LMP 1210H: Basic Principles of Machine Learning in Biomedical Research

Lecture 3: Linear models

Sana Tonekaboni

January 25, 2024

Quick check in…

- Assignment 1 is due next week.
    - Start early!
    - Ask questions!
- Start thinking about your final project.
    - Form groups of 2-3
    - Think about interesting research ideas and look for datasets online
    - More details to come next week!

Recap

- So far, we've talked about algorithms/procedures for learning: KNN, decision trees
- For the remainder of this course, we'll take a more modular approach:
  - choose a **model** describing the relationships between variables of interest
  - define a **loss function** quantifying how bad is the fit to the data
  - choose a **regularizer** saying how much we prefer different candidate explanations
  - fit the model that minimizes the loss function and satisfies the constrain imposed by the regularizer, possibly using an **optimization algorithm**
- By mixing and matching these modular components, your ML skills become combinatorially more powerful!

Linear models

Problem setup

Recall that in supervised learning:

- There is target $t \in \mathcal{T}$ (also called response, outcome, output, class)
- There are features $x \in \mathcal{X}$ (also called inputs or covariates)
- The objective is to learn a function $f : \mathcal{X} \to \mathcal{T}$ such that: $t \approx y = f(x)$
- based on some data $\mathcal{D} = \{(x^{(i)}, t^{(i)})$ for $i = 1,2,\ldots,N\}$

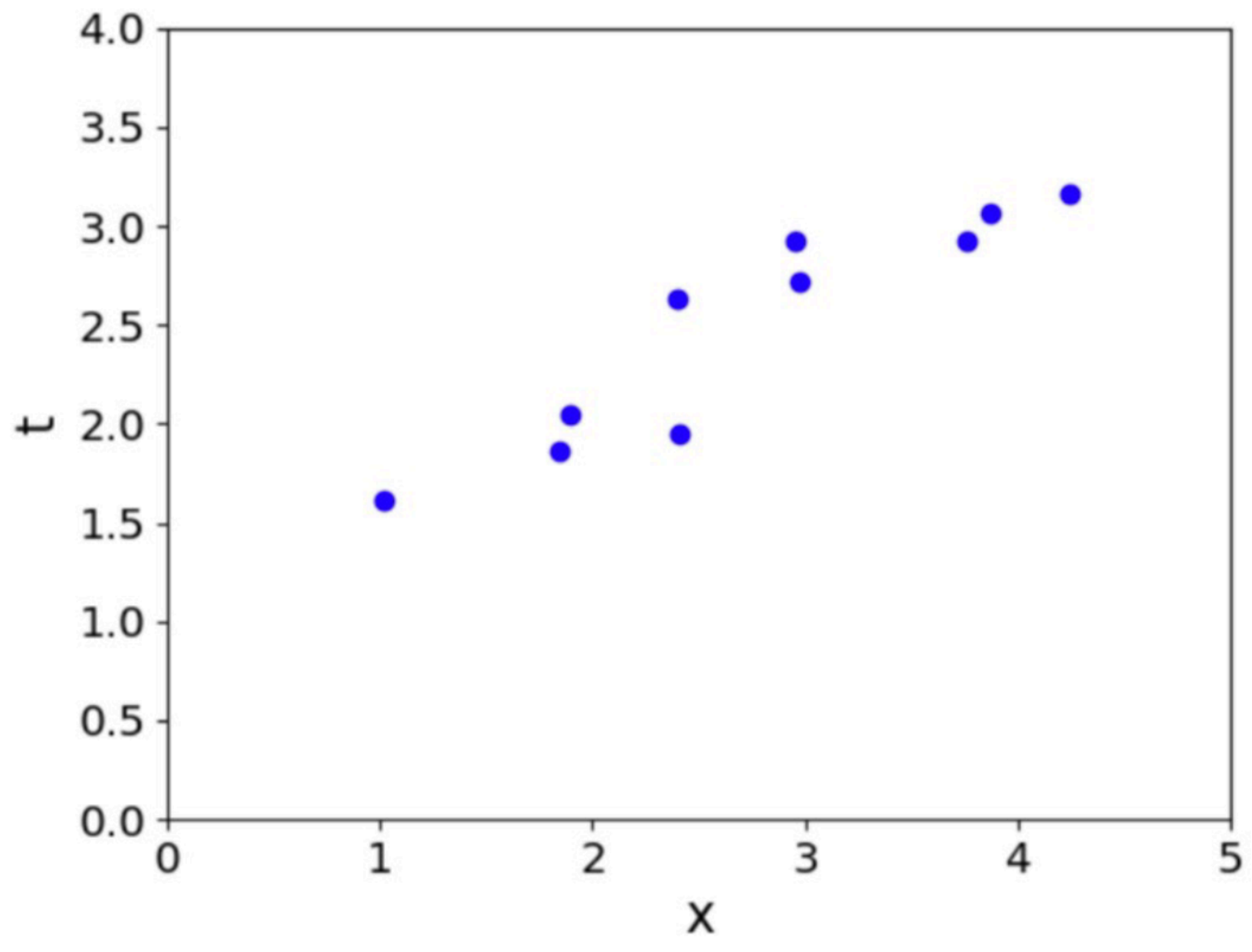
Linear models

Linear regression

In linear regression, we use a linear function of the inputs to make prediction of the target:

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

- y is the prediction
- w is the weights
- b is the bias (or intercept) — don't confuse it with bias/variance that comes later
- w, b together are the parameters
- Our goal is to make predictions that are as close to the target $y \approx t$

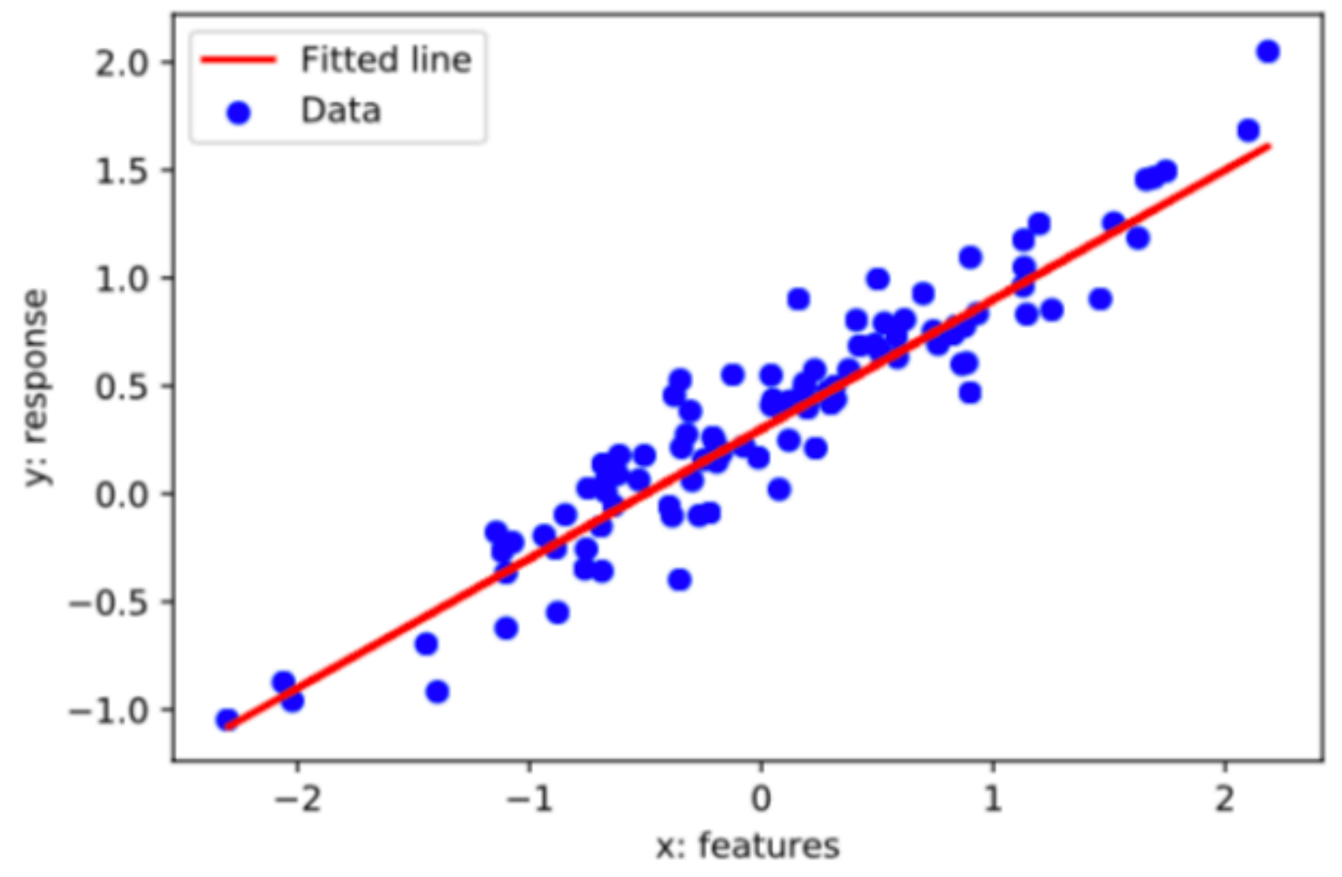


Linear models

Linear regression

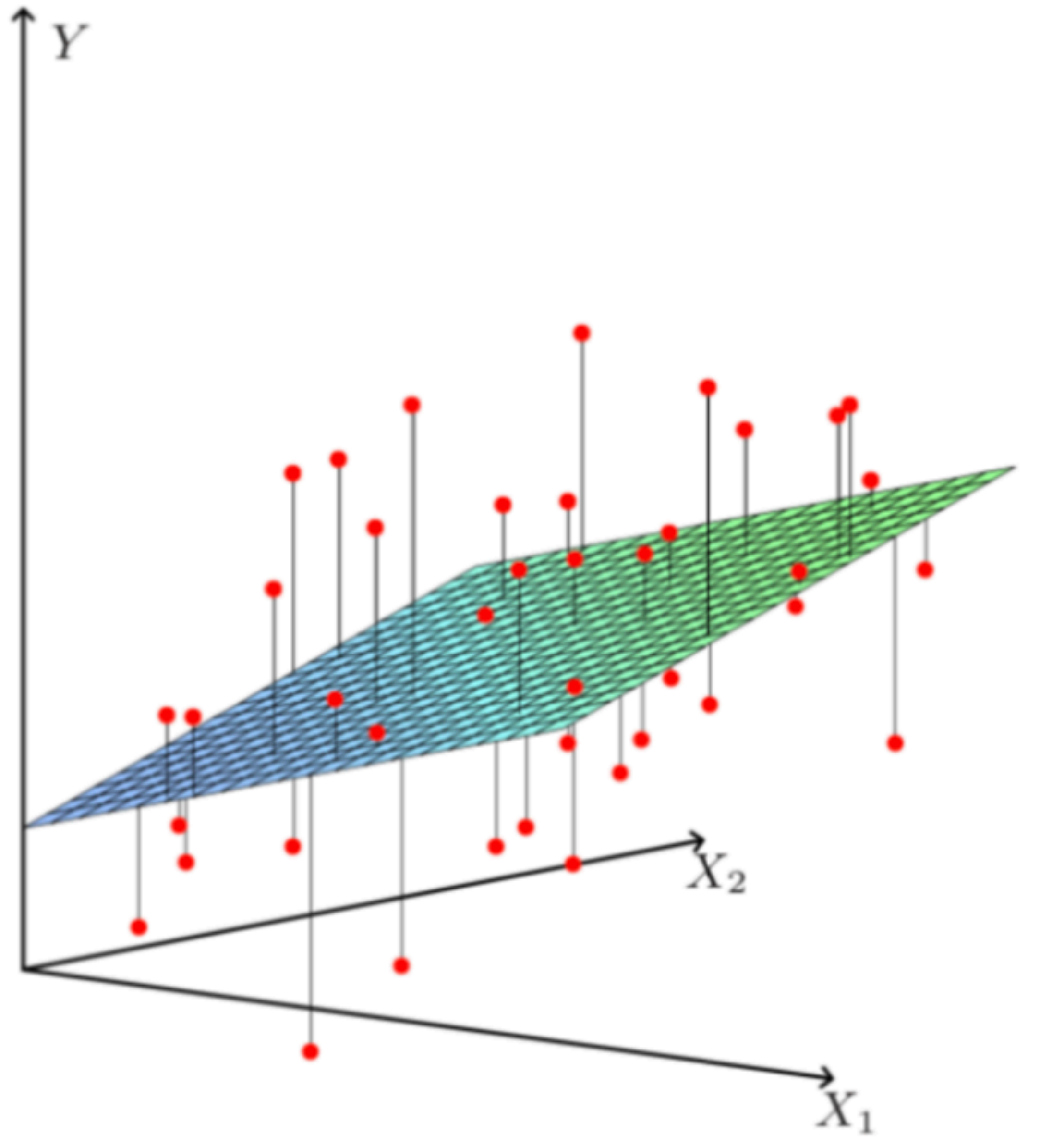
If we have only 1 feature:

- $y = wx + b$ where $w, x, b \in \mathbb{R}$
- y is linear in x



If we have D feature:

- $y = \mathbf{w}^T\mathbf{x} + b$ where $\mathbf{w}, \mathbf{x} \in \mathbb{R}^D, d \in \mathbb{R}$
- y is linear in x



Relationship between input and output is linear in both cases!

