DS 4400

Machine Learning and Data Mining I
Spring 2024

David Liu
Khoury College of Computer Science
Northeastern University

Friday January 26, 2024

Today's Outline

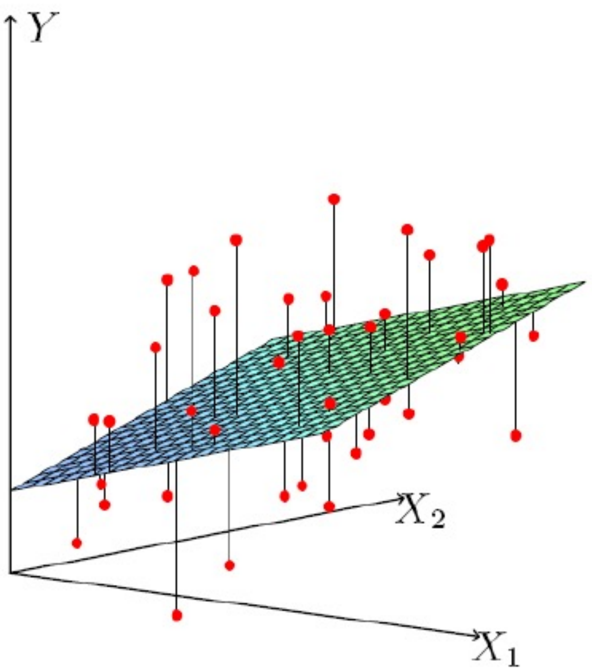
- Gradient descent
  - General optimization algorithm
  - Instantiation for linear regression
  - Issues with gradient descent
  - Comparison with closed-form solution
- Non-linear regression
  - Polynomial regression
  - Cubic, spline regression

Multiple Linear Regression

- Dataset: $x_i \in R^d, y_i \in R$
- Hypothesis $h_\theta(x) = \theta^T x$
- MSE $= \frac{1}{N}\sum (\theta^T x_i - y_i)^2$ Loss / cost

$$\theta = (X^\top X)^{-1} X^\top y$$

Closed-form optimum solution for linear regression



Recap Linear Regression

- Optimal solution to min MSE
  - Use vectorization for compact representation
- Advantages of linear regression
  - Simplicity and interpretability
  - Closed-form optimal solution (depends uniquely on training data)
- Limitations of linear regression
  - Small capacity in number of parameters (d+1)
  - Does not fit well non-linear data
- Practical issues
  - Feature standardization
  - Outliers
  - Categorical features

Practical issues: Categorical features

- Predict credit card balance
  - Age
  - Income
  - Number of cards
  - Credit limit
  - Credit rating
- Categorical variables
  - Student (Yes/No)
  - State (50 different levels)

How to generate numerical representations of these?

Indicator Variables

- One-hot encoding
- Binary (two-level) variable
  - Add new feature $x_j = 1$ if student and 0 otherwise
- Multi-level variable
  - State: 50 values
  - $x_{MA} = 1$ if State $=$ MA and $0,$ otherwise
  - $x_{NY} = 1$ if State $=$ NY and $0,$ otherwise
  - …
  - How many indicator variables are needed?
- Disadvantages: data becomes too sparse for large number of levels
  - Will discuss feature selection later in class

Training phase of most ML

- Input: labeled data
- Define objective / loss metric
  - MSE for regression
  - Specific loss functions for classification
- Run an optimization procedure to minimize loss (error) on training data
- Output: trained model that best fits the training data

How to optimize loss functions?

- Dataset: $x_i \in R^d, y_i \in R$
- Hypothesis $h_\theta(x) = \theta^T x$
- $J(\theta) = \frac{1}{N} \sum (\theta^T x_i - y_i)^2$ Loss / cost for regression
- General method to optimize a multi-variate function
  - Practical and efficient
  - Generally applicable to different loss functions
  - Convergence guarantees for certain loss functions (e.g., convex)

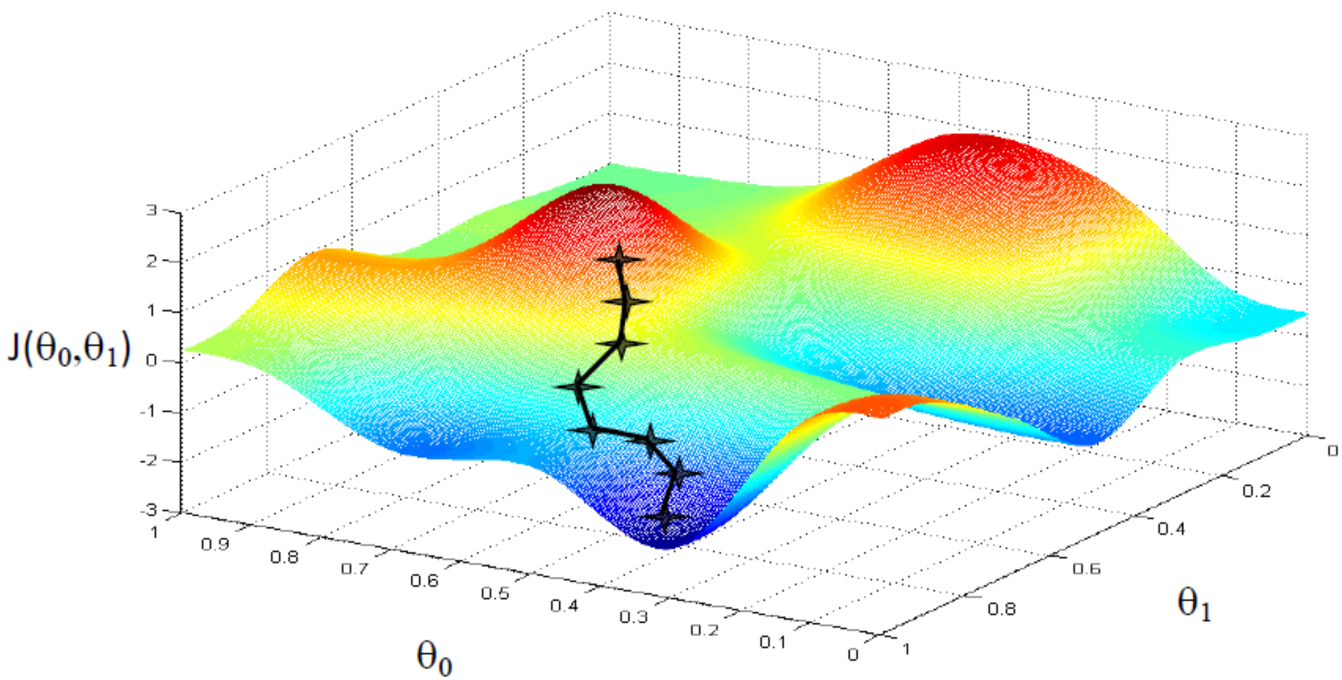
What Strategy to Use?

Follow the Slope

Follow the direction of steepest descent!

How to optimize $J(\theta)$?

