Machine Learning
CS 4900/5900

Linear Regression

Razvan C. Bunescu

School of Electrical Engineering and Computer Science

bunescu@ohio.edu

Supervised Learning

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

- Target function $t(\mathbf{x})$ is known (only) through (noisy) set of training examples:

$$(\mathbf{x}_1,t_1), (\mathbf{x}_2,t_2), \ldots (\mathbf{x}_n,t_n)$$

Supervised Learning

- **Task** = learn an (unknown) function $t : X \rightarrow T$ that maps input instances $\mathbf{x} \in X$ to output targets $t(\mathbf{x}) \in T$:
  - function t is known (only) through (noisy) set of training examples:
    - Training/Test data: $(\mathbf{x}_1,t_1)$, $(\mathbf{x}_2,t_2)$, … $(\mathbf{x}_n,t_n)$

- **Task** = build a function $h(\mathbf{x})$ such that:
  - *h* matches *t* well on the *training data*:

    => *h* is able to fit data that it has seen.
  - *h* also matches target *t* well on *test data*:

    => *h* is able to generalize to unseen data.

Parametric Approaches to Supervised Learning

- **Task** = build a function $h(\mathbf{x})$ such that:
  - h matches t well on the training data:

    => h is able to fit data that it has seen.
  - h also matches t well on test data:

    => h is able to generalize to unseen data.

- **Task** = choose h from a "nice" *class of functions* that depend on a vector of parameters $\mathbf{w}$:
  - $h(\mathbf{x}) \equiv h_{\mathbf{w}}(\mathbf{x}) \equiv h(\mathbf{w},\mathbf{x})$
  - **what classes of functions are "nice"?**

Linear Regression

1. (Simple) Linear Regression
    - House price prediction

2. Linear Regression with Polynomial Features
    - Polynomial curve fitting
    - Regularization
    - Ridge regression

3. Multiple Linear Regression
    - House price prediction
    - Normal equations

House Price Prediction

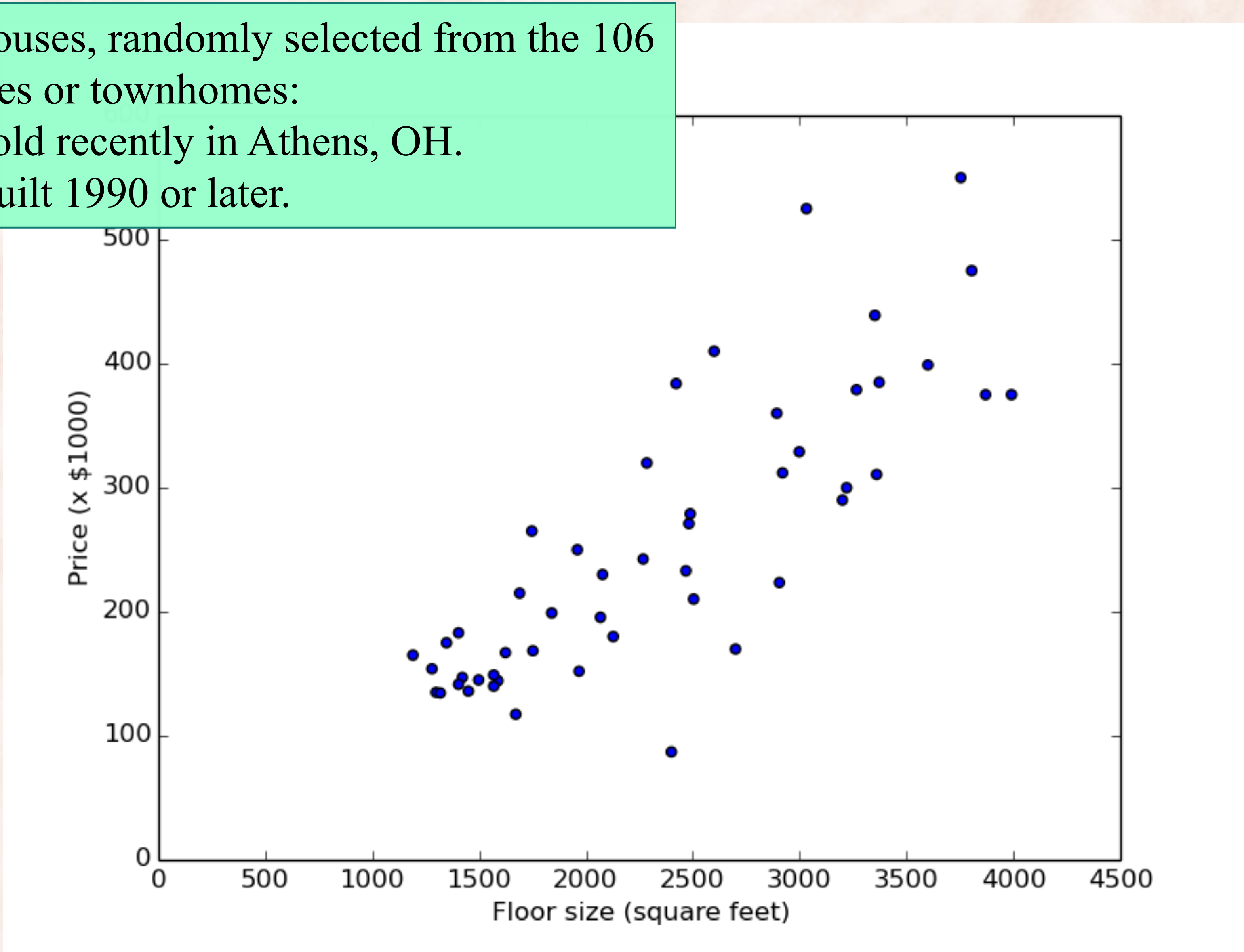
- Given the floor size in square feet, predict the selling price:
  - x is the size, t is the price
  - Need to learn a function h such that $h(x) \approx t(x)$.

- Is this classification or regression?
  - **Regression**, because price is real valued.
    - and there are many possible prices.
  - (Simple) linear regression, because one input value.
  - Would a problem with only two labels $t_1 = 0.5$ and $t_2 = 1.0$ still be regression?

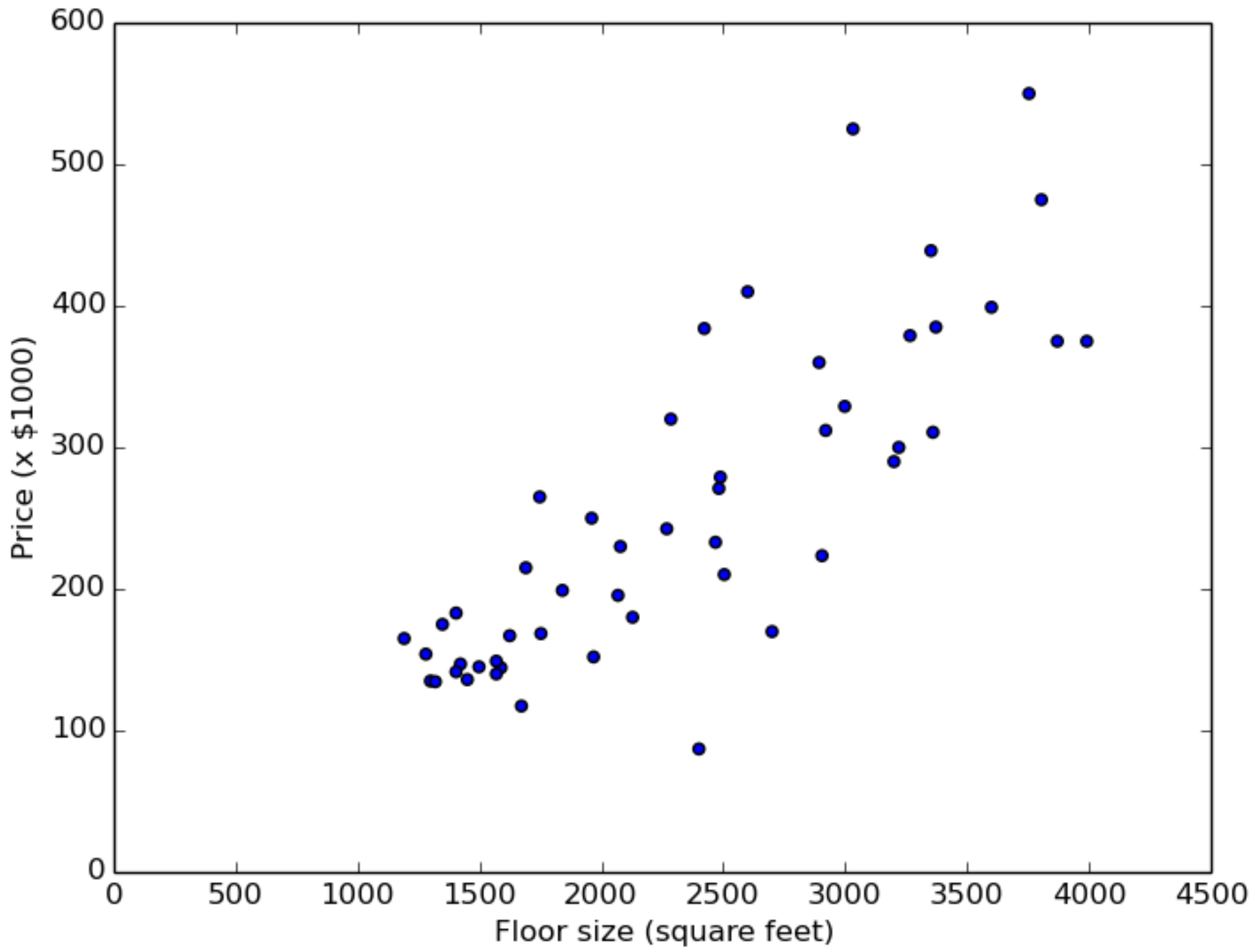
House Prices in Athens

50 houses, randomly selected from the 106 houses or townhomes:

- sold recently in Athens, OH.
- built 1990 or later.



