Machine Learning
CS 6830

Lecture 02

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

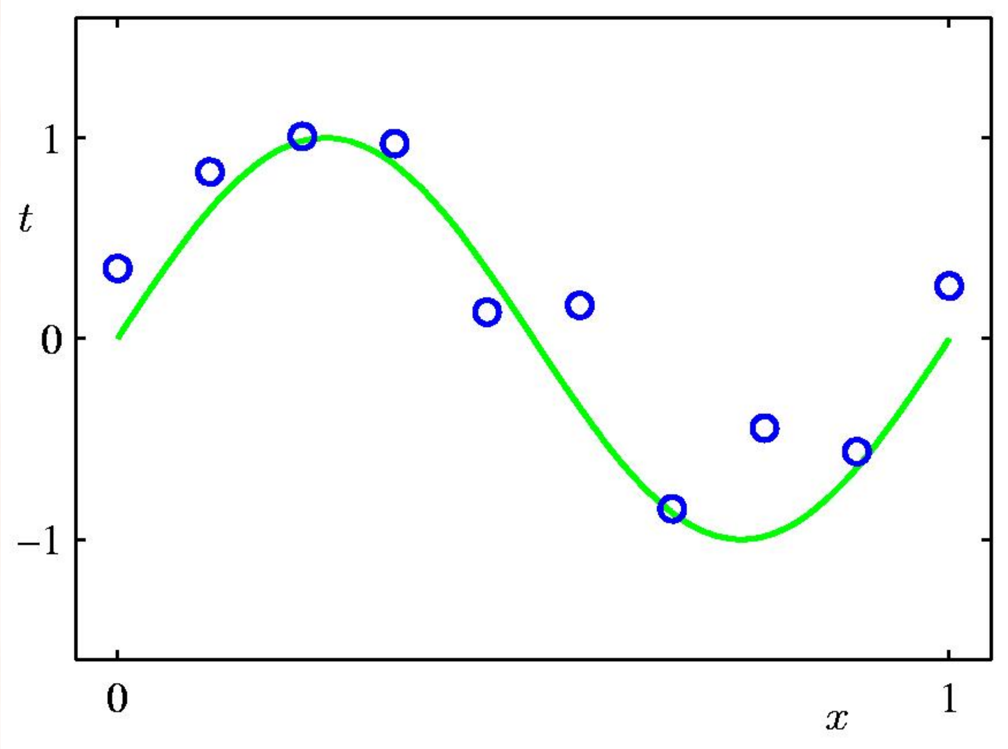
Supervised Learning

- Task = learn a function $y : X \rightarrow T$ that maps input instances $x \in X$ to output targets $t \in T$:
  - Classification:
    - The output $t \in T$ is one of a finite set of discrete categories.
  - Regression:
    - The output $t \in T$ is continuous, or has a continuous component.

- Supervision = set of training examples:

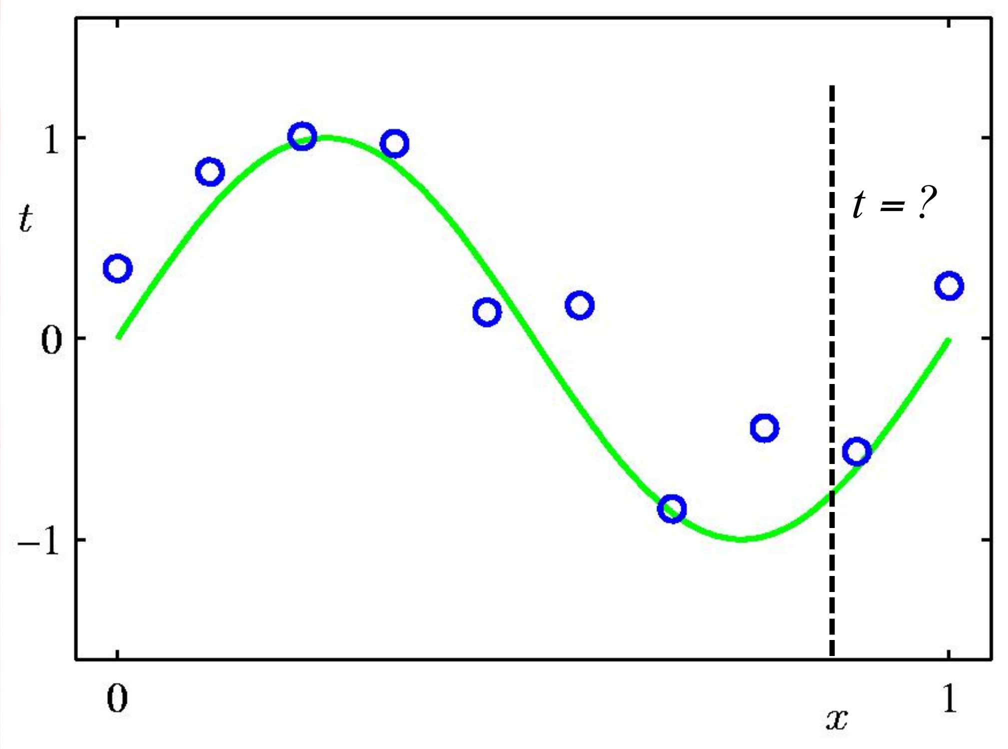
  $(x_1, t_1), (x_2, t_2), \ldots (x_n, t_n)$

Regression: Curve Fitting



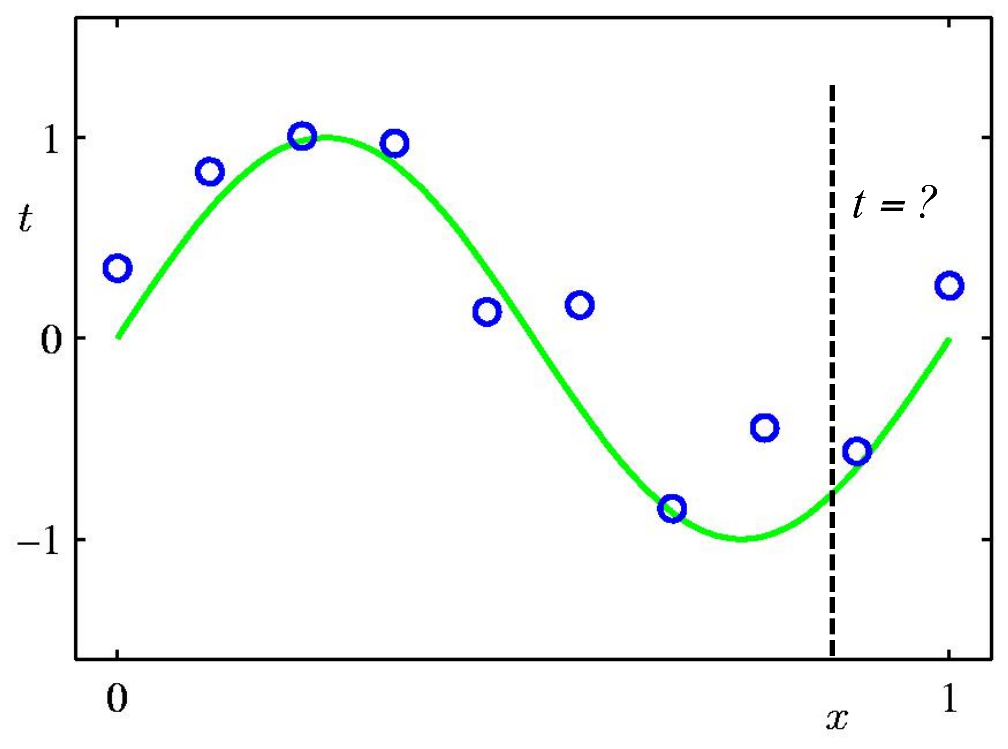
- Training: examples (x_1, t_1), (x_2, t_2), … (x_n, t_n)

Regression: Curve Fitting



- Testing: for arbitrary (unseen) instance $x \in X$, compute target output $y(x) = t \in T$.

Polynomial Curve Fitting



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

parameters *features*

Polynomial Curve Fitting

- Learning = finding the "right" parameters $\mathbf{w}^T = [w_0, w_1, \ldots, w_M]$
  - Find $\mathbf{w}$ that minimizes an *error function* $E(\mathbf{w})$ which measures the misfit between $y(x_n, \mathbf{w})$ and t_n.
  - Expect that $y(x,\mathbf{w})$ performing well on training examples $x_n \Rightarrow y(x,\mathbf{w})$ will perform well on arbitrary test examples $x \in X$.

