ITCS 6156/8156 Fall 2023
Machine Learning

Linear Regression

Instructor: Hongfei Xue
Email: hongfei.xue@charlotte.edu
Class Meeting: Mon & Wed, 4:00 PM – 5:15 PM, CHHS 376

UNIVERSITY OF NORTH CAROLINA CHARLOTTE

Some content in the slides is based on Dr. Razvan's lecture

Machine Learning as Optimization

At this point, we move away from the situation where a perfect solution exists, and the learning task it to reach the perfect solution. Instead, we focus on finding the *best possible* solution which optimizes certain criterion.

- Learning is optimization
- Faster optimization methods for faster learning
- Let $w \in \mathbb{R}^d$ and $S \subset \mathbb{R}^d$ and $f_0(w), f_1(w), \ldots, f_m(w)$ be real-valued functions.
- Standard optimization formulation is:

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & f_0(w) \\ \text{subject to} \quad & f_i(w) \leq 0, \ i = 1, \ldots, m. \end{aligned}$$

- Methods for **general optimization problems**
  - Simulated annealing, genetic algorithms
- Exploiting *structure* in the optimization problem
  - **Convexity**, Lipschitz continuity, smoothness

Convexity

Convexity is a property of certain functions which can be exploited by optimization algorithms. The idea of convexity can be understood by first considering *convex sets*. A convex set is a set of points in a coordinate space such that every point on the line segment joining any two points in the set are also within the set. Mathematically, this can be written as:

$$w_1, w_2 \in S \Rightarrow \lambda w_1 + (1 - \lambda) w_2 \in S$$

where $\lambda \in [0, 1]$. A *convex function* is defined as follows:

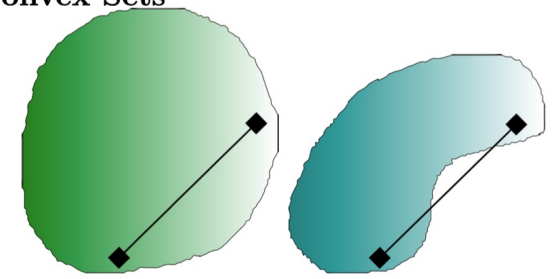
- $f : \mathbb{R}^d \to \mathbb{R}$ is a convex function if the domain of f is a convex set and for all $\lambda \in [0, 1]$:

$$f(\lambda w_1 + (1 - \lambda) w_2) \leq \lambda f(w_1) + (1 - \lambda) f(w_2)$$

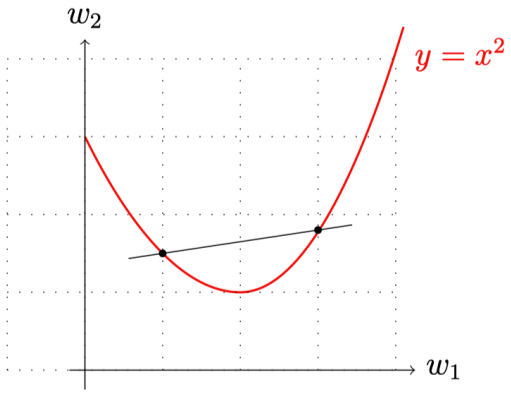
Some examples of convex functions are:

- Affine functions: $w^\top x + b$
- $||w||_p$ for $p \geq 1$
- Logistic loss: $\log(1 + e^{-y w^\top x})$

Convex Sets



Convex Functions



Convex Optimization

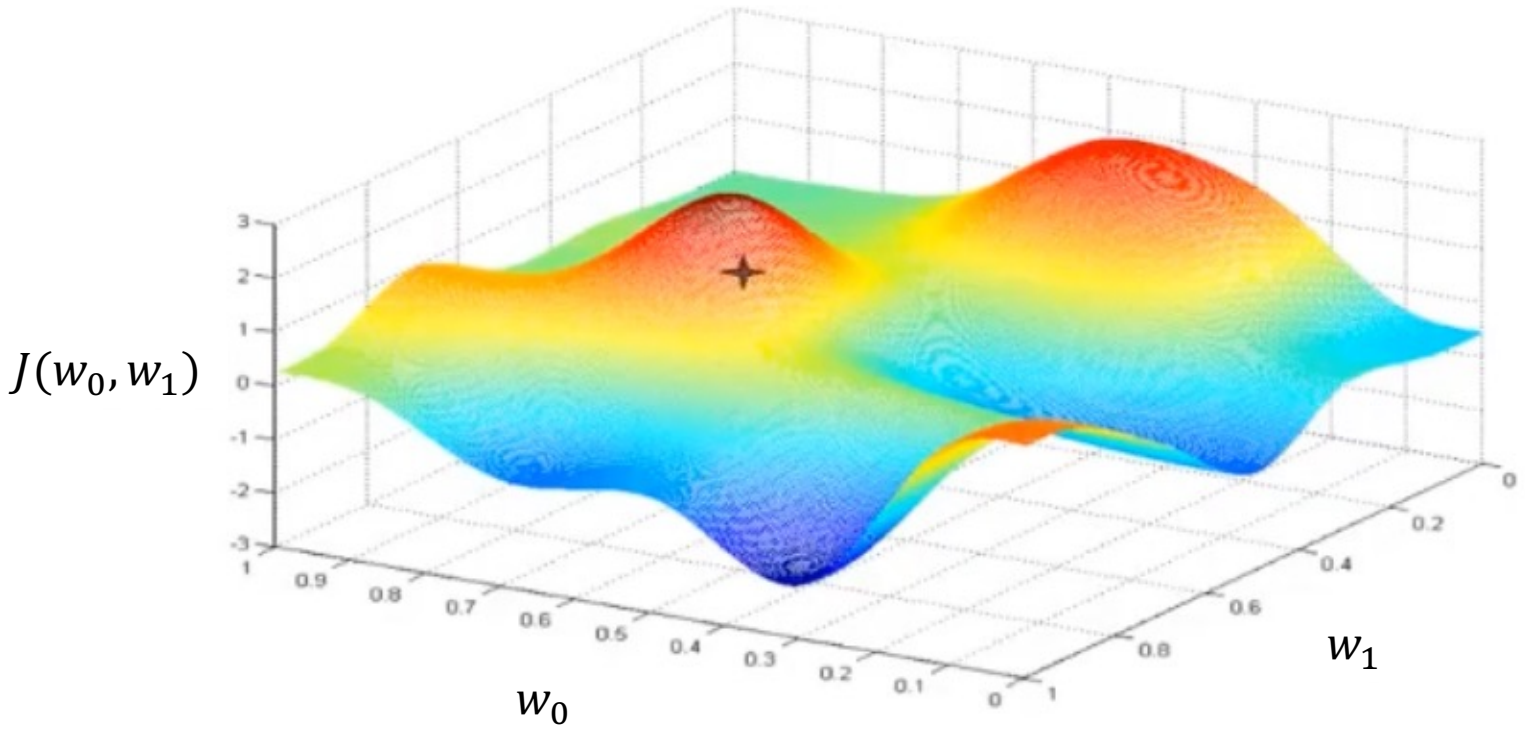
- Optimality Criterion

$$\begin{aligned} \underset{w}{\text{minimize}} \quad & f_0(w) \\ \text{subject to} \quad & f_i(w) \leq 0,\ i = 1, \ldots, m. \end{aligned}$$

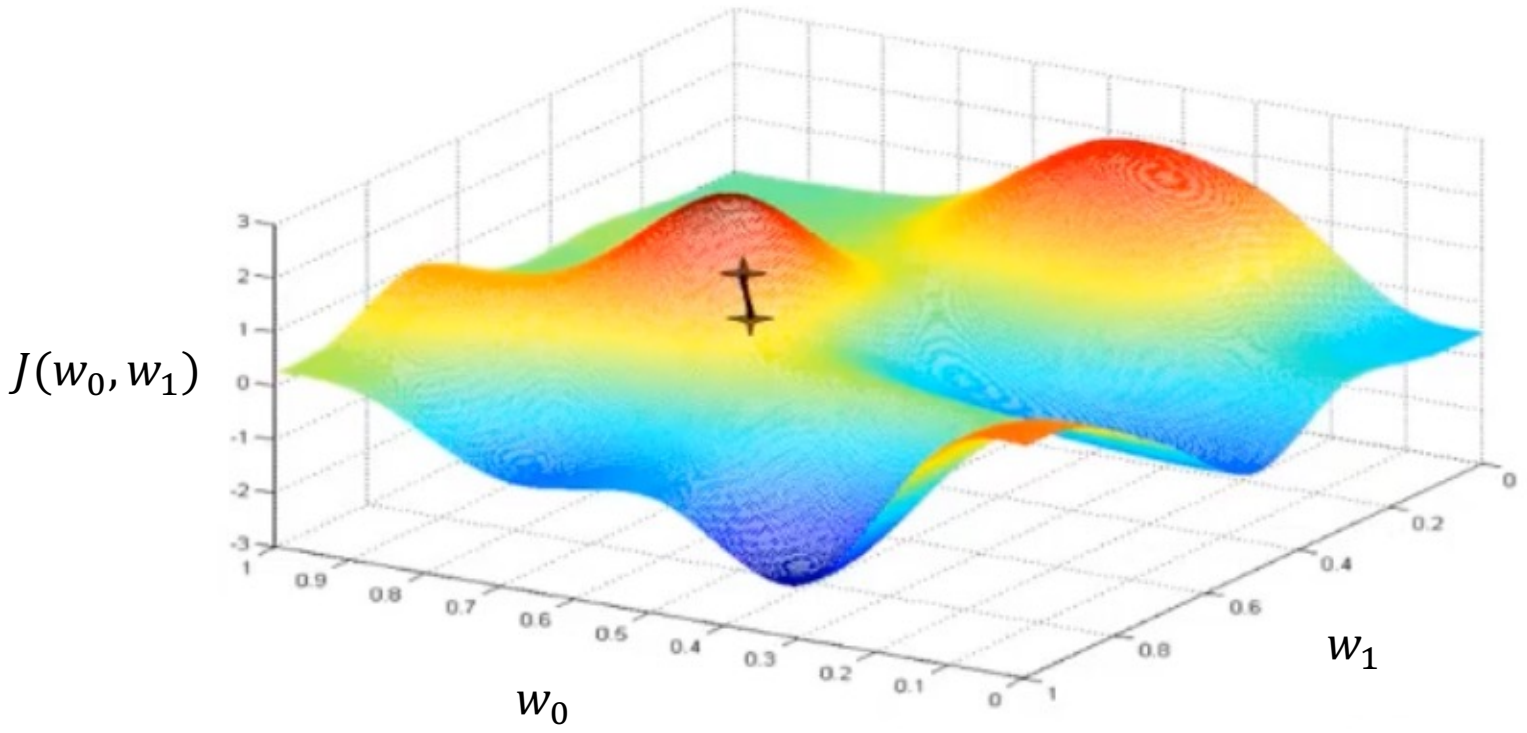
  where all $f_i(w)$ are **convex functions**.