House Prices in Athens



Parametric Approaches to Supervised Learning

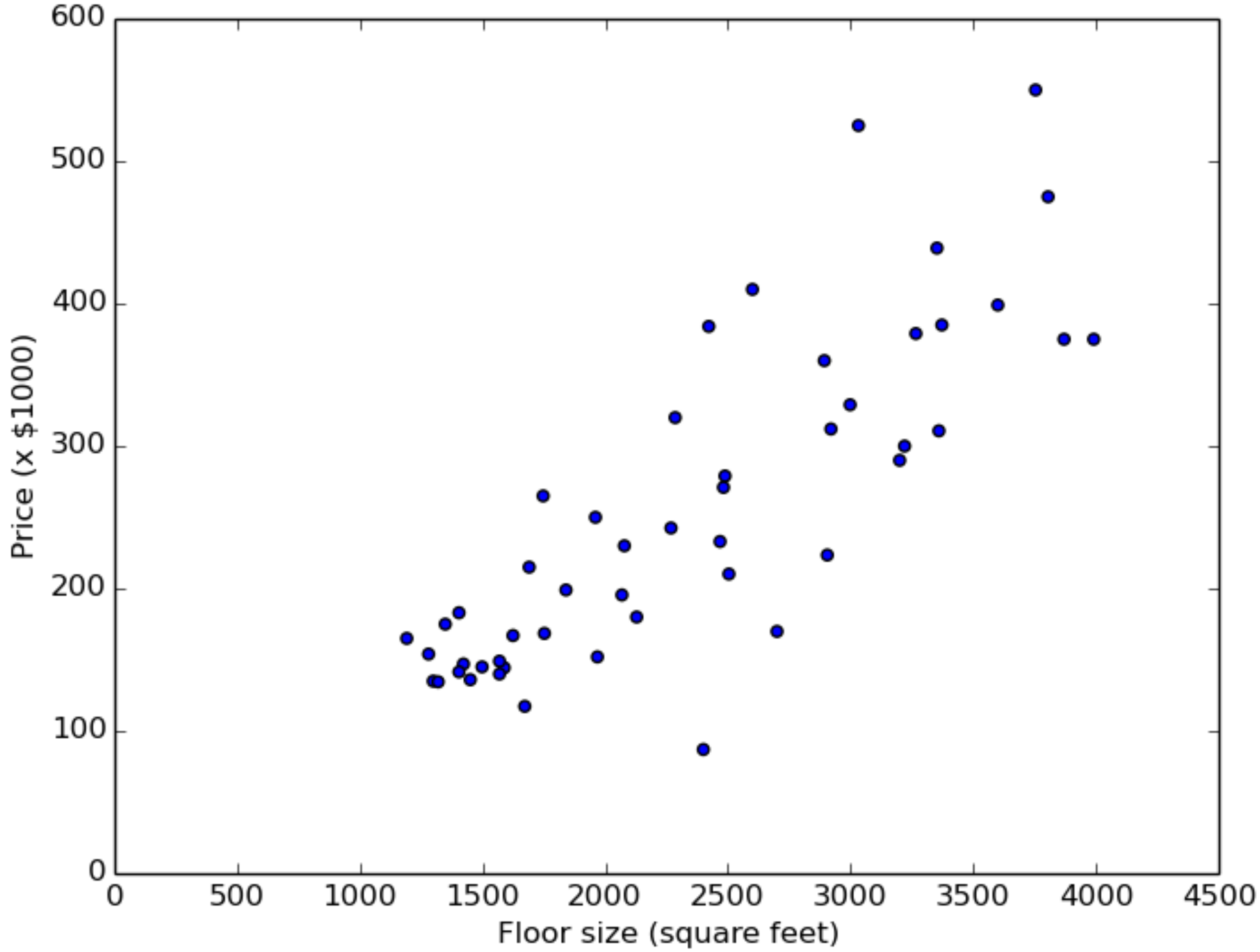
- **Task** = build a function $h(\mathbf{x})$ such that:
  - h matches t well on the training data:

    => h is able to fit data that it has seen.
  - h also matches t well on test data:

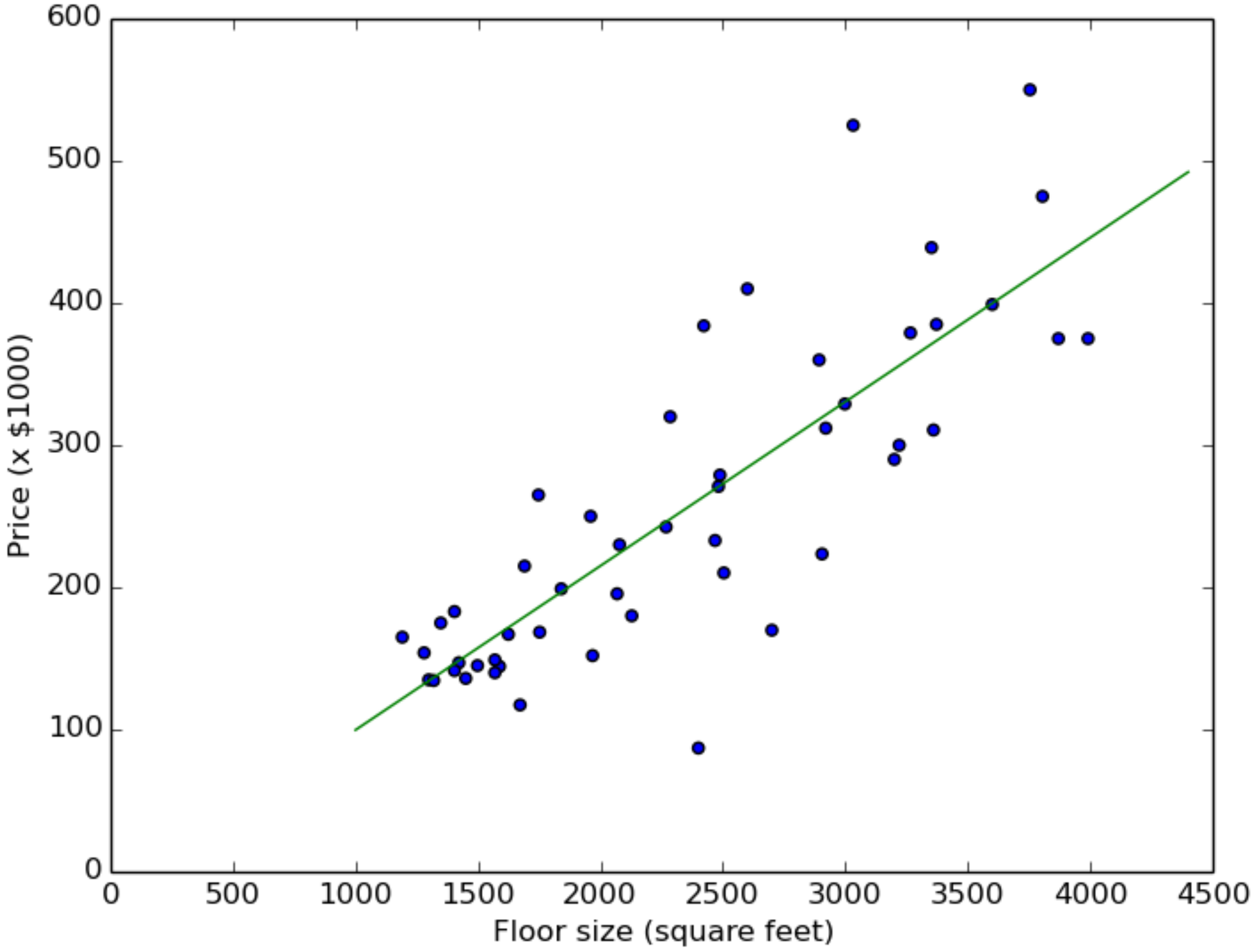
    => h is able to generalize to unseen data.

- **Task** = choose h from a "nice" *class of functions* that depend on a vector of parameters $\mathbf{w}$:
  - $h(\mathbf{x}) \equiv h_{\mathbf{w}}(\mathbf{x}) \equiv h(\mathbf{w},\mathbf{x})$
  - **what classes of functions are "nice"?**

House Prices in Athens



House Prices in Athens



Linear Regression

- Use a linear function approximation:
  - $h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} = [w_0, w_1]^{\mathrm{T}}[1, x] = w_1 x + w_0$.
    - w_0 is the intercept (or the bias term).
    - w_1 controls the slope.

  - Learning = optimization:
    - Find **w** that obtains the best fit on the training data, i.e. find **w** that minimizes the **sum of square errors**:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(h_{\mathbf{w}}(\mathbf{x}_{\mathrm{n}}) - t_n\right)^2$$

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, J(\mathbf{w})$$

Univariate Linear Regression

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

Inductive Learning Hyphotesis

- Sum-of-Squares error function:

$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_\mathrm{n}) - t_n)^2$$

Minimizing Sum-of-Squares Error

- Sum-of-Squares error function:

why squared?

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N}\left(h_{\mathbf{w}}(\mathbf{x}_\mathrm{n}) - t_n\right)^2$$

- How do we find **w*** that minimizes *E*(**w**)?

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

- Least Square solution is found by solving a system of 2 linear equations:

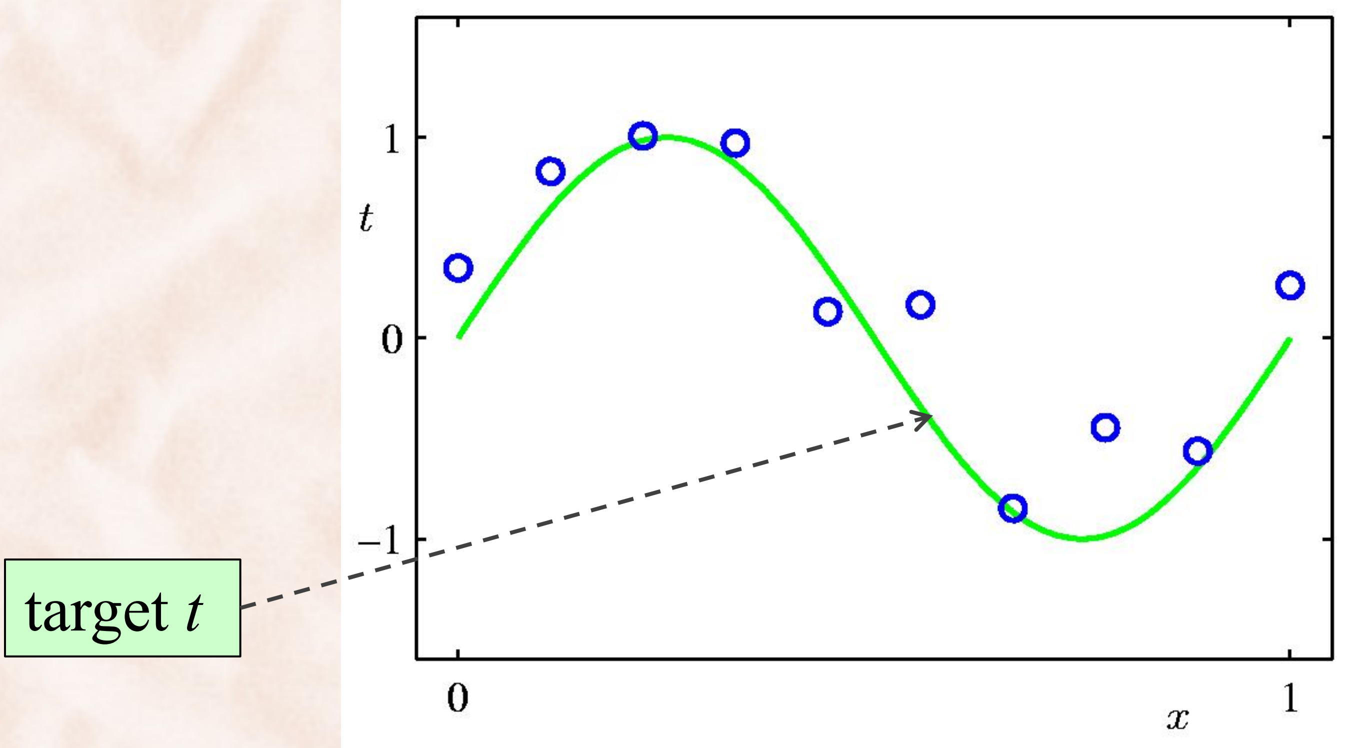
$$w_0 N + w_1 \sum_{n=1}^{N} x_n = \sum_{n=1}^{N} t_n$$

$$w_0 \sum_{n=1}^{N} x_n + w_1 \sum_{n=1}^{N} x_n^2 = \sum_{n=1}^{N} t_n\, x_n$$

Polynomial Basis Functions

- *Q*: What if the raw feature is insufficient for good performance?
  - Example: non-linear dependency between label and raw feature.
- *A*: Engineer [CS 4900] /Learn [CS 6890] higher-level features, as functions of the raw feature.
- **Polynomial curve fitting**:
  - Add new features, as polynomials of the original feature.