- Choose initial value for θ
- Until we reach a minimum:
  - Choose a new value for θ to reduce $J(\theta)$



Gradient Descent

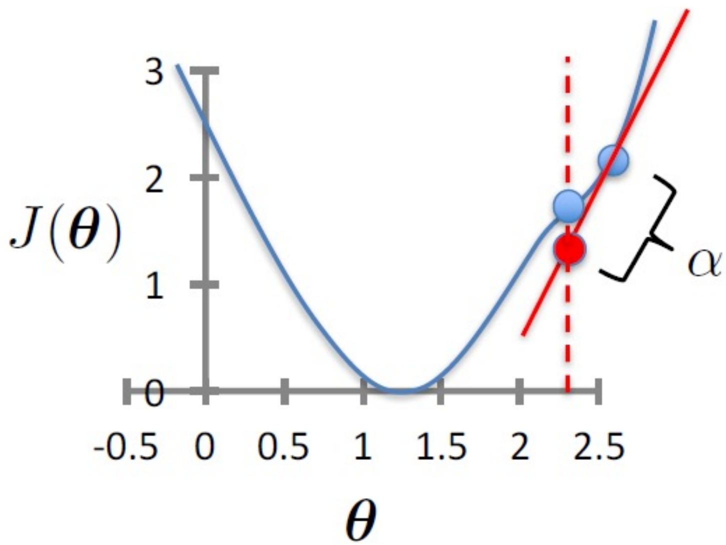
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

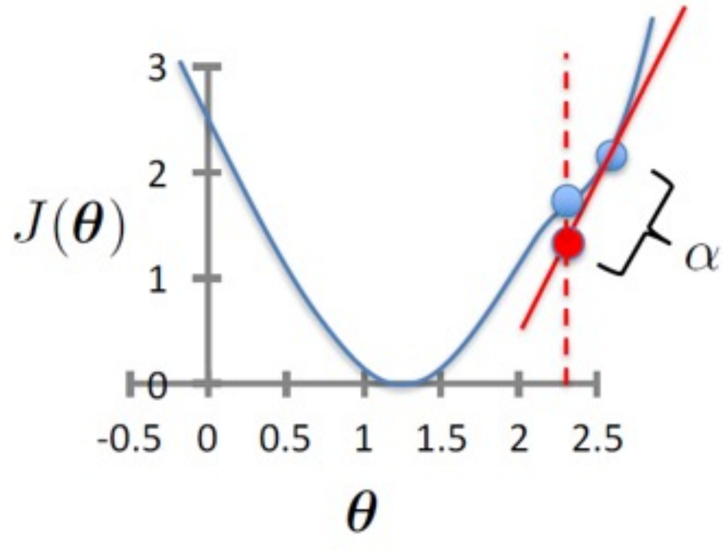
$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

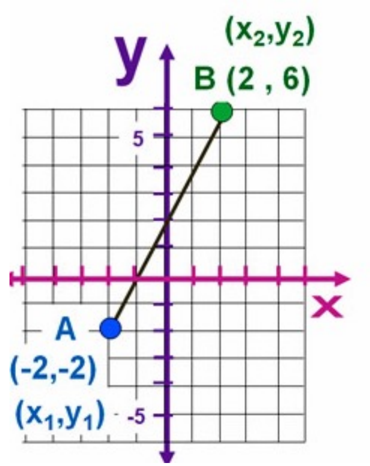
simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05

