# CS536: Machine Learning

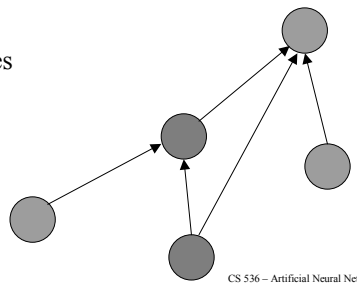
# Artificial Neural Networks

Fall 2005
Ahmed Elgammal
Dept of Computer Science
Rutgers University

## Neural Networks

Biological Motivation: Brain

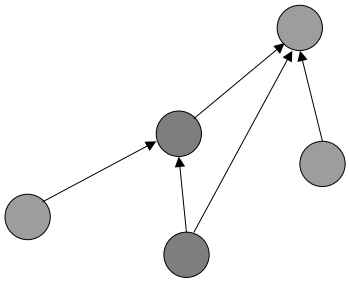
- Networks of processing units (neurons) with connections (synapses) between them
- Large number of neurons: $10^{11}$
- Large connectitivity: each connected to, on average, $10^4$ others
- Switching time $10^{-3}$ second
- Parallel processing
- Distributed computation/memory
- Processing is done by neurons and the memory is in the synapses
- Robust to noise, failures



## Neural Networks

Characteristic of Biological Computation

- Massive Parallelism
- Locality of Computation
- Adaptive (Self Organizing)
- Representation is Distributed





## Understanding the Brain

- Levels of analysis (Marr, 1982)
  1. Computational theory
  2. Representation and algorithm
  3. Hardware implementation
- Reverse engineering: From hardware to theory
- Parallel processing: SIMD vs MIMD
  Neural net: SIMD with modifiable local memory
  Learning: Update by training/experience

- ALVINN system

Pomerleau (1993)

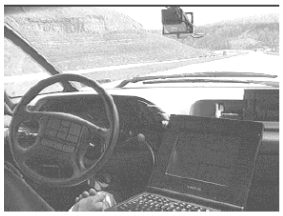
- Many successful examples:

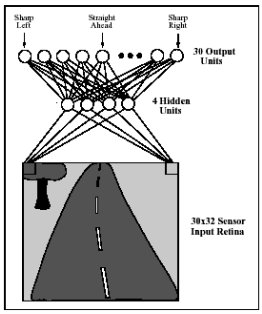
Speech phoneme recognition [Waibel]

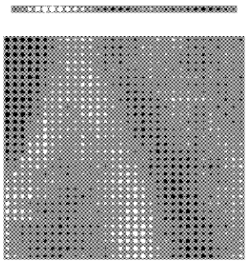
Image classification [Kanade, Baluja, Rowley]

Financial prediction

Backgammon [Tesauro]







## When to use ANN

- Input is high-dimensional discrete or real-valued (e.g. raw sensor input). Inputs can be highly correlated or independent.
- Output is discrete or real valued
- Output is a vector of values
- Possibly noisy data. Data may contain errors
- Form of target function is unknown
- Long training time are acceptable
- Fast evaluation of target function is required
- Human readability of learned target function is unimportant

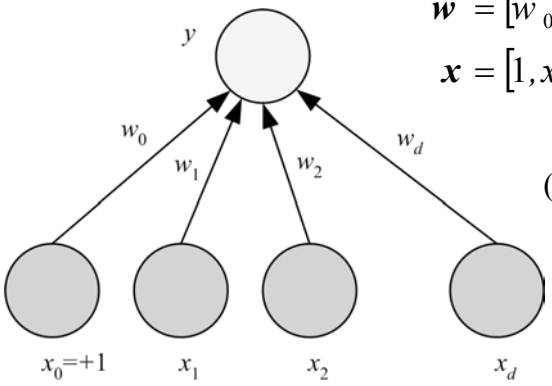
⇒ **ANN is much like a black-box**

## Perceptron

$$y = \sum_{j=1}^{d} w_j x_j + w_0 = \boldsymbol{w}^T \boldsymbol{x}$$

$$\boldsymbol{w} = [w_0, w_1, \ldots, w_d]^T$$

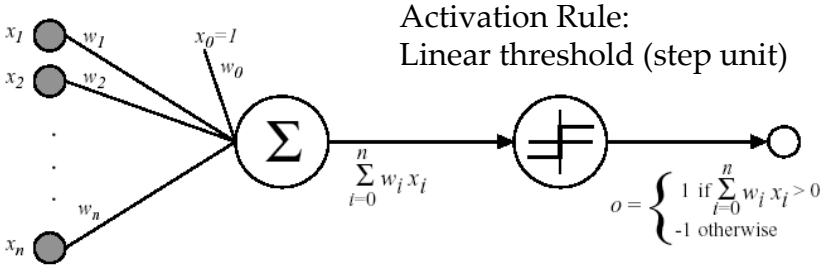
$$\boldsymbol{x} = [1, x_1, \ldots, x_d]^T$$




(Rosenblatt, 1962)



## Perceptron



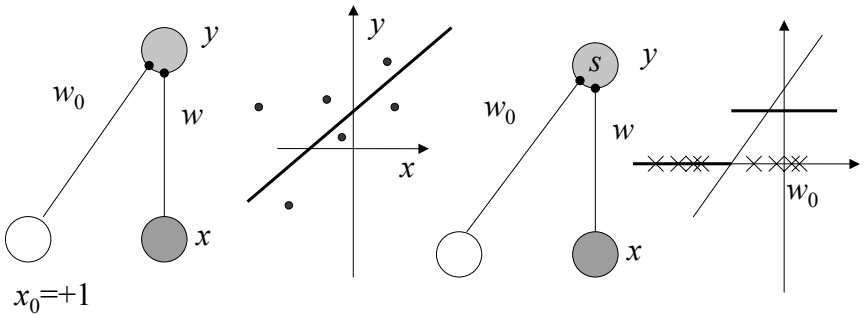

Activation Rule:
Linear threshold (step unit)

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } w_0 + w_1 x_1 + \cdots + w_n x_n > 0 \\ -1 & \text{otherwise.} \end{cases}$$

Or, more succinctly: $o(\boldsymbol{x}) = \text{sgn}(\boldsymbol{w} \cdot \boldsymbol{x})$