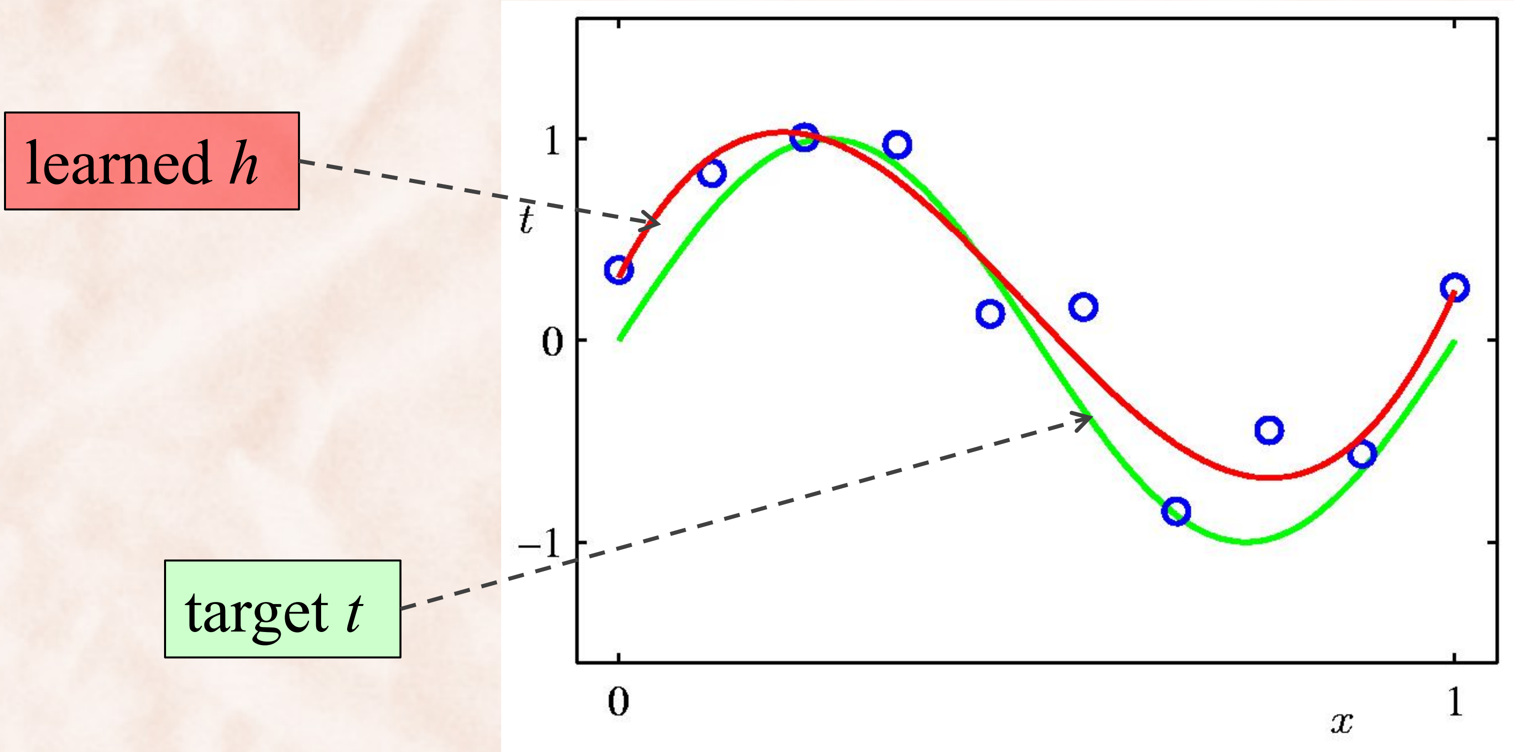
Regression: Curve Fitting



target *t*

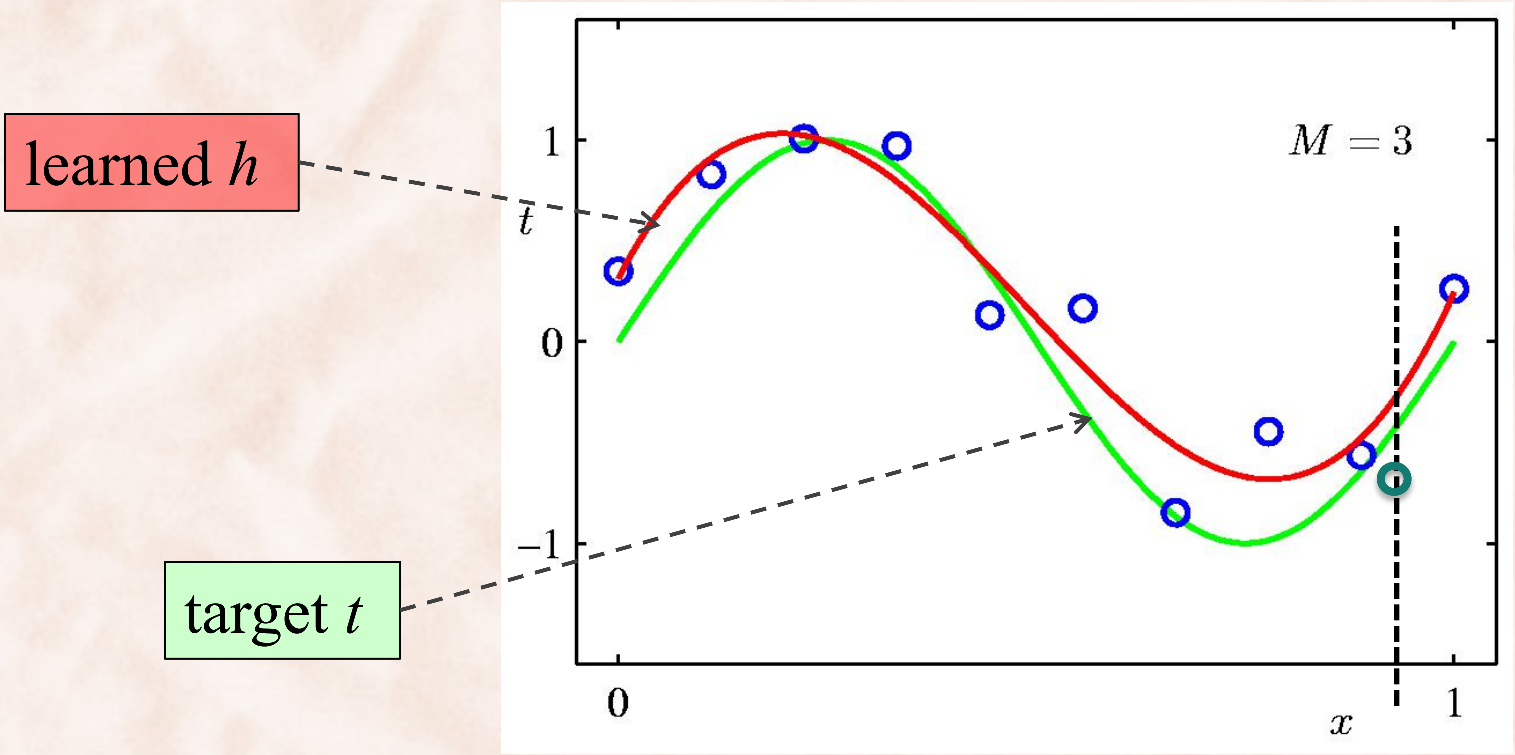
- **Training**: Build a function $h(x)$, based on (noisy) training examples (x_1,t_1), (x_2,t_2), … (x_N,t_N)

Regression: Curve Fitting



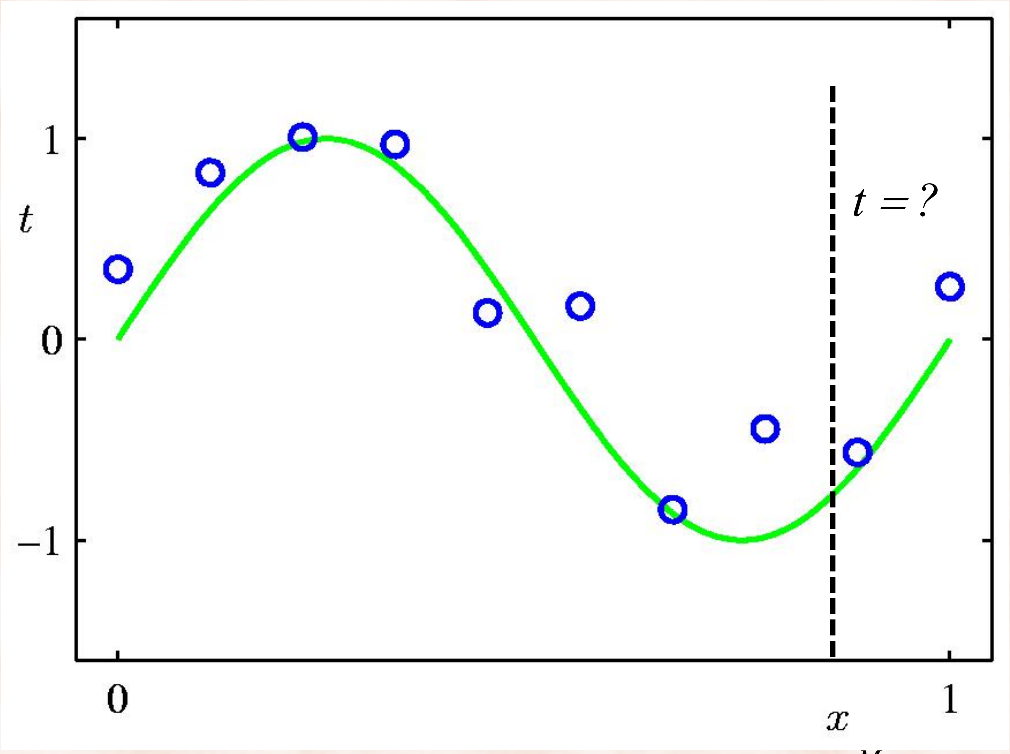
- **Training**: Build a function $h(x)$, based on (noisy) training examples (x_1,t_1), (x_2,t_2), … (x_N,t_N)

Regression: Curve Fitting



- **Testing**: for arbitrary (unseen) instance $x \in X$, compute target output $h(x)$; want it to be close to $t(x)$.