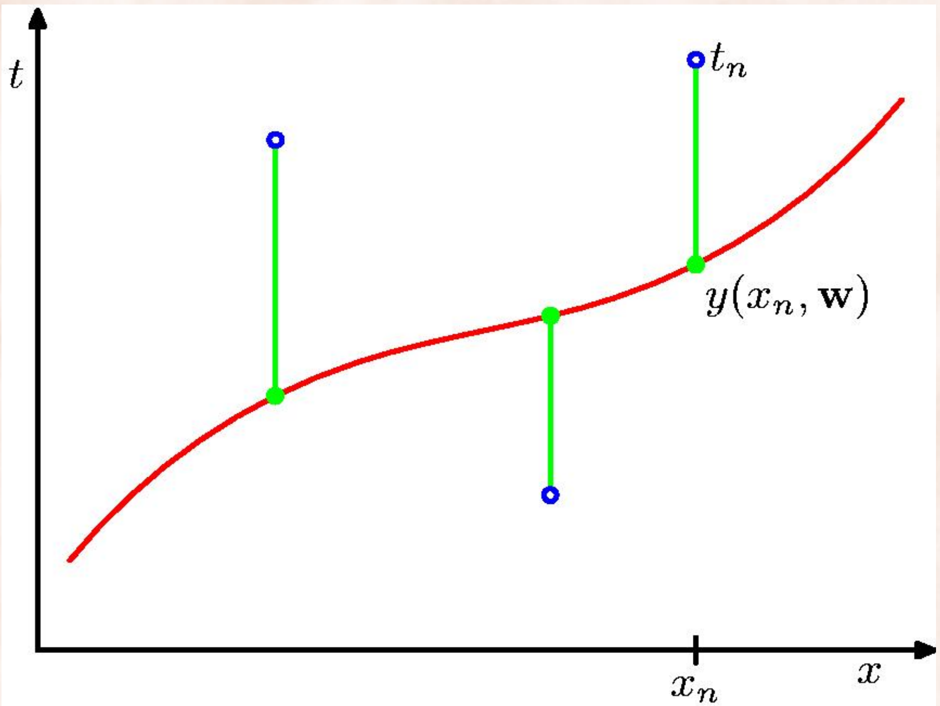
Inductive Learning Hyphotesis

- Sum-of-Squares error function:

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

why squared?

Sum-of-Squares Error Function



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

- How do we find **w*** that minimizes $E(\mathbf{w})$?

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$$

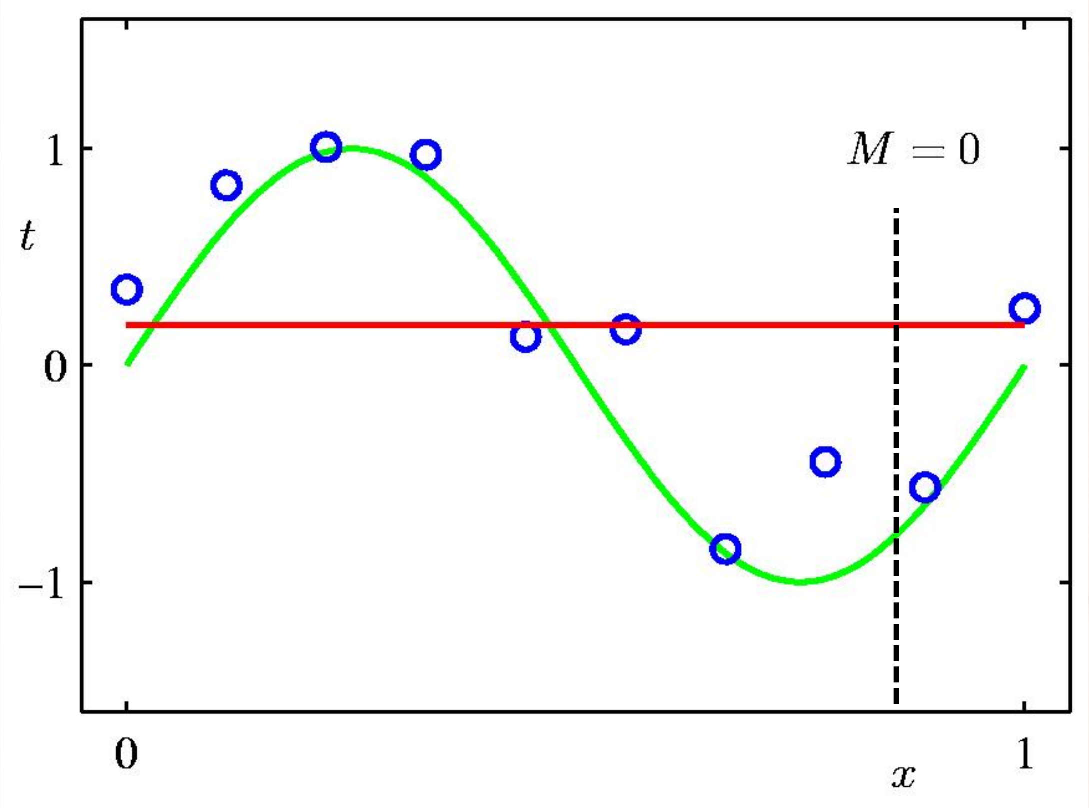
Polynomial Curve Fitting

- *Least Square* solution is found by solving a set of M + 1 linear equations:

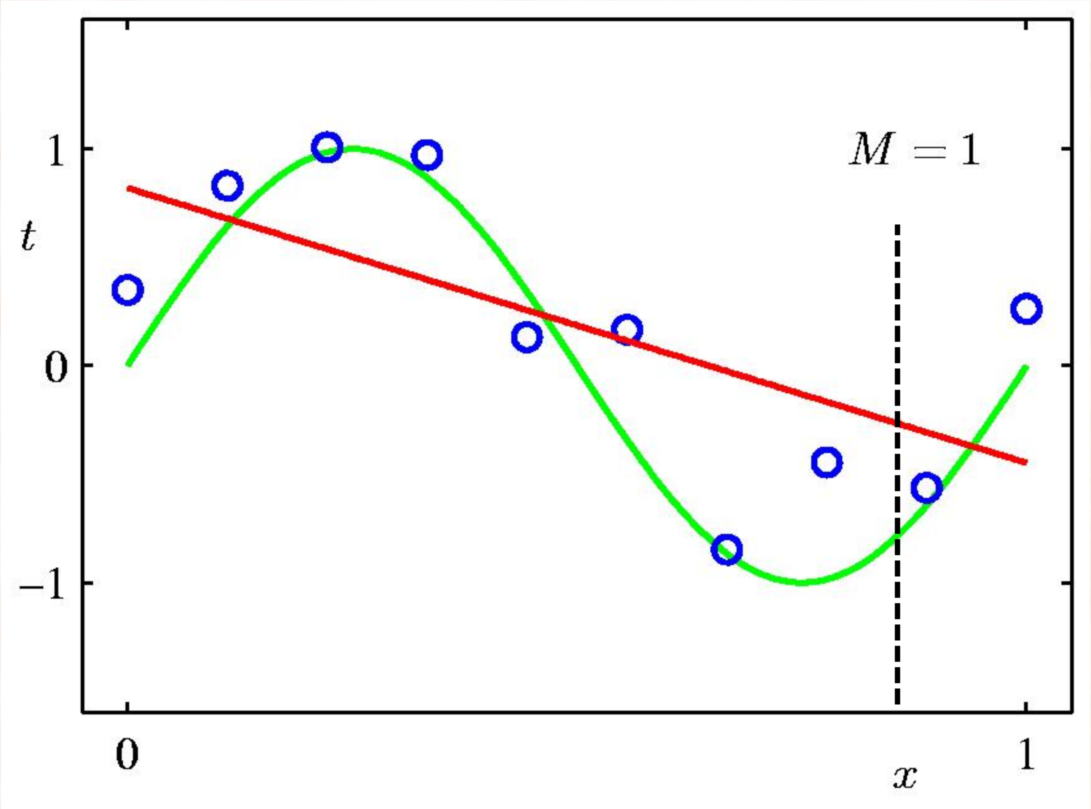
$$\sum_{j=0}^{M} A_{ij} w_j = T_i \text{ , where } A_{ij} = \sum_{n=1}^{N} x_n^{i+j} \text{, and } T_i = \sum_{n=1}^{N} t_n x_n^i$$

- Generalization = how well the parameterized $y(x,\mathbf{w}^*)$ performs on arbitrary (unseen) test instances $x \in X$.
  - Generalization performance depends on the value of M.