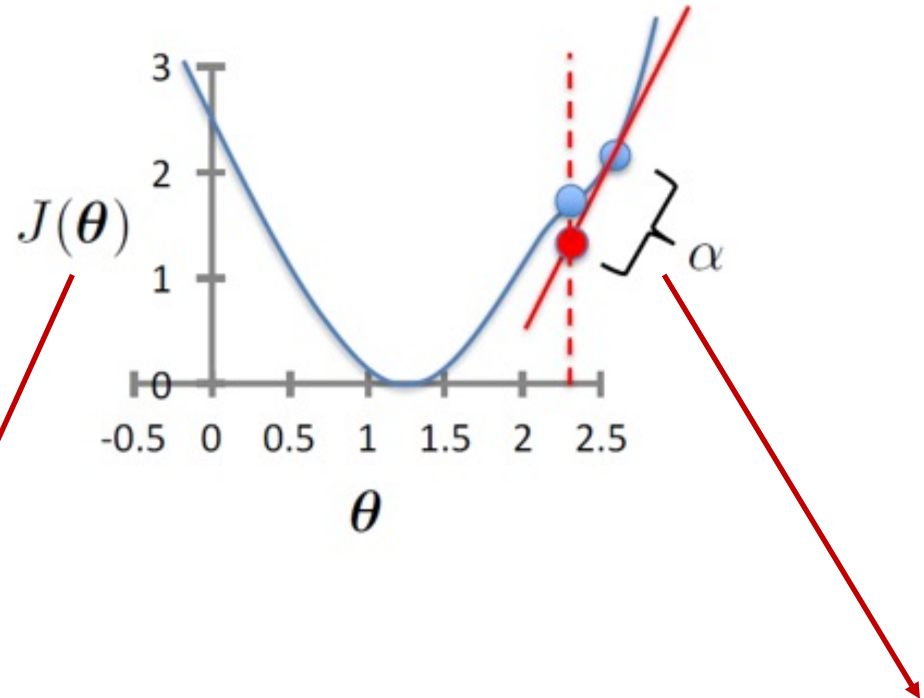


Gradient Descent

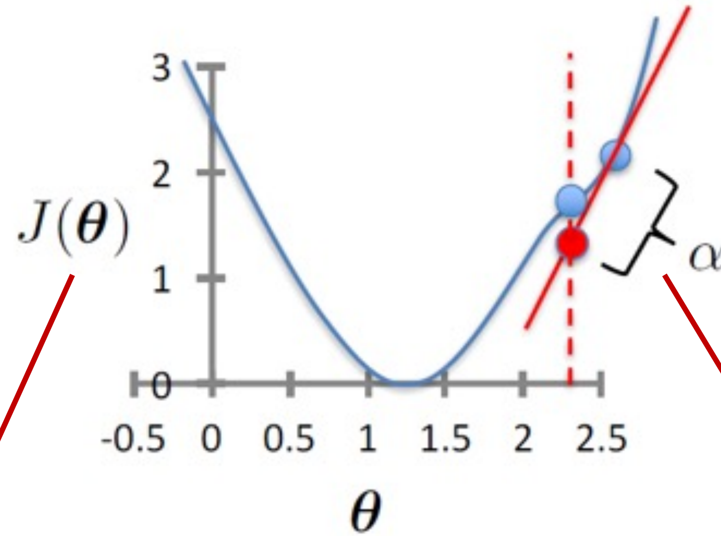




Gradient Descent



Gradient Descent



- If θ is on the left of minimum, slope is negative
- Increase value of θ

- If θ is on the right of minimum, slope is positive
- Decrease value of θ

In both cases θ gets closer to minimum

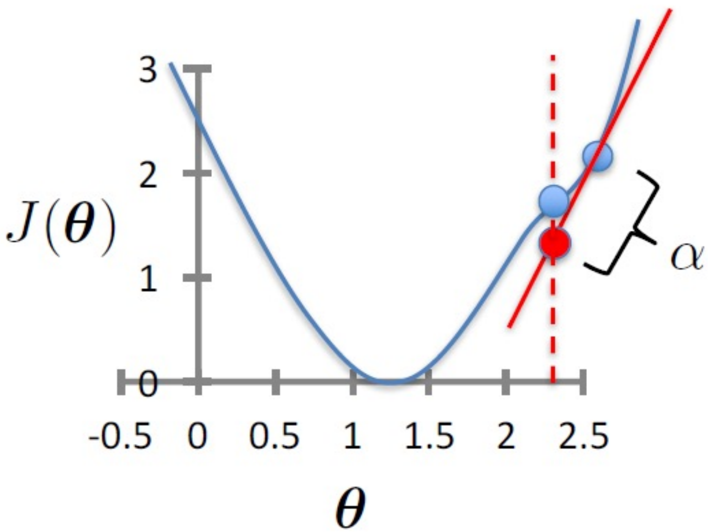
Stopping Condition

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

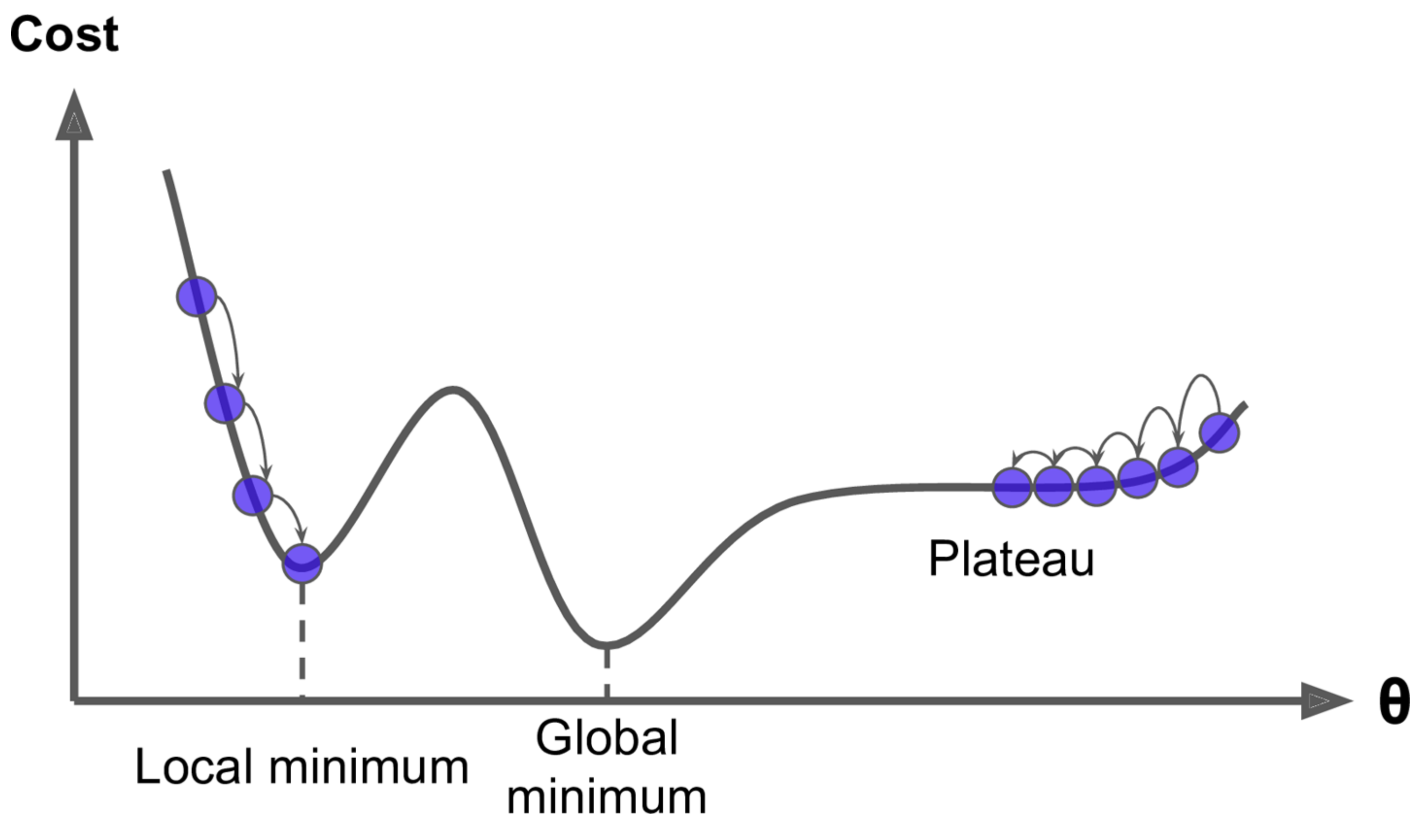
simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



- When should the algorithm stop?

GD Convergence Issues



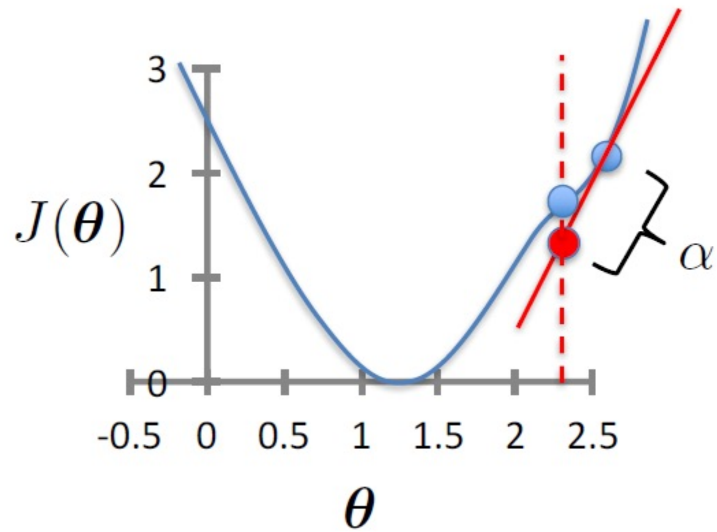
Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

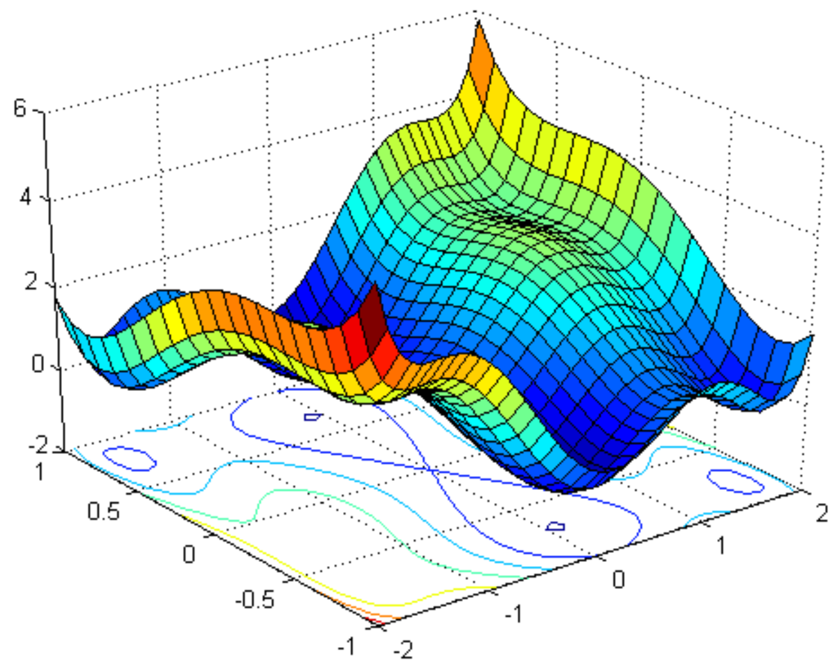
simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