Regression: Polynomial Curve Fitting



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

parameters *features*

Polynomial Curve Fitting

- Parametric model:

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

- Polynomial curve fitting is (Multiple) Linear Regression:

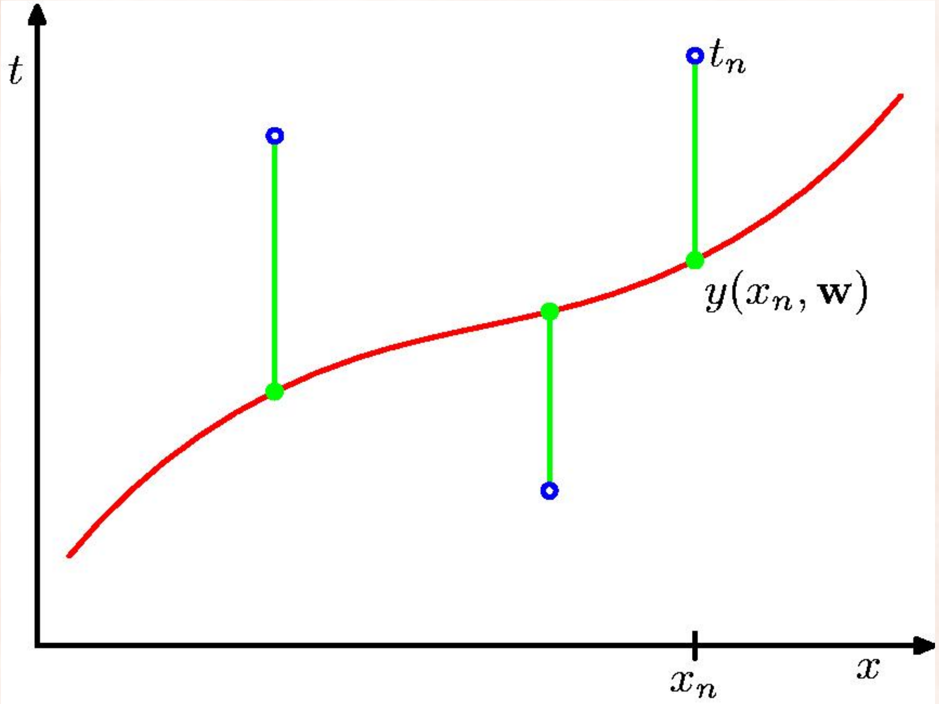
$\mathbf{x} = [1, x, x^2, \ldots, x^{\mathrm{M}}]^{\mathrm{T}}$

$h(x) = h(\mathbf{x},\mathbf{w}) = \mathbf{w}^{\mathrm{T}}\mathbf{x}$

- **Learning** = minimize the Sum-of-Squares error function:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w}) \qquad J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_{\mathrm{n}}) - t_n)^2$$

Sum-of-Squares Error Function



$$J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_{\mathrm{n}}) - t_n)^2$$

- How to find **w*** that minimizes $E(\mathbf{w})$, i.e. $\mathbf{w}^* = \arg\min_{\mathbf{w}} E(\mathbf{w})$
- Solve $\nabla J(\mathbf{w}) = 0$.

Polynomial Curve Fitting

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

$$\mathrm{A}\mathbf{w} = \mathrm{T}$$

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

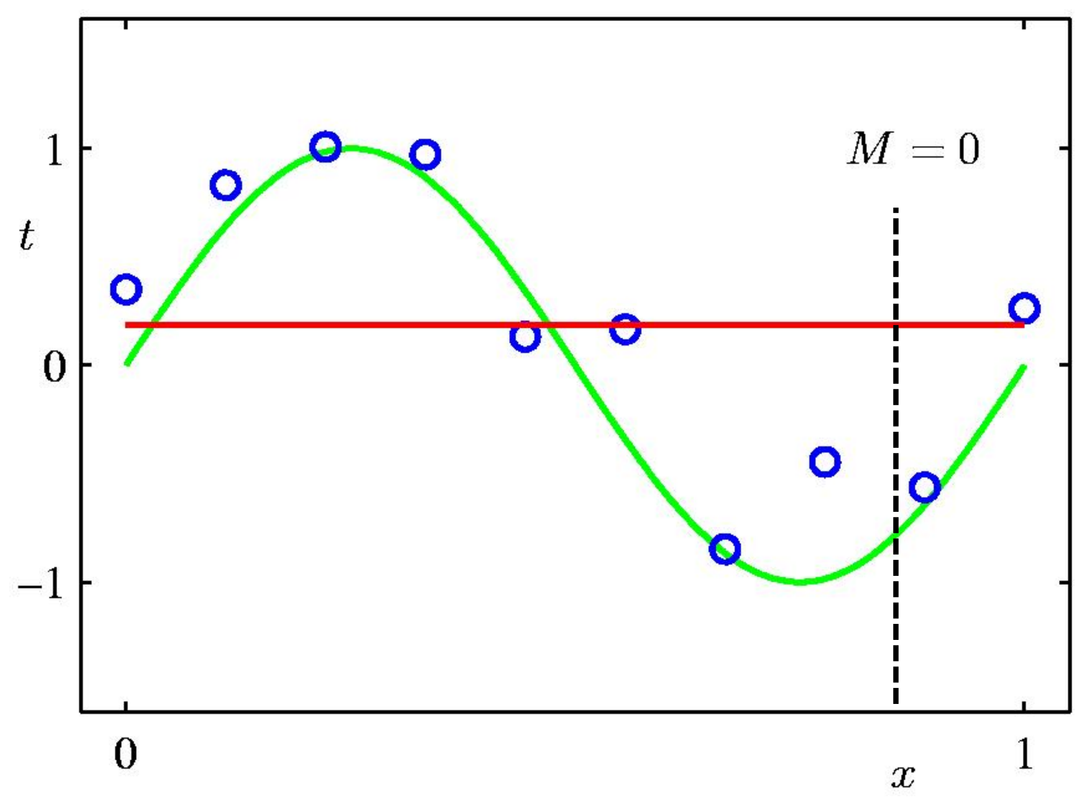
- Prove it.

Polynomial Curve Fitting

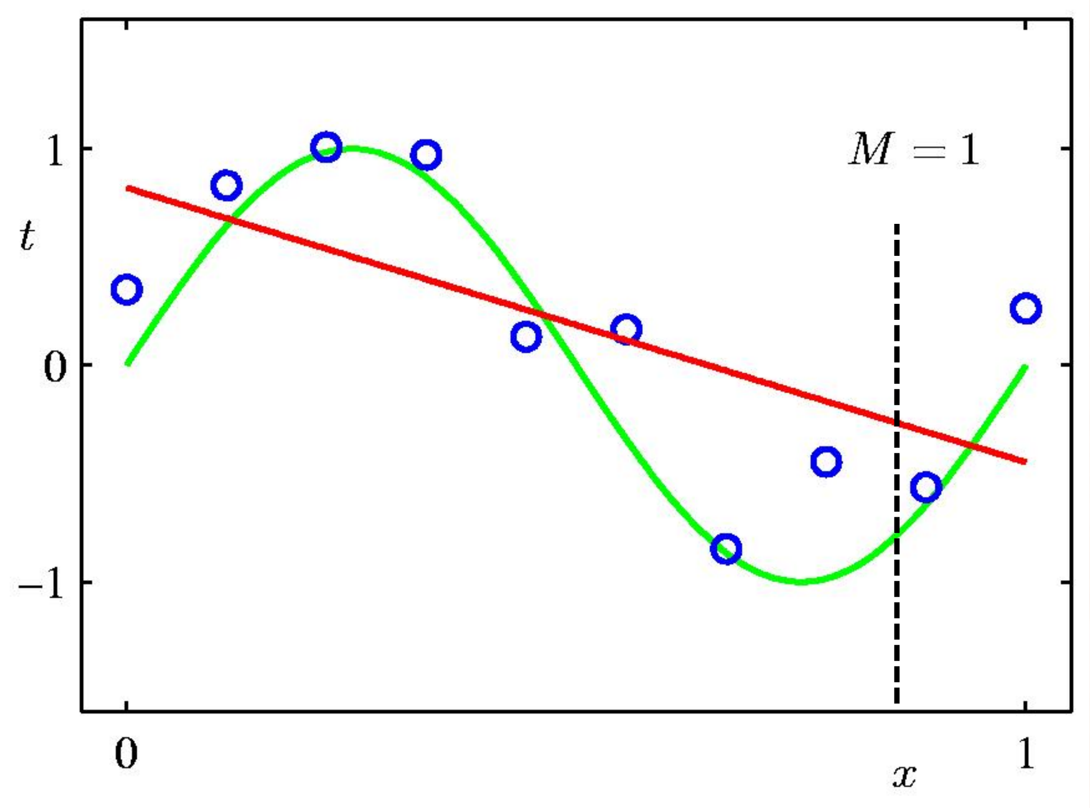
- Generalization = how well the parameterized $h(x,\mathbf{w})$ performs on arbitrary (unseen) test instances $x \in X$.

- Generalization performance depends on the value of M.

