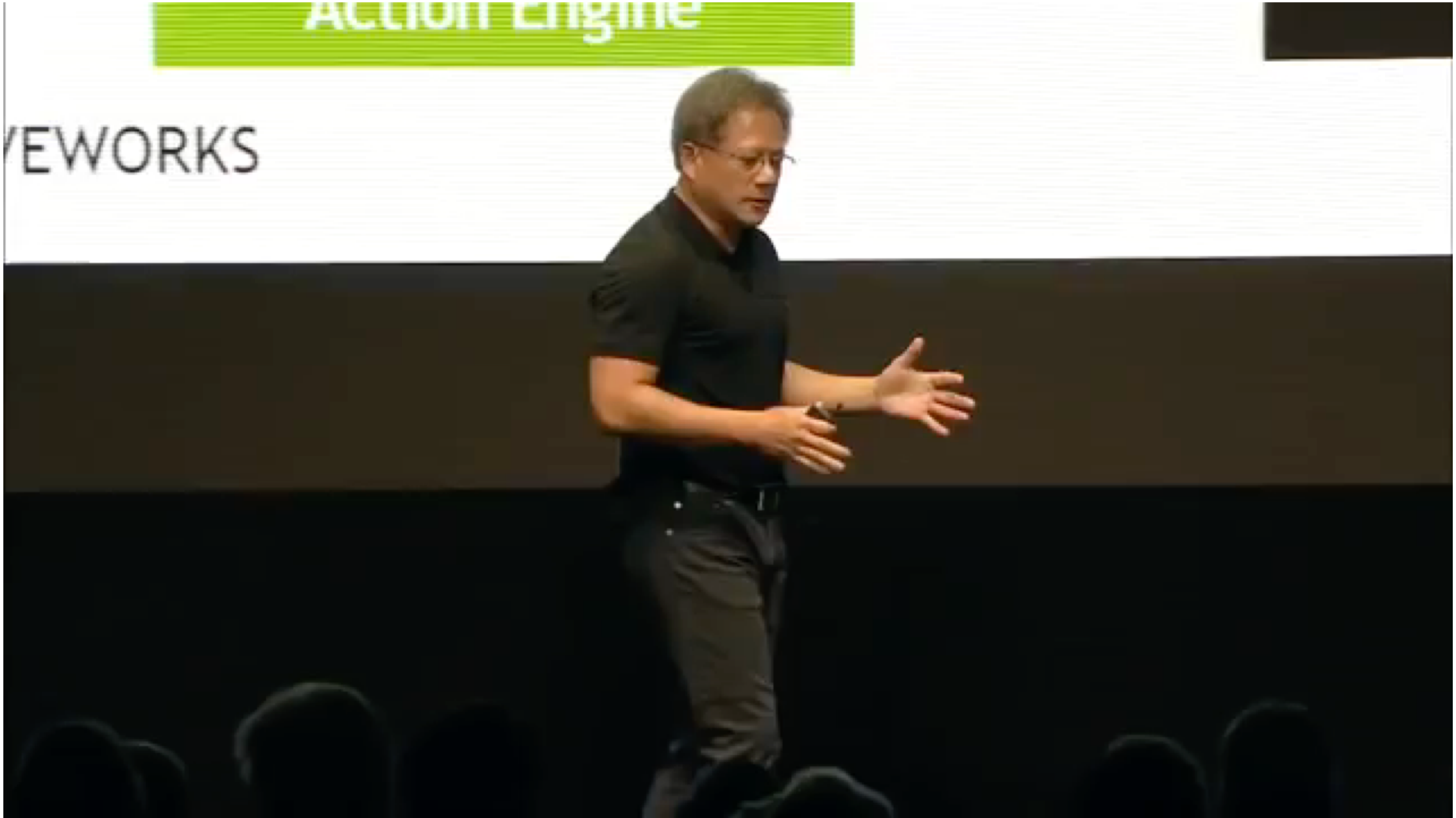
FPGA (2000)