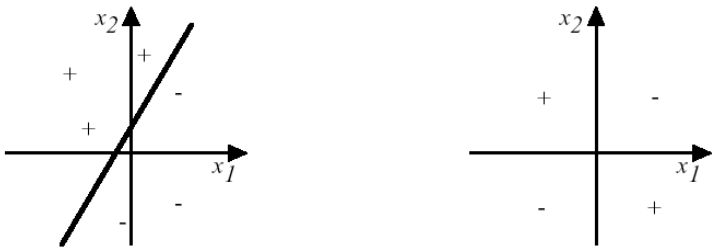
## What a Perceptron Does

- 1 dimensional case:
- Regression: $y=wx+w_0$
- Classification: $y=1(wx+w_0>0)$






## Perceptron Decision Surface

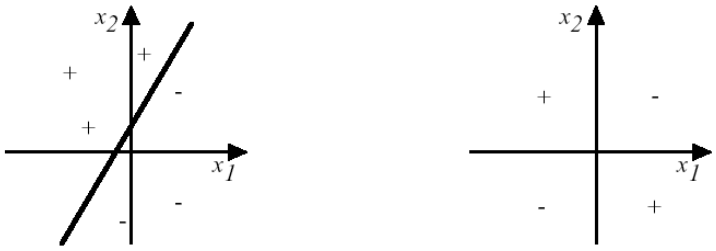



Perceptron as hyperplane decision surface in the n-dimensional input space

The perceptron outputs 1 for instances lying on one side of the hyperplane and outputs –1 for instances on the other side

Data that can be separated by a hyperplane: *linearly separable*

## Perceptron Decision Surface




A single unit can represent some useful functions
• What weights represent
g(x1, x2) = AND(x1, x2)? Majority, Or
But some functions not representable
• e.g., not linearly separable
• Therefore, we'll want networks of these...



## Perceptron training rule

$$w_i \leftarrow w_i + \Delta w_i$$

where

$$\Delta w_i = \eta (t-o) x_i$$

Where:

- $t = c(\boldsymbol{x})$ is target value
- $o$ is perceptron output
- $\eta$ is small constant (e.g., .1) called the *learning rate* (or *step size*)

## Perceptron training rule

Can prove it will converge

- If training data is linearly separable
- and η sufficiently small
- Perceptron Conversion Theorem (Rosenblatt): if the data are linearly separable then the perceptron learning algorithm converges in finite time.



## Gradient Descent – Delta Rule

Also know as LMS (least mean squares) rule or widrow-Hoff rule.

To understand, consider simpler *linear unit*, where

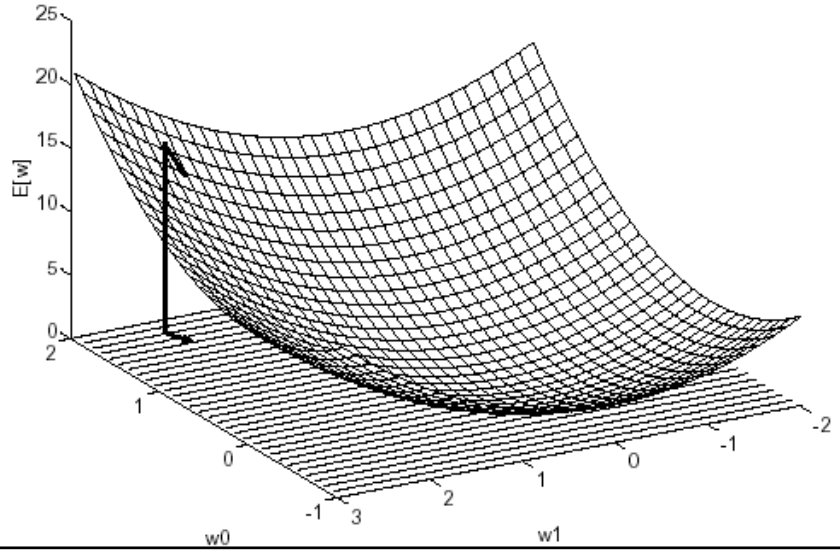
$$o = w_0 + w_1 x_1 + \ldots + w_n x_n$$

Let's learn $w_i$'s to minimize squared error

$$E[\boldsymbol{w}] \equiv 1/2 \sum_{d \text{ in } D} (t_d - o_d)^2$$

Where $D$ is set of training examples

## Error Surface




## Gradient Descent

Gradient

$\nabla E[\boldsymbol{w}] = [\partial E/\partial w_0, \partial E/\partial w_1, \ldots, \partial E/\partial w_n]$

When interpreted as a vector in weight space, the gradient specifies the direction that produces the steepest increase in E

Training rule:

$$\Delta \boldsymbol{w} = -\eta \nabla E[\boldsymbol{w}]$$

in other words:

$$\Delta w_i = -\eta \, \partial E/\partial w_i$$

This results in the following update rule:

$$\Delta w_i = \eta \sum_d (t_d - o_d)(x_{i,d})$$

## Gradient of Error

$\partial E/\partial w_i$

$= \partial/\partial w_i \ 1/2 \sum_d (t_d - o_d)^2$

$= 1/2 \sum_d \partial/\partial w_i (t_d - o_d)^2$

$= 1/2 \sum_d 2 (t_d - o_d) \partial/\partial w_i (t_d - o_d)$

$= \sum_d (t_d - o_d) \partial/\partial w_i (t_d - \boldsymbol{w}\, \boldsymbol{x}_d)$

$= \sum_d (t_d - o_d) (-x_{i,d})$

Learning Rule:

$\Delta w_i = -\eta \, \partial E/\partial w_i$

$\Rightarrow \Delta w_i = \eta \sum_d (t_d - o_d) (x_{i,d})$



## Stochastic Gradient Descent

**Batch mode** Gradient Descent:

Do until satisfied

1. Compute the gradient $\nabla E_D [\boldsymbol{w}]$
2. $\boldsymbol{w} \leftarrow \boldsymbol{w} - \nabla E_D [\boldsymbol{w}]$

**Incremental mode** Gradient Descent:

Do until satisfied

- For each training example $d$ in $D$
  1. Compute the gradient $\nabla E_d [\boldsymbol{w}]$
  2. $\boldsymbol{w} \leftarrow \boldsymbol{w} - \nabla E_d [\boldsymbol{w}]$

# More Stochastic Grad. Desc.

$E_D[\boldsymbol{w}] \equiv 1/2 \sum_{d \text{ in } D} (t_d - o_d)^2$

$E_d[\boldsymbol{w}] \equiv 1/2 (t_d - o_d)^2$

*Incremental Gradient* Descent can approximate *Batch Gradient Descent* arbitrarily closely if $\eta$ set small enough

Incremental Learning Rule: $\Delta w_i = \eta (t_d - o_d) (x_{i,d})$

Delta Rule: $\Delta w_i = \eta (t - o) (x_i)$

$\delta = (t - o)$



# Gradient Descent Code

GRADIENT-DESCENT(training examples, $\eta$)

*Each training example is a pair of the form <**x**, t>, where **x** is the vector of input values, and t is the target output value. $\eta$ is the learning rate (e.g., .05).*