- w_0 is feasible if $w_0 \in Dom\ f_0$ and all constraints are satisfied
- A feasible w^* is optimal if $f_0(w^*) \leq f_0(w)$ for all w satisfying the constraints

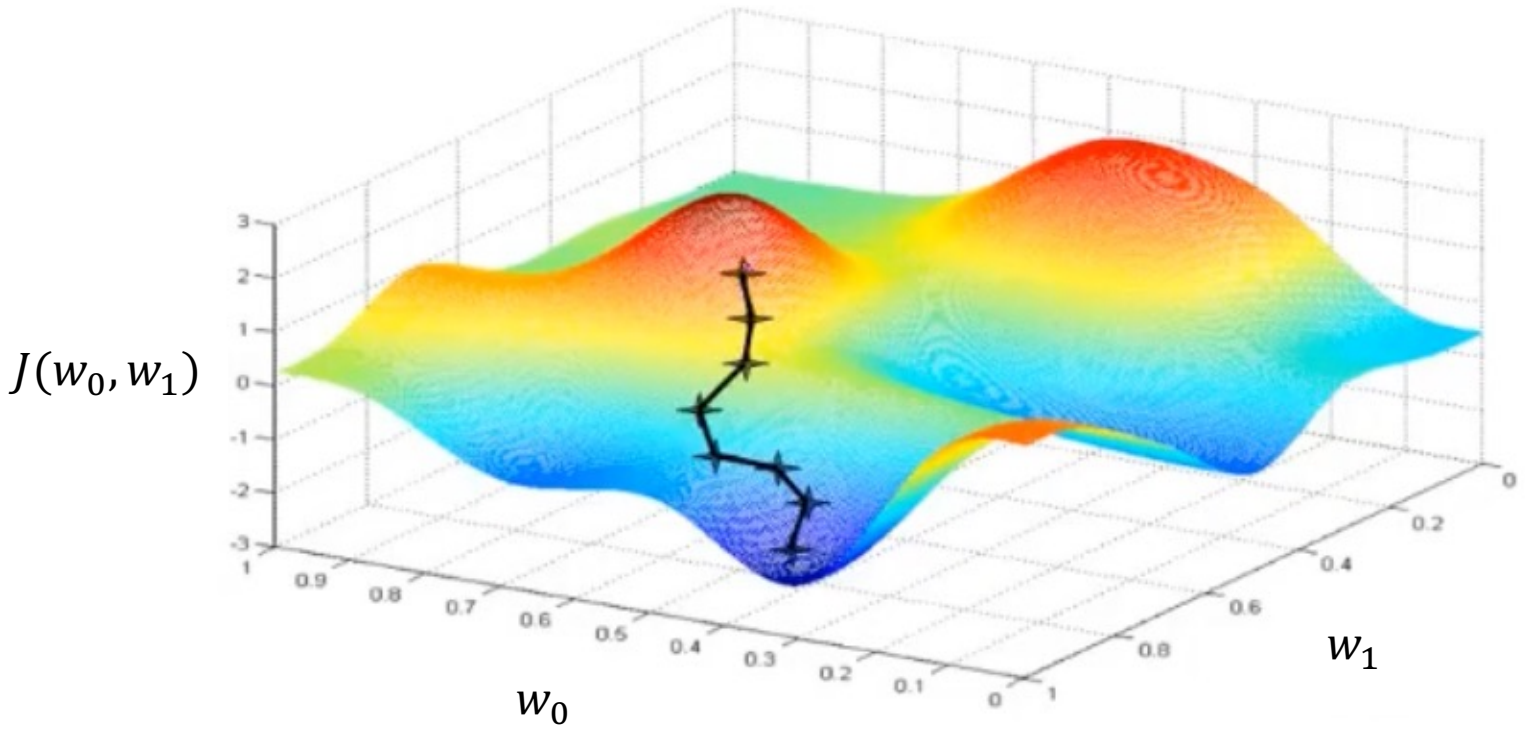
Gradient Descent



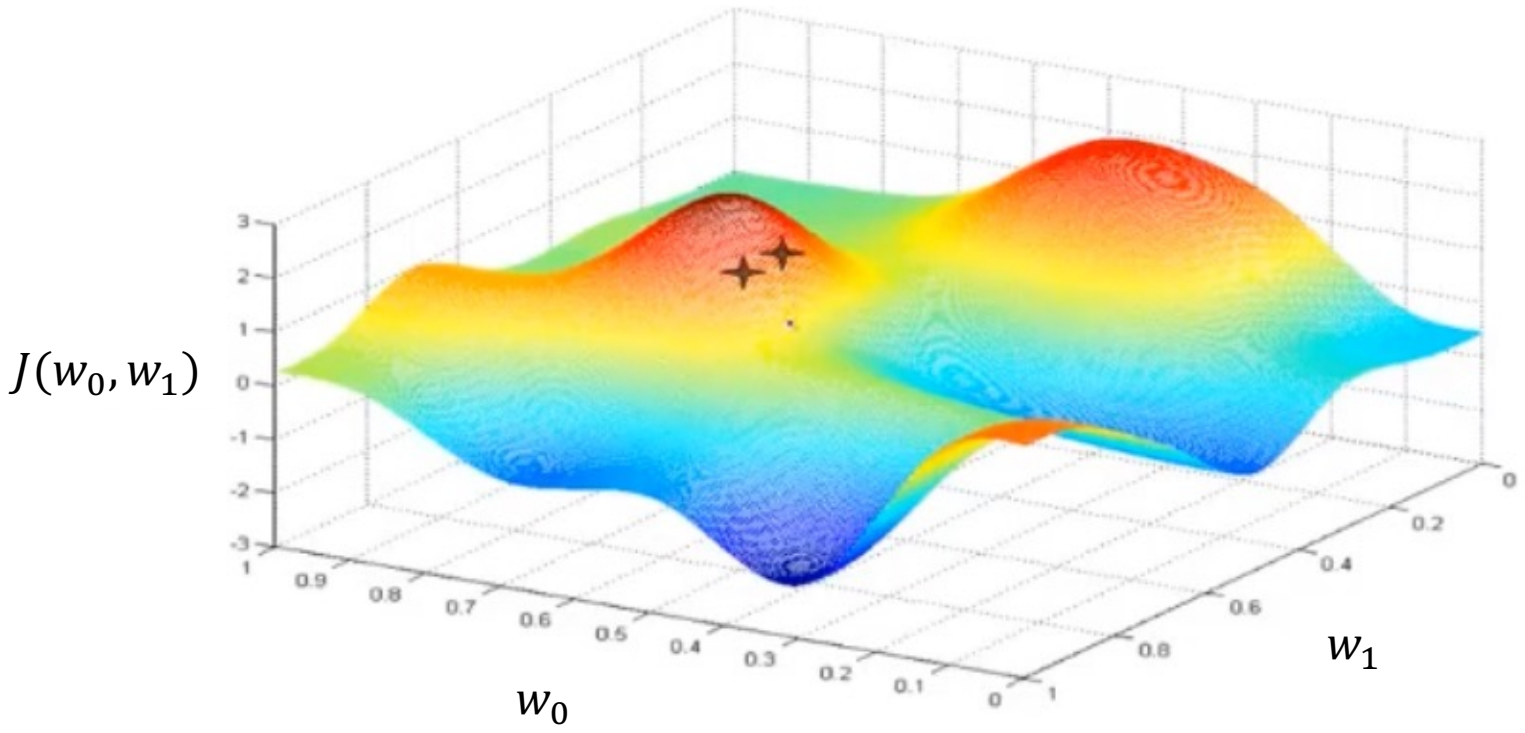
Gradient Descent



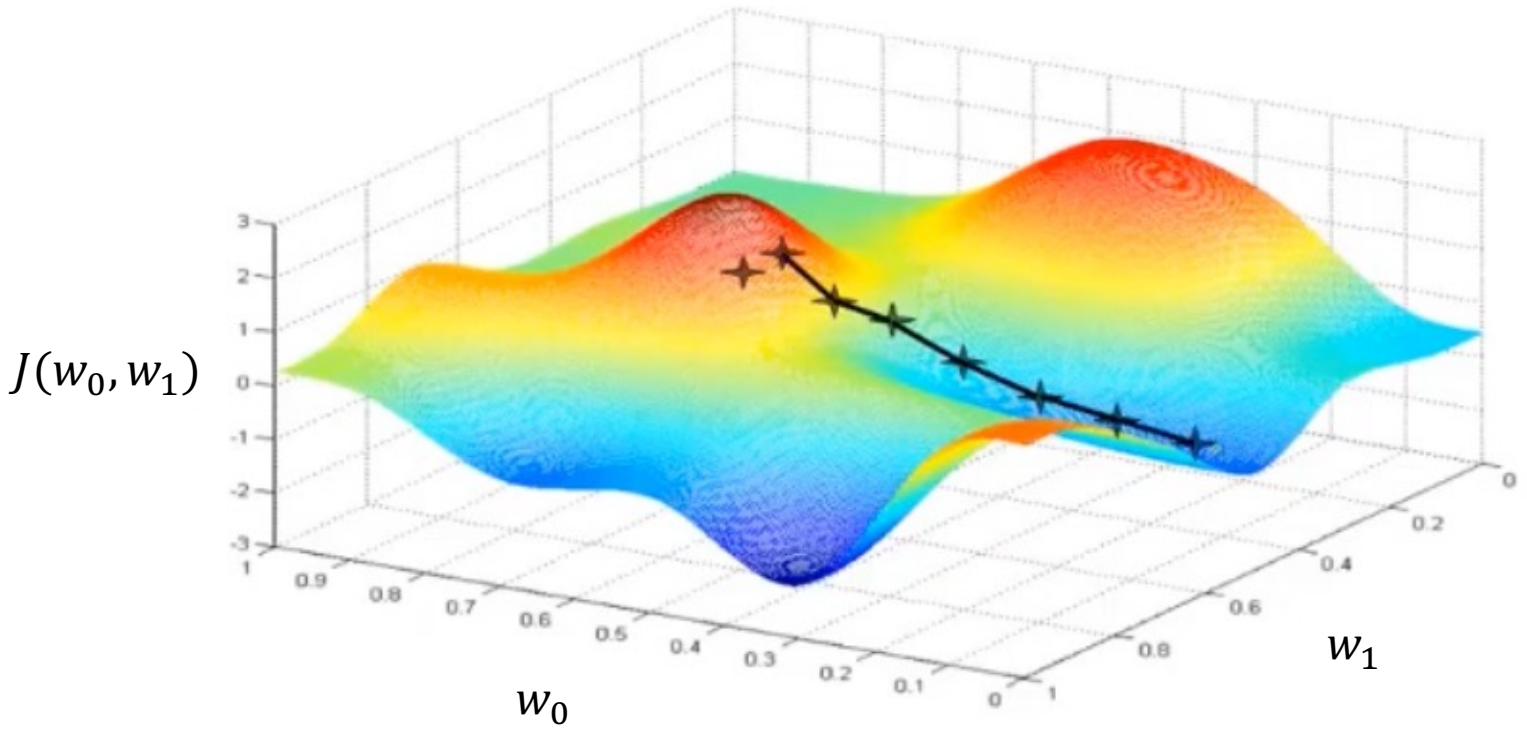
Gradient Descent



Gradient Descent



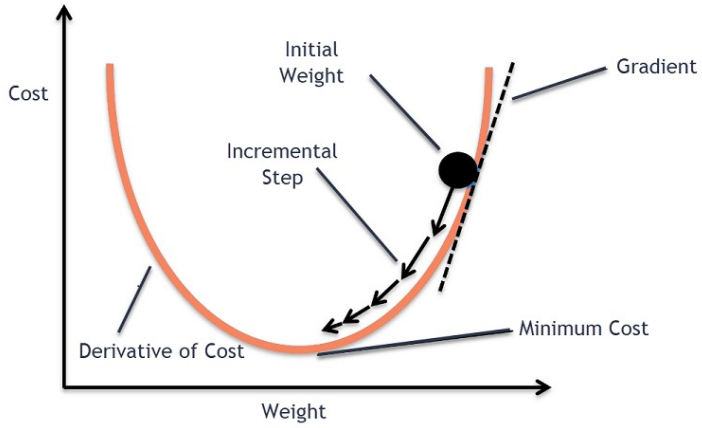
Gradient Descent



Gradient Descent

- Denotes the direction of steepest ascent

$$\nabla E(\mathbf{w}) = \begin{bmatrix} \frac{\partial E}{\partial w_0} \\ \frac{\partial E}{\partial w_1} \\ \vdots \\ \frac{\partial E}{\partial w_d} \end{bmatrix}$$



- A small step in the weight space from $\mathbf{w}$ to $\mathbf{w} + \delta\mathbf{w}$ changes the objective (or error) function. This change is maximum if $\delta\mathbf{w}$ is along the direction of the gradient at $\mathbf{w}$ and is given by:

$$\delta E \simeq \delta\mathbf{w}^{\top} \nabla E(\mathbf{w})$$

- Since $E(\mathbf{w})$ is a smooth continuous function of $\mathbf{w}$, the extreme values of E will occur at the location in the input space ($\mathbf{w}$) where the gradient of the error function vanishes, such that:

$$\nabla E(\mathbf{w}) = 0$$

- The vanishing points can be further analyzed to identify them as saddle, minima, or maxima points.

Taylor Expansion

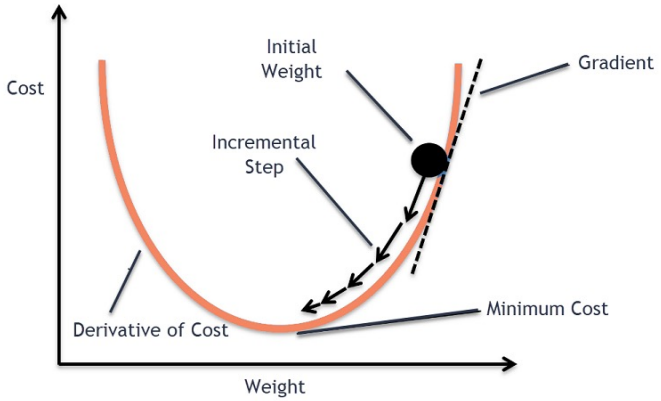
One can also derive the local approximations done by first order and second order methods using the Taylor expansion of $E(\mathbf{w})$ around some point $\mathbf{w}'$ in the weight space.

$$E(\mathbf{w}') \simeq E(\mathbf{w}) + (\mathbf{w}' - \mathbf{w})^\top \nabla + \frac{1}{2}(\mathbf{w}' - \mathbf{w})^\top \mathbf{H}(\mathbf{w}' - \mathbf{w})$$

For first order optimization methods, we ignore the second order derivative (denoted by $\mathbf{H}$ or the *Hessian*). It is easy to see that for $\mathbf{w}$ to be the local minimum, $E(\mathbf{w}) - E(\mathbf{w}') \leq 0$, $\forall \mathbf{w}'$ in the vicinity of $\mathbf{w}$. Since we can choose any arbitrary $\mathbf{w}'$, it means that every component of the gradient ∇ must be zero.

Gradient Decent

1. Start from any point in variable space
2. Move along the direction of the steepest descent
    - By how much?
    - A learning rate (η)
    - What is the direction of steepest descent?



- Gradient descent is a first-order optimization method for convex optimization problems. It is analogous to "hill-climbing" where the gradient indicates the direction of steepest ascent in the local sense.

Gradient Decent

- **Training Rule for Gradient Descent**