Linear models

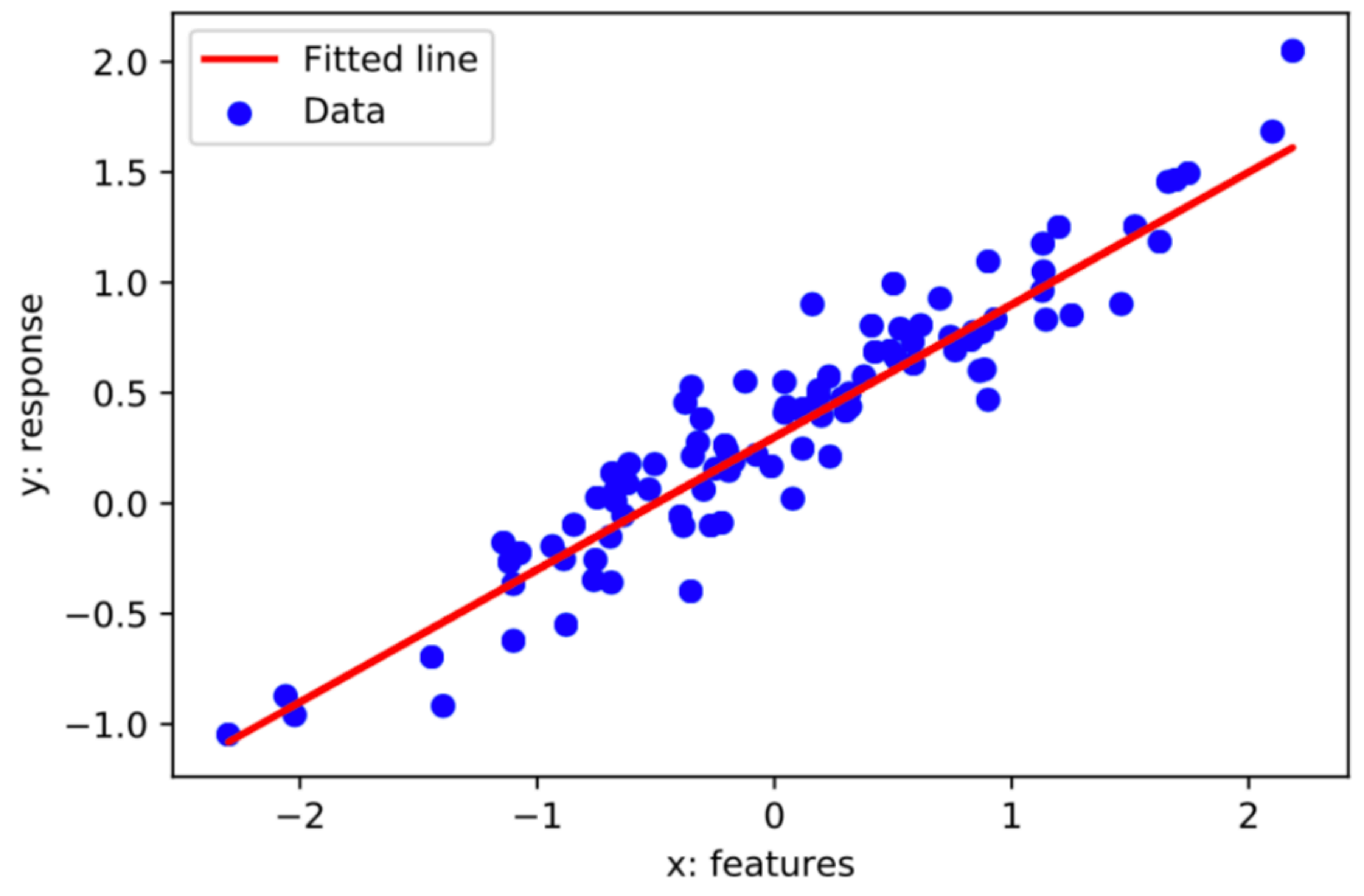
Linear regression

We have a dataset $\mathscr{D} = \{(\mathbf{x}^i, t^{(i)})$ for $i = 1,2,\ldots,N\}$ where:

$\mathbf{x}^{(i)} = (x_1^{(i)}, x_2^{(i)}, \ldots, x_D^{(i)})^T \in \mathbb{R}^D$ are the inputs, e.g. age, education, ….

$t^{(i)} \in \mathbb{R}$ is the target or response, e.g. income.

Predict $t^{(i)}$ with a linear function of $\mathbf{x}^{(i)}$.



Find the best line that minimizes error on sum of all errors!

Linear models

Quantify the quality of fit

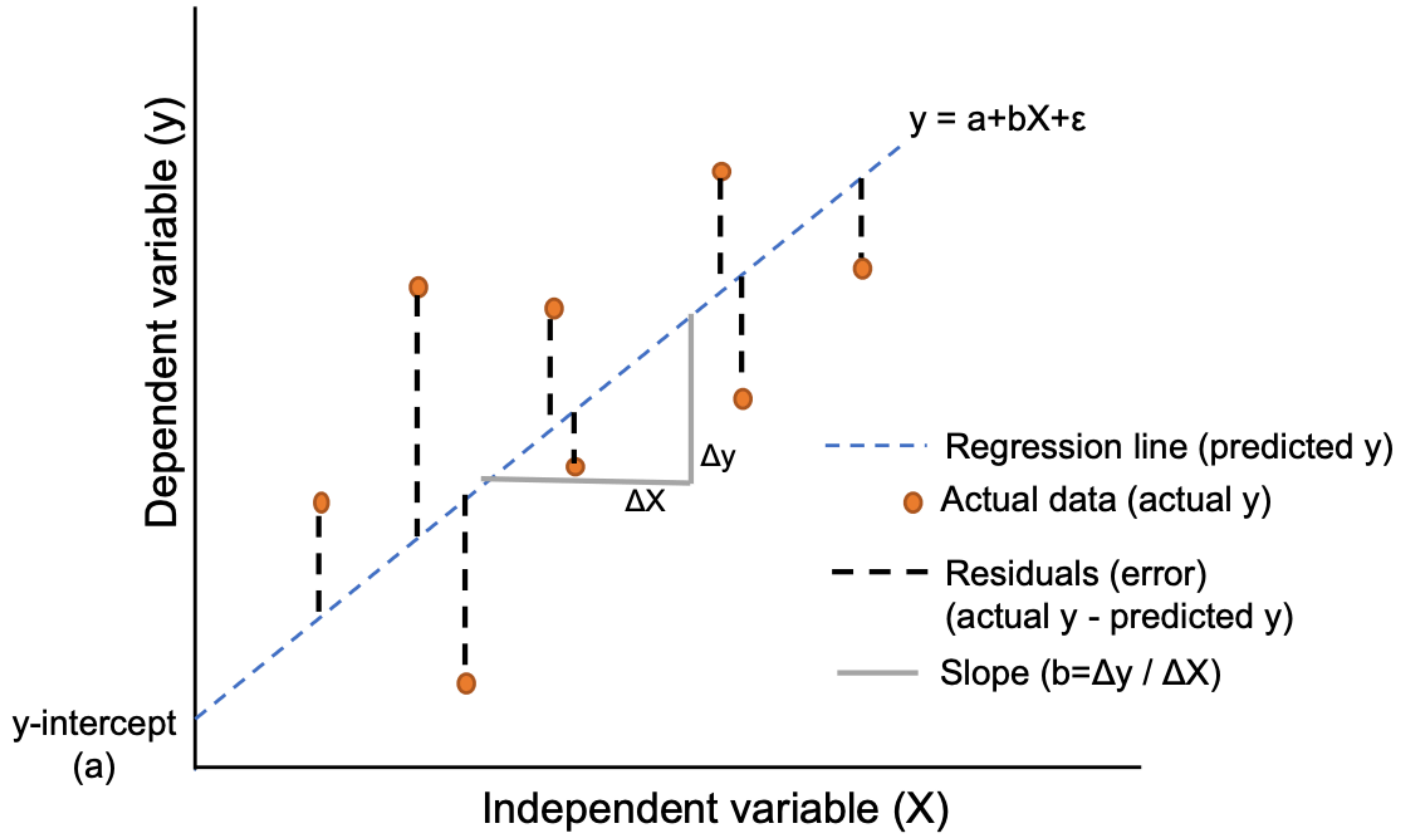
Loss function $\mathscr{L}(y, t)$ measures how bad it is if a model predicts y for a sample with label t

Examples:

Squared error: $\mathscr{L}(y, t) = (y - t)^2$

Absolute error: $\mathscr{L}(y, t) = |y - t|$

$y - t$ is the **residual**, and we want to make this small in magnitude



Note: There are many different loss functions that can be used and they each have different behaviours.

Linear models

Cost function vs. loss function

Cost function: Loss function averaged over all training samples.

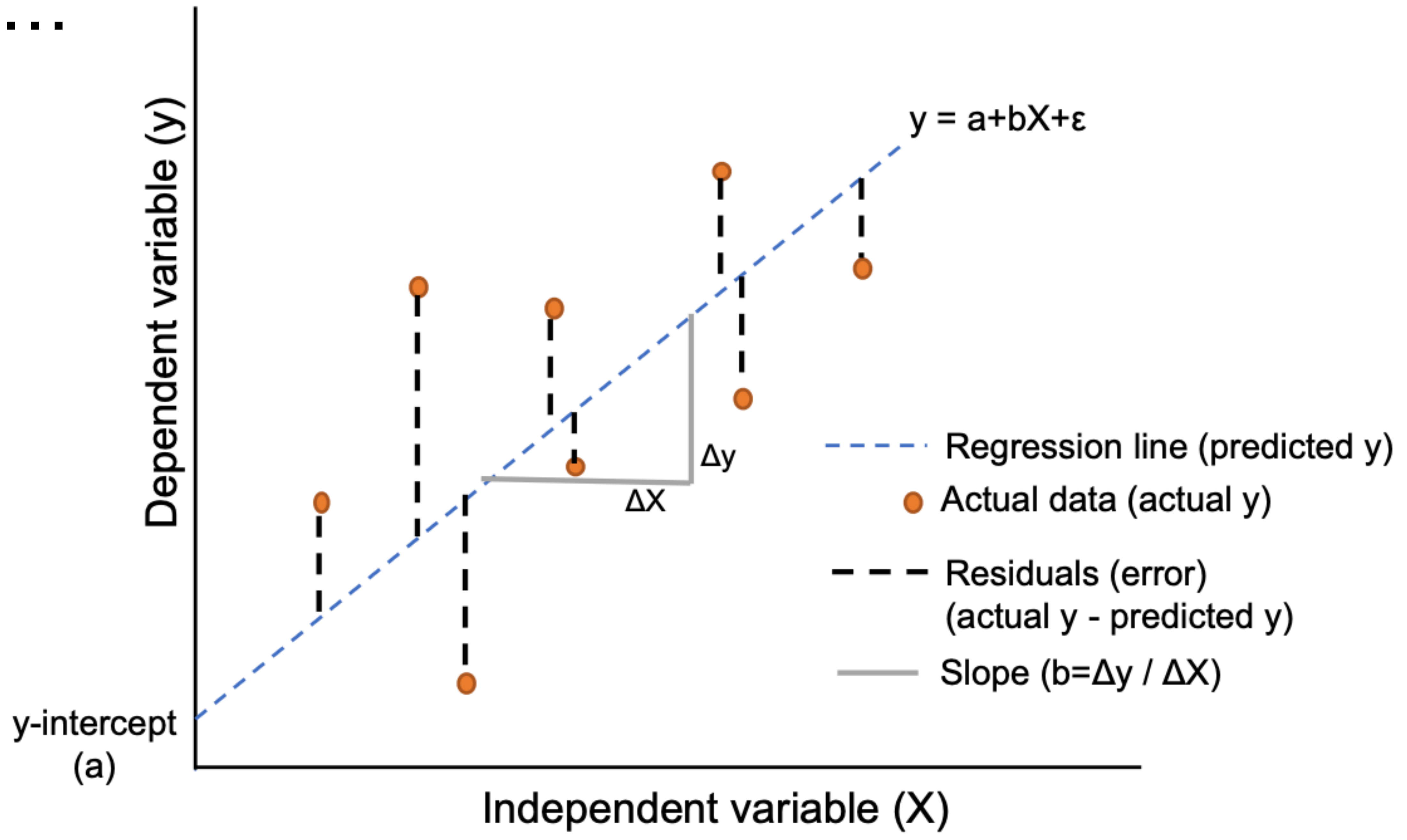
It is also referred to Empirical loss, average loss, …

The terminology is not universal .

$$\mathcal{L}(y,t) = \frac{1}{2N}\sum_{i=0}^{N}(y^{(i)} - t^{(i)})^2$$

$\frac{1}{2}$ is for computational convenience.

You will see later!



Linear models

Vectorization

We organize all training samples as a **matrix** where each row represents one training sample.

We organize all training targets as a **Vector**, with each sample as one dimension

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)T} \\ \mathbf{x}^{(2)T} \\ \mathbf{x}^{(3)T} \end{bmatrix} = \begin{array}{cc} \begin{array}{cccc} & & \text{one feature across all training samples} & \end{array} & \\ \begin{bmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 1 \end{bmatrix} & \begin{array}{l} \\ \text{one training sample} \\ \\ \end{array} \end{array} \qquad \mathbf{y} = \begin{bmatrix} 0.2 \\ 4 \\ 0 \end{bmatrix}$$

Linear models

Vectorization

We can compute the prediction for the whole dataset by matrix multiplication $\mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix} = \begin{bmatrix} 0.2 \\ 1 \\ 0.5 \\ 1 \end{bmatrix}$

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

We can compute the squared error loss on all samples as: $\mathcal{J} = \frac{1}{2N} \|\mathbf{y} - \mathbf{t}\|^2$

Linear models

Vectorization

We can also add a column of 1s to the data matrix, and combine b with $\mathbf{w}$. How?

