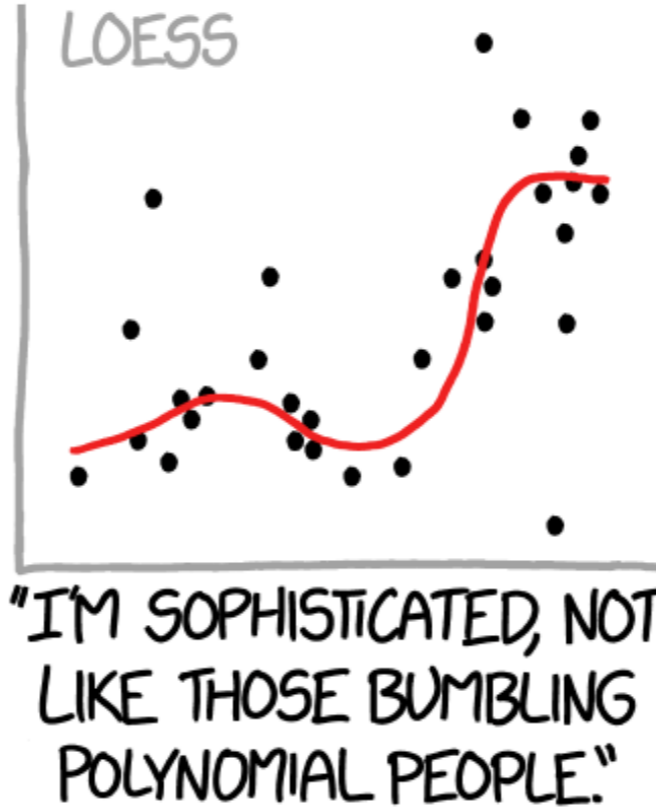
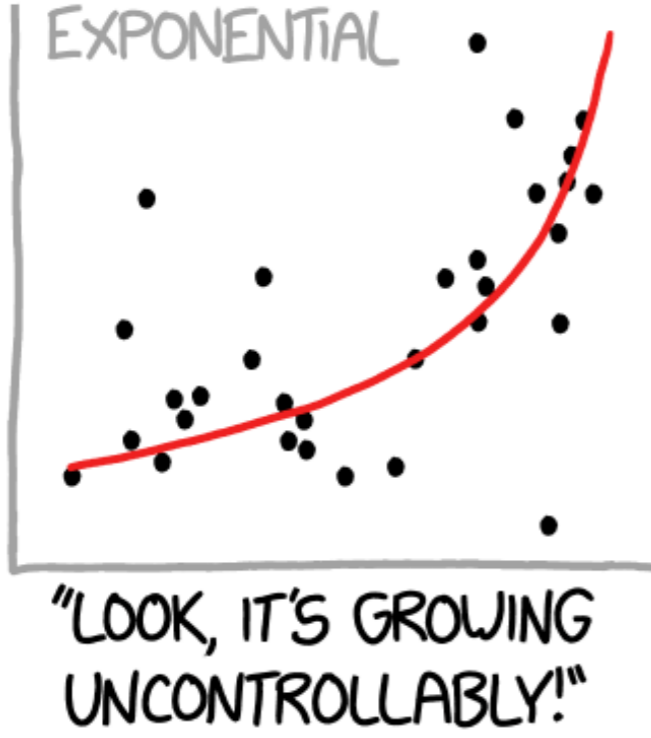
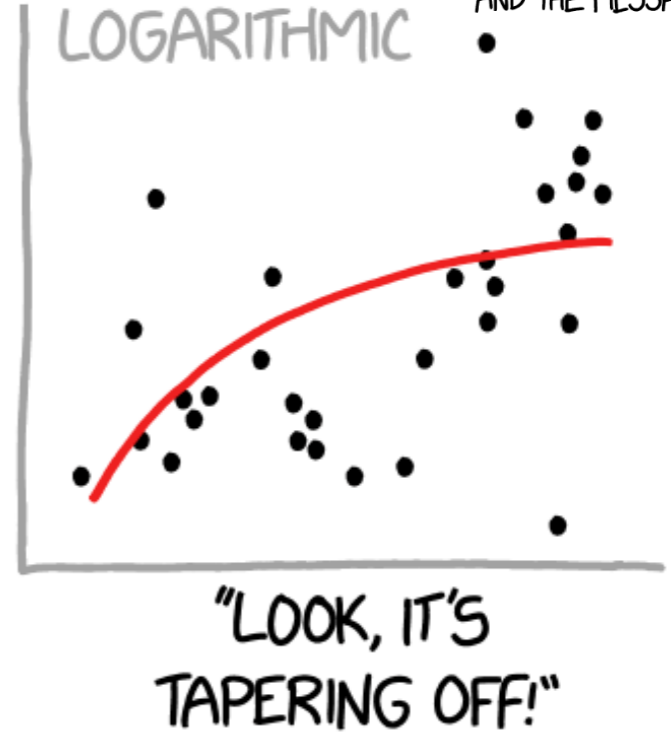
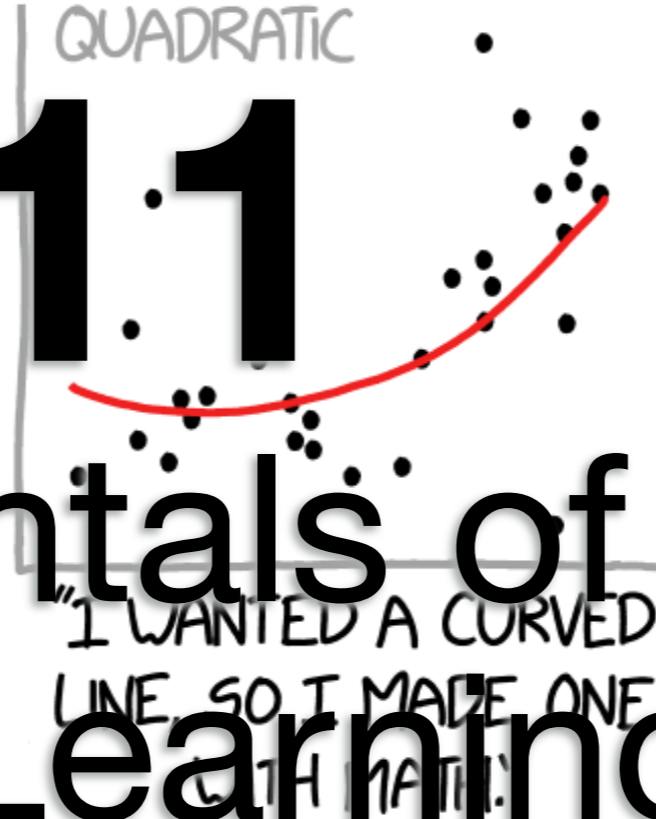
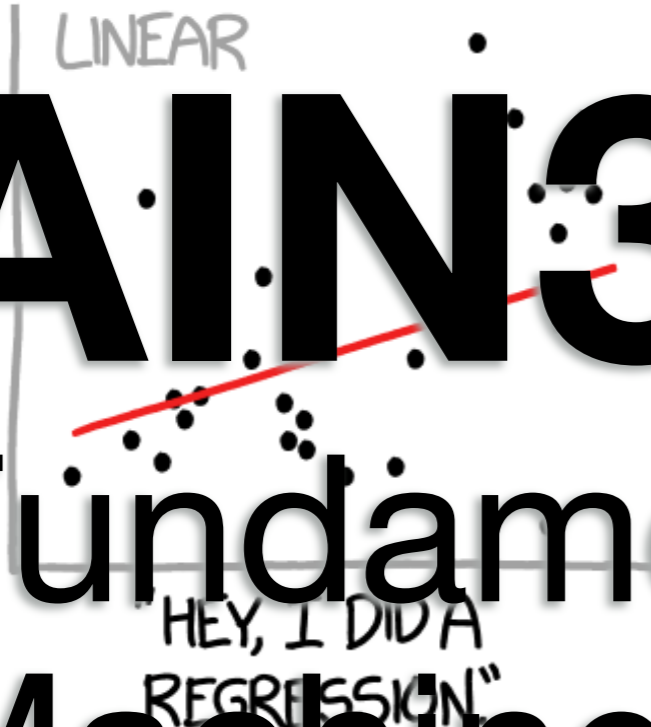


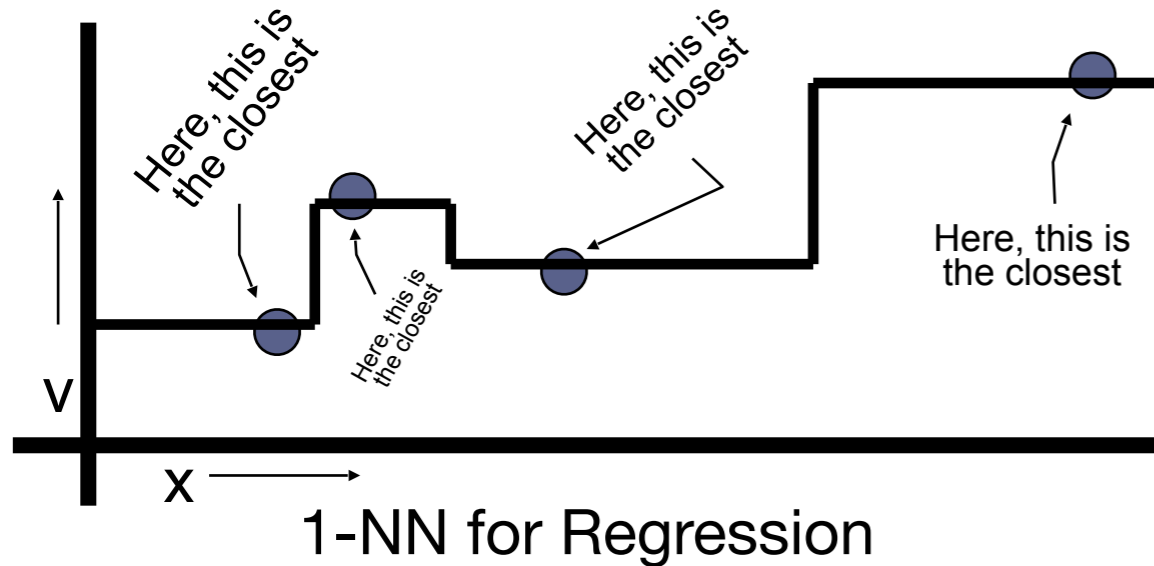
AIN311

Fundamentals of Machine Learning



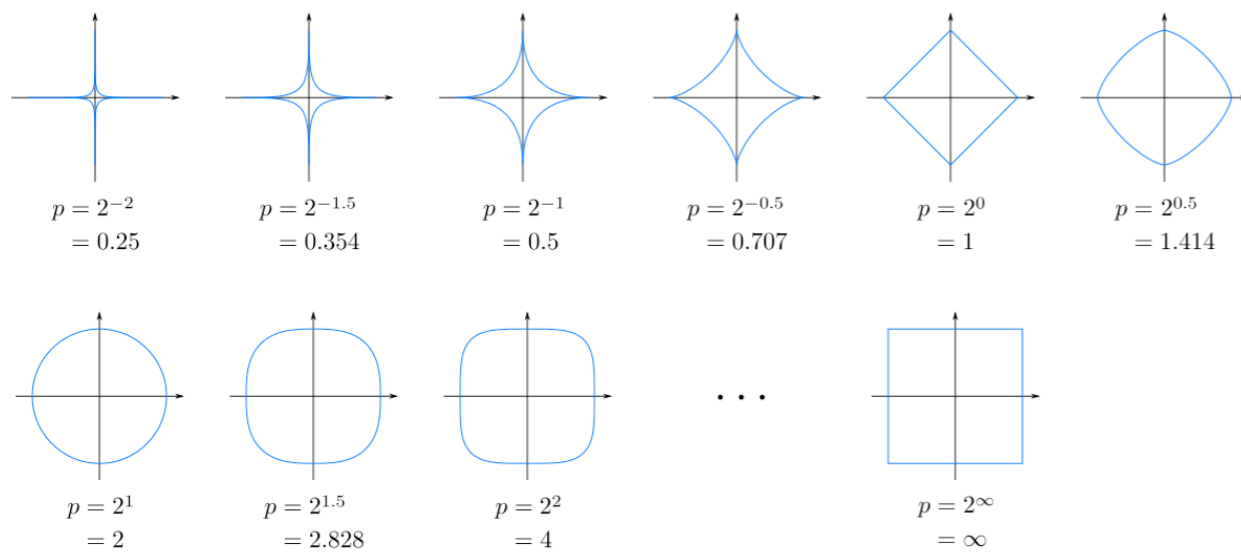
Lecture 4: Linear Regression, Optimization, Generalization, Model complexity, Regularization

Recall from last time... Kernel Regression



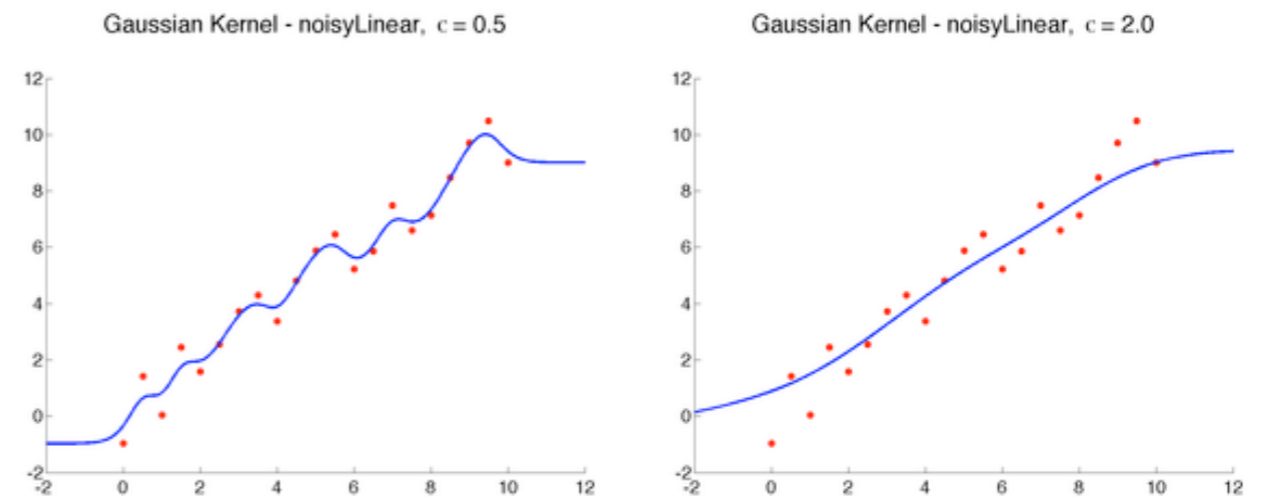
$$h(\vec{x}') = \frac{\sum_{i \in \text{knn}(\vec{x}')} y_i K(\vec{x}_i, \vec{x}')}{\sum_{i \in \text{knn}(\vec{x}')} K(\vec{x}_i, \vec{x}')}$$

