349:Machine Learning

Fall 2024

Gradient Descent and Regularization

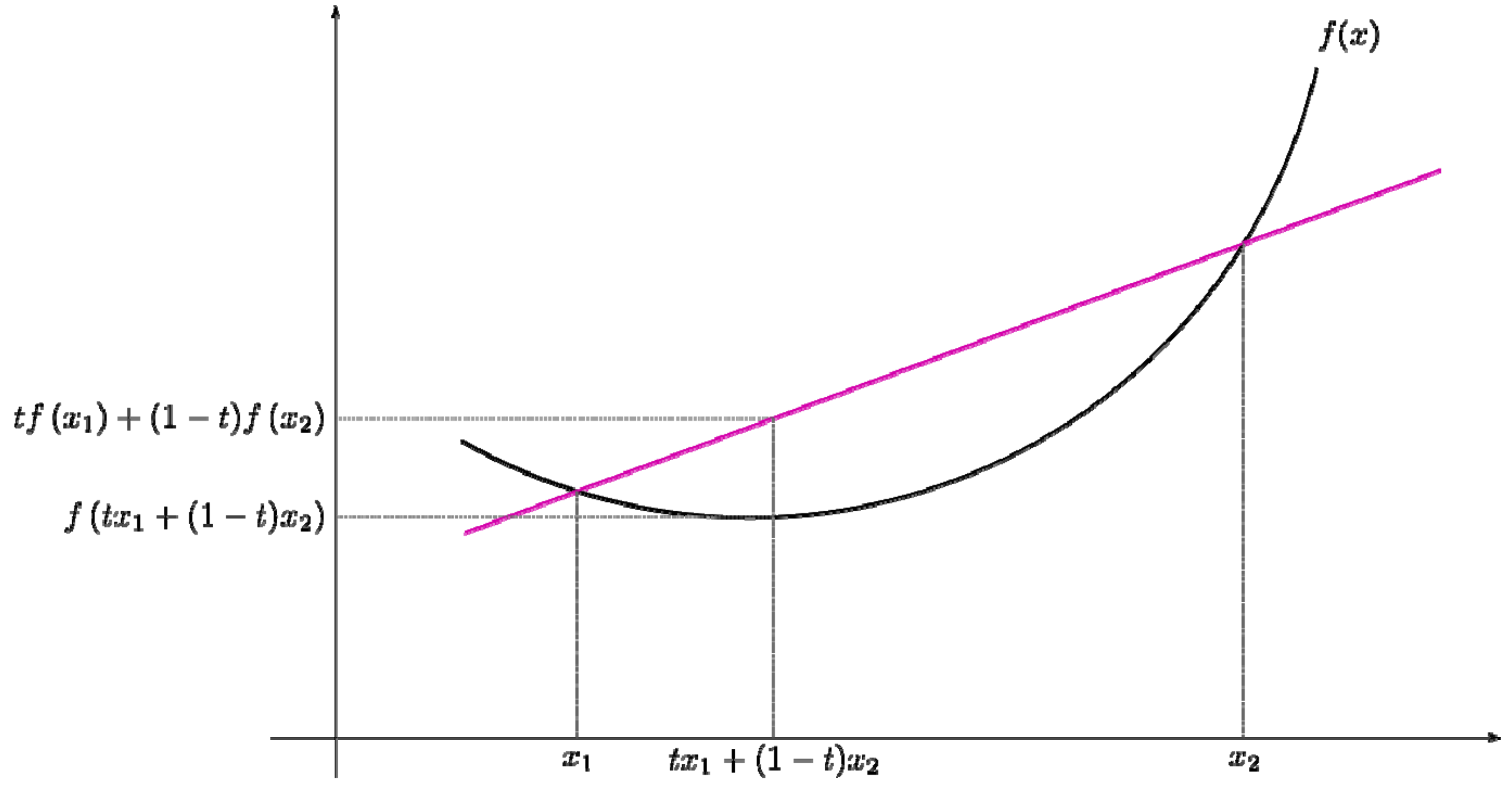
Why not solve analytically?

- Often we don't know how.
- Even when we do have an exact (analytical) solutions to machine learning problems (e.g., linear regression), it may be problematic.
- For example with linear regression we solve with this formula..

$$\theta^* = (X^T X)^{-1} X^T Y$$

  - But this requires our hypothesis space to contain only convex functions
  - And requires us to use all data at once
  - And requires $\sim O(n^3)$ matrix multiplication and $O(d^3)$ matrix inverse
- Gradient descent avoids these issues

Example of a Convex Function



Supervised Machine Learning in one slide

1. Pick data X, labels Y, model M(θ) and loss function $L(\mathrm{X}, \mathrm{Y}; \theta)$

2. Initialize model parameters θ, somehow

3. Measure model performance with the loss function $L(X, Y; \theta)$

4. Modify parameters θ somehow, hoping to improve $L(X, Y; \theta)$

5. Repeat 3 and 4 until you stop improving or run out of time

Supervised Machine Learning in one slide

1. Pick data X, labels Y, model M(θ) and loss function $L(\mathrm{X}, \mathrm{Y}; \theta)$
2. Initialize model parameters θ, somehow
3. Measure model performance with the loss function $L(X, Y; \theta)$

HOW?

4. Modify parameters θ somehow, hoping to improve $L(X, Y; \theta)$
5. Repeat 3 and 4 until you stop improving or run out of time

Modifying the parameters with Gradient Descent

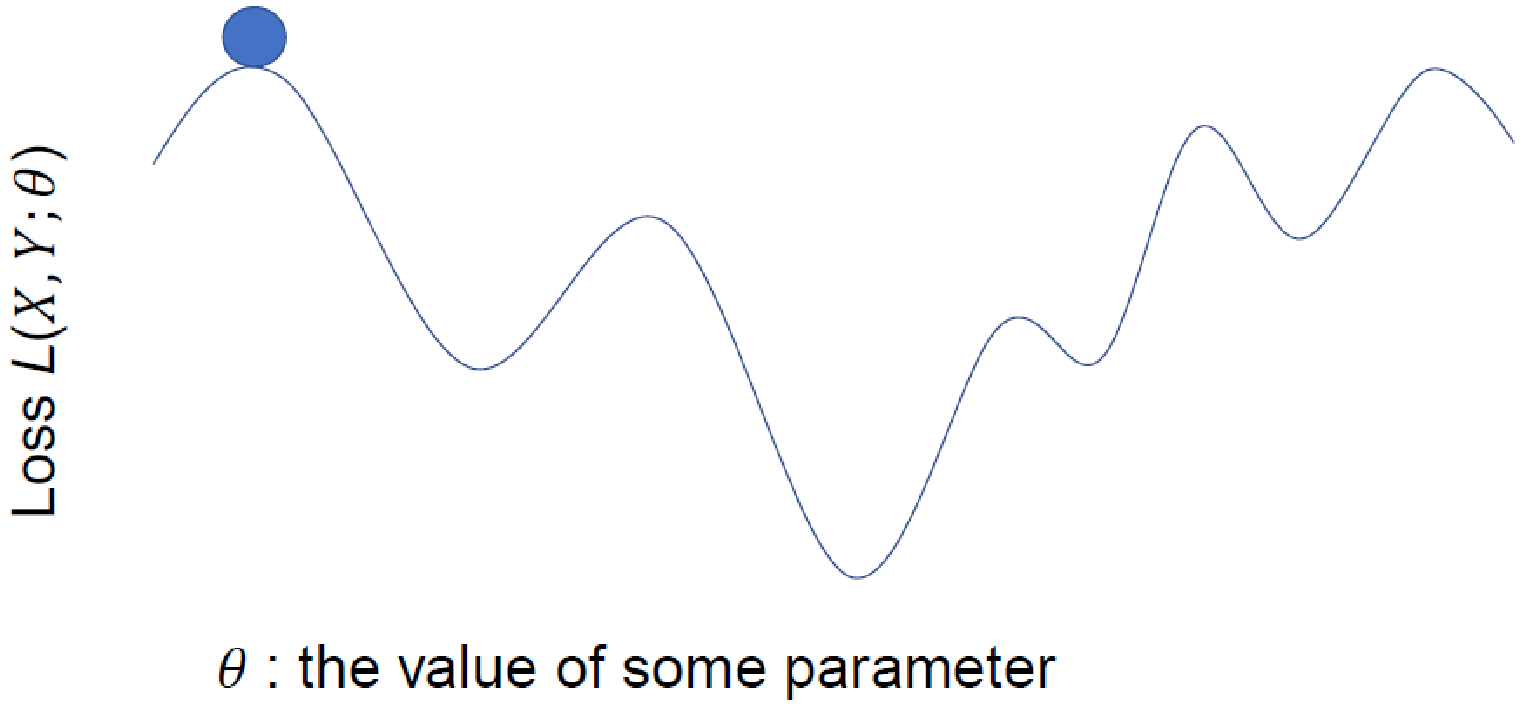
1. Measure how the the loss changes when we change the parameters θ slightly (measure the gradient with respect to the weights).