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}^{(1)T} \\ \mathbf{x}^{(2)T} \\ \mathbf{x}^{(3)T} \end{bmatrix} = \begin{bmatrix} 8 & 0 & 3 & 0 \\ 6 & -1 & 5 & 3 \\ 2 & 5 & -2 & 1 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

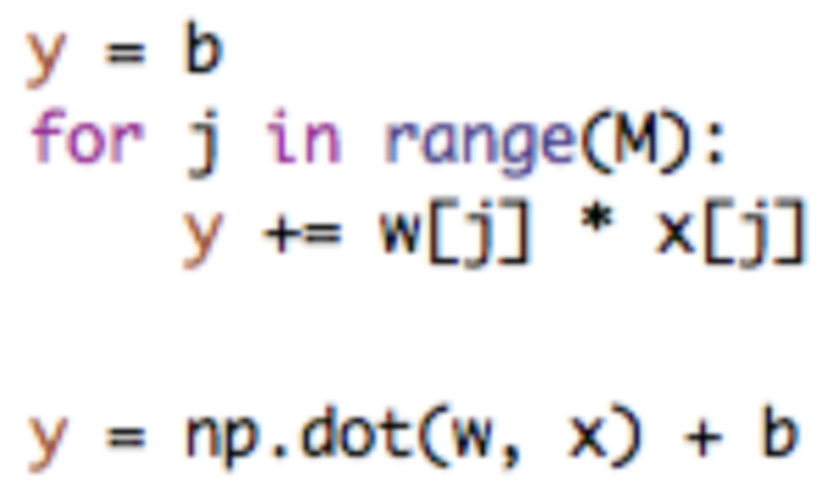
$$\mathbf{X} = \begin{bmatrix} 1 & 8 & 0 & 3 & 0 \\ 1 & 6 & -1 & 5 & 3 \\ 1 & 2 & 5 & -2 & 1 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} b \\ w_1 \\ w_2 \\ w_3 \\ w_4 \end{bmatrix}$$

Linear models

Vectorization

Why Vectorization?

- Because for loops are very slow in Python!
- The equations, and the code, will be simpler and more readable.
- Gets rid of dummy variables/indices!
- Vectorized code is much faster
  - Cut down on Python interpreter overhead
  - Use highly optimized linear algebra libraries
  - Matrix multiplication is very fast on a Graphics Processing Unit (GPU)



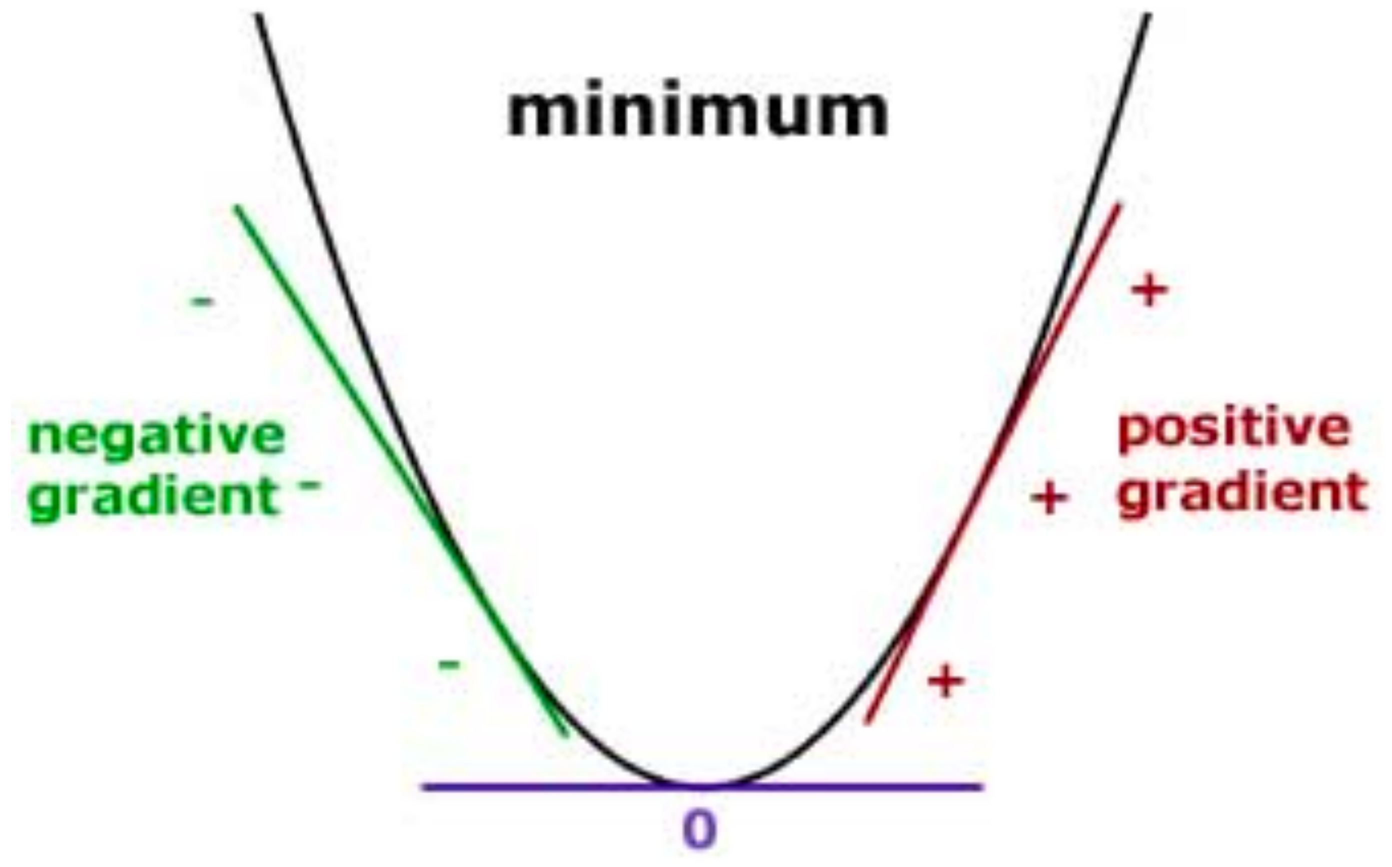
```
y = b
for j in range(M):
    y += w[j] * x[j]

y = np.dot(w, x) + b
```

Linear models

Optimization

- We defined a cost function that we'd like to minimize.
- Recall from calculus class: minimum of a smooth function (if it exists) occurs at a **critical point**, i.e. point where the derivative is zero.
- Multivariate generalization: partial derivatives must be zero.



- We would like to find a point where the gradient is (close to) zero. How?
  - Sometimes it is possible to directly find the parameters that make a gradient zero in a closed-form. We call this **direct solution**.
  - We may also use optimization techniques that iteratively get us closer to the solution.

Linear models

Optimization

- **Partial derivatives**: derivatives of a multivariate function with respect to one of its arguments.

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

- To compute, take the single variable derivatives, pretending the other arguments are constant.
- Example: partial derivatives of the prediction y

$$\begin{aligned}\frac{\partial y}{\partial w_j} &= \frac{\partial}{\partial w_j}\left[\sum_{j'} w_{j'} x_{j'} + b\right] \\ &= x_j \\ \frac{\partial y}{\partial b} &= \frac{\partial}{\partial b}\left[\sum_{j'} w_{j'} x_{j'} + b\right] \\ &= 1\end{aligned}$$

Linear models

Optimization

- **Chain rule for derivatives**

$$y = \sum_j w_j x_j + b$$

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

$$\frac{\partial \mathcal{L}}{\partial w_j} = \frac{\mathrm{d}\mathcal{L}}{\mathrm{d}y}\frac{\partial y}{\partial w_j}$$

$$= \frac{\mathrm{d}}{\mathrm{d}y}\left[\frac{1}{2}(y-t)^2\right] \cdot x_j$$

$$= (y-t)x_j$$

$$\frac{\partial \mathcal{L}}{\partial b} = y - t$$

- Cost derivatives (Averaged over all samples)

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

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

Remember the $\frac{1}{2}$ that was for computational convenience!

Linear models

Optimization

The minimum must occur at a point where the partial derivatives are zero.

$$\frac{\partial \mathcal{J}}{\partial w_j} = 0 \qquad \frac{\partial \mathcal{J}}{\partial b} = 0$$

If $\partial \mathcal{J} / \partial w_j \neq 0$ you could reduce the cost by changing w_j. This turns out to give a system of linear equations, which we can solve efficiently. Full derivation in the readings.

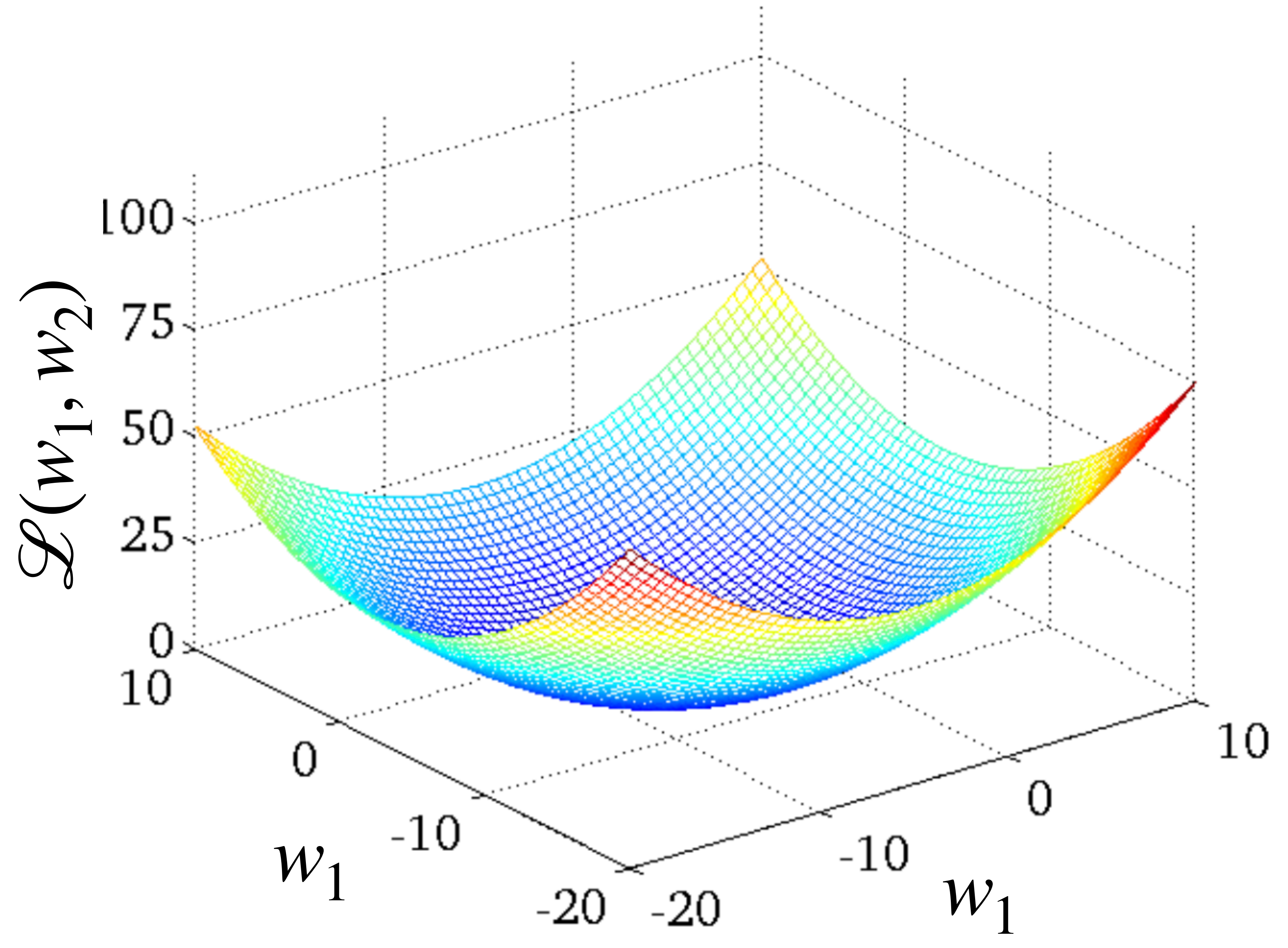
Optimal weights: $(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T t$

Note: Linear regression is one of only a handful of models in this course that permit direct solution.