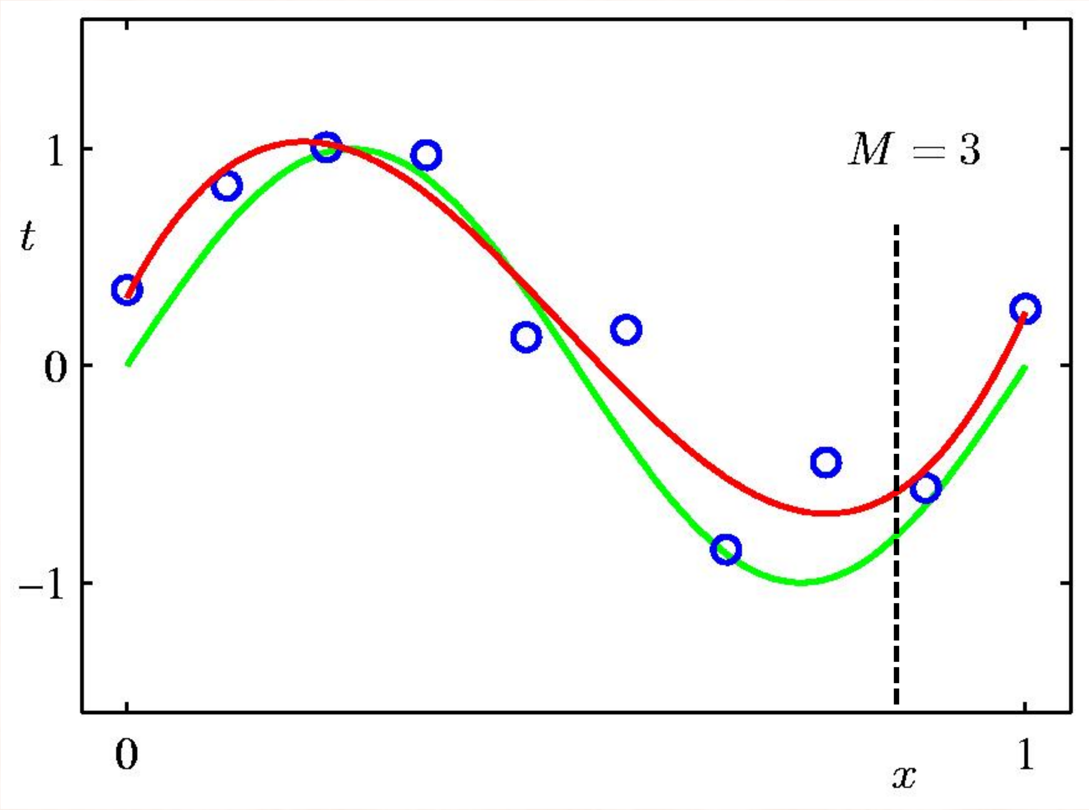
0^{th} Order Polynomial



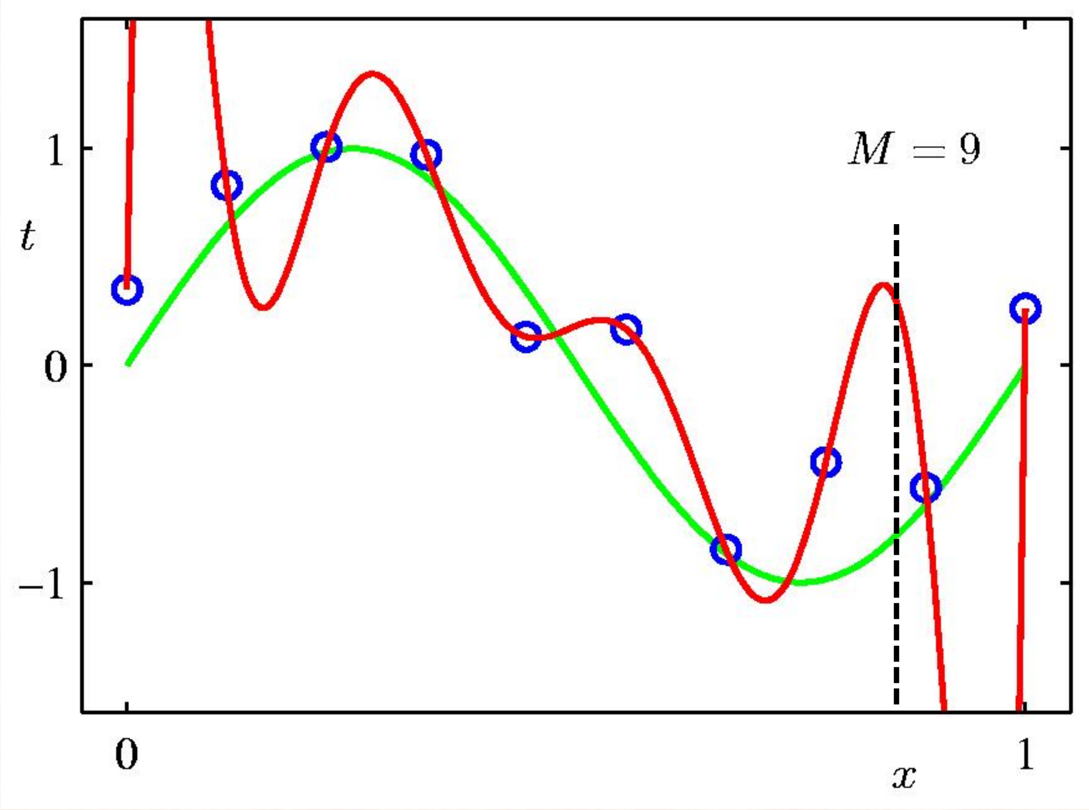
1^{st} Order Polynomial



3^{rd} Order Polynomial



9^{th} Order Polynomial



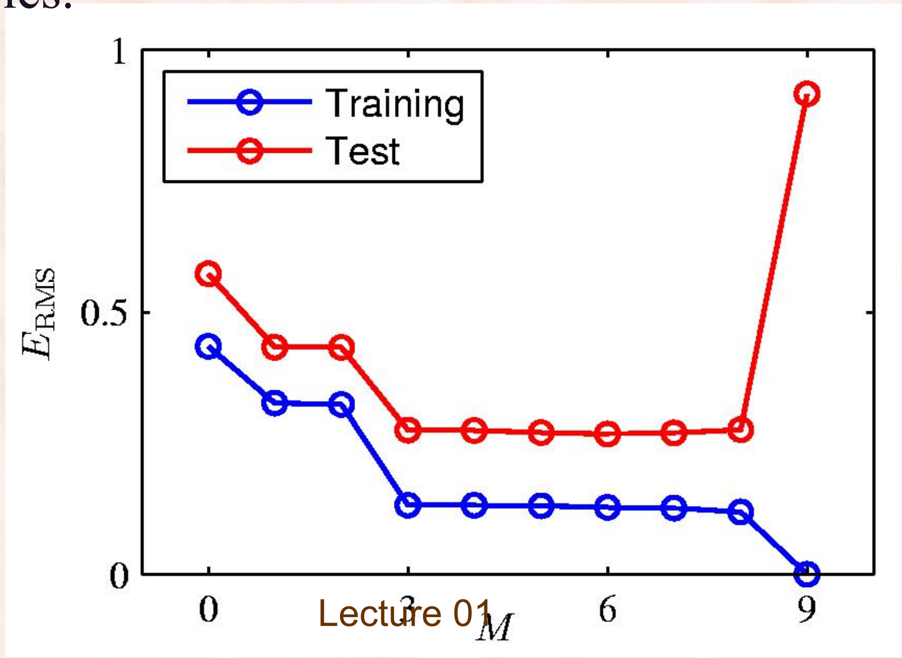
Polynomial Curve Fitting

- Model Selection: choosing the order M of the polynomial.
  - Best generalization obtained with $M = 3$.
  - $M = 9$ obtains poor generlization, even though it fits training examples perfectly:
    - But $M = 9$ polynomials subsume $M = 3$ polynomials!

