Institute for Advanced Studies
in Basic Sciences
Gava Zang, Zanjan, Iran

Regression with Linear Models

By Dr. Parvin Razzaghi (p.razzaghi@iasbs.ac.ir)

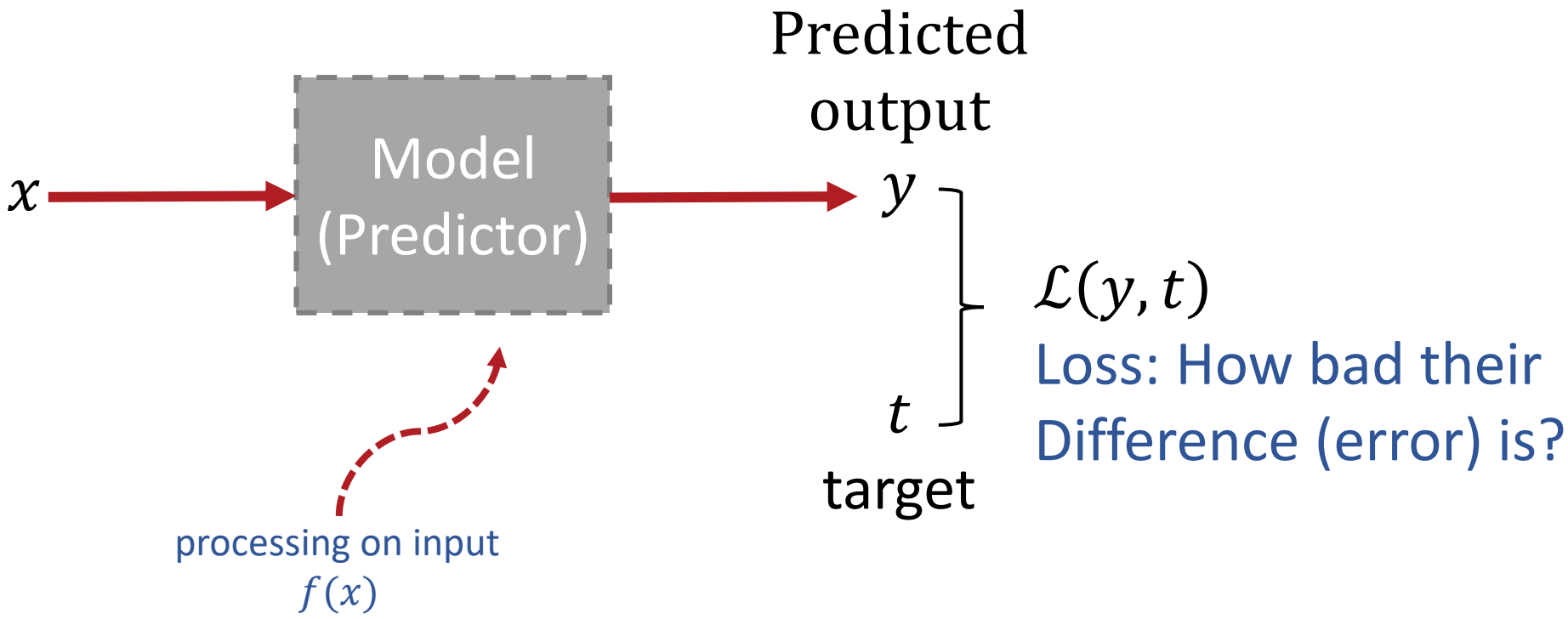
Winter 2024

Slides credit

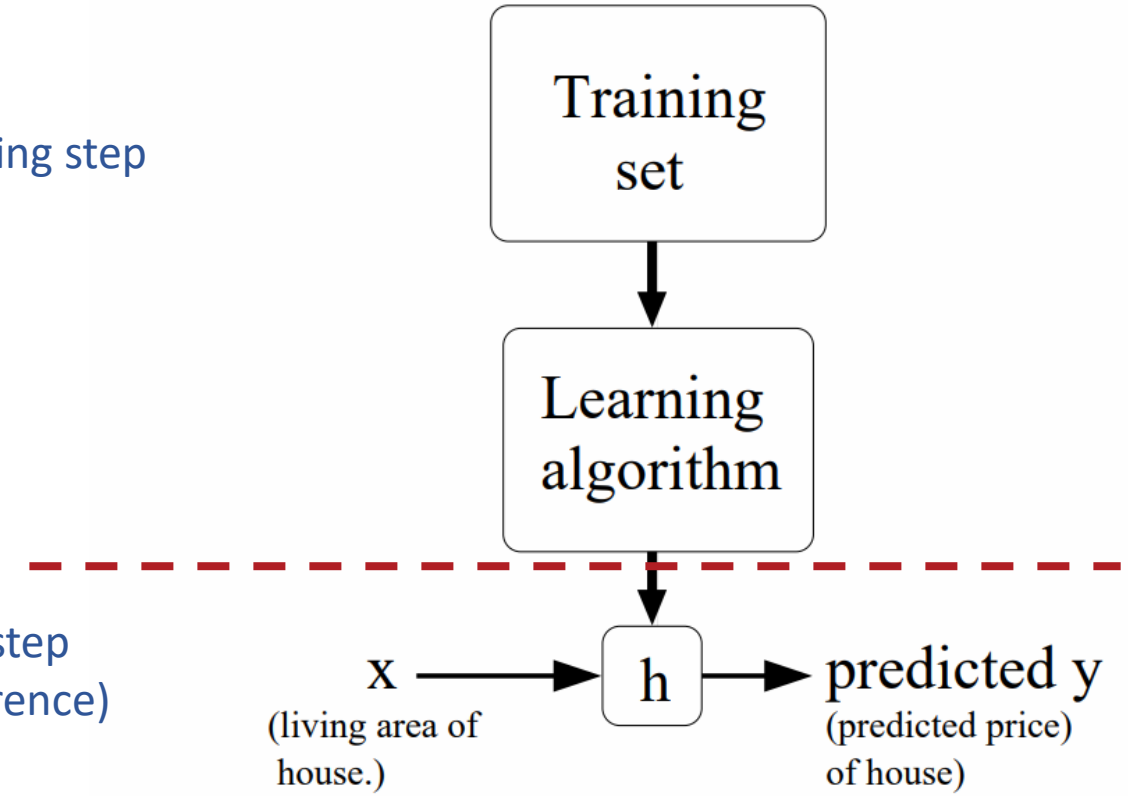
- Professor:
  - Dr. Parvin Razzaghi
- Original Slides:
  - Amir-massoud Farahman
  - Eric Eaton
- Link:
  - https://B2n.ir/n25709
- University:
  - University of Toronto and Vector Institute
- Acknowledgment:
  - Credit for slides goes to many members of the ML Group at the U of T, and beyond, including (recent past): Roger Grosse, Murat Erdogdu, Richard Zemel, Juan Felipe Carrasquilla, Emad Andrews.
  - For any comments or suggestions, please contact: p.razzaghi@iasbs.ac.ir

Table of Contents

Modular Approach ML Algorithm Design



Training step



Modular Approach ML Algorithm Design

- we will take a more modular approach:
  - define the problem
  - provide a training dataset
  - choose a model describing the relationships between variables of interest
  - define a loss function quantifying how bad the fit to the data is
  - (possibly) choose a regularizer saying how much we prefer different candidate models (or explanations of data), before (prior to) seeing the data
  - fit the model that minimizes the loss function and satisfy the constraint/penalty imposed by the regularizer, possibly using an optimization algorithm
- Mixing and matching these modular components gives us a lot of new ML methods.

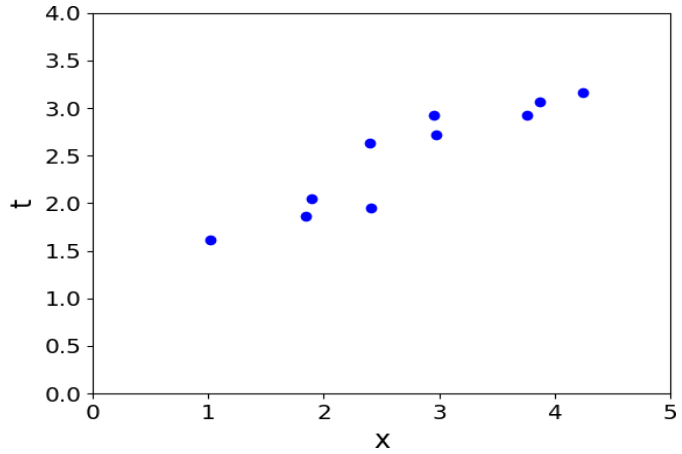
Define the problem & provide a dataset

The Supervised Learning Setup

- Features
  - Living area
  - Number of bedrooms
  - …
- Target variable
  - Price

Living area (feet^2)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
1416	2	232
3000	4	540
⋮	⋮	⋮

The Supervised Learning Setup



Recall that in supervised learning:

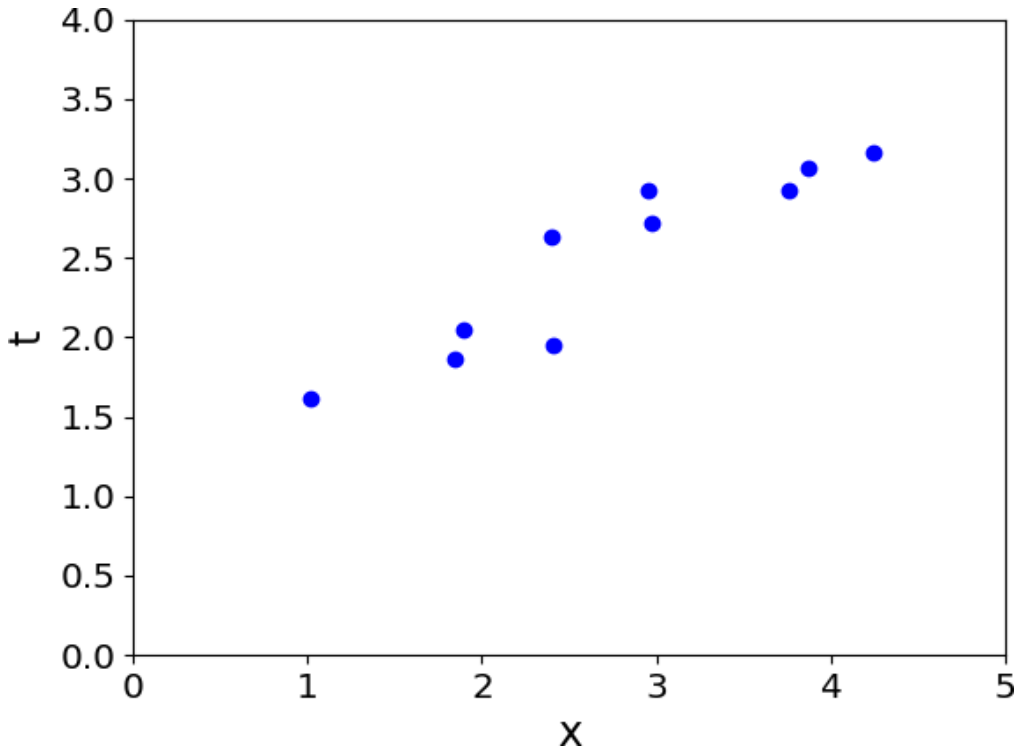
- There is a target $t \in T$ (also called response, outcome, output, class)
- There are features $\boldsymbol{x} \in X$ (also called inputs or covariates)

  The goal is to learn a function $f : X \rightarrow T$ such that

$$t \approx y = f(x),$$

based on given data $D = \{(\boldsymbol{x}^{(i)}, t^{(i)}) \; for \; i = 1, 2, \ldots, N\}$.

choose a model describing the relationships between variables of interest?