- Initialize each $w_i$ to some small random value
- Until the termination condition is met, Do
  - Initialize each $\Delta w_i$ to zero.
  - For each <**x**, $t$> in training examples, Do
    - Input the instance **x** to the unit and compute the output $o$
    - For each linear unit weight $w_i$, Do

$$\Delta w_i \leftarrow \Delta w_i + \eta (t - o) x_i$$

  - For each linear unit weight $w_i$, Do

$$w_i \leftarrow w_i + \Delta w_i$$

## Summary

Perceptron training rule will succeed if

- Training examples are linearly separable
- Sufficiently small learning rate η

Linear unit training uses gradient descent (delta rule)

- Guaranteed to converge to hypothesis with minimum squared error
- Given sufficiently small learning rate η
- Even when training data contains noise
- Even when training data not *H* separable



## K Outputs

Regression:

$$y_i = \sum_{j=1}^{d} w_{ij} x_j + w_{i0} = \boldsymbol{w}_i^T \boldsymbol{x}$$

$$\boldsymbol{y} = \mathbf{W}\boldsymbol{x}$$ Linear Map from $R^d \Rightarrow R^k$

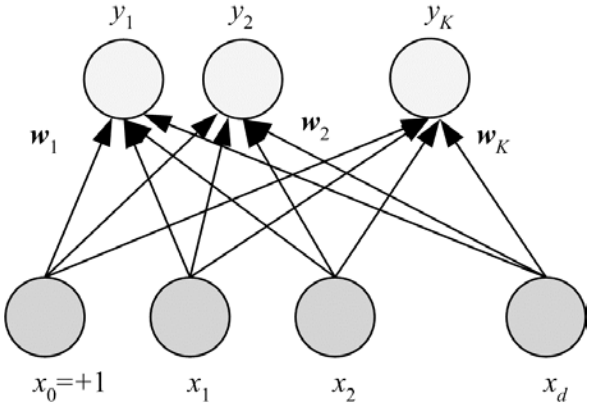
Classification:

$$o_i = \boldsymbol{w}_i^T \boldsymbol{x}$$

$$y_i = \frac{\exp o_i}{\sum_k \exp o_k}$$

choose $C_i$

if $y_i = \max_k y_k$



## Training

- Online (instances seen one by one) vs batch (whole sample) learning:
  - No need to store the whole sample
  - Problem may change in time
  - Wear and degradation in system components
- Stochastic gradient-descent: Update after a single pattern
- Generic update rule (LMS rule):

$$\Delta w_{ij}^t = \eta\left(r_i^t - y_i^t\right)x_j^t$$

$$\text{Update} = \text{LearningFactor} \cdot (\text{DesiredOutput} - \text{ActualOutput}) \cdot \text{Input}$$



## Training a Perceptron: Regression

- Regression (Linear output):

$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, r^t\right) = \frac{1}{2}\left(r^t - y^t\right)^2 = \frac{1}{2}\left[r^t - \left(\mathbf{w}^T\mathbf{x}^t\right)\right]^2$$

$$\Delta w_j^t = \eta\left(r^t - y^t\right)x_j^t$$

## Classification

- Single sigmoid output

$$y^t = \text{sigmoid}\left(\mathbf{w}^T\mathbf{x}^t\right)$$

- $K>2$ softmax outputs

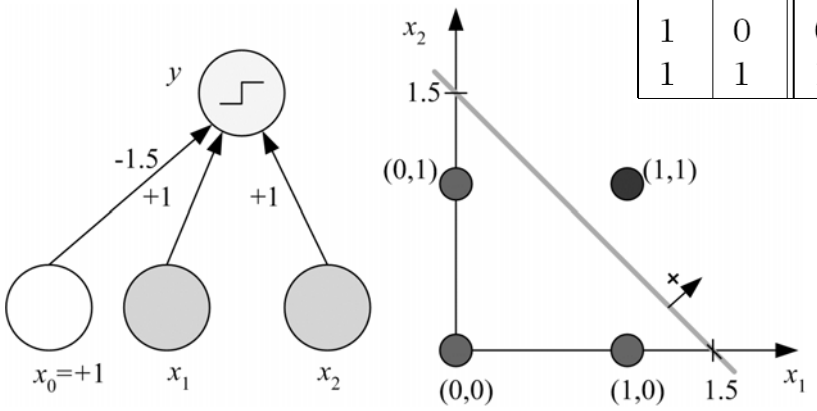
$$E^t\left(\mathbf{w} \mid \mathbf{x}^t, \mathbf{r}^t\right) = -r^t \log y^t - \left(1 - r^t\right)\log\left(1 - y^t\right)$$
$$\Delta w_j^t = \eta\left(r^t - y^t\right)x_j^t$$

$$y^t = \frac{\exp \boldsymbol{w}_i^T \boldsymbol{x}^t}{\sum_k \exp \boldsymbol{w}_k^T \boldsymbol{x}^t} \qquad E^t\left(\{\boldsymbol{w}_i\}_i \mid \boldsymbol{x}^t, \boldsymbol{r}^t\right) = -\sum_i r_i^t \log y_i^t$$
$$\Delta w_{ij}^t = \eta\left(r_i^t - y_i^t\right)x_j^t$$



## Learning Boolean AND

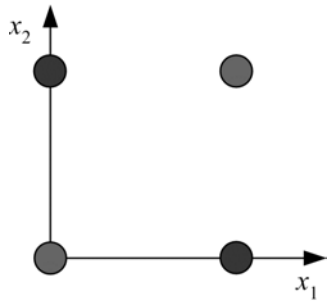
| $x_1$ | $x_2$ | $r$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |



## XOR

- No $w$

| $x_1$ | $x_2$ | $r$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

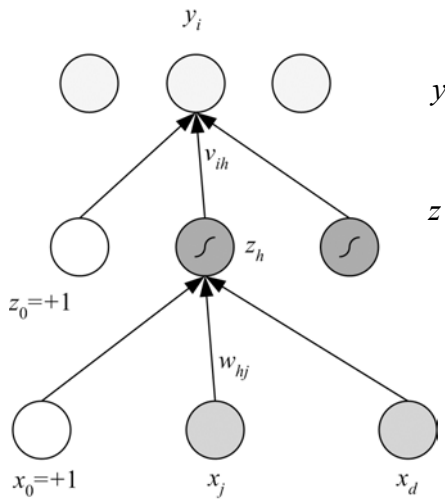



$$w_0 \le 0$$
$$w_2 + w_0 > 0$$
$$w_1 + w_0 > 0$$
$$w_1 + w_2 + w_0 \le 0$$

(Minsky and Papert, 1969)