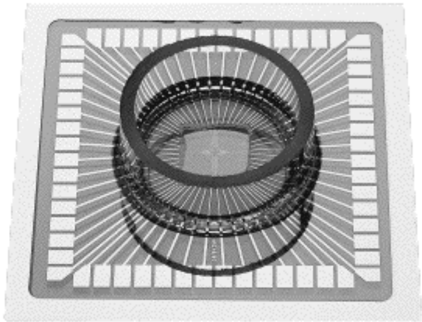
Graphic Processing Units (2007)

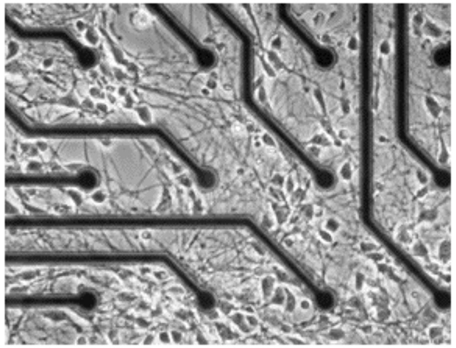


NVIDIA @ CES January 2017



Hybrid Neural Systems: *Multi-Electrode-Array*

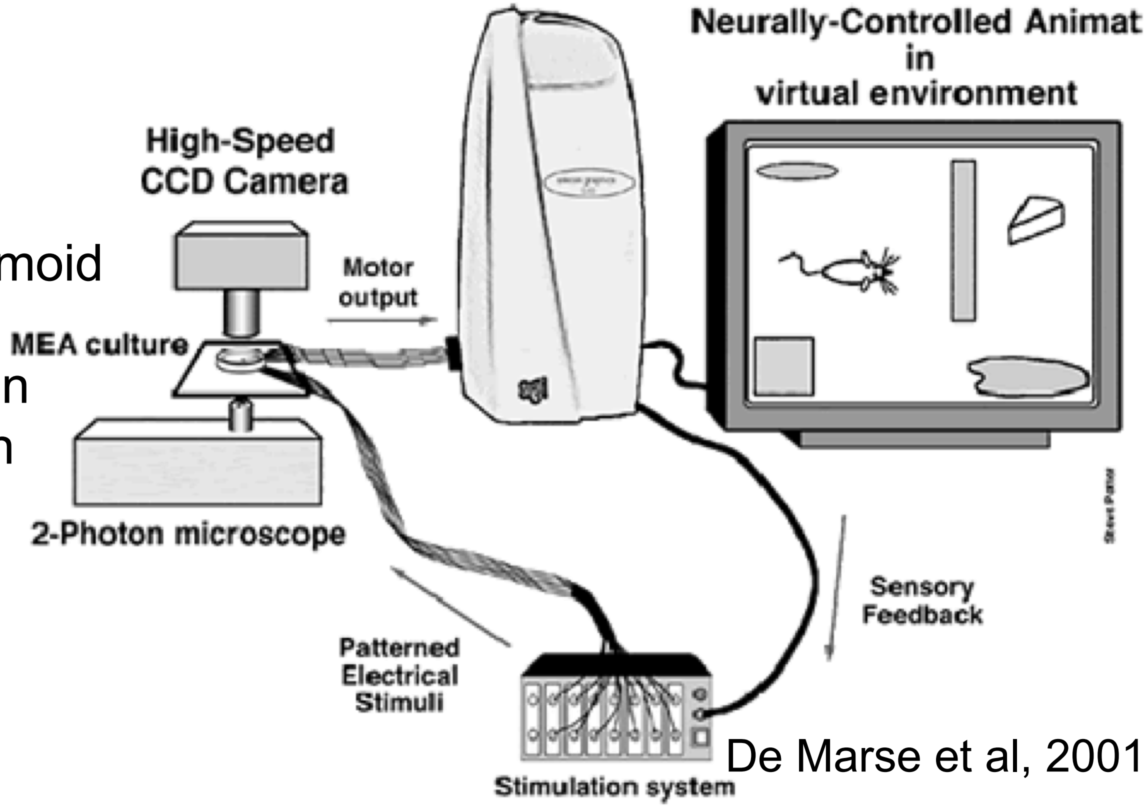




Records/stimulates groups of neurons

Neurons in sealed container (only oxigen and carbon dioxide exchange)
Activity for several months