4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
t
0
1
2
3
4
5
x

Regression with Linear Models

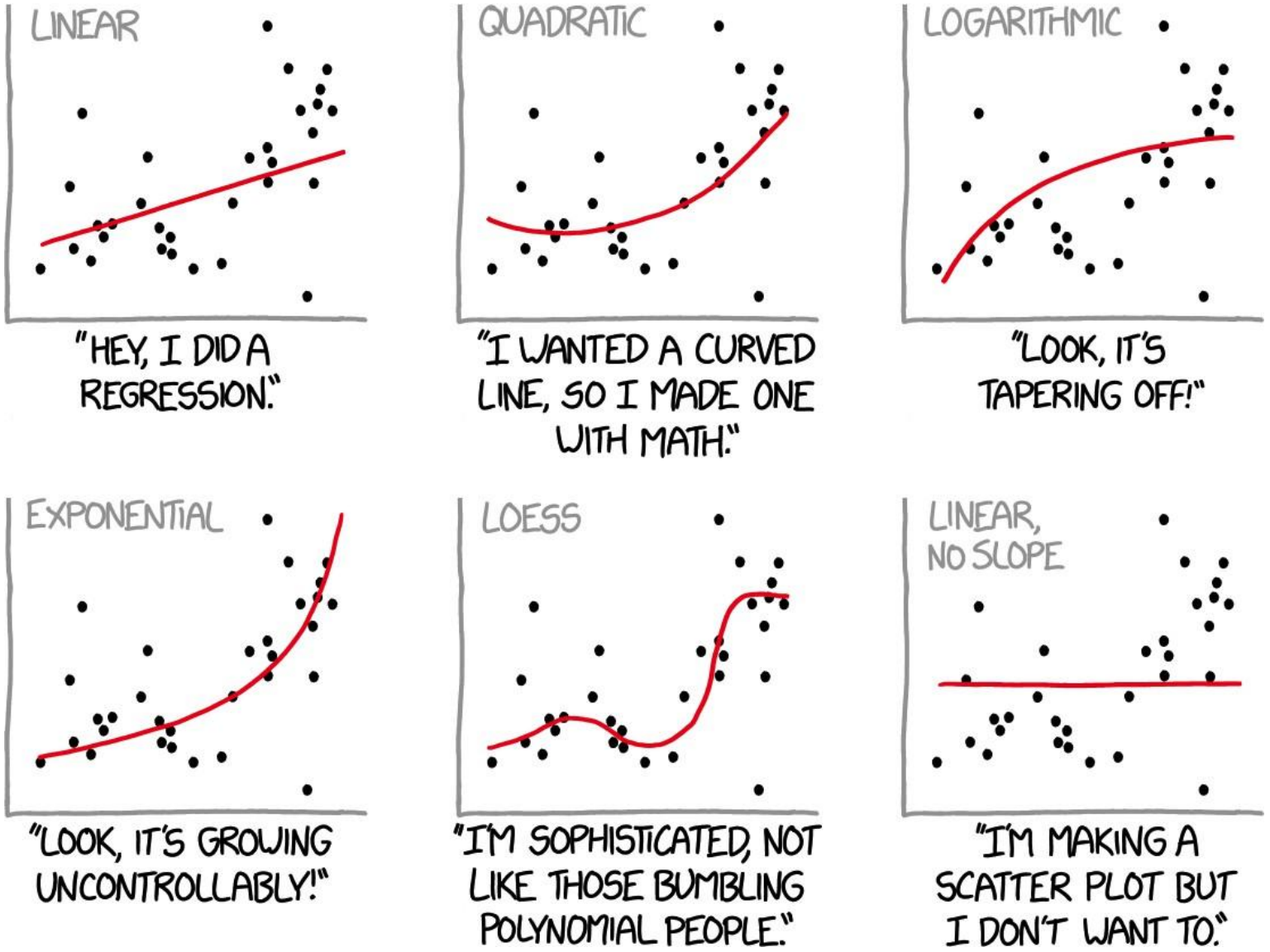


Image credit: xkcd (cropped)

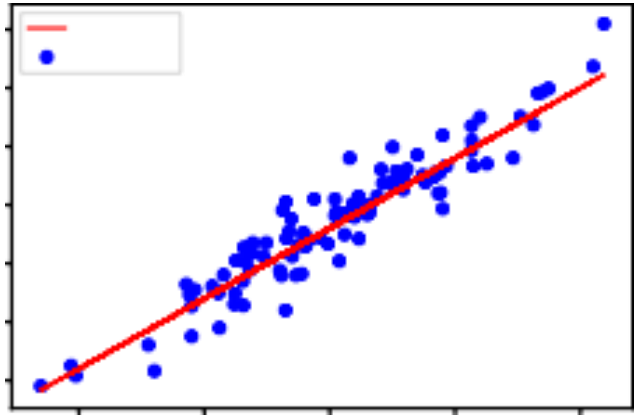
Linear Regression – Model

- Model: In linear regression, we use linear functions of the inputs $\boldsymbol{x} = (x_1, \ldots, x_D)$ to make predictions y of the target value t:

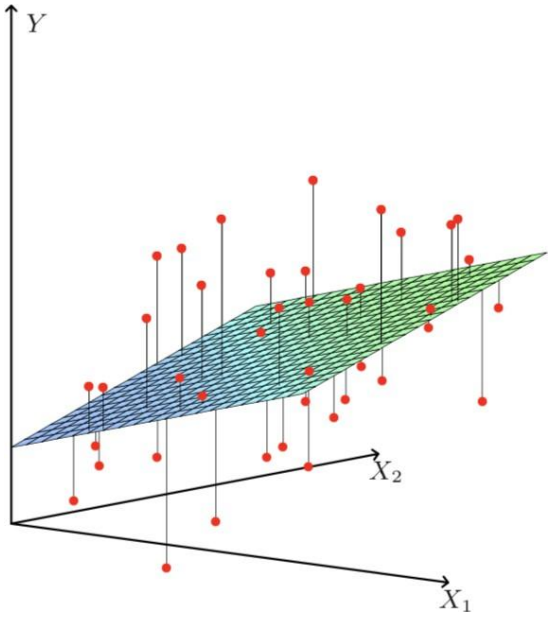
$$y = f(x) = \sum_j w_j x_j + b$$

  - y is the prediction
  - $\boldsymbol{w}$ is the weights
  - b is the bias (or intercept) (do not confuse with the bias-variance tradeoff in the next lecture)
- $\boldsymbol{w}$ and b together are the parameters
- We hope that our prediction is close to the target: $y \approx t$.

What is Linear? 1 Feature vs. D Features



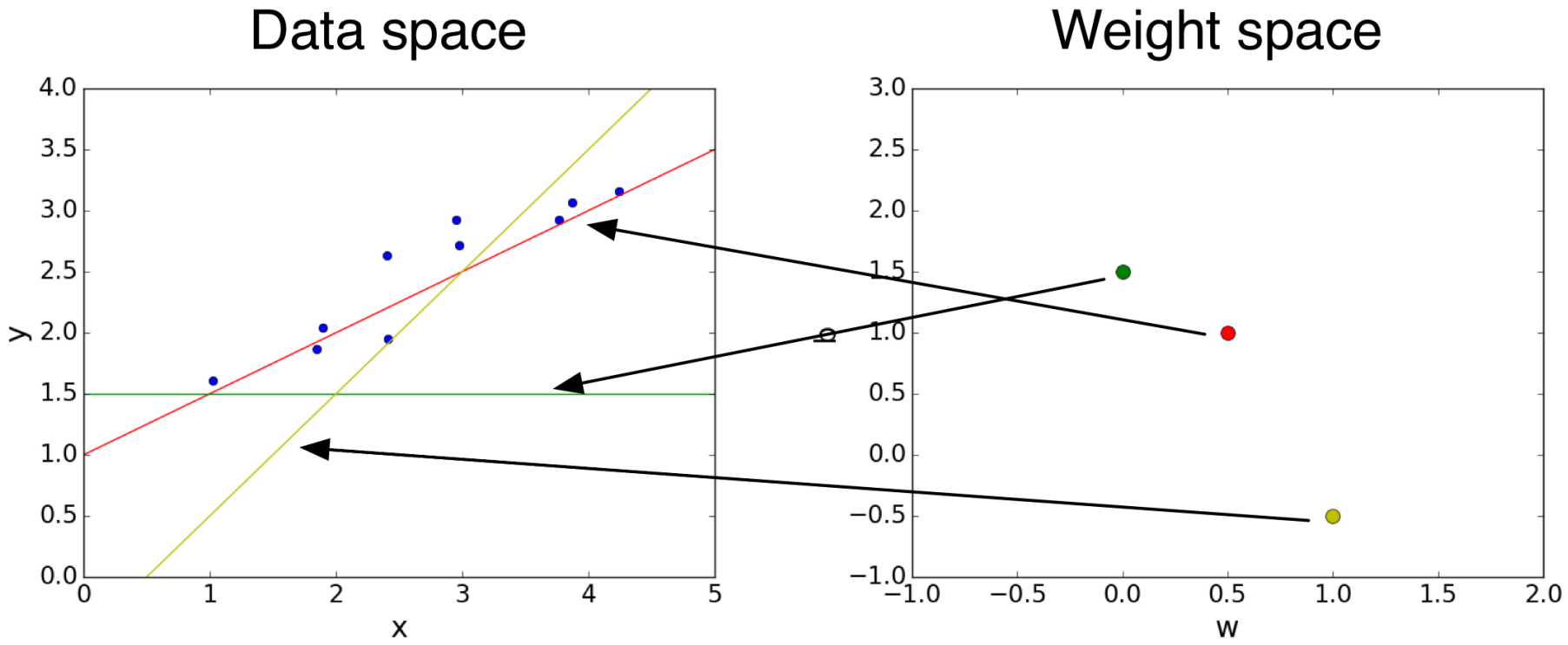
- If we have only 1 feature:
  $y = wx + b$ where $w, x, b \in R$
- y is linear in x



- If we have D features:
  $y = \boldsymbol{w}^T \boldsymbol{x} + b$ where $\boldsymbol{w}, \boldsymbol{x} \in R^D, b \in R$
- y is linear in $\boldsymbol{x}$

Relation between the prediction y and inputs **x** is linear in both cases.