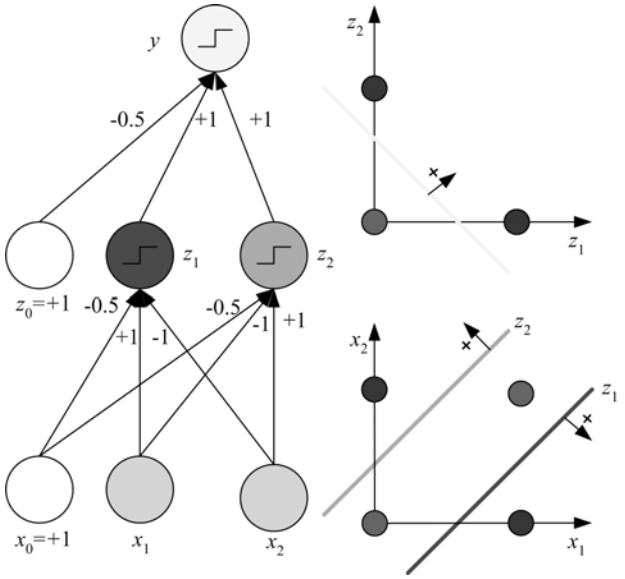
## Multilayer Perceptrons




$$y_i = \boldsymbol{v}_i^T \boldsymbol{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\boldsymbol{w}_h^T \boldsymbol{x}\right) = \frac{1}{1+\exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

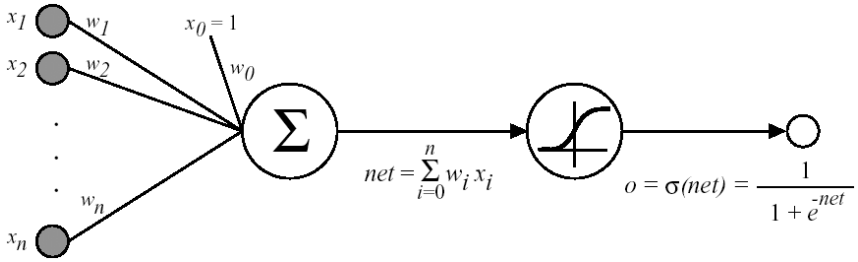
(Rumelhart et al., 1986)




$x_1$ XOR $x_2$ = ($x_1$ AND ~$x_2$) OR (~$x_1$ AND $x_2$)



## Sigmoid Activation




σ(x) is the sigmoid (s-like) function

$1/(1 + e^{-x})$

Nice property:

$d\,\sigma(x)/dx = \sigma(x)\,(1-\sigma(x))$

Other variations:

σ(x) $= 1/(1 + e^{-k.x})$ where k is a positive constant that determines the steepness of the threshold.

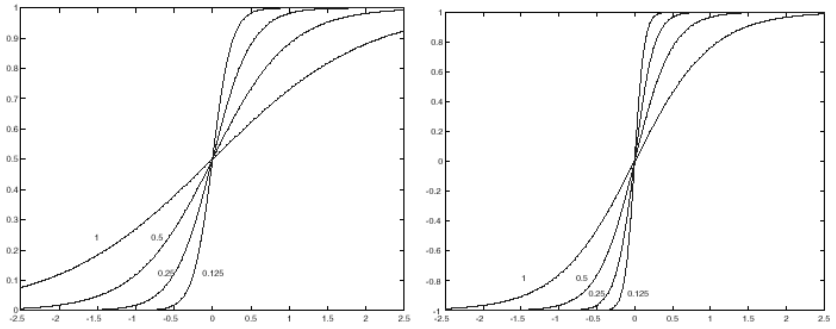



**Figure 22.14.** On the **left**, a series of squashing functions obtained using $\phi(x;\nu) = \frac{e^{x/\nu}}{1+e^{x/\nu}}$, for different values of $\nu$ indicated on the figure. On the **right**, a series of squashing functions obtained using $\phi(x;\nu,A) = A\tanh(x/\nu)$ for different values of $\nu$ indicated on the figure. Generally, for $x$ close to the center of the range, the squashing function is linear; for $x$ small or large, it is strongly non-linear.



# Error Gradient for Sigmoid

$\partial E/\partial w_i$

$= \partial/\partial w_i \; 1/2 \sum_d (t_d - o_d)^2$

$= 1/2 \sum_d \partial/\partial w_i (t_d - o_d)^2$

$= 1/2 \sum_d 2 (t_d - o_d) \partial/\partial w_i (t_d - o_d)$

$= \sum_d (t_d - o_d) (-\partial o_d/\partial w_i)$

$= - \sum_d (t_d - o_d) (\partial o_d/\partial net_d \partial net_d/\partial w_i)$

## Even more...

But we know:

$\partial o_d/\partial net_d$

$= \partial\sigma(net_d)/\partial net_d = o_d\,(1- o_d\,)$

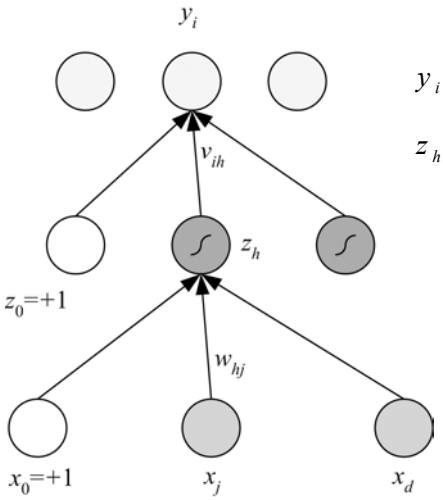
$\partial net_d/\partial w_i = \partial(\boldsymbol{w}\cdot\boldsymbol{x}_d)\,/\partial w_i = x_{i,d}$

So:

$$\partial E/\partial w_i = -\sum_d (t_d - o_d) o_d\,(1 - o_d)\, x_{i,d}$$



## Backpropagation




$$y_i = \boldsymbol{v}_i^T \boldsymbol{z} = \sum_{h=1}^{H} v_{ih} z_h + v_{i0}$$

$$z_h = \text{sigmoid}\left(\boldsymbol{w}_h^T \boldsymbol{x}\right) = \frac{1}{1+\exp\left[-\left(\sum_{j=1}^{d} w_{hj} x_j + w_{h0}\right)\right]}$$

$$\frac{\partial E}{\partial w_{hj}} = \frac{\partial E}{\partial y_i}\frac{\partial y_i}{\partial z_h}\frac{\partial z_h}{\partial w_{hj}}$$

# Backpropagation Algorithm

Initialize all weights to small random numbers.