$$\mathbf{w} = \mathbf{w} - \eta \nabla E(\mathbf{w})$$

- For each weight component:

$$w_j = w_j - \eta \frac{\partial E}{\partial w_j}$$

  - The key operation in the above update step is the calculation of each partial derivative.

$$\begin{aligned}
\frac{\partial E}{\partial w_j} &= \frac{\partial}{\partial w_j} \frac{1}{2} \sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \\
&= \frac{1}{2} \sum_i \frac{\partial}{\partial w_j} (y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \\
&= \frac{1}{2} \sum_i 2(y_i - \mathbf{w}^\top \mathbf{x}_i) \frac{\partial}{\partial w_j} (y_i - \mathbf{w}^\top \mathbf{x}_i) \\
&= \sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i)(-x_{ij})
\end{aligned}$$

  - where x_{ij} denotes the j^{th} attribute value for the i^{th} training example.

Gradient Decent

- The final weight update rule:

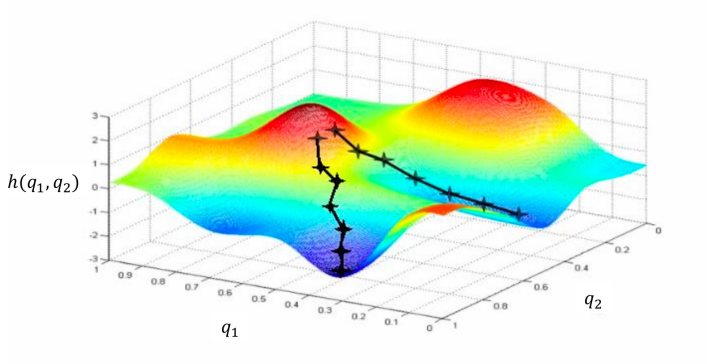
$$w_j = w_j + \eta \sum_i (y_i - \mathbf{w}^\top \mathbf{x}_i) x_{ij}$$

  - Error surface contains only one global minimum
  - Algorithm *will* converge
    - Examples need not be linearly separable
  - η should be *small enough*
  - Impact of too large η?
  - Too small η?

Issues with Gradient Decent

- Issues with Gradient Decent:
  - Slow convergence
  - Stuck in local minima

Non-convex Example



- One should note that the second issue will not arise in the case of convex problem as the error surface has only one global minima.

- More efficient algorithms exist for batch optimization, including Conjugate Gradient Descent and other quasi-Newton methods. Another approach is to consider training examples in an online or incremental fashion, resulting in an online algorithm called Stochastic Gradient Descent.

Stochastic Gradient Descent (SGD)

- **Update weights after every (or a small subset of) training example(s).**
- For sufficiently small η, closely approximates Gradient Descent.

Gradient Descent	**Stochastic Gradient Descent**
Weights updated after summing error over all examples	Weights updated after examining each example
More computations per weight update step	Significantly lesser computations
Risk of local minima	Avoids local minima

- Why SGD?

The analytical approach discussed earlier involves a matrix inversion $((\mathbf{X}^\top\mathbf{X})^{-1})$ which is a $(D+1) \times (D+1)$ matrix. Alternatively, one could solve a system of equations. When D is large, this inversion can be computationally expensive $(O * D^3)$ for standard matrix inversion. Moreover, often, the linear system might have singularities and inversion or solving the system of equations might yield numerically unstable results.