Weight Space vs. Data Space



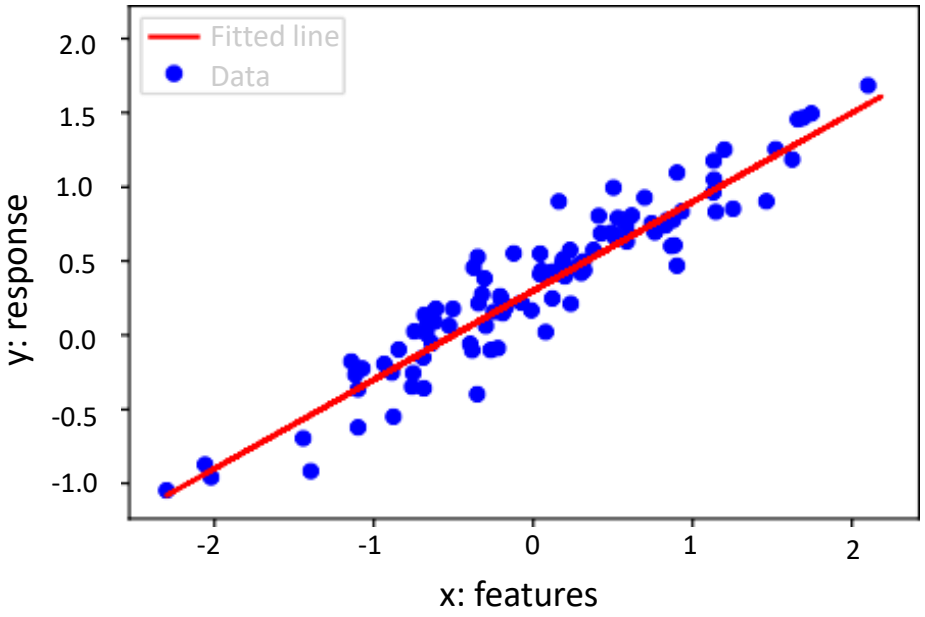
Recall that:

$$y = f(x) = \sum_j w_j x_j + b$$

Linear Regression

We have a dataset $D = \{(x^{(i)}, t^{(i)})\}_{i=1}^{N}$ where:

- $x^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_D^{(i)})^T \in \mathbb{R}^D$ are the inputs, e.g., age, height
- $t^{(i)} \in \mathbb{R}$ is the target or response, e.g., income
- predict $t^{(i)}$ with a linear function of $x^{(i)}$:



- $t^{(i)} \approx y^{(i)} = \boldsymbol{w}^T \boldsymbol{x}^{(i)} + b$
- Find the "best" line $(\boldsymbol{w}, b)$.

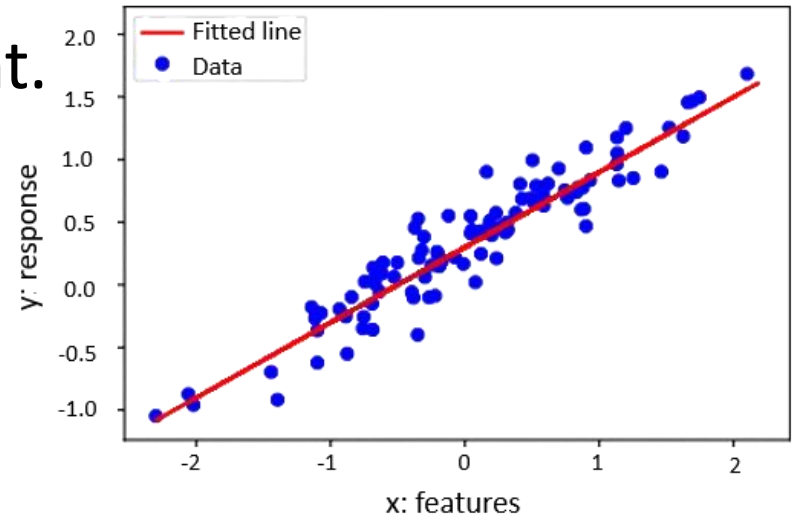
- Q: How should we find the best line?

Linear Regression – Loss Function

- How to quantify the quality of the fit to data?
- A loss function $\mathcal{L}(y, t)$ defines how bad it is if, for some input $\boldsymbol{x}$, the algorithm predicts y, but the target is actually t.
- Squared error loss function:

$$\mathcal{L}(y, t) = \frac{1}{2}(y - t)^2$$

- $y - t$ is the residual, and we want to make its magnitude small
- The $\frac{1}{2}$ factor is just to make the calculations convenient.



Linear Regression – Loss Function

- Cost function: loss function averaged over all training examples

$$\mathcal{J}(\boldsymbol{w}, b) = \frac{1}{N}\sum_{i=1}^{N} \mathcal{L}\left(y^{(i)}, t^{(i)}\right)$$

$$= \frac{1}{2N}\sum_{i=1}^{N}\left(y^{(i)} - t^{(i)}\right)^2$$

$$= \frac{1}{2N}\sum_{i=1}^{N}(\boldsymbol{w}^T x^{(i)} + b - t^{(i)})^2$$

Linear Regression – Loss Function

- To find the best fit, we find a model (parameterized by its weights $\boldsymbol{w}$ and b) that minimizes the cost:

$$\underset{(\boldsymbol{w}, b)}{\text{minimize}}\, \mathcal{J}(\boldsymbol{w}, b) = \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(y^{(i)}, t^{(i)}\right)$$

- The terminology is not universal. Some might call “loss” pointwise loss and the “cost function” the empirical loss or average loss.

Vector Notation

- We can organize all the training examples into a design matrix **X** with one row per training example, and all the targets into the target vector **t**.

$$X = \begin{bmatrix} x^{(1)T} \\ x^{(2)T} \\ x^{(3)T} \end{bmatrix} = \begin{bmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 8 \end{bmatrix}$$

One feature across all training examples

One training example (vector)

- Computing the predictions for the whole dataset:

$$\boldsymbol{X}\boldsymbol{w} + b\mathbf{1} = \begin{bmatrix} w^T x^{(1)} + b \\ \vdots \\ w^T x^{(N)} + b \end{bmatrix} = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(N)} \end{bmatrix} = y$$

Vectorization

- Computing the squared error cost across the whole dataset:

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{w} + b\mathbf{1}$$

$$\mathcal{J} = \frac{1}{2N} \|y - t\|^2$$

- Note that sometimes we may use $\mathcal{J} = \frac{1}{2N} \|y - t\|^2$, without $\frac{1}{N}$ normalizer. That would correspond to the sum of losses, and not the average loss. That does not matter as the minimizer does not depend on N .

Vectorization

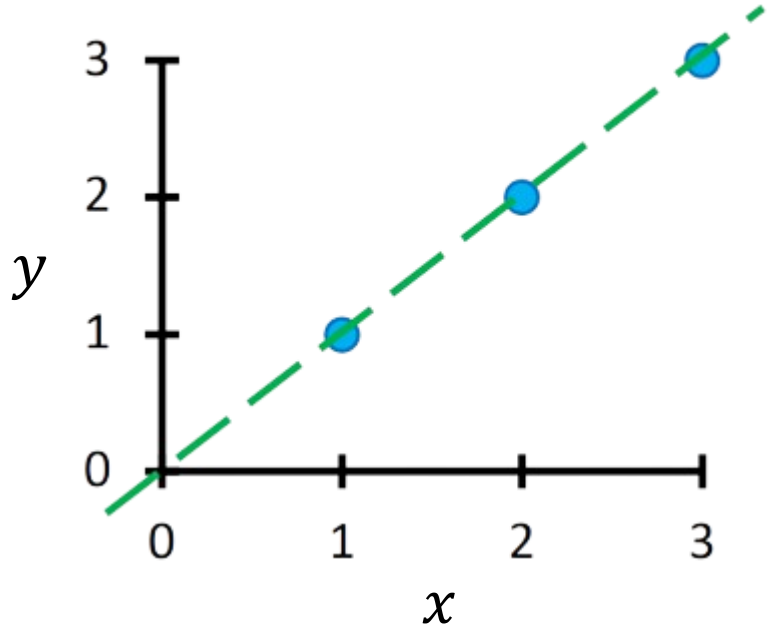
- We can also add a column of 1s to the design matrix, combine the bias and the weights, and conveniently write

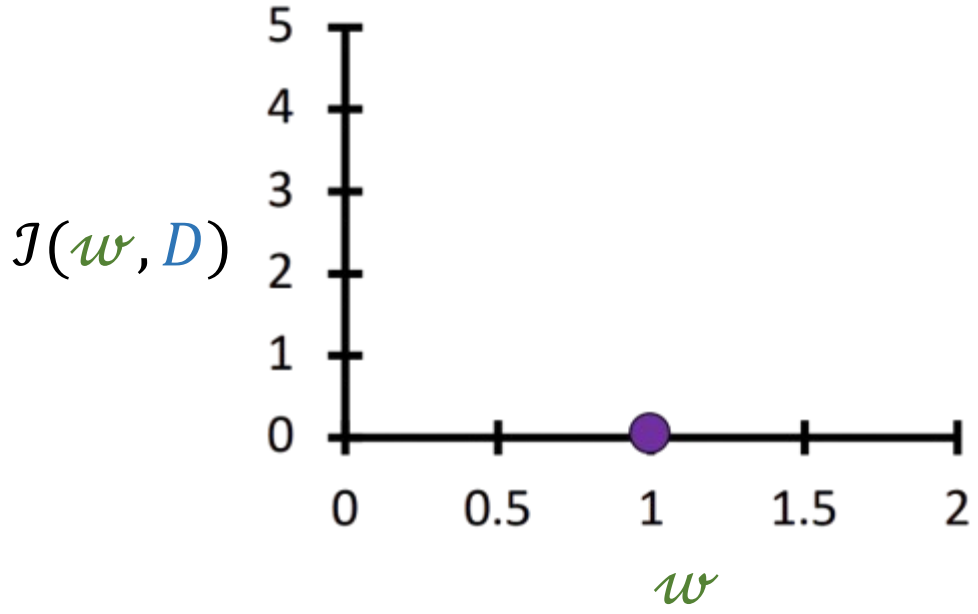
$$X = \begin{bmatrix} 1 & \left[x^{(1)}\right]^T \\ 1 & \left[x^{(2)}\right]^T \\ 1 & \vdots \end{bmatrix} \in \mathbb{R}^{N \times D+1} \text{ and } w = \begin{bmatrix} b \\ w_1 \\ w_2 \\ \vdots \end{bmatrix} \in \mathbb{R}^{D+1}$$

Then, our predictions reduce to $\boldsymbol{y} = \boldsymbol{Xw}$.

