# Learning Models and Rules (I)

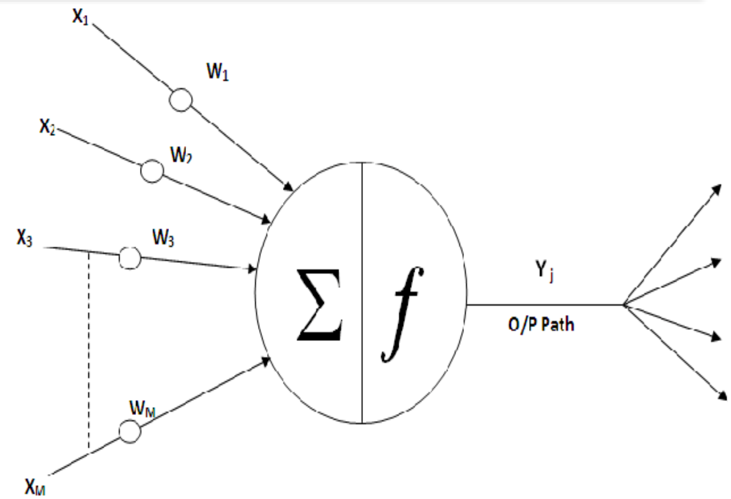
**Jin Young Choi**

**Seoul National University**

# Outline

- Learning rules and models
- Statistical nature of learning
- Empirical risk minimization
- Structural risk minimization

# Learning Models

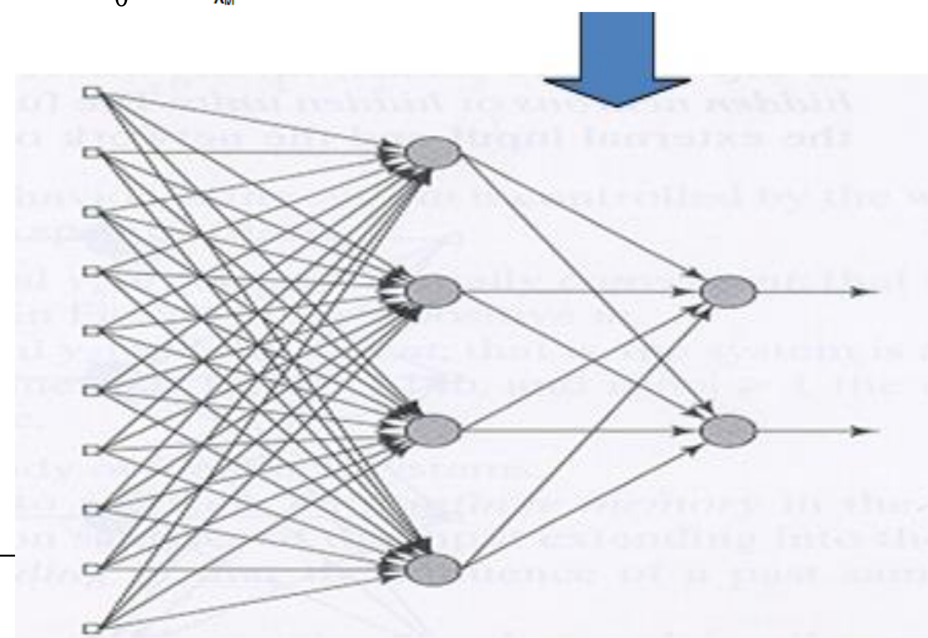



- Probability Discriminant Functions
  - A Posteriori Probability Function

$$g_i(\mathbf{x}) = p(\omega_i \mid \mathbf{x})$$

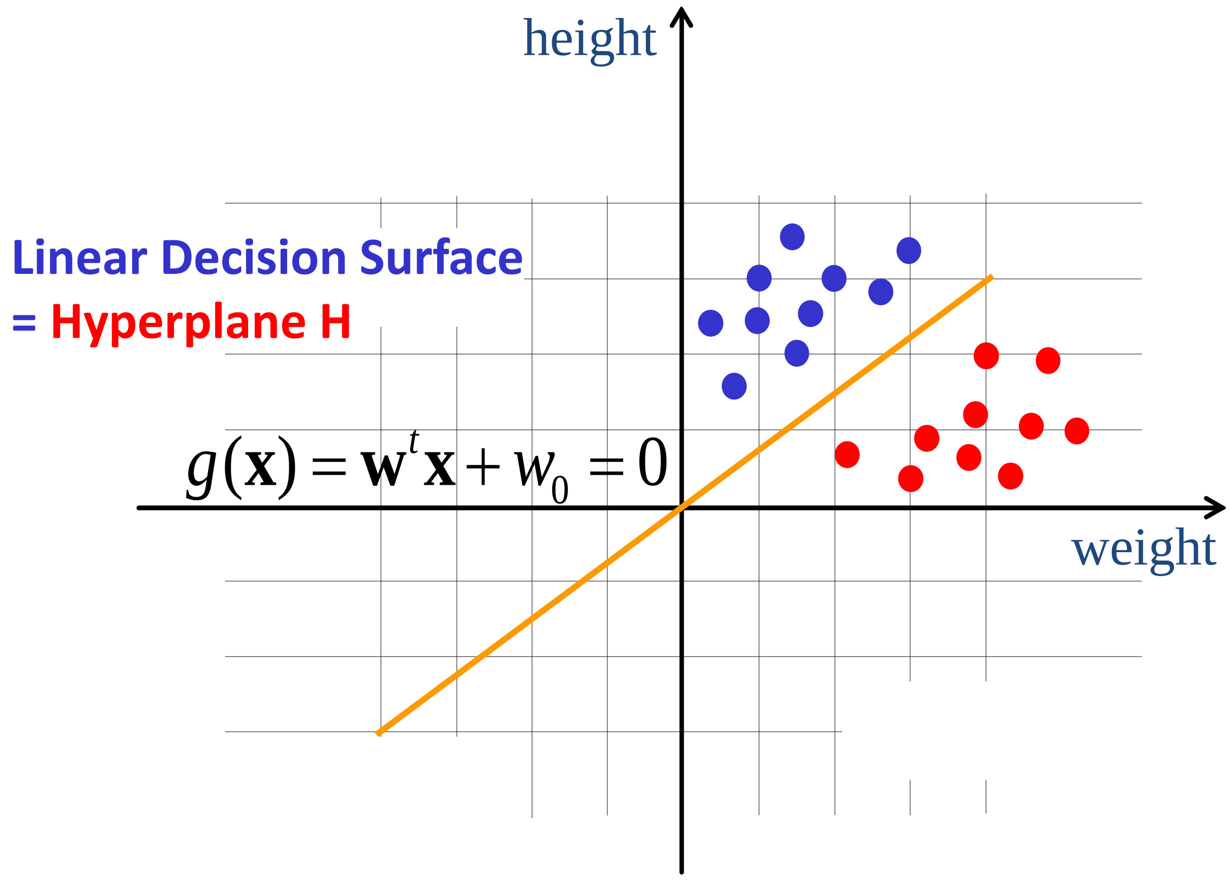
- Linear Learning Models

$$g(\mathbf{x}) = w_1 x_1 + \ldots + w_d x_d + w_0 = \mathbf{w}^t \mathbf{x} + w_0$$

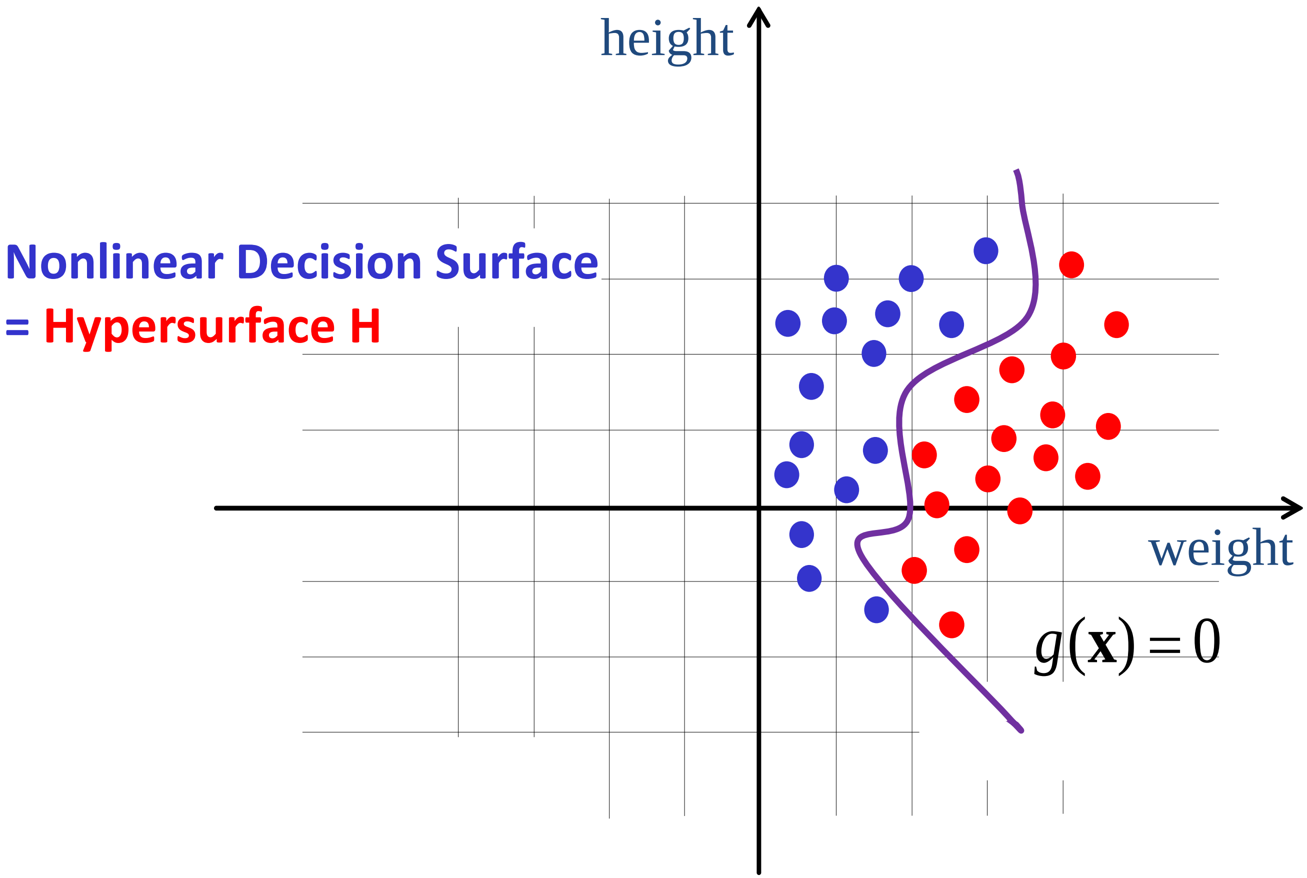


- Universal Learning Models
  - Basis networks
    - Gaussian basis
    - High order basis
    - Sine, cosine, wavelet basis
  - (Deep) Neural networks

# Linear Learning Models



# Universal Learning Models



# Learning Strategy

- Nonlinear Learning Approach
  - Learning of nonlinear kernel parameters
  - Steepest  descent method (error backpropagation Learning)
  - Approximated newton method
  - Genetic algorithm, simulated annealing
  - Slow learning speed
  - Local minima
- Linear learning approach
  - Convex optimization (least squares, support vector machine)
  - Fast learning speed, mathematically sound
  - Kernel determination problems
  - Generalization problems: high dimensional feature space

# Statistical Nature of Learning

- The generalized linear discriminant function
  (General Learning Model) $g(x,\theta) = \mathbf{w}^t \boldsymbol{\varphi}(x,\mathbf{v}), \quad \theta^t = [\mathbf{w}^t \quad \mathbf{v}^t]$

- Observations (i.i.d. examples) $\Im = \{(x_i, d_i)\}_{i=1}^{N}$

- For binary classification
$$d_i = 1, \quad \forall x_i \in \omega_1,$$
$$d_i = -1, \quad \forall x_i \in \omega_2,$$

- Multi-class classification
$$d_i = 1, \ \forall x_i \in \omega_i$$
$$d_j = 0, \ \forall x_j \notin \omega_i$$

- Multi-label classification: [ 0  1 0 1  1 0 1  1 0]  … …
  - Ex)  portrait with attributes (Bald, Mustache, Gold hair, Blue eye, ….etc.)