Gradient descend

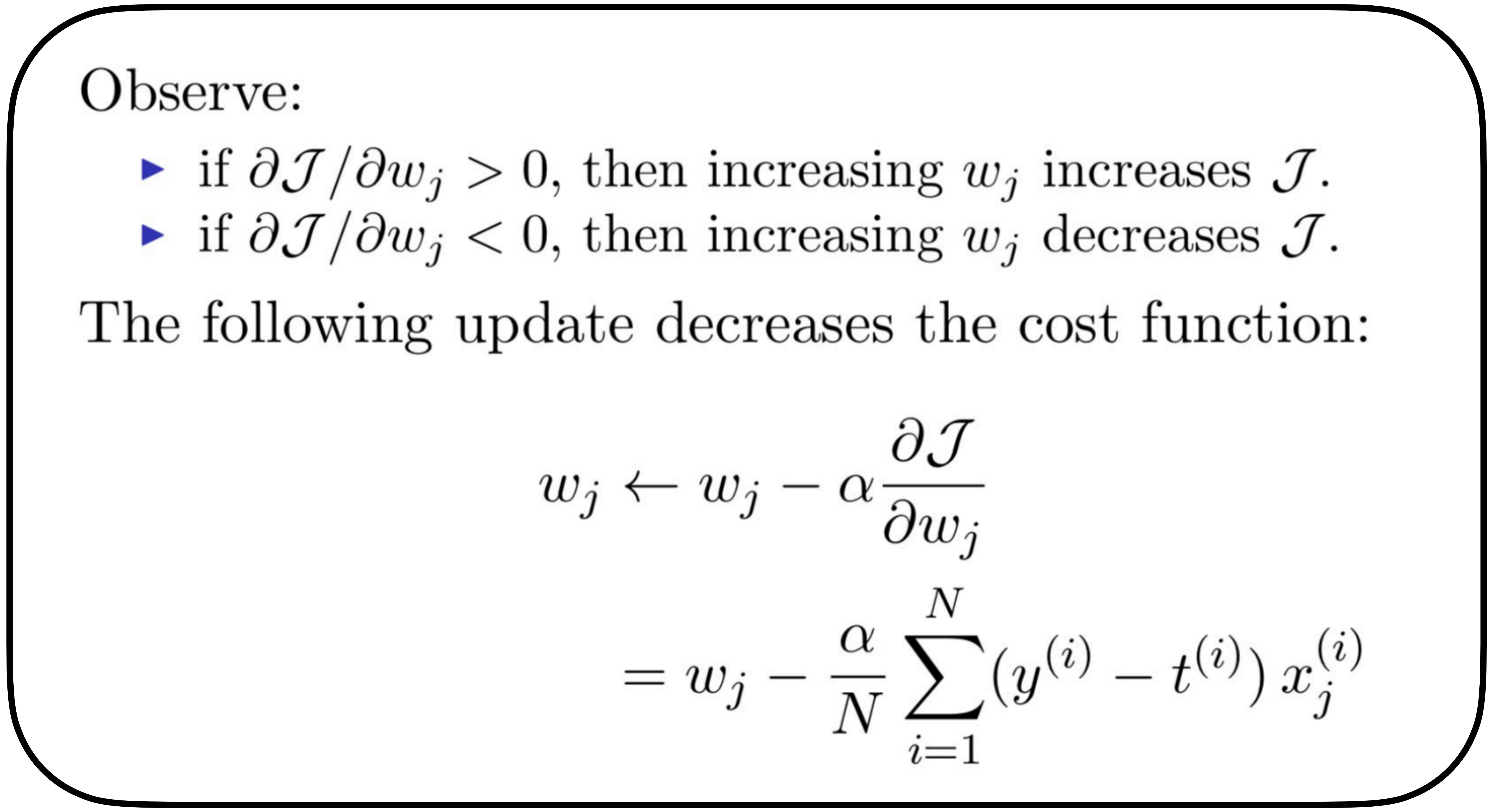
Optimization

- Now let's see a second way to minimize the cost function which is more broadly applicable: **gradient descent**.
- Gradient descent is an **iterative algorithm**, which means we apply an update repeatedly until some criterion is met.
- We initialize the weights to something reasonable (e.g. all zeros) and repeatedly adjust them in the direction of steepest descent.



Gradient descend

Optimization



Observe:

- if $\partial \mathcal{J} / \partial w_j > 0$, then increasing w_j increases $\mathcal{J}$.
- if $\partial \mathcal{J} / \partial w_j < 0$, then increasing w_j decreases $\mathcal{J}$.

The following update decreases the cost function:

$$\begin{aligned} w_j &\leftarrow w_j - \alpha \frac{\partial \mathcal{J}}{\partial w_j} \\ &= w_j - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, x_j^{(i)} \end{aligned}$$

α is the learning rate. The larger it is, the faster $\mathbf{w}$ changes.

We will see later how to tune the learning rate, but the values typically are small, e.g. 0.01, 0.0001, …

Gradient descend

Optimization

The gradient is the direction of fastest increase in the loss.

$$\frac{\partial \mathcal{J}}{\partial \mathbf{w}} = \begin{pmatrix} \frac{\partial \mathcal{J}}{\partial w_1} \\ \vdots \\ \frac{\partial \mathcal{J}}{\partial w_D} \end{pmatrix}$$

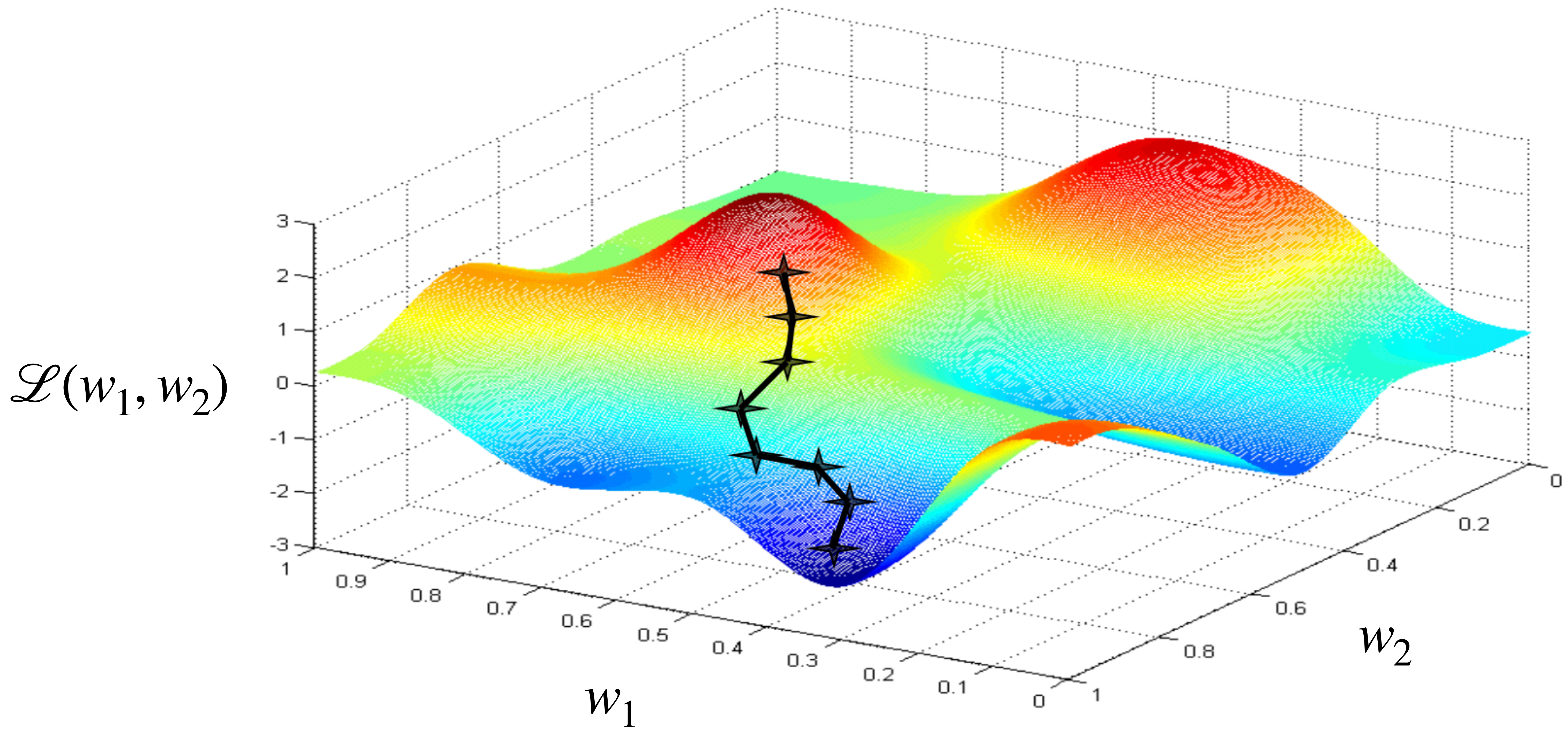
Update rule in vector form:

$$\begin{aligned} \mathbf{w} &\leftarrow \mathbf{w} - \alpha \frac{\partial \mathcal{J}}{\partial \mathbf{w}} \\ &= \mathbf{w} - \frac{\alpha}{N} \sum_{i=1}^{N} (y^{(i)} - t^{(i)})\, \mathbf{x}^{(i)} \end{aligned}$$

Hence, gradient descent updates the weights in the direction of fastest decrease.

Gradient descend

Optimization



Gradient descend

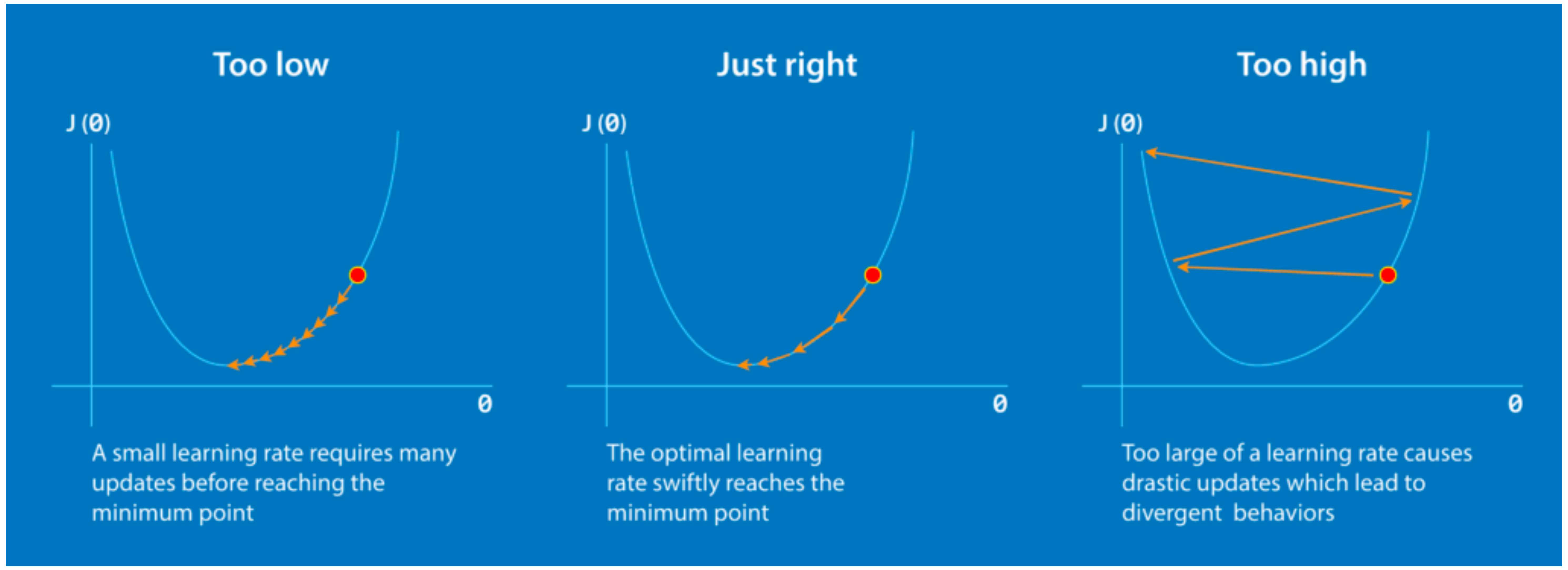
Optimization

- Why gradient descent, if we can find the optimum directly?
  - GD can be applied to a much broader set of models
  - GD can be easier to implement than direct solutions, especially with automatic differentiation software
  - For regression in high-dimensional spaces, GD is more efficient than direct solution ($(\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T t$). Why?
    - Matrix inversion is an $\mathcal{O}(D^3)$ algorithm
    - Each GD update costs $\mathcal{O}(ND)$

Gradient descend

Optimization

- In gradient descend, the learning rate α is a hyper-parameter that needs to be tuned. Here are some of the ways things can go:


Too low
J (θ)
θ
A small learning rate requires many updates before reaching the minimum point
Just right
J (θ)
θ
The optimal learning rate swiftly reaches the minimum point
Too high
J (θ)
θ
Too large of a learning rate causes drastic updates which lead to divergent behaviors