- Overfitting $\equiv$ good performance on training examples, poor performance on test examples.

Overfitting

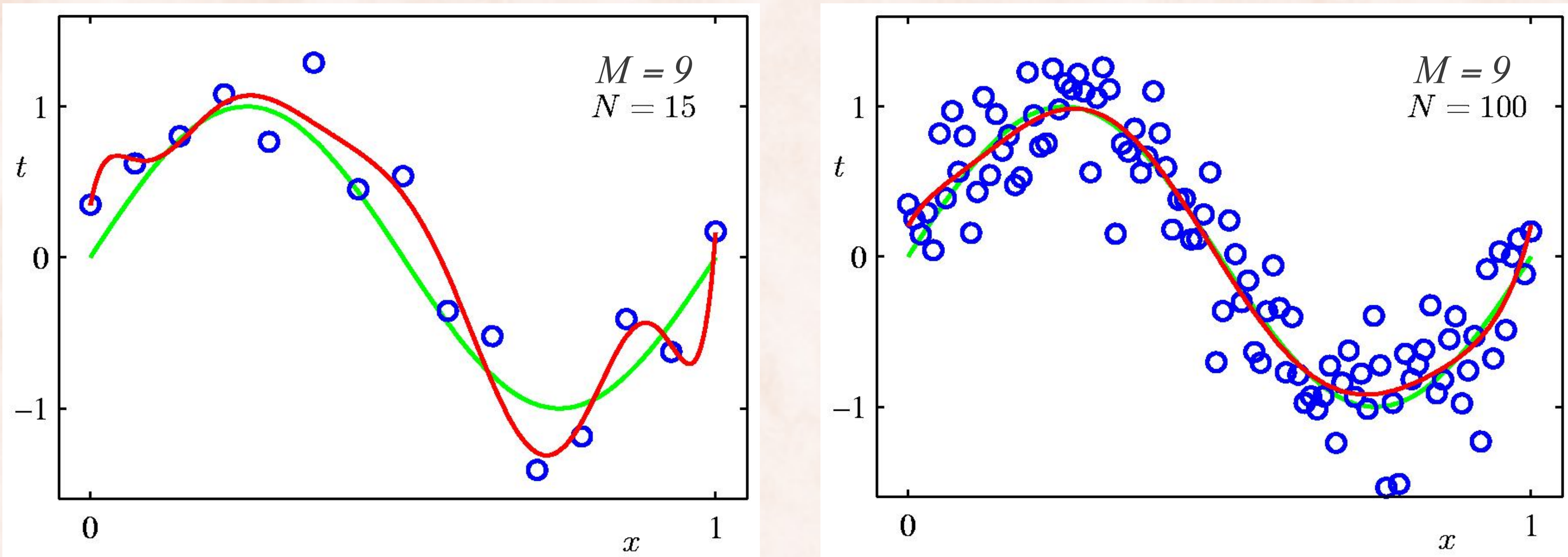
- Measure fit to training/testing examples using the Root-Mean-Square (RMS) error: $E_{RMS} = \sqrt{2E(w^*)/N}$
- Use 100 random test examples, generated in the same way as the training examples.



Over-fitting and Parameter Values

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Overfitting vs. Data Set Size



- More training data $\Rightarrow$ less overfitting.
- What if we do not have more training data?
  - Use **regularization**.
  - Use a probabilistic model in a Bayesian setting.

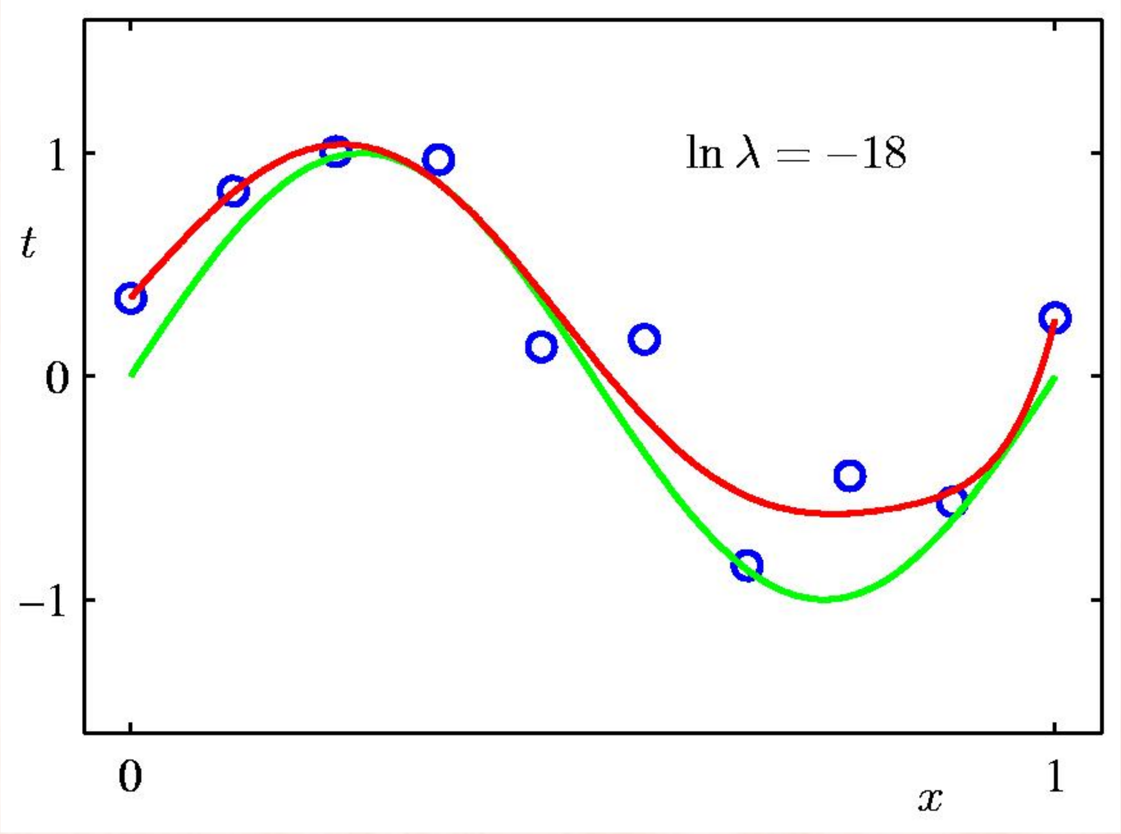
Regularization

- Penalize large parameter values:

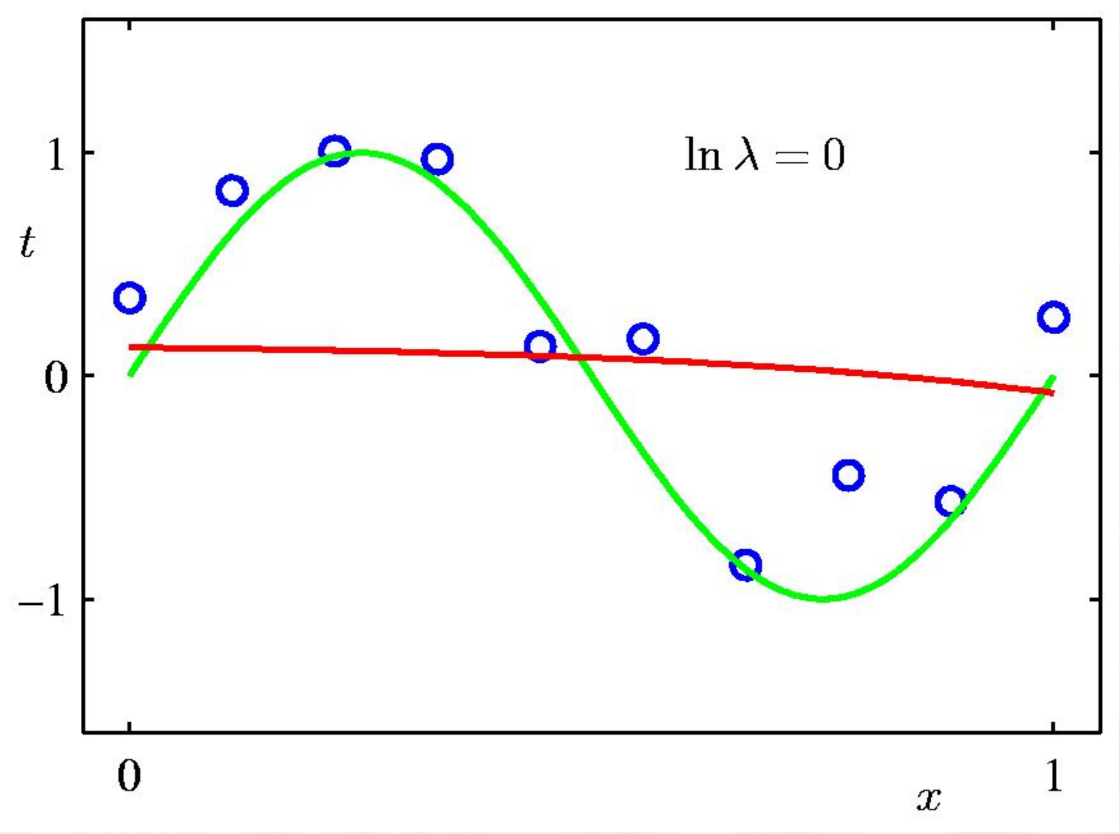
$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\textit{regularizer}}$$