Stochastic Gradient Descent (SGD)

To compute the gradient update rule one can differentiate the error with respect to each entry of $\mathbf{w}$.

$$\begin{aligned}\frac{\partial J(\mathbf{w})}{\partial w_j} &= \frac{1}{2}\frac{\partial}{\partial w_j}\sum_{i=1}^{N}(y_i - \mathbf{w}^\top \mathbf{x}_i)^2 \\ &= \sum_{i=1}^{N}(\mathbf{w}^\top \mathbf{x}_i - y_i)x_{ij}\end{aligned}$$

N → 1 or K (a small number)

N → 1 or K (a small number)

Using the above result, one can perform repeated updates of the weights:

$$w_j := w_j - \eta\frac{\partial J(\mathbf{w})}{\partial w_j}$$

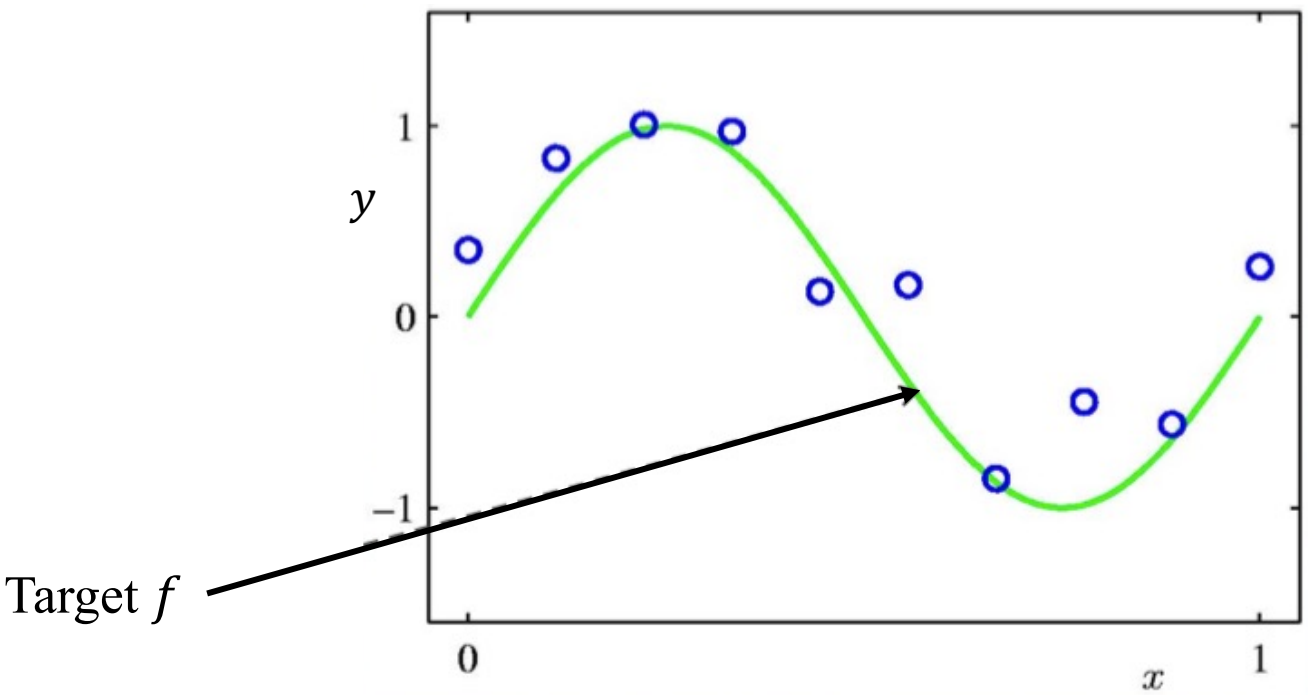
Polynomial Basis Functions

- Q: What if the raw feature is insufficient for good performance?
  - Example: non-linear dependency between label and raw feature.

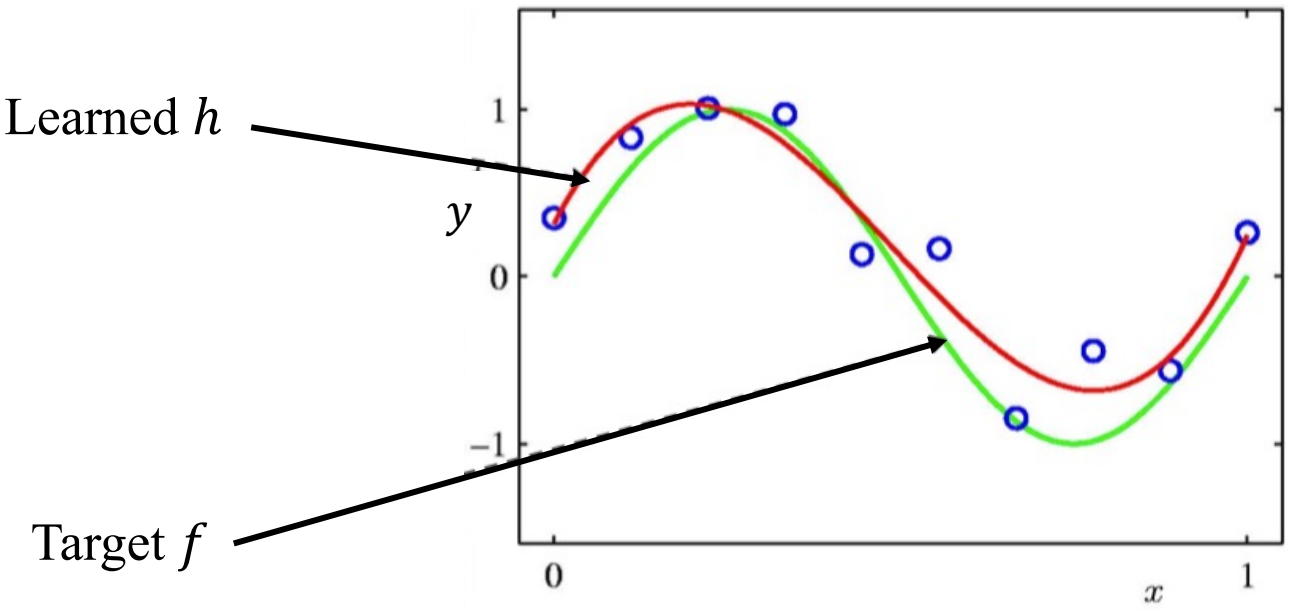
- A: Engineer / Learning higher-level features, as functions of the raw feature.

- Polynomial curve fitting:
  - Add new features, as polynomials of the original feature.