# Statistical Nature of Learning

- Loss Function

$$L(d, g(x,\theta)) = (d - g(x,\theta))^2 (\text{or Entropy, KLD})$$
$$= -d \log g(x,\theta)$$

- Risk Functional (Expectation Value of $L$) (or Total Loss)

$$R(\theta) = \int L(d, g(x,\theta)) dp(x,d)$$

where $p(x,d)$ is joint PDF for $x$ & $d$, but unknown.

- Empirical Risk Functional

$$R_{emp}(\theta) = \frac{1}{N} \sum_{i=1}^{N} L(d_i, g(x_i,\theta))$$

# Statistical Nature of Learning

- Definition(Convergence in Probability):

  Sequence of random variables $\theta_1, \theta_2, \theta_3, \ldots, \theta_N$, is said to converge in probability to $\theta_*$ ,i.e.,

  $$\theta_N \xrightarrow{P} \theta_*,$$

  if for any $\delta > 0$,

  $$p(|\theta_N - \theta_*| > \delta) \to 0 \quad \text{as} \quad N \to \infty.$$

- Definition(Strict Consistency):

  Let $\Theta(c) = \{\theta \mid R(\theta) \geq c\}$ ,

  If

  $$\inf_{\theta \in \Theta(c)} R_{emp}(\theta) \xrightarrow{P} \inf_{\theta \in \Theta(c)} R(\theta), \quad as \quad N \to \infty$$

  The empirical risk function is said to be strictly consistent.

*Vapnik, Vladimir (2000). The nature of statistical learning theory. Springer.*

# Empirical Risk Minimization

- Empirical Risk Minimization

$$\theta_{emp} = \arg\min_{\theta} R_{emp}(\theta)$$

- Original Risk Minimization

$$\theta_o = \arg\min_{\theta} R(\theta)$$

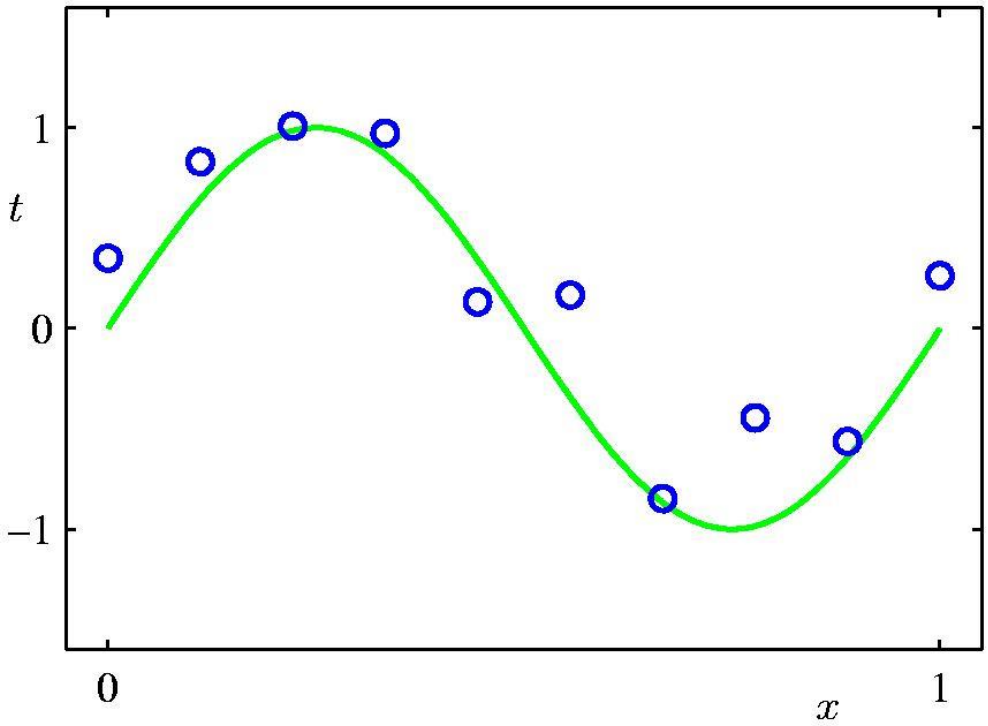
- Meaning

$R(\theta)$ is probability of optimal decision error

$R_{emp}(\theta)$ is probability of training error

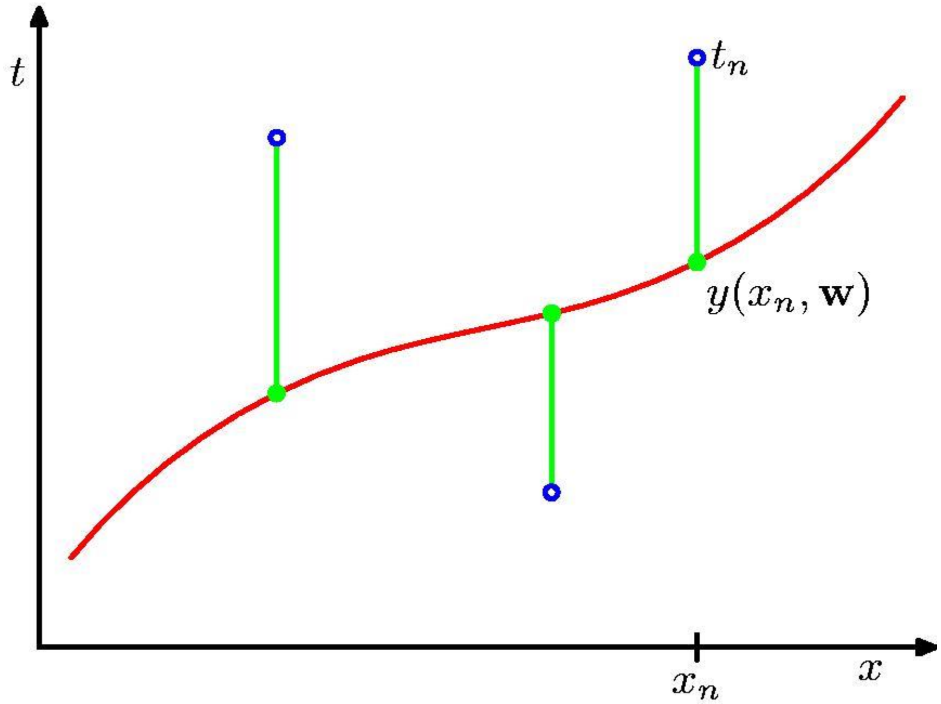
- The difference between $R(\theta)$ and $R_{emp}(\theta)$ depends #of samples and the complexity of a classifier.

# Polynomial Curve Fitting



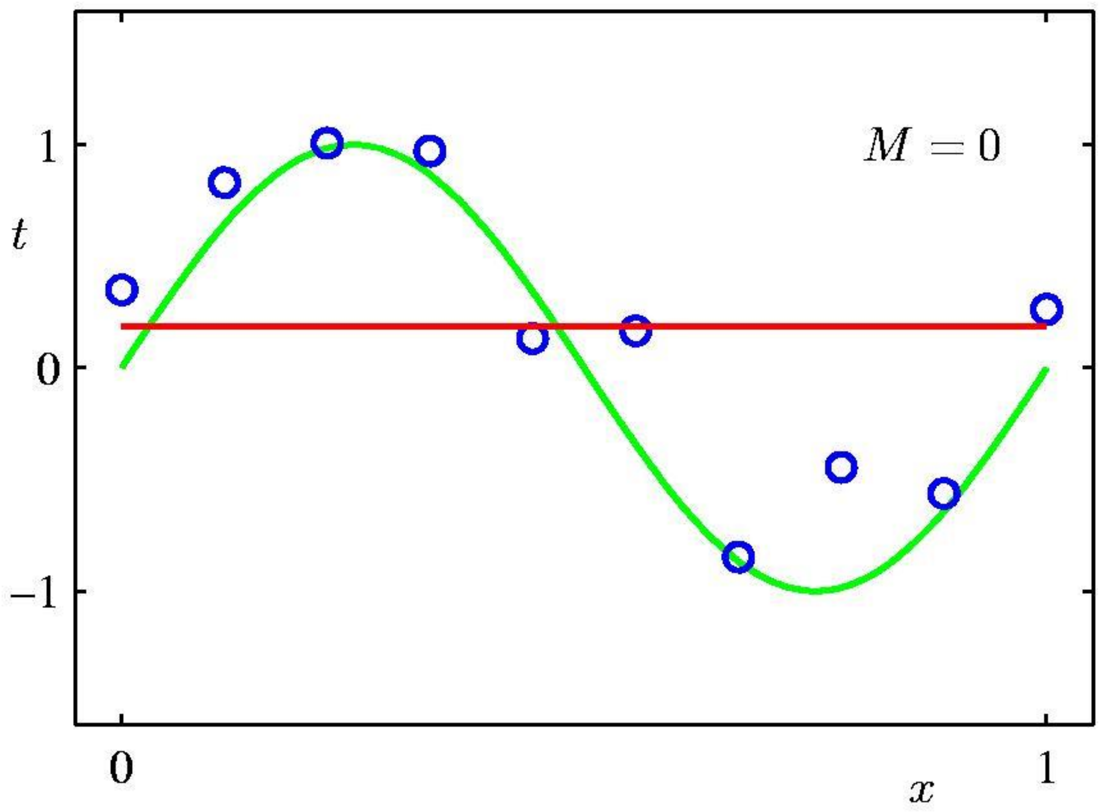

$$y(x, \mathbf{w}) = w_0 + w_1 x + w_2 x^2 + \ldots + w_M x^M = \sum_{j=0}^{M} w_j x^j$$

# Sum-of-Squares Error Function

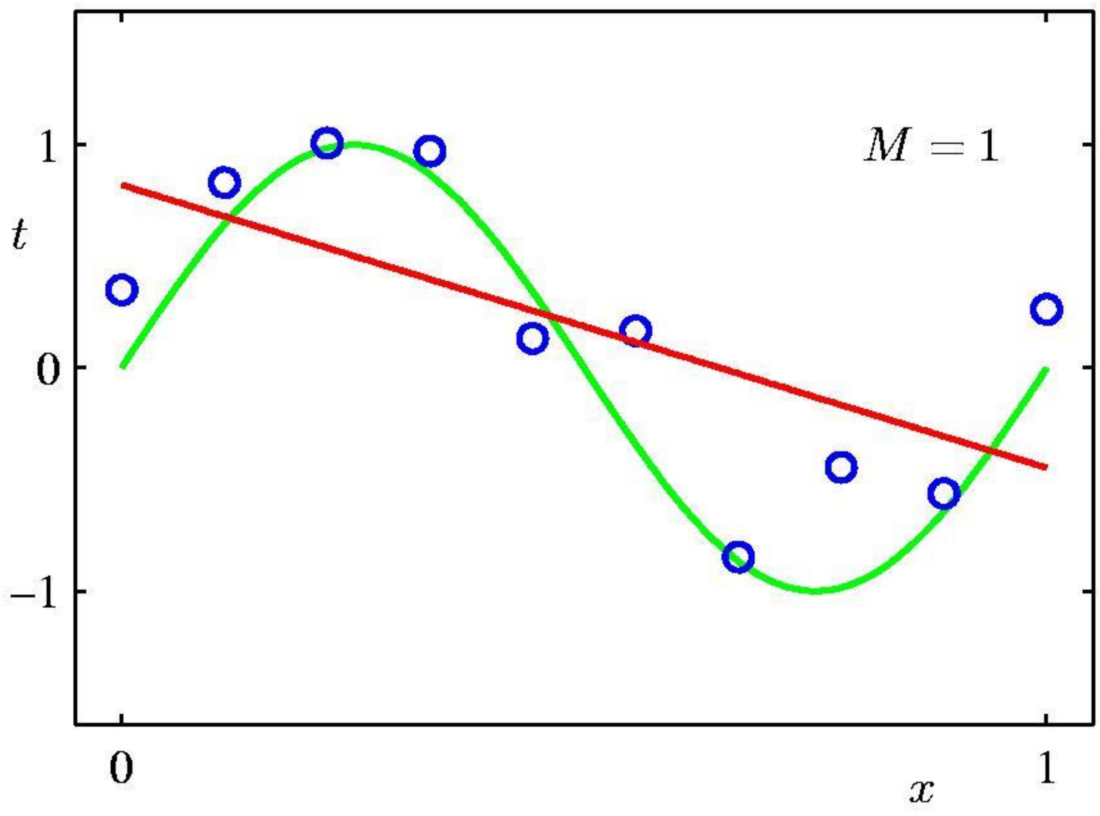



$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{y(x_n, \mathbf{w}) - t_n\}^2$$