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$$

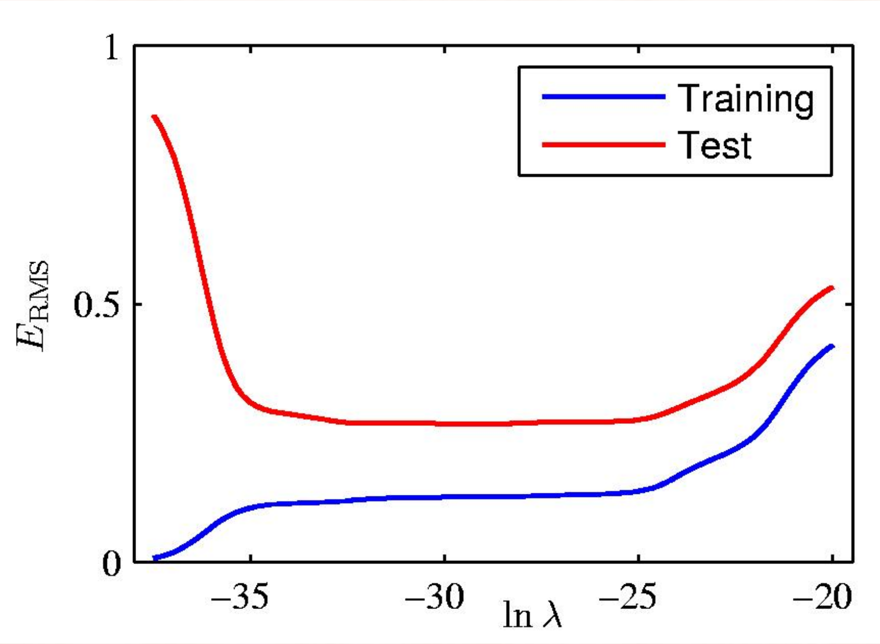
9^{th} Order Polynomial with Regularization



9^{th} Order Polynomial with Regularization



Training & Test error vs. $\ln \lambda$

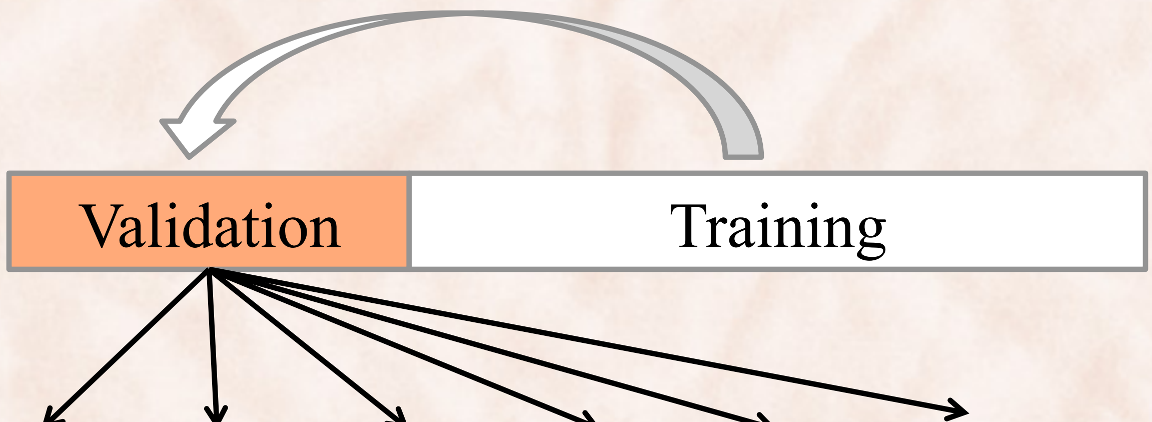


How do we find the optimal value of λ?

Model Selection

- Put aside an independent *validation set*.
- Select parameters giving best performance on validation set.

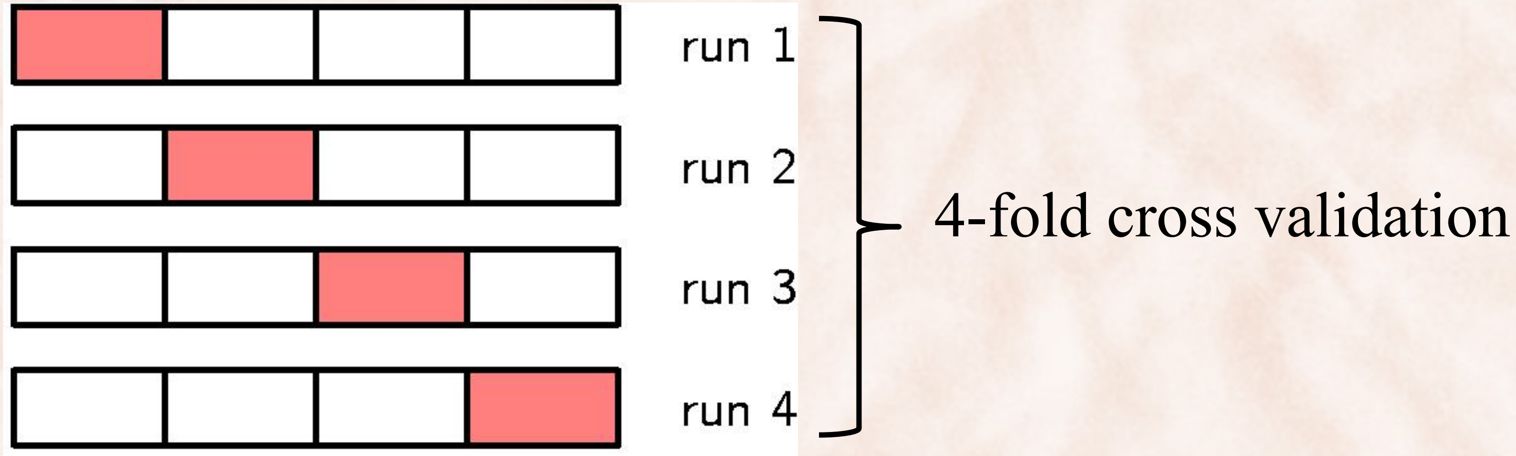
$$\ln \lambda \in \{-40, -35, -30, -25, -20, -15\}$$