Intuition on Minimizing MSE Loss

- Consider $x \in \mathbb{R}$ and $w \in \mathbb{R}$

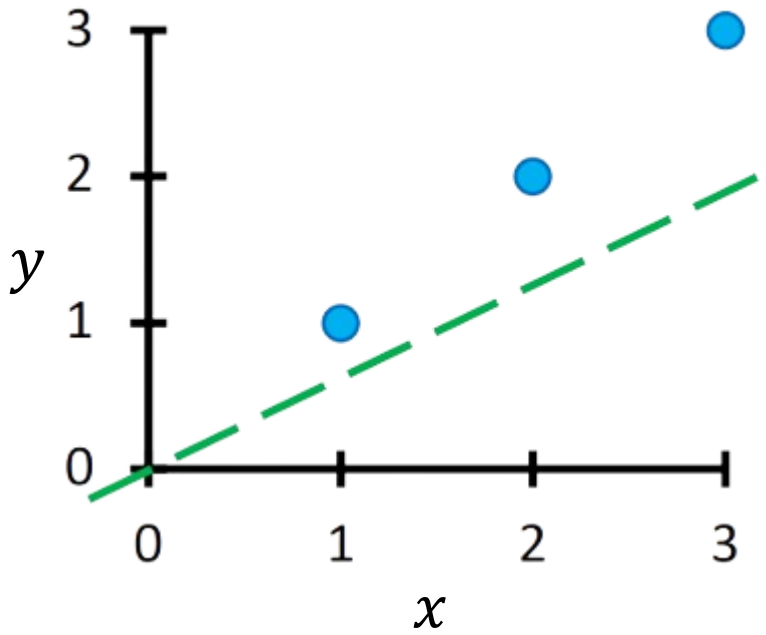


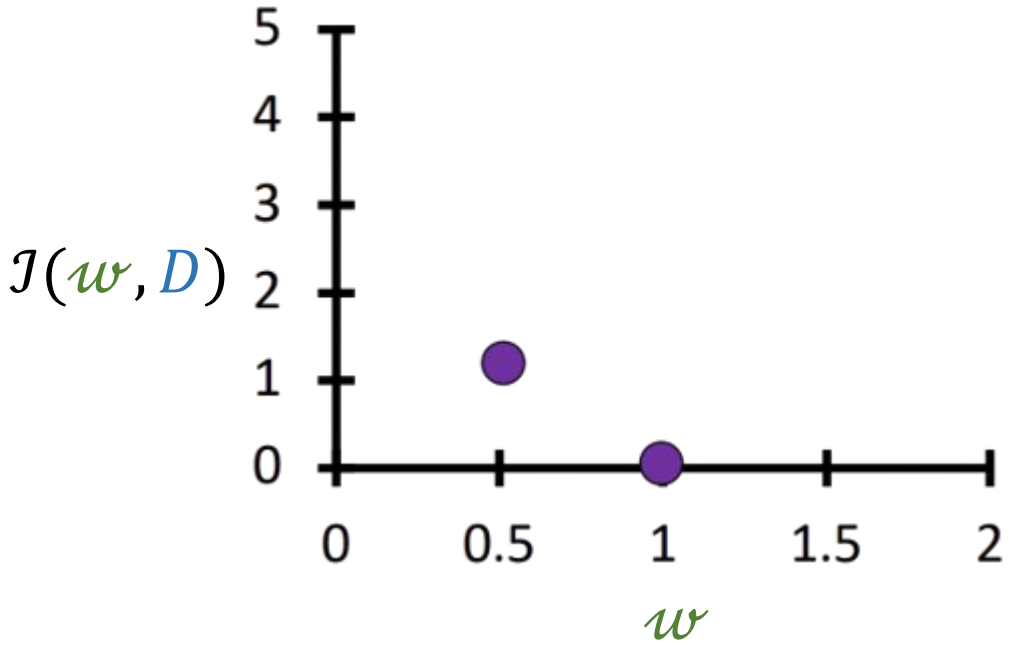


$w = 1$

Intuition on Minimizing MSE Loss

- Consider $x \in \mathbb{R}$ and $w \in \mathbb{R}$

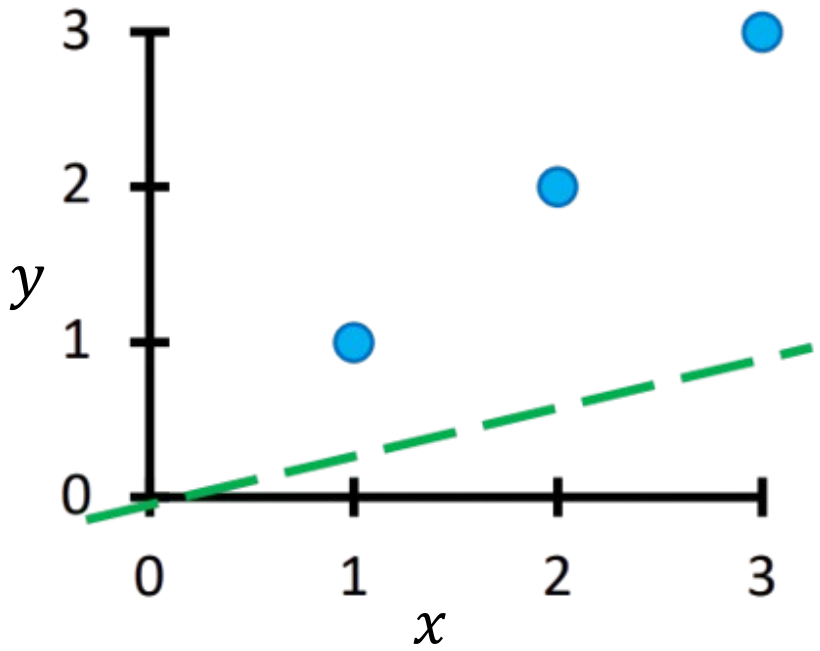




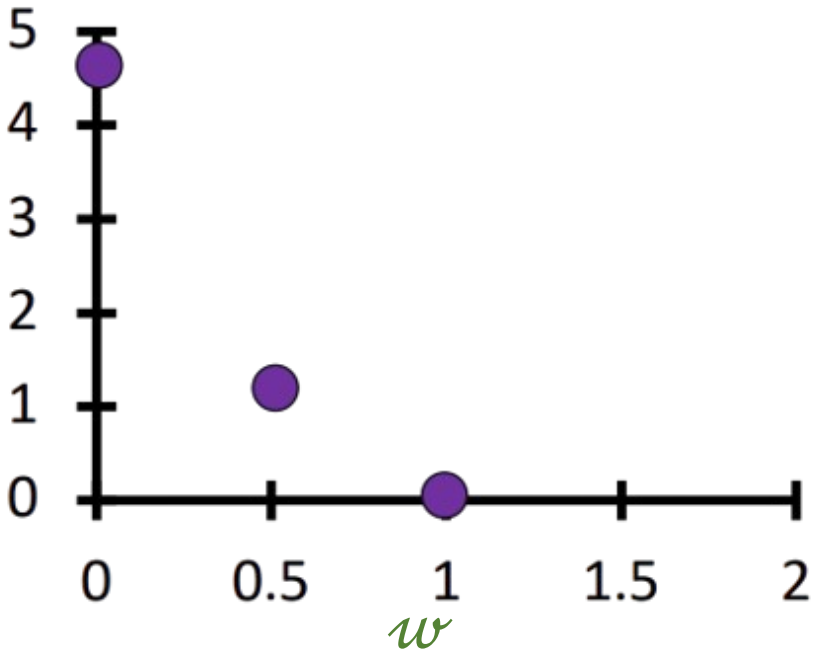
$w = 0.5$

Intuition on Minimizing MSE Loss

- Consider $x \in \mathbb{R}$ and $w \in \mathbb{R}$



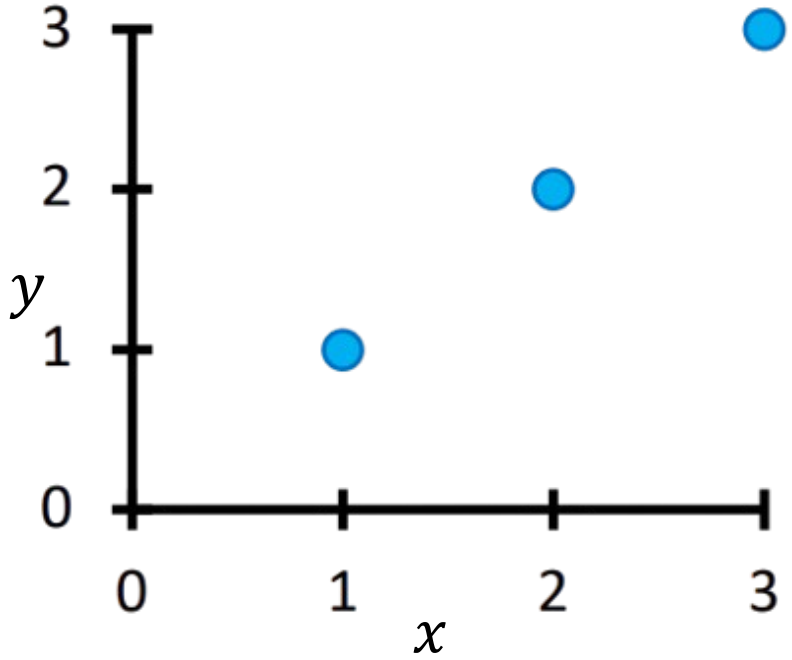
$\mathcal{J}(w, D)$

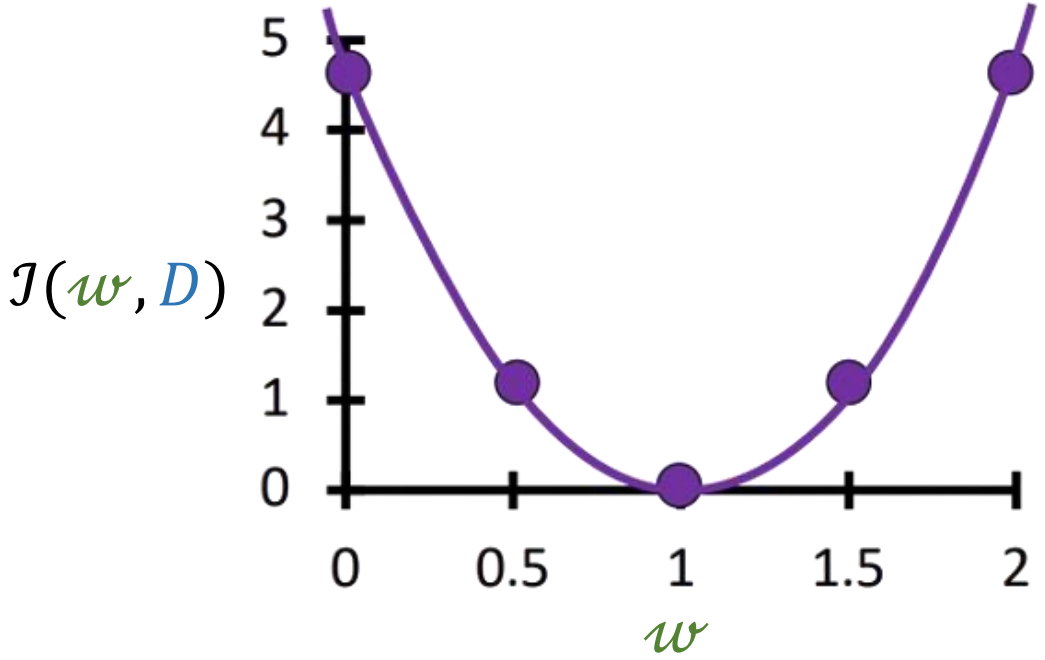


$w = 0.25$

Intuition on Minimizing MSE Loss

- Consider $x \in \mathbb{R}$ and $w \in \mathbb{R}$

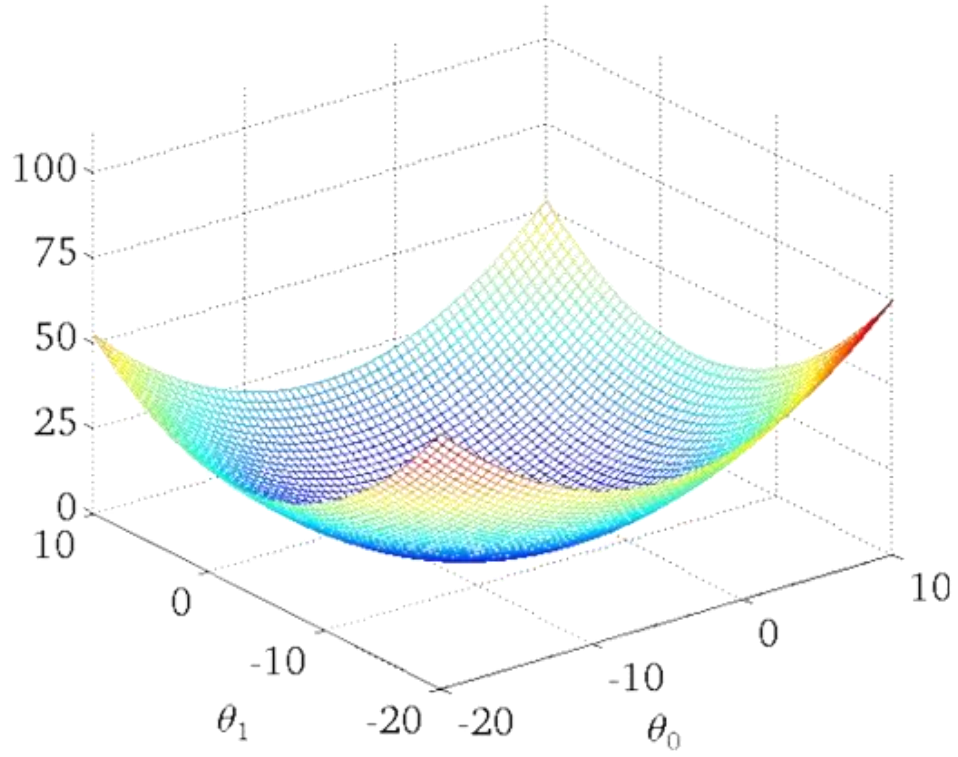


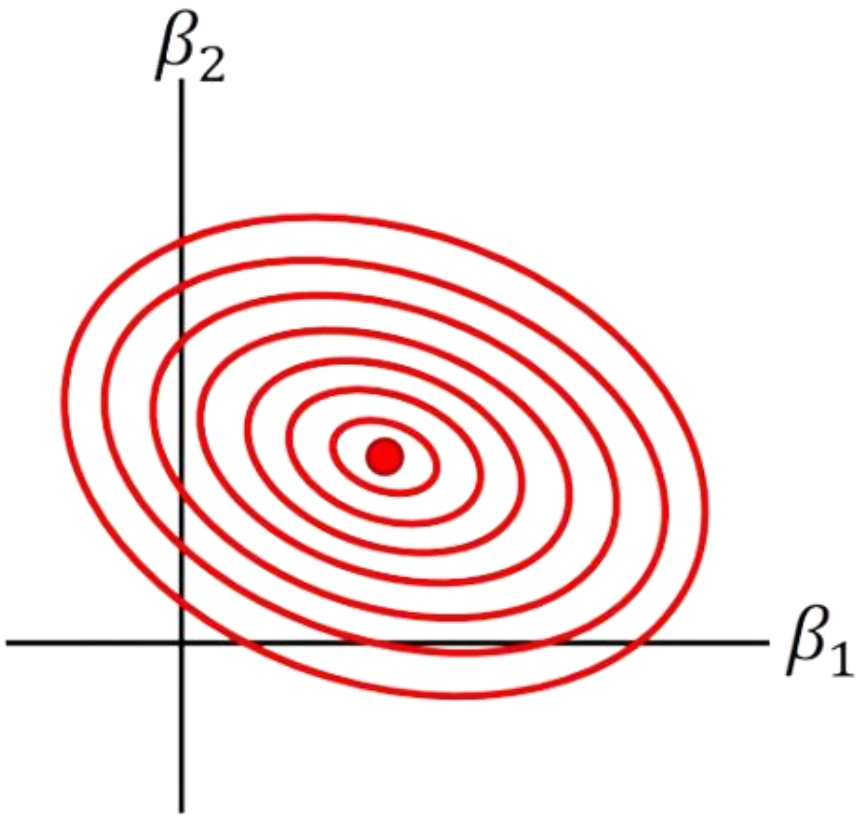


Intuition on Minimizing MSE Loss

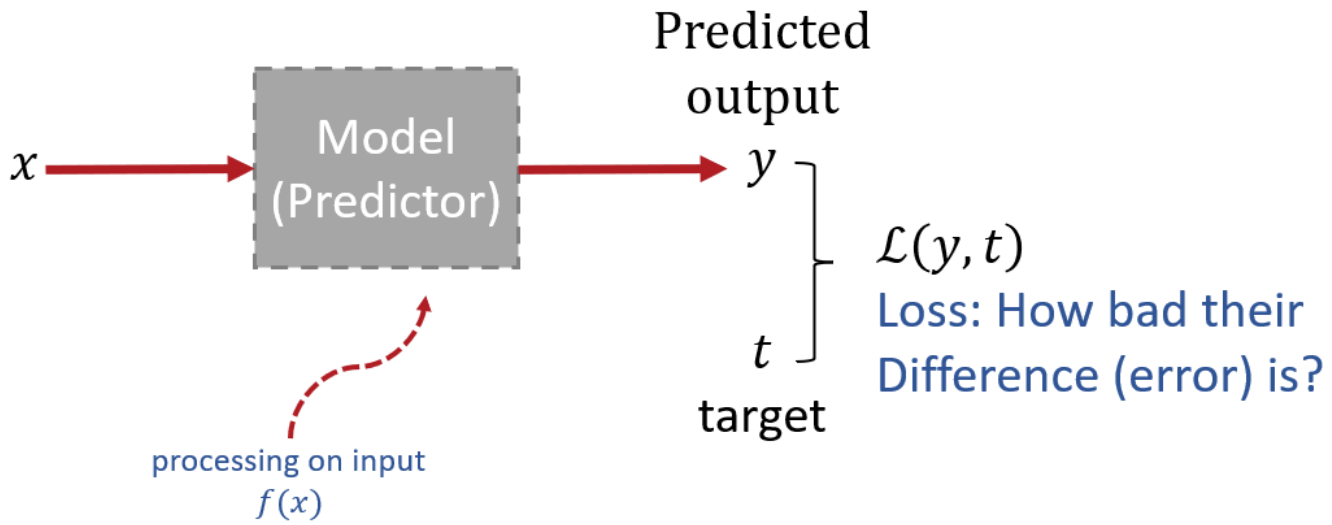
- **Convex** (“bowl shaped”) in general

$\mathcal{J}(w, D)$





Solving the Minimization Problem



- We defined a model (linear).
- We defined a loss and the cost function to be minimized.
- Q: How should we solve this minimization problem?

Solving the Minimization Problem

- Recall from your calculus class: minimum of a differentiable function (if it exists) occurs at a critical point, i.e., point where the derivative is zero.
- Multivariate generalization: set the partial derivatives to zero (or equivalently the gradient).
- We would like to find a point where the gradient is (close to) zero. How can we do it?
- Sometimes it is possible to directly find the parameters that make the gradient zero in a closed-form. We call this the direct solution.
- We may also use optimization techniques that iteratively get us closer to the solution. We will get back to this soon.

Direct Solution

- Partial derivatives: derivatives of a multivariate function with respect to (w.r.t.) one of its arguments.

$$\frac{\partial}{\partial x_1} f(x_1, x_2) = \lim_{h \to 0} \frac{f(x_1 + h, x_2) - f(x_1, x_2)}{h}$$