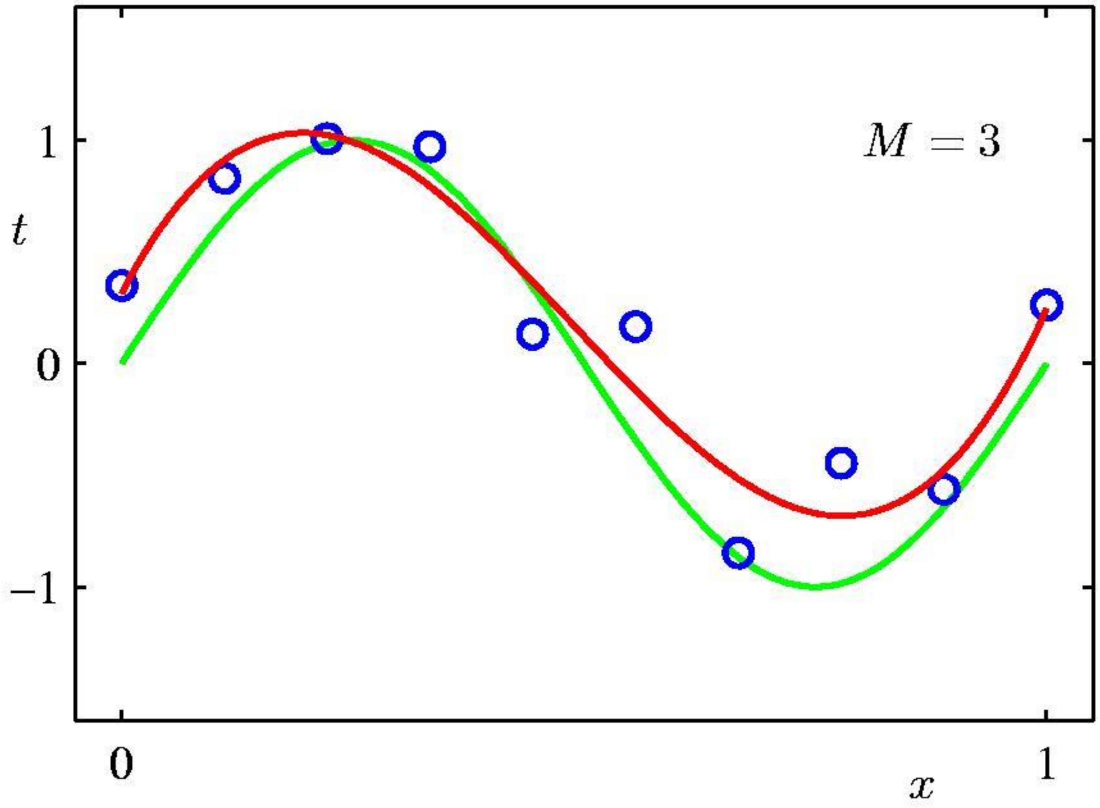
# $0^{th}$ Order Polynomial



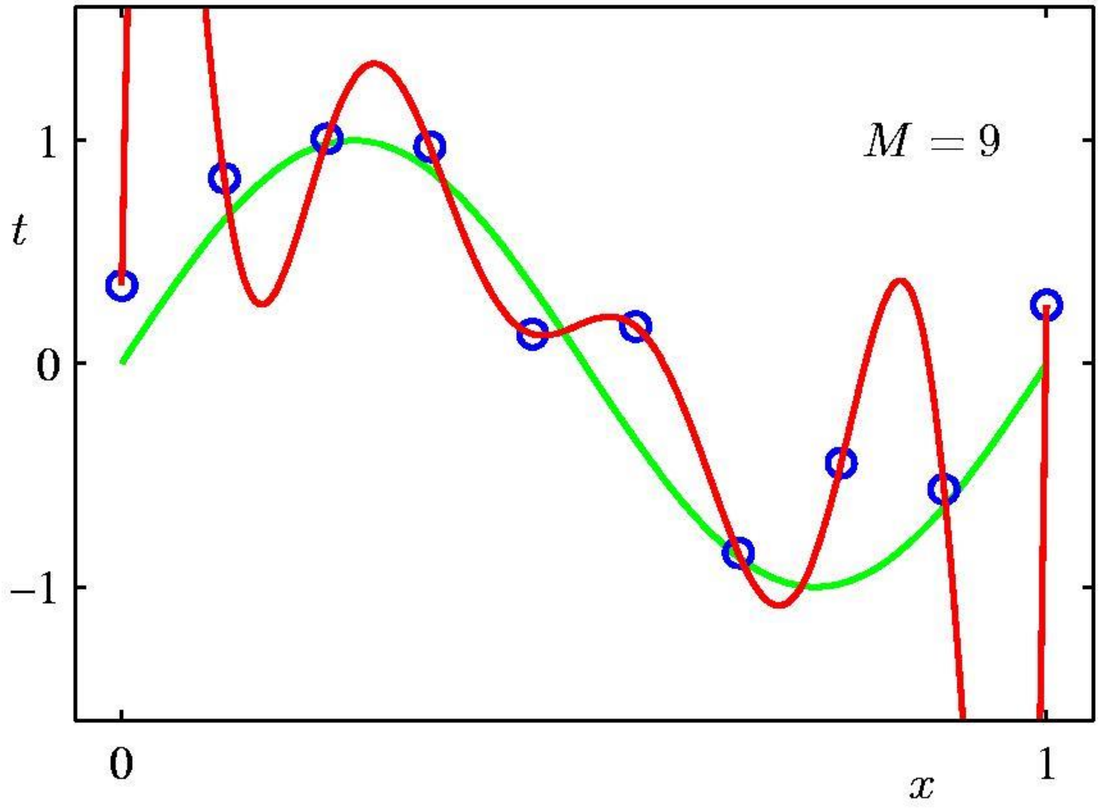
# $1^{st}$ Order Polynomial



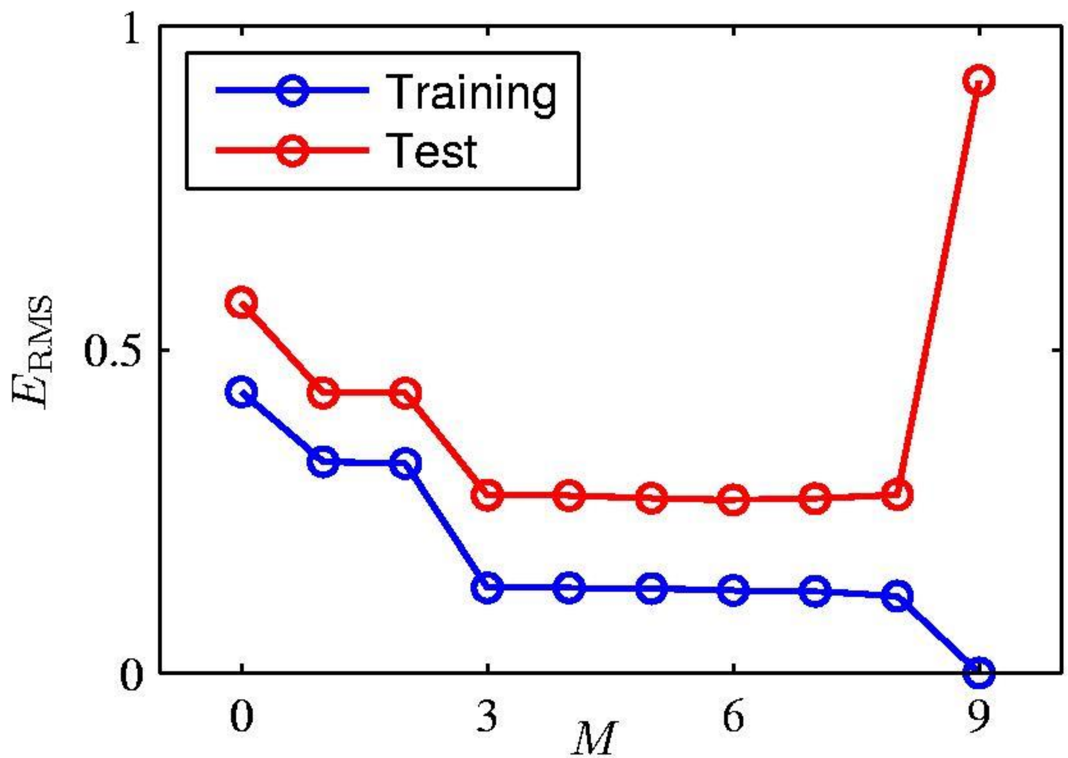
# 3rd Order Polynomial



# 9th Order Polynomial



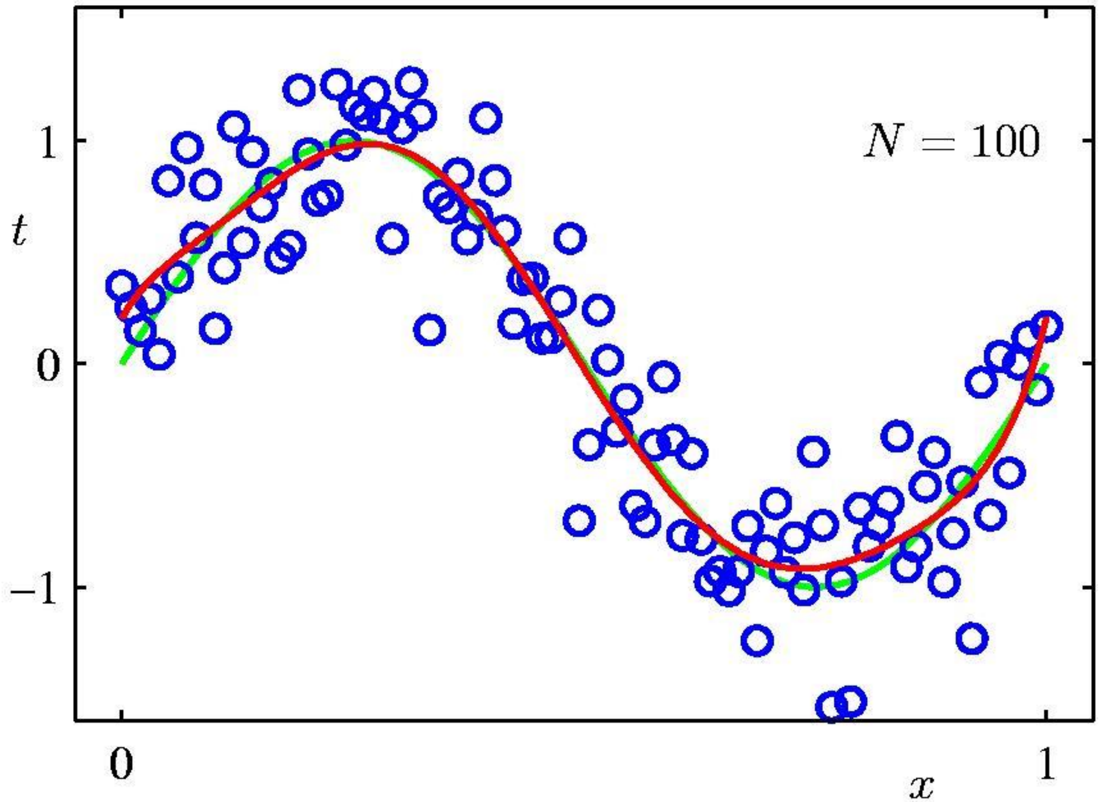
# Over-fitting




Root-Mean-Square (RMS) Error: $E_{\mathrm{RMS}} = \sqrt{2E(\mathbf{w}^{\star})/N}$