Regression: Curve Fitting

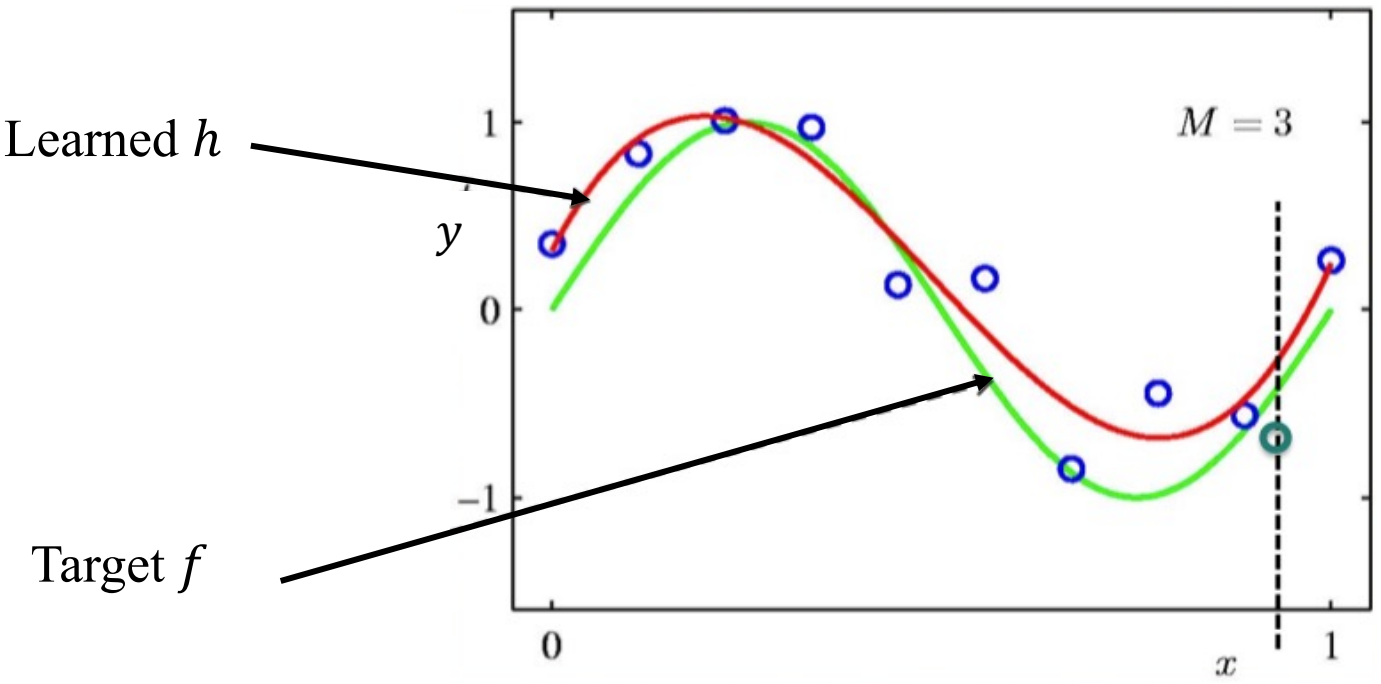


Regression: Curve Fitting



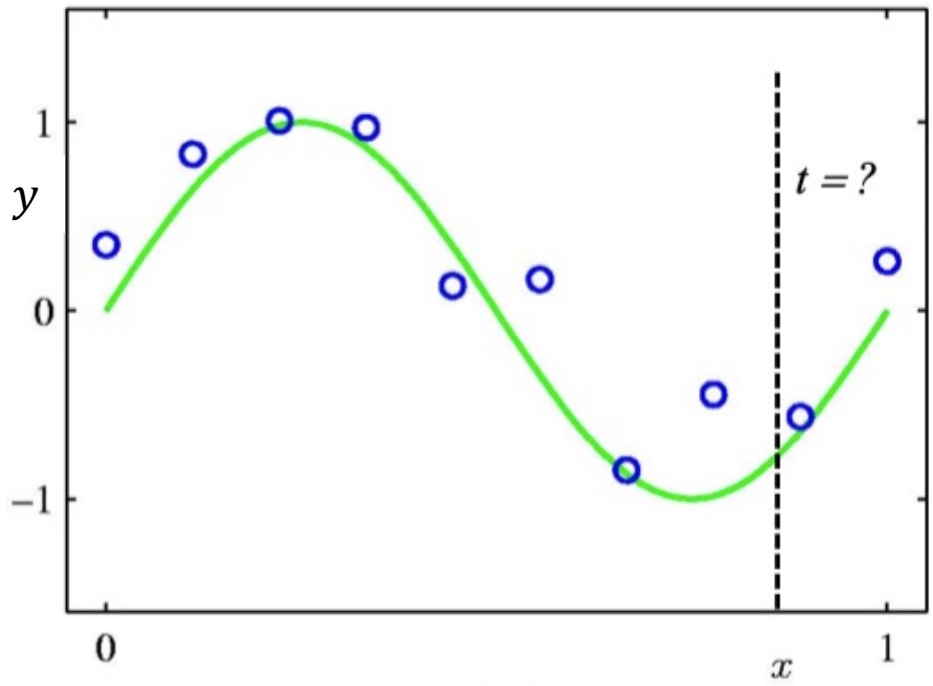
- Training: Build a function $h(x)$, based on (noisy) training examples $(x_1, y_1), (x_2, y_2), \cdots, (x_N, y_N)$.

Regression: Curve Fitting



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

Regression: Polynomial Curve Fitting



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

parameters

features

Polynomial Curve Fitting

- Parametric model:

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

- Polynomial curve fitting is (Multiple) Linear Regression:

$$\mathbf{x} = [1, x, x^2, \cdots, x^M]^T$$
$$h(x) = h(\mathbf{x}, \mathbf{w}) = h_{\mathbf{w}}(\mathbf{x}) = \mathbf{w}^{\boldsymbol{T}}\mathbf{x}$$

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

$$\widehat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w}) \qquad J(\mathbf{w}) = \frac{1}{2N} \sum_{n=1}^{N} (h_{\mathbf{w}}(\mathbf{x}_n) - y_n)^2$$

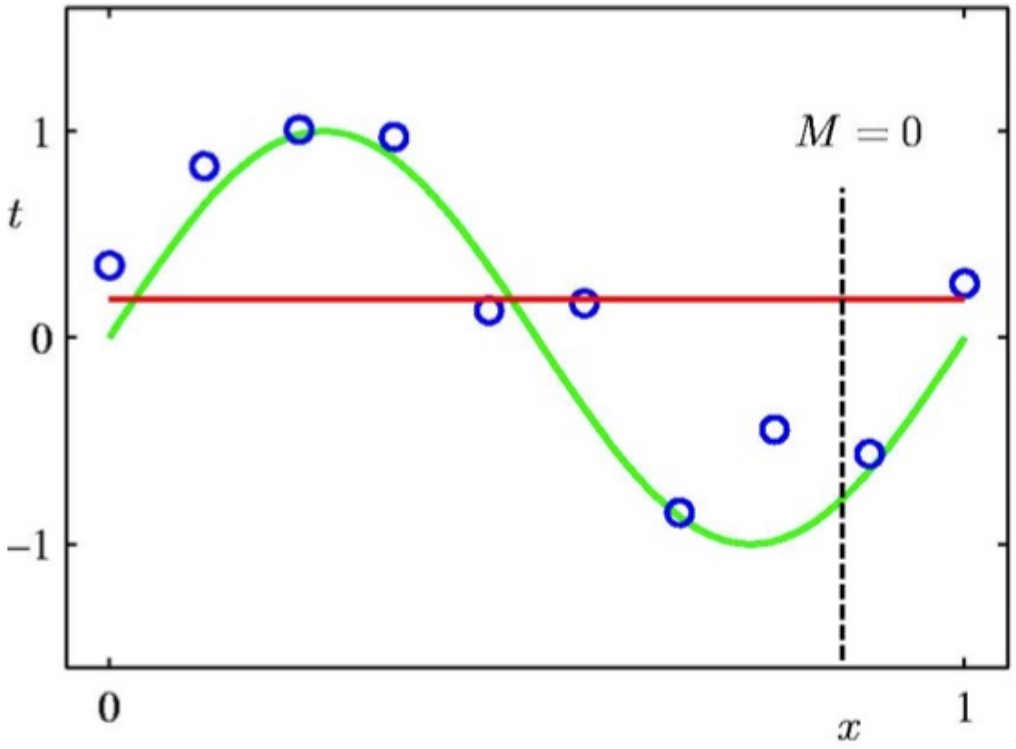
- Least Square Estimate:

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

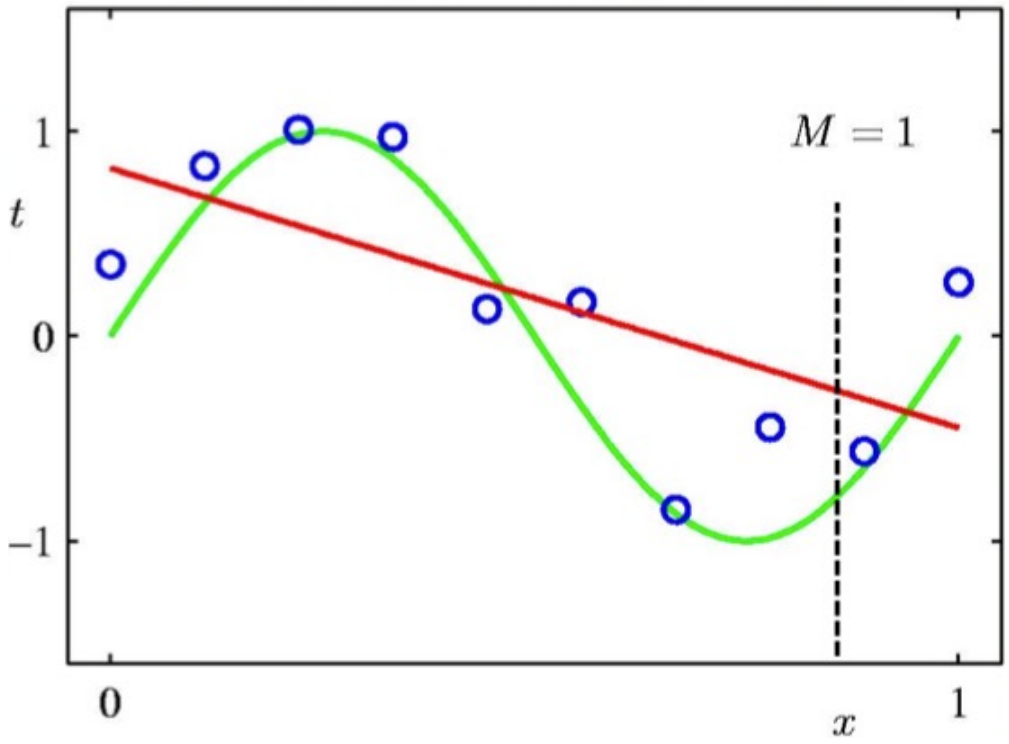
Polynomial Curve Fitting

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

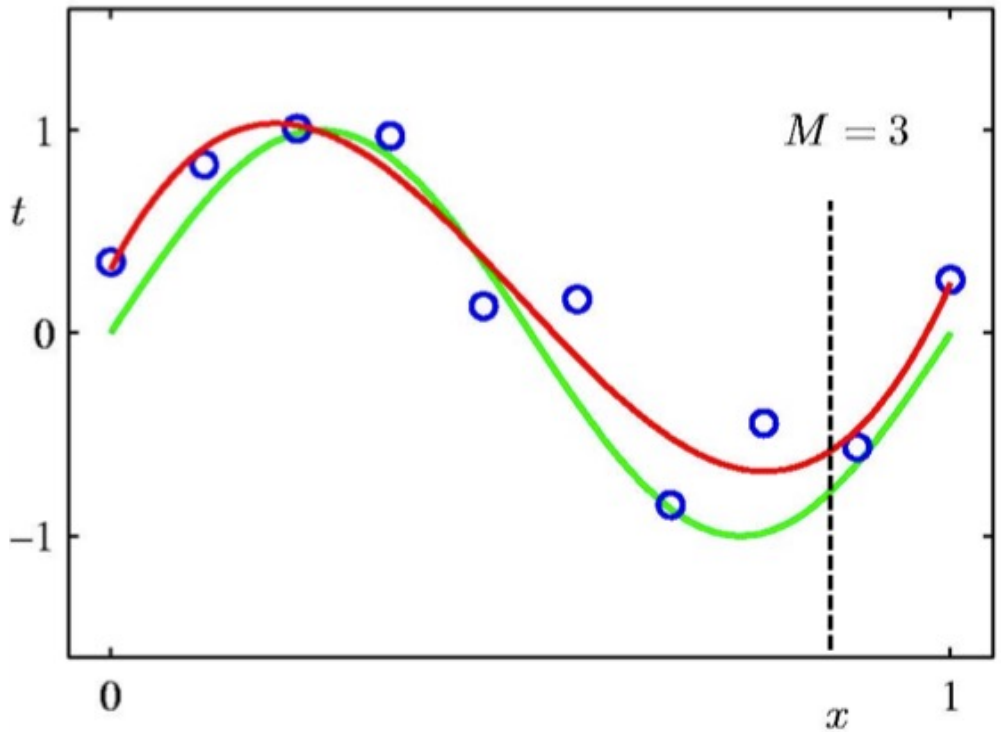
0^{th} Order Polynomial



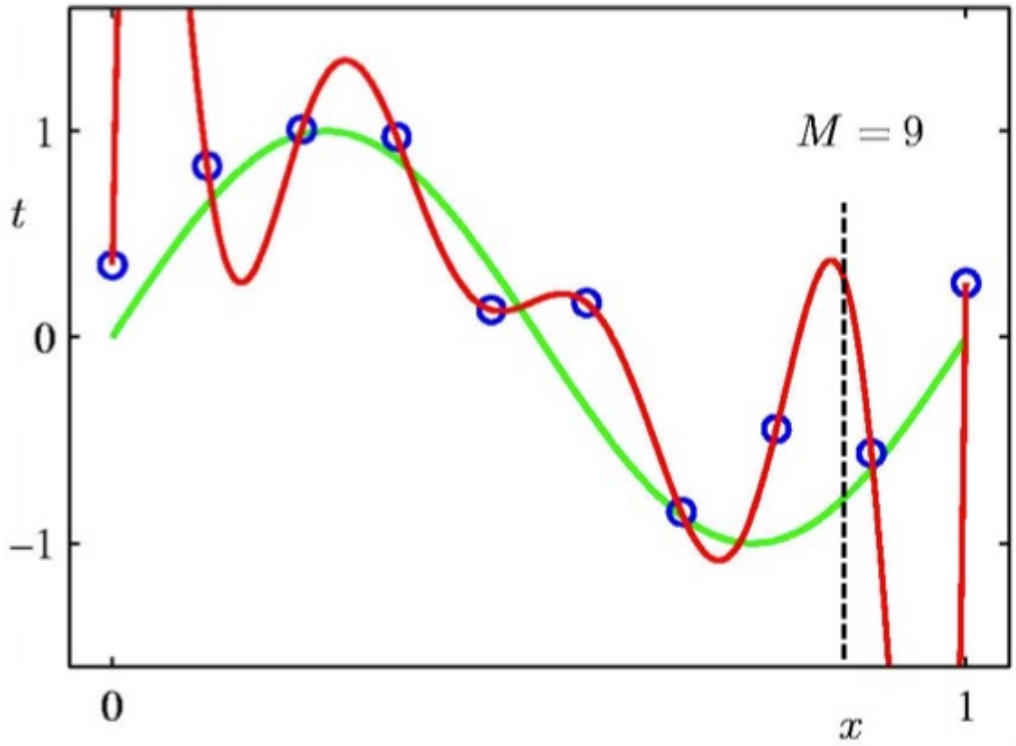
1st Order Polynomial



3rd Order Polynomial



9th Order Polynomial



- Which M to pick? Why?
- Follow the wisdom of a philosopher.

Occam's Razor

William of Occam (1288 – 1348)

English Franciscan friar, theologian and philosopher.

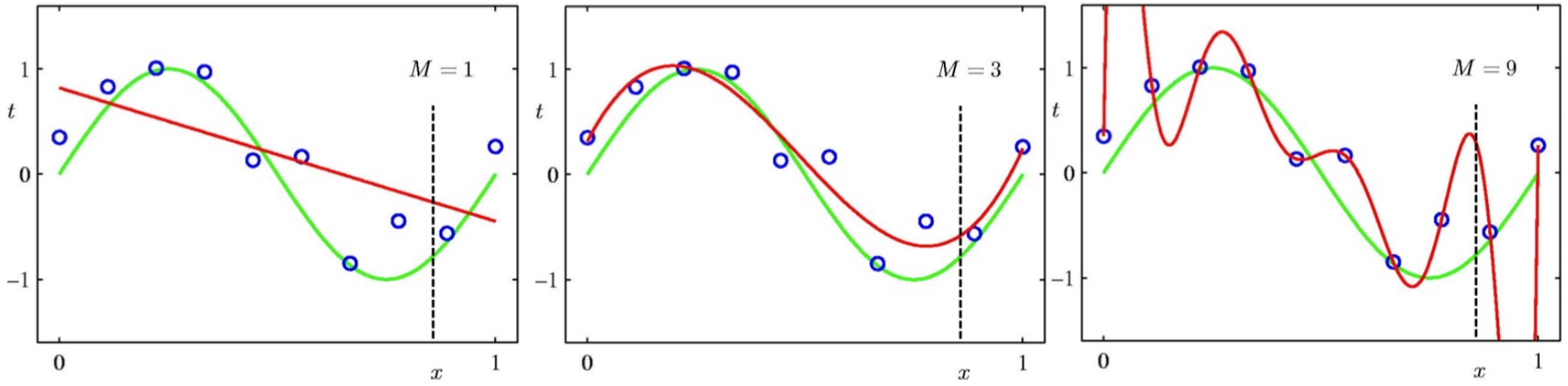
"*Entia non sunt multiplicanda praeter necessitatem*"

- 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.

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.

Over-fitting and Parameter Values

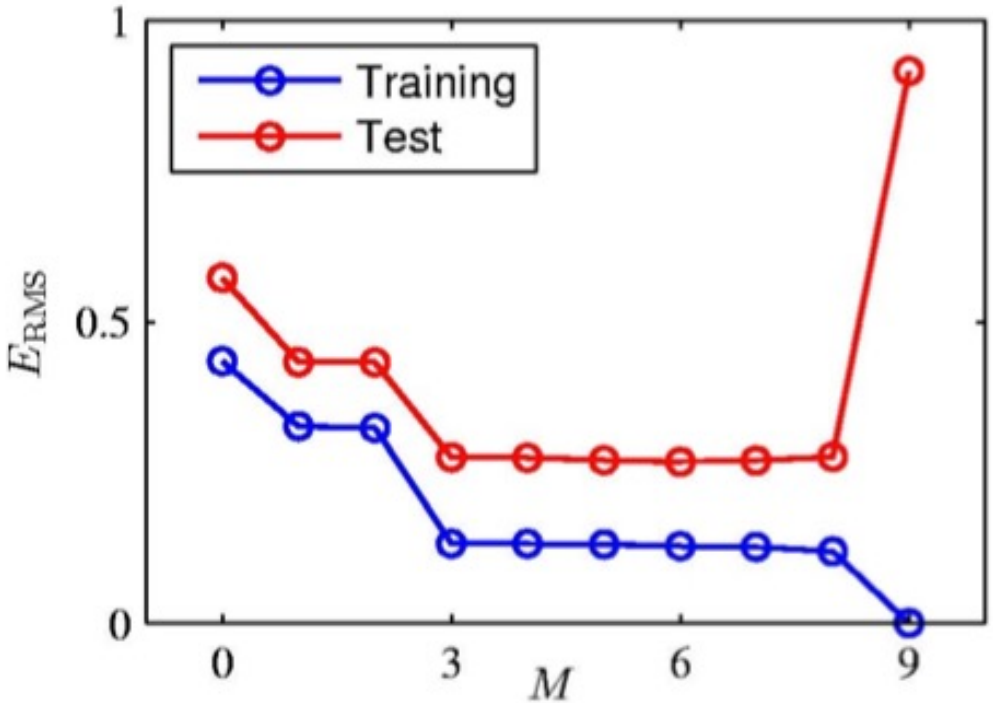
	$M = 0$	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.19	0.82	0.31	0.35
$w_1^\star$		-1.27	7.99	232.37
$w_2^\star$			-25.43	-5321.83
$w_3^\star$			17.37	48568.31
$w_4^\star$				-231639.30
$w_5^\star$				640042.26
$w_6^\star$				-1061800.52
$w_7^\star$				1042400.18
$w_8^\star$				-557682.99
$w_9^\star$				125201.43

Overfitting

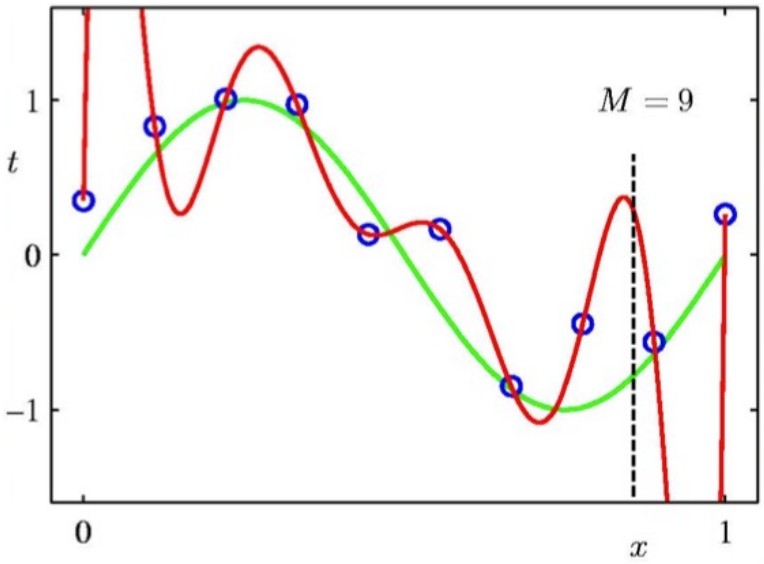
- Measure fit using the Root-Mean-Square (RMS) error (RMSE):