Direct Solution

- To compute, take the single variable derivatives, pretending the other arguments are constant.
- Example: partial derivatives of the prediction y with respect to weight w_j and bias b:

$$\frac{\partial y}{\partial w_j} = \frac{\partial}{\partial w_j}\left[\sum_{j'} w_{j'} x_{j'} + b\right] = x_j$$

$$\frac{\partial y}{\partial b} = \frac{\partial}{\partial b}\left[\sum_{j'} w_{j'} x_{j'} + b\right] = 1$$

Direct Solution

- The derivative of loss: We apply the chain rule: first we take the derivative of the loss L w.r.t. output y of the model, and then the derivative of the output y w.r.t. a parameter of the model such as w_j or b:

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{d\mathcal{L}}{dy}\frac{\partial y}{\partial w_j}$$
$$= \frac{d}{dy}\left[\frac{1}{2}(y-t)^2\right].x_j$$
$$= (y-t)x_j$$
$$\frac{\partial \mathcal{L}}{\partial b} = y - t$$

Direct Solution

- Cost derivatives (average over data points):

$$\frac{\partial \mathcal{J}}{\partial w_j} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})x_j^{(i)}$$

$$\frac{\partial \mathcal{J}}{\partial b} = \frac{1}{N}\sum_{i=1}^{N}(y^{(i)} - t^{(i)})$$

Direct Solution

- Recall that the output y is a function of the parameters as $y = \boldsymbol{w}^T \boldsymbol{x}$.
- The minimum of the cost function must occur at a point where the partial derivatives are zero, i.e.,

$$\nabla_w \mathcal{J} = 0 \Leftrightarrow \frac{\partial \mathcal{J}}{\partial w_j} = 0 \quad (\forall j), \qquad \frac{\partial \mathcal{J}}{\partial b} = 0.$$

- If $\partial \mathcal{J} / \partial w_j \neq 0$, you could reduce the cost by changing w_j.

Direct Solution

If we follow this recipe, we get that we have to set the gradient of

$\mathcal{J} = \frac{1}{2N}\|y - t\|^2$, with $\boldsymbol{y} = \boldsymbol{Xw}$ (bias absorbed in $\boldsymbol{X}$) equal to zero.