# Data Set Size: $N = 100$

9th Order Polynomial

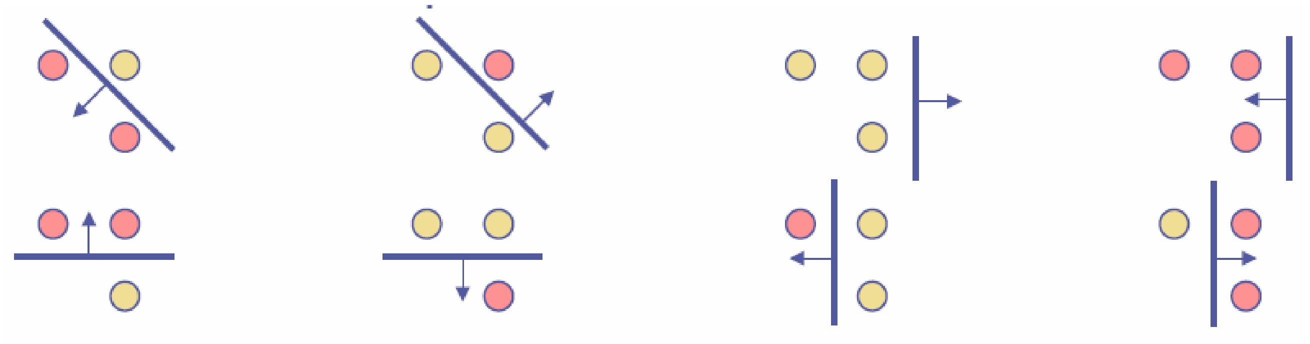


# Structural Risk Minimization

- VC(Vapnik-Chervonenkis )-dimension:

  Maximum number of points that can be labeled in all possible way
- VC dimension of linear classifiers in *N* dimensions

  is $h$=$N$+1 (= #of weights, $n_w$), cf.) MLP: O($n_w{}^2$ )



- Measure of Complexity of a classifier
- **Minimizing VC dim. == Minimizing Complexity**

# Structural Risk Minimization

- Generalization Ability

$$p(|R(\theta_o) - R_{emp}(\theta_{emp})| > \varepsilon) < \left(\frac{2eN}{h}\right)^h \exp(-\varepsilon^2 N) = \alpha$$

  - N: Training sample size
  - h: VC-dimension
  - e: Natural logarithm
- Generalization bound: $p\left(\left|R - R_{emp}\right| < \varepsilon\right) > 1 - \alpha$

$$R(\theta_o) - \varepsilon < R_{emp}(\theta_{emp}) < R(\theta_o) + \varepsilon, \quad with \quad p = 1 - \alpha$$

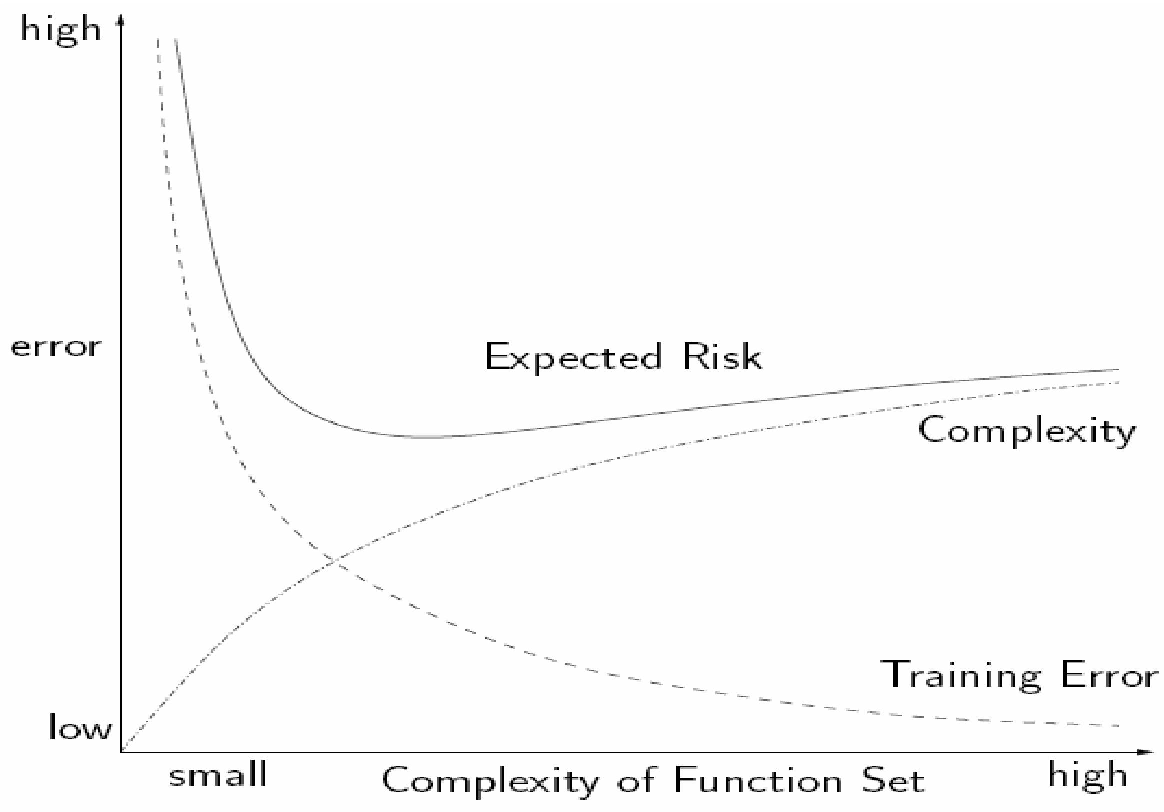
- Confidential interval

$$\varepsilon(N, h, \alpha) = \sqrt{\frac{h}{N}\left(\ln\frac{2N}{h} + 1\right) - \frac{1}{N}\ln\alpha}$$