Weighted K-NN for Regression



$$D = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

Distance metrics

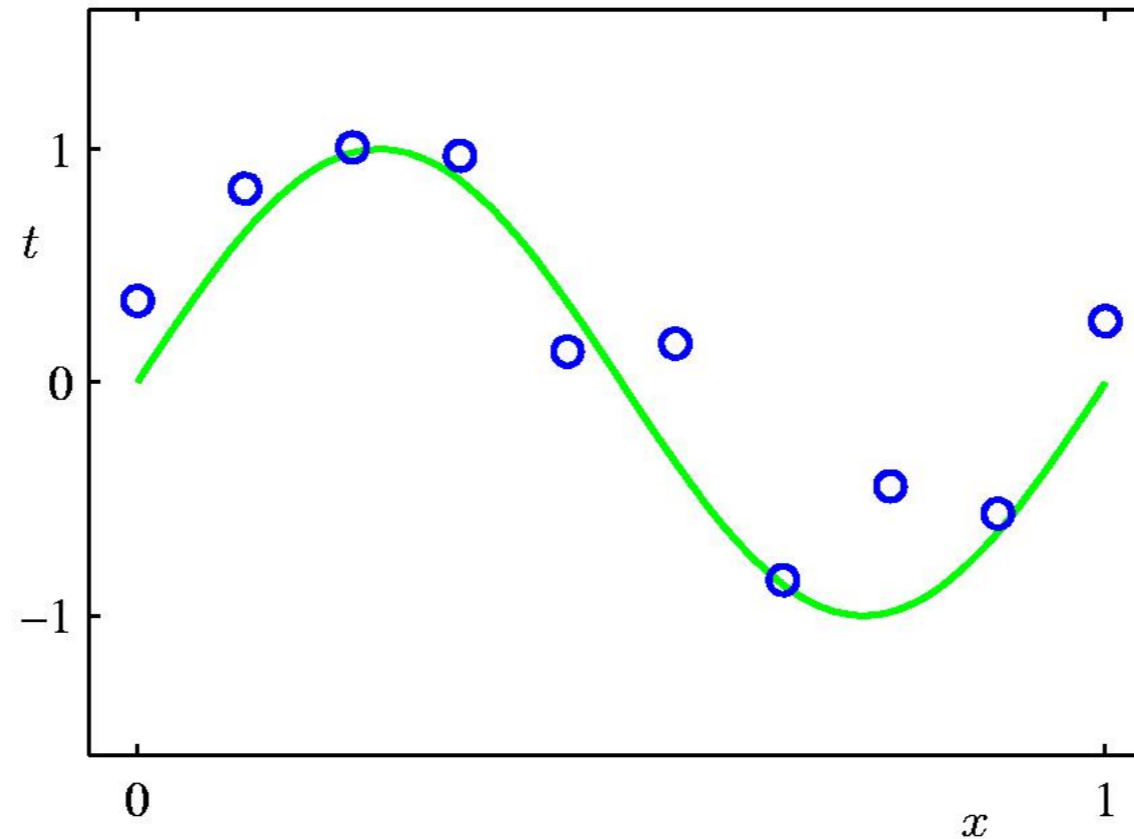


$$w_i = \exp(-d(x_i, \text{query})^2 / \sigma^2)$$

Kernel width

Linear Regression

Simple 1-D Regression



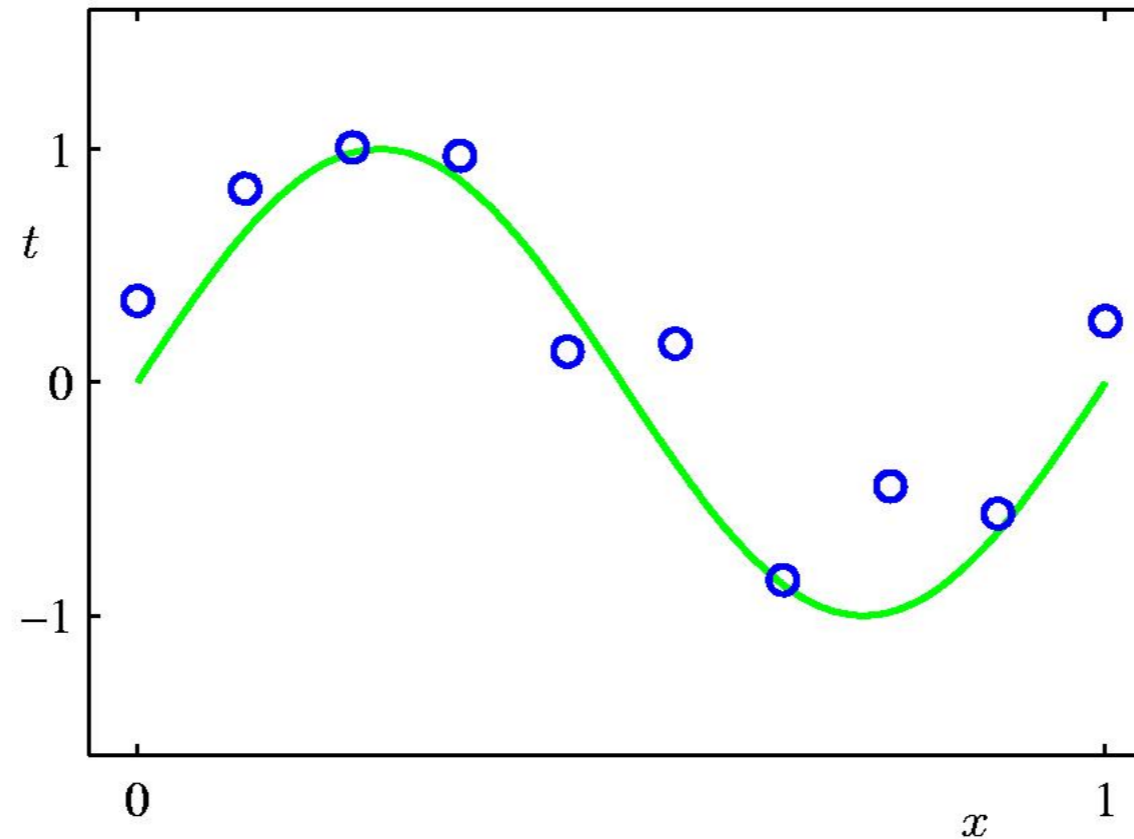
- Circles are data points (i.e., training examples) that are given to us
- The data points are uniform in x , but may be displaced in y

$$t(x) = f(x) + \varepsilon$$

with ε some noise

- In **green** is the “true” curve that we don’t know
- **Goal: We want to fit a curve to these points**

Simple 1-D Regression

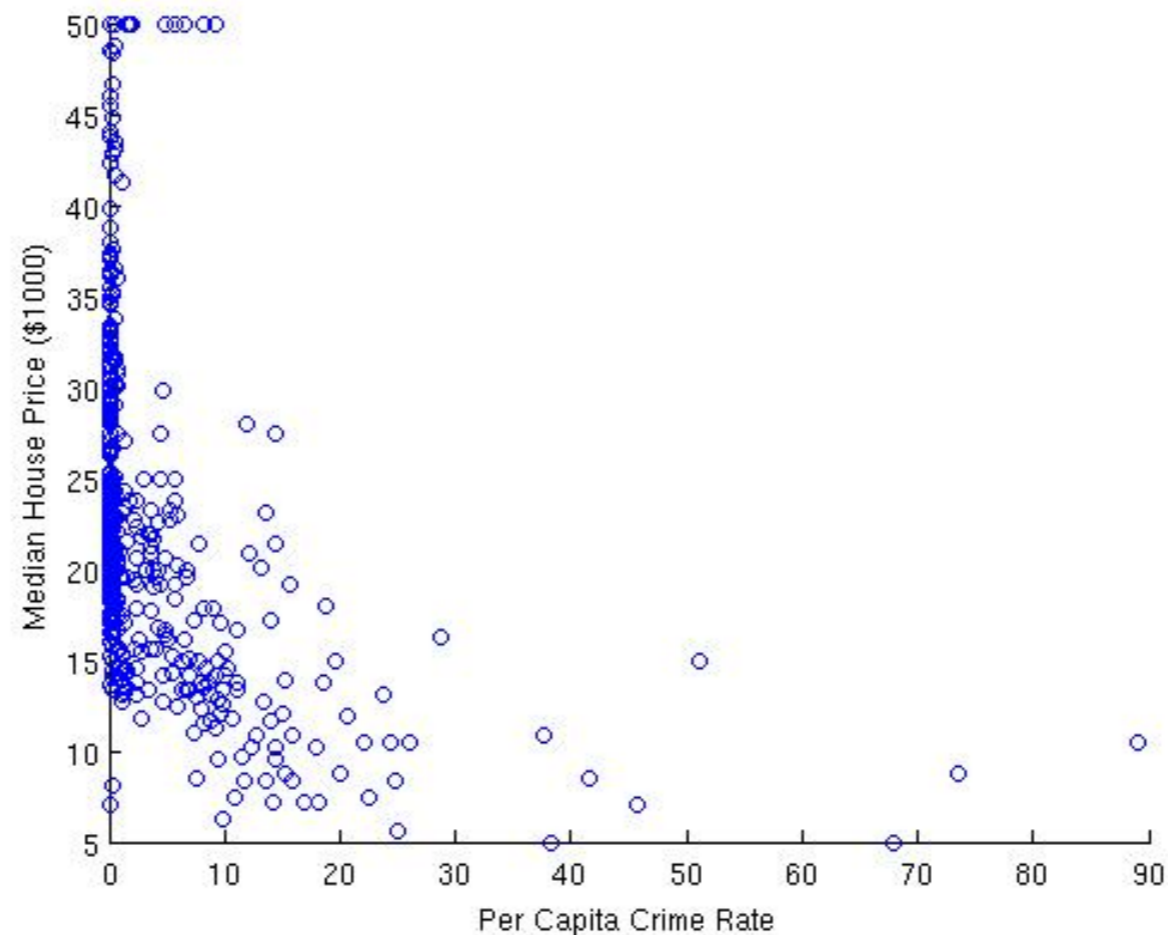


- **Key Questions:**

- How do we parametrize the **model** (the curve)?
- What **loss (objective) function** should we use to judge fit?
- How do we optimize fit to unseen test data (**generalization**)?

Example: Boston House Prices

- Estimate median house price in a neighborhood based on neighborhood statistics
- Look at first (of 13) attributes: per capita crime rate



- Use this to predict house prices in other neighborhoods
- Is this a **good input (attribute) to predict** house prices?

Represent the data

- Data described as pairs $D = \{(x_{(1)}, t_{(1)}), (x_{(2)}, t_{(2)}), \dots, (x_{(N)}, t_{(N)})\}$
 - x is the **input feature** (per capita crime rate)
 - t is the **target output** (median house price)
 - (i) simply indicates the training examples (we have N in this case)
- Here t is continuous, so this is a **regression problem**
- Model outputs y , an estimate of t

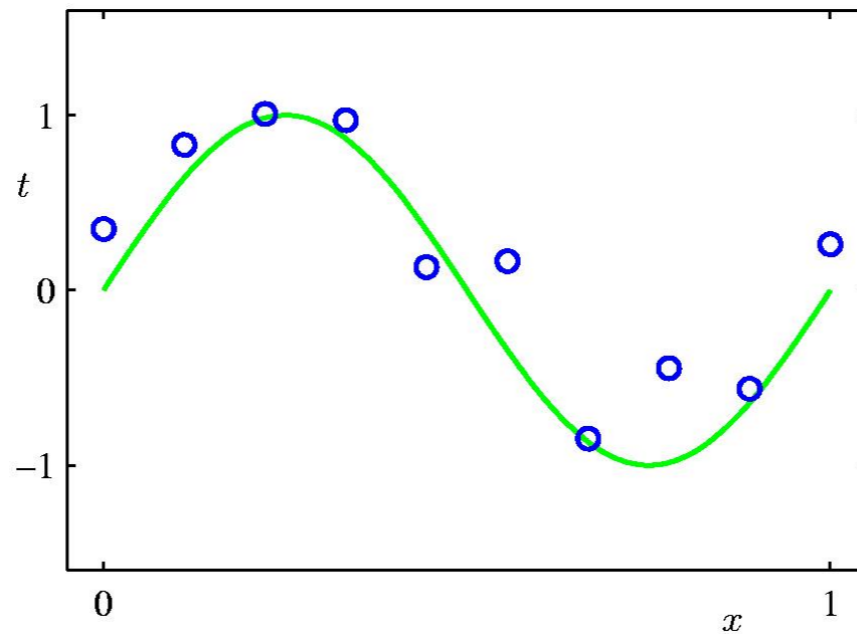
$$y(x) = w_0 + w_1x$$

- What type of **model** did we choose?
- Divide the dataset into training and testing examples
 - Use the training examples to construct hypothesis, or function approximator, that maps x to predicted y
 - Evaluate hypothesis on test set

Noise

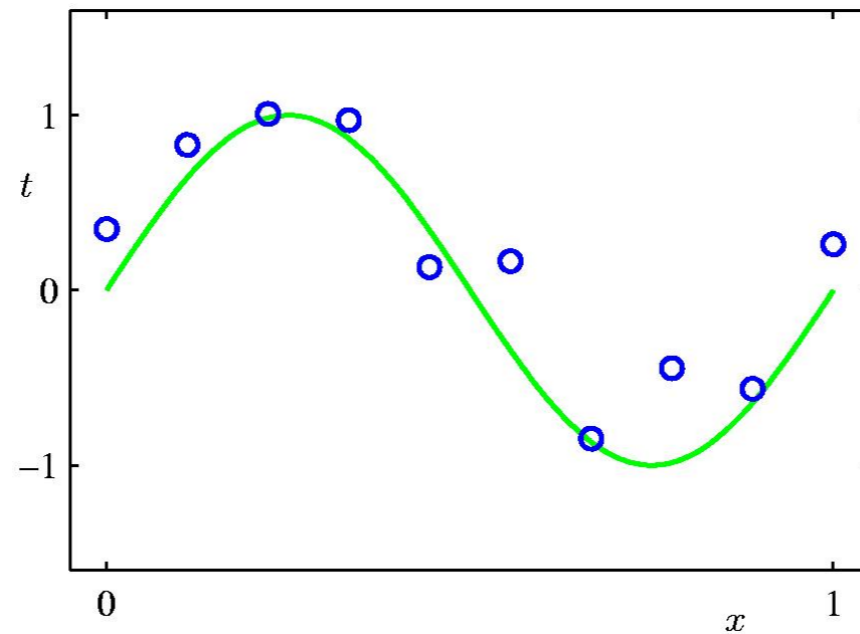
- A simple model typically does not exactly fit the data — lack of fit can be considered noise
- **Sources of noise:**
 - Imprecision in data attributes (input noise, e.g. noise in per-capita crime)
 - Errors in data targets (mislabeling, e.g. noise in house prices)
 - Additional attributes not taken into account by data attributes, affect target values (latent variables). In the example, what else could affect house prices?
 - Model may be too simple to account for data targets

Least-Squares Regression



$$y(x) = \text{function}(x, \mathbf{w})$$

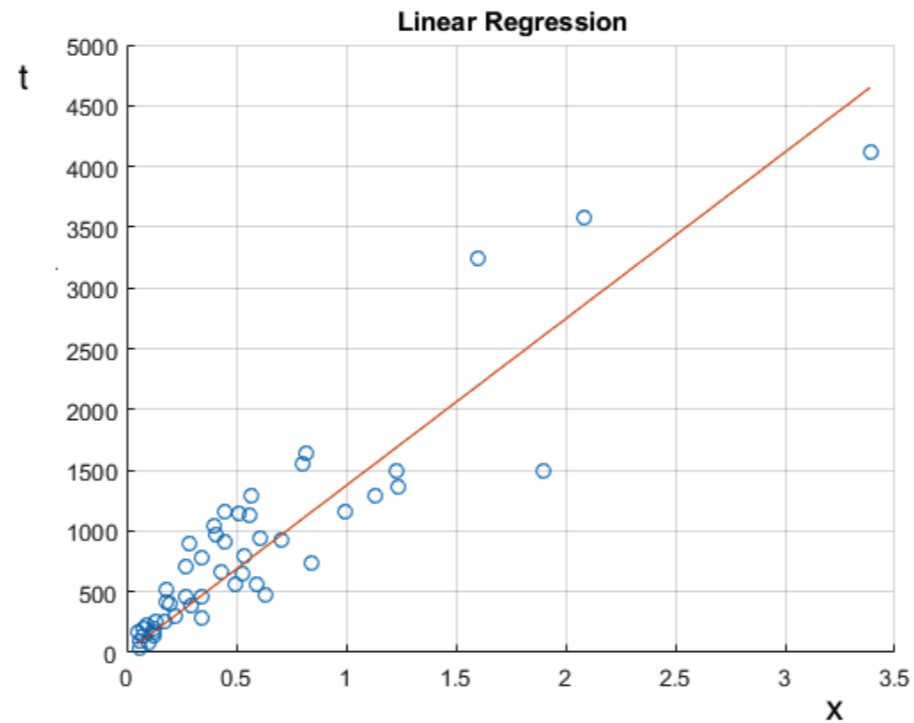
Least-Squares Regression



- Define a model

Linear: $y(x) = \text{function}(x, \mathbf{w})$

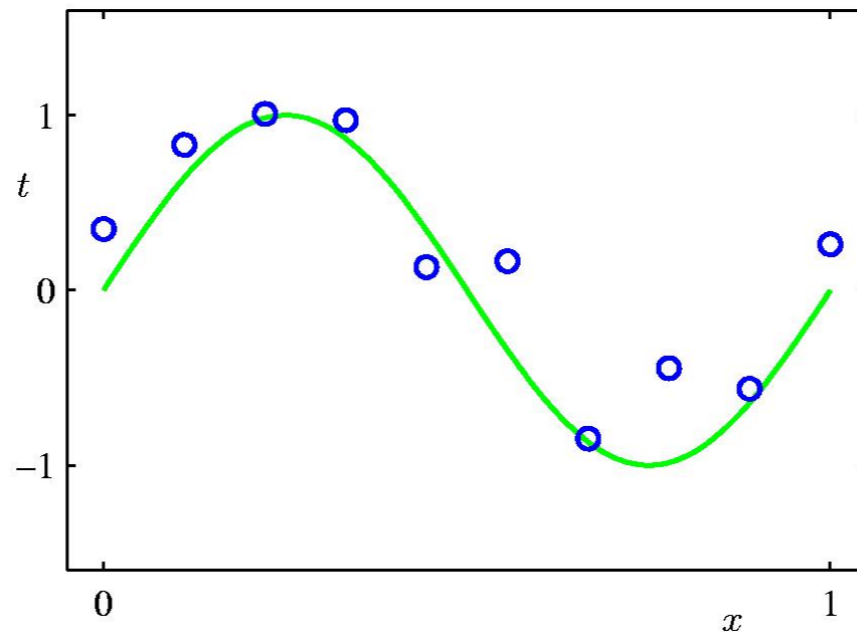
Least-Squares Regression



- Define a model

Linear: $y(x) = w_0 + w_1x$

Least-Squares Regression



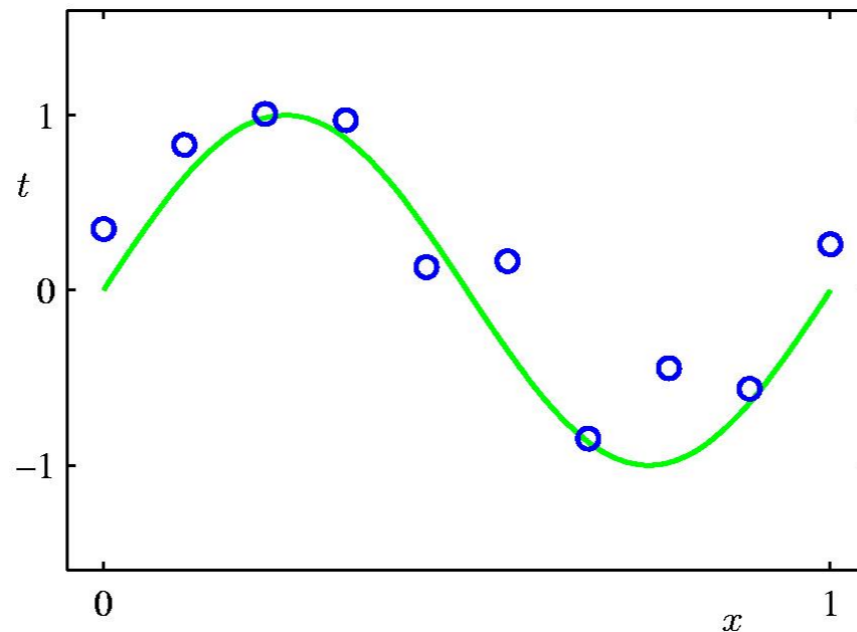
- Define a model

Linear: $y(x) = w_0 + w_1 x$

- Standard loss/cost/objective function measures the squared error between y and the true value t