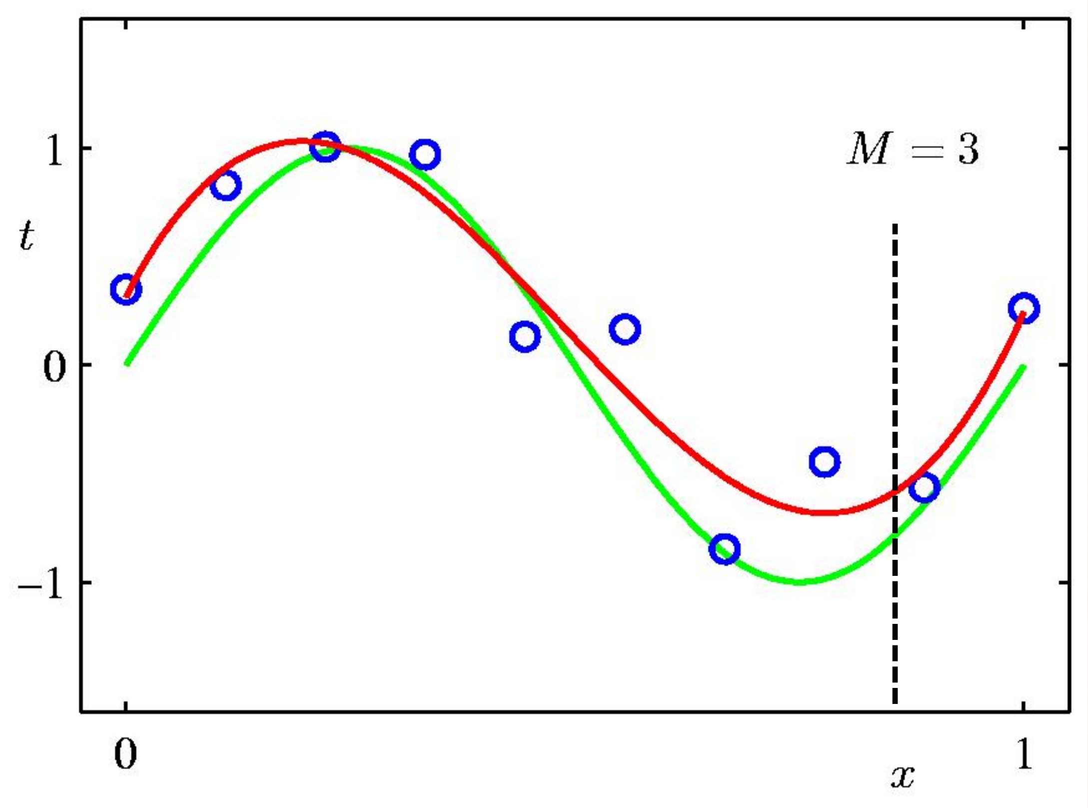
0^{th} Order Polynomial



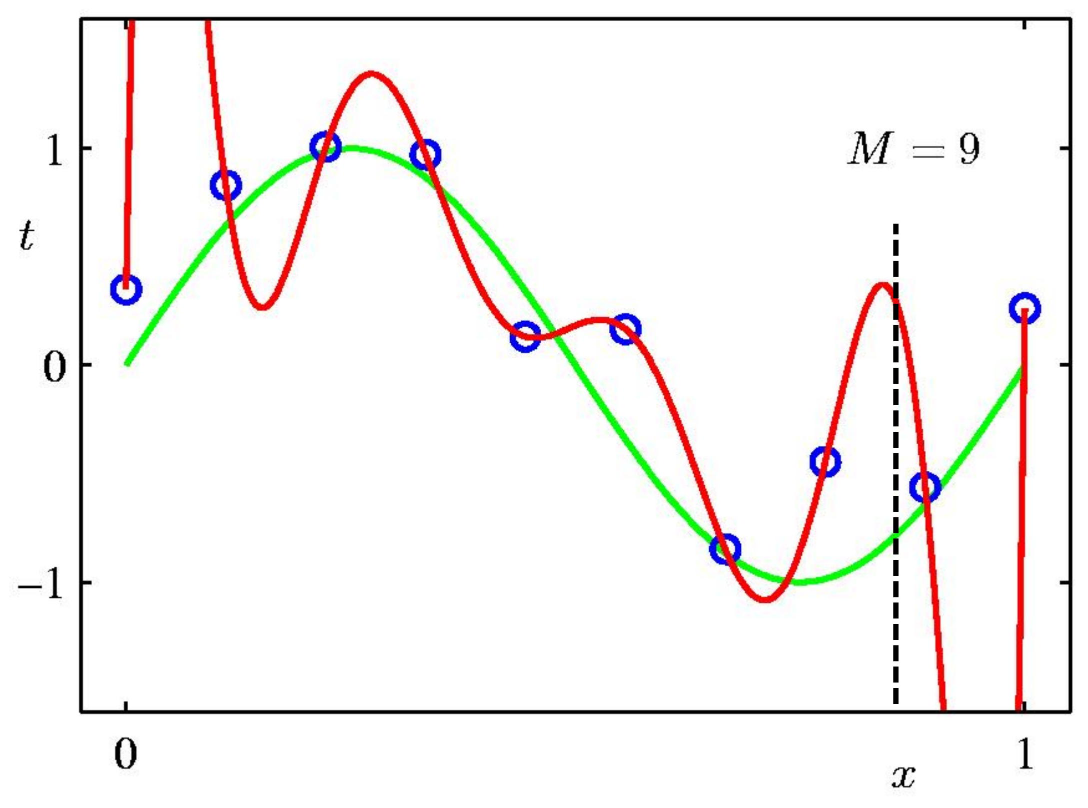
1^{st} Order Polynomial



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 generalization, even though it fits training examples perfectly:
    - But M = 9 polynomials subsume M = 3 polynomials!

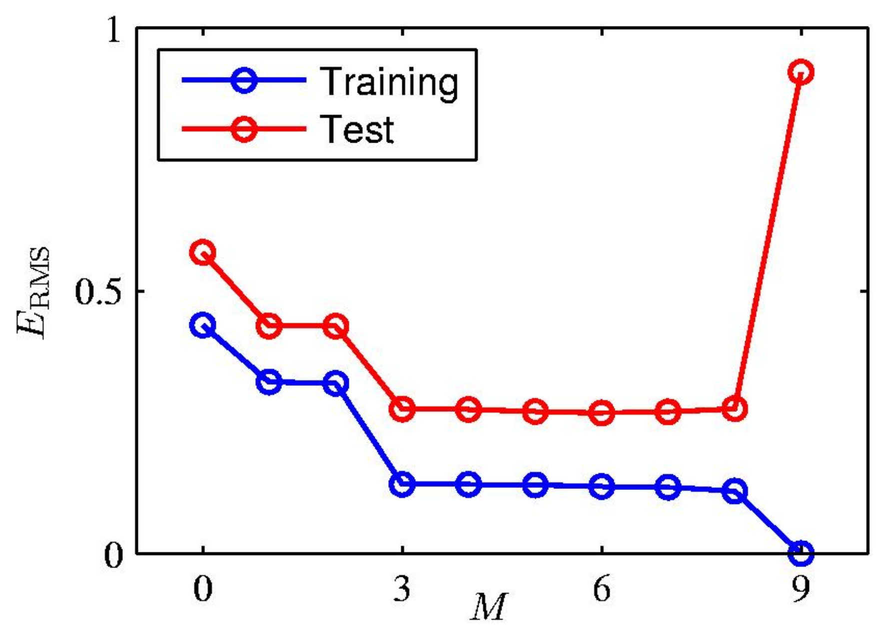
- Overfitting ≡ good performance on training examples, poor performance on test examples.

Overfitting

- Measure fit using the Root-Mean-Square (RMS) error:

$$E_{RMS}(\mathbf{w}) = \sqrt{\frac{\sum_n \left(\mathbf{w}^T \mathbf{x}_n - t_n\right)^2}{N}}$$

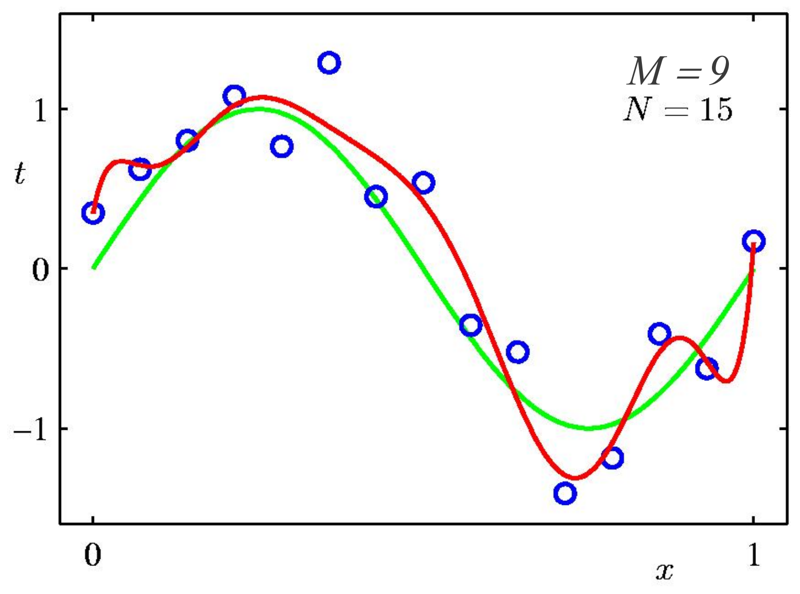
- Use 100 random test examples, generated in the same way:

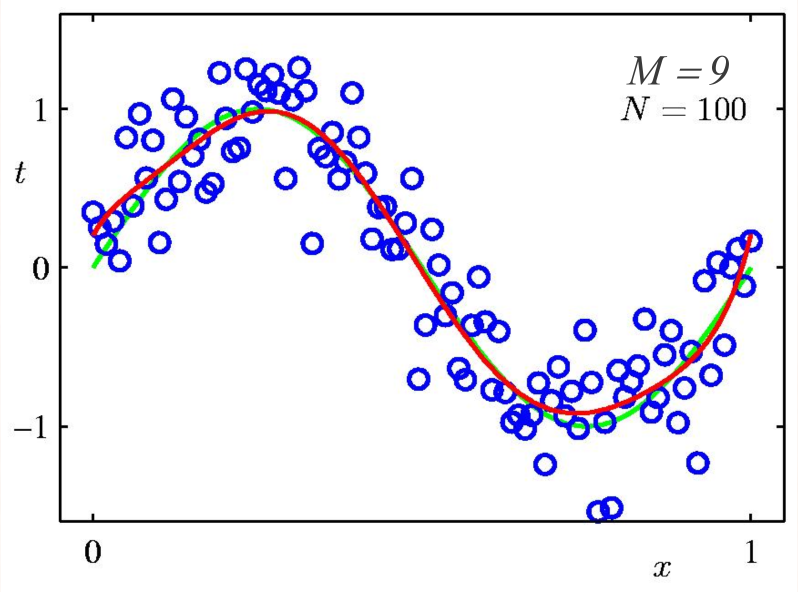


Over-fitting and Parameter Values

	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	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 ⇒ less overfitting.
- What if we do not have more training data?
  - Use **regularization**.

Regularization

- **Parameter norm penalties** (term in the objective).
- Limit parameter norm (constraint).
- Dataset augmentation.
- Dropout.
- Ensembles.
- Semi-supervised learning.
- Early stopping.
- Noise robustness.
- Sparse representations.
- Adversarial training.

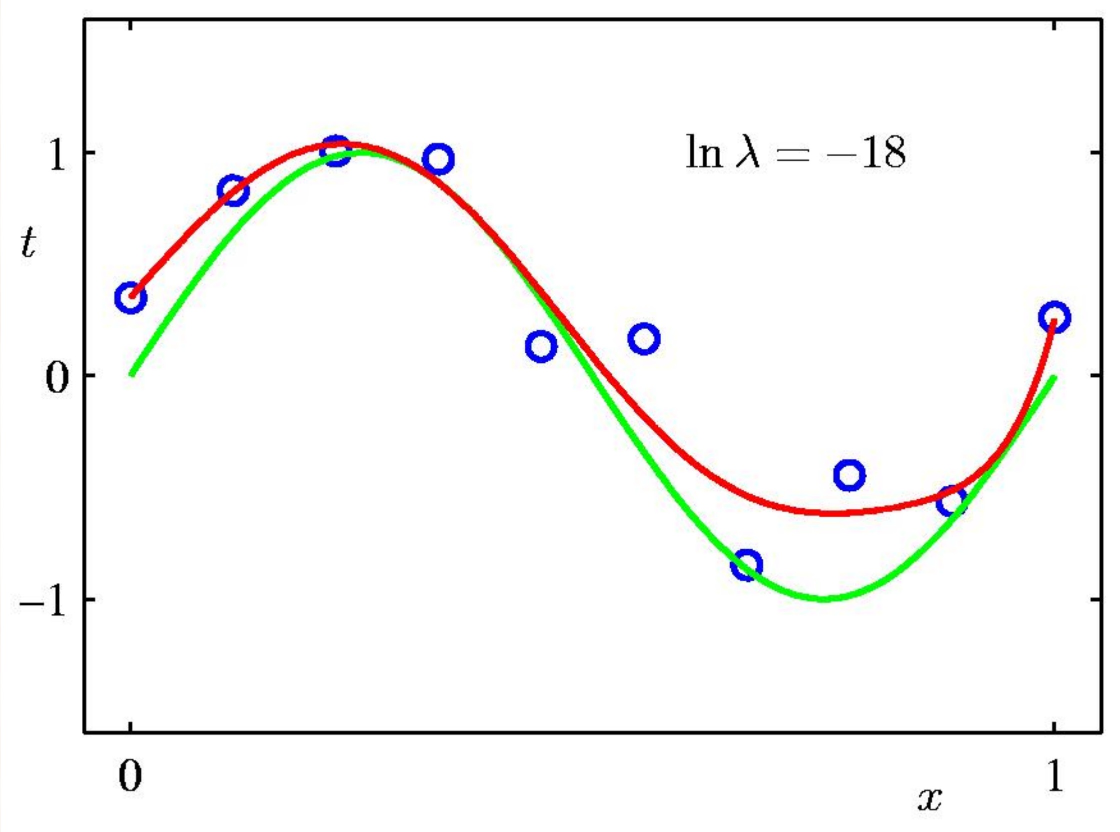
Regularization

- Penalize large parameter values:

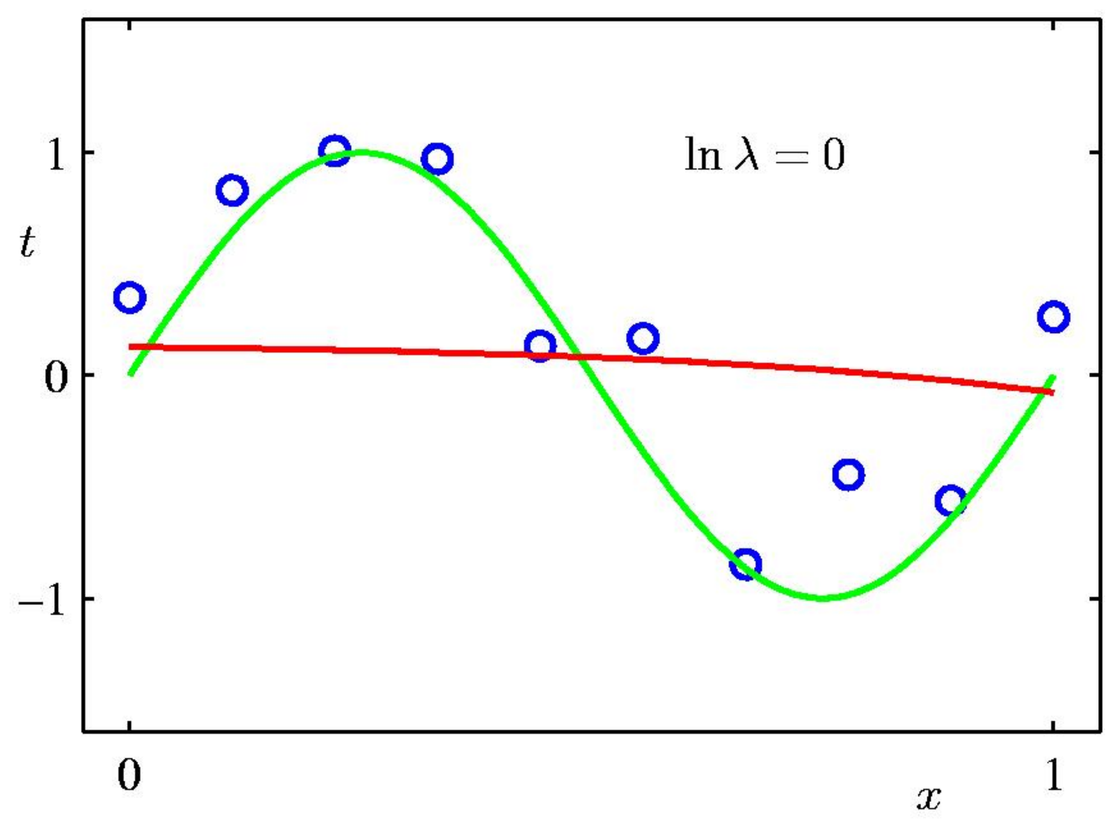
$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_{\mathrm{n}}) - t_n)^2 + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\textit{regularizer}}$$

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

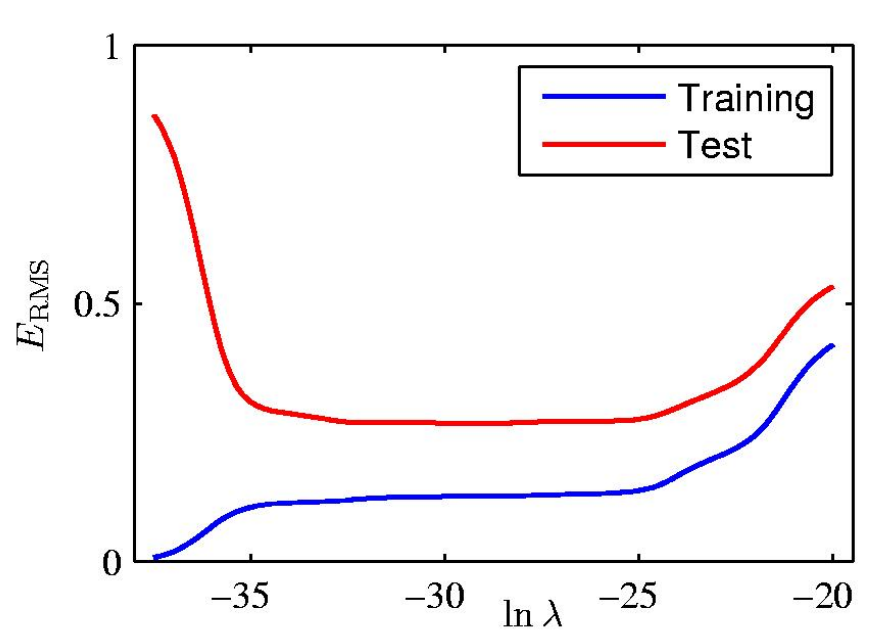
9^{th} Order Polynomial with Regularization



9^{th} Order Polynomial with Regularization



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

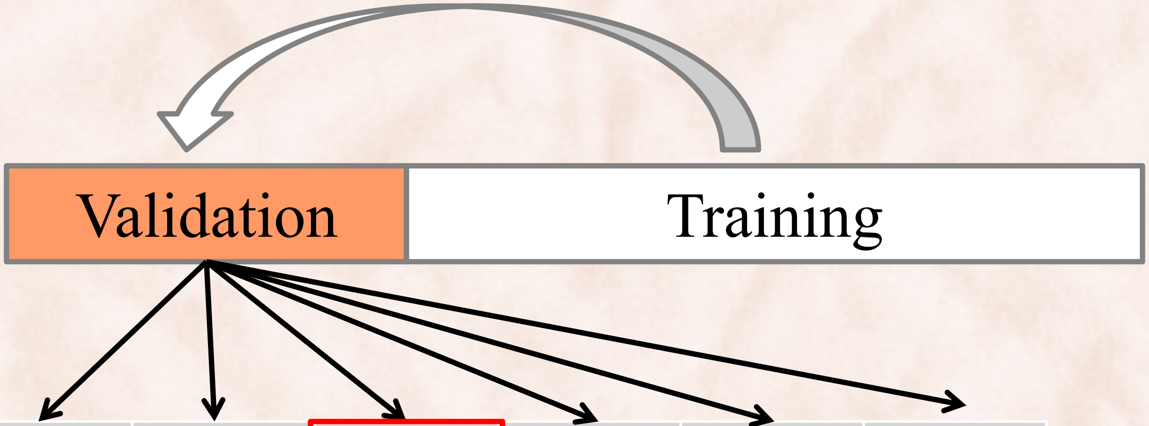


How do we find the optimal value of λ?

Model Selection

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

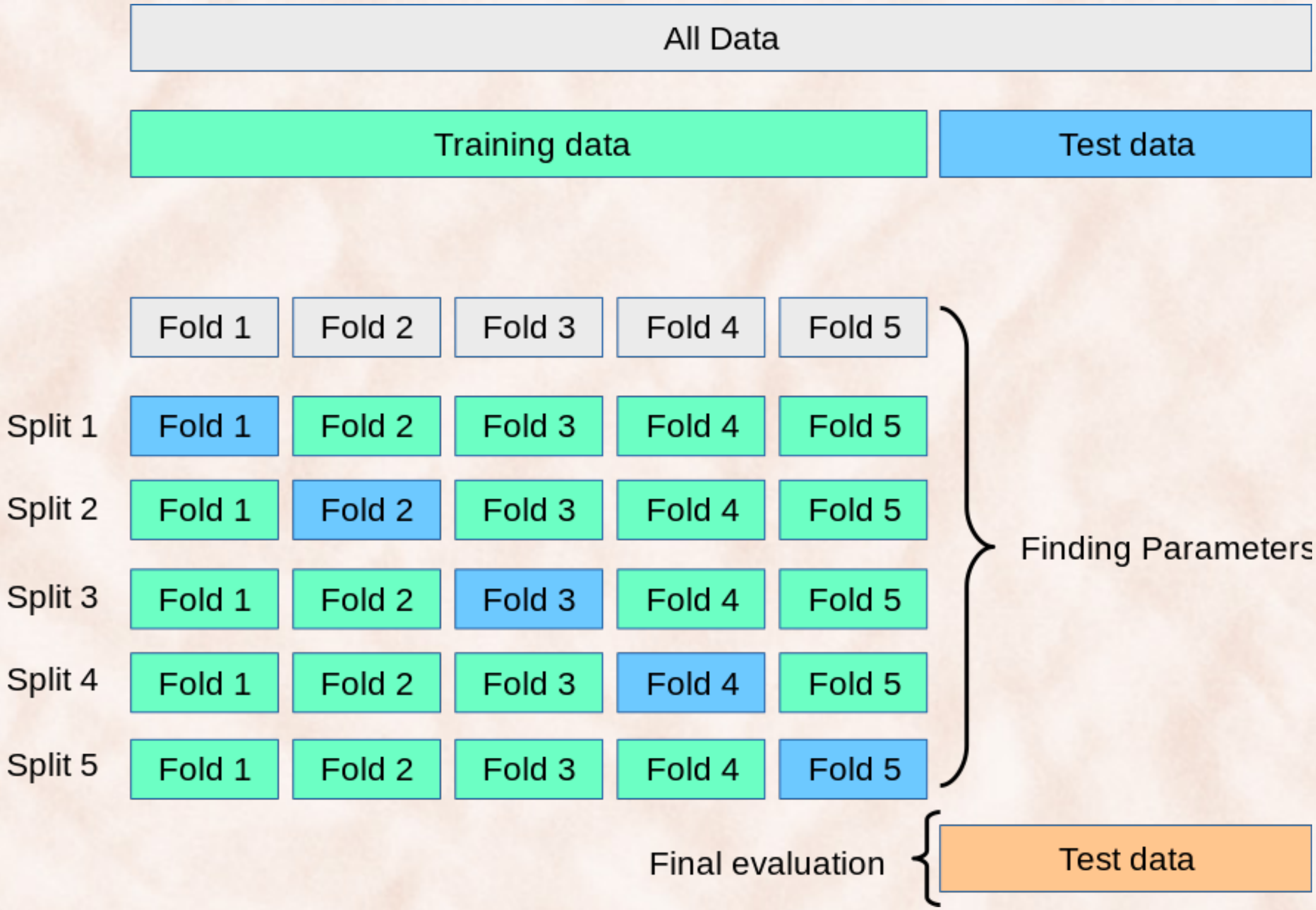
$$\ln \lambda \in \{-40,-35,-30,-25,-20,-15\}$$