2. Pick the next set of parameters to be close to the current set, but in the direction that most changes the objection function for the better (follow the gradient)

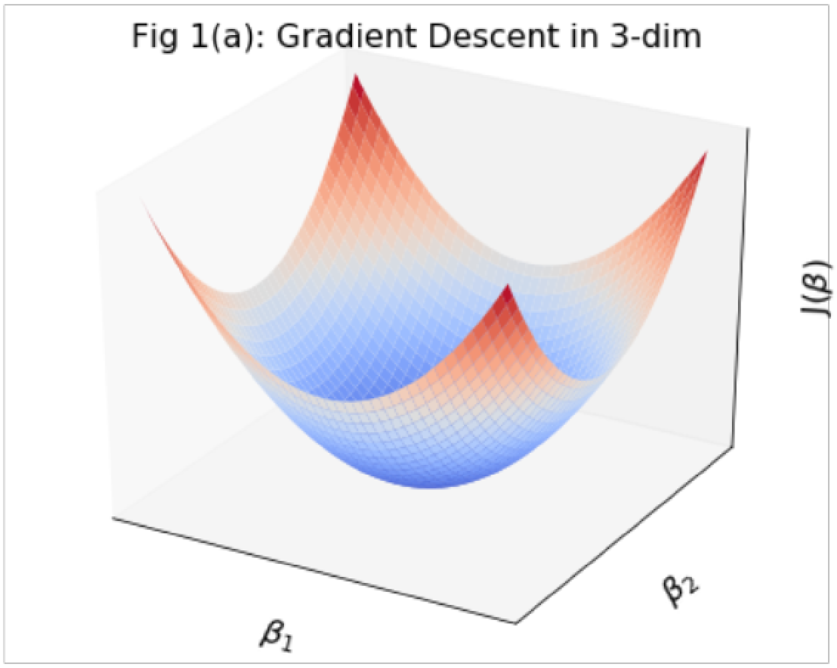
3. Repeat

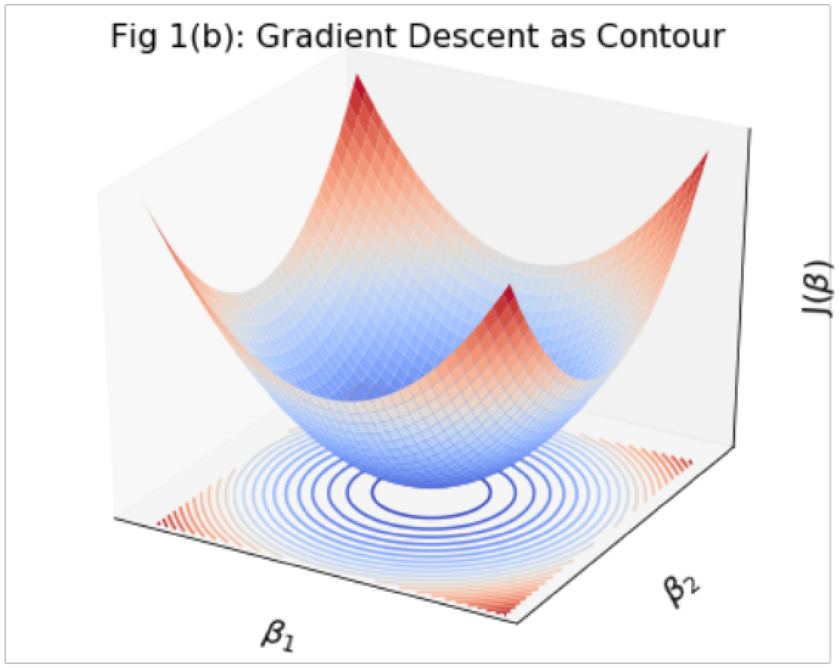
Gradient Descent: Promises & Caveats

- Much faster than guessing new parameters randomly
- Finds the global optimum only if the objective function is convex



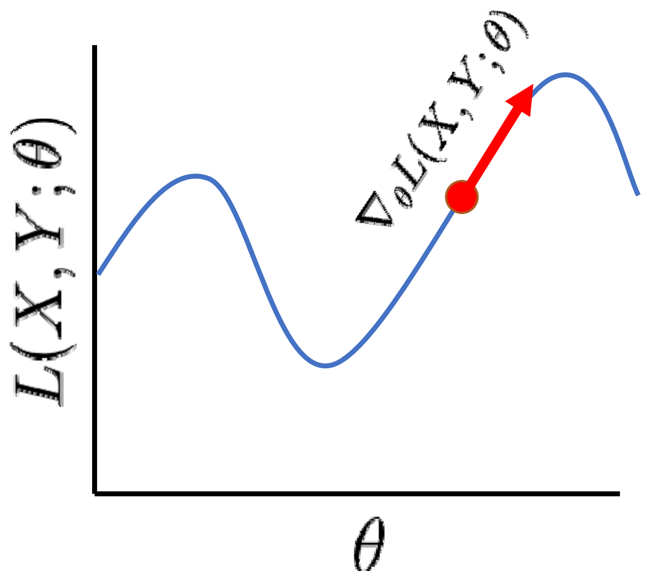
Cost Function Landscape





What does the gradient tell us?

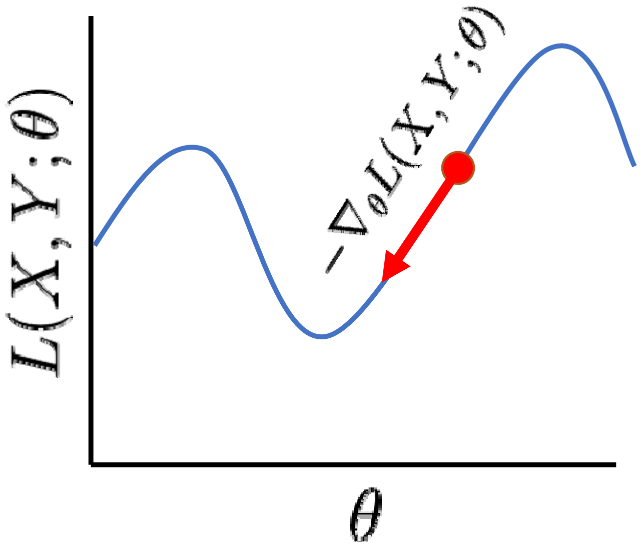
- If the loss function and hypothesis function are differentiable* (i.e., the gradient exists)
- We can evaluate the gradient for some fixed value of θ and get the *direction* in which the loss *increases* fastest



*or subdifferentiable

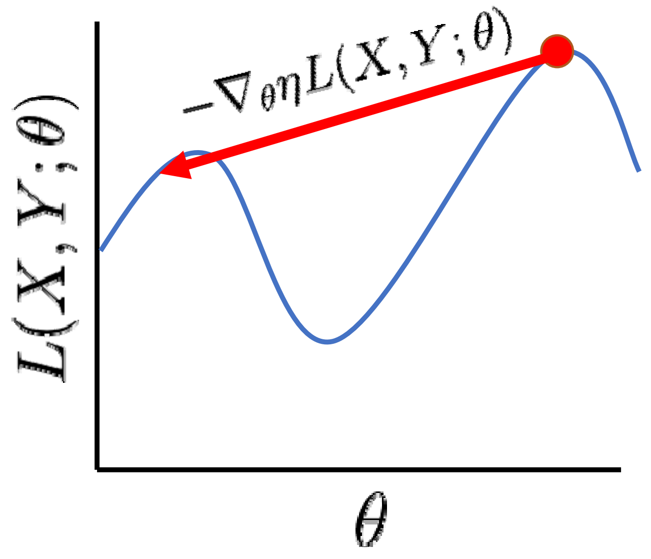
What does the gradient tell us?

- We want to *decrease* our loss, so let's go the other way instead

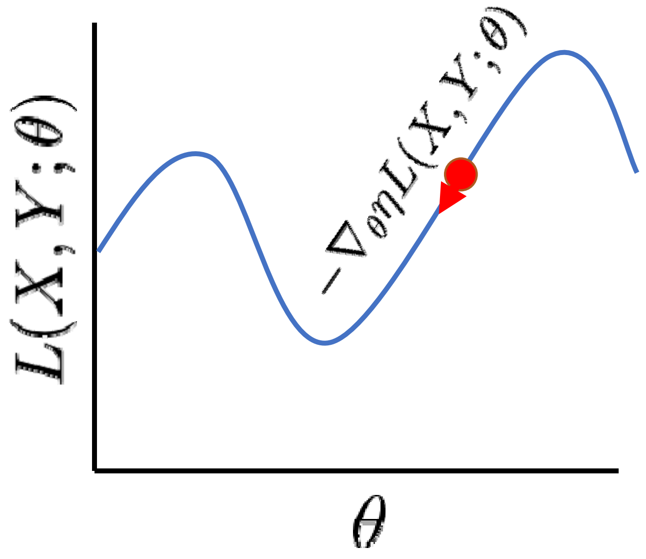


Step Size: how far should we go?