We have

$$\mathcal{J} = \frac{1}{2N}(Xw - t)^T(Xw - t),$$

So

$$\nabla_w \mathcal{J} = \frac{1}{N}X^T(Xw - t) = 0 \Longrightarrow (X^T X)w = X^T t.$$

This is a linear system of equations.

Q: What are the dimensions of each component?

Direct Solution

- Assuming that $\boldsymbol{X}^T\boldsymbol{X}$ is invertible, the optimal weights are

$$w^{LS} = (X^T X)^{-1} X^T t.$$

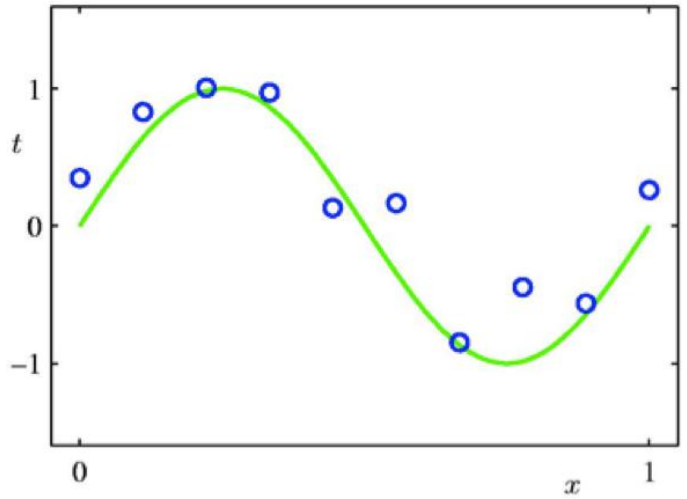
This solution is also called Ordinary Least Squares (OLS) solution.

At an arbitrary point **x**, our prediction is $y = \boldsymbol{w}^{LS^T}\boldsymbol{x}$.

Q: What happens if $\boldsymbol{X}^T\boldsymbol{X}$ is not invertible?

Basis Expansion (Feature Mapping)

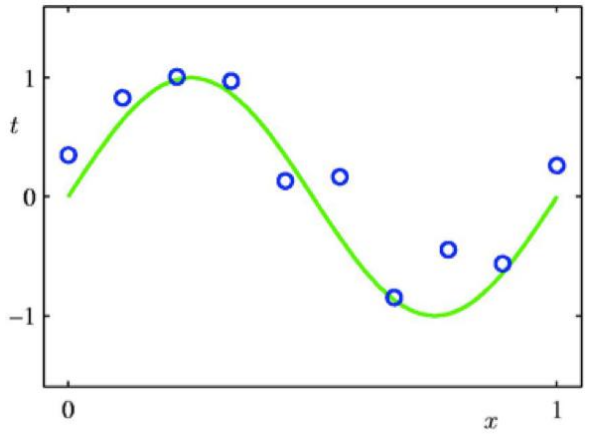
- The relation between the input and output may not be linear.



- We can still use linear regression by mapping the input feature to another space using basis expansion (or feature mapping) $\psi(\boldsymbol{x}) : \mathbb{R}^D \rightarrow \mathbb{R}^d$ and treat the mapped feature (in $\mathbb{R}^d$) as the input of a linear regression procedure.

- Let us see how it works when $x \in \mathbb{R}$and we use polynomial feature mapping.

Polynomial Feature Mapping



- Fit the data using a degree-M polynomial function of the form:

$$y = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{i=0}^{M} w_i x^i$$

- The feature mapping is $\psi(x) = [1, x, x^2, \ldots, x^M]^T$.
- We can still use the linear regression framework with least squares loss to find $\boldsymbol{w}$ since $y = \psi(x)^T w$ is linear in $w_0, w_1, \ldots$.
- In general, ψ can be any function. Another example: Fourier map

$$\psi = [1, sin(2\pi x), cos(2\pi x), sin(4\pi x), cos(4\pi x), sin(6\pi x), cos(6\pi x), \cdots]^T.$$

Polynomial Feature Mapping with $M = 0$

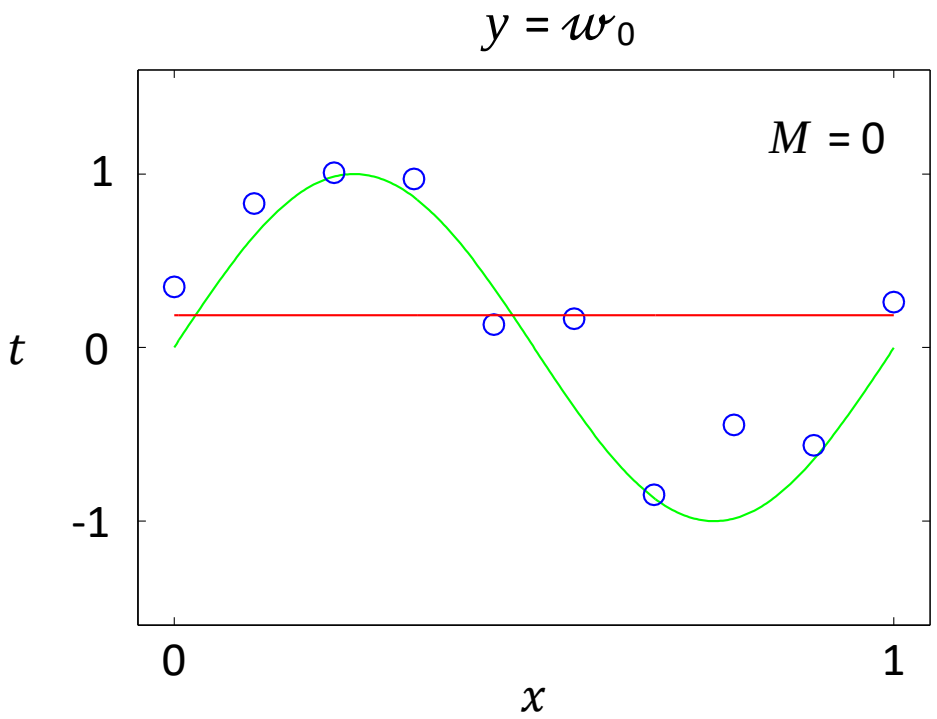


Image credit: Pattern Recognition and Machine Learning, Christopher Bishop.

Polynomial Feature Mapping with $M = 1$

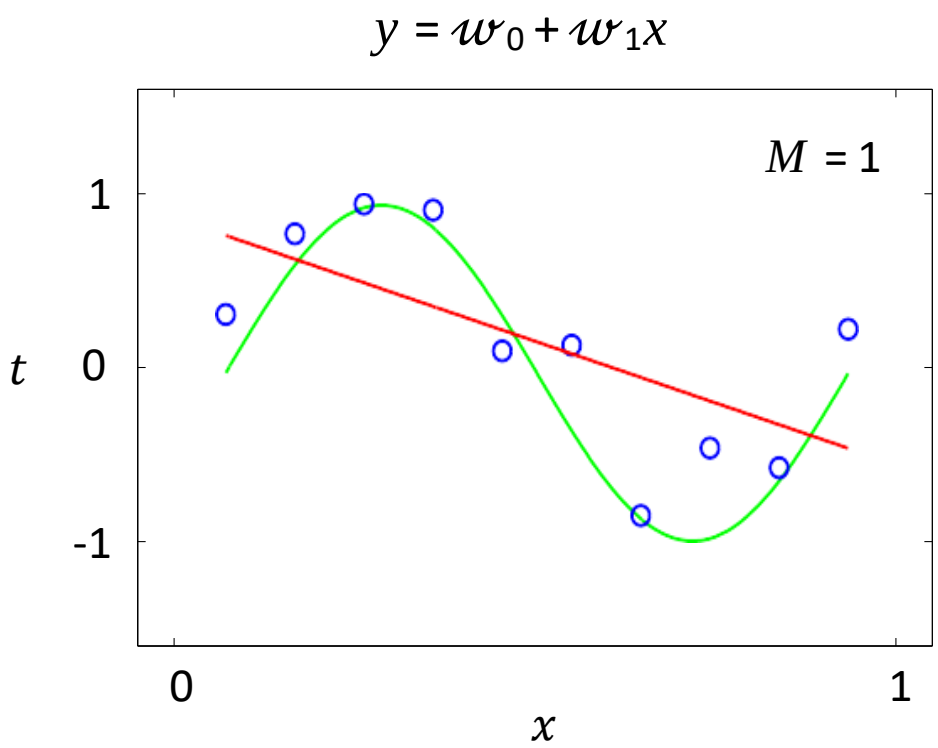


Image credit: Pattern Recognition and Machine Learning, Christopher Bishop.

Polynomial Feature Mapping with $M = 3$

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3$$

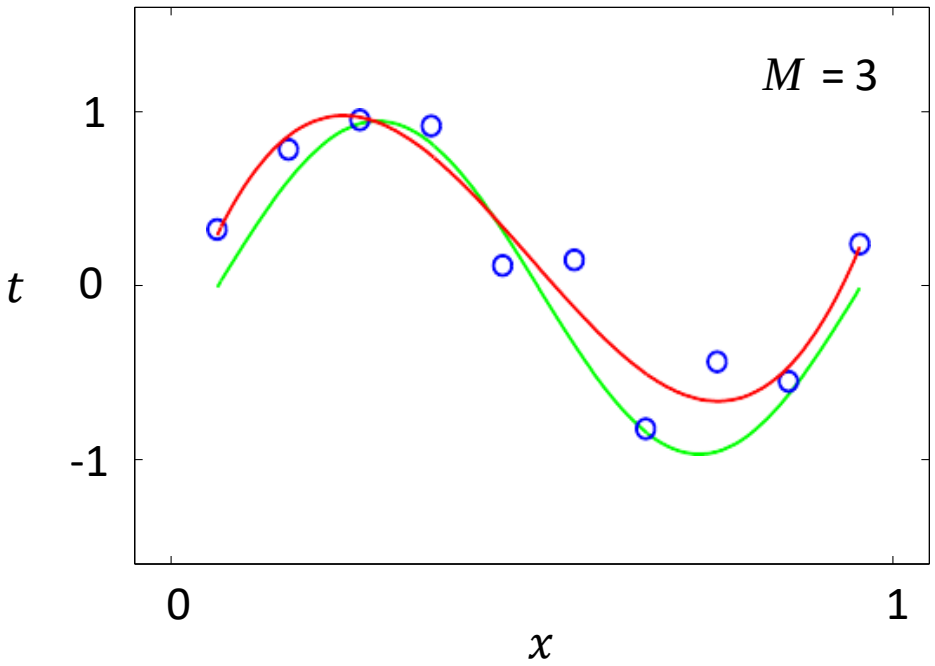


Image credit: Pattern Recognition and Machine Learning, Christopher Bishop.