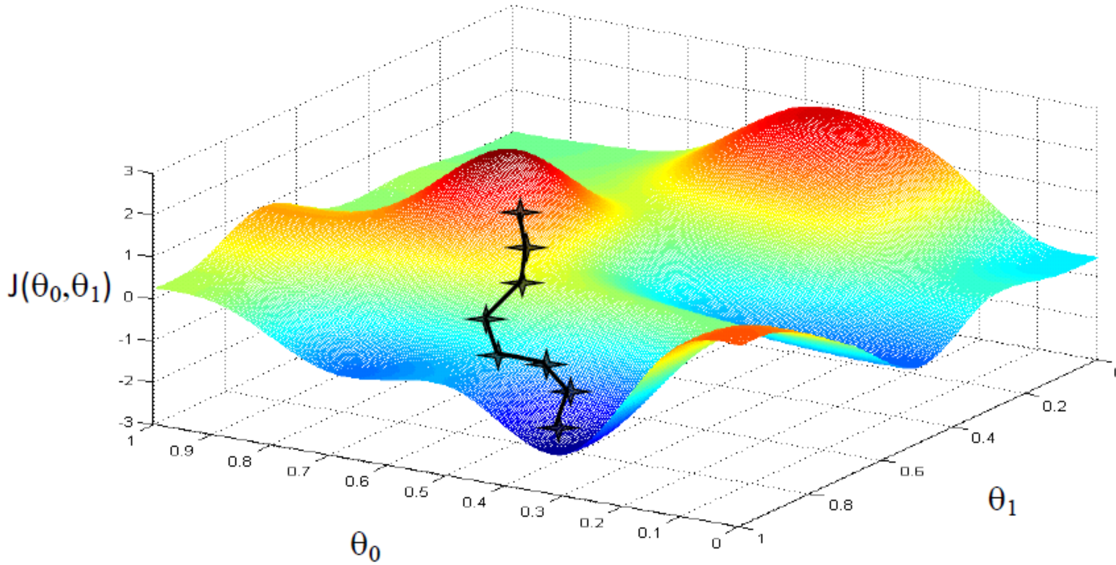
- What happens when θ reaches a local minimum?

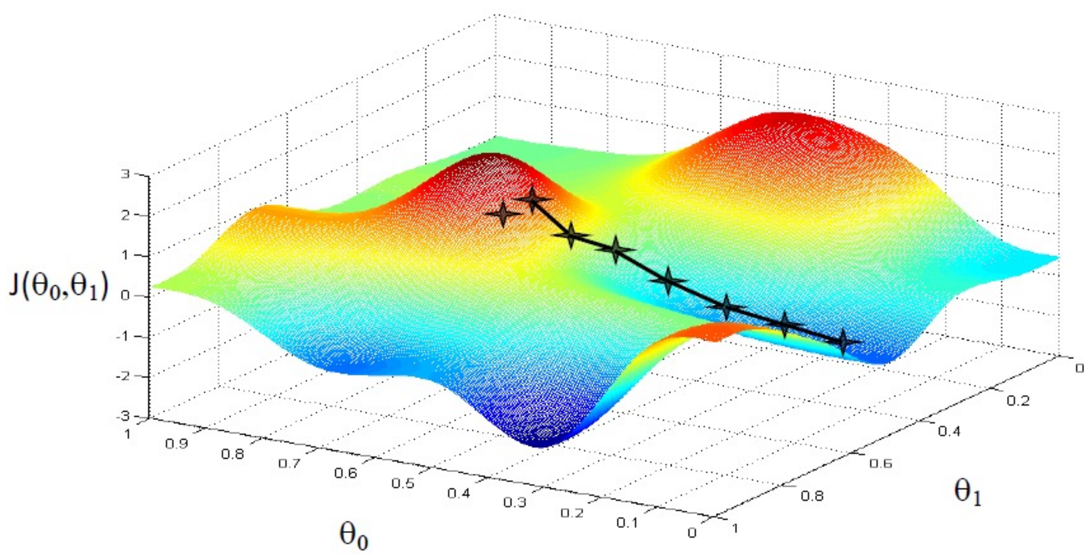
Complex loss function



- Convex loss functions only have global minimum, no local minima
- Complex loss functions are more difficult to optimize as they have multiple local optima