$$\ell(\mathbf{w}) = \sum_{n=1}^N \left[t^{(n)} - y(x^{(n)}) \right]^2$$

Least-Squares Regression



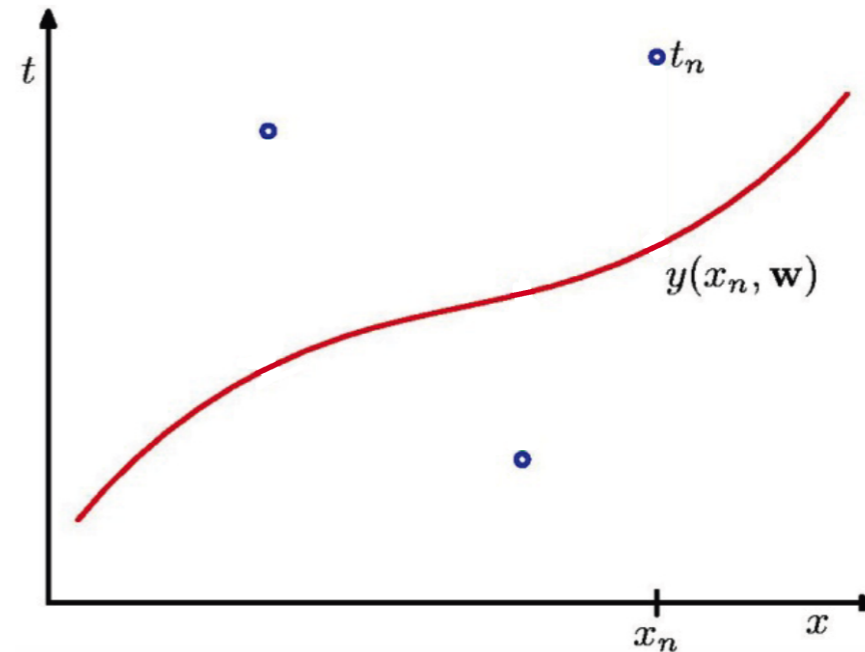
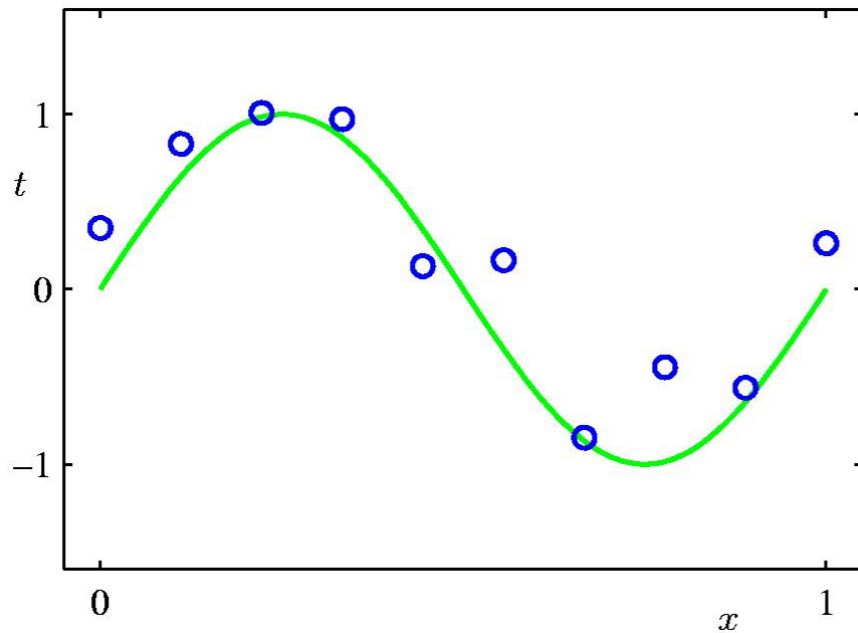
- Define a model

Linear: $y(x) = w_0 + w_1 x$

- Standard loss/cost/objective function measures the squared error between y and the true value t

Linear model:
$$\ell(\mathbf{w}) = \sum_{n=1}^N \left[t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2$$

Least-Squares Regression



- Define a model

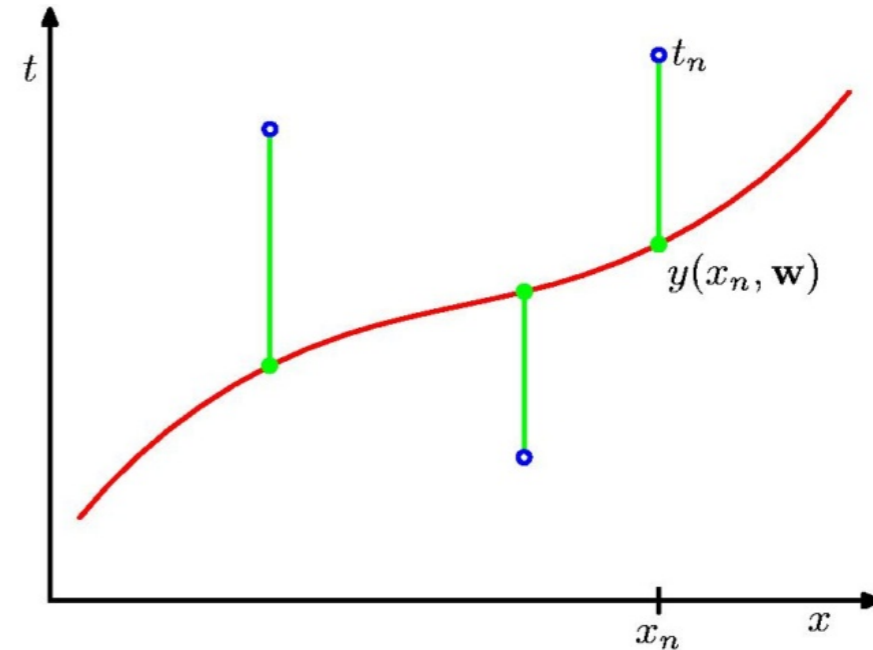
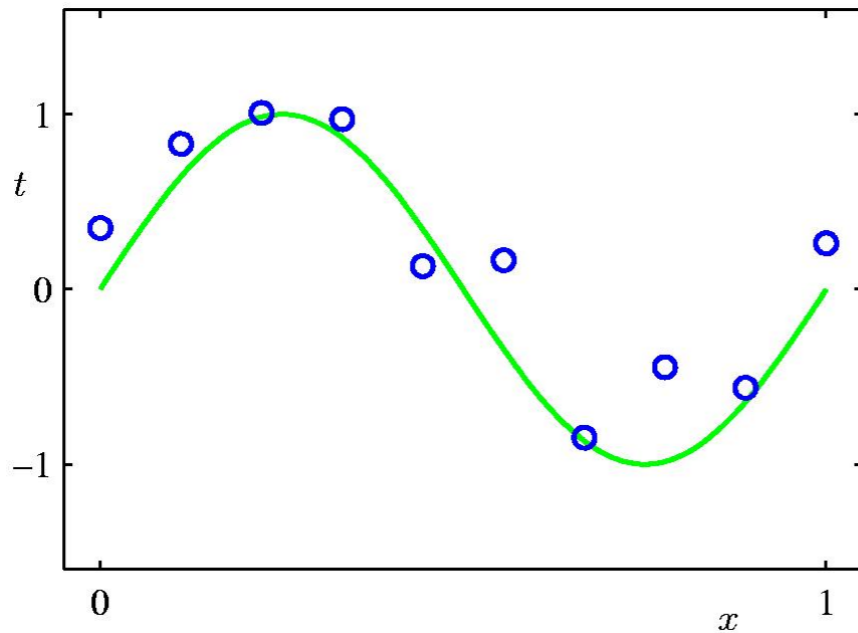
Linear: $y(x) = w_0 + w_1 x$

- Standard loss/cost/objective function measures the squared error between y and the true value t

Linear model:
$$\ell(\mathbf{w}) = \sum_{n=1}^N \left[t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2$$

- For a particular hypothesis ($y(x)$ defined by a choice of \mathbf{w} , drawn in red), what does the loss represent geometrically?

Least-Squares Regression



- Define a model

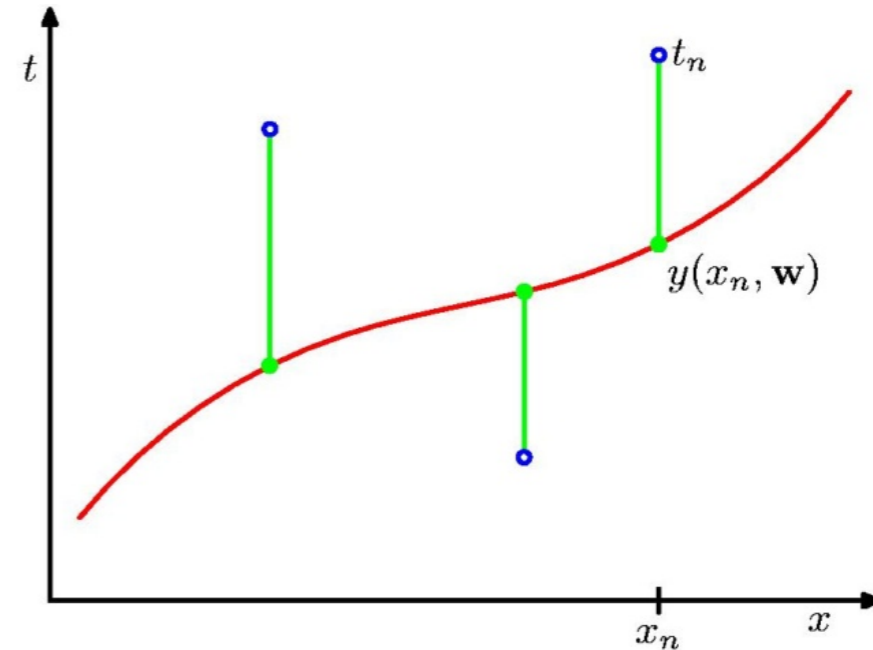
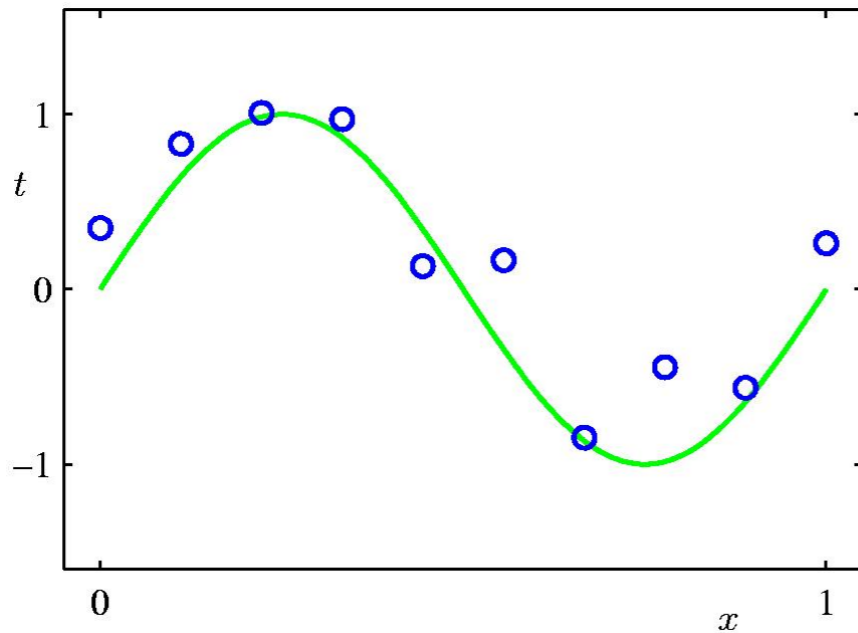
Linear: $y(x) = w_0 + w_1 x$

- Standard loss/cost/objective function measures the squared error between y and the true value t

Linear model:
$$\ell(\mathbf{w}) = \sum_{n=1}^N \left[t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2$$

- The loss for the red hypothesis is the **sum of the squared vertical errors** (squared lengths of green vertical lines)

Least-Squares Regression



- Define a model

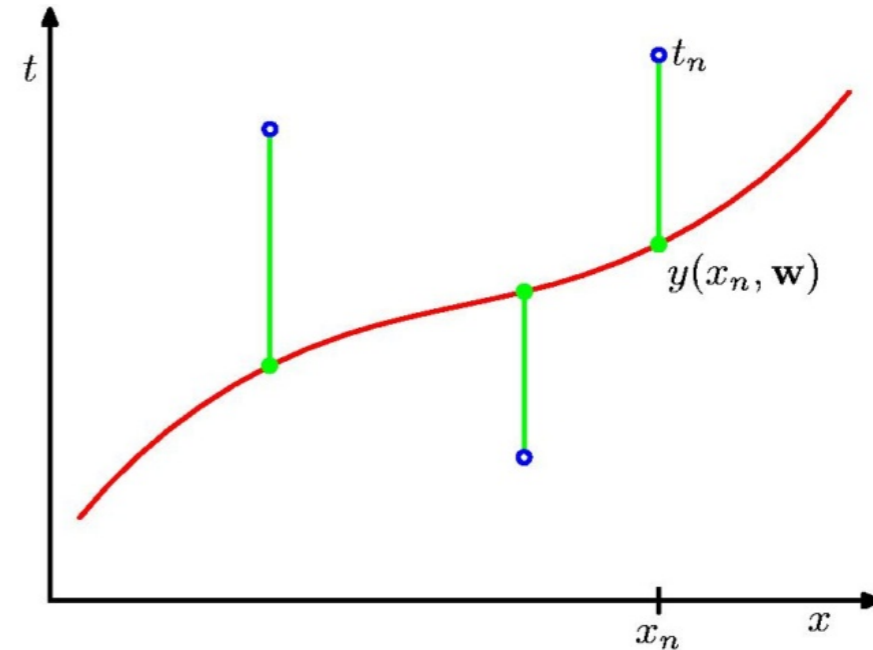
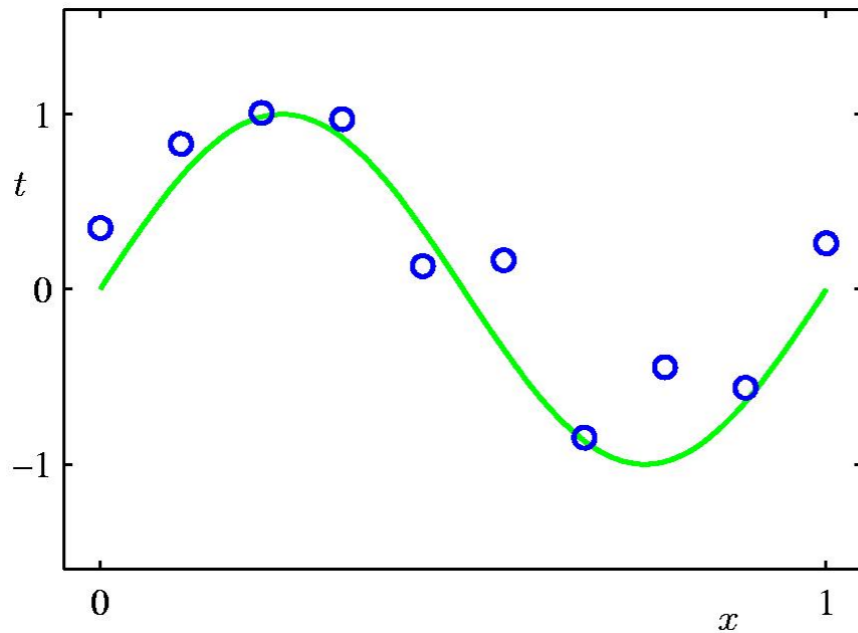
Linear: $y(x) = w_0 + w_1 x$

- Standard loss/cost/objective function measures the squared error between y and the true value t

Linear model:
$$\ell(\mathbf{w}) = \sum_{n=1}^N \left[t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2$$

- How do we obtain weights $\mathbf{w} = (w_0, w_1)$?