Until satisfied, Do

- For each training example, Do
  1. Input the training example to the network and compute the network outputs
  2. For each output unit $k$

     $\delta_k = o_k(1 - o_k)(t_k - o_k)$
  3. For each hidden unit $h$

     $\delta_h = o_h(1 - o_h) \sum_{k \text{ in } outputs} w_{h,k}\, \delta_k$
  4. Update each network weight $w_{i,j}$

     $w_{i,j} \leftarrow w_{i,j} + \Delta w_{i,j}$ where $\Delta w_{i,j} = \eta\, \delta_j a_i$



# *Regression*

$$E(\mathbf{W}, \boldsymbol{v} \mid \mathrm{X}) = \frac{1}{2}\sum_t \left(r^t - y^t\right)^2$$

$$y^t = \sum_{h=1}^{H} v_h z_h^t + v_0$$

$$\Delta v_h = \sum_t \left(r^t - y^t\right) z_h^t$$

*Backward*

*Forward*

$$z_h = \text{sigmoid}\left(\boldsymbol{w}_h^T \boldsymbol{x}\right)$$

$$\boldsymbol{x}$$

$$\begin{aligned}
\Delta w_{hj} &= -\eta \frac{\partial E}{\partial w_{hj}} \\
&= -\eta \sum_t \frac{\partial E}{\partial y^t} \frac{\partial y^t}{\partial z_h^t} \frac{\partial z_h^t}{\partial w_{hj}} \\
&= -\eta \sum_t -\left(r^t - y^t\right) v_h z_h^t \left(1 - z_h^t\right) x_j^t \\
&= \eta \sum_t \left(r^t - y^t\right) v_h z_h^t \left(1 - z_h^t\right) x_j^t
\end{aligned}$$

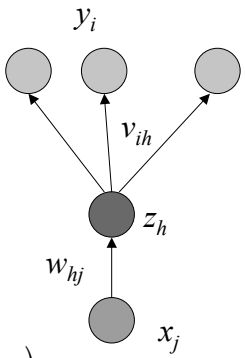
## Regression with Multiple Outputs

$$E(\mathbf{W},\mathbf{V}\mid \mathbf{X})=\frac{1}{2}\sum_t\sum_i\left(r_i^t-y_i^t\right)^2$$

$$y_i^t=\sum_{h=1}^{H}v_{ih}z_h^t+v_{i0}$$

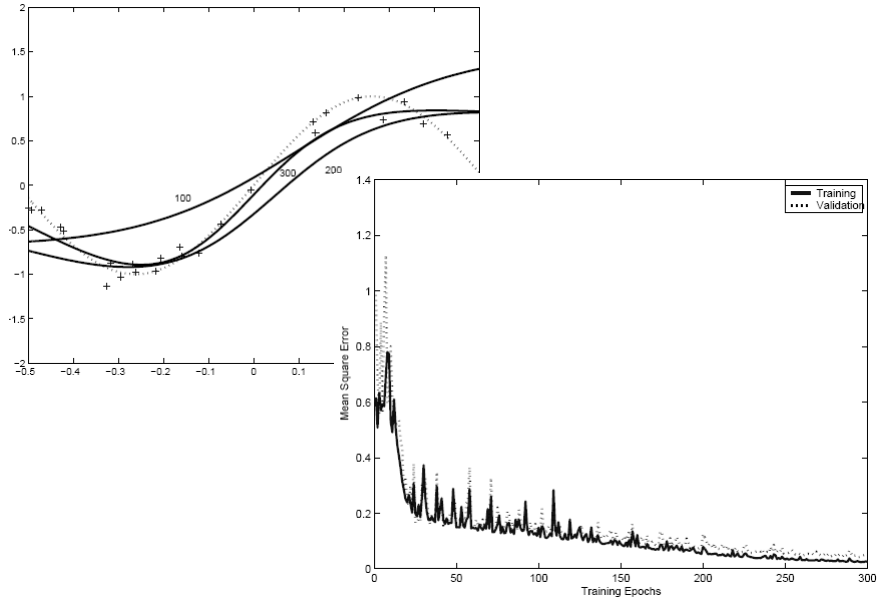
$$\Delta v_{ih}=\eta\sum_t\left(r_i^t-y_i^t\right)z_h^t$$

$$\Delta w_{hj}=\eta\sum_t\left[\sum_i\left(r_i^t-y_i^t\right)v_{ih}\right]z_h^t\left(1-z_h^t\right)x_j^t$$

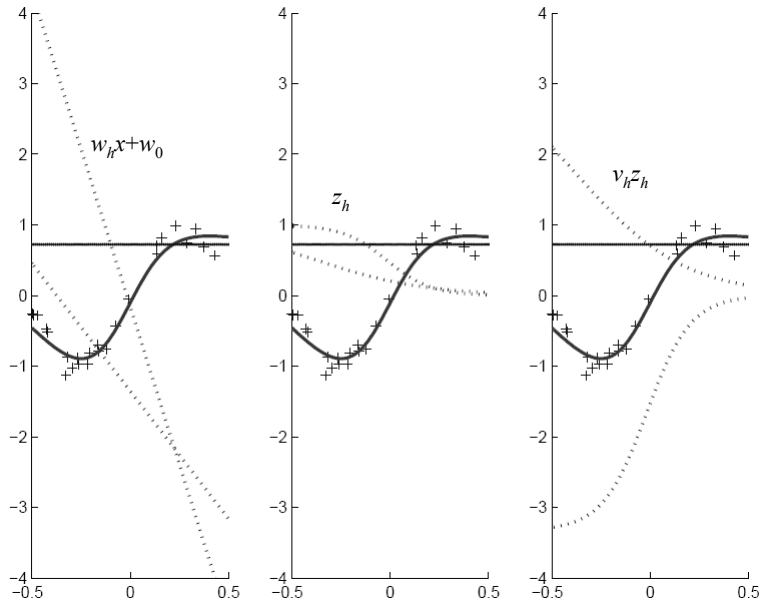





Initialize all $v_{ih}$ and $w_{hj}$ to rand$(-0.01, 0.01)$
Repeat
  For all $(\boldsymbol{x}^t, r^t) \in \mathcal{X}$ in random order
    For $h = 1, \ldots, H$
      $z_h \leftarrow \text{sigmoid}(\boldsymbol{w}_h^T \boldsymbol{x}^t)$
    For $i = 1, \ldots, K$
      $y_i = \boldsymbol{v}_i^T \boldsymbol{z}$
    For $i = 1, \ldots, K$
      $\Delta \boldsymbol{v}_i = \eta(r_i^t - y_i^t)\boldsymbol{z}$
    For $h = 1, \ldots, H$
      $\Delta \boldsymbol{w}_h = \eta(\sum_i (r_i^t - y_i^t) v_{ih}) z_h (1 - z_h) \boldsymbol{x}^t$
    For $i = 1, \ldots, K$
      $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \Delta \boldsymbol{v}_i$
    For $h = 1, \ldots, H$
      $\boldsymbol{w}_h \leftarrow \boldsymbol{w}_h + \Delta \boldsymbol{w}_h$
Until convergence