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

K-fold Cross-Validation

https://scikit-learn.org/stable/modules/cross_validation.html



All Data

Training data | Test data

	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 1	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 2	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 3	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 4	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5
Split 5	Fold 1	Fold 2	Fold 3	Fold 4	Fold 5

Finding Parameters

Final evaluation | Test data

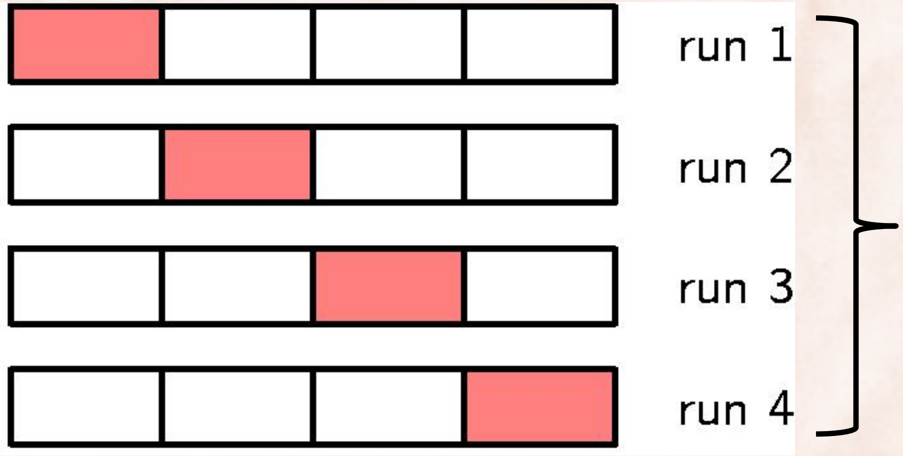
K-fold Cross-Validation

- Split the training data into K folds and try a wide range of tunning parameter values:
  - split the data into K folds of roughly equal size
  - iterate over a set of values for λ
    - iterate over k=1,2,..., K
      - use all folds except k for training
      - validate (calculate test error) in the k-th fold
    - error[λ] = average error over the K folds
  - choose the value of λ that gives the smallest error.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LassoCV.html

Model Evaluation

- K-fold evaluation
  - randomly partition dataset in K equally sized subsets P_1, P_2, … 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.



4-fold evaluation

Multiple Linear Regression

- *Q*: What if the raw feature is insufficient for good performance?
  - Example: house prices depend not only on *floor size*, but also number of *bedrooms*, *age*, *location*, …

- *A*: Use **Multiple Linear Regression**.

Multivariate Linear Regression

- Polynomial curve fitting:

  $\mathbf{x} = [1, x, x^2, ..., x^M]^T$

  $= [x_0, x_1, \ldots, x_M]^T$

  $h(x) = h(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\mathbf{x}$

- **Multiple linear regression**:

  $\mathbf{x} = [x_0, x_1, \ldots, x_M]^T$

  $h(x) = h(\mathbf{x},\mathbf{w}) = \mathbf{w}^T\mathbf{x}$

- Training examples: $(\mathbf{x}^{(1)},t_1)$, $(\mathbf{x}^{(2)},t_2)$, … $(\mathbf{x}^{(N)},t_N)$

Multiple Linear Regression

- **Learning** = minimize the Sum-of-Squares error function:

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w}) \qquad J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} \left(h_{\mathbf{w}}(\mathbf{x}^{(n)}) - t_n\right)^2$$

- Computing the gradient $\nabla J(\mathbf{w})$ and setting it to zero:

$$\sum_{n=1}^{N} \left(\mathbf{w}^{\mathrm{T}}\mathbf{x}^{(n)} - t_n\right) \mathbf{x}^{(n)} = 0$$

- Solving for $\mathbf{w}$ yields $\mathbf{w} = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{t}$
  - Prove it.

Normal Equations

- Solution is $\mathbf{w} = \left(\mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{t}$

- X is the data matrix, or the **design matrix**:

$$X = \begin{pmatrix} \mathbf{x}^{(1)^{\mathrm{T}}} \\ \mathbf{x}^{(2)^{\mathrm{T}}} \\ \dots \\ \dots \\ \mathbf{x}^{(N)^{\mathrm{T}}} \end{pmatrix} = \begin{pmatrix} x_0^{(1)} \; x_1^{(1)} \; \dots \; x_M^{(1)} \\ x_0^{(2)} \; x_1^{(2)} \; \dots \; x_M^{(2)} \\ \dots \\ \dots \\ x_0^{(N)} \; x_1^{(N)} \; \dots \; x_M^{(N)} \end{pmatrix}$$

- $\mathbf{t} = [t_1, t_2, \dots, t_N]^{\mathrm{T}}$ is the vector of labels.

Ridge Regression

- Multiple linear regression with L2 regularization:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_{\mathrm{n}}) - t_n)^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

$$\hat{\mathbf{w}} = \arg\min_{\mathbf{w}} J(\mathbf{w})$$

- Solution is $\mathbf{w} = \left(\lambda N\mathbf{I} + \mathrm{X}^{\mathrm{T}}\mathrm{X}\right)^{-1}\mathrm{X}^{\mathrm{T}}\mathbf{t}$
  - Prove it.

Regularization: Ridge vs. Lasso

- Ridge regression:

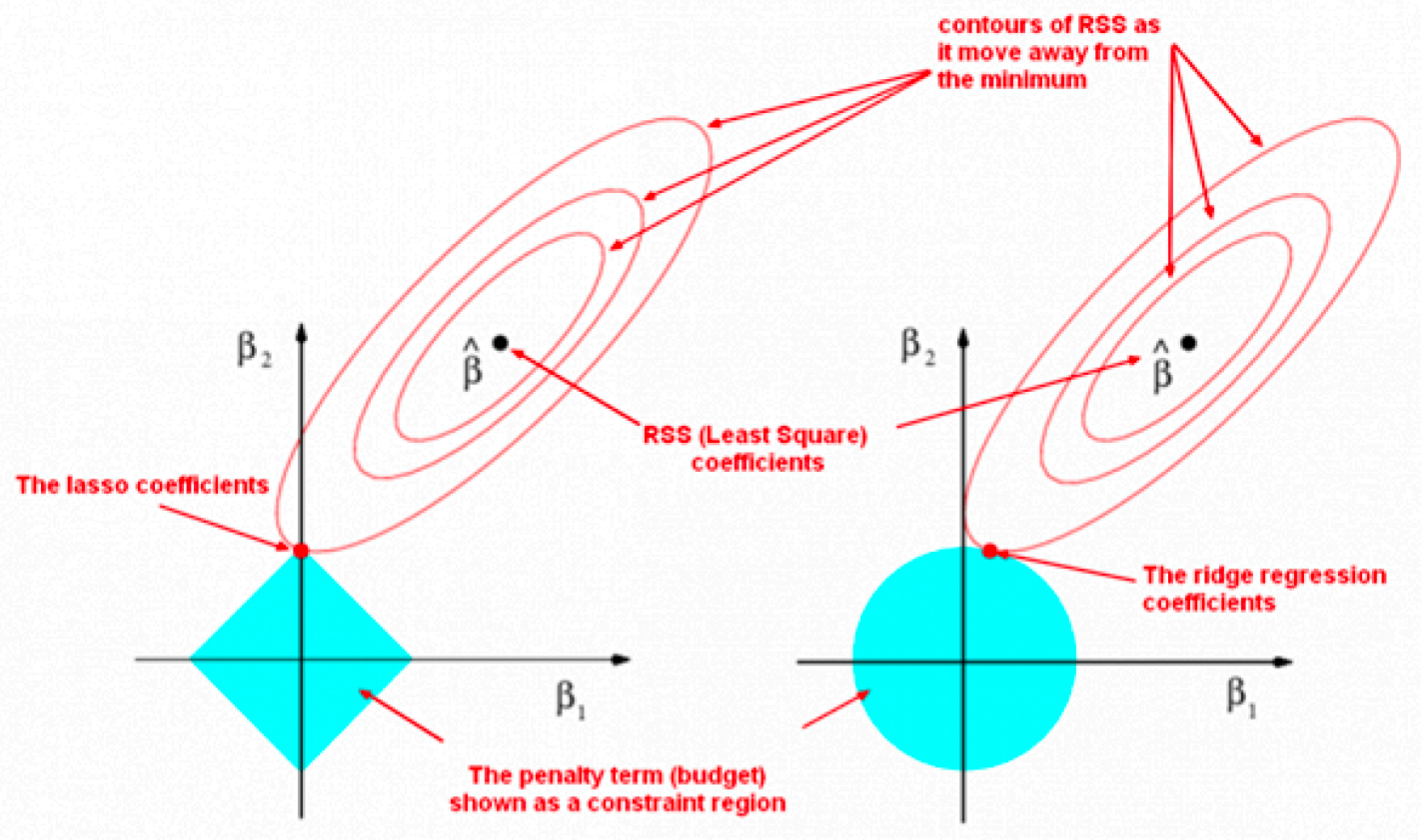
$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_{\mathrm{n}}) - t_n)^2 + \frac{\lambda}{2}\sum_{j=1}^{M} w_j^2$$

- Lasso:

$$J(\mathbf{w}) = \frac{1}{2N}\sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_{\mathrm{n}}) - t_n)^2 + \frac{\lambda}{2}\sum_{j=1}^{M} |w_j|$$

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

Regularization: Ridge vs. Lasso



Regularization

- Regularization alleviates overfitting when using models with high capacity (e.g. high degree polynomials):
  - Want high capacity because we do not know how complicated the data is.

- *Q*: Can we achieve high capacity when doing curve fitting without using high degree polynomials?

- *A*: Use piecewise polynomial curves.
  - Example: **Cubic spline smoothing**.