Possible Solution



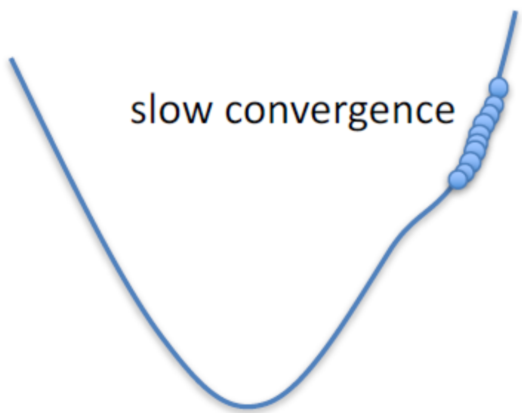


Adjusting Learning Rate

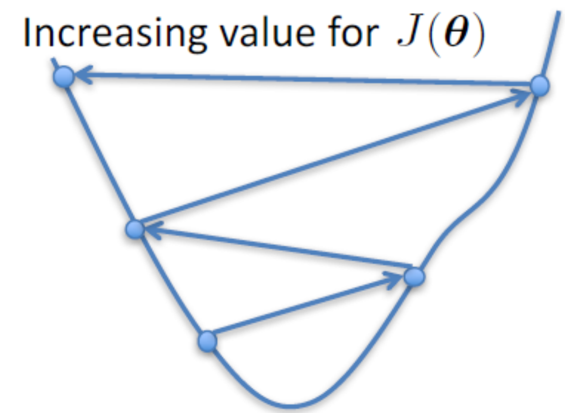
α too small	α too large

Choosing Learning Rate

α too small



α too large

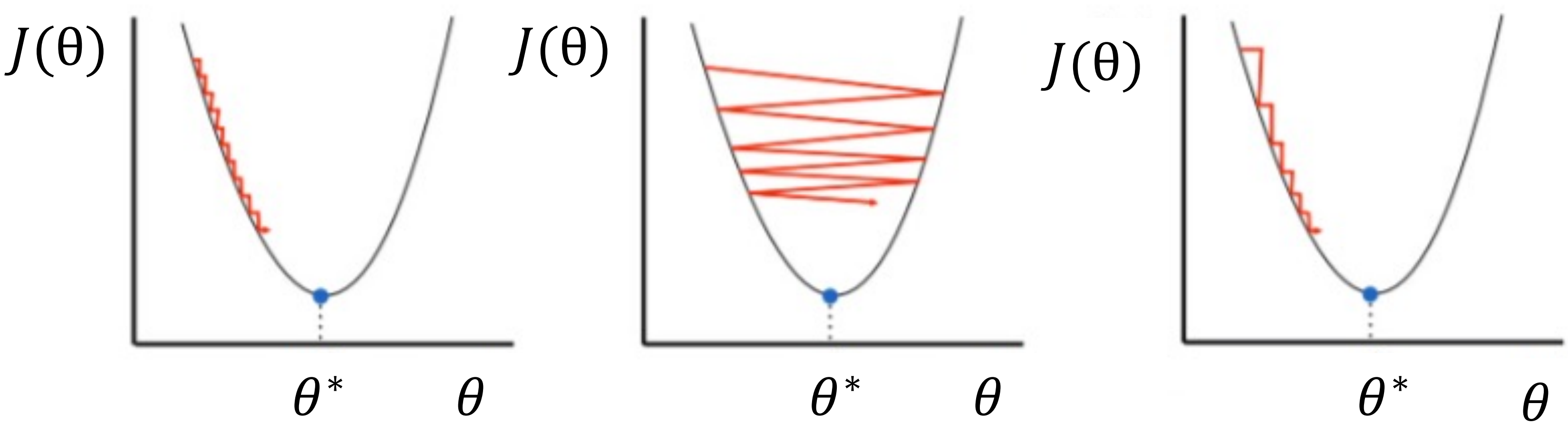


- May overshoot the minimum
- May fail to converge
- May even diverge

To see if gradient descent is working, print out $J(\boldsymbol{\theta})$ each iteration

- The value should decrease at each iteration
- If it doesn't, adjust α

Adaptive step size

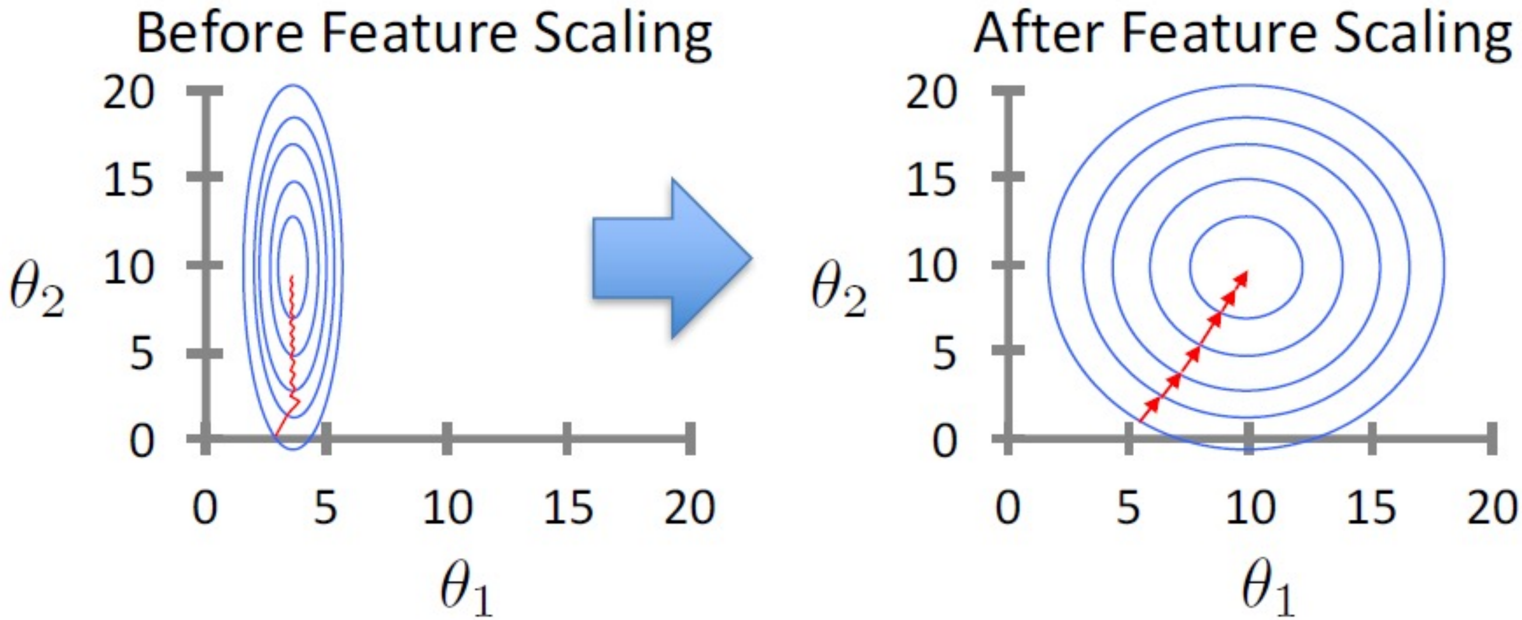


(a) Step-size too small (b) Step-size too big (c) Adaptive step-size

- Start with large step size and reduce over time, adaptively
- Line search method
- Measure how objective decreases

Feature Scaling

- **Idea:** Ensure that feature have similar scales



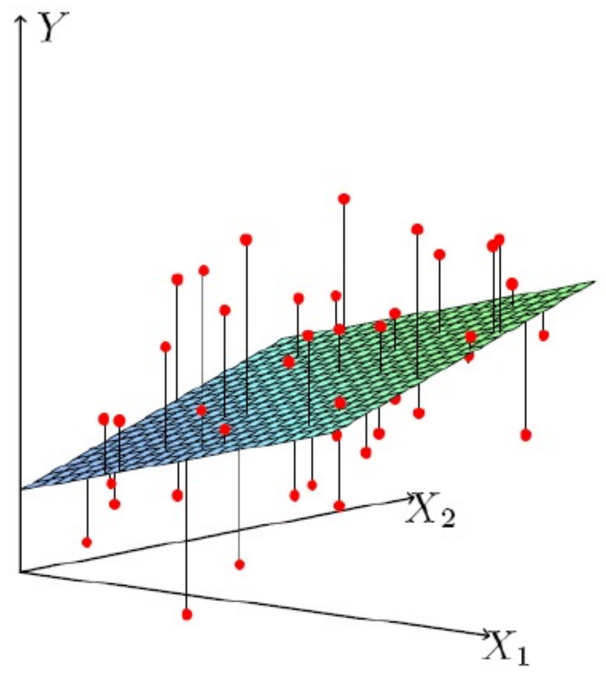
- Makes gradient descent converge *much* faster

Multiple Linear Regression

- Dataset: $x_i \in R^d, y_i \in R$
- Hypothesis $h_\theta(x) = \theta^T x$
- MSE $= \frac{1}{N}\sum (h_\theta(x_i) - y_i)^2$ Loss / cost

$$\theta = (X^\intercal X)^{-1} X^\intercal y$$

MSE is a strictly convex function and has unique minimum



GD for Multiple Linear Regression

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update
for j = 0 ... d

GD for Linear Regression

- Initialize θ
- Repeat until convergence

$||\theta_{new} - \theta_{old}|| < \epsilon$ or
iterations == MAX_ITER

$$\theta \leftarrow \theta - \alpha \frac{2}{N}(X\theta - y)^T X$$

Equivalent

$$\theta_j \leftarrow \theta_j - \alpha \frac{2}{N}\sum_{i=1}^{N}(h_\theta(x_i) - y_i)x_{ij}\,, j = 0, \ldots, d$$

- Assume convergence when $\|\boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old}\|_2 < \epsilon$