- The gradient we calculated was based on a fixed value of θ
- As we move away from this point, the gradient changes



If the step size is too large, we may overshoot the minimum



If the step size is too small, we need to take more steps (more computation)

Gradient Descent Pseudocode

```
Initialize $\theta^{(0)}$
Repeat until convergence:
        $\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_{\theta^{(t)}} L(X, Y; \theta^{(t)})$
Return $\theta^{(t_{max})}$
```

Design Choices:

- Initialization of θ
- Convergence criteria
- Choosing a loss function
- How much data to use during each step (batch size)
- Step size (η)

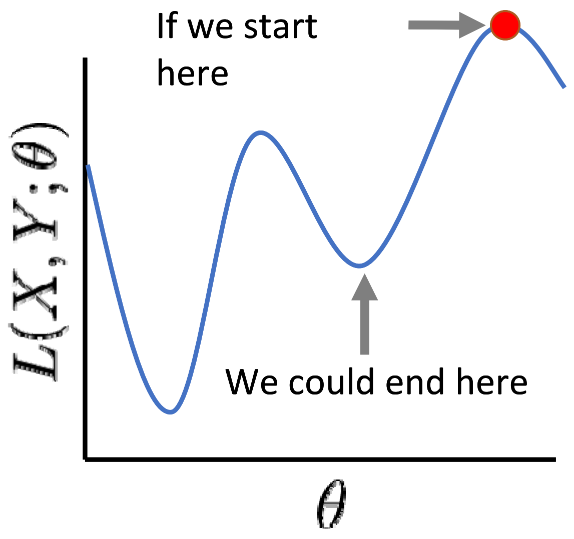
Parameter Initialization

Common initializations:

- $\theta^{(0)} = 0$
- $\theta^{(0)} =$ random values

What happens if our initialization is bad?

- Convergence to a *local* minimum
- No way to determine if you've converged to the global minimum



Knowing when to stop gradient descent

- Stop when the gradient is close (within ε) to 0 (i.e., we reached a minimum)
- Stop after some fixed number of iterations
- Stop when the loss on a *validation set* stops decreasing
  - This helps prevent overfitting

Loss Functions

A good objective (loss) function $L(\underset{\text{data}}{\uparrow \mathrm{X}}, \underset{\text{labels}}{\uparrow \mathrm{Y}}; \underset{\text{parameters}}{\uparrow \theta})$

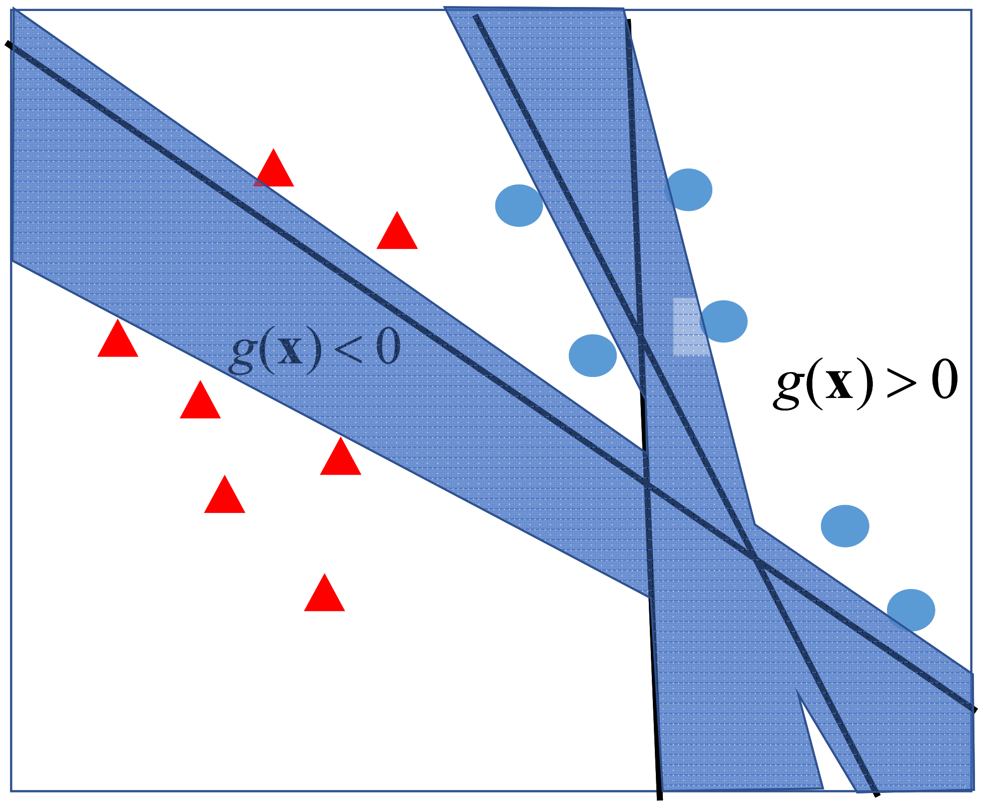
Required $\begin{cases} L(\mathrm{X}, \mathrm{Y}; \theta) \geq 0 \\ L(\mathrm{X}, \mathrm{Y}; \theta) \text{ decreases as performance improves} \end{cases}$

Required for gradient descent $\begin{cases} L(\mathrm{X}, \mathrm{Y}; \theta) \text{ is differentiable*, with respect to } \theta \end{cases}$

helpful For gradient descent $\begin{cases} \text{The gradient of } L \text{ is bounded ... } \mathbf{0} < |\nabla L| \ll \infty \end{cases}$

*or subdifferentiable

Example: 0 1 loss



$$g(\mathbf{x}) = w_0 + w_1 x_1 + w_2 x_2 = 0$$

$$h(\mathbf{x}) = \begin{cases} 1 & \text{if } g(\mathbf{x}) > 0 \\ -1 & \text{otherwise} \end{cases}$$

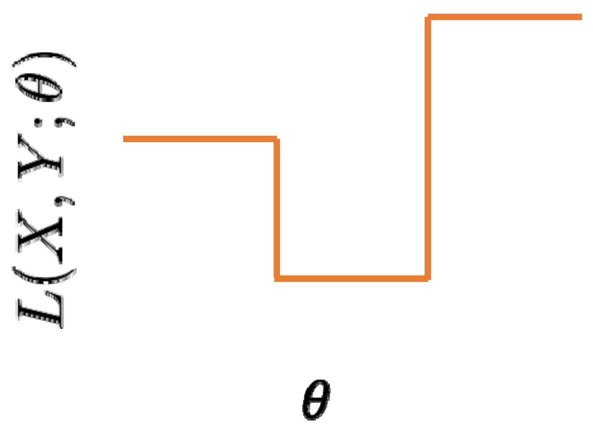
$$SSE = \sum_i^n (y_i - h(\mathbf{x}_i))^2$$

SSE is same everywhere in the blue

Gradient 0 in the blue region!

The 0 1 Loss function

- A generic term for machine learning model parameters is θ
- Loss = 1 if $y \neq h_\theta(x)$, else it's 0
- A count of mislabeled items
- Results in a step function
- Not useful for for gradient descent