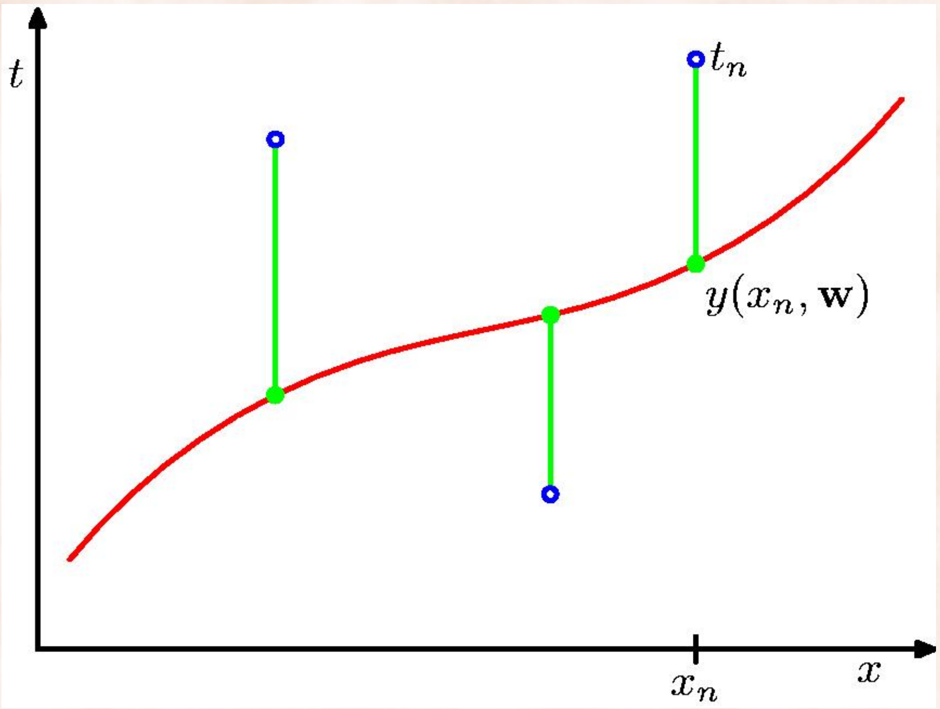
$\ln \lambda$	-40	-35	$\boxed{-30}$	-25	-20	-15
E_{RMS}	1.05	0.30	$\boxed{0.25}$	0.27	0.52	0.55

Model Evaluation

- K-fold cross-validation
  - randomly partition dataset in K equally sized subsets $P_1, P_2, \ldots P_k$
  - for each fold *i* in {1, 2, …, k}:
    - test on P_i, train on $P_1 \cup \ldots \cup P_{i-1} \cup P_{i+1} \cup \ldots \cup P_k$
  - compute average error/accuracy across K folds.



Sum-of-Squares Error Function (Revisited)



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

$$= \frac{1}{2}\sum_{n=1}^{N}\{\mathbf{w}^T\phi(x_n) - t_n\}^2$$

- Training objective: *minimize sum-of-squares error*.
- Why *least squares*?

Least Squares <=> Maximum Likelihood (1)

- Assume observations from a deterministic function with added Gaussian noise:

$$t = y(\mathbf{x}, \mathbf{w}) + \epsilon \qquad \text{where} \qquad p(\epsilon|\beta) = \mathcal{N}(\epsilon|0, \beta^{-1})$$

  which is the same as saying:

$$p(t|\mathbf{x}, \mathbf{w}, \beta) = \mathcal{N}(t|y(\mathbf{x}, \mathbf{w}), \beta^{-1}).$$

- Given observed inputs $\mathbf{X} = \{\mathbf{x}_1, ..., \mathbf{x}_N\}$ and targets $\mathbf{t} = [\mathbf{t}_1, ..., \mathbf{t}_N]^T$, we obtain the *likelihood* function:

$$p(\mathbf{t}|\mathbf{X}, \mathbf{w}, \beta) = \prod_{n=1}^{N} \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}).$$

Least Squares <=> Maximum Likelihood (2)

- Taking the logarithm, we get the *log-likelihood* function:

$$\begin{aligned} \ln p(\mathbf{t}|\mathbf{w}, \beta) &= \sum_{n=1}^{N} \ln \mathcal{N}(t_n|\mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n), \beta^{-1}) \\ &= \frac{N}{2}\ln\beta - \frac{N}{2}\ln(2\pi) - \beta E_D(\mathbf{w}) \end{aligned}$$

where

$$E_D(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2$$

- $E_D(\mathbf{w})$ is the sum-of-squares error!

Least Squares <=> Maximum Likelihood (3)

- Minimizing square error <=> maximizing likelihood:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E_D(\mathbf{w}) = \mathbf{w}_{ML} = \arg\max_{\mathbf{w}} \ln p(\mathbf{t} \mid \mathbf{w}, \beta)$$

- How do we find **w** (and β)?

Least Squares <=> Maximum Likelihood (4)

- Computing the gradient and setting it to zero yields:

$$\nabla_{\mathbf{w}} \ln p(\mathbf{t}|\mathbf{w}, \beta) = \beta \sum_{n=1}^{N} \left\{ t_n - \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x}_n) \right\} \boldsymbol{\phi}(\mathbf{x}_n)^{\mathrm{T}} = \mathbf{0}.$$

- Solving for $\mathbf{w}$, we get

$$\mathbf{w}_{\mathrm{ML}} = \overbrace{\left(\boldsymbol{\Phi}^{\mathrm{T}} \boldsymbol{\Phi} \right)^{-1} \boldsymbol{\Phi}^{\mathrm{T}}}\mathbf{t}$$

The Moore-Penrose pseudo-inverse, $\boldsymbol{\Phi}^{\dagger}$.

where

$$\boldsymbol{\Phi} = \begin{pmatrix} \phi_0(\mathbf{x}_1) & \phi_1(\mathbf{x}_1) & \cdots & \phi_{M-1}(\mathbf{x}_1) \\ \phi_0(\mathbf{x}_2) & \phi_1(\mathbf{x}_2) & \cdots & \phi_{M-1}(\mathbf{x}_2) \\ \vdots & \vdots & \ddots & \vdots \\ \phi_0(\mathbf{x}_N) & \phi_1(\mathbf{x}_N) & \cdots & \phi_{M-1}(\mathbf{x}_N) \end{pmatrix}.$$

Least Squares <=> Maximum Likelihood (5)

- Minimizing square error <=> maximizing likelihood:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} E_D(\mathbf{w}) = \mathbf{w}_{ML} = \arg\max_{\mathbf{w}} \ln p(\mathbf{t} \mid \mathbf{w}, \beta)$$

- Maximizing with respect to w gives:

$$\mathbf{w}_{\mathrm{ML}} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{t}$$

- Maximizing with respect to β gives:

$$\frac{1}{\beta_{\mathrm{ML}}} = \frac{1}{N}\sum_{n=1}^{N}\{t_n - \mathbf{w}_{\mathrm{ML}}^{\mathrm{T}}\phi(\mathbf{x}_n)\}^2$$

Regularized Least Square

- Consider the error function:

$$E_D(\mathbf{w}) + \lambda E_W(\mathbf{w})$$

Data term + Regularization term

- With the sum-of-squares error function and a quadratic regularizer, we get:

$$\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^{\mathrm{T}}\boldsymbol{\phi}(\mathbf{x}_n)\}^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

which is minimized by:

$$\mathbf{w} = \left(\lambda\mathbf{I} + \boldsymbol{\Phi}^{\mathrm{T}}\boldsymbol{\Phi}\right)^{-1}\boldsymbol{\Phi}^{\mathrm{T}}\mathbf{t}.$$

λ is called the *regularization coefficient*.

Regularized Least Square <=> Maximum A Posteriori (MAP)

- Define a conjugate prior over w

$$p(\mathbf{w}) = \mathcal{N}(\mathbf{w}|\mathbf{0}, \alpha^{-1}\mathbf{I})$$

- Combining this with the likelihood function and using results for marginal and conditional Gaussian distributions, gives the posterior

$$p(\mathbf{w}|\mathbf{t}) = \mathcal{N}(\mathbf{w}|\mathbf{m}_N, \mathbf{S}_N)$$

where

$$\begin{aligned} \mathbf{m}_N &= \beta \mathbf{S}_N \mathbf{\Phi}^{\mathrm{T}} \mathbf{t} \\ \mathbf{S}_N^{-1} &= \alpha \mathbf{I} + \beta \mathbf{\Phi}^{\mathrm{T}} \mathbf{\Phi}. \end{aligned}$$

Regularized Least Square <=> Maximum A Posteriori (MAP)

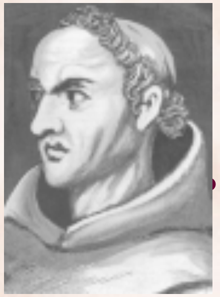
- Taking the logarithm of the posterior distribution:

$$\ln p(\mathbf{w} \mid \mathbf{t}) = -\frac{\beta}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\varphi(x_n)\}^2 - \frac{\alpha}{2}\mathbf{w}^T\mathbf{w} + const$$

- The MAP estimate of w is:

$$\begin{aligned}
\mathbf{w}_{MAP} &= \arg\max_{\mathbf{w}} \ \ln p(\mathbf{w} \mid \mathbf{t}) \\
&= \arg\max_{\mathbf{w}} \ -\frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\varphi(x_n)\}^2 - \frac{\boxed{\alpha/\beta}}{2}\mathbf{w}^T\mathbf{w} \\
&= \arg\min_{\mathbf{w}} \ \frac{1}{2}\sum_{n=1}^{N}\{t_n - \mathbf{w}^T\varphi(x_n)\}^2 + \frac{\boxed{\lambda}}{2}\mathbf{w}^T\mathbf{w} \\
&= \arg\min_{\mathbf{w}} \ E_D(\mathbf{w}) + E_W(\mathbf{w})
\end{aligned}$$

Regularization & Occam's Razor



William of Occam (1288 – 1348)

- English Franciscan friar, theologian and philosopher.