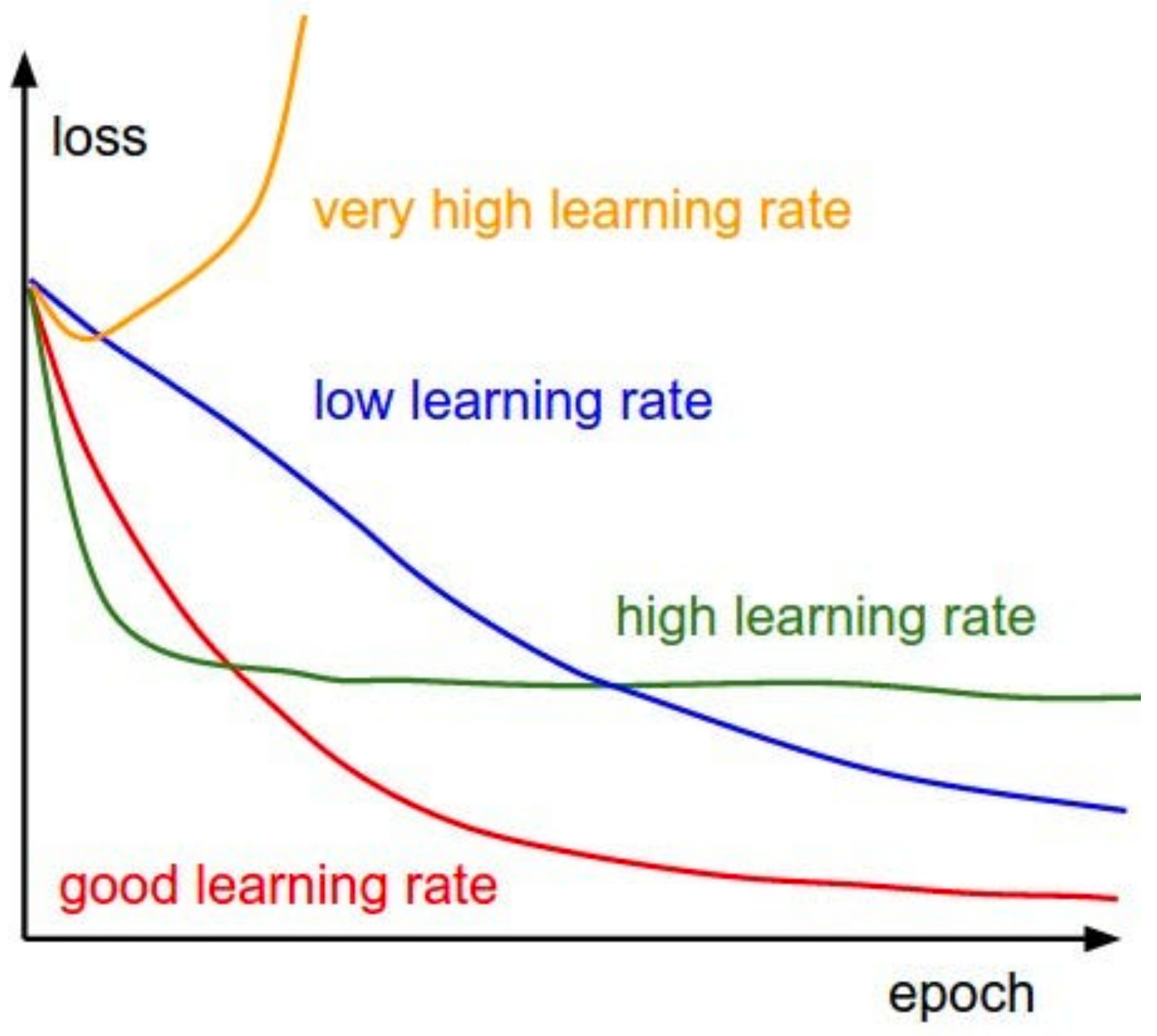
https://www.mygreatlearning.com/blog/gradient-descent/

- To find the optimal value, use the validation set to perform a grid search.

Gradient descend

Optimization

- To diagnose optimization, it is very helpful to look at the **training curves:** Training cost as a function of number of iterations.



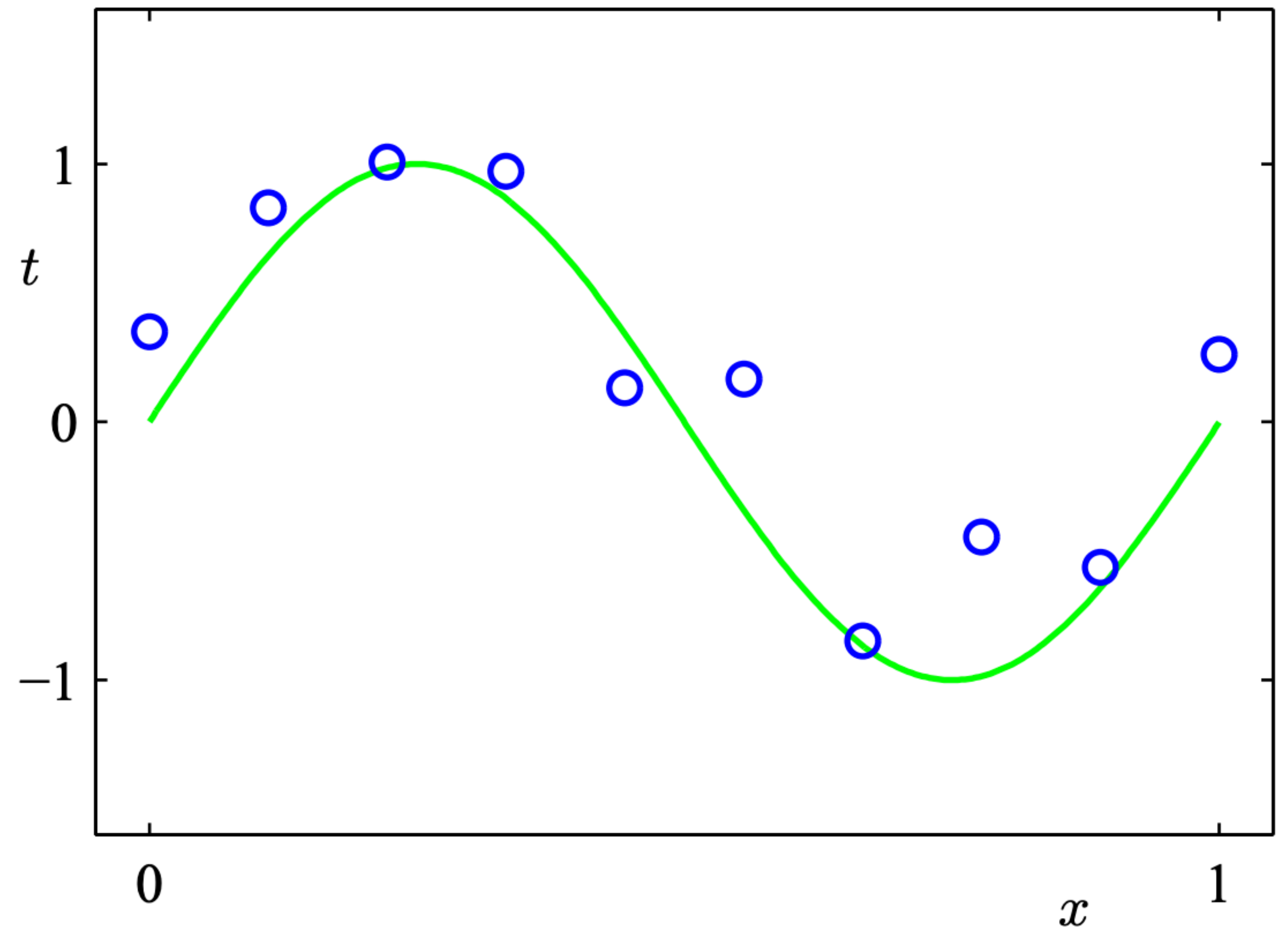
- It is very hard to tell from the training curves whether an optimizer has converged. These plots can reveal big problems, but can't guarantee convergence.

Linear models

Feature mapping

Let’s go back to our linear regression problem.

Suppose we want to model the following data:



One option: fit a degree-M polynomial; this is known as **polynomial regression**

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

Do we need to derive a whole new algorithm?

Linear models

Feature mapping

We get polynomial regression for free by mapping the input features to another space using **feature mapping.**

Let's define a **feature map** as: $\psi(\mathbf{x}) = \begin{bmatrix} 1 \\ x \\ x^2 \\ x^3 \end{bmatrix}$

Polynomial regression model: $y = \mathbf{w}^T \psi(\mathbf{x})$

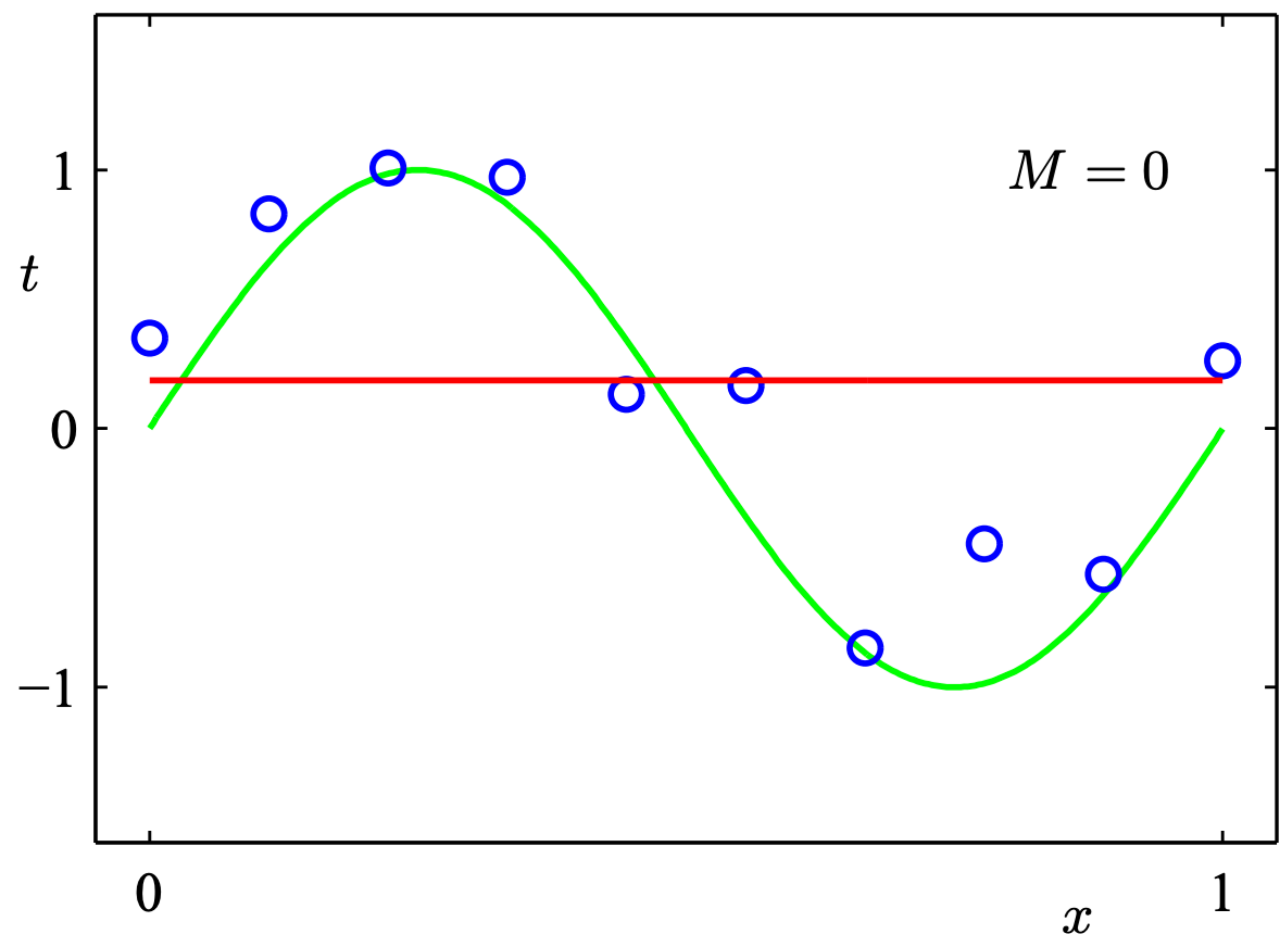
All of the derivations and algorithms so far in this lecture remain exactly the same!
We can still use least square to find $\mathbf{w}$ since $y = \mathbf{w}^T \psi(\mathbf{x})$ is linear in $\mathbf{w}$.