Stochastic, Batch, Mini-Batch Descent

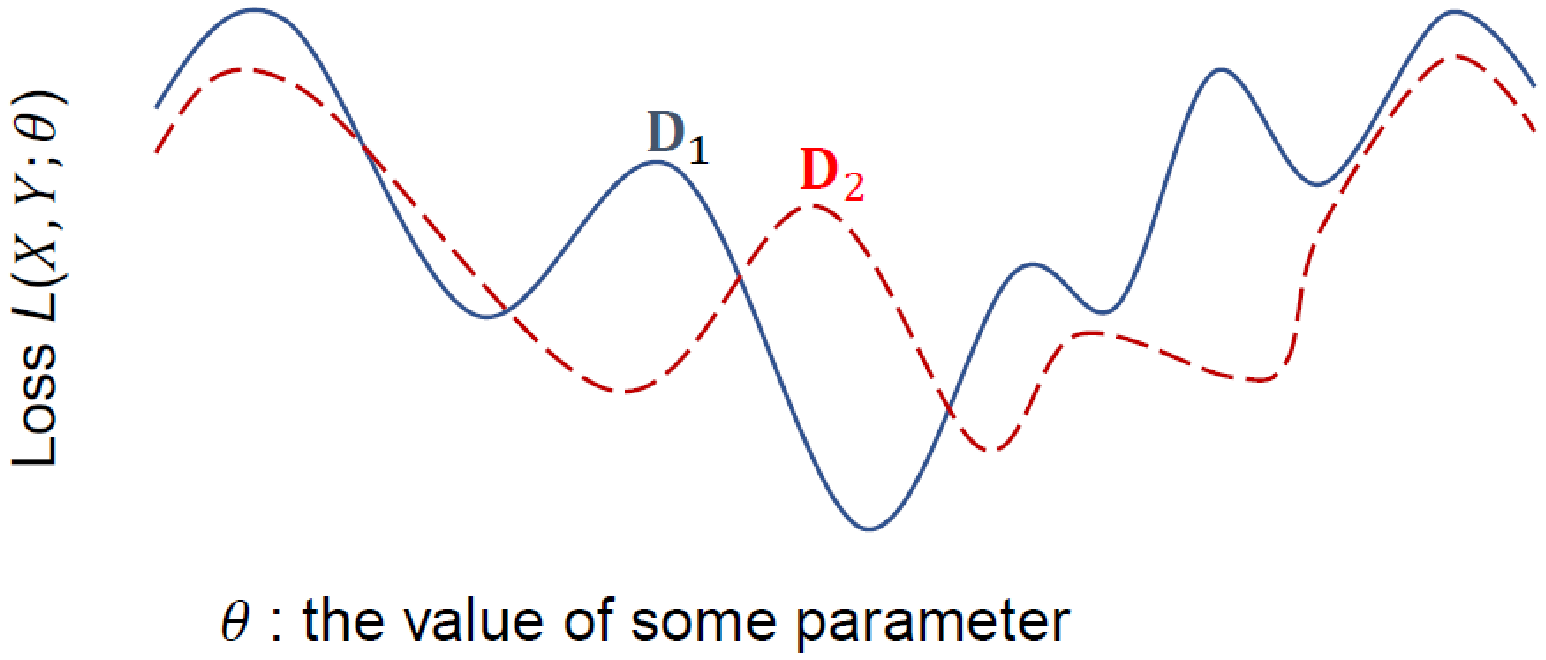
- In **batch gradient descent**, the loss is a function of both the parameters and the set of all training data **D.** (What if if |**D**| > memory?)

- In **stochastic gradient descent**, los is a function of the parameters and a different single random training sample at each iteration. (How does the gradient change when you change **D** at every step?)

- In **mini-batch gradient descent**, random subsets of the data (e.g. 100 examples) are used at each step in the iteration.

Different data, different loss

- If the data D changes….then the landscape of the loss function changes
- You typically won't know how it has changed.

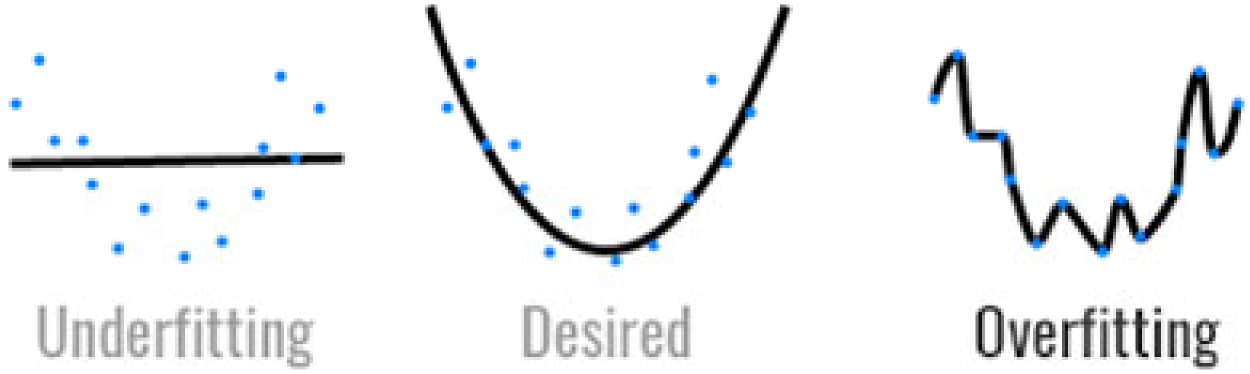


How much data to use in each step?

- **All of it (*batch gradient descent*)**
  - The *most accurate* representation of your training loss
  - It can be slow
  - Not possible if data does not fit in RAM
- Just one data point (*stochastic gradient descent*)
  - A *noisy, inaccurate* representation of your training loss
  - *very* fast
  - Random shuffling is important
- More than one data point, but less than all (*mini-batch gradient descent)*
  - Most common approach today
  - Balances *speed* and *accuracy*
  - Random shuffling is important
  - Usually want batch size to be as large as possible for your machine

Revisiting Overfitting

- Overfitting occurs when your model begins to “memorize” the training data
  - Can detect overfitting from an increasing gap between training and validation loss.
  - Performance on the training set improves, but performance on the validation set does not.



Revisiting Overfitting: Regularization

- Big idea (**Occam’s Razor**) – Given two models with equal performance, prefer the *simpler* model.
  - E.g., models with fewer parameters or smaller coefficients

- Regularization can be applied to any loss function

$$L_R(X, Y; \theta) = L(X, Y; \theta) + \lambda R(\theta)$$

- The amount of regularization is controlled by the hyperparameter λ

L1- and L2-regularization

- Recall the l_p-norm:

$$\ell_p(\theta) = \sqrt[p]{\sum_{i=1}^{d} |\theta_i|^p}$$

- l_1-regularization penalizes high values of the l_1-norm of the model parameters:

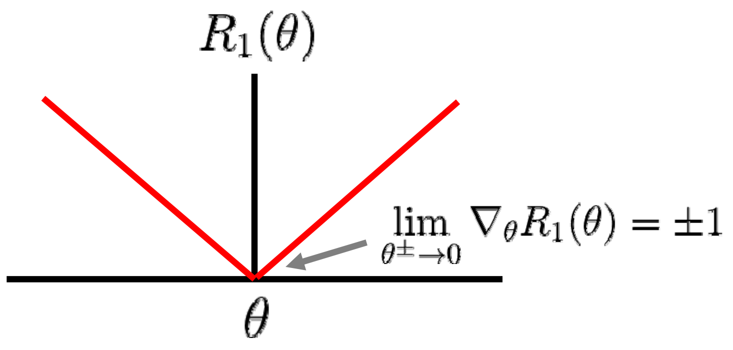
$$R_1(\theta) = \sum_{i=1}^{d} |\theta_i|$$

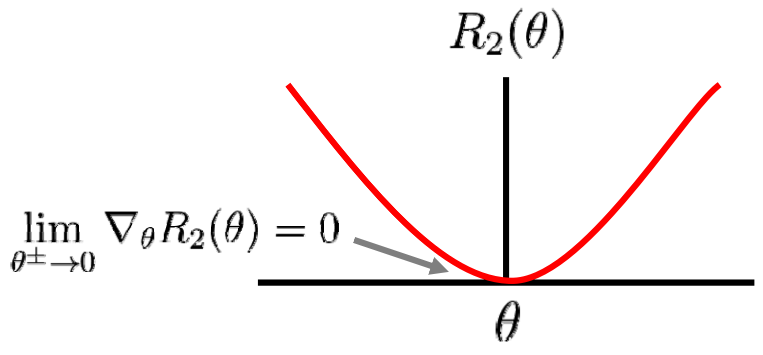
- l_2-regularization penalizes high values of the l_2-norm:

$$R_2(\theta) = \frac{1}{2} \sum_{i=1}^{d} |\theta_i|^2$$

L1-regularization and sparsity

- L1-regularization encourages the model parameters to be *sparse*
  - This is a form of feature selection
  - Only features with non-zero coefficients contribute to the model's prediction
- This is because the gradient of L1-regularization moves model parameters towards 0 at a *constant* rate





Regularization and offset (aka bias)

- Many ML models include a bias term, b.

- Example: A linear model: $g_\theta(x) = \theta^T x + b$

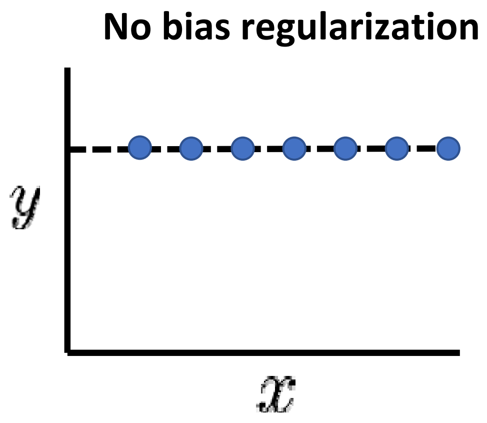
- Or equivalently, by augmenting θ and x, like we did with perceptrons…

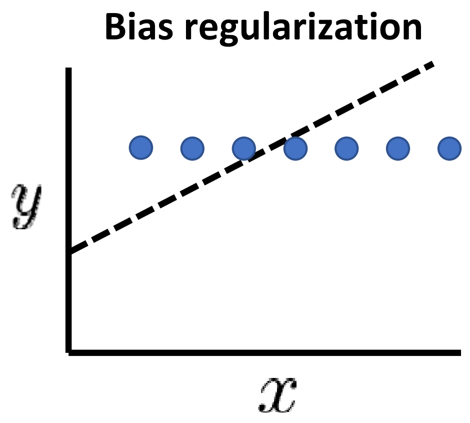
$$\theta' = [\theta_1, \theta_2, \ldots, \theta_d, b],\ x' = [x_1, x_2, \ldots, x_d, 1]$$

- What happens if we regularize the bias term?