L_2 norm: $\|\boldsymbol{v}\|_2 = \sqrt{\sum_i v_i^2} = \sqrt{v_1^2 + v_2^2 + \ldots + v_{|v|}^2}$

Gradient Descent in Practice

- Asymptotic complexity
  - $O(NTd)$, *N* is size of training data, *d* is feature dimension, and *T* is number of iterations
- Most popular optimization algorithm in use today
- At the basis of training
  - Linear Regression
  - Logistic regression
  - SVM
  - Neural networks and Deep learning
  - Stochastic Gradient Descent variants

Gradient Descent vs Closed Form

Gradient Descent

- Initialize $\boldsymbol{\theta}$
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update for j = 0 ... d

Closed form solution for LR

$$\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

• Gradient Descent	• Closed Form

Gradient Descent vs Closed Form

Gradient Descent

- Initialize θ
- Repeat until convergence

$$\theta_j \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$$

simultaneous update for j = 0 ... d

Closed form

$$\boldsymbol{\theta} = (\boldsymbol{X}^\intercal \boldsymbol{X})^{-1} \boldsymbol{X}^\intercal \boldsymbol{y}$$

• Gradient Descent	• Closed Form
+ Linear increase in d and N	+ No parameter tuning
+ Generally applicable	+ Gives the global optimum
- Need to choose α and stopping conditions	- Not generally applicable
- Might get stuck in local optima	- Slow computation: $O(d^3)$

Issues with Gradient Descent

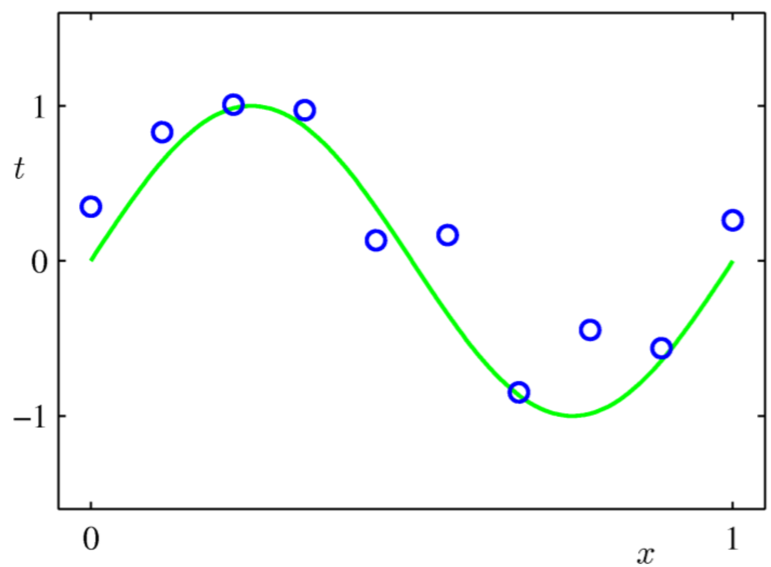
- Might get stuck in local optimum and not converge to global optimum
  - Restart from multiple initial points
- Only works with differentiable loss functions
- Small or large gradients
  - Feature scaling helps
- Tune learning rate
  - Can use line search for determining optimal learning rate

Review Gradient Descent

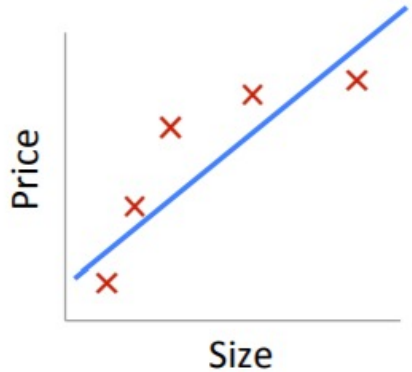
- Gradient descent is an efficient algorithm for optimization and training ML models
  - The most widely used algorithm in ML!
  - Faster than using closed-form solution for linear regression
  - Main issues with Gradient Descent is convergence and getting stuck in local optima (for neural networks)
- Gradient descent is guaranteed to converge to optimum for strictly convex functions if run long enough

Polynomial Regression

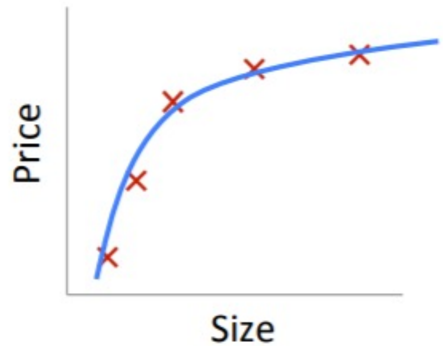
- Polynomial function on single feature
  - $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_p x^p$



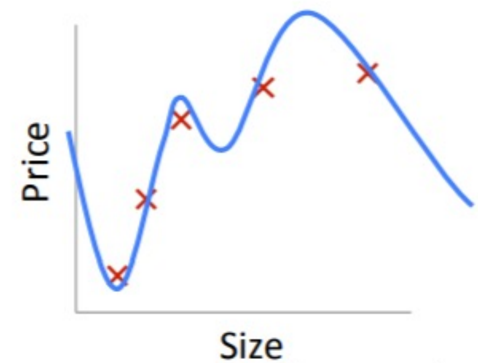
Polynomial Regression



$\theta_0 + \theta_1 x$

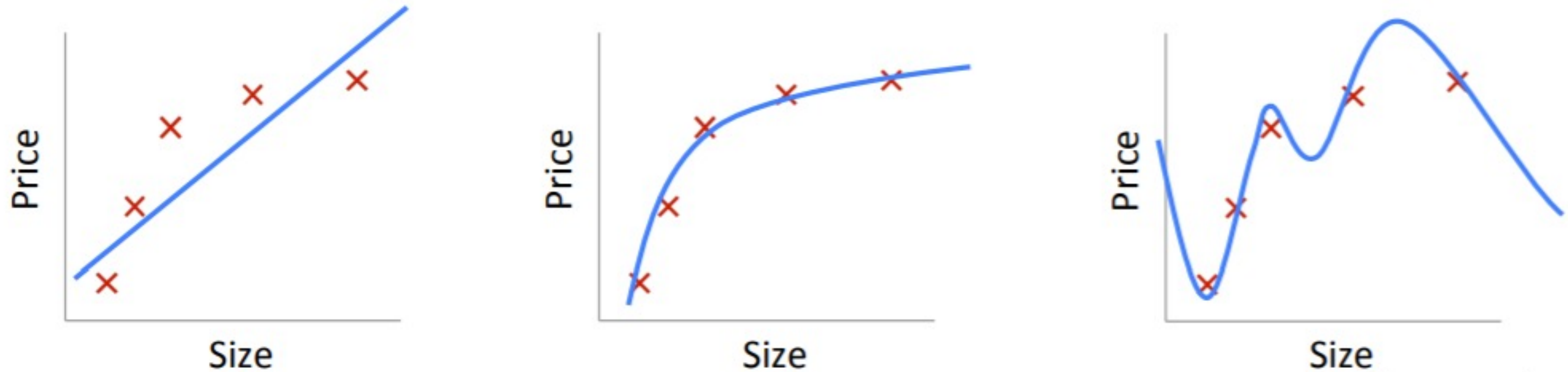


$\theta_0 + \theta_1 x + \theta_2 x^2$



$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

Polynomial Regression



Underfitting (high bias)	Correct fit	Overfitting (high variance)
$\theta_0 + \theta_1 x$	$\theta_0 + \theta_1 x + \theta_2 x^2$	$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3 + \theta_4 x^4$