In general ψ can be any function, e.g.
$\psi = [1, \sin(2\pi x), \cos(2\pi x), \sin(4\pi x), \cos(4\pi x)]^T$

Linear models

Feature mapping

$$y = w_0$$

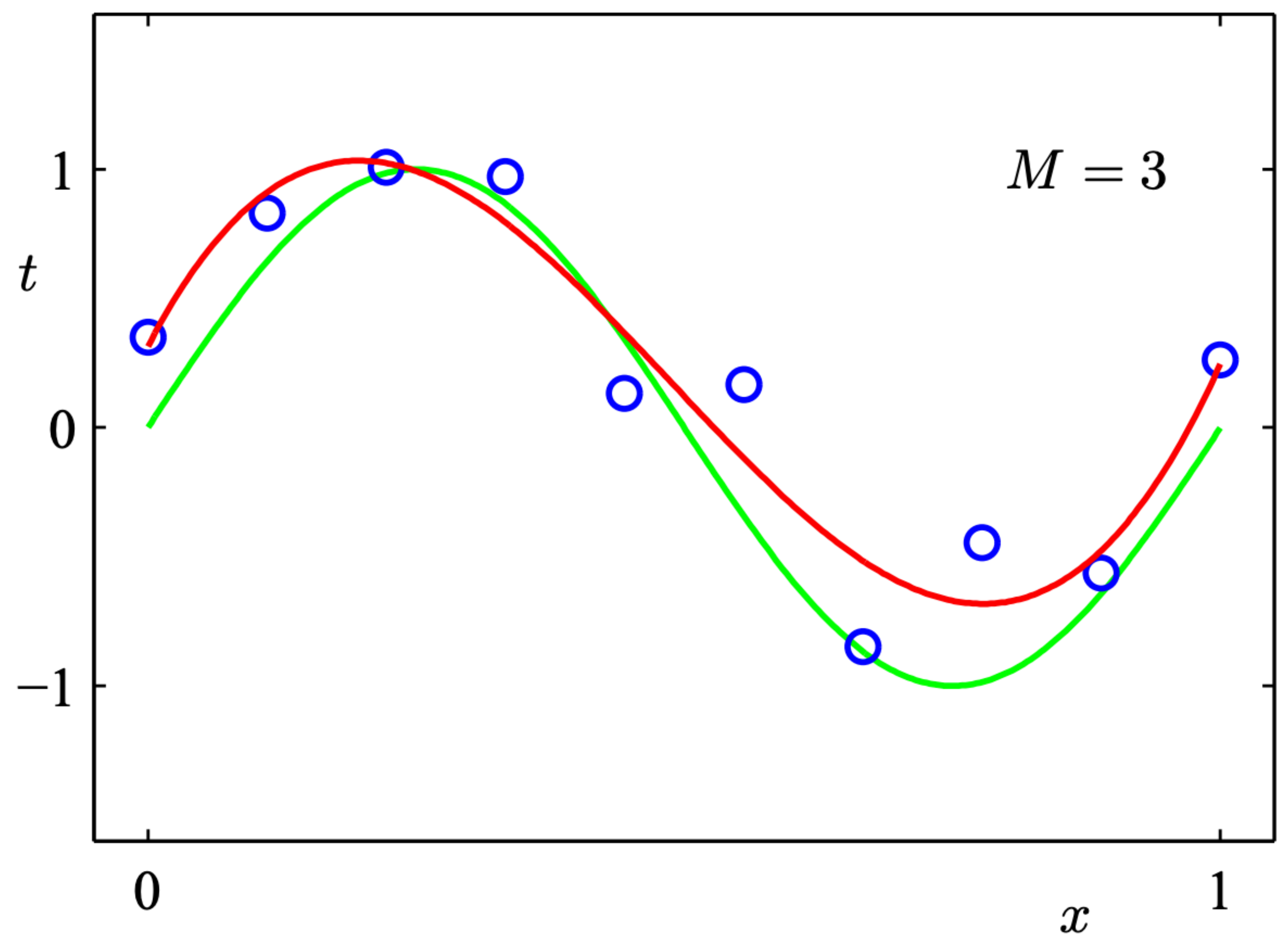


Pattern Recognition and Machine Learning, Christopher Bishop

Linear models

Feature mapping

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

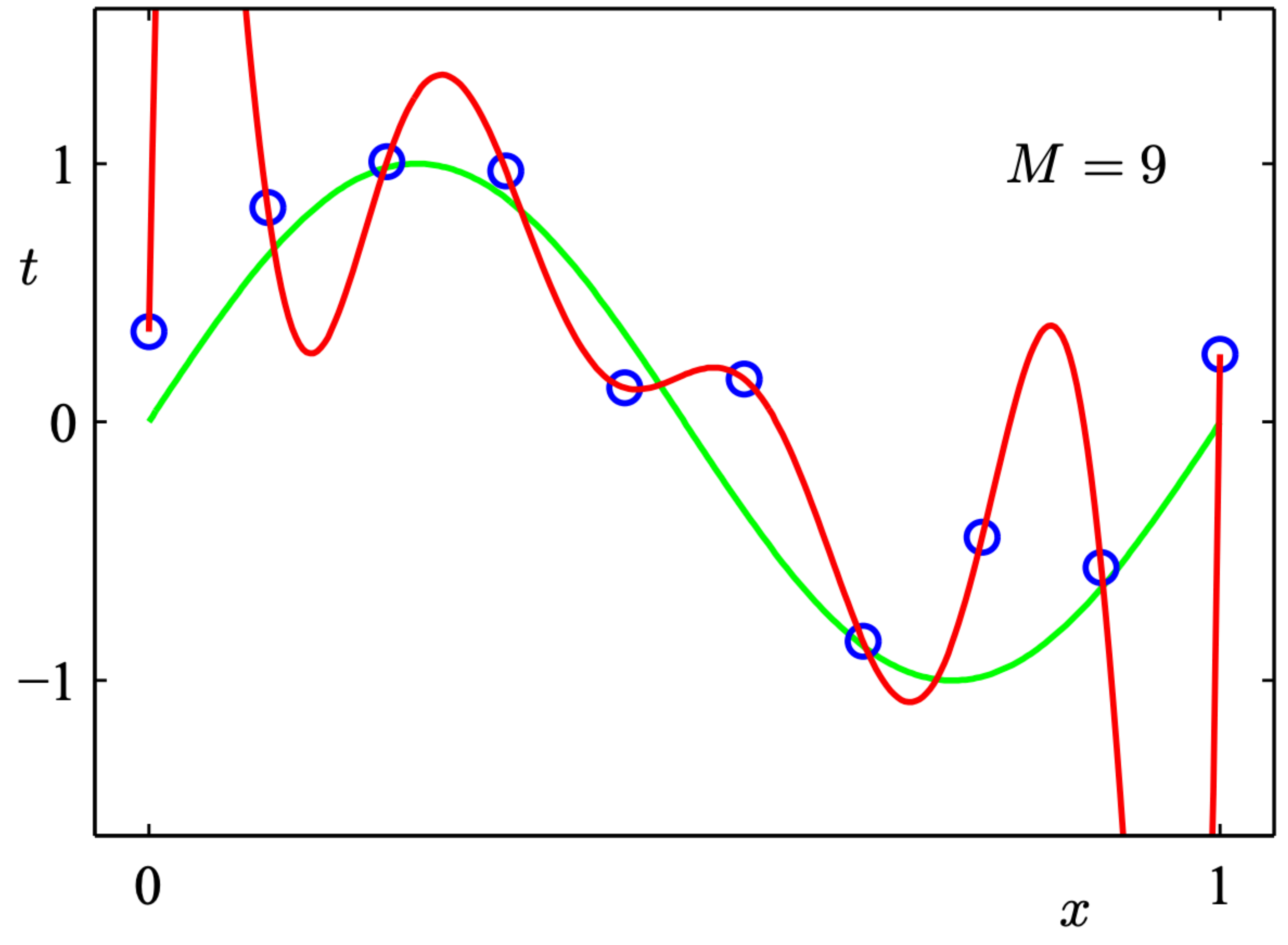


Pattern Recognition and Machine Learning, Christopher Bishop

Linear models

Feature mapping

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



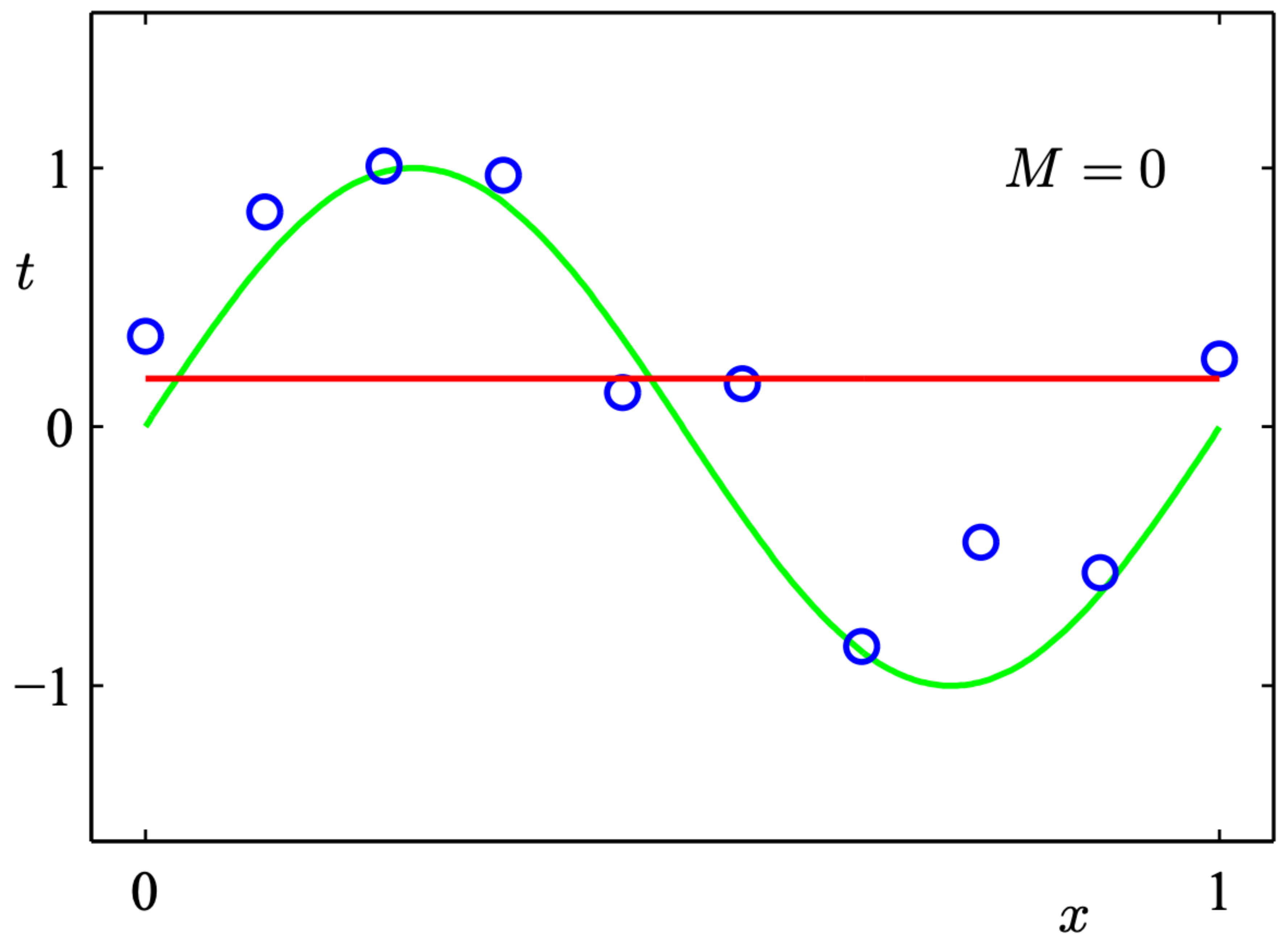
Pattern Recognition and Machine Learning, Christopher Bishop

Linear models

Feature mapping

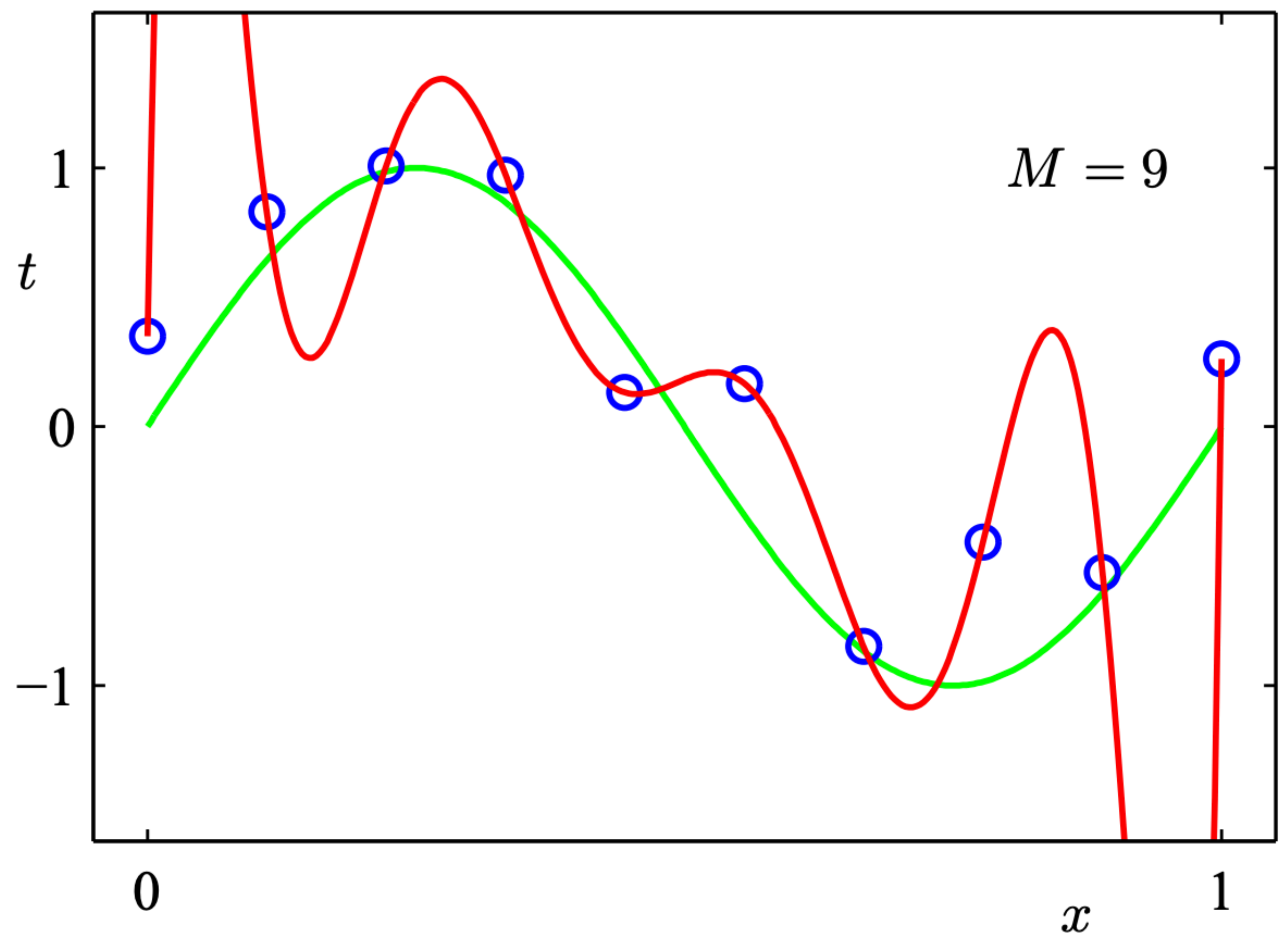
Underfitting : model is too simple — does not fit the data.

$$y = w_0$$



Overfitting : model is too complex — fits perfectly, does not generalize.