*Vapnik, Vladimir (2000). The nature of statistical learning theory. Springer.*

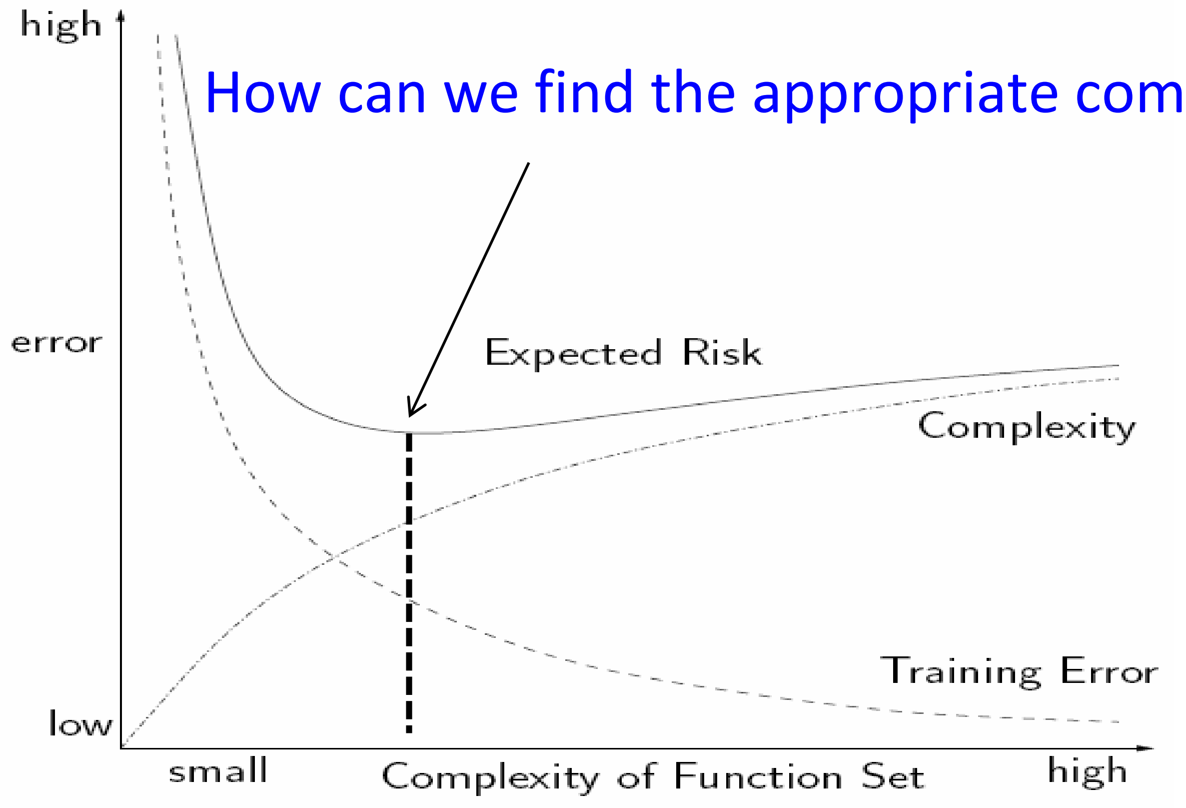
# Structural Risk Minimization

- For fixed training samples N



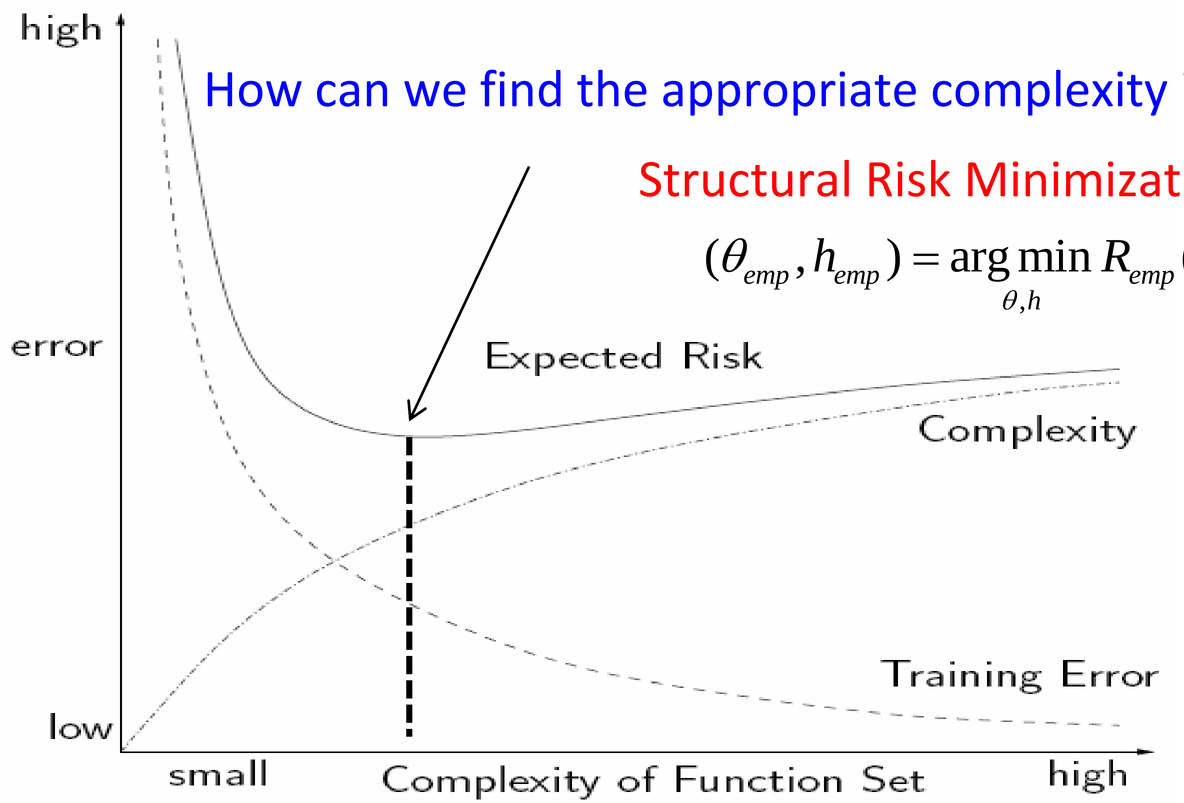
# Structural Risk Minimization

For fixed training samples N



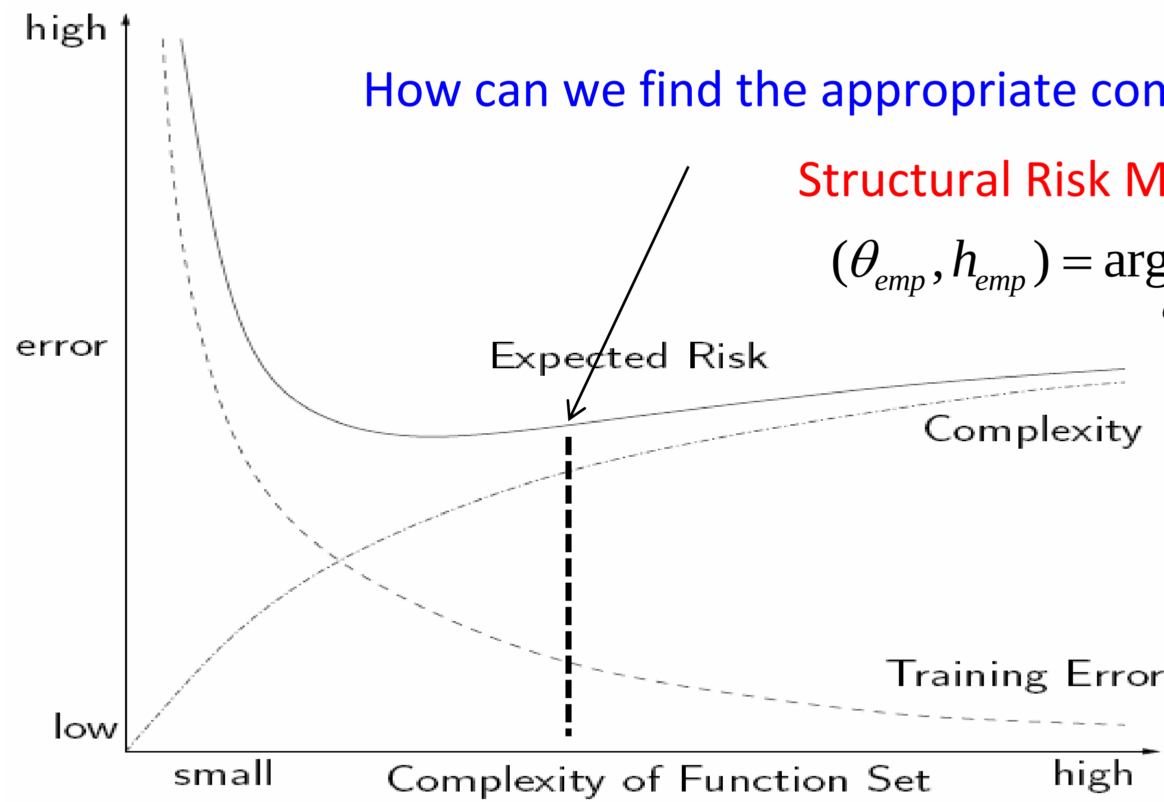
# Structural Risk Minimization

- For fixed training samples N



# Structural Risk Minimization

- For fixed training samples N




How can we find the appropriate complexity ?

Structural Risk Minimization

$$(\theta_{emp}, h_{emp}) = \arg\min_{\theta, h} R_{emp}(\theta, h)$$

SVM

Pruning

Regularization

Drop out

Use of pre-trained Net

# Interim Summary

- Learning rules and models : Parametric Optimization
- Statistical nature of learning

$$\inf_{\theta \in \Theta(c)} R_{emp}(\theta) \xrightarrow{P} \inf_{\theta \in \Theta(c)} R(\theta), \quad as \quad N \to \infty$$

- Empirical risk minimization

$$E(\theta) = \frac{1}{2} \sum_{i=1}^{N} (d_i - g(x_i, \theta))^2 = R_{emp}(\theta)$$

- Structural risk minimization

$$p(| R(\theta_o) - R_{emp}(\theta_{emp}) | > \varepsilon) < \left( \frac{2eN}{h} \right)^h \exp(-\varepsilon^2 N) = \alpha$$

# Learning Rules

- Gradient descent method: Error backpropagation method
- Gauss newton method
- Levenberg–Marquardt method
- Newton (-Raphson) method
- Support Vector Machine
- Parametric PDF estimation: MLE, MAP, Bayesian Learning
- Non-parametric estimation: EM, MCMC
- Boltzman Machine: MAP, Boltzman distribution, MLP, unsupervised L.
- Bayesian Learning: MCMC, Variational Inference

# Learning Models and Rules (II)

**Jin Young Choi**
Seoul National University

# Outline

- Gradient descent method
  - Error backpropagation method
- Gauss newton method
- Levenberg–Marquardt method
- Newton (-Raphson) method

# Empirical Risk Minimization

- Empirical Risk Functional

$$E(\theta) = \frac{1}{2}\sum_{i=1}^{N} (d_i - g(x_i, \theta))^2 = R_{emp}(\theta)$$

$$E(\theta) = p_i \ln q_i(W)$$

$$E(\theta) = \sum_i [p_i \ln q_i(W) + (1 - p_i) \ln(1 - q_i(W))]$$

- Learning Goal $\quad Find \quad \theta^* = \arg\min_{\theta} E(\theta)$

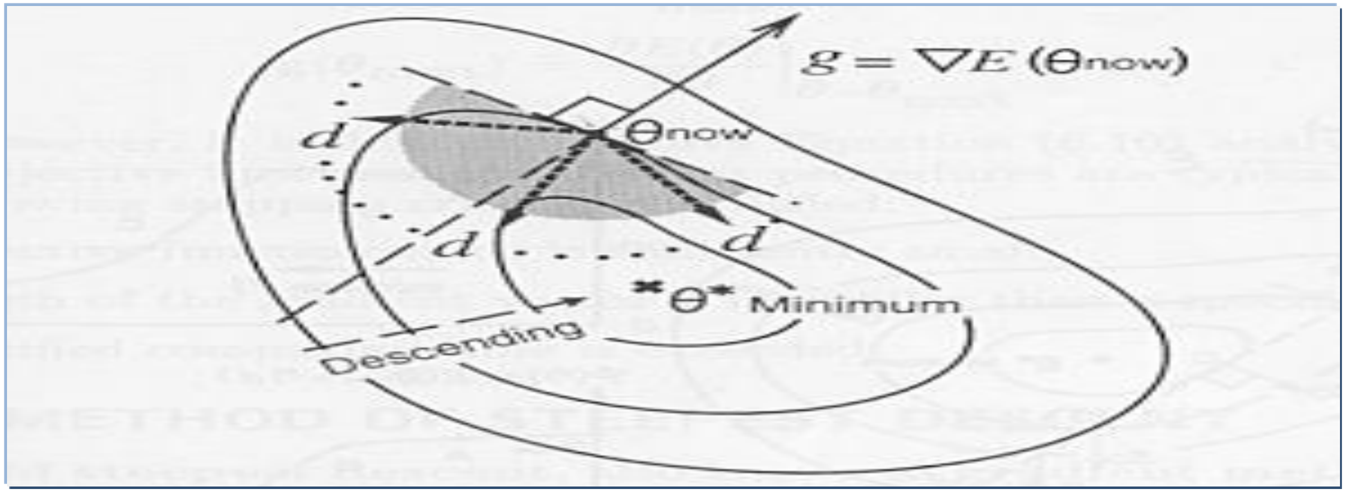
- Convexity
  - Generally, $E(\theta)$ is not convex.
  - For linear case, $E(\theta)$ is convex.

# Gradient Descent Methods

- Gradient Descent Update Rule (Steepest Descent for *G*=I)

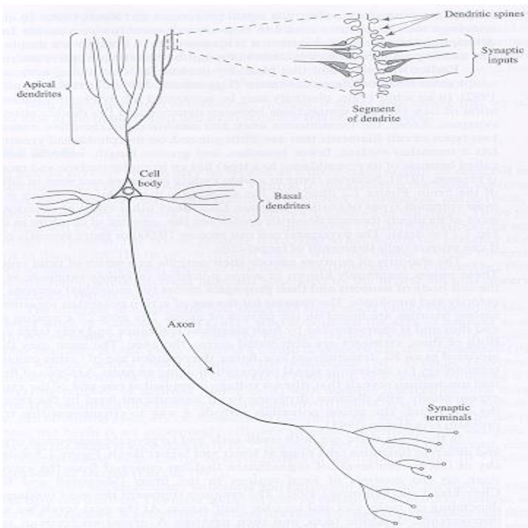
$$\theta(k+1) = \theta(k) - \eta(k) G \nabla E(\theta(k)), \quad G > 0$$

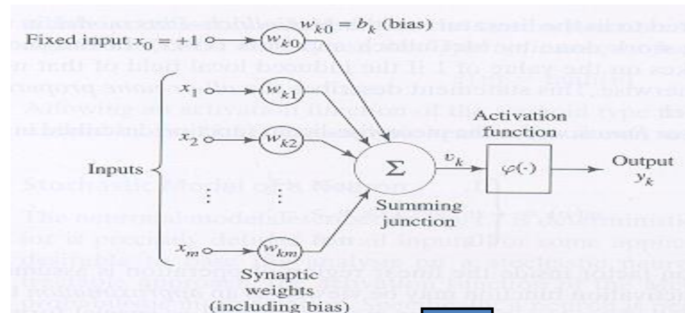
$$\nabla E(\theta) \overset{def}{=} \left[ \frac{\partial E(\theta)}{\partial \theta_1}, \frac{\partial E(\theta)}{\partial \theta_2}, \ldots, \frac{\partial E(\theta)}{\partial \theta_n} \right]^t$$

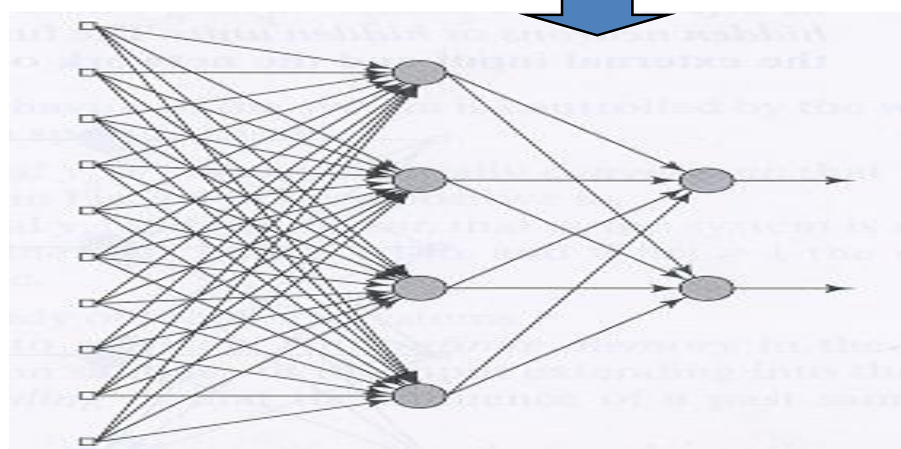


# Backpropagation Learning Rule

- Feedforward Neural Networks



# Backpropagation Learning Rule

- Empirical Risk Function:

$$E_d(w) \equiv \frac{1}{2} \sum_{k \in outputs} (t_k - o_k)^2$$

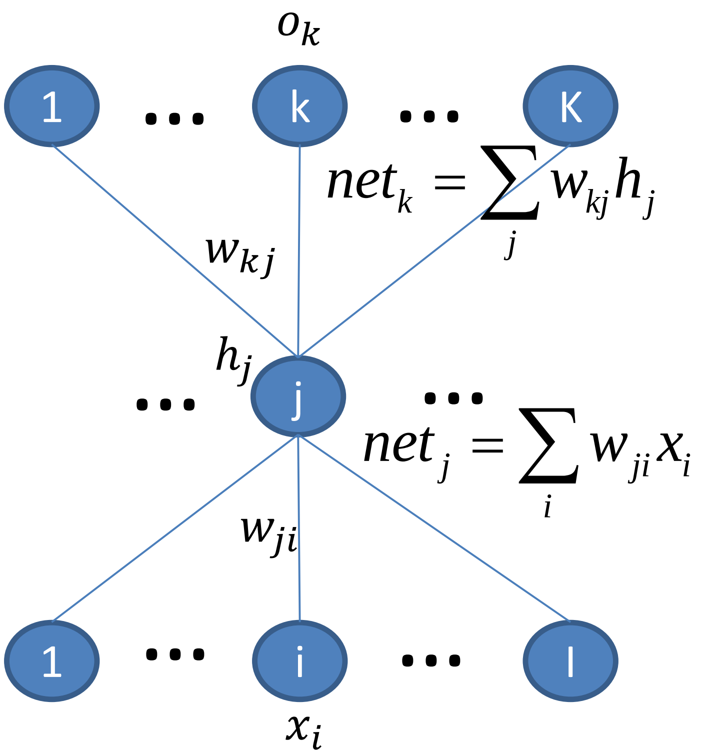
$$E(w) = \frac{1}{2} ||t - o(x, w)||^2$$