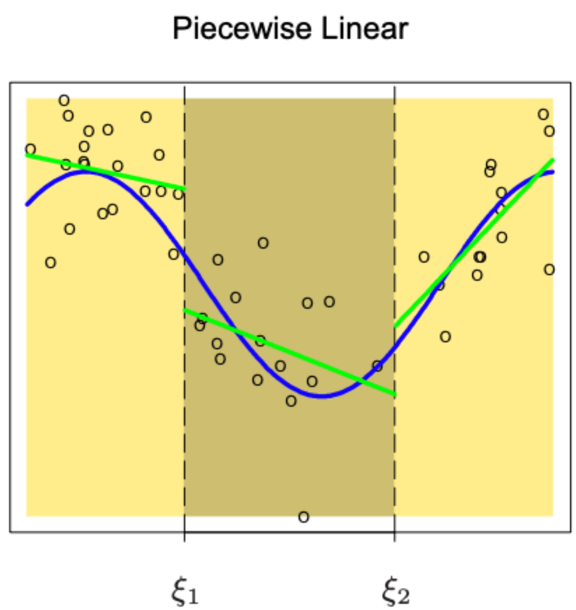
- Typically to avoid overfitting $d \leq 4$

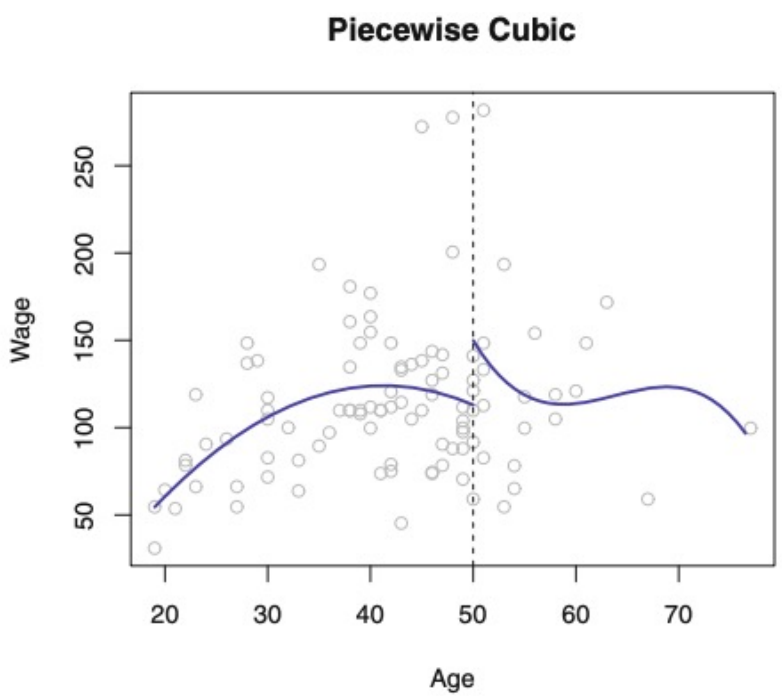
Polynomial Regression Training

- Simple Linear Regression
- $h_\theta(x) = \theta_0 + \theta_1 x + \cdots + \theta_p x^p$
- How to train model?

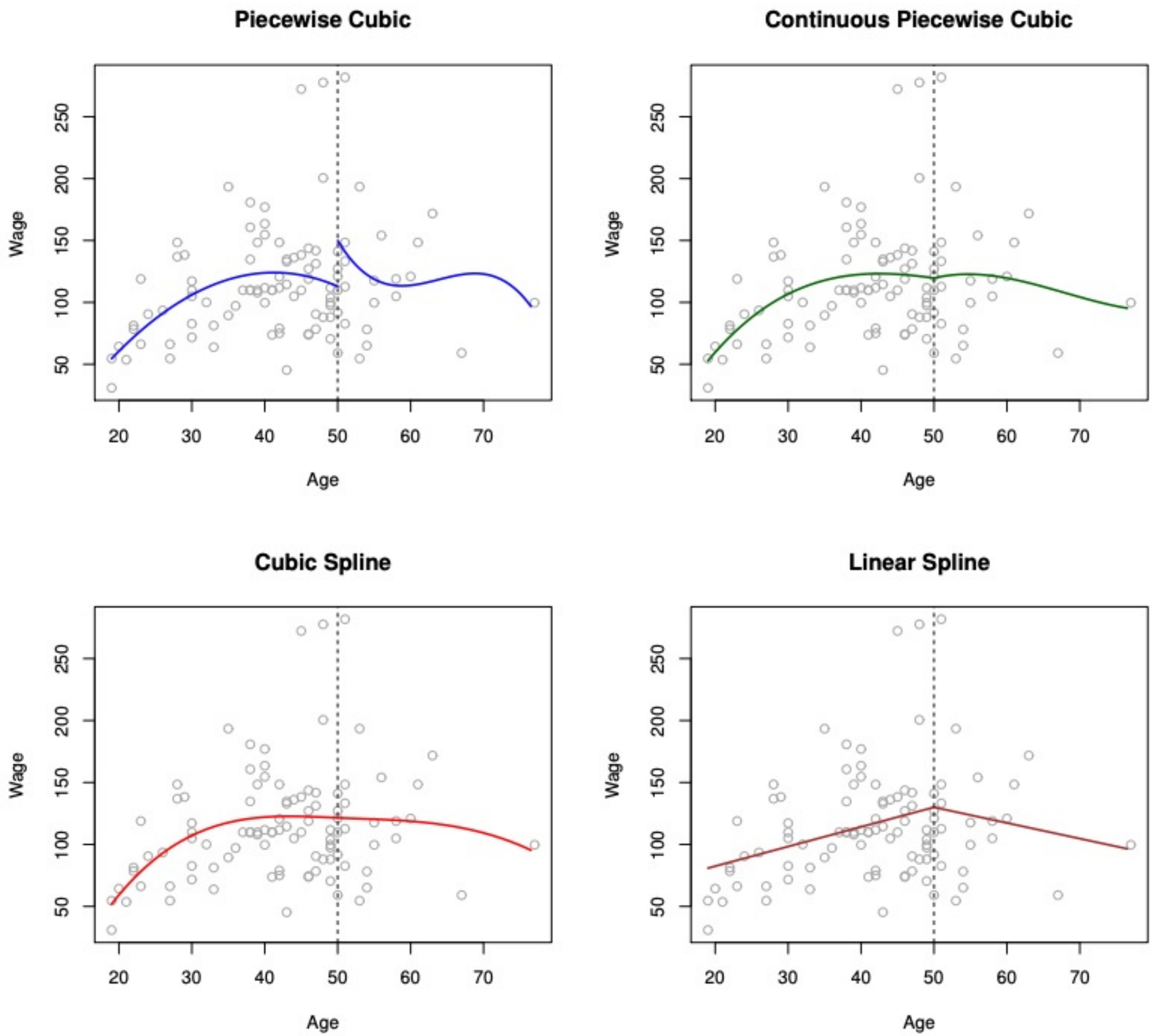
Piecewise Polynomial

- Divide the space into regions
- Polynomial regression on each region
  - Linear piecewise (degree 1), quadratic piecewise (degree 2), cubic piecewise (degree 3)