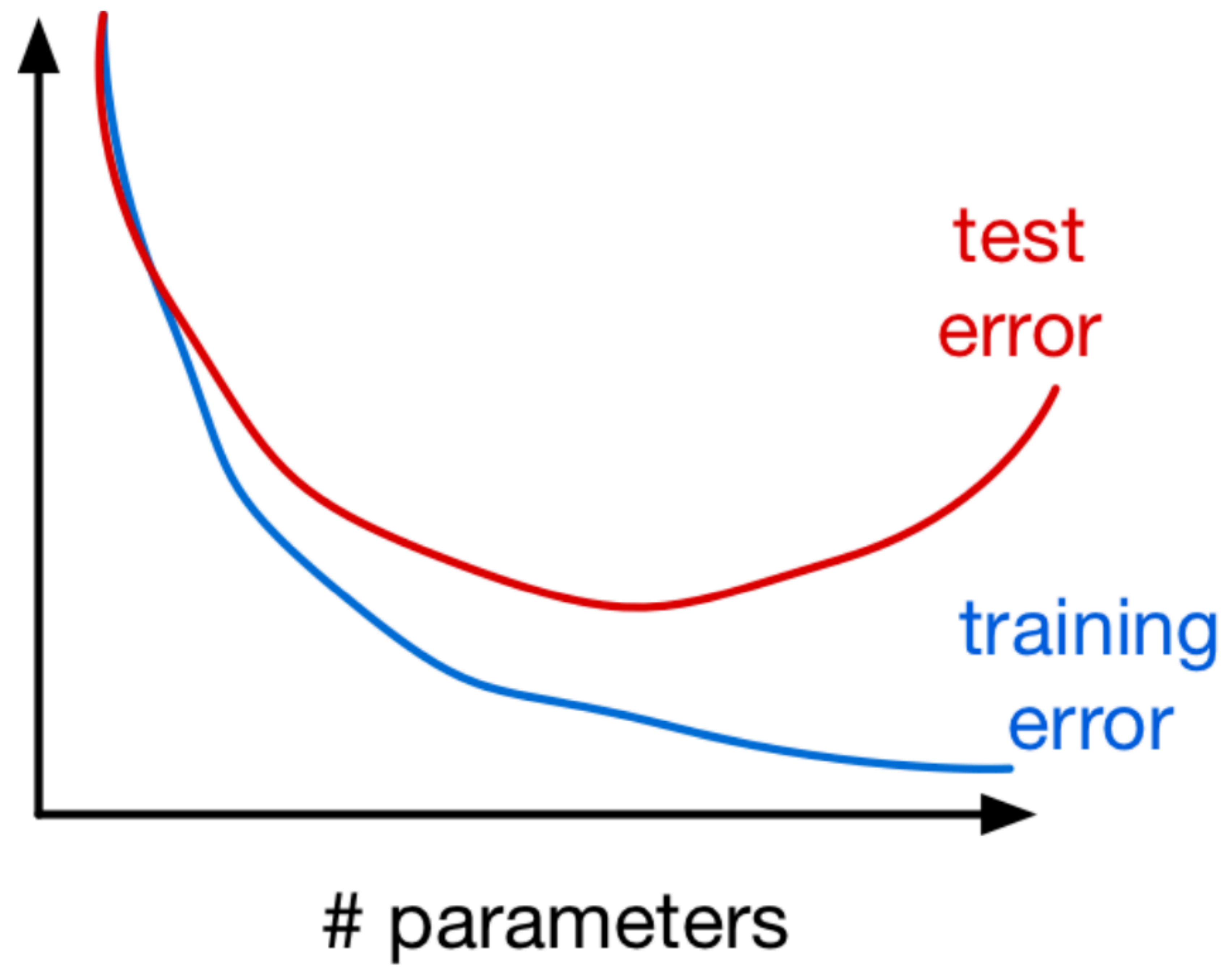
$$y = w_0 + w_1 x + w_2 x^2 + w_3 x^3 + \ldots + w_9 x^9$$



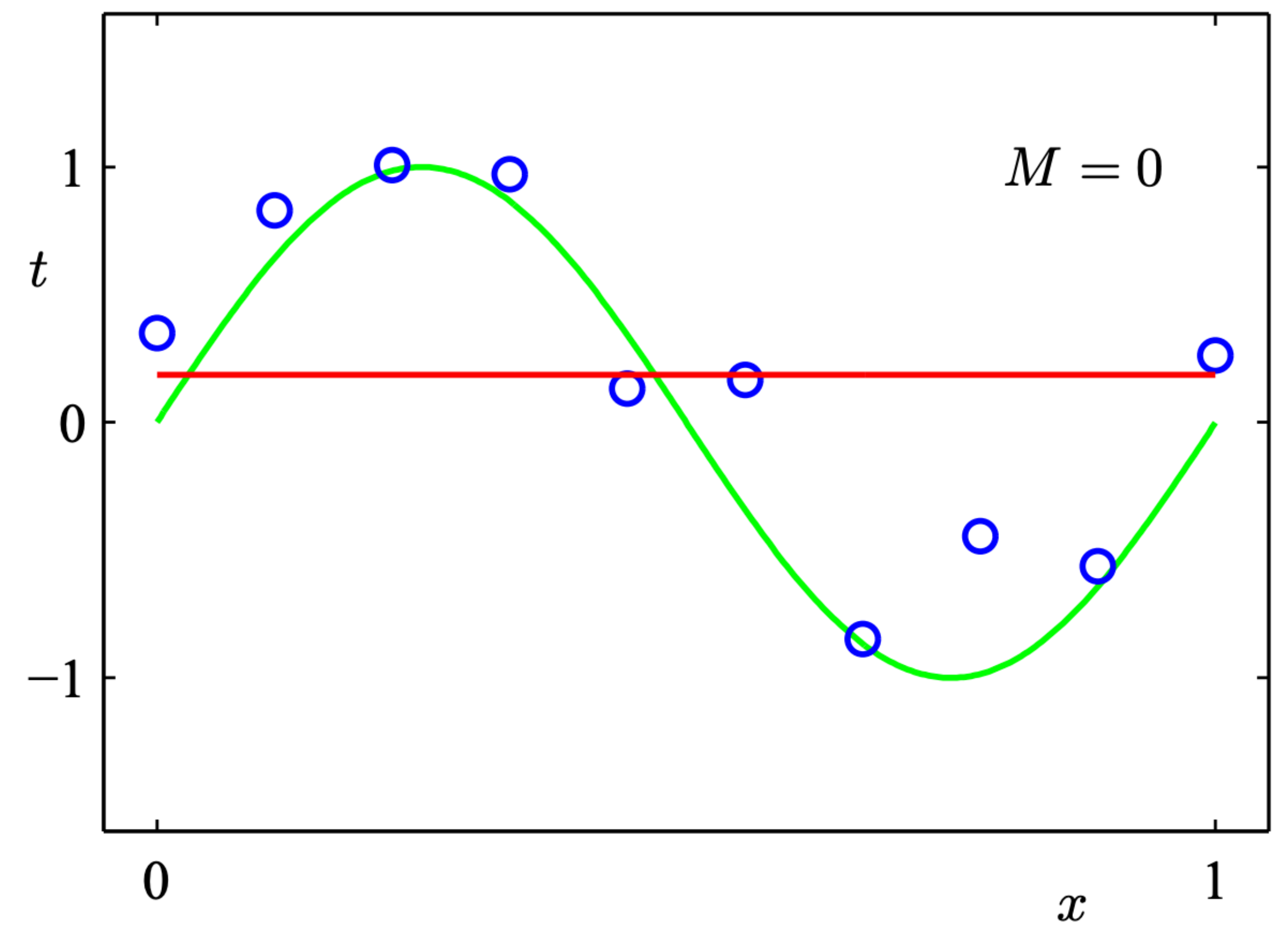
Linear models

Feature mapping

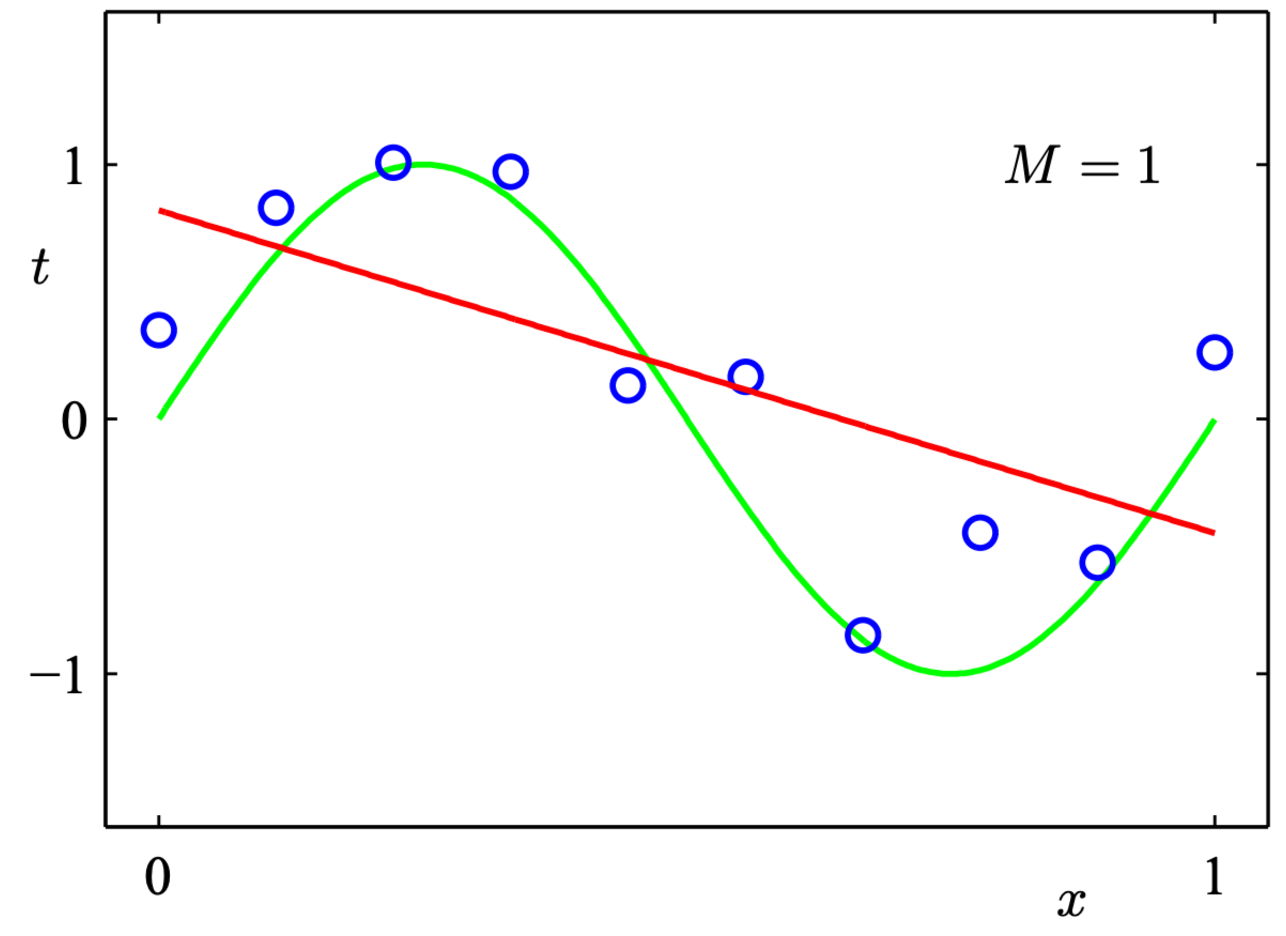
Training and test error as a function of #parameters:



$y = w_0$



$y = w_0 + w_1 x$



Linear models

Model selection

- The **degree of the polynomial** is a hyperparameter, just like k in KNN. We can tune it using a validation set.
- Restricting the parameters of the model (M in this case) is a crude solution to controlling complexity.
- A better solution is to keep the model large, but enforce a simpler solution.
- This is done through **regularization** or **penalization**
- How?