- "*Entia non sunt multiplicanda praeter necessitatem*"
  - Entities must not be multiplied beyond necessity.

i.e. Do not make things needlessly complicated.

i.e. Prefer the simplest hypothesis that fits the data.

Gradient Descent (Batch)

- Want to minimize a function $f : R^{n} \rightarrow R$.
  - f is differentiable and convex.
  - compute gradient of f i.e. *direction of steepest increase*:

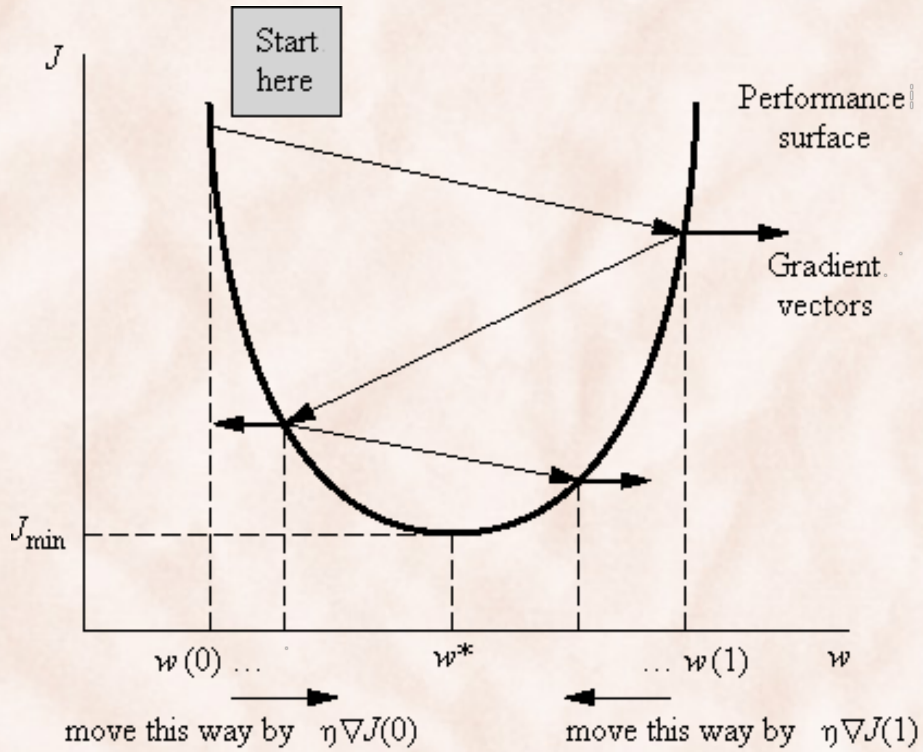
$$\nabla f(\mathbf{x}) = \left[\frac{df}{dx_1}(\mathbf{x}), \frac{df}{dx_2}(\mathbf{x}), \ldots, \frac{df}{dx_n}(\mathbf{x}) \right]$$

  - choose a sequence of points x^1, x^2, … and a learning rate η such that:

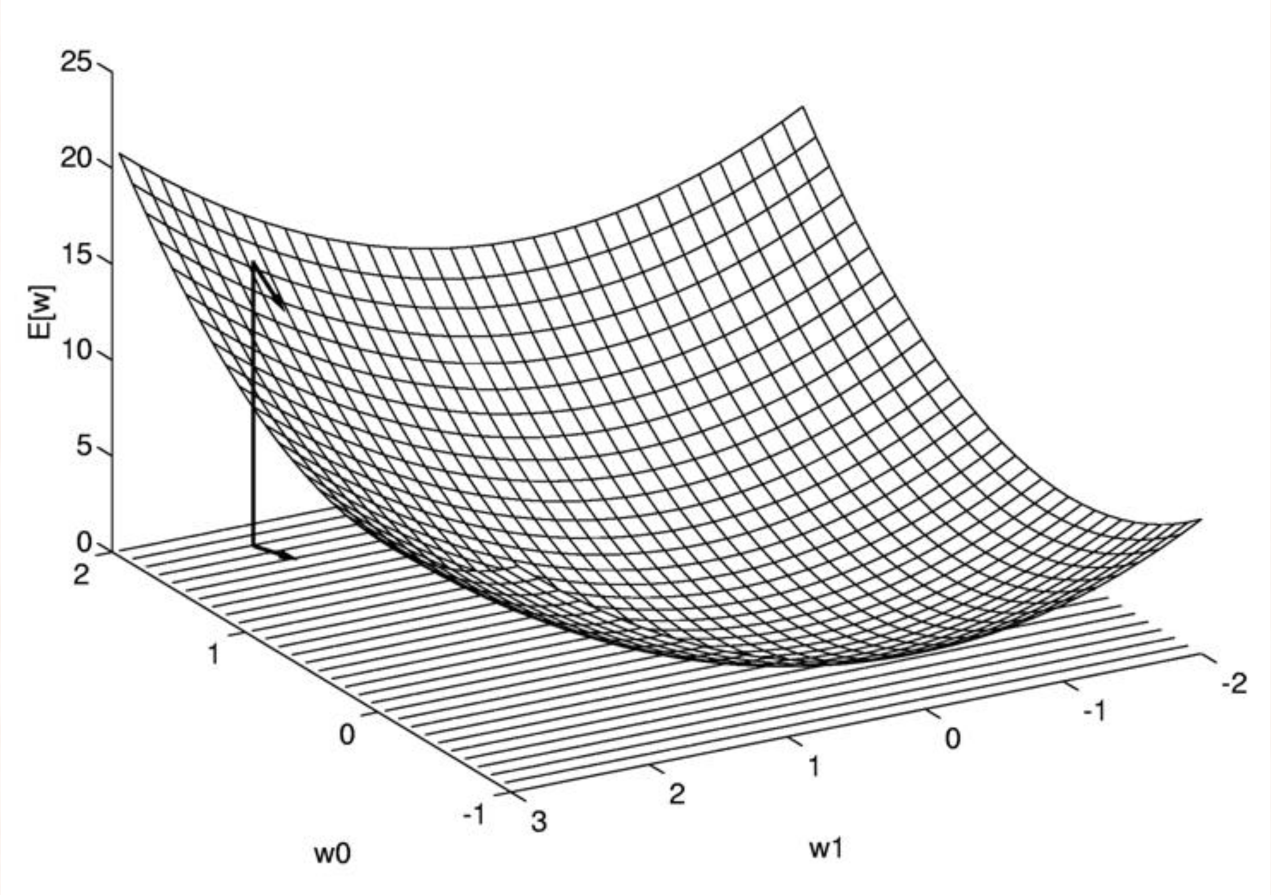
$$\mathbf{x}^{\tau+1} = \mathbf{x}^{\tau} - \eta \nabla f(\mathbf{x}^{\tau})$$

- Sum-of-squares error: $E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^{N} \{\mathbf{w}^T \phi(x_n) - t_n\}^2$

Gradient Descent



Gradient Descent



Stochastic Gradient Descent (Online)

- Decompose error function in sum of example errors:

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

- Update parameters **w** after each example, sequentially:

$$\mathbf{w}^{(\tau+1)} = \mathbf{w}^{(\tau)} - \eta\nabla E_n(\mathbf{w}^{(\tau)})$$
$$= \mathbf{w}^{(\tau)} + \eta(t_n - \mathbf{w}^{(\tau)T}\varphi(\mathbf{x}_n))\varphi(\mathbf{x}_n)$$

=> the *least-mean-square* (LMS) algorithm.