Regularization and offset (aka bias)

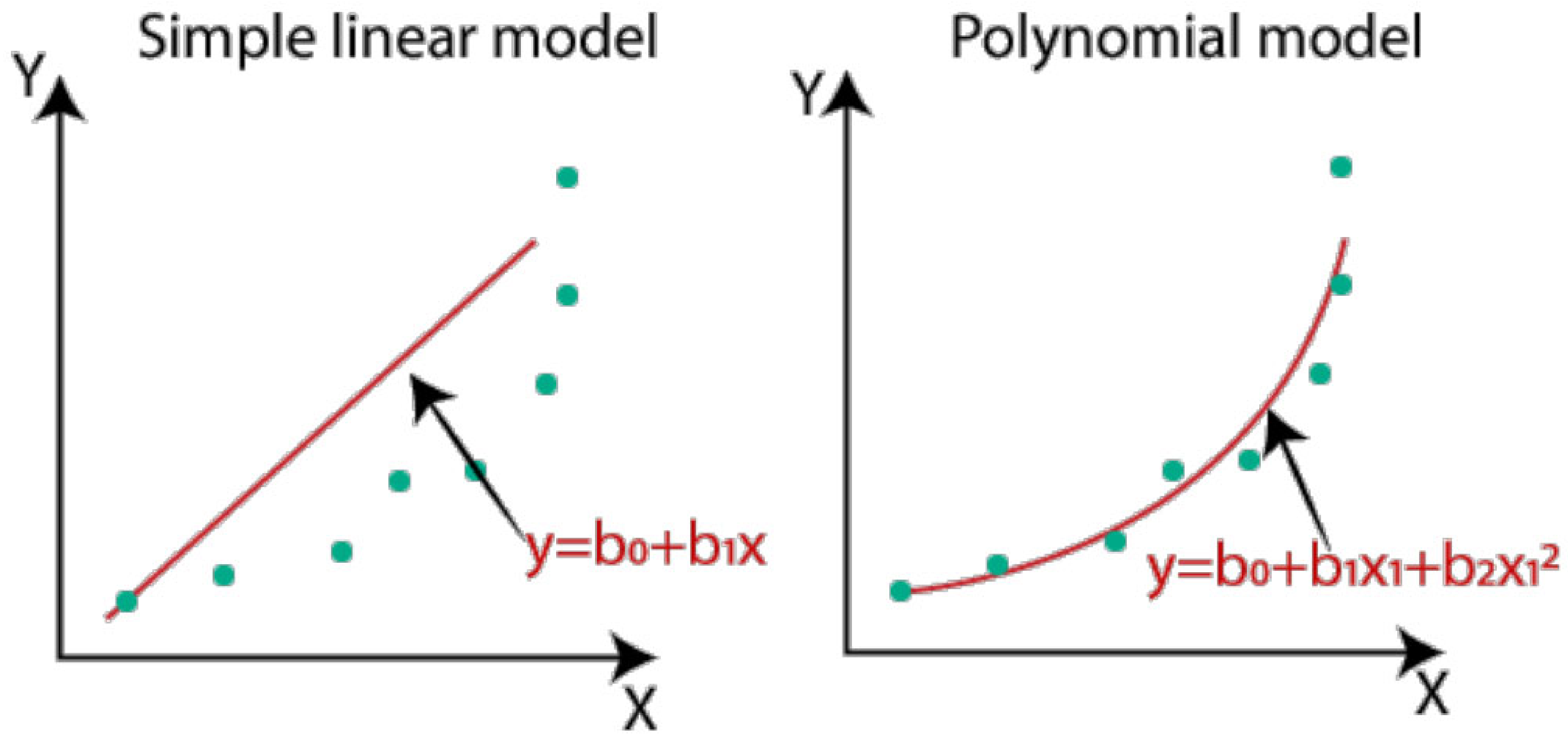
- Recall that “regularizing” a model parameter means encouraging that model parameter to tend towards 0.
- How would a linear model represent horizontal line?
- How does shrinking the bias affect its ability to do so?



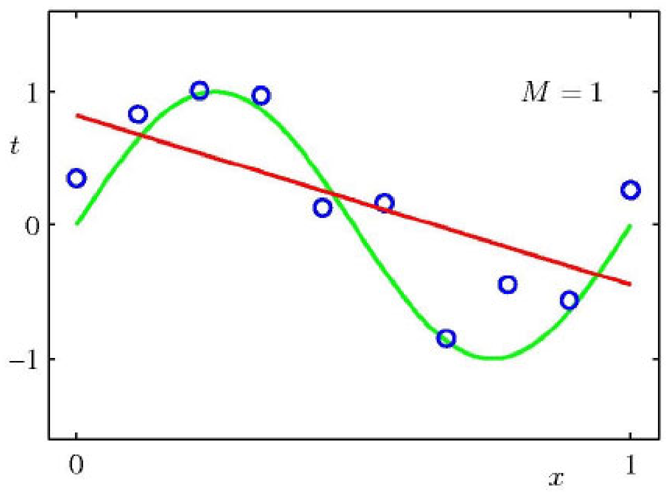


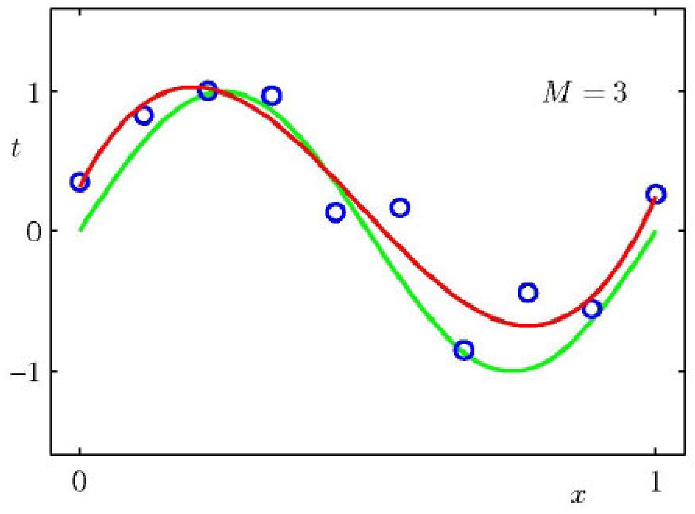
Don’t regularize the bias term!