- Gradient descent for output layer:

$$\Delta w_{kj} = -\eta \frac{\partial E_d}{\partial w_{kj}}$$

- Chain rule:

$$\frac{\partial E_d}{\partial w_{kj}} = \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E_d}{\partial net_k} h_j$$



# Backpropagation Learning Rule

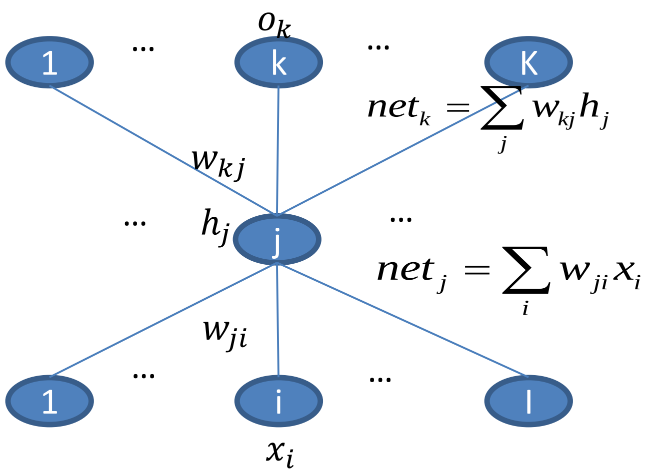
- To learn the regression problem, the linear activation function was used and the sum-of-squares error function was used as the loss function. Let's define the loss function as $E(w) = \frac{1}{2}||o(x,\ w) - t\ ||^2$. At this time, the $k$-th value of $o(x,\ w)$ is defined by $o_k = net_k = w_k^T h = \sum_j w_{kj} h_j$. Then find $\frac{\partial \mathrm{E}}{\partial net_k}$.

Sol.)

Since $E(w) = {}^1/_2 \sum_k (net_k - t_k)^2$, $\frac{\partial \mathrm{E}}{\partial net_k} = net_k - t_k = o_k - t_k = -(t_k - o_k) = -\ \delta_k$.

$$\frac{\partial E_d}{\partial w_{kj}} = \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E_d}{\partial net_k} h_j$$

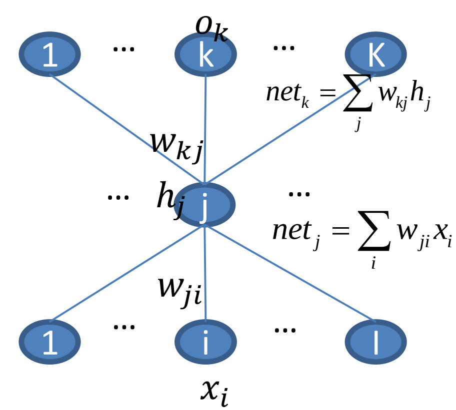
$$\Delta w_{kj} = -\eta \frac{\partial E_d}{\partial w_{kj}} = \eta \delta_k h_j$$



# Backpropagation Learning Rule

- For multi-label classification (ex, output: 0110100), sigmoid activation function is used and the loss is defined by the cross entropy loss function: $E(w) = -\sum_k^K [t_k \log o_k(x,w) + (1-t_k)\log\big(1-o_k(x,w)\big)]$ , where

$o_k = \sigma(net_k) = \frac{1}{1+e^{-net_k}}$ . Then find $\frac{\partial \mathrm{E}}{\partial net_k}$ .




Sol.)

Since $\frac{\partial o_k}{\partial net_k} = \sigma(net_k)\big(1-\sigma(net_k)\big) = o_k(1-o_k)$ ,

$$\frac{\partial \mathrm{E}}{\partial net_k} = -t_k \frac{1}{o_k}\frac{\partial o_k}{\partial net_k} - (1-t_k)\frac{-1}{1-o_k}\frac{\partial o_k}{\partial net_k}$$

$$= -t_k \frac{1}{o_k} o_k(1-o_k) - (1-t_k)\frac{-1}{1-o_k} o_k(1-o_k)$$

$$= -t_k(1-o_k) + (1-t_k)o_k = o_k - t_k = -(t_k - o_k) = -\delta_k$$

$$\frac{\partial E_d}{\partial w_{kj}} = \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E_d}{\partial net_k} h_j$$

$$\Delta w_{kj} = -\eta \frac{\partial E_d}{\partial w_{kj}} = \eta \delta_k h_j$$

# Backpropagation Learning Rule

- For multi-class classification (ex, [0 0 0 1 0 0]), the softmax activation function is used and the loss is defined by the cross entropy loss function: $E(w) = -\sum_i^K t_i \log(o_i(x,w))$, where $o_k(x,w) = \frac{e^{net_k}}{\sum_j e^{net_j}}$.
  The target value $t_k \in \{0,1\}$ is labelled by 1 hot vector. Then find $\frac{\partial E}{\partial net_k}$.

Sol.)

$$\frac{\partial E_n}{\partial net_k} = \frac{\partial}{\partial net_k}\left(-\sum_i^K t_i \log\left(\frac{e^{net_i}}{\sum_j e^{net_j}}\right)\right) = \frac{\partial}{\partial net_k}\left(-\sum_i^K [t_i \log(e^{net_i}) - t_i \log(\sum_j e^{net_j})]\right)$$

$$= \frac{\partial}{\partial net_k}\left(-\sum_i^K [t_i net_i - t_i \log(\sum_j e^{net_j})]\right) = -t_k + \sum_i t_i \frac{e^{net_k}}{\sum_j e^{net_j}}$$

$$= -t_k + \frac{e^{net_k}}{\sum_j e^{net_j}} \sum_i t_i = o_k - t_k = -(t_k - o_k) = -\delta_k$$

$$\frac{\partial E_d}{\partial w_{kj}} = \frac{\partial E_d}{\partial net_k}\frac{\partial net_k}{\partial w_{kj}} = \frac{\partial E_d}{\partial net_k} h_j$$

$$\Delta w_{kj} = -\eta \frac{\partial E_d}{\partial w_{kj}} = \eta \delta_k h_j$$

# Backpropagation Learning Rule

- Empirical Risk Function:

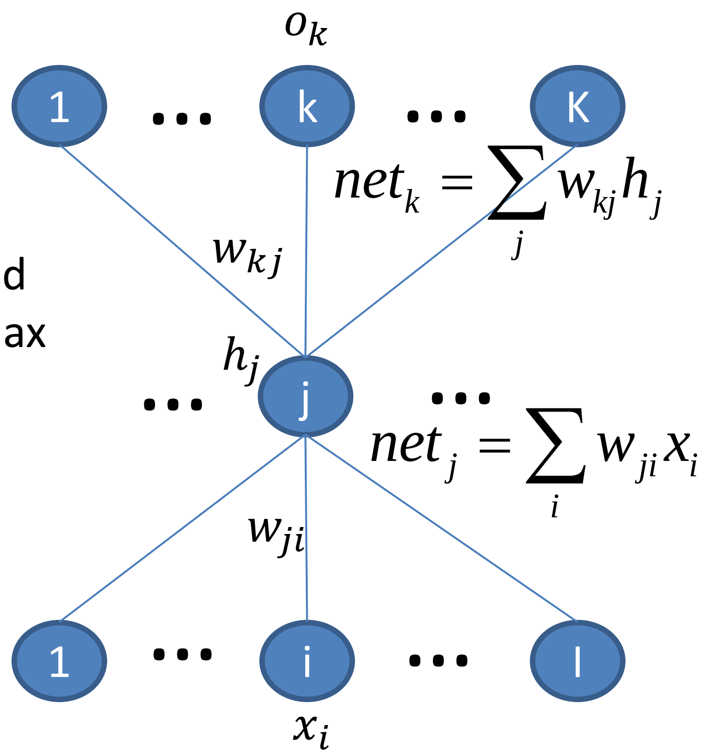
$E_d(w)$

Regression: $L_2$, linear
01001101: cross-entropy, sigmoid
00001000: cross-entropy, soft-max

- Gradient descent for hidden layer:

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

- Chain rule:



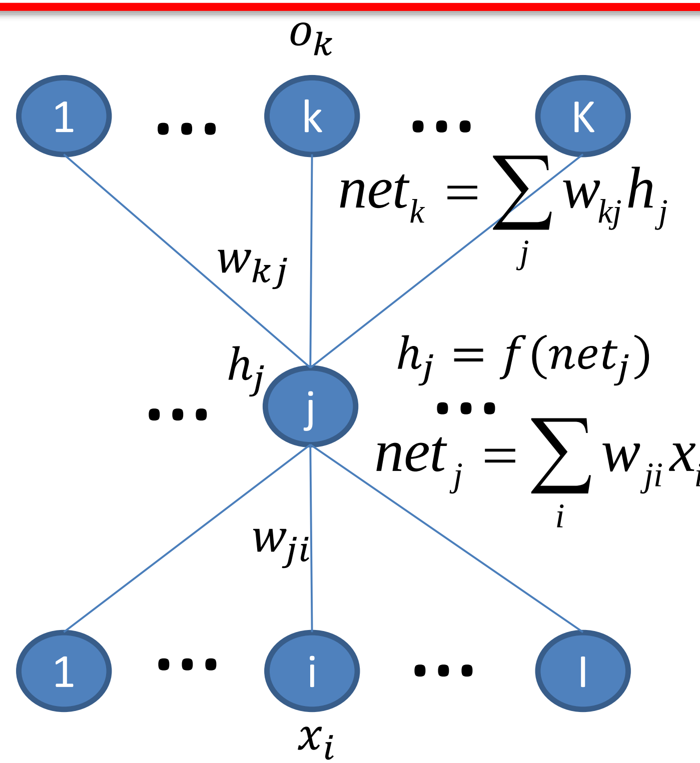

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j}\frac{\partial net_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_i$$

# Backpropagation Learning Rule

- Chain rule:

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial net_j} x_i, \qquad \frac{\partial E_d}{\partial net_k} = -\delta_k$$

$$\begin{aligned}
\frac{\partial E_d}{\partial net_j} &= \sum_{k \in outputs} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial h_j} \frac{\partial h_j}{\partial net_j} \\
&= \sum_{k \in outputs} -\delta_k \frac{\partial net_k}{\partial h_j} \frac{\partial h_j}{\partial net_j} \\
&= \sum_{k \in outputs} -\delta_k w_{kj} \frac{\partial h_j}{\partial net_j} \\
&= \sum_{k \in outputs} -\delta_k w_{kj} f'(net_j) \boxed{= -\delta_j}
\end{aligned}$$




$$\Delta w_{ji} = \eta \delta_j x_i,$$

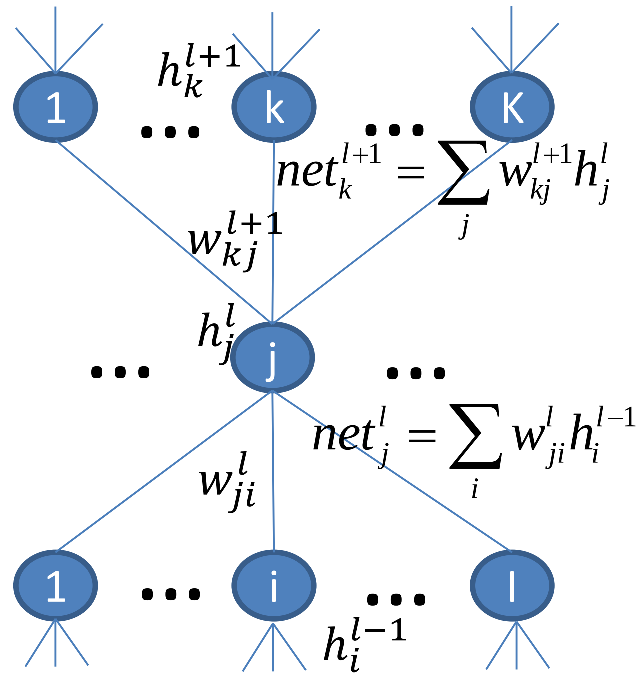
$$\delta_j = f'(net_j) \sum_{k \in outputs} \delta_k w_{kj}$$