Least-Squares Regression



- Define a model

Linear: $y(x) = w_0 + w_1 x$

- Standard loss/cost/objective function measures the squared error between y and the true value t

Linear model:
$$\ell(\mathbf{w}) = \sum_{n=1}^N \left[t^{(n)} - (w_0 + w_1 x^{(n)}) \right]^2$$

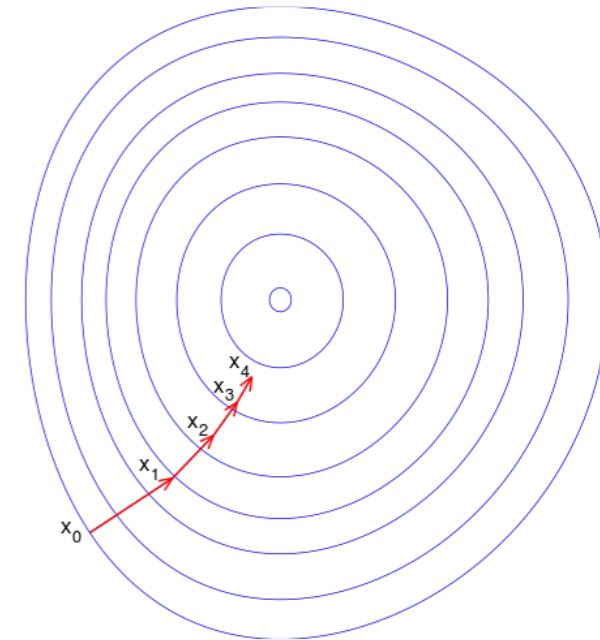
- How do we obtain weights $\mathbf{w} = (w_0, w_1)$? Find \mathbf{w} that minimizes loss $\ell(\mathbf{w})$

Optimizing the Objective

- One straightforward method: **gradient descent**

- initialize \mathbf{w} (e.g., randomly)
- repeatedly update \mathbf{w} based on the gradient

$$\mathbf{w} \leftarrow \mathbf{w} - \lambda \frac{\partial \ell}{\partial \mathbf{w}}$$



- λ is the **learning rate**
- For a single training case, this gives the **LMS update rule:**

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \underbrace{\left(t^{(n)} - y(x^{(n)}) \right)}_{\text{error}} x^{(n)}$$

- Note: As error approaches zero, so does the update (\mathbf{w} stops changing)

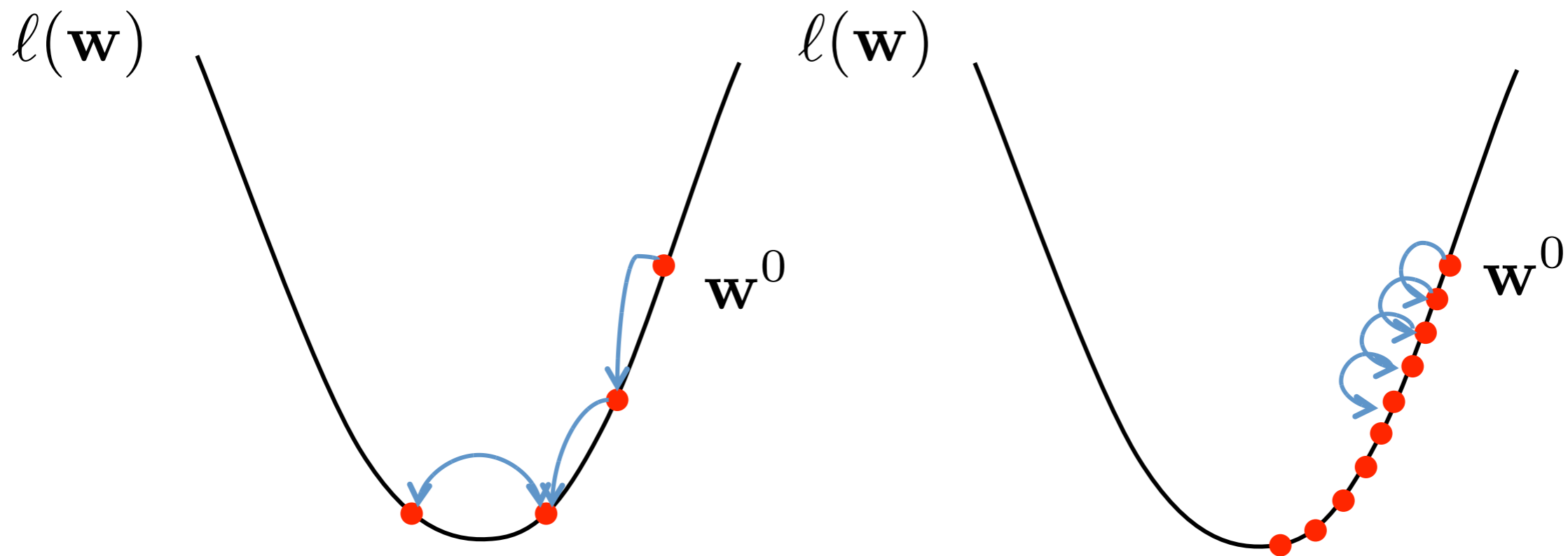
Optimizing the Objective



Optimizing the Objective



Effect of learning rate λ



- Large $\lambda \Rightarrow$ Fast convergence but larger residual error
Also possible oscillations
- Small $\lambda \Rightarrow$ Slow convergence but small residual error

Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:

1. **Batch updates:** sum or average updates across every example n , then change the parameter values

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \left(t^{(n)} - y(x^{(n)}) \right) x^{(n)}$$

2. **Stochastic/online updates:** update the parameters for each training case in turn, according to its own gradients

Algorithm 1 Stochastic gradient descent

- 1: Randomly shuffle examples in the training set
- 2: **for** $i = 1$ to N **do**
- 3: Update:

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda(t^{(i)} - y(x^{(i)}))x^{(i)} \quad \text{(update for a linear model)}$$

- 4: **end for**

Optimizing Across Training Set

- Two ways to generalize this for all examples in training set:
 1. **Batch updates:** sum or average updates across every example n , then change the parameter values

$$\mathbf{w} \leftarrow \mathbf{w} + 2\lambda \left(t^{(n)} - y(x^{(n)}) \right) x^{(n)}$$

2. **Stochastic/online updates:** update the parameters for each training case in turn, according to its own gradients
- Underlying assumption: sample is independent and identically distributed (i.i.d.)

Analytical Solution

- For some objectives we can also find the optimal solution analytically
- This is the case for linear least-squares regression
- How?

Vectorization

- Consider our model:

$$y(x) = w_0 + w_1 x$$

- Let

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \quad \mathbf{x}^T = [1 \quad x]$$

- Can write the model in vectorized form as

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

Vectorization

- Consider our model with N instances:

$$\mathbf{t} = \left[t^{(1)}, t^{(2)}, \dots, t^{(N)} \right]^T \xrightarrow{\text{blue arrow}} \mathbb{R}^{N \times 1}$$

$$\mathbf{X} = \begin{bmatrix} 1, x^{(1)} \\ 1, x^{(2)} \\ \dots \\ 1, x^{(N)} \end{bmatrix} \xrightarrow{\text{blue arrow}} \mathbb{R}^{N \times 2}$$

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} \xrightarrow{\text{blue arrow}} \mathbb{R}^{2 \times 1}$$

- Then:

$$\begin{aligned} \ell(\mathbf{w}) &= \sum_{n=1}^N \left[\mathbf{w}^T \mathbf{x}^{(n)} - t^{(n)} \right]^2 \\ &= \underbrace{(\mathbf{X}\mathbf{w} - \mathbf{t})^T}_{\mathbb{R}^{1 \times N}} \underbrace{(\mathbf{X}\mathbf{w} - \mathbf{t})}_{\mathbb{R}^{N \times 1}} \end{aligned}$$

Analytical Solution

- Instead of using GD, solve for optimal \mathbf{w} analytically

- Notice the solution is when $\frac{\partial}{\partial \mathbf{w}} \ell(\mathbf{w}) = 0$

- Derivation:

$$\begin{aligned}\ell(\mathbf{w}) &= (\mathbf{X}\mathbf{w} - \mathbf{t})^T (\mathbf{X}\mathbf{w} - \mathbf{t}) \\ &= \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - \mathbf{t}^T \mathbf{X} \mathbf{w} - \mathbf{w}^T \mathbf{X}^T \mathbf{t} + \mathbf{t}^T \mathbf{t} \\ &= \mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{t} + \mathbf{t}^T \mathbf{t}\end{aligned}$$

- Take derivative and set equal to 0, then solve for

$$\frac{\partial}{\partial \mathbf{w}} (\mathbf{w}^T \mathbf{X}^T \mathbf{X} \mathbf{w} - 2\mathbf{w}^T \mathbf{X}^T \mathbf{t} + \cancel{\mathbf{t}^T \mathbf{t}}) = 0$$

$$(\mathbf{X}^T \mathbf{X}) \mathbf{w} - \mathbf{X}^T \mathbf{t} = 0$$

$$(\mathbf{X}^T \mathbf{X}) \mathbf{w} = \mathbf{X}^T \mathbf{t}$$

If $\mathbf{X}^T \mathbf{X}$ is not invertible (i.e., singular), may need to:

- Use pseudo-inverse instead of the inverse
 - In Python,
`numpy.linalg.pinv(a)`
- Remove redundant (not linearly independent) features
- Remove extra features to ensure that $d \leq N$

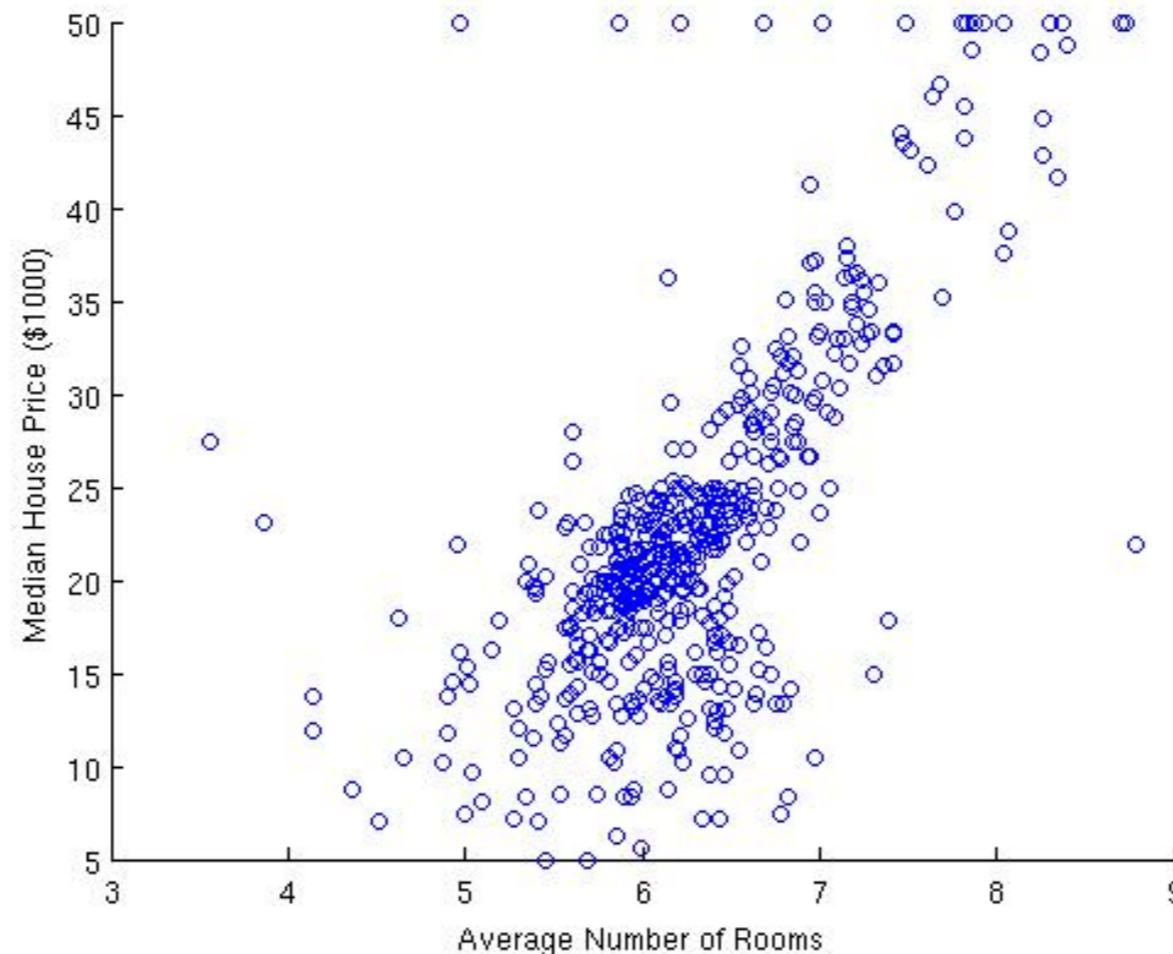
Closed Form Solution: $\mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$

Multi-dimensional Inputs

- One method of extending the model is to consider other input dimensions

$$y(\mathbf{x}) = w_0 + w_1x_1 + w_2x_2$$

- In the Boston housing example, we can look at the number of rooms



Linear Regression with Multi-dimensional Inputs

- Imagine now we want to predict the median house price from these multi-dimensional observations
- Each house is a data point n , with observations indexed by j :

$$\mathbf{x}^{(n)} = \left(x_1^{(n)}, \dots, x_j^{(n)}, \dots, x_d^{(n)} \right)$$

- We can incorporate the bias w_0 into \mathbf{w} , by using $x_0 = 1$, then

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

- We can then solve for $\mathbf{w} = (w_0, w_1, \dots, w_d)$. How?
- We can use gradient descent to solve for each coefficient, or compute \mathbf{w} analytically (how does the solution change?)

$$\text{recall: } \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{t}$$

More Powerful Models?

- What if our linear model is not good? How can we create a more complicated model?

Fitting a Polynomial

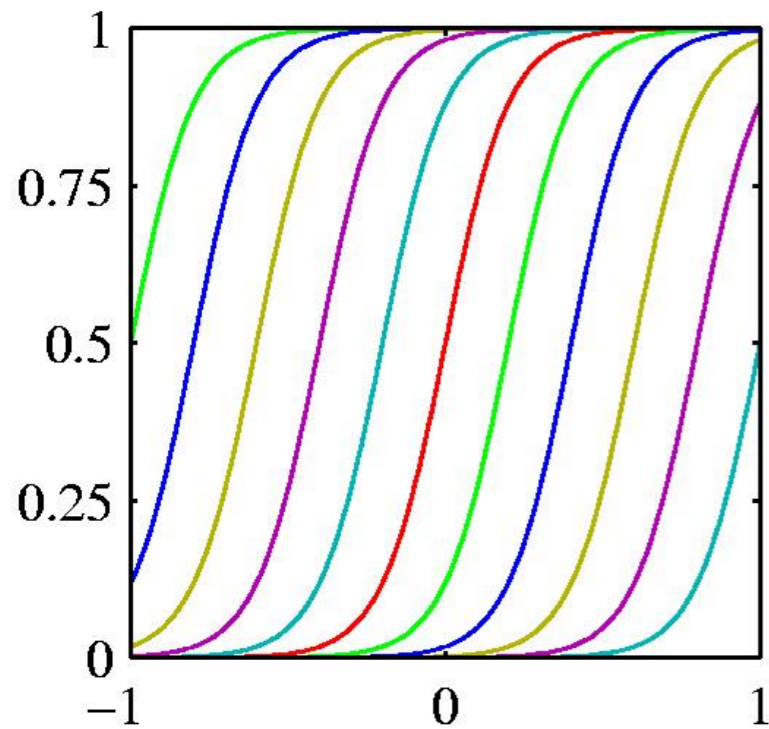
- What if our linear model is not good? How can we create a more complicated model?
- We can create a more complicated model by defining input variables that are combinations of components of \mathbf{x}
- Example: an M -th order polynomial function of one dimensional feature x :

$$y(x, \mathbf{w}) = w_0 + \sum_{j=1}^M w_j x^j$$

where x_j is the j -th power of x

- We can use the same approach to optimize for the weights \mathbf{w}
- How do we do that?

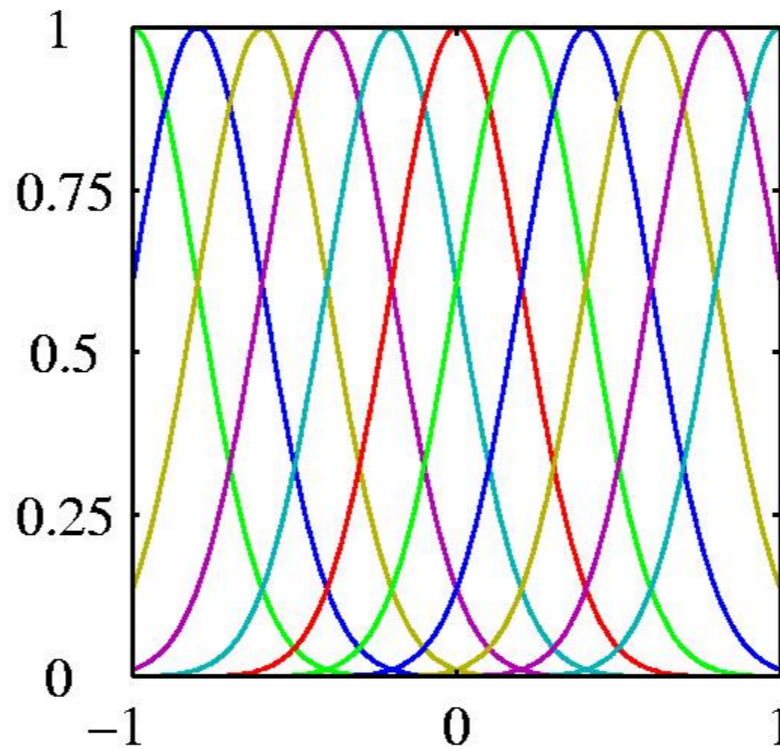
Some types of basis functions in 1-D



Sigmoids

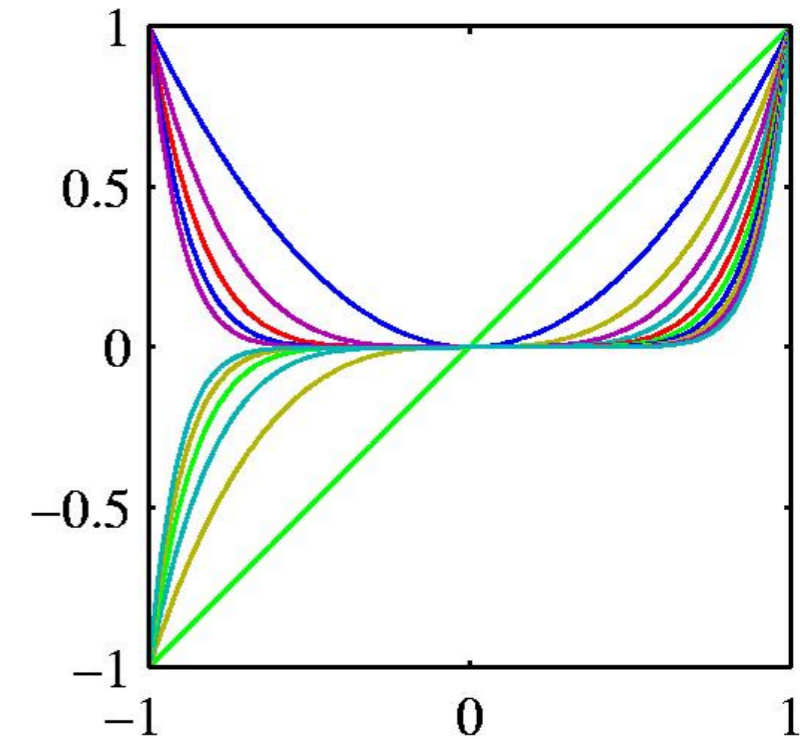
$$\phi_j(x) = \sigma\left(\frac{x - \mu_j}{s}\right)$$

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



Gaussians

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



Polynomials

Two types of linear model that are equivalent with respect to learning

$$y(\mathbf{x}, \mathbf{w}) = \overset{\text{bias}}{\downarrow} w_0 + w_1 x_1 + w_2 x_2 + \dots = \mathbf{w}^T \mathbf{x}$$

$$y(\mathbf{x}, \mathbf{w}) = w_0 + w_1 \phi_1(\mathbf{x}) + w_2 \phi_2(\mathbf{x}) + \dots = \mathbf{w}^T \Phi(\mathbf{x})$$

- The first model has the same number of adaptive coefficients as the dimensionality of the data +1.
- The second model has the same number of adaptive coefficients as the number of basis functions +1.
- Once we have replaced the data by the outputs of the basis functions, fitting the second model is exactly the same problem as fitting the first model (unless we use the [kernel trick](#))

General linear regression problem

- Using our new notations for the basis function linear regression can be written as

$$y = \sum_{j=0}^n w_j \phi_j(x)$$

where $\phi_j(x)$ can be either x_j for multivariate regression or one of the nonlinear basis we defined

- Once again we can use “least squares” to find the optimal solution.