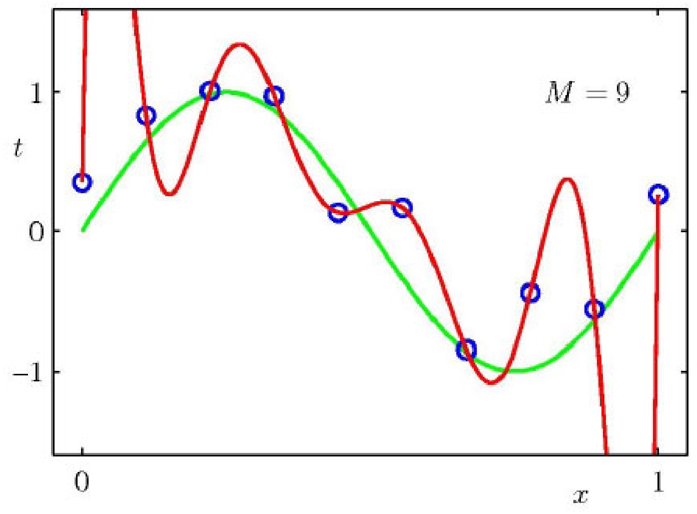
Polynomial Regression



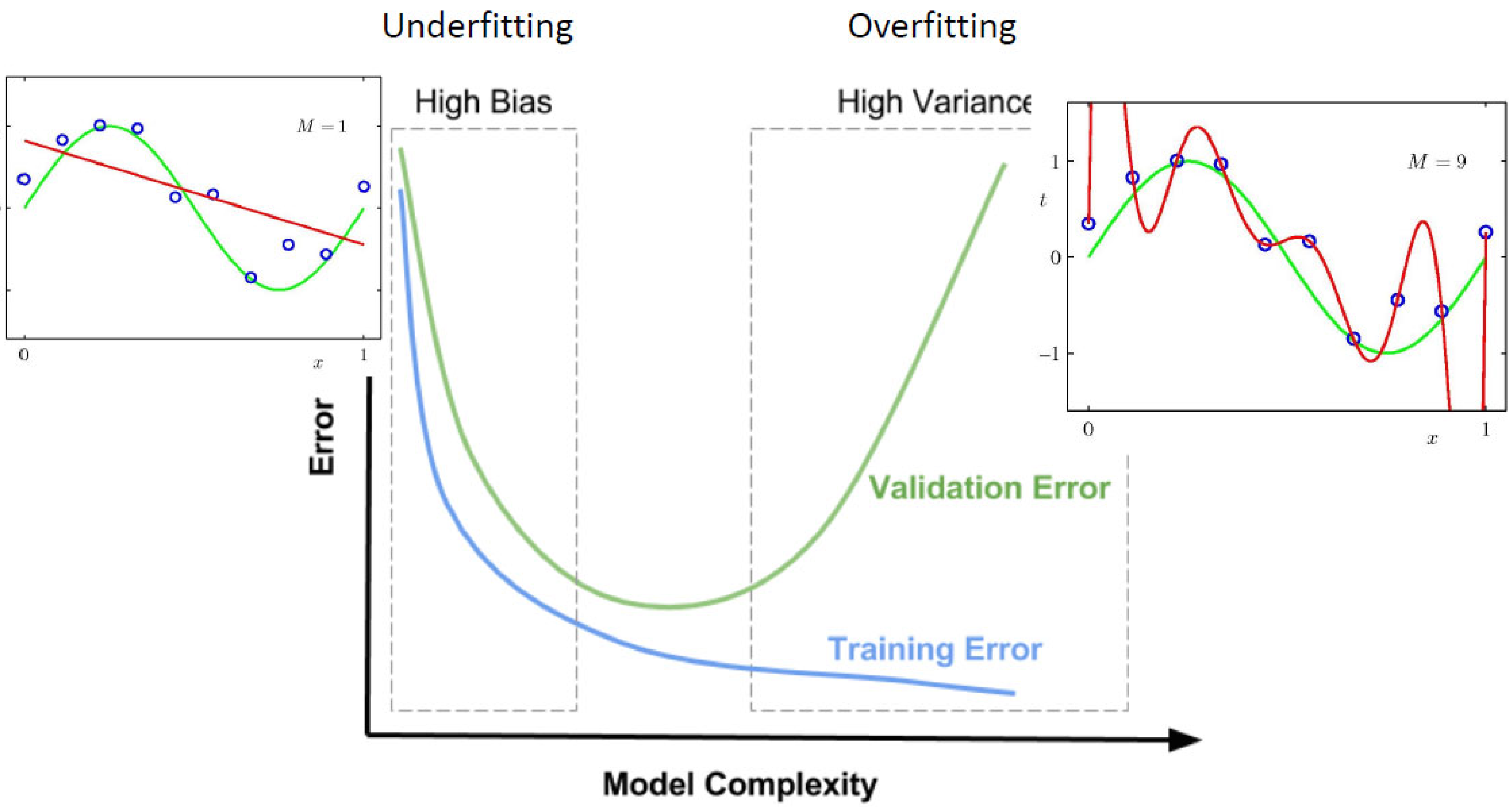
Under/Over Fitting







Bias/Variance Trade-off

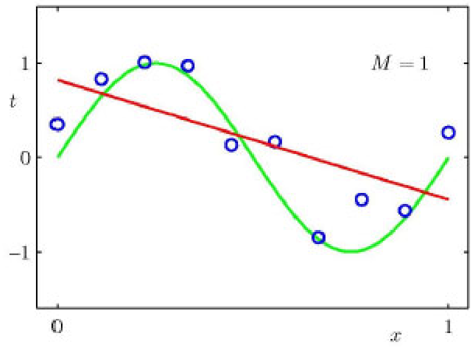


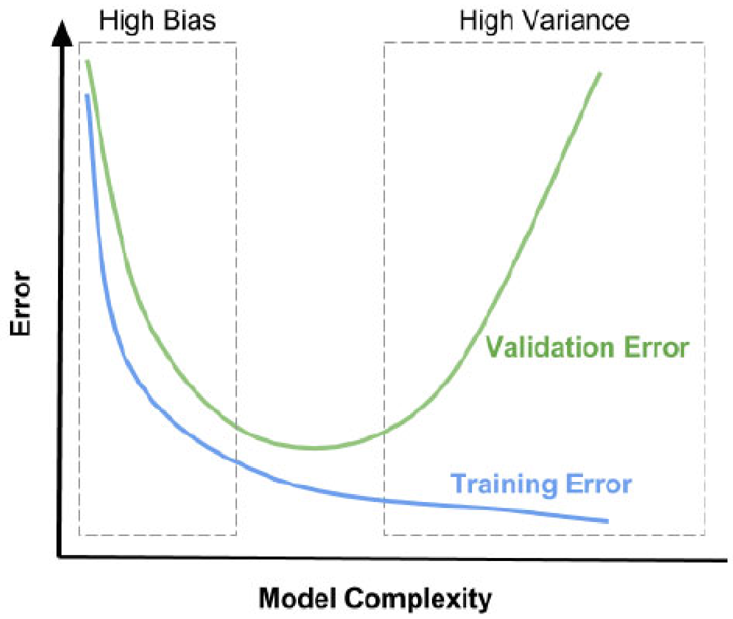
What Goes Wrong?

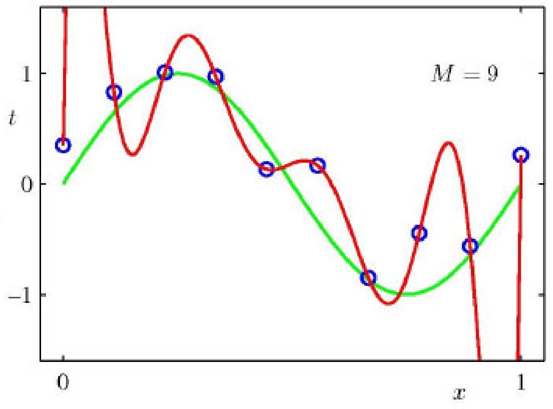
- Underfit: too many assumptions
  - Hypothesis class doesn’t contain the optimal hypothesis
    - (or even a “good” hypothesis)
  - Data representation discards essential information

- Overfit: not enough assumptions
  - Hypothesis class is “hard to search”
  - Learning algorithm is inefficient, can’t optimize parameters
  - Many hypotheses perfectly fit training data