100
200
300
Training
Validation
Mean Square Error
Training Epochs




$w_h x+w_0$
$z_h$
$v_h z_h$

## Two-Class Discrimination

- One sigmoid output $y^t$ for $P(C_1|\boldsymbol{x}^t)$ and $P(C_2|\boldsymbol{x}^t) \equiv 1-y^t$

$$y^t = \text{sigmoid}\left(\sum_{h=1}^{H} v_h z_h^t + v_0\right)$$

$$E(\mathbf{W},\boldsymbol{v} \mid \mathrm{X}) = -\sum_t r^t \log y^t + (1-r^t)\log(1-y^t)$$

$$\Delta v_h = \eta \sum_t (r^t - y^t) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t (r^t - y^t) v_h z_h^t (1-z_h^t) x_j^t$$



## K>2 Classes

$$o_i^t = \sum_{h=1}^{H} v_{ih} z_h^t + v_{i0} \qquad y_i^t = \frac{\exp o_i^t}{\sum_k \exp o_k^t} \equiv P(C_i \mid \boldsymbol{x}^t)$$

$$E(\mathbf{W},\boldsymbol{v} \mid \mathrm{X}) = -\sum_t \sum_i r_i^t \log y_i^t$$

$$\Delta v_{ih} = \eta \sum_t (r_i^t - y_i^t) z_h^t$$

$$\Delta w_{hj} = \eta \sum_t \left[\sum_i (r_i^t - y_i^t) v_{ih}\right] z_h^t (1-z_h^t) x_j^t$$

## Multiple Hidden Layers

- MLP with one hidden layer is a universal approximator (Hornik et al., 1989), but using multiple layers may lead to simpler networks

$$z_{1h} = \text{sigmoid}\left(\boldsymbol{w}_{1h}^T \boldsymbol{x}\right) = \text{sigmoid}\left(\sum_{j=1}^{d} w_{1hj} x_j + w_{1h0}\right), h = 1,..., H_1$$

$$z_{2l} = \text{sigmoid}\left(\boldsymbol{w}_{2l}^T \boldsymbol{x}\right) = \text{sigmoid}\left(\sum_{h=1}^{H_1} w_{2lh} z_{1h} + w_{2l0}\right), l = 1,..., H_2$$

$$y = \boldsymbol{v}^T \boldsymbol{z}_2 = \sum_{l=1}^{H_2} v_l z_{2l} + v_0$$



## Improving Convergence

- Momentum

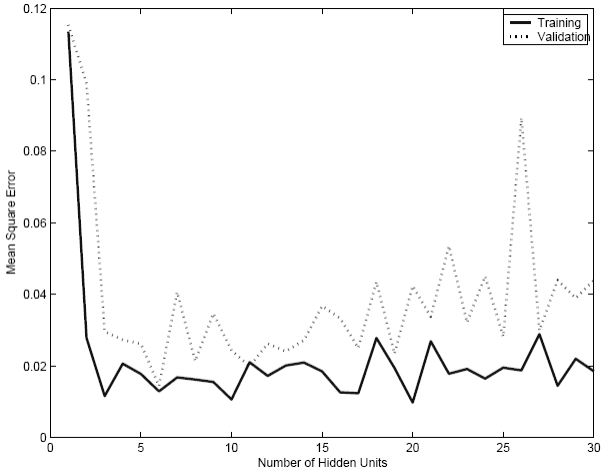
$$\Delta w_i^t = -\eta \frac{\partial E^t}{\partial w_i} + \alpha \Delta w_i^{t-1}$$

- Adaptive learning rate

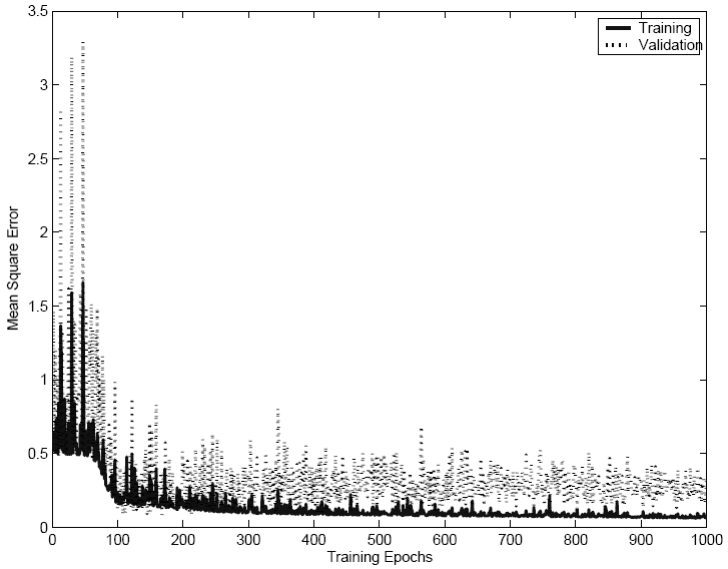
$$\Delta\eta = \begin{cases} +a & \text{if } E^{t+\tau} < E^t \\ -b\eta & \text{otherwise} \end{cases}$$

# Overfitting/Overtraining

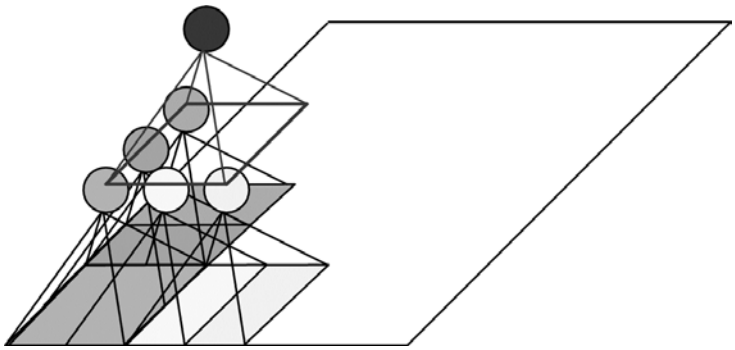
Number of weights: $H\,(d+1)+(H+1)*K$





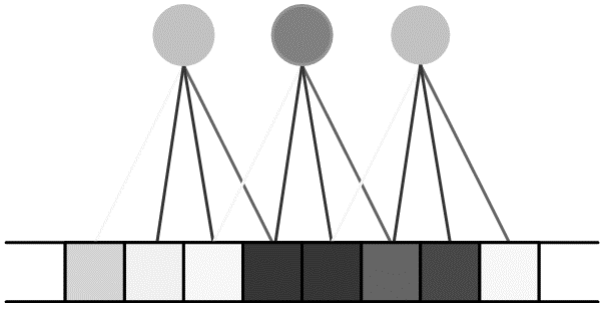
## Structured MLP



(Le Cun et al, 1989)



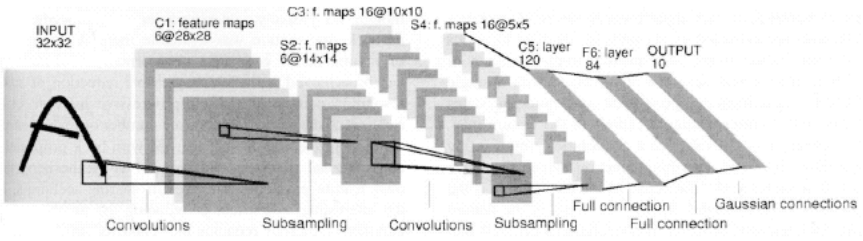
## Weight Sharing



# Convolutional neural networks

- Also known as gradient-based learning
- Template matching using NN classifiers seems to work
- Natural features are filter outputs
  - probably, spots and bars, as in texture
  - but why not learn the filter kernels, too?
- a perceptron approximates convolution.
- Network architecture: Two types of layers
  - Convolution layers: convolving the image with filter kernels to obtain filter maps
  - Subsampling layers: reduce the resolution of the filter maps
  - The number of filter maps increases as the resolution decreases






A convolutional neural network, LeNet; the layers filter, subsample, filter, subsample, and finally classify based on outputs of this process.