LMS for the general linear regression problem

Our goal is to minimize the following loss function:

$$J(\mathbf{w}) = \sum_i (y^i - \sum_j w_j \phi_j(x^i))^2$$

$$y = \sum_{j=0}^n w_j \phi_j(x)$$

w – vector of dimension $k+1$
 $\phi(x^i)$ – vector of dimension $k+1$
 y^i – a scalar

Moving to vector notations we get:

$$J(\mathbf{w}) = \sum_i (y^i - \mathbf{w}^T \phi(x^i))^2$$

We take the derivative w.r.t \mathbf{w}

$$\frac{\partial}{\partial \mathbf{w}} \sum_i (y^i - \mathbf{w}^T \phi(x^i))^2 = 2 \sum_i (y^i - \mathbf{w}^T \phi(x^i)) \phi(x^i)^T$$

Equating to 0 we get $2 \sum_i (y^i - \mathbf{w}^T \phi(x^i)) \phi(x^i)^T = 0 \Rightarrow$

$$\sum_i y^i \phi(x^i)^T = \mathbf{w}^T \left[\sum_i \phi(x^i) \phi(x^i)^T \right]$$

LMS for the general linear regression problem

$$J(\mathbf{w}) = \sum_i (y^i - \mathbf{w}^T \phi(x^i))^2$$

We take the derivative w.r.t \mathbf{w}

$$\frac{\partial}{\partial \mathbf{w}} \sum_i (y^i - \mathbf{w}^T \phi(x^i))^2 = 2 \sum_i (y^i - \mathbf{w}^T \phi(x^i)) \phi(x^i)^T$$

Equating to 0 we get $2 \sum_i (y^i - \mathbf{w}^T \phi(x^i)) \phi(x^i)^T = 0 \Rightarrow$

$$\sum_i y^i \phi(x^i)^T = \mathbf{w}^T \left[\sum_i \phi(x^i) \phi(x^i)^T \right]$$

Define:

$$\Phi = \begin{pmatrix} \phi_0(x^1) & \phi_1(x^1) & \cdots & \phi_m(x^1) \\ \phi_0(x^2) & \phi_1(x^2) & \cdots & \phi_m(x^2) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_0(x^n) & \phi_1(x^n) & \cdots & \phi_m(x^n) \end{pmatrix}$$

Then deriving \mathbf{w}
we get:

$$\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

LMS for the general linear regression problem

$$J(\mathbf{w}) = \sum_i (y^i - \mathbf{w}^T \phi(x^i))^2$$

Deriving \mathbf{w} we get:

$$\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$$

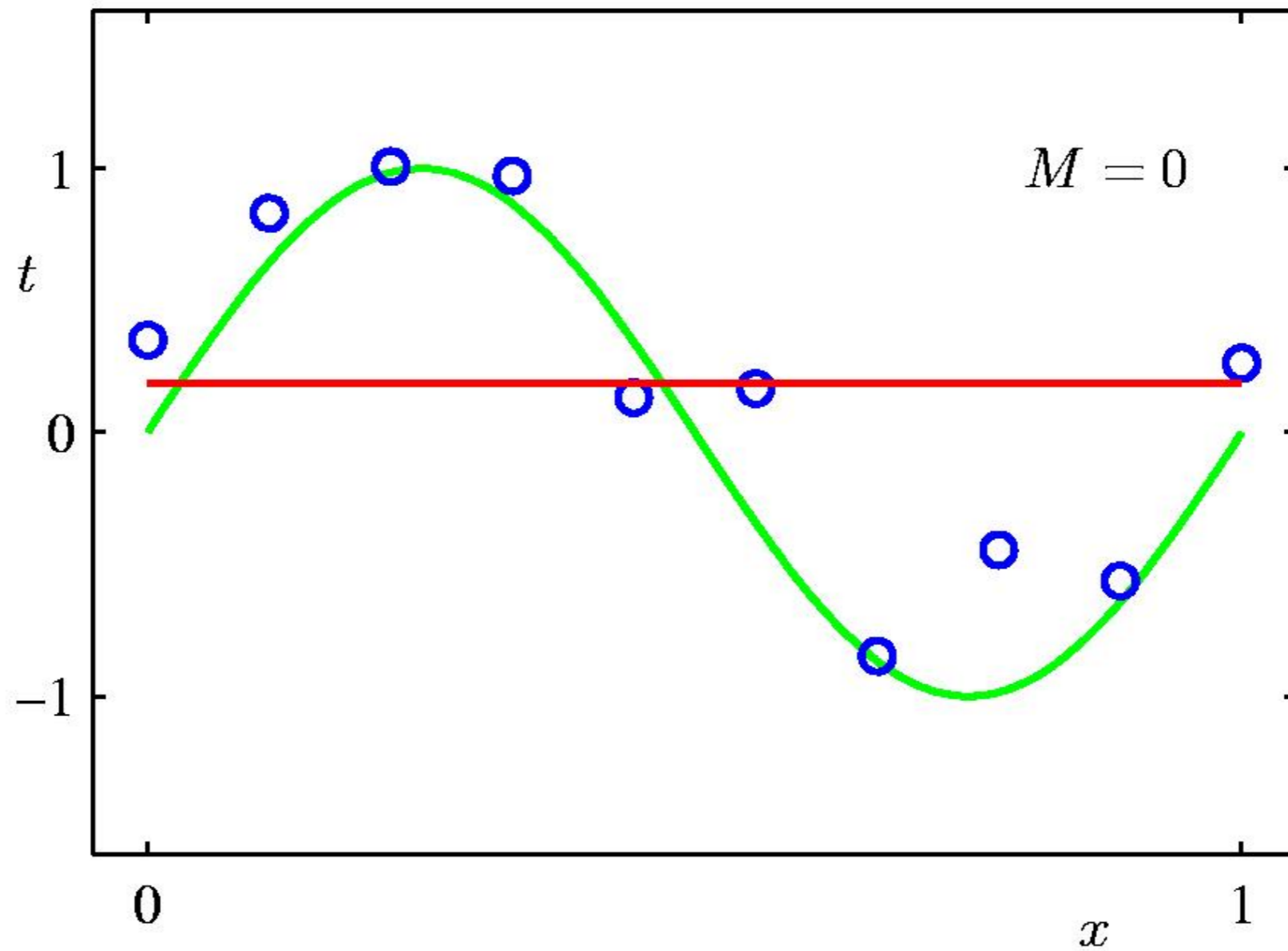
\mathbf{w} is a $k+1$ entries vector

Φ is an n by $k+1$ matrix

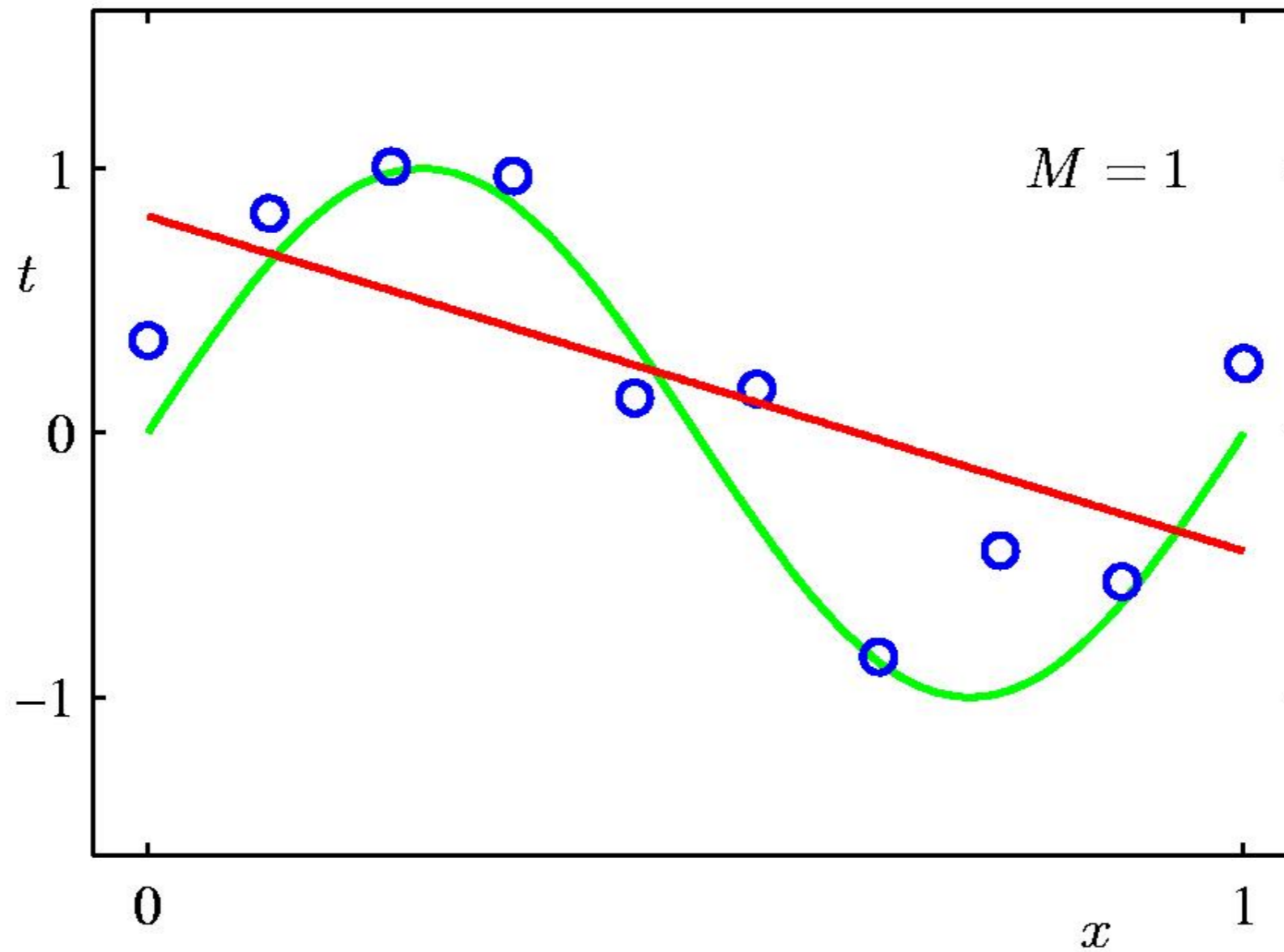
\mathbf{y} is an n entries vector

This solution is also known as 'psuedo inverse'