How Do We Stop Overfitting



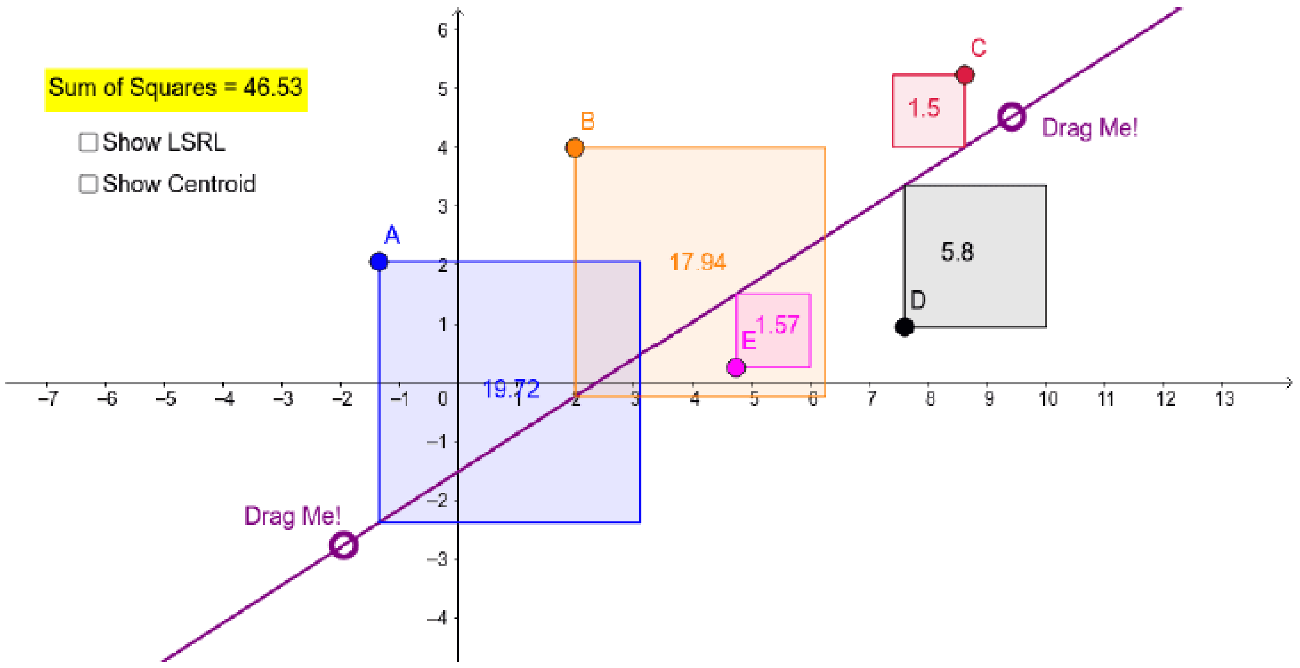




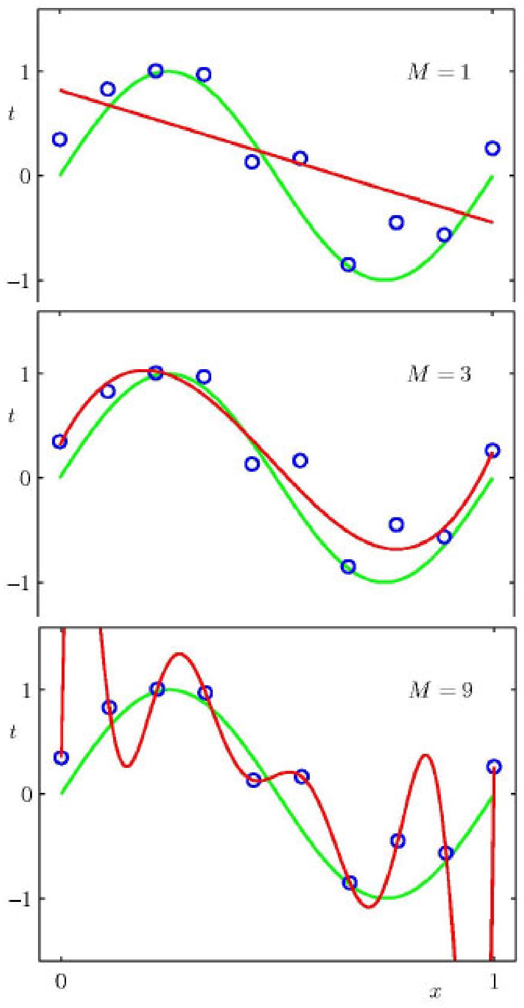
Motivating Regularization

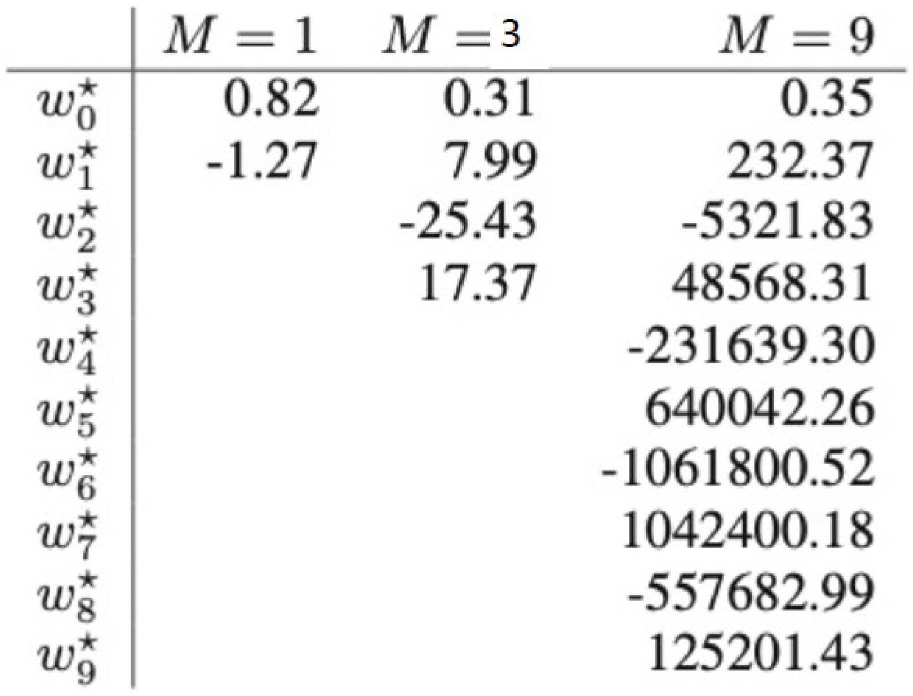
- Our predictor f with parameters w has loss L:

$$L(y,\ f(x;\ w)) = (y - w \cdot x)^2$$



Motivating Regularization



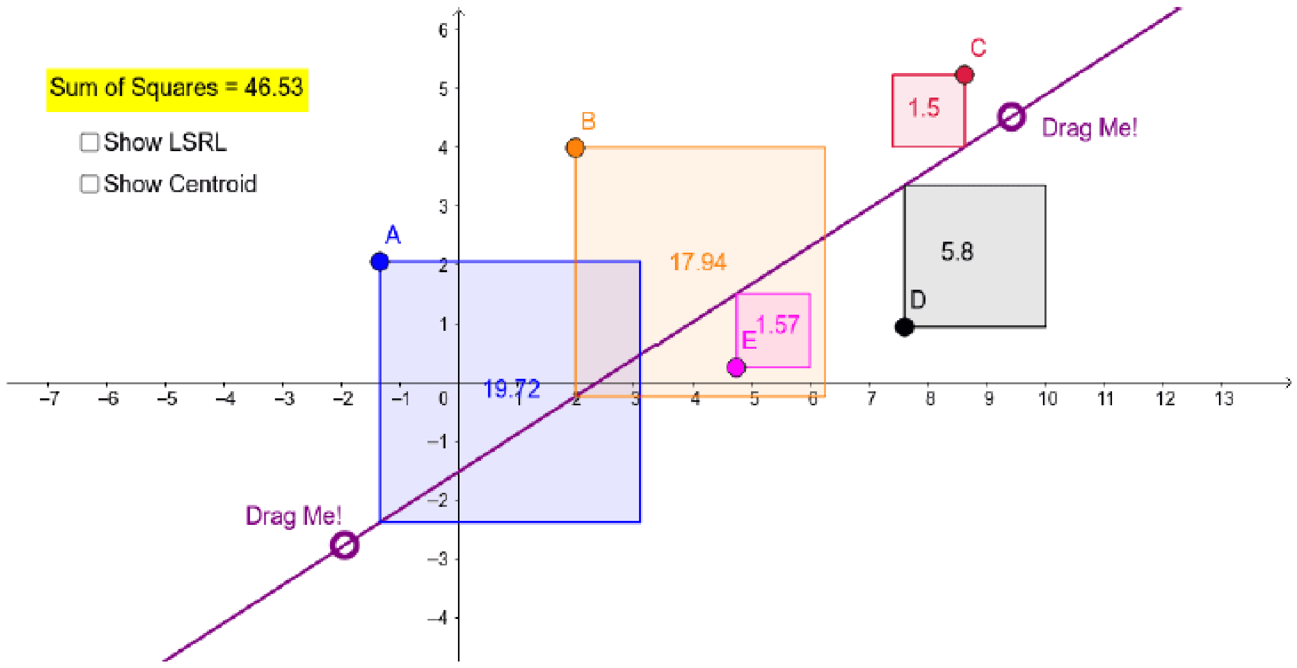


	$M = 1$	$M = 3$	$M = 9$
$w_0^\star$	0.82	0.31	0.35
$w_1^\star$	-1.27	7.99	232.37
$w_2^\star$		-25.43	-5321.83
$w_3^\star$		17.37	48568.31
$w_4^\star$			-231639.30
$w_5^\star$			640042.26
$w_6^\star$			-1061800.52
$w_7^\star$			1042400.18
$w_8^\star$			-557682.99
$w_9^\star$			125201.43

Motivating Regularization

- What if we add a term to our loss?

$$L(y,\ f(x;\ w)) = (y - w \cdot x)^2 + \lambda w^2$$



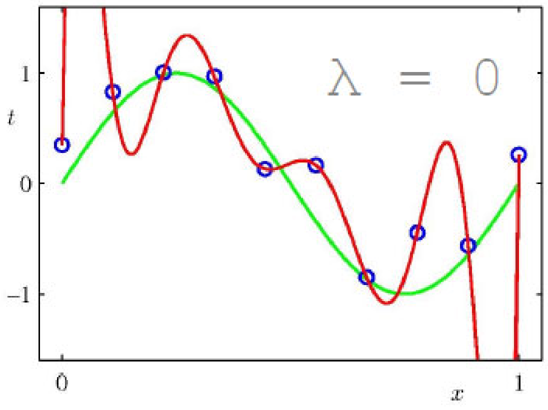
How Do We Stop Overfitting?