60 electrodes spaced every 200 μm
Spike count over 200 ms through sigmoid
Cluster patterns of 60 values
Associate each cluster with one action
Agent's sensors to neuron stimulation

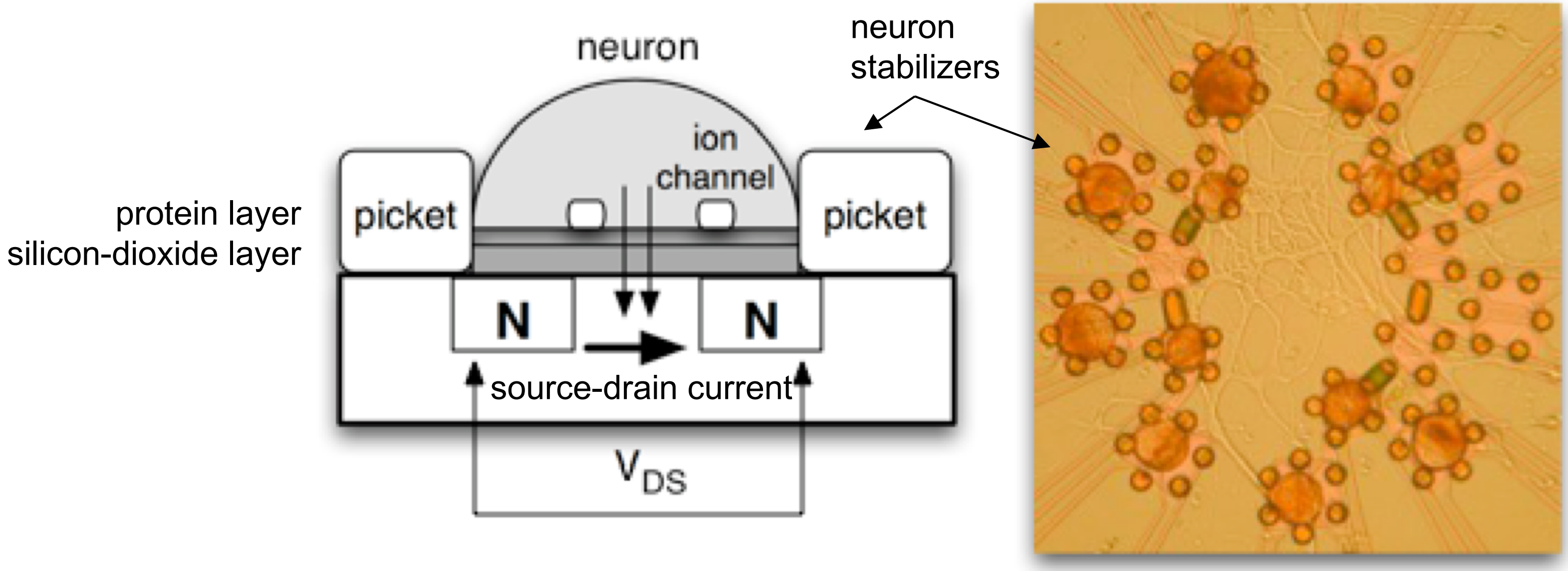


De Marse et al, 2001

Hybrid Neural Systems: *Field-Effect-Transistor*

<u>Records/stimulates single neuron</u>

Monitor biological neural communication
Connect distant neurons by electrical connections
Stimulate neurons and record network activity
Grow biological networks
Interface with artificial networks



Fromherz, 2003