Polynomial Feature Mapping with $M = 9$

$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_9 x^9$$

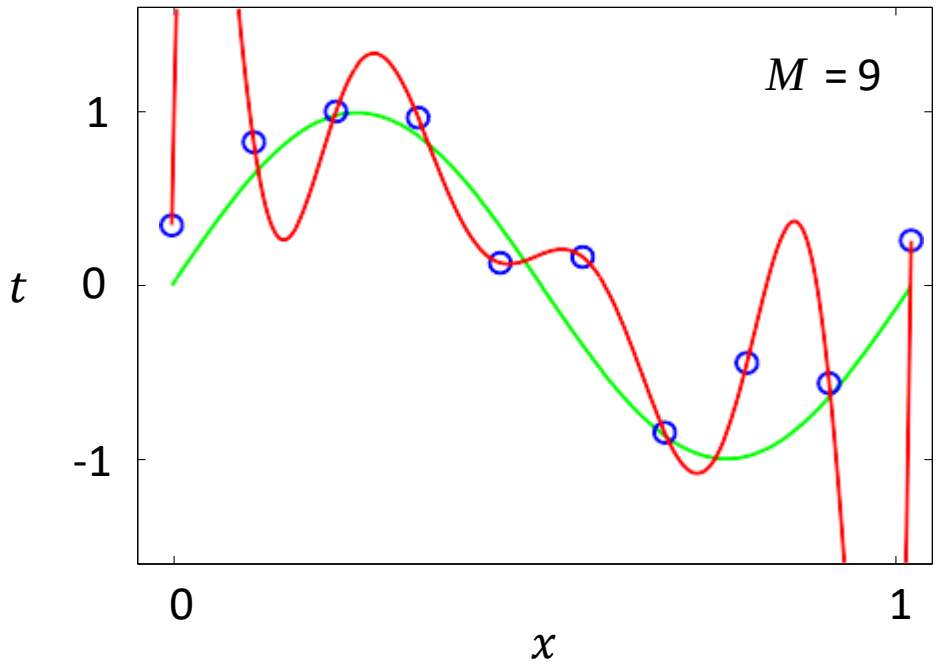
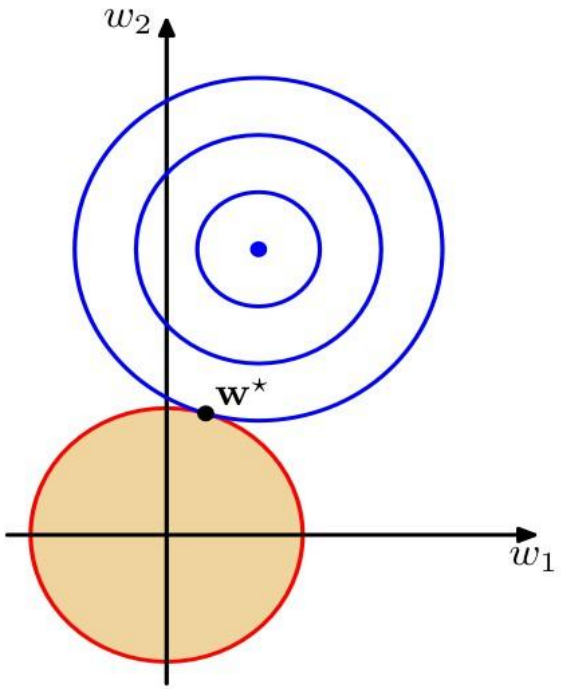


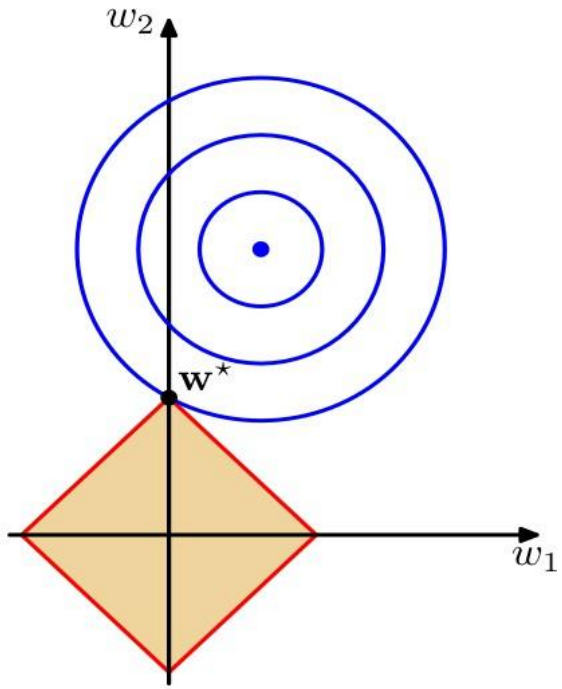
Image credit: Pattern Recognition and Machine Learning, Christopher Bishop.

Model Complexity and Regularization



L2 regularization

$$\mathcal{R} = \sum_i w_i^2$$



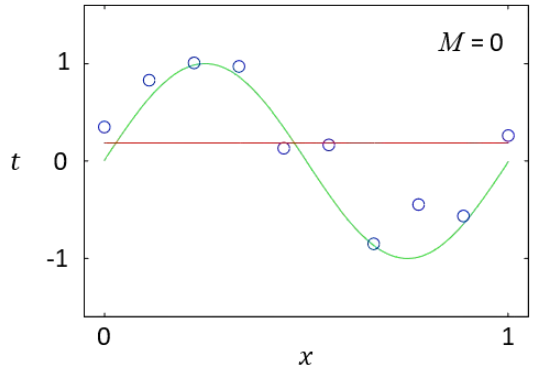
L1 regularization

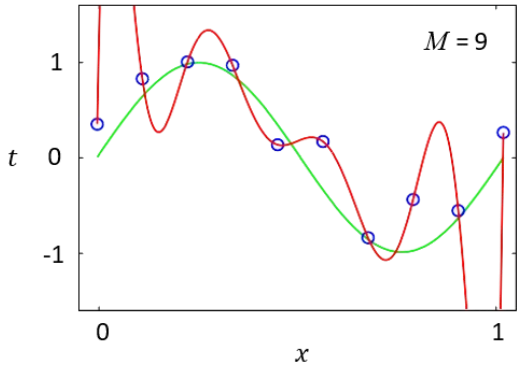
$$\mathcal{R} = \sum_i |w_i|$$

Image credit: Pattern Recognition and Machine Learning (Chapter 3), Christopher Bishop.

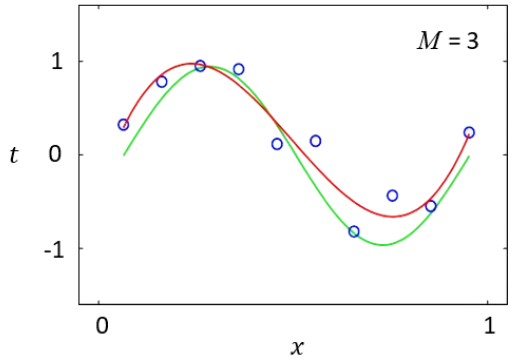
Model Complexity and Generalization

- Underfitting (M=0): model is too simple — does not fit the data.
- Overfitting (M=9): model is too complex — fits perfectly.



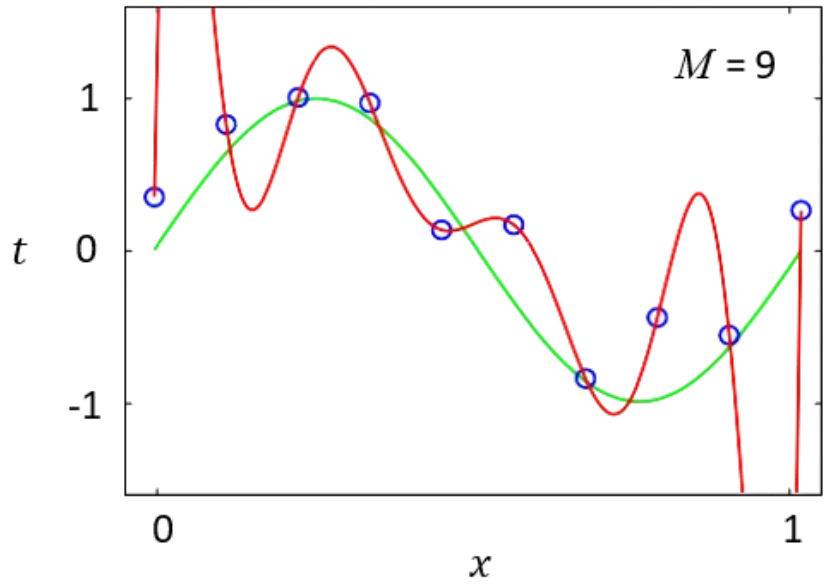


- Good model (M=3): Achieves small test error (generalizes well).



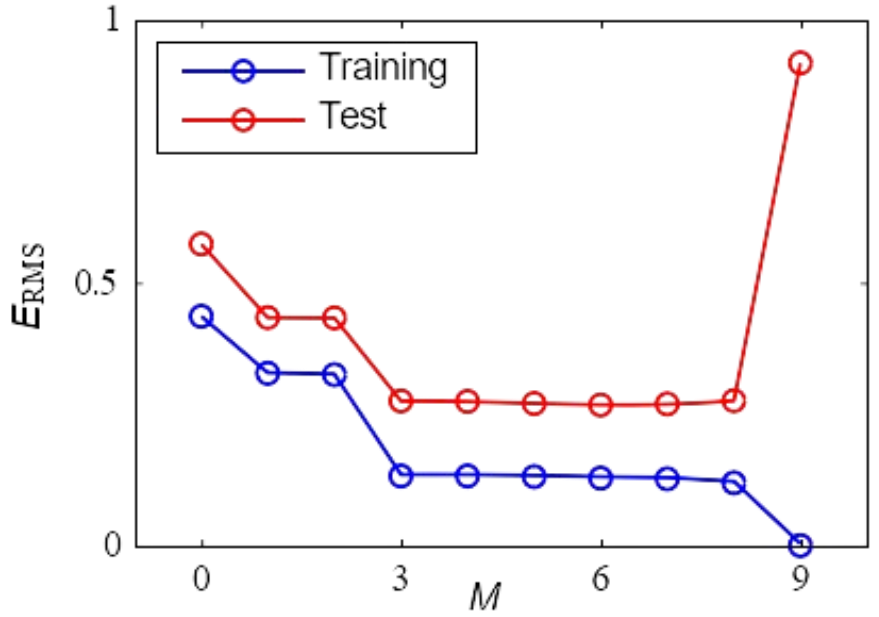
Model Complexity and Generalization

	$M=0$	$M=1$	$M=3$	$M=9$
$w_0^\star$	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



- As M increases, the magnitude of coefficients gets larger.
- For M = 9, the coefficients have become finely tuned to the data.
- Between data points, the function exhibits large oscillations.