# Backpropagation Learning Rule

$$\Delta w_{ji}^{l} = \eta \delta_j^{l} h_i^{l-1},$$
$$\delta_j^{l} = f^{'}(net_j^{l}) \sum_{k \in l+1 layer} \delta_k^{l+1} w_{kj}^{l+1}$$

$$\delta_k^L = -\frac{\partial E_d}{\partial net_k} = t_k - o_k$$

Regression: $L_2$, linear
01001101: cross-entropy, sigmoid
00001000: cross-entropy, soft-max



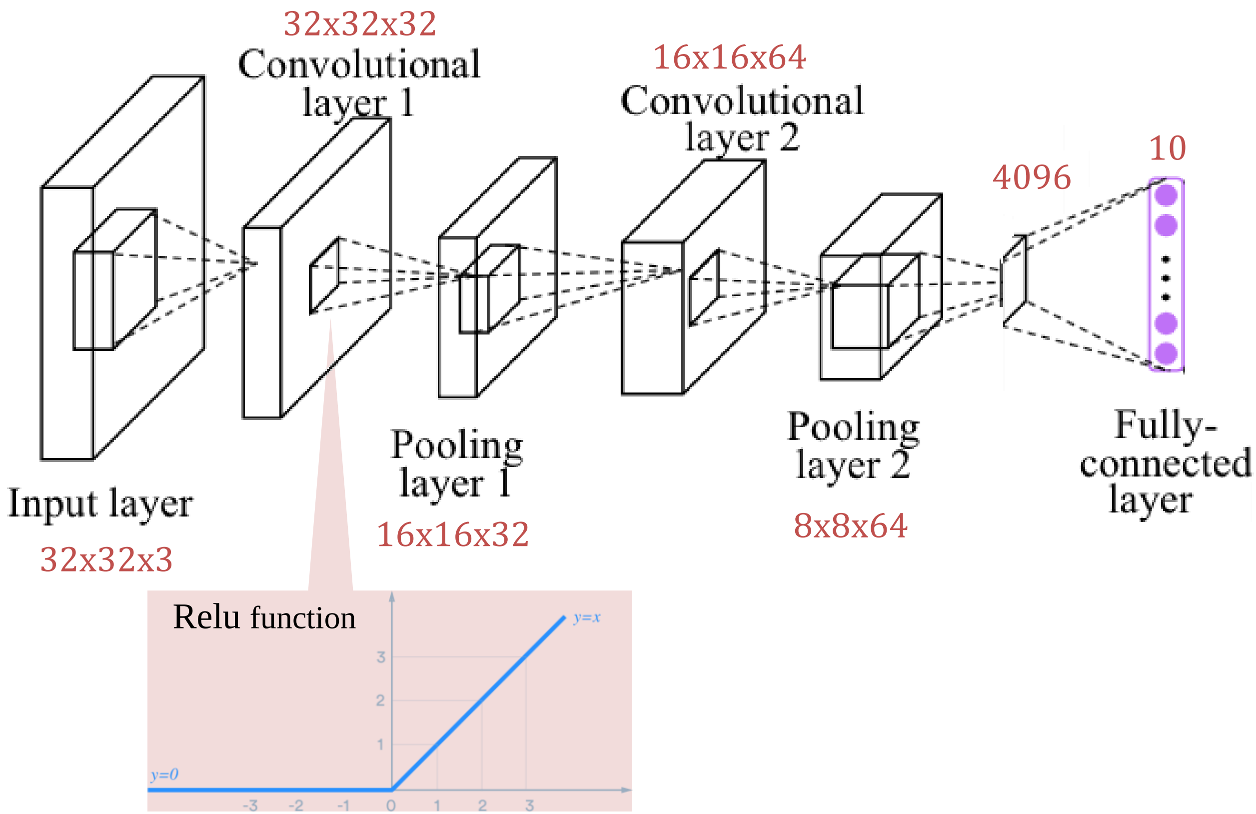

## Matrix Form (Backward. EBP)

$$\Delta W^l = \eta \delta^l {h^{l-1}}^T + \rho \Delta W^{l(old)}$$
$$\delta^l = Diag[f'(net^l)] {W^{l+1}}^T \delta^{l+1}$$

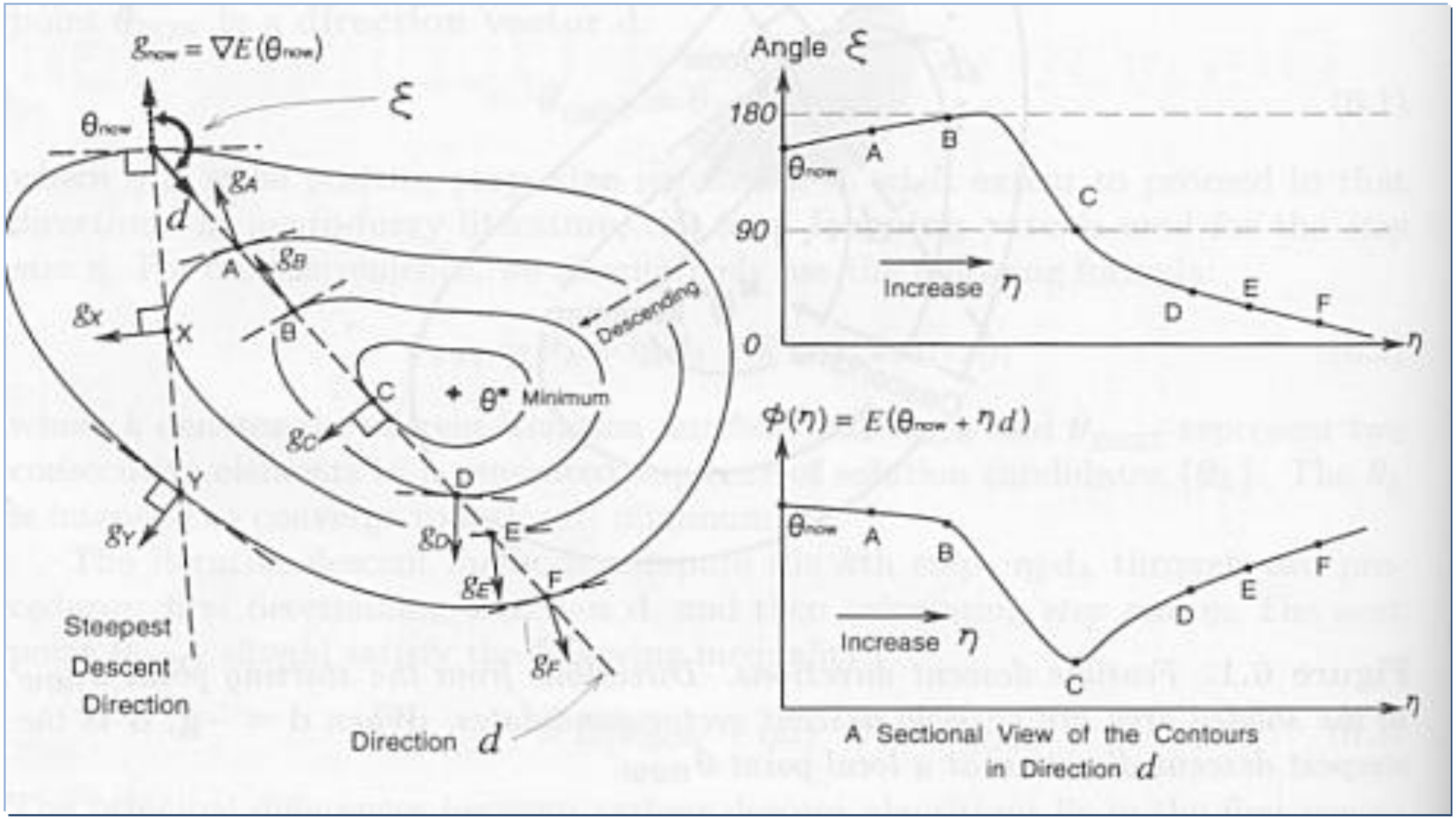
## Matrix Form (Forward)

$$h^0 = x$$
$$h^{l+1} = Diag[f] \circ W^{l+1} h^l$$

# Two Layer Convolutional Neural Network



# Gradient Descent Methods



# Gradient Descent Methods

- Optimal Learning Rate
  - Necessary Condition

$$\nabla^T E\left(\theta_{next}\right)\nabla E\left(\theta_{now}\right)=0,$$
$$\nabla^T E\left(\theta_{now}-\eta\nabla E\left(\theta_{now}\right)\right)\nabla E\left(\theta_{now}\right)=0$$

- Learning Rate Search Methods
  - Initial Bracketing
  - Line Searching
  - Secant Method (Approximate Newton Method)
  - Bisection Method
  - Golden section search method

# Newton(-Raphson) Method

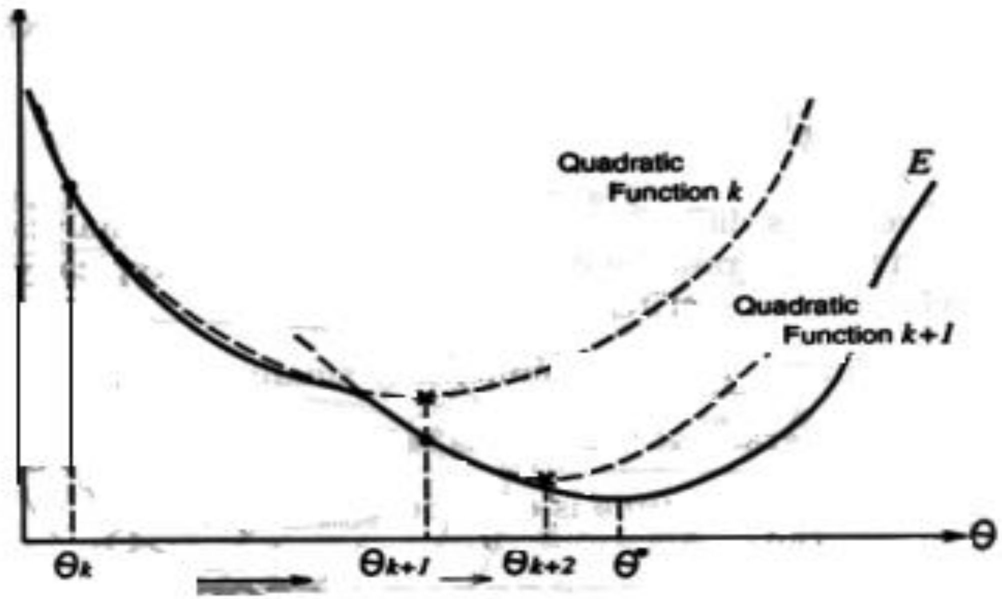
- Principle: The descent direction is determined by using the <u>second derivatives</u> of the objective function $f(\theta)$ if available

- If the starting position $\theta_{now}$ is sufficient close to a local minimum, the objective function $f(\theta)$ can be approximated by a quadratic form:

$$E(\theta) = E(\theta_{now}) + \Lambda^T(\theta - \theta_{now}) + \frac{1}{2}(\theta - \theta_{now})^T \mathbf{H}(\theta - \theta_{now})$$

$$where \quad \mathbf{H} = \nabla^2 E(\theta_{now}), \quad \Lambda = \nabla E(\theta_{now}).$$

# Newton(-Raphson) Method




$$E(\theta) \cong E(\theta_{now}) + \Lambda^T(\theta - \theta_{now}) + \frac{1}{2}(\theta - \theta_{now})^T \mathbf{H}(\theta - \theta_{now})$$

$$where \quad \mathbf{H} = \nabla^2 E(\theta), \quad \Lambda = \nabla E(\theta).$$

# Newton (-Raphson) Method

- Since the equation defines a quadratic function
  - its minimum can be determined by differentiating & setting to 0.

$$E(\theta) \cong E(\theta_{now}) + \Lambda^T(\theta - \theta_{now}) + \frac{1}{2}(\theta - \theta_{now})^T \mathbf{H}(\theta - \theta_{now})$$

$$\Lambda + \mathbf{H}(\theta_{next} - \theta_{now}) = 0$$

$$\theta_{next} = \theta_{now} - \mathbf{H}^{-1}\Lambda$$

$cf)$

$$\theta(k+1) = \theta(k) - \eta(k)\boxed{G}\nabla E(\theta(k)), \quad G > 0$$

# Gauss Newton Method

- Key idea: Not to use Hassian matrix, we use linearized approximation of learning model.