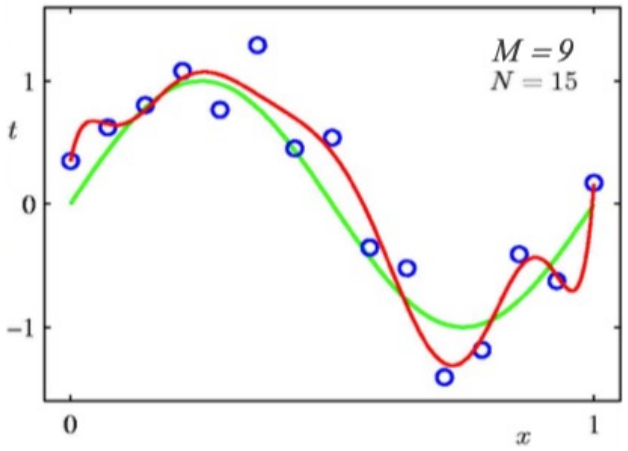
$$E_{RMS(\mathbf{w})} = \sqrt{\frac{\sum_n (\mathbf{w}^T \mathbf{x}_n - \boldsymbol{t}_n)^2}{N}}$$

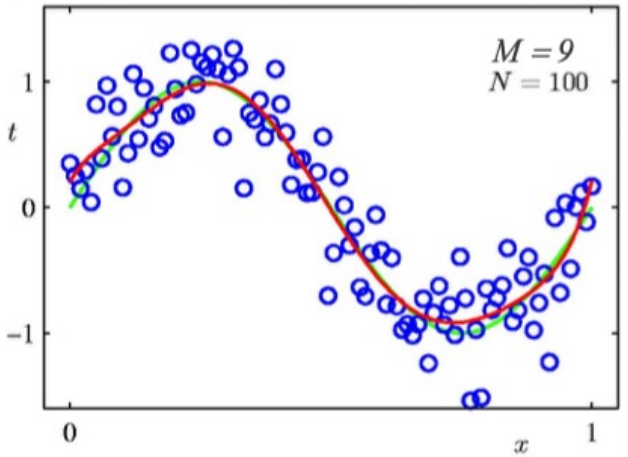
- Use 100 random test examples, generated in the same way:



Overfitting vs. Data Set Size







- More training data $\Longrightarrow$ less overfitting

- What if we do not have more training data?
  - Use **regularization**

Regularization

- Penalize large parameter values:

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

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} E(\mathbf{w})$$

Ridge Regression

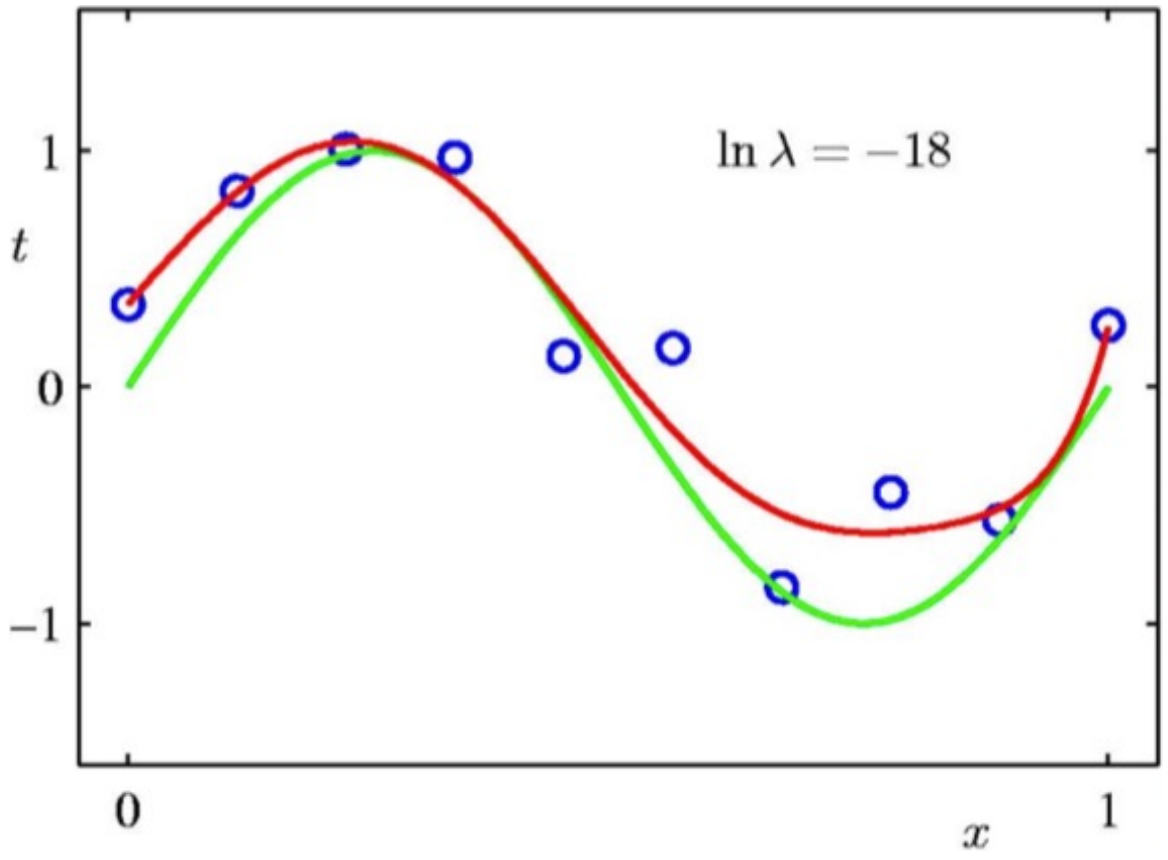
- Multiple linear regression with L2 regularization:

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

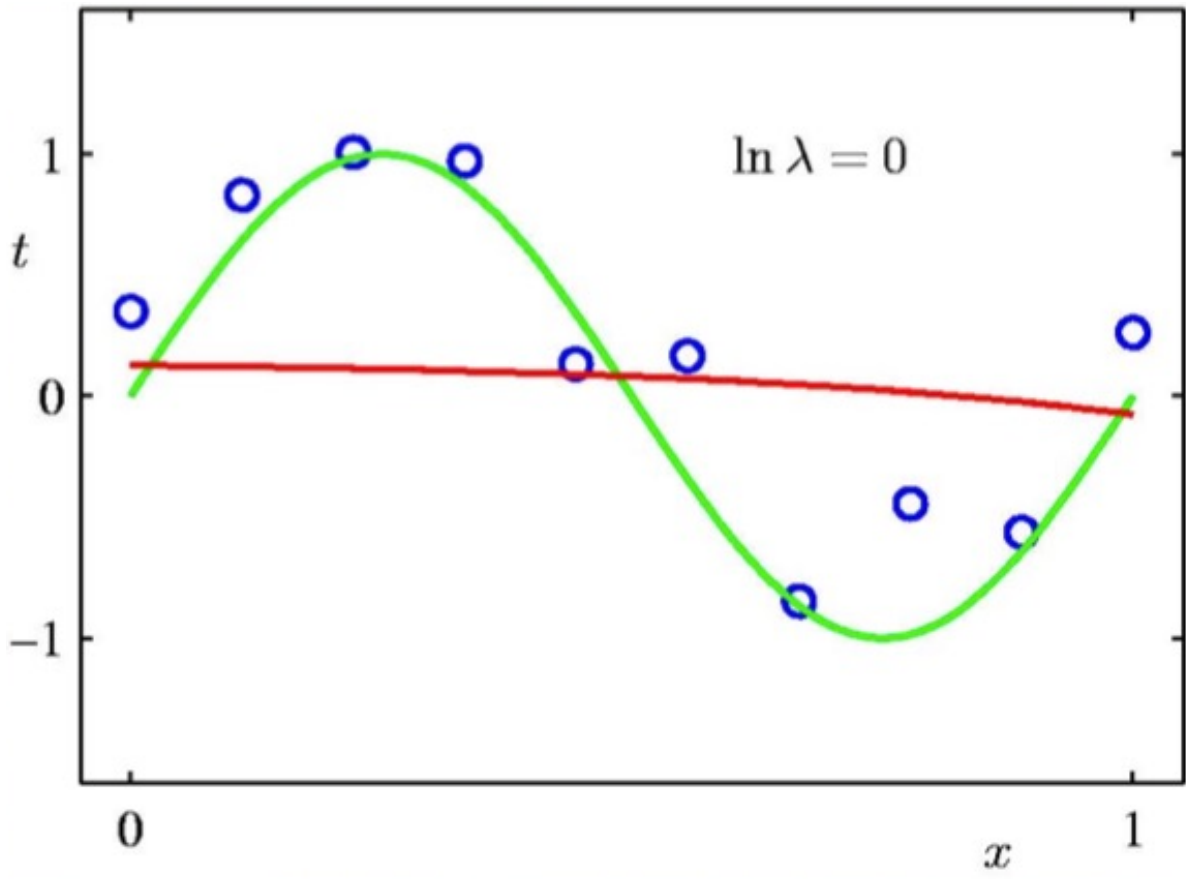
$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}} J(\mathbf{w})$$

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

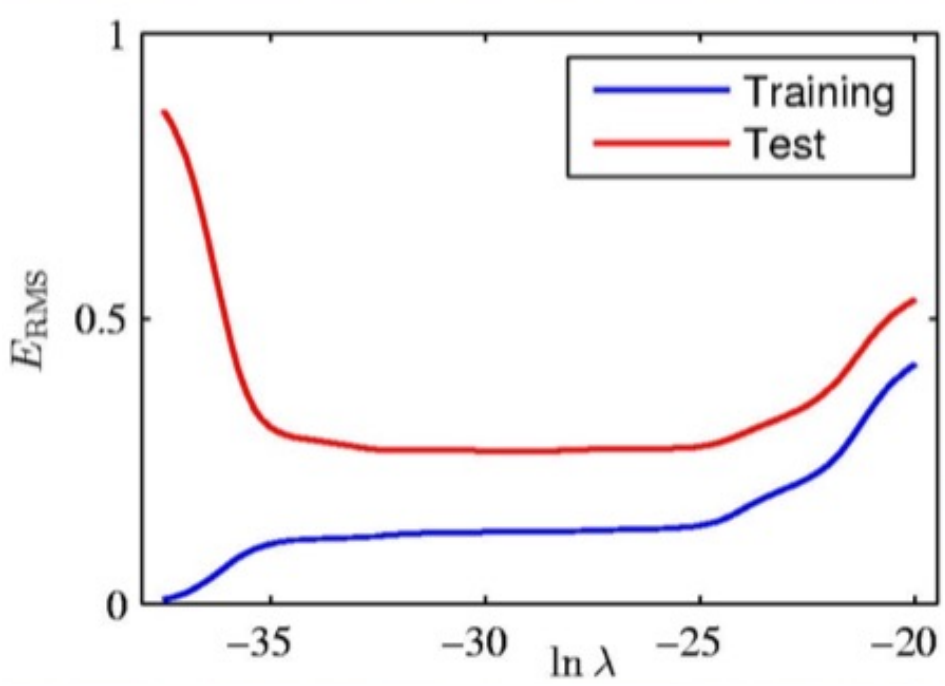
9th Order Polynomial with Regularization