Cubic Spline Smoothing

- **Cubic spline smoothing** is a regularized version of cubic spline interpolation.
  - Cubic spline interpolation: given *n* points $\{(x_i, y_i)\}$, connect adjacent points using cubic functions S_i, requiring that the spline and its first and second derivative remain continuous at all points:

$$S_i(x) = a_i(x-x_i)^3+b_i(x-x_i)^2+c_i(x-x_i)+d_i, \forall x \in [x_i, x_{i+1}]$$

  - **Cubic spline smoothing**: the spline $S = \{S_i\}$ is allowed to deviate from the data points and has low curvature => constrained optimization problem with objective:

$$L = \sum_{i=1}^{n} \frac{w_i}{Z}(S_i(x_i) - y_i)^2 + \frac{\lambda}{x_n - x_1} \int_{x_1}^{x_n} |S''(x)|^2 dx$$

$$w_i = \begin{cases} C, & \text{if } (x_i, y_i) \text{ is a significant local optima} \\ 1, & \text{otherwise} \end{cases}$$

Cubic Spline Smoothing

http://ace.cs.ohio.edu/~razvan/papers/icmla11.pdf

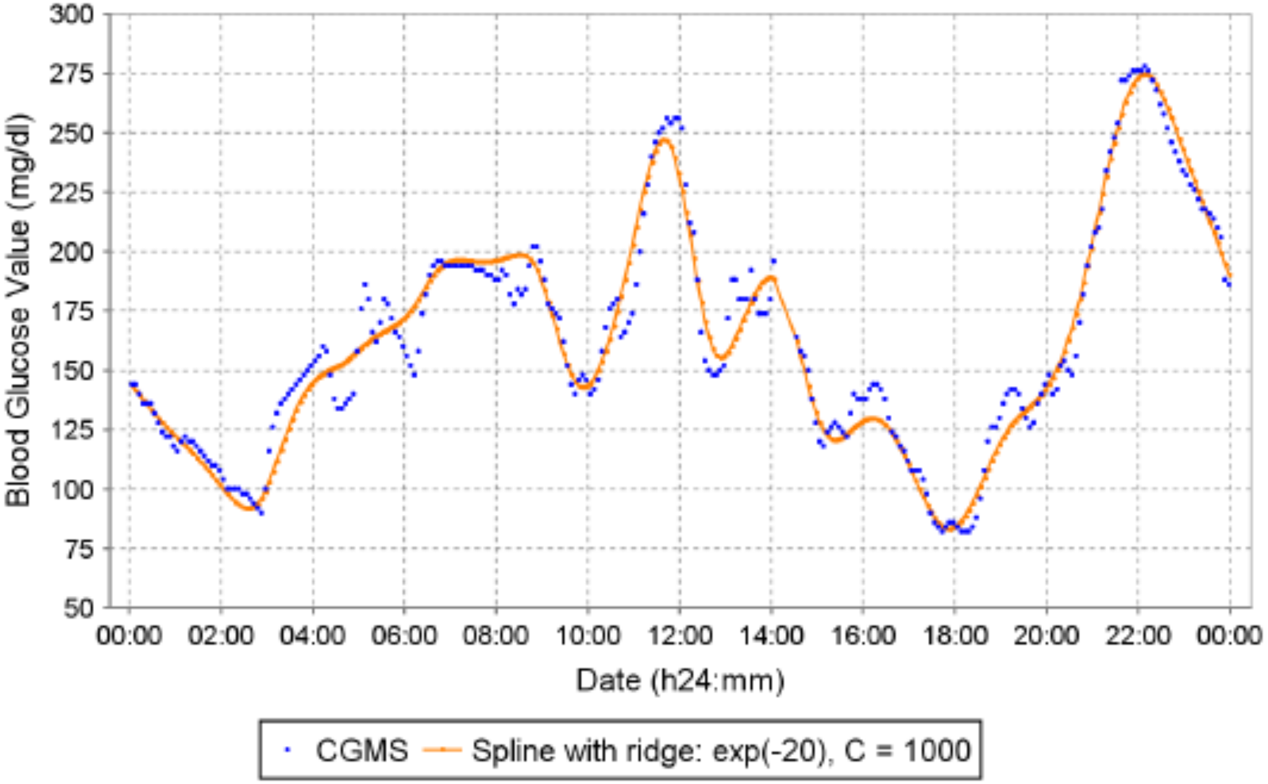
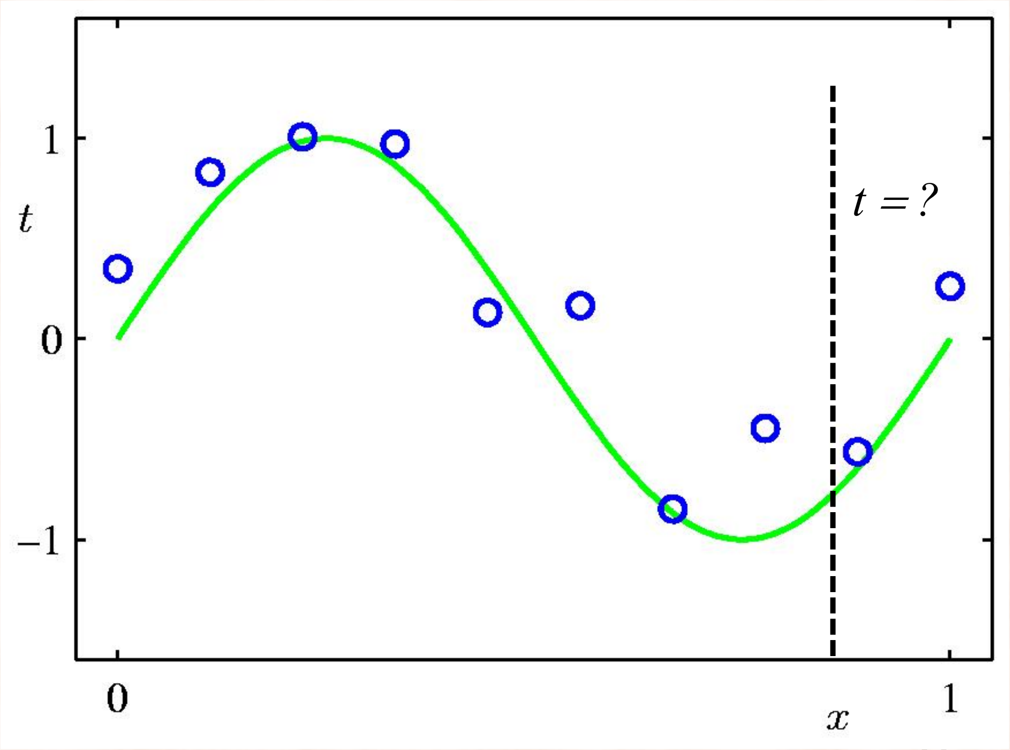


Fig. 3. Cubic spline smoothing with $\lambda = e^{-20}$ and $C = 1000$.

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.

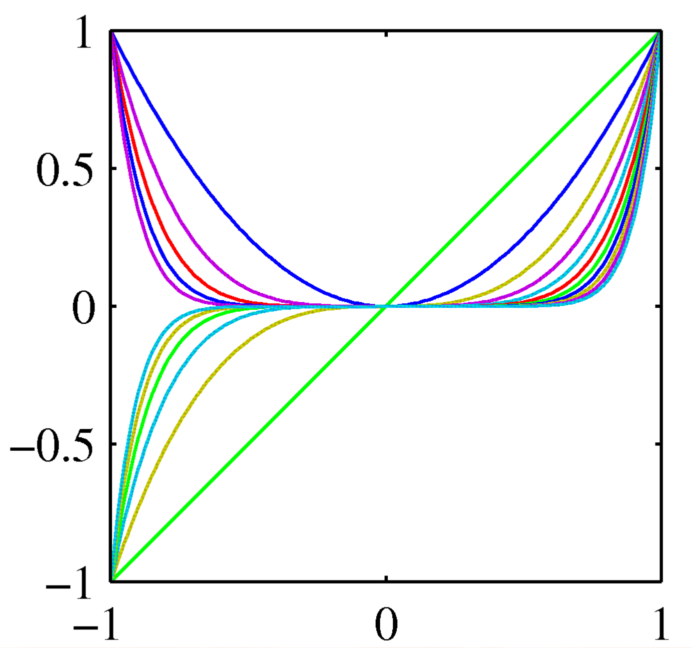
- In the simplest case, use linear basis functions : $\varphi_d(\mathbf{x}) = x_d$.

Linear Basis Function Models (1)

- Polynomial basis functions:

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

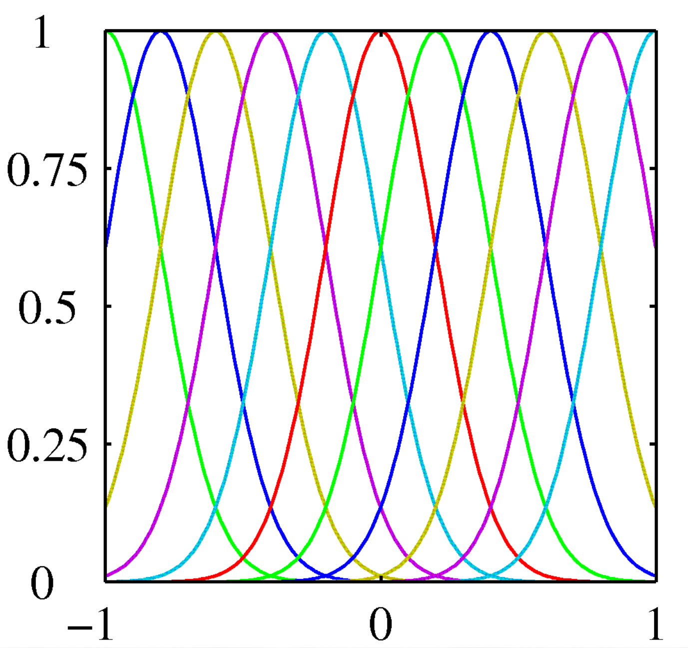
- Global behavior:
  - a small change in *x* affect all basis functions.



Linear Basis Function Models (2)

- Gaussian basis functions:

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



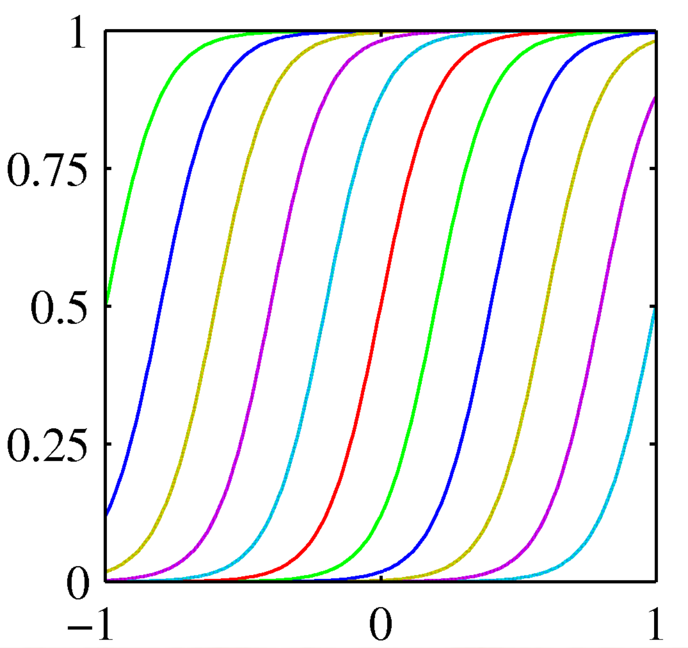
- Local behavior:
  - a small change in x only affects nearby basis functions.
  - μ_j and s control location and scale (width).

Linear Basis Function Models (3)

- Sigmoidal basis functions:

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

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



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