Figure from "Gradient-Based Learning Applied to Document Recognition", Y. Lecun et al Proc. IEEE, 1998 copyright 1998, IEEE

Fig. 4. Size-normalized examples from the MNIST database.

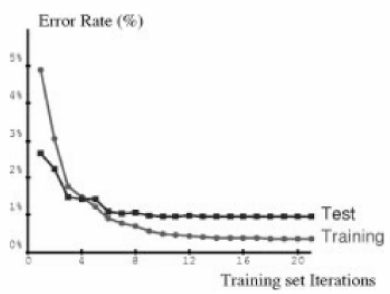



Fig. 5. Training and test error of LeNet-5 as a function of the number of passes through the 60 000 pattern training set (without distortions). The average training error is measured on-the-fly as training proceeds. This explains why the training error appears to be larger than the test error initially. Convergence is attained after 10–12 passes through the training set.

LeNet is used to classify handwritten digits. Notice that the test error rate is not the same as the training error rate, because the test set consists of items not in the training set. Not all classification schemes necessarily have small test error when they have small training error. Error rate 0.95% on MNIST database

Figure from "Gradient-Based Learning Applied to Document Recognition", Y. Lecun et al Proc. IEEE, 1998 copyright 1998, IEEE

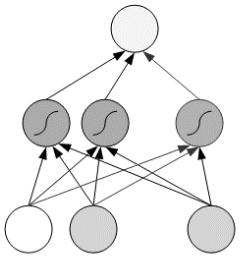


# Tuning the Network Size

- Destructive
- Weight decay:
- Constructive
- Growing networks

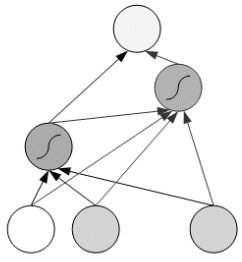
$$\Delta w_i = -\eta \frac{\partial E}{\partial w_i} - \lambda w_i$$

$$E' = E + \frac{\lambda}{2} \sum_i w_i^2$$



*Dynamic Node Creation*

(Ash, 1989)



*Cascade Correlation*

(Fahlman and Lebiere, 1989)

## Bayesian Learning

$$p(\mathbf{w}\mid \mathsf{X}) = \frac{p(\mathsf{X}\mid \mathbf{w})p(\mathbf{w})}{p(\mathsf{X})} \quad \hat{\mathbf{w}}_{MAP} = \arg\max_{\mathbf{w}} \log p(\mathbf{w}\mid \mathsf{X})$$

$$\log p(\mathbf{w}\mid \mathsf{X}) = \log p(\mathsf{X}\mid \mathbf{w}) + \log p(\mathbf{w}) + C$$

- Consider weights $w_i$ as random vars, prior $p(w_i)$

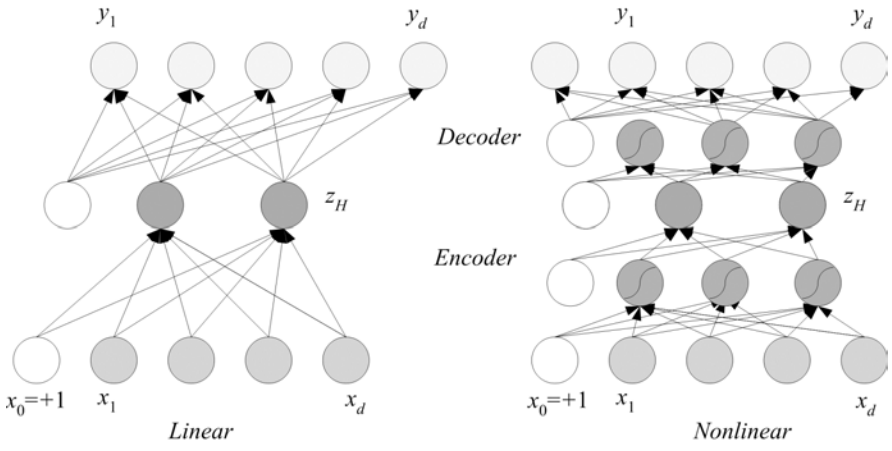
$$p(\mathbf{w}) = \prod_i p(w_i) \text{ where } p(w_i) = c \cdot \exp\left[-\frac{w_i^2}{2(1/2\lambda)}\right]$$

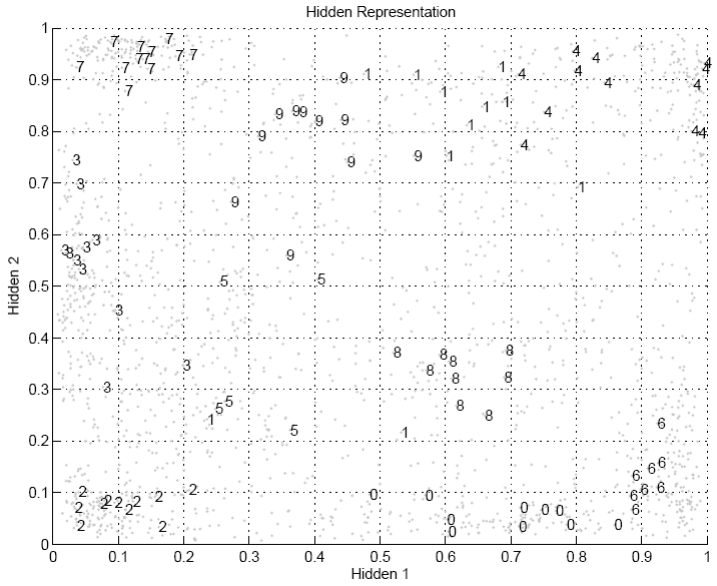
$$E' = E + \lambda \|\mathbf{w}\|^2$$

- Weight decay, ridge regression, regularization
  cost=data-misfit + λ complexity