Model Complexity and Generalization

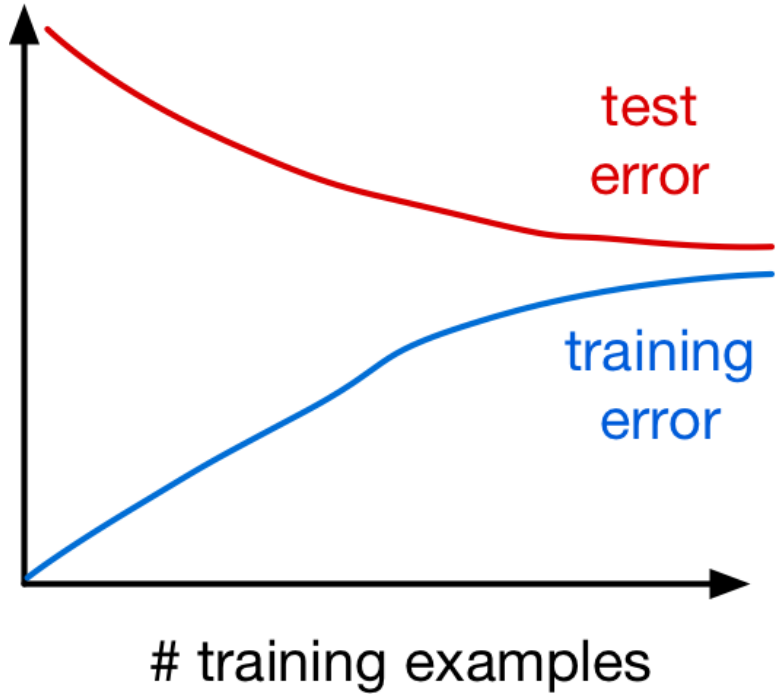


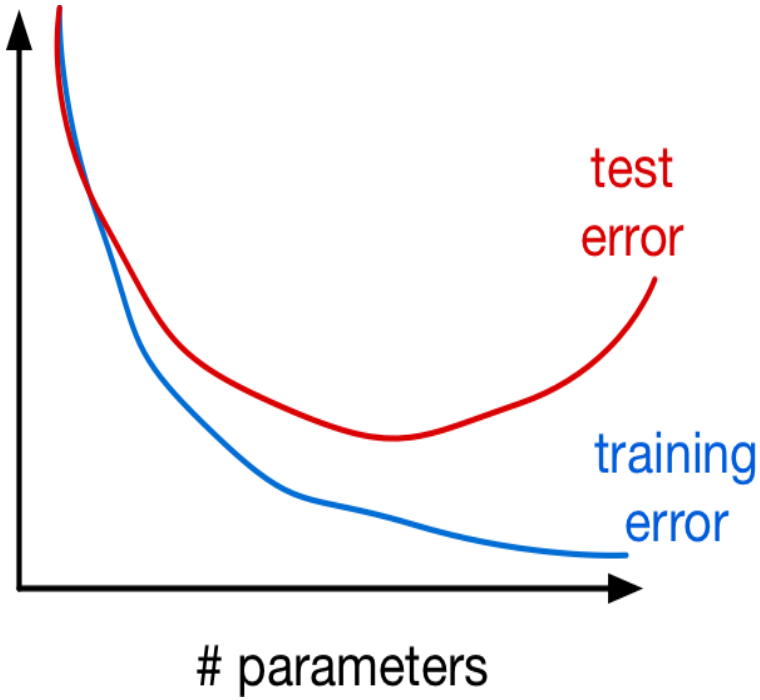
As the degree M of the polynomial increases

- the training errors decreases;
- the test error, however, initially decreases, but then increases.

Model Complexity and Generalization

- Training and test error as a function of # training examples and parameters:





Regularization for Controlling the Model Complexity

- The degree of the polynomial M controls the complexity of the model.
- The value of M is a hyperparameter for polynomial expansion, just like K in KNN or the depth of a tree in a decision tree. We can tune it using a validation set.
- Restricting the number of parameters of a model (M here) is a crude approach to control the complexity of the model.
- A better solution: keep the number of parameters of the model large, but enforce "simpler" solutions within the same space of parameters.
- This is done through regularization or penalization.
  - Regularizer (or penalty): a function that quantifies how much we prefer one hypothesis vs. another, prior to seeing the data.
- Q: How?!

ℓ_2 or (L^2) Regularization

- We can encourage the weights to be small by choosing the ℓ_2 (or L^2) of the weights as our regularizer or penalty:

$$\mathcal{R} = \frac{1}{2}\|w\|_2^2 = \frac{1}{2}\sum_j w_j^2.$$

  - Note: To be precise, we are regularizing the *squared* l_2 norm.

- The regularized cost function makes a tradeoff between fit to the data and the norm of the weights:

$$\mathcal{J}_{reg}(w) = \mathcal{J}(w) + \lambda\mathcal{R}(w) = \mathcal{J}(w) + \frac{\lambda}{2}\sum_j w_j^2.$$

ℓ_2 or (L^2) Regularization

- The regularized cost function

$$\mathcal{J}_{reg}(w) = \mathcal{J}(w) + \lambda\mathcal{R}(w) = \mathcal{J}(w) + \frac{\lambda}{2}\sum_j w_j^2.$$

- The basic idea is that "simpler" functions have weights $\boldsymbol{w}$ with smaller ℓ_2-norm and we prefer them to functions with larger ℓ_2-norm.
  - Intuition: Large weights makes the function f have more abrupt changes as a function of the input **x**; it will be less smooth.
- If you fit training data poorly, $\mathcal{J}$ is large. If the fitted weights have high values, $\mathcal{R}$ is large.
- Large λ penalizes weight values more.
- Here λ is a hyperparameter that we can tune with a validation set.

ℓ_2 Regularized Least Squares: Ridge Regression

For the least squares problem, we have $\mathcal{J}(w) = \frac{1}{2N}\|Xw - t\|^2$.

- When $\lambda > 0$ (with regularization), regularized cost gives

$$w_\lambda^{Ridge} = \underset{w}{\operatorname{argmin}}\, \mathcal{J}_{reg}(w) = \underset{w}{\operatorname{argmin}}\, \frac{1}{2N}\|Xw - t\|_2^2 + \frac{\lambda}{2}\|w\|_2^2$$
$$= (X^T X + \lambda N \mathbf{I})^{-1} X^T t.$$

- The case of $\lambda = 0$ (no regularization) reduces to the least squares solution!
- Q: What happens when $\lambda \to \infty$?
- Note that it is also common to formulate this problem as

$\underset{w}{\operatorname{argmin}}\|Xw - t\|_2^2 + \frac{\lambda}{2}\|w\|_2^2$ in which case the solution is

$$w_\lambda^{Ridge} = (X^T X + \lambda \mathbf{I})^{-1} X^T t.$$

Lasso and the ℓ_1 Regularization

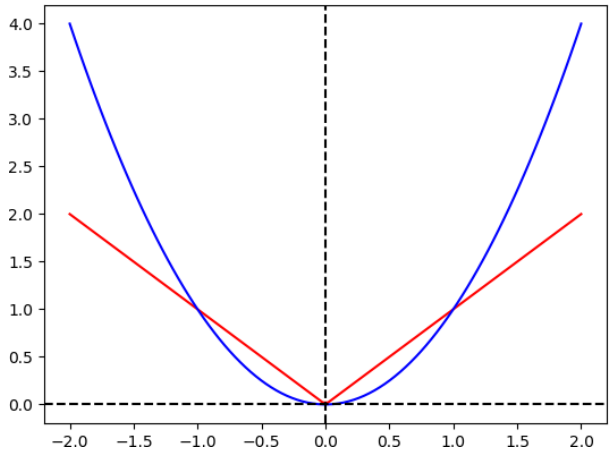
- The ℓ_1 norm, or sum of absolute values, is another regularizer:

$$\mathcal{R}(w) = \|w\|_1 = \sum_j |w_j|$$

- The Lasso (Least Absolute Shrinkage and Selection Operator) is

$$\min_w \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}\|_2^2 + \lambda\|\boldsymbol{w}\|_1 .$$

- It can be shown that Lasso encourages weights to be exactly zero.
  - Q: When is this helpful?



Ridge vs. Lasso – Geometric Viewpoint

- We presented regularization as a penalty on the weights, in which we solve

$$\min_{w} \mathcal{J}(w) + \lambda \mathcal{R}(w)$$

- We can also write an equivalent form as a constraint optimization:

$$\underset{w}{\operatorname{argmin}}\, \mathcal{J}(w)$$

$$s.t.\ \ \mathcal{R}(w) \leq \mu$$

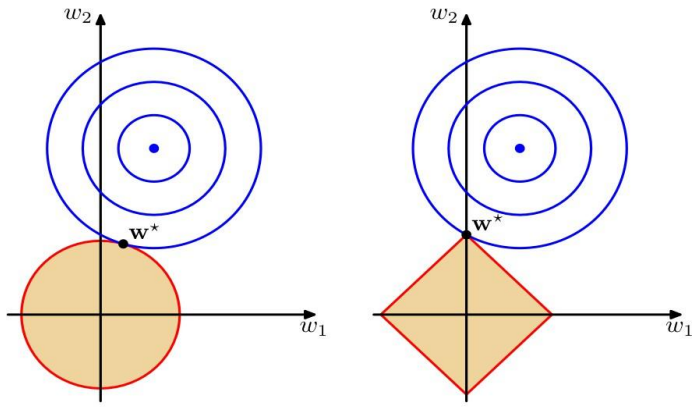
  for a corresponding value of μ

- The Ridge regression and the Lasso can then be written as

$$\underset{w}{\operatorname{argmin}} \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}\|_2^2$$

$$s.t.\ \ \|\boldsymbol{w}\|_p \leq \mu \qquad (Lasso:\ p = 1; Ridge: p = 2)$$

Ridge vs. Lasso – Geometric Viewpoint



L2 regularization

$$\mathcal{R} = \sum_i w_i^2$$

L1 regularization

$$\mathcal{R} = \sum_i |w_i|$$

- The set $\{\boldsymbol{w}: \|\boldsymbol{X}\boldsymbol{w} - \boldsymbol{t}\|_2^2 \leq \varepsilon\}$ defines ellipsoids of ε cost in the weights space.
- The set $\{\boldsymbol{w}: \|\boldsymbol{w}\|_p \leq \mu\}$ defines the constraint on weights defined by the regularizer.
- The solution would be the smallest ε for which these two sets intersects.
- For p = 1, the diamond-shaped constraint set has corners. When the intersection happens at a corner, some of the weights are zero.
- For p = 2, the disk-shaped constraint set does not have corners. It does not induce any zero weights.

Alternative loss function

- **Mean absolute error** $\frac{1}{n}\sum_{i=1}^{n}|y_i - t_i|$
- **Mean relative error** $\frac{1}{n}\sum_{i=1}^{n}\frac{|y_i - t_i|}{|t_i|}$
- $\boldsymbol{R^2}$ **score:** $1 - \frac{MSE}{Variance}$
  - "Coefficient of determination"
  - Higher is better, $R^2 = 1$ is perfect

Any questions?