Regularization: Ridge vs. Lasso

- Ridge regression:

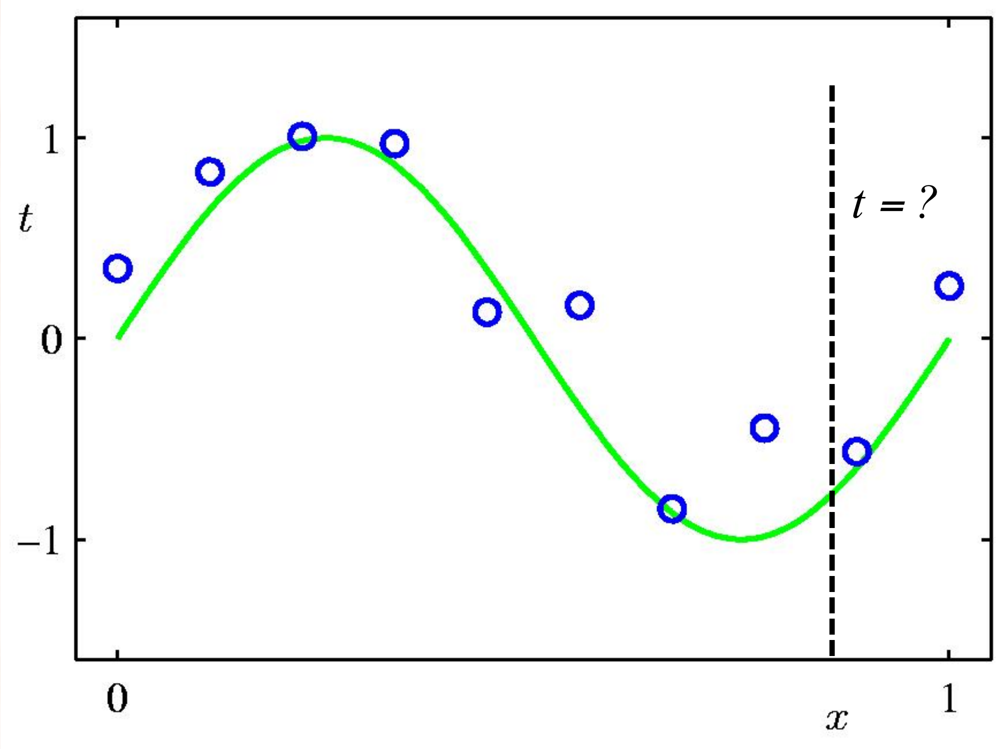
$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M} w_j^2$$

- Lasso:

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n,\mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\sum_{j=1}^{M} |w_j|$$

  – If λ is sufficiently large, some of the coefficients w_j are driven to 0 => *sparse* model.

Polynomial Curve Fitting (Revisited)



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

parameters *features*

Generalization: Basis Functions as Features

- Generally

$$y(\mathbf{x}, \mathbf{w}) = \sum_{j=0}^{M-1} w_j \phi_j(\mathbf{x}) = \mathbf{w}^{\mathrm{T}} \boldsymbol{\phi}(\mathbf{x})$$

where $\varphi_j(\mathbf{x})$ are known as *basis functions*.

- Typically $\varphi_0(\mathbf{x}) = 1$, so that w_0 acts as a bias.

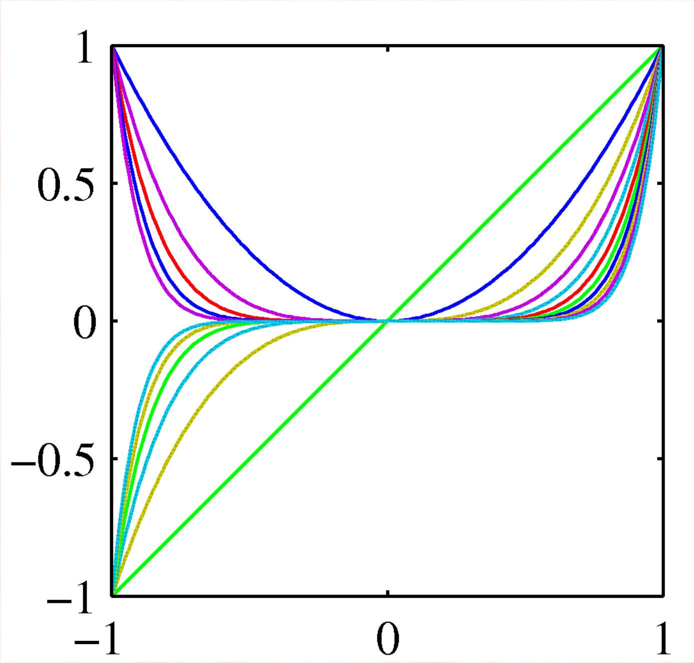
- In the simplest case, use linear basis functions : $\varphi_d(\mathbf{x}) = x_d$.

Linear Basis Function Models (1)

- Polynomial basis functions:

$$\phi_j(x) = x^j.$$

- Global behavior:
  - a small change in *x* affect all basis functions.

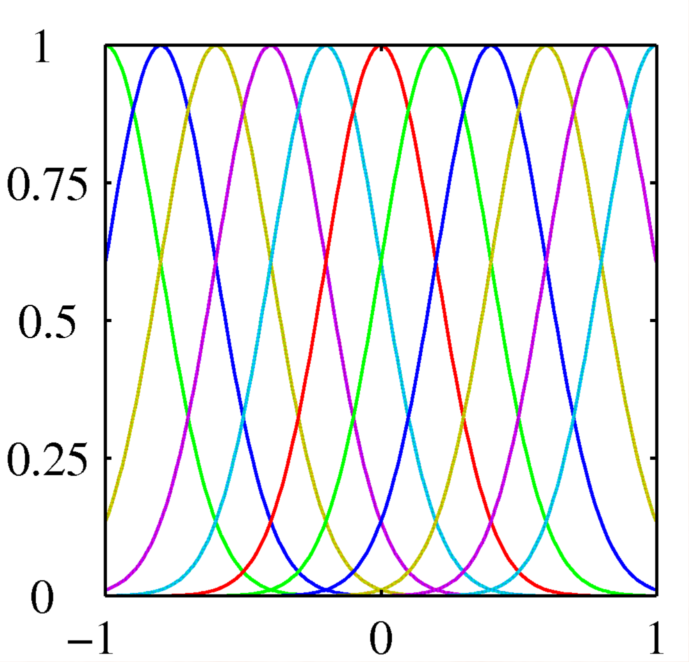


Linear Basis Function Models (2)

- Gaussian basis functions:

$$\phi_j(x) = \exp\left\{-\frac{(x-\mu_j)^2}{2s^2}\right\}$$

- Local behavior:
  - a small change in x only affects nearby basis functions.
  - μ_j and s control location and scale (width).

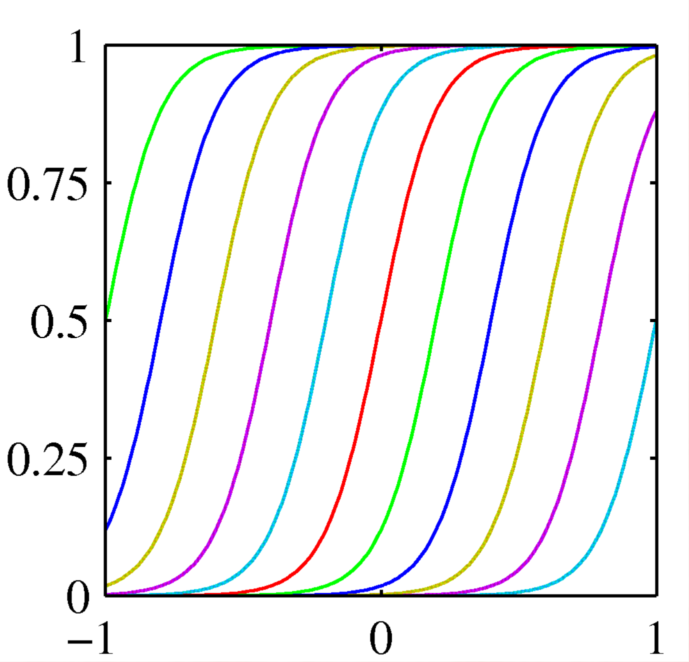


Linear Basis Function Models (3)

- Sigmoidal basis functions:

$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

where $\sigma(a) = \dfrac{1}{1 + \exp(-a)}.$



- Local behavior:
  - a small change in x only affect nearby basis functions.
  - μ_j and s control location and scale (slope).