- $E(\theta) = \frac{1}{2}\|d - g(x,\theta)\|^2$

  $g(x,\theta) \approx g(x,\theta_{now}) + J^T(\theta - \theta_{now})$,

  where $J = {}^{dg(x,\theta)}/_{d\theta}\big|_{\theta=\theta_{now}}$

- $E(\theta) = \frac{1}{2}\|d - g(x,\theta_{now}) - J^T(\theta - \theta_{now})\|^2$

# Gauss Newton Method

- Since the function is quadratic for $\theta$,
  - Its minimum can be determined by differentiating & setting to 0.
  - $E(\theta) = \frac{1}{2}\|d - g(x,\theta_{now}) - J^T(\theta - \theta_{now})\|^2$
  - $\nabla E(\theta) = -J\big(d - g(x,\theta_{now}) - J^T(\theta - \theta_{now})\big) = 0$
- Update Rule
  - $\theta_{next} = \theta_{now} + (JJ^T)^{-1}J(d - g(x,\theta_{now}))$
  - $\theta_{next} = \theta_{now} - (JJ^T)^{-1}\nabla E(\theta_{now})$

    $[\because \nabla E(\theta_{now}) = -J\big(d - g(x,\theta_{now})\big)]$

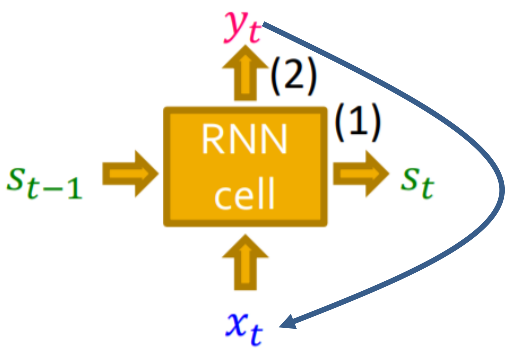
$cf)$

$$\theta(k+1) = \theta(k) - \eta(k) G \nabla E(\theta(k)), \quad G > 0$$

# RNN: Recurrent Neural Networks

Recurrent Neural Networks

- $s_t$: State of RNN after time $t$
- $x_t$: Input to RNN at time $t$
- $y_t$: Output of RNN at time $t$
- $W, U, V$: parameter matrices, $\sigma(\cdot)$ : non-linear activation function




(1) $s_t = \sigma(W\, s_{t-1} + U\, x_t\ )$

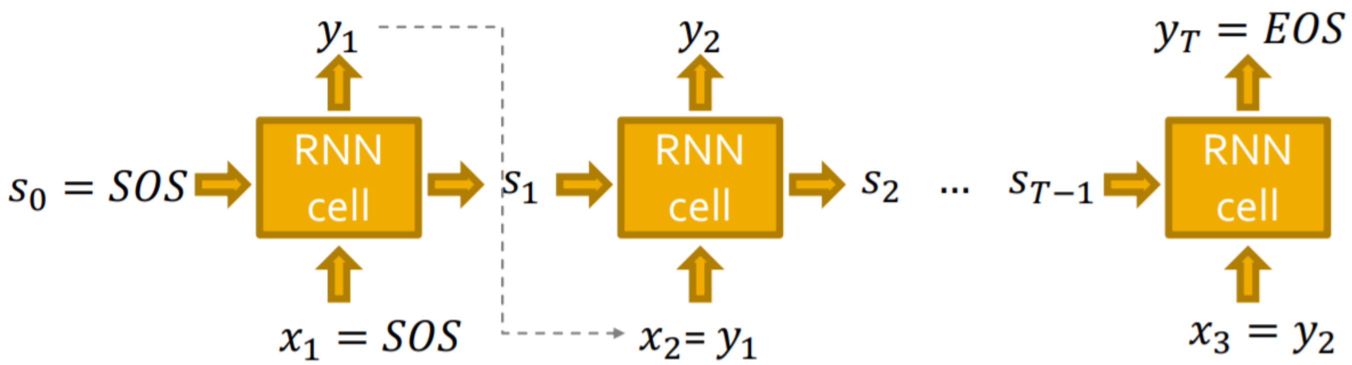
(2) $y_t = V \cdot s_t$

- More expressive cells: GRU(Gated Recurrent Unit), LSTM(Long-Short-Term Memory), etc.

# **RNN:** Recurrent Neural Networks

**RNN** for Sequence Generation

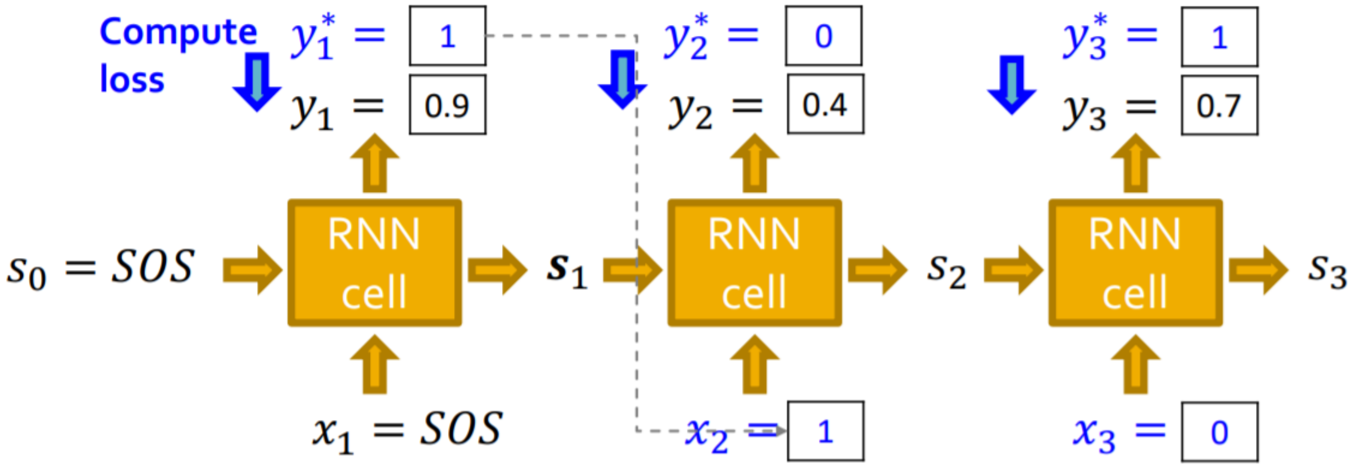
- **Q:** How to use RNN to generate sequences?
- **A:** Let $x_{t+1} = y_t$
- **Q:** How to initialize $s_0$, $x_1$? When to stop generation?
- **A:** Use start/end of sequence token (SOS, EOS)



# **RNN:** Recurrent Neural Networks

Training **RNN**

- We observe a sequence $y^*$ of edges [1,0,…]
- Principle: Teacher Forcing -- Replace input and output by the real sequence



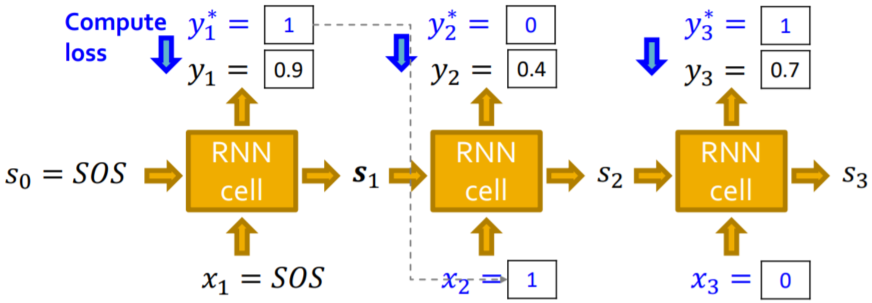
# **RNN:** Recurrent Neural Networks

Training **RNN**

- Loss $L$: Binary cross entropy
- Minimize:

$$L = -\sum_i [y_i^* \log y_i + (1 - y_i^*) \log(1 - y_i)]$$

- If $y_i^* = 1,$ we minimize $-\log y_i$, making $y_i$ higher to approach 1
- If $y_i^* = 0,$ we minimize $-\log(1 - y_i)$, making $y_i$ lower to approach 0
- $y_i$ is computed by RNN, this loss will adjust RNN parameters accordingly, using back propagation!



# Interim Summary

$$\theta_{next} = \theta_{now} - \eta_k \boxed{G} \nabla E(\theta_{now})$$

**Gradient Descent**

$$\theta_{next} = \theta_{now} - \eta_k \boxed{(JJ^t)^{-1}} \nabla E(\theta_{now})$$

**Gauss Newton**

$$\theta_{next} = \theta_{now} - \eta_k \boxed{(\varepsilon \mathbf{I} + JJ^t)^{-1}} \nabla E(\theta_{now})$$

**Levenberg**

$$\theta_{next} = \theta_{now} - \eta_k \boxed{(\varepsilon Diag(JJ^t) + JJ^t)^{-1}} \nabla E(\theta_{now})$$

**Levenberg–Marquardt**

$$\theta_{next} = \theta_{now} - \eta_k \boxed{\mathbf{H}^{-1}} \nabla E(\theta_{now})$$

**Newton (-Raphson)**

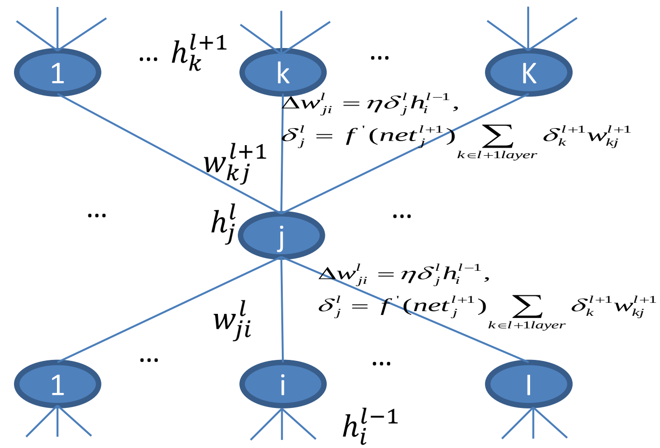
# Interim Summary

$$\Delta w_{ji}^{l} = \eta \delta_j^l h_i^{l-1},$$

$$\delta_j^l = f'(net_j^l) \sum_{k \in l+1 layer} \delta_k^{l+1} w_{kj}^{l+1}$$

$$\delta_k^L = -\frac{\partial E_d}{\partial net_k} = t_k - o_k$$

**Error Backpropagation Rule (Gradient Descent)**




## Matrix Form (Backward. EBP)

$$\Delta W^l = \eta \delta^l {h^{l-1}}^T + \rho \Delta W^{l(old)}$$

$$\delta^l = Diag[f'(a^l)] {W^{l+1}}^T \delta^{l+1}$$

## Matrix Form (Forward)

$$h^0 = x$$

$$h^{l+1} = Diag[f] \circ W^{l+1} h^l$$