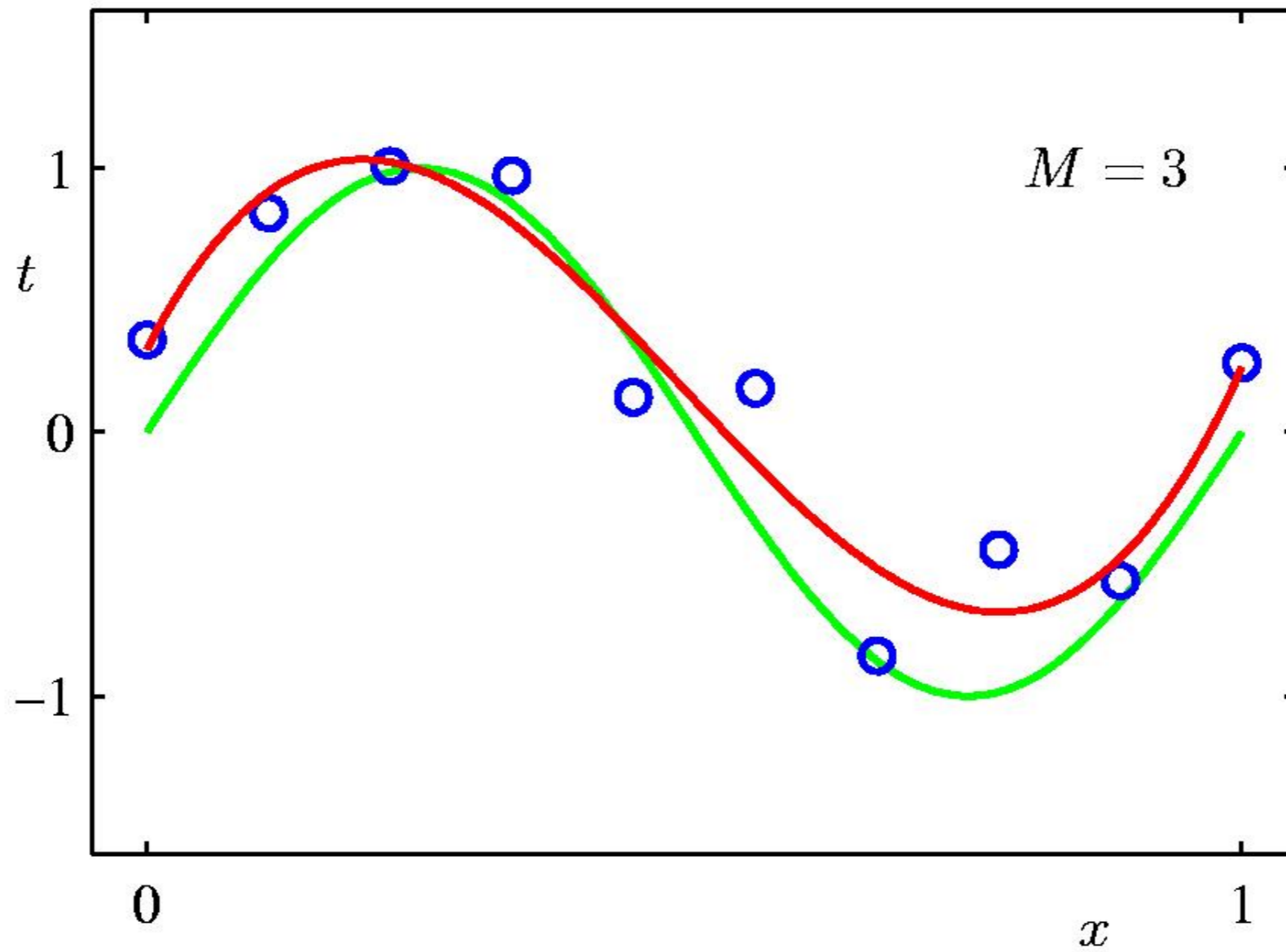
0th order polynomial



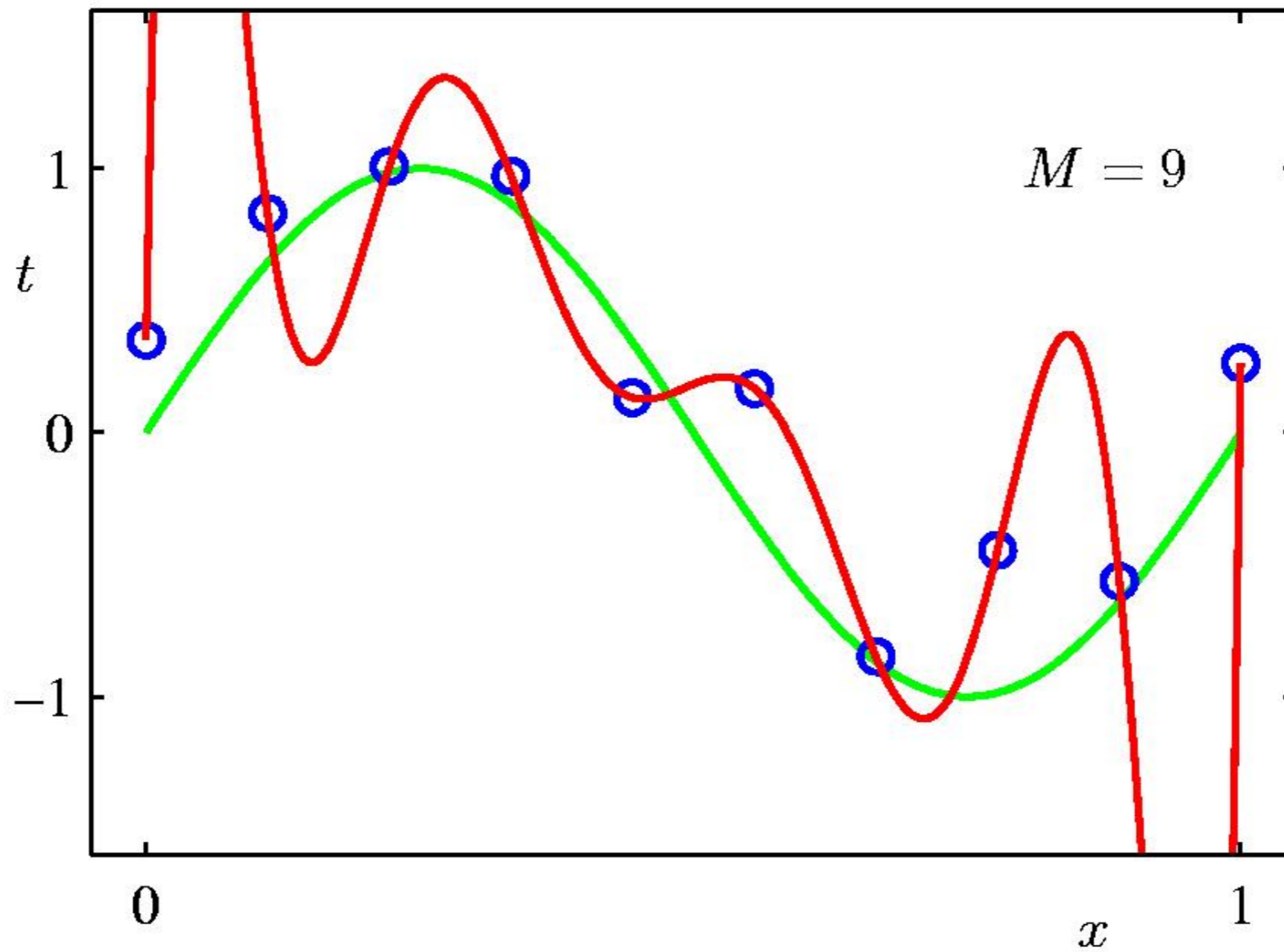
1st order polynomial



3rd order polynomial

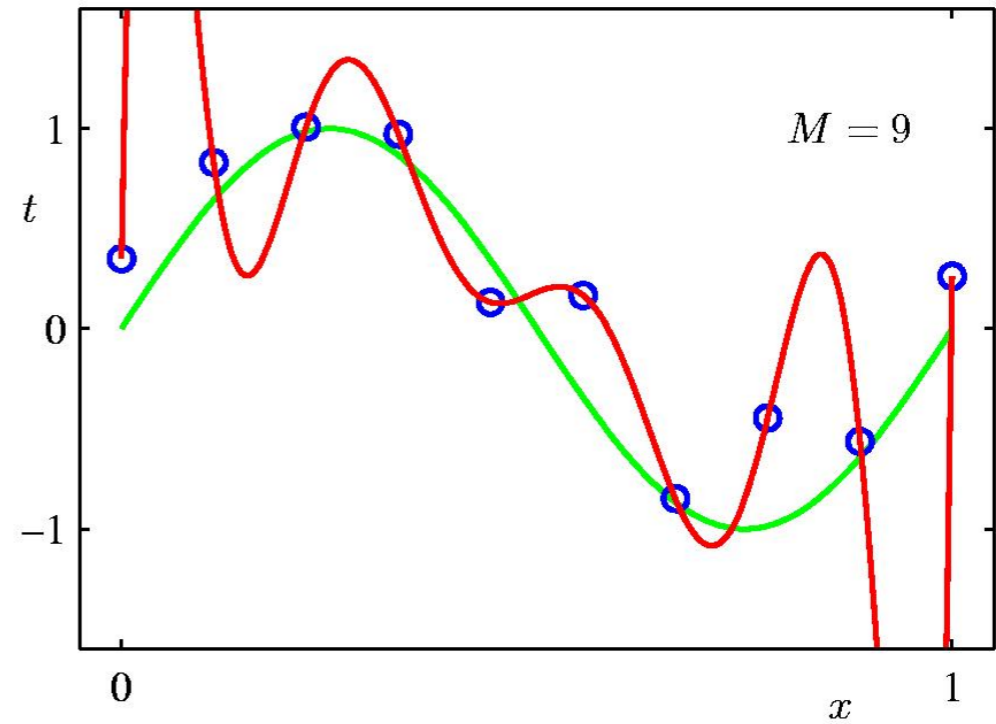
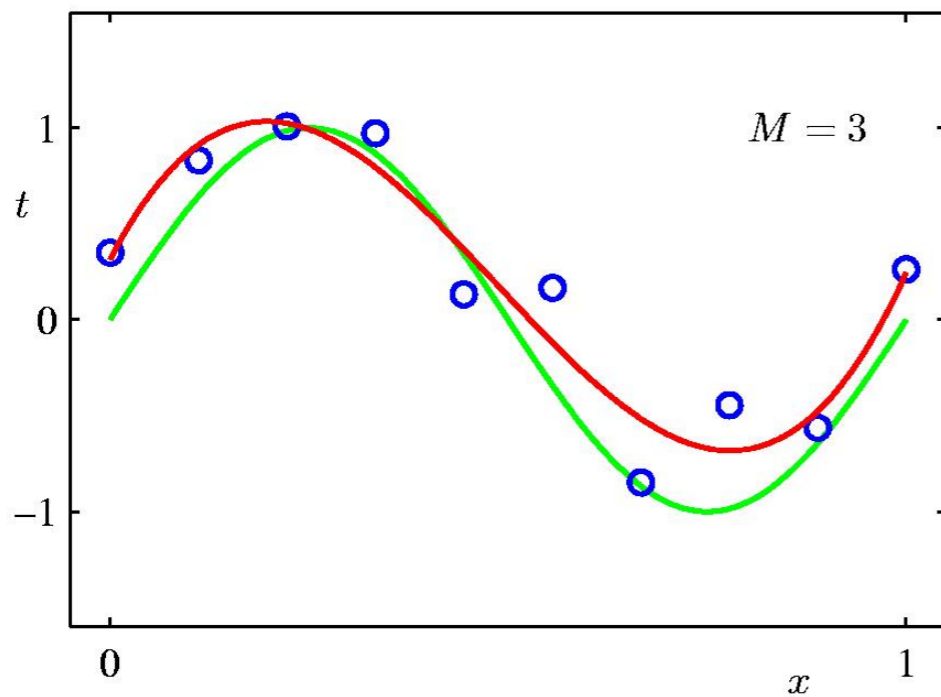
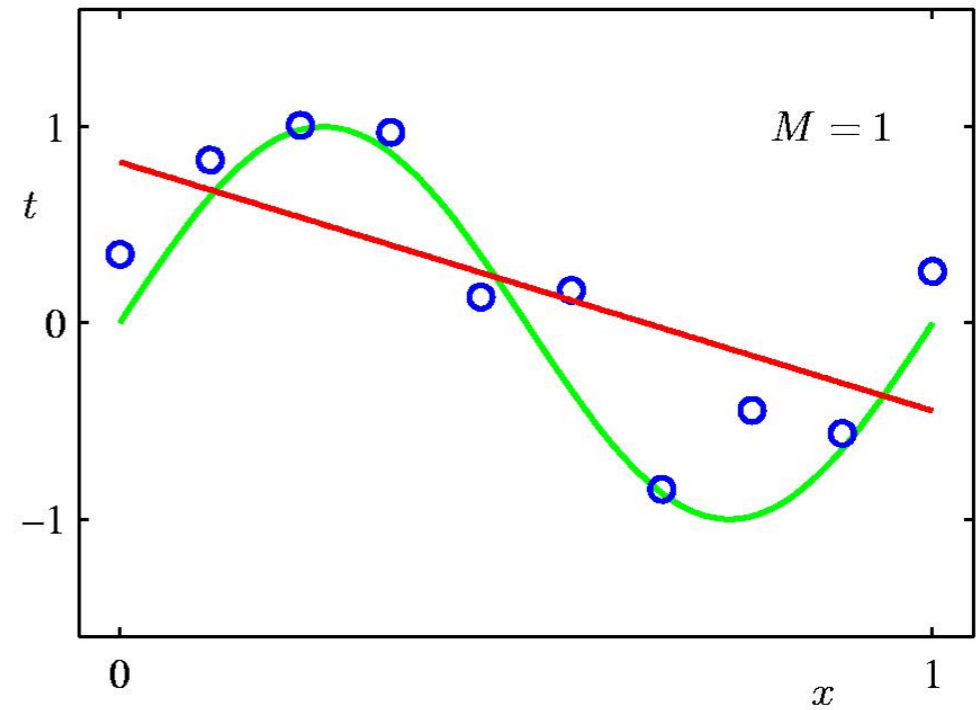
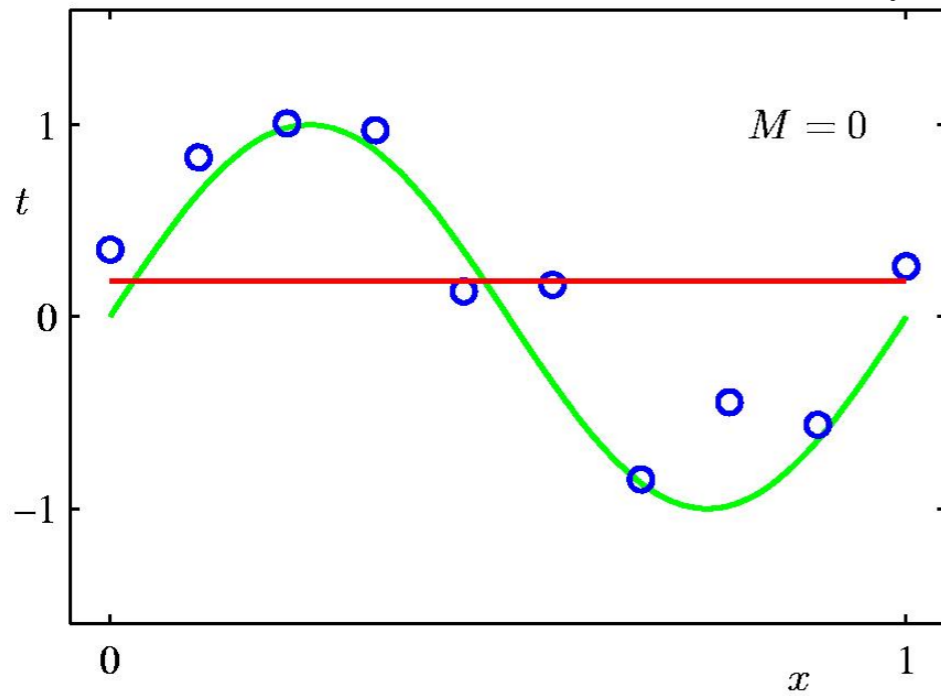


9th order polynomial



Which Fit is Best?

from Bishop

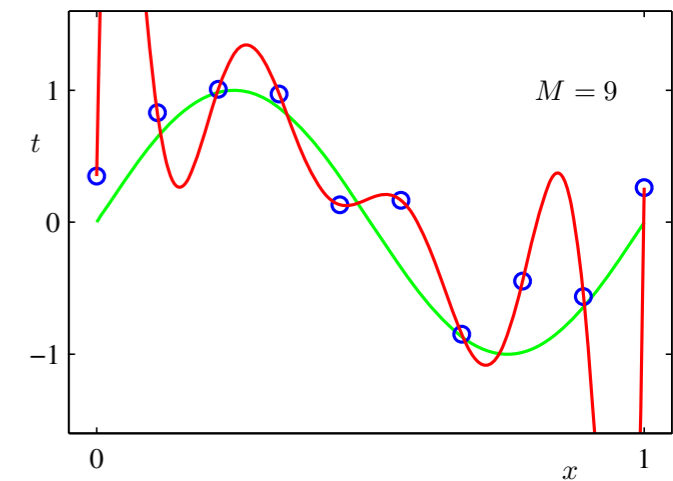
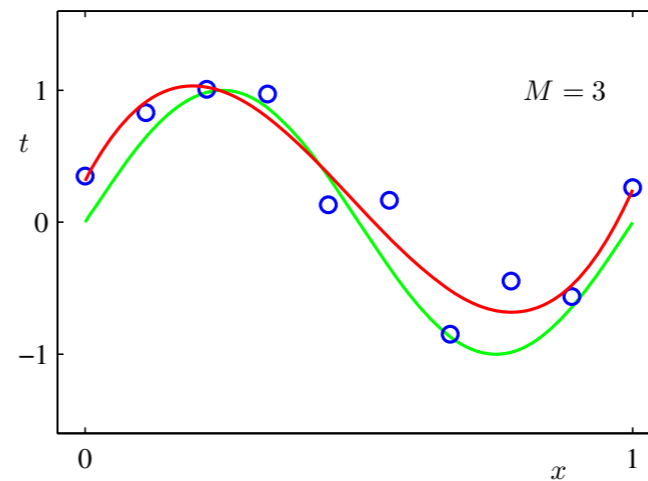
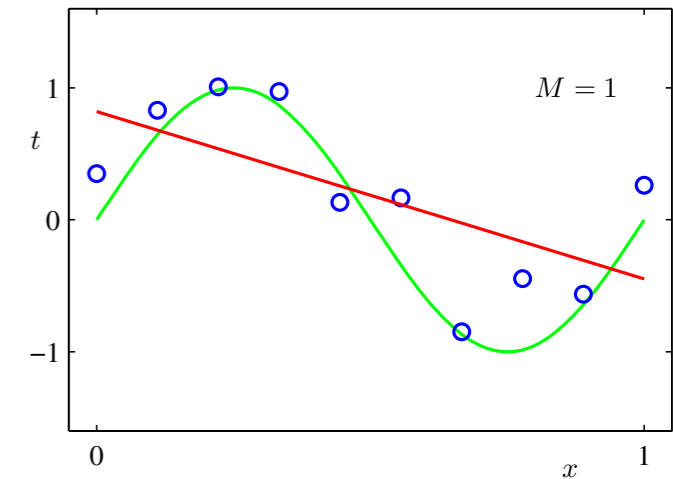
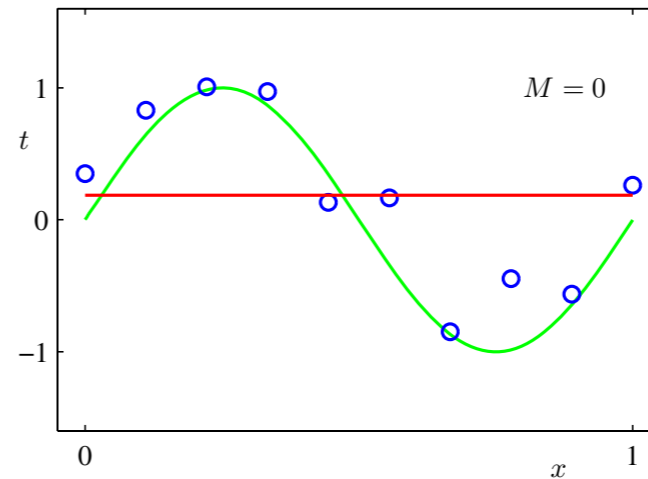


Root Mean Square (RMS) Error

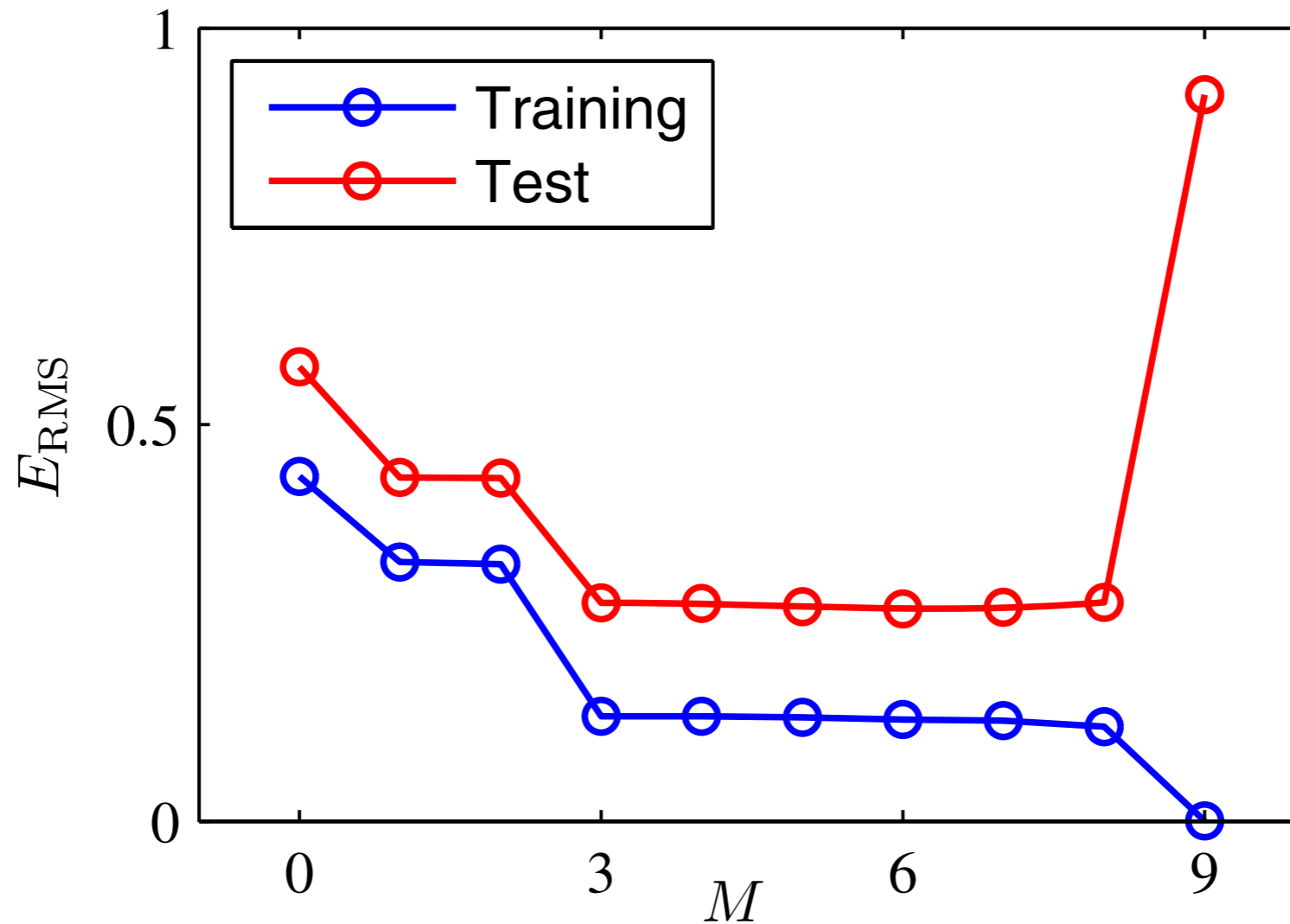
$$E(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2$$

$$E_{\text{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$$

The division by N allows us to compare different sizes of data sets on an equal footing, and the square root ensures that E_{RMS} is measured on the same scale (and in the same units) as the target variable t