Break

10 minutes

Linear models

Regularization

- A **regularizer** is a function that quantifies how much we prefer one hypothesis vs. another, encouraging a simpler solution.
- E.g. We can encourage the weights to be small by choosing as our regularizer the L^2 **penalty (ℓ_2-norm)**.

$$\mathcal{R}(\mathbf{w}) = \tfrac{1}{2}\|\mathbf{w}\|^2 = \frac{1}{2}\sum_j w_j^2$$

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

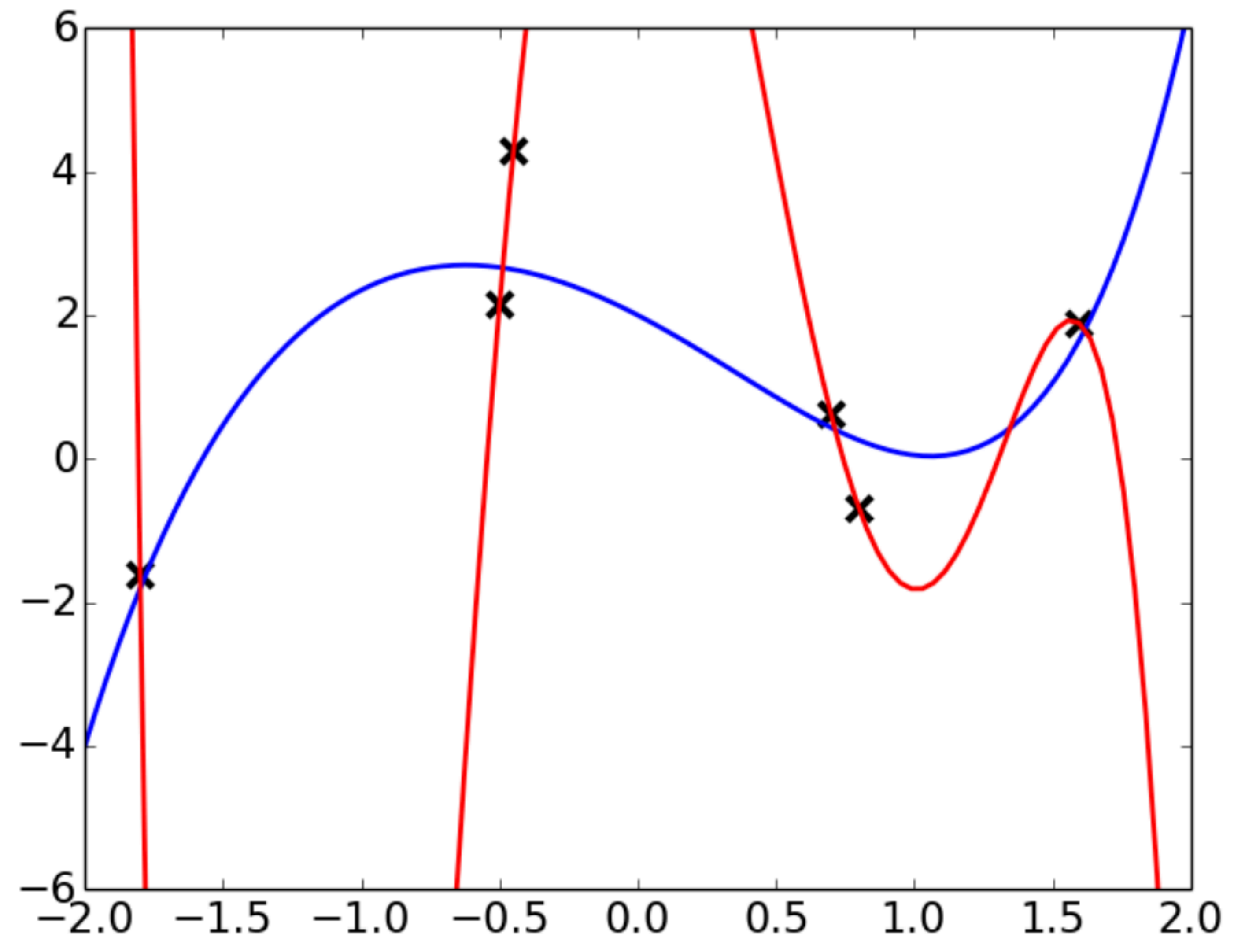
$$\mathcal{J}_{\text{reg}} = \mathcal{J} + \lambda\mathcal{R} = \mathcal{J} + \frac{\boxed{\lambda}}{2}\sum_j w_j^2$$

a hyperparameter that we can tune using a validation set

Linear models

Regularization

- The idea is that simpler functions have smaller L^2 norm of their weight.
- E.g. polynomials that overfit often have large coefficients.



$$y = 0.1x^5 + 0.2x^4 + 0.75x^3 - x^2 - 2x + 2$$

$$y = -7.2x^5 + 10.4x^4 + 24.5x^3 - 37.9x^2 - 3.6x + 12$$

Linear models

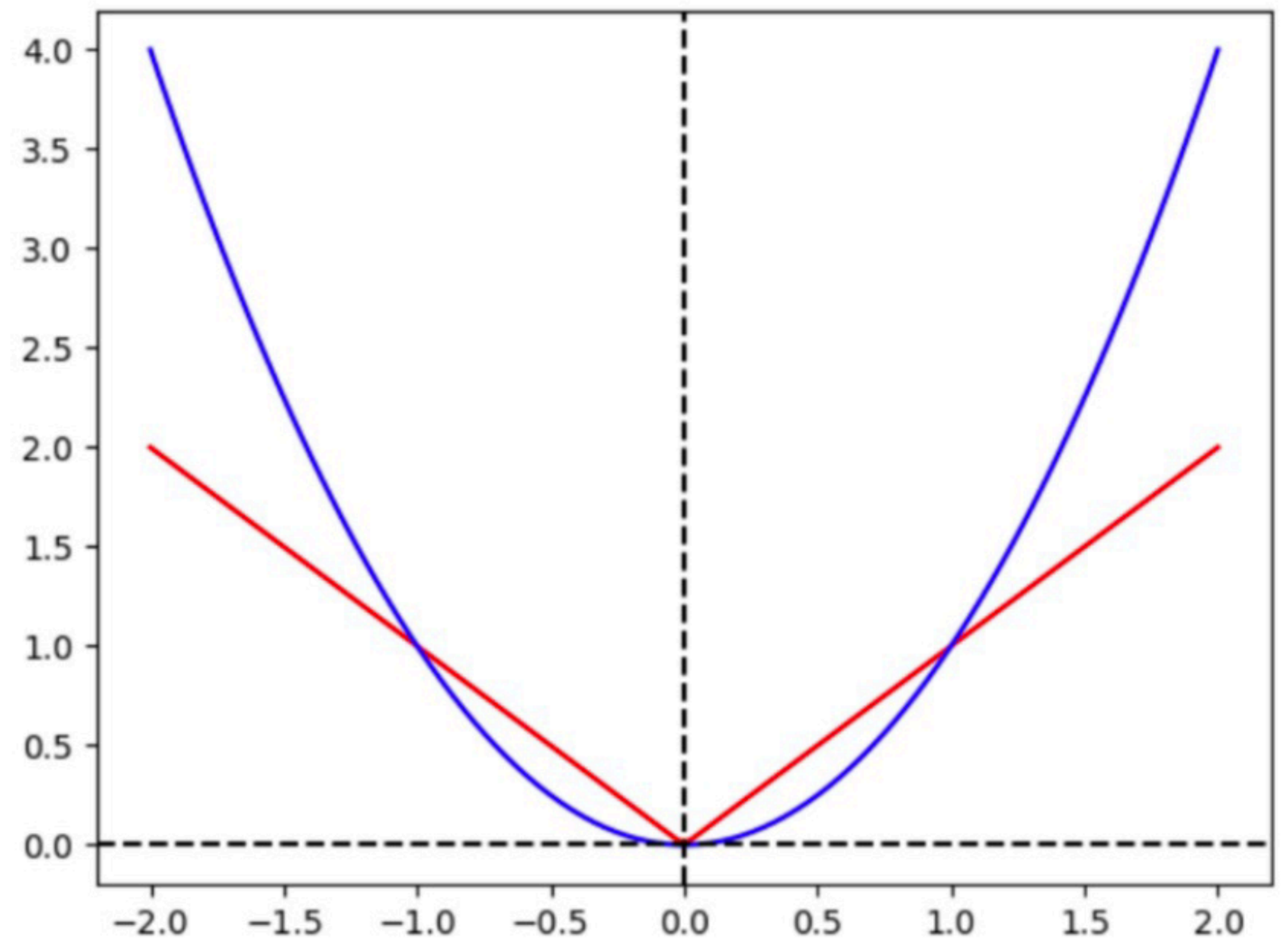
Regularization for linear regression

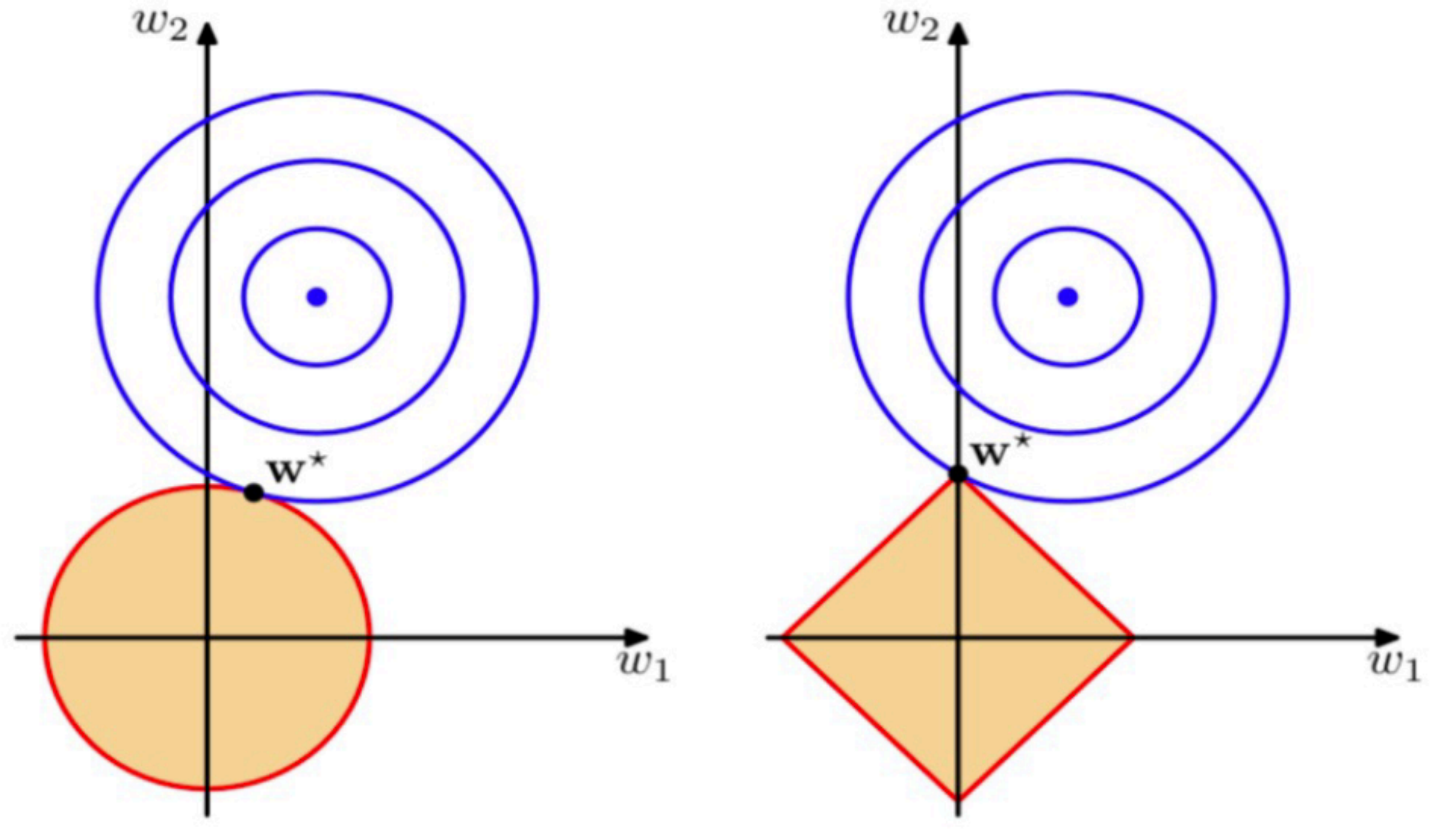
- The least square loss for linear regression is: $\mathcal{J}(\mathbf{w}) = \frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|^2$
- With $\lambda > 0$, the regularize loss is: $$\frac{1}{2N}\|\mathbf{X}\mathbf{w} - \mathbf{t}\|_2^2 + \frac{\lambda}{2}\|\mathbf{w}\|_2^2$$
- This makes the closed for solution: $(\mathbf{X}^T\mathbf{X} + \lambda N\mathbf{I})^{-1}\mathbf{X}^T\mathbf{t}$

Linear models

Regularization

- The ℓ_1-norm or sum of absolute values is another regularizer that encourages weights to be exactly zero. What do you think is the different behaviour?





L2 regularization

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

L1 regularization

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

- We can design regularizers based on whatever property we'd like to encourage!

Linear models

Conclusion

- Linear regression exemplifies recurring themes of this course:
  - choose a **model** describing the relationships between variables of interest
  - define a **loss function** quantifying how bad is the fit to the data
  - choose a **regularizer** saying how much we prefer different candidate explanations
  - fit the model that minimizes the loss function using an **optimization algorithm**
- Optimization can be done through **closed-form solution** or **gradient descend**.
- Linear models can be made more powerful using feature mapping.
- Generalization can be improved by adding regularization.
- Next lecture:
  - Neural networks!

Tutorial

Colab

- Introduction to Python (continued)
- Evaluation metrics