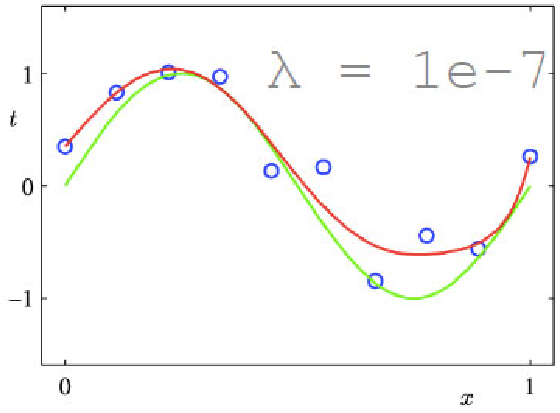
$$L(y, f(x; w)) = (y - w \cdot x)^2 + \lambda w^2$$

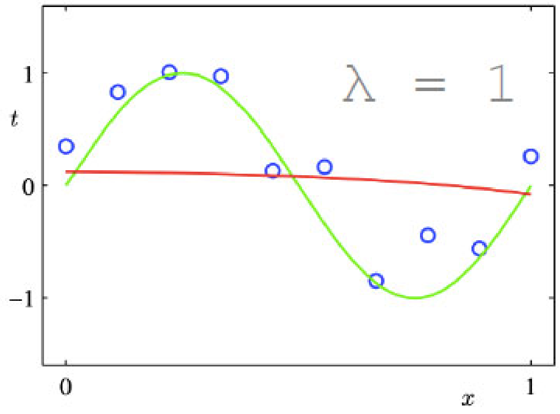
	$\lambda = 0$	1e-7	1
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

How Do We Stop Overfitting?

$$L(y, f(x; w)) = (y - w \cdot x)^2 + \lambda w^2$$







Regularized Least Squares

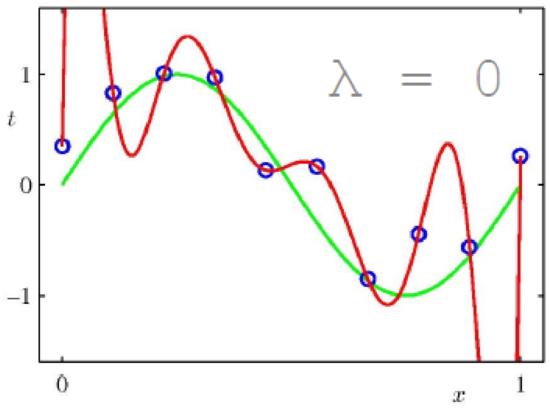
$$\mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\{y_i - \mathbf{w}^T \cdot \mathbf{x}_i\}^2 \qquad \mathcal{L}(\mathbf{w}) = \frac{1}{2}\sum_{i=1}^{N}\{y_i - \mathbf{w}^T \cdot \mathbf{x}_i\}^2 + \lambda\frac{1}{2}\mathbf{w}^T\mathbf{w}$$

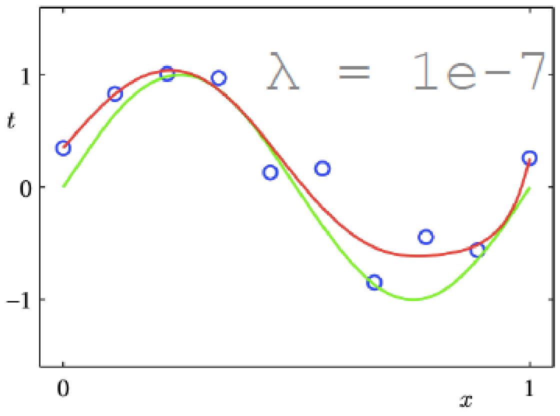
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y} \qquad \mathbf{w} = \left(\lambda\mathbf{I} + \mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{Y}$$

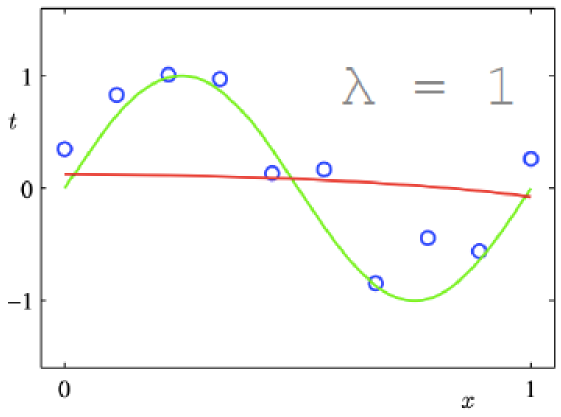
How Do We Stop Overfitting?

$$L(y, f(x; w)) = (y - w \bullet x)^2 + \lambda w^2$$

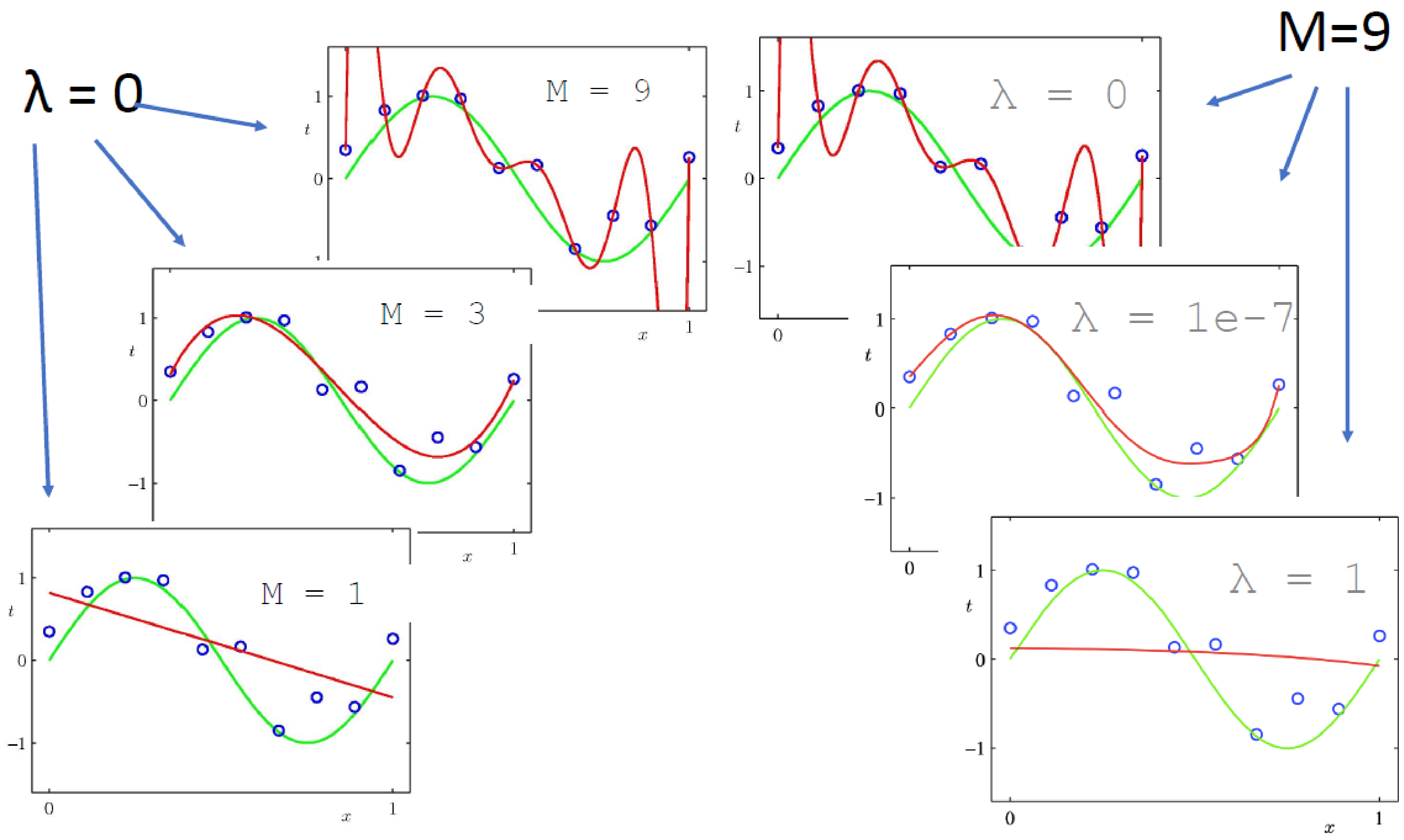
$$\mathbf{w} = (\lambda \mathbf{I} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{Y}$$







Summary of Regularization


λ = 0
M=9
M = 9
λ = 0
M = 3
λ = 1e-7
M = 1
λ = 1
t
x
0
1
−1