Root Mean Square (RMS) Error

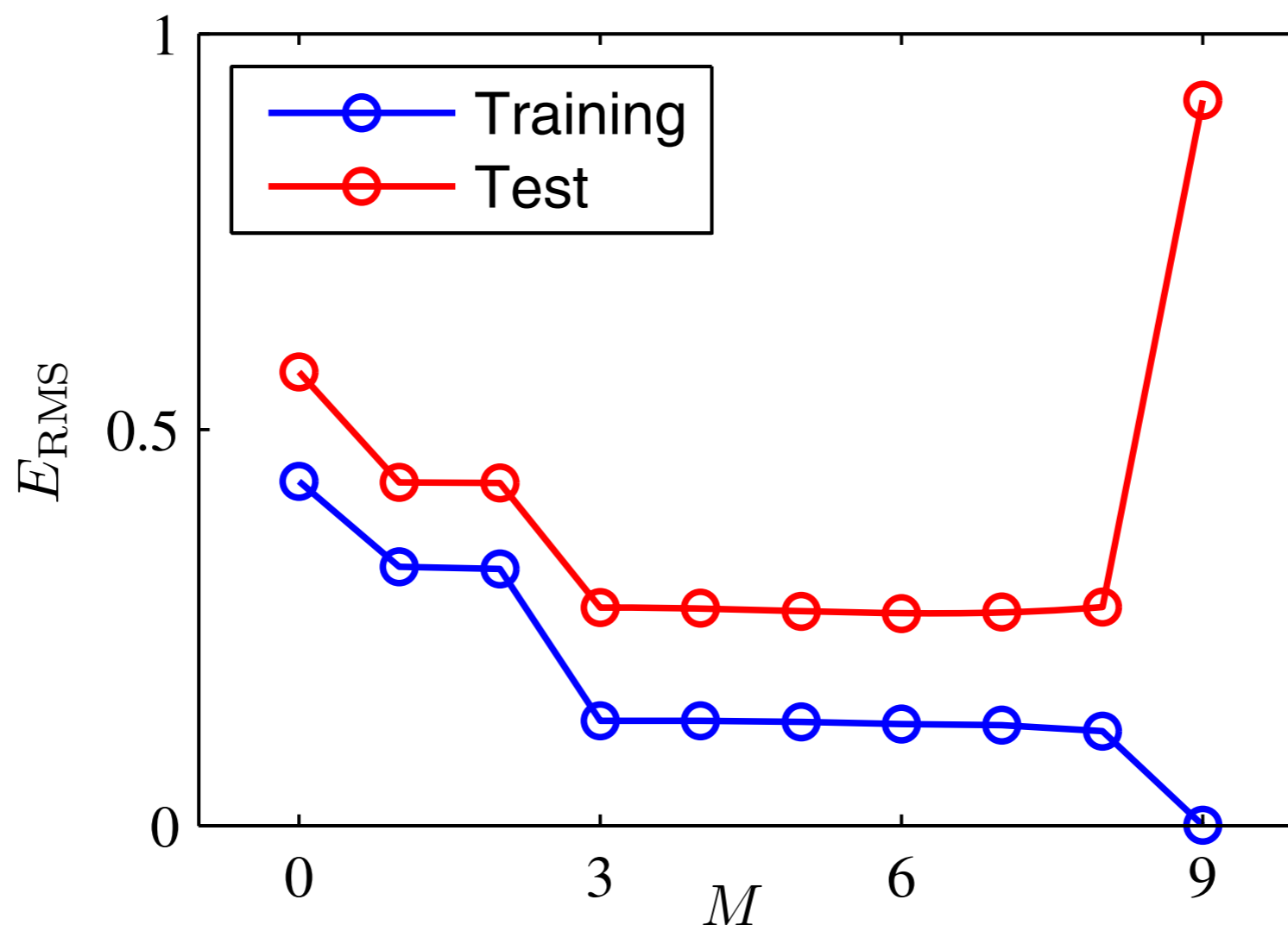


Root-Mean-Square (RMS) Error: $E_{\text{RMS}} = \sqrt{2E(\mathbf{w}^*)/N}$

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

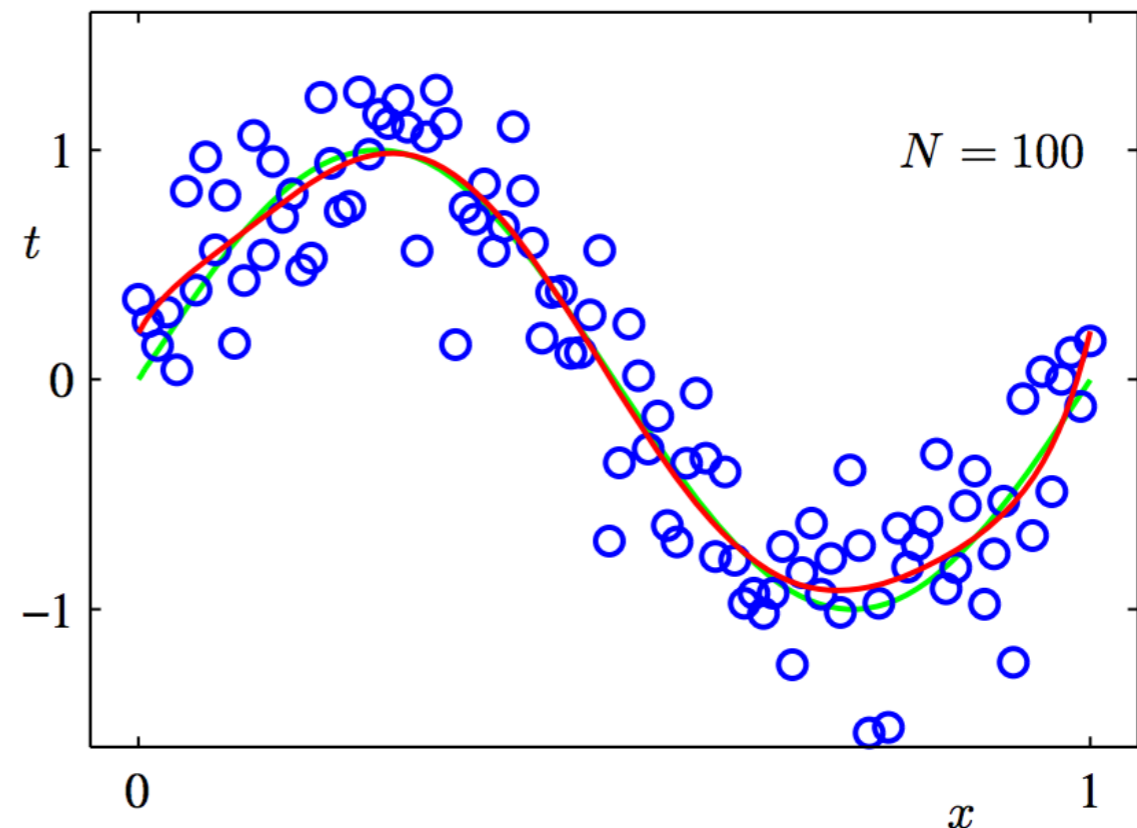
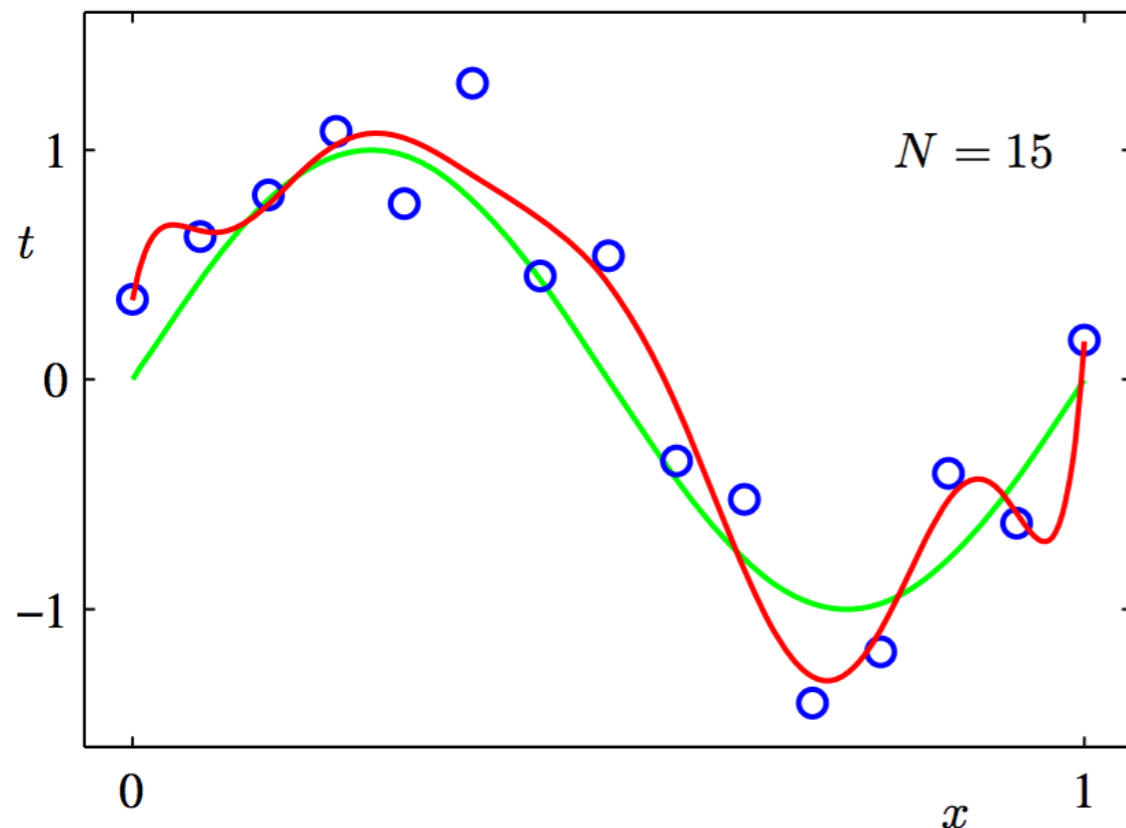
Generalization

- **Generalization** = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ **overfits** the data (it models also noise)



Generalization

- **Generalization** = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ **overfits** the data (it models also noise)
- Not a problem if we have lots of training examples



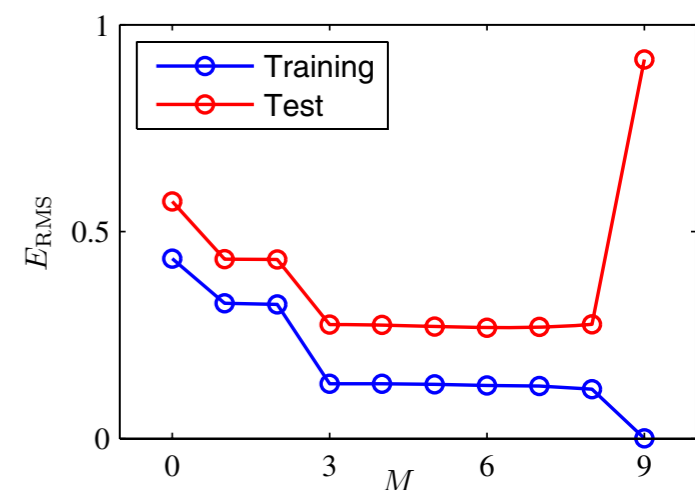
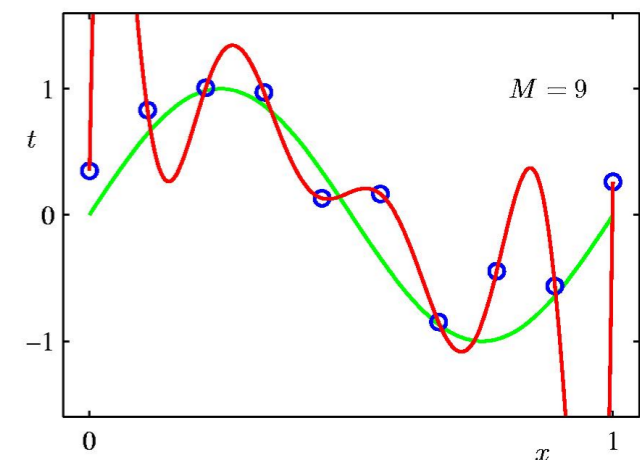
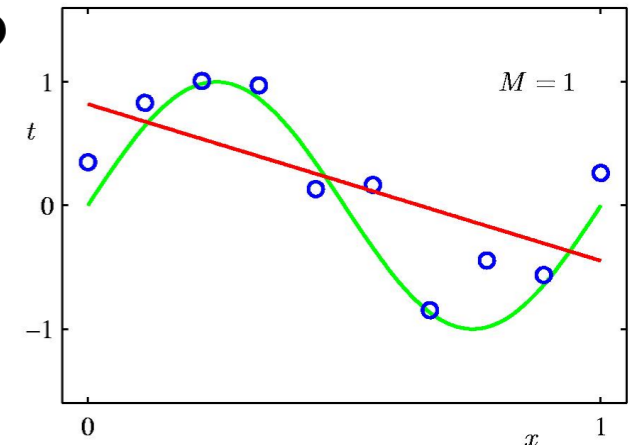
Generalization

- **Generalization** = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ **overfits** the data (it models also noise)
- Let's look at the estimated weights for various M in the case of fewer examples

	$M = 0$	$M = 1$	$M = 6$	$M = 9$
w_0^*	0.19	0.82	0.31	0.35
w_1^*		-1.27	7.99	232.37
w_2^*			-25.43	-5321.83
w_3^*			17.37	48568.31
w_4^*				-231639.30
w_5^*				640042.26
w_6^*				-1061800.52
w_7^*				1042400.18
w_8^*				-557682.99
w_9^*				125201.43

1-D regression illustrates key concepts

- Data fits – is linear model best (**model selection**)?
 - Simplest models do not capture all the important variations (signal) in the data: **underfit**
 - More complex model may **overfit** the training data (fit not only the signal but also the **noise** in the data), especially if not enough data to constrain model
- One method of assessing fit:
 - test **generalization** = model's ability to predict the held out data
- Optimization is essential: stochastic and batch iterative approaches; analytic when available



Generalization

- **Generalization** = model's ability to predict the held out data
- What is happening?
- Our model with $M = 9$ **overfits** the data (it models also noise)
- Let's look at the estimated weights for various M in the case of fewer examples
- The weights are becoming huge to compensate for the noise
- One way of dealing with this is to encourage the weights to be small (this way no input dimension will have too much influence on prediction). This is called **regularization**.