## Dimensionality Reduction



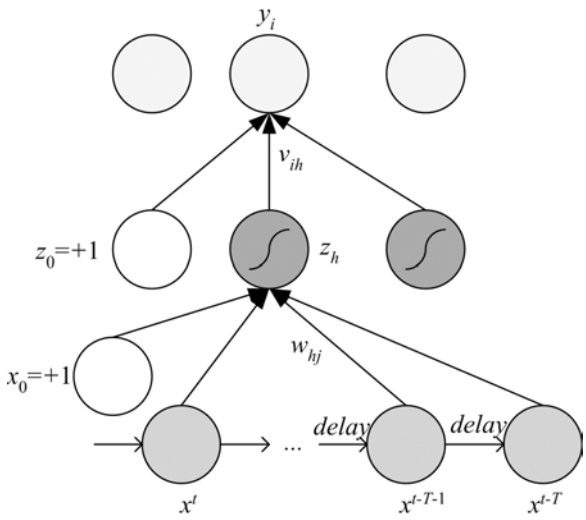





# Learning Time

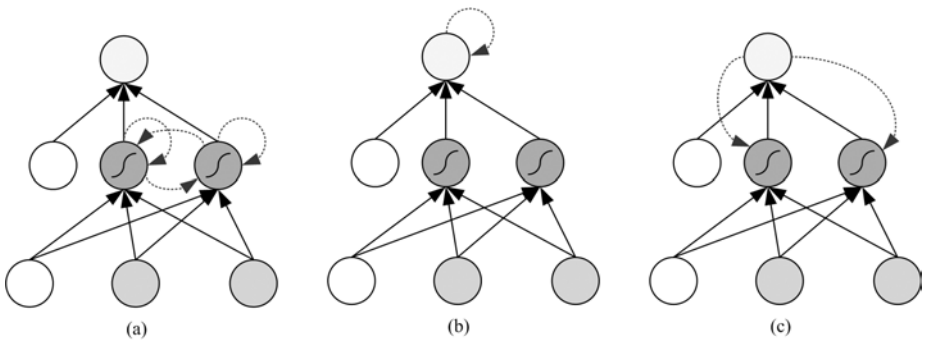
- Applications:
  - Sequence recognition: Speech recognition
  - Sequence reproduction: Time-series prediction
  - Sequence association
- Network architectures
  - Time-delay networks (Waibel et al., 1989)
  - Recurrent networks (Rumelhart et al., 1986)

## Time-Delay Neural Networks

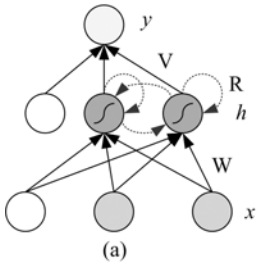


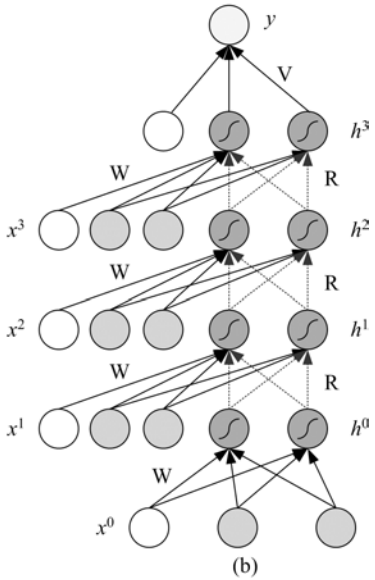



## Recurrent Networks

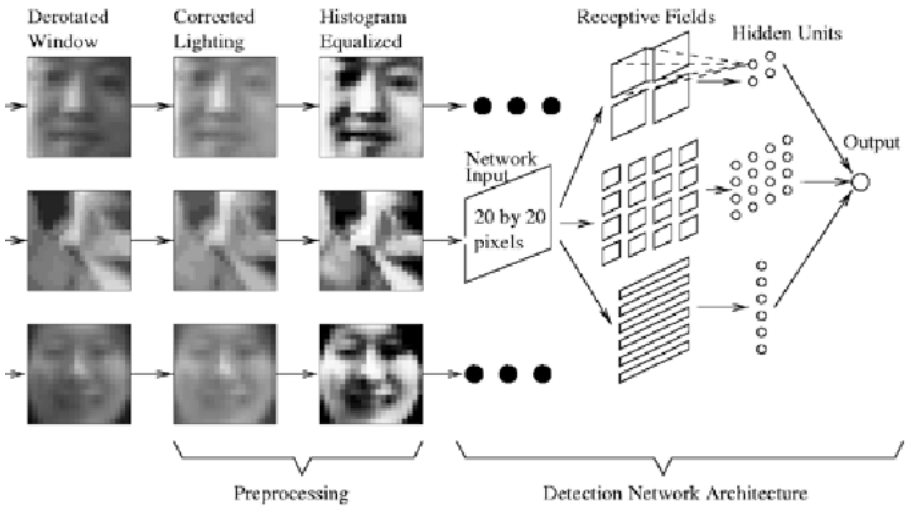


# Unfolding in Time



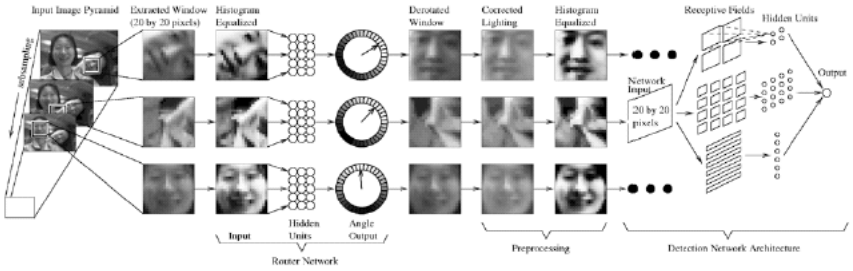






The vertical face-finding part of Rowley, Baluja and Kanade's system

Figure from "Rotation invariant neural-network based face detection," H.A. Rowley, S. Baluja and T. Kanade, Proc. Computer Vision and Pattern Recognition, 1998, copyright 1998, IEEE




Architecture of the complete system: they use another neural net to estimate orientation of the face, then rectify it. They search over scales to find bigger/smaller faces.

Figure from "Rotation invariant neural-network based face detection," H.A. Rowley, S. Baluja and T. Kanade, Proc. Computer Vision and Pattern Recognition, 1998, copyright 1998, IEEE



Figure from "Rotation invariant neural-network based face detection," H.A. Rowley, S. Baluja and T. Kanade, Proc. Computer Vision and Pattern Recognition, 1998, copyright 1998, IEEE

## Sources

- Slides by Ethem Elpaydin, “introduction to machine learning” © The MIT Press, 2004
- Slides by Tom M. Mitchell
- Ethem Elpaydin, “introduction to machine learning” Chapter 10
- Tom M. Mitchell “Machine Learning”