9th Order Polynomial with Regularization



Training & Test error vs. ln λ

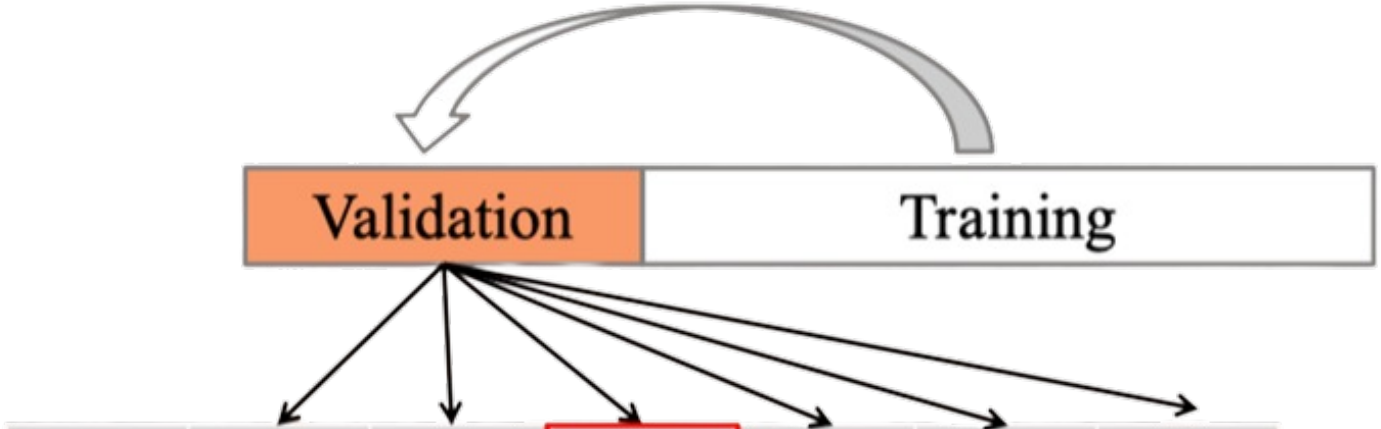


How do we find the optimal value of λ?

Model Selection

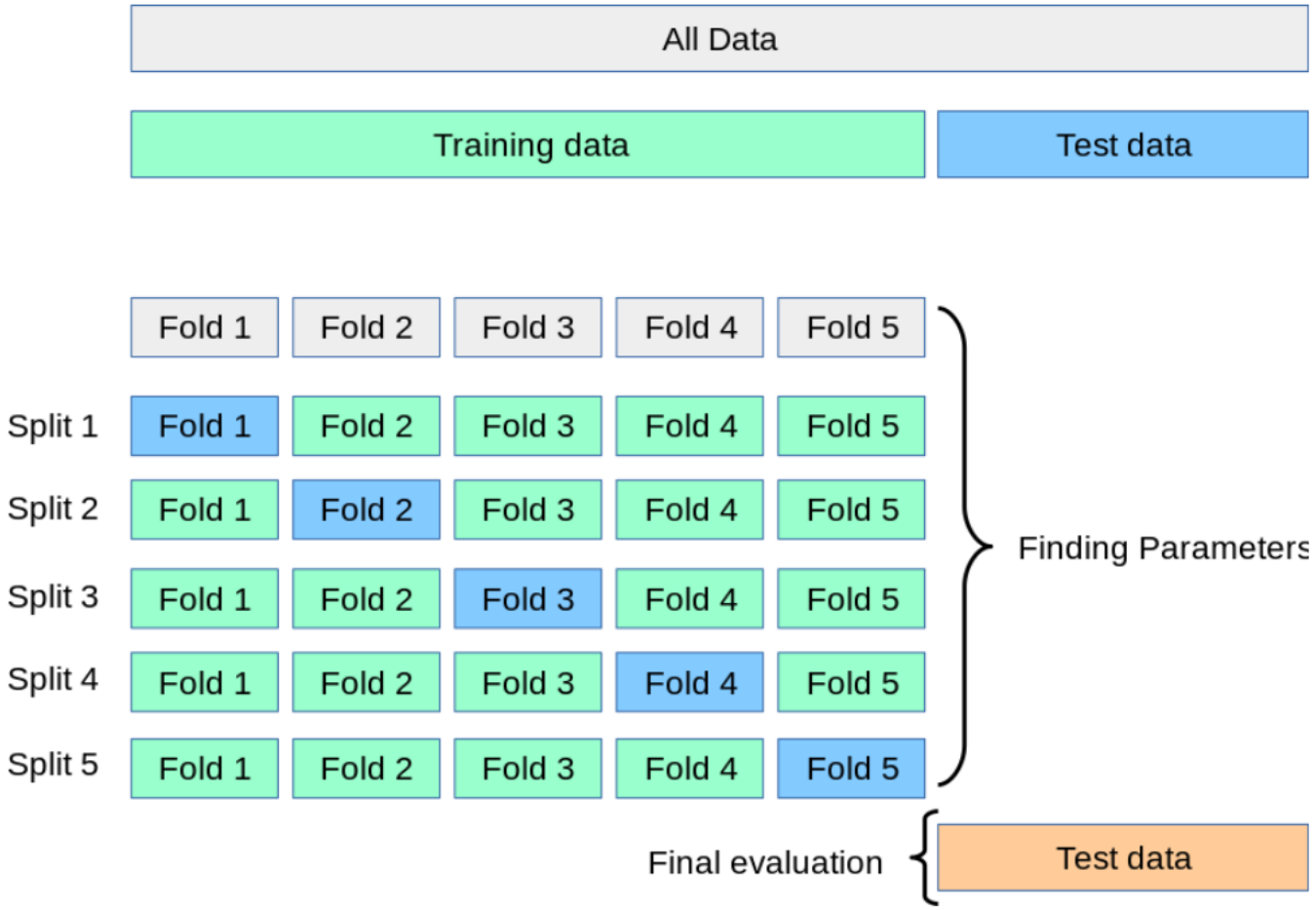
- Put aside an independent validation set.
- Select parameters giving best performance on validation set.

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



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

K-fold Cross-Validation



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

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 $k = 1,2,\cdots,K$
      - 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.

Regularization: Ridge vs. Lasso

- Ridge regression:

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

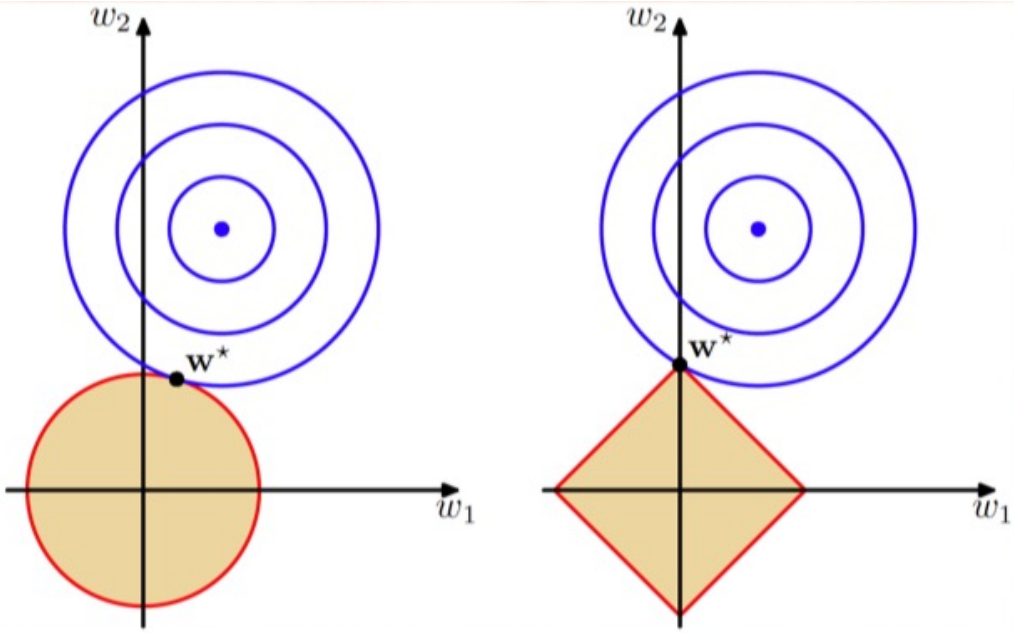
- Lasso:

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

- if λ is sufficiently large, some of the coefficients w_j are driven to $0 \Longrightarrow$ sparse model

Regularization: Ridge vs. Lasso

Plot of the contours of the unregularized error function (blue) along with the constraint region (3.30) for the quadratic regularizer $q = 2$ on the left and the lasso regularizer $q = 1$ on the right, in which the optimum value for the parameter vector $\mathbf{w}$ is denoted by $\mathbf{w}^*$. The lasso gives a sparse solution in which $\mathbf{w}^* = \mathbf{0}$.



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.

Questions?

UNIVERSITY OF NORTH CAROLINA CHARLOTTE