Regularized Least Squares

- A technique to control the overfitting phenomenon
- Add a penalty term to the error function in order to discourage the coefficients from reaching large values

Ridge
regression

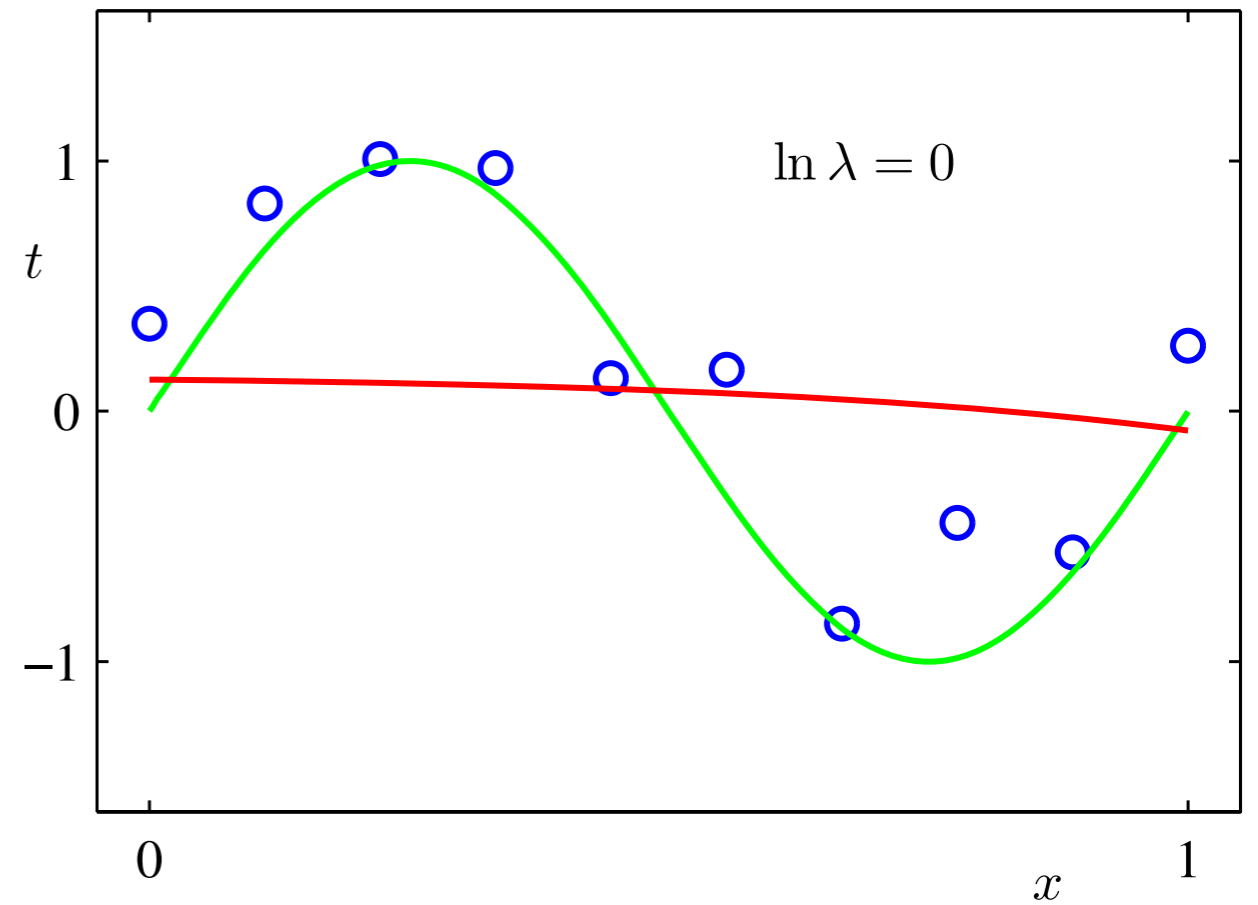
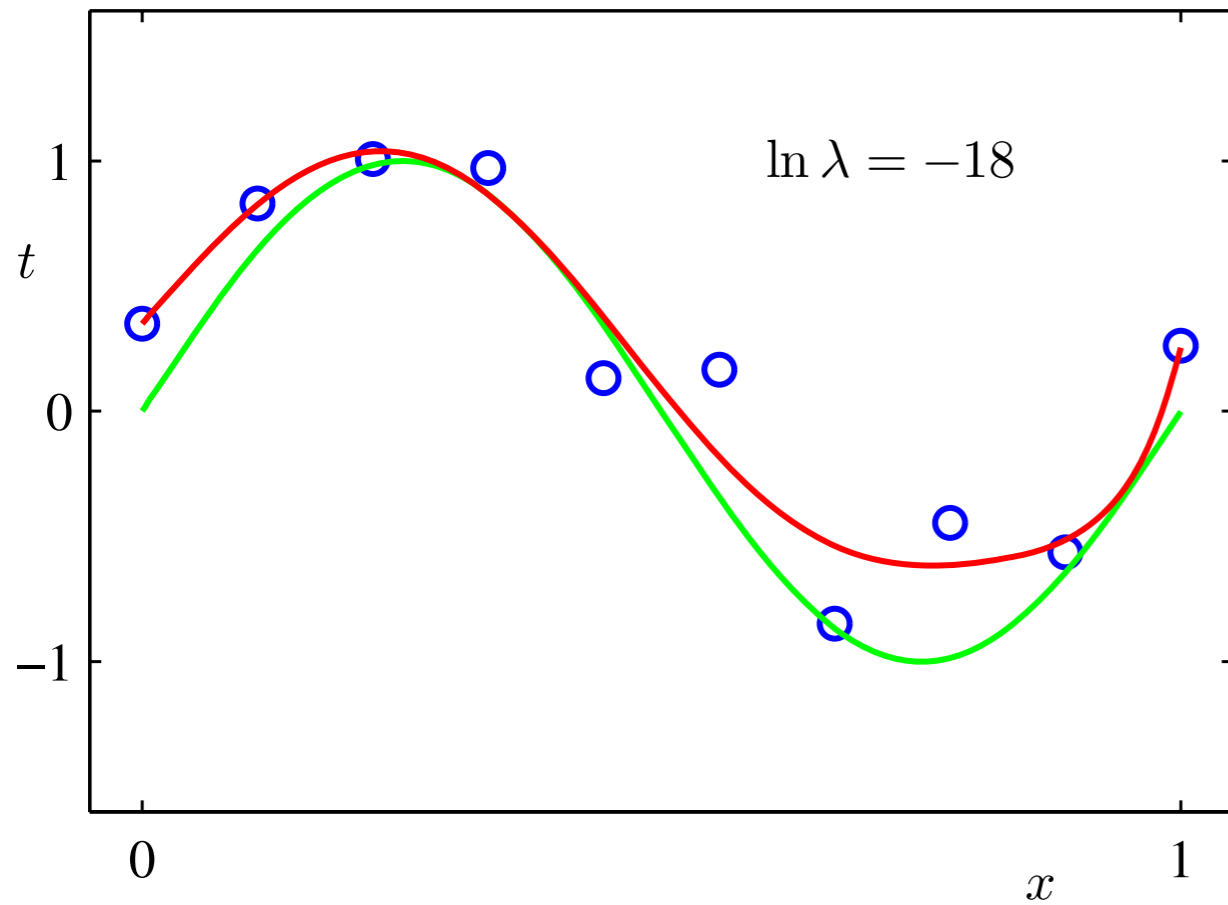
$$\tilde{E}(\mathbf{w}) = \frac{1}{2} \sum_{n=1}^N \{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

$$\|\mathbf{w}\|^2 \equiv \mathbf{w}^T \mathbf{w} = w_0^2 + w_1^2 + \dots + w_M^2$$

which is minimized by

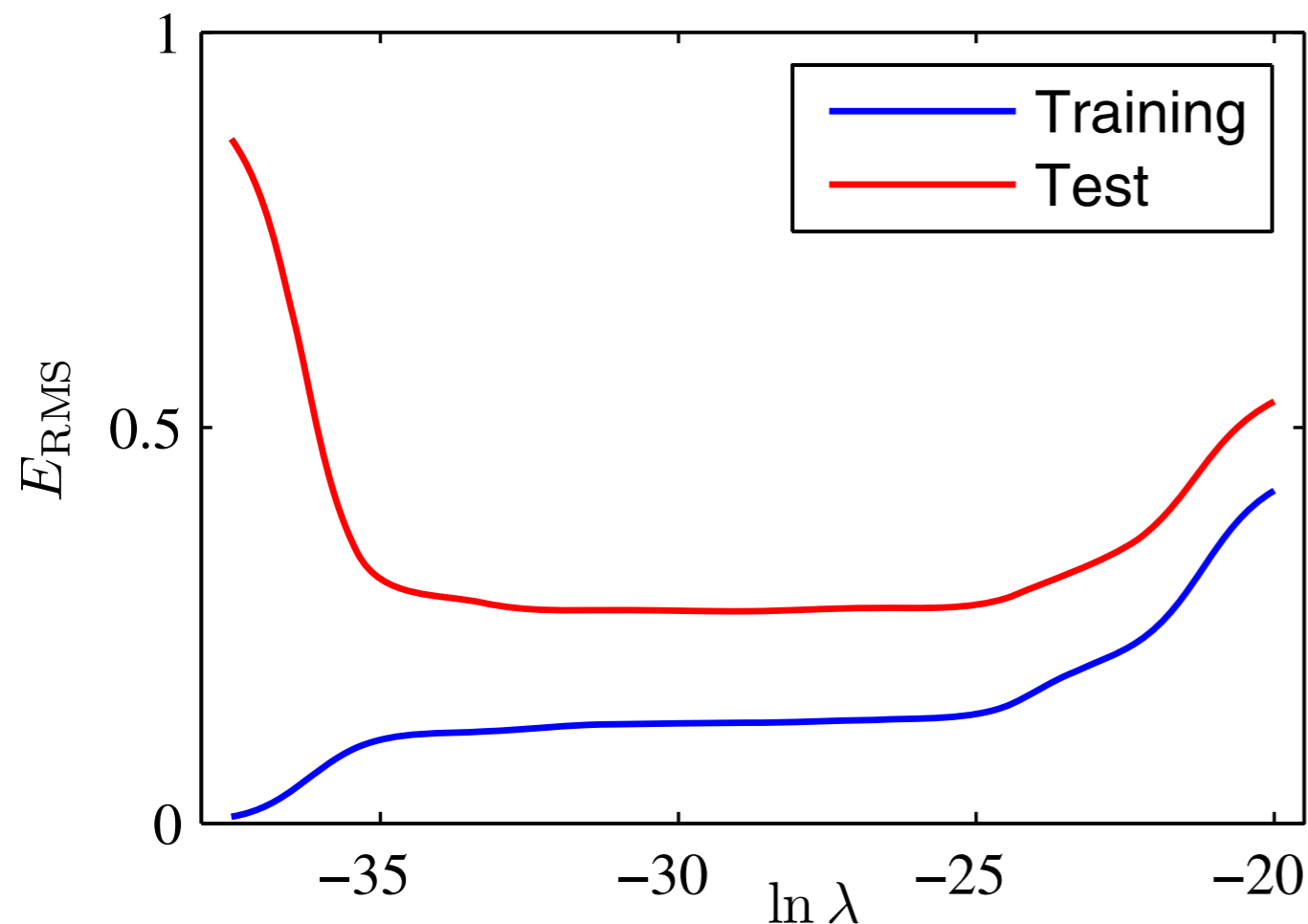
$$\mathbf{w} = \left(\lambda \mathbf{I} + \Phi^T \Phi \right)^{-1} \Phi^T \mathbf{t}.$$

The effect of regularization



$$M = 9$$

The effect of regularization

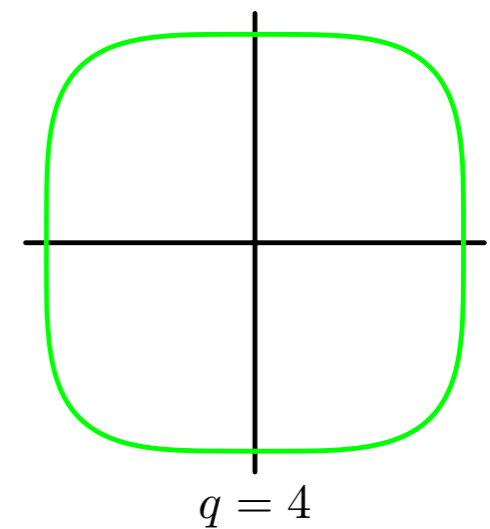
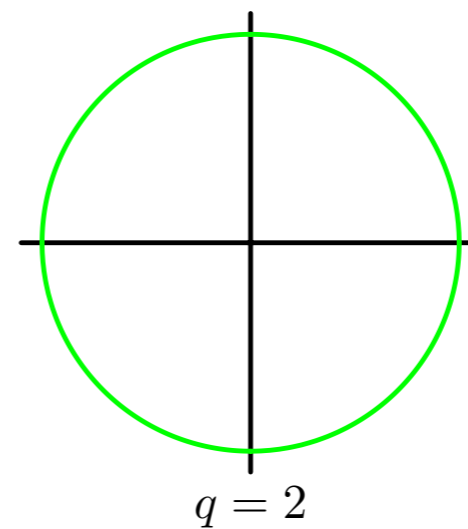
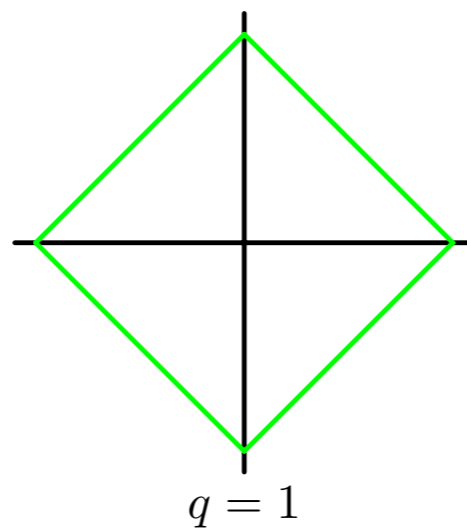
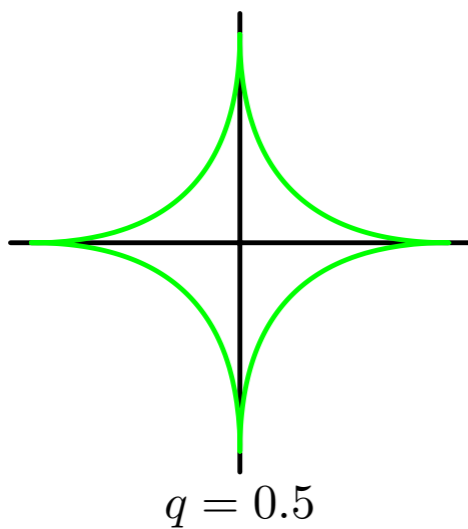


	$\ln \lambda = -\infty$	$\ln \lambda = -18$	$\ln \lambda = 0$
w_0^*	0.35	0.35	0.13
w_1^*	232.37	4.74	-0.05
w_2^*	-5321.83	-0.77	-0.06
w_3^*	48568.31	-31.97	-0.05
w_4^*	-231639.30	-3.89	-0.03
w_5^*	640042.26	55.28	-0.02
w_6^*	-1061800.52	41.32	-0.01
w_7^*	1042400.18	-45.95	-0.00
w_8^*	-557682.99	-91.53	0.00
w_9^*	125201.43	72.68	0.01

The corresponding coefficients from the fitted polynomials, showing that regularization has the desired effect of reducing the magnitude of the coefficients.

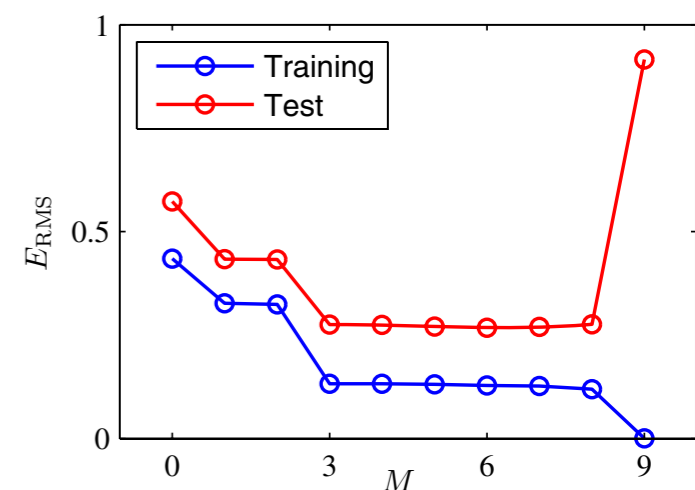
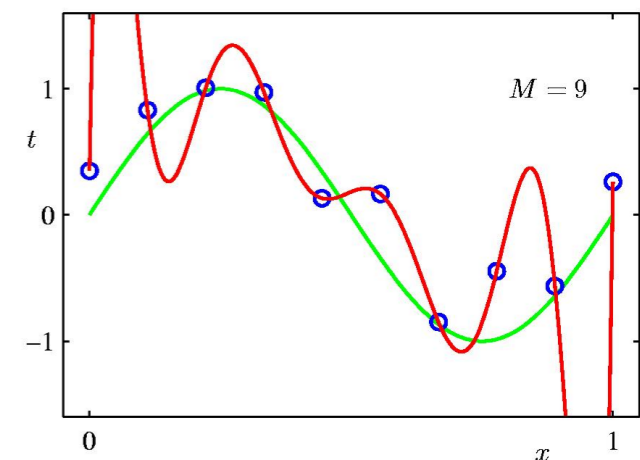
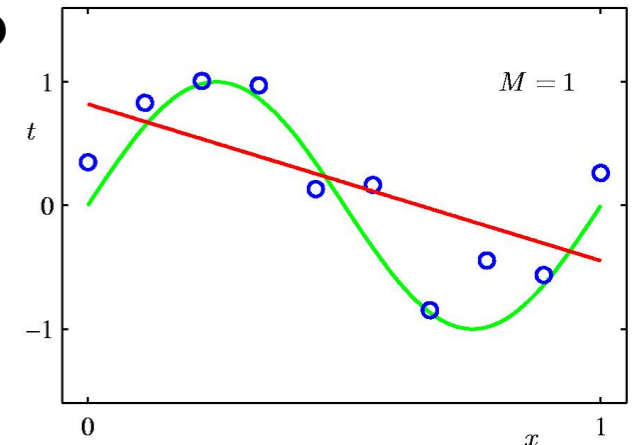
A more general regularizer

$$\frac{1}{2} \sum_{n=1}^N \{t_n - \mathbf{w}^T \phi(\mathbf{x}_n)\}^2 + \frac{\lambda}{2} \sum_{j=1}^M |w_j|^q$$



1-D regression illustrates key concepts

- Data fits – is linear model best (**model selection**)?
 - Simplest models do not capture all the important variations (signal) in the data: **underfit**
 - More complex model may **overfit** the training data (fit not only the signal but also the **noise** in the data), especially if not enough data to constrain model
- One method of assessing fit:
 - test **generalization** = model's ability to predict the held out data
- Optimization is essential: stochastic and batch iterative approaches; analytic when available



Next Lecture: Machine Learning Methodology