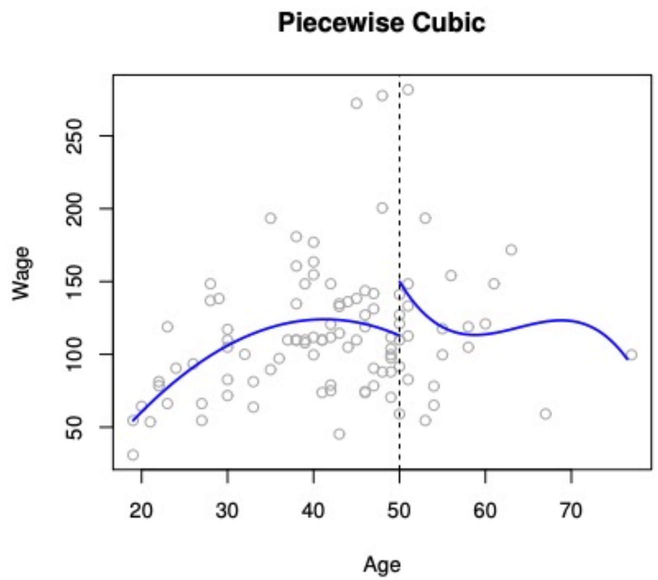




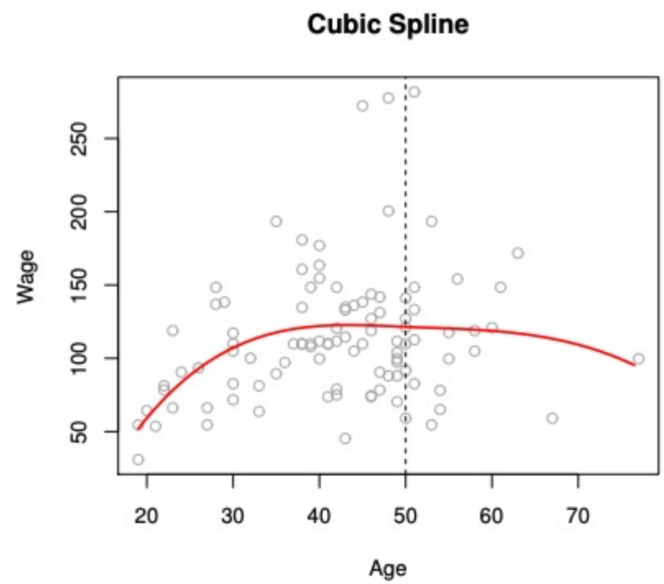
Piecewise and spline regression



Piecewise polynomial vs Regression spline



1 break at Age $= 50$

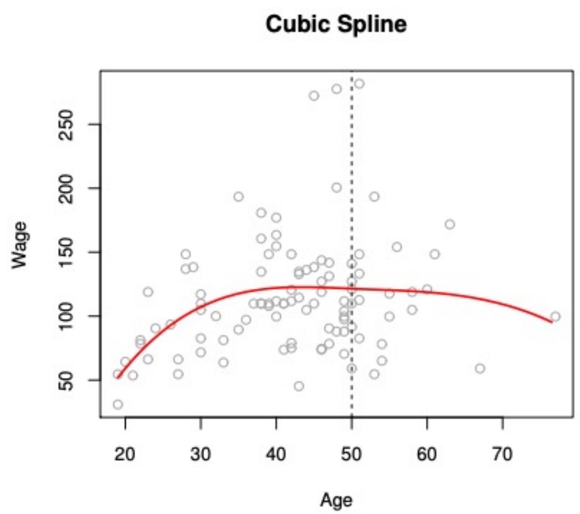


1 knot at Age $= 50$

Definition: Cubic spline

A cubic spline with knots at x-values $\xi_1, \ldots, \xi_K$ is a continuous piecewise cubic polynomial with *continuous derivates* and *continuous second derivatives* at each knot.

Cubic splines



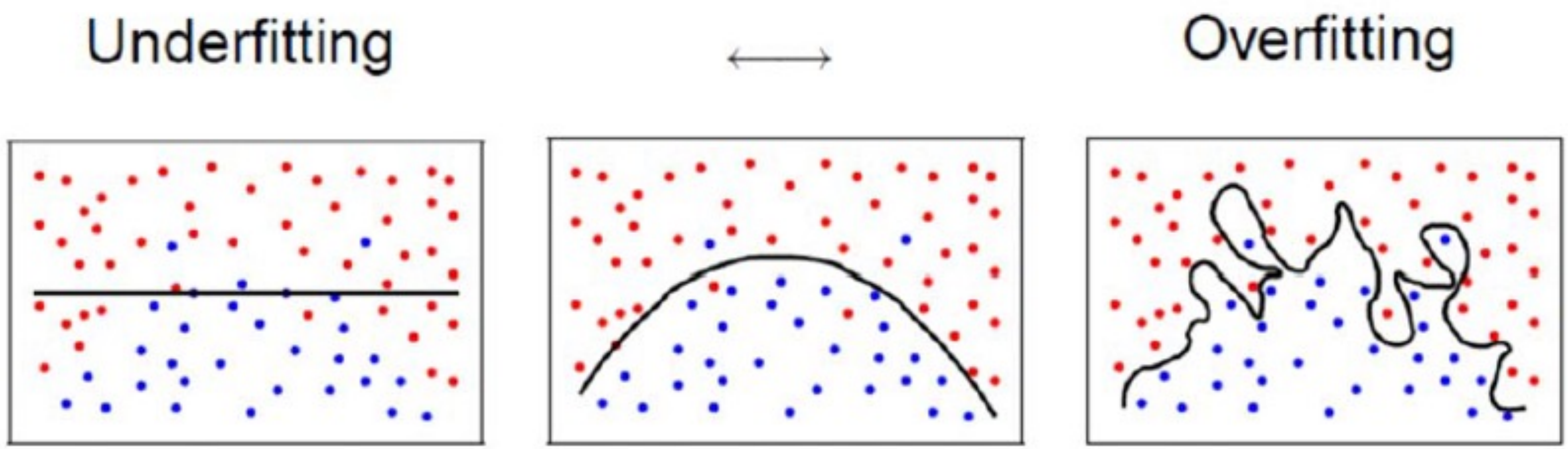
- Turns out, cubic splines are sufficiently flexible to *consistently* estimate smooth regression functions f
- You can use higher-degree splines, but *there's no need to*
- To fit a cubic spline, we just need to pick the knots

A cubic spline with K knots has K+4 parameters

Additive Models

- Multiple Linear Regression Model
  - $y_i = \theta_0 + \theta_1 x_1 + \cdots + \theta_d x_d$
- Additive Models
  - $y_i = \theta_0 + f_1(x_1) + \cdots + f_d(x_d)$
- Can instantiate functions f with:
  - Linear functions: $f_i(x_i) = \theta_i x_i$
  - Quadratic: $f_i(x_i) = \theta_i^1 x_i + \theta_i^2 x_i^2$
  - Cubic: $f_i(x_i) = \theta_i^1 x_i + \theta_i^2 x_i^2 + \theta_i^3 x_i^3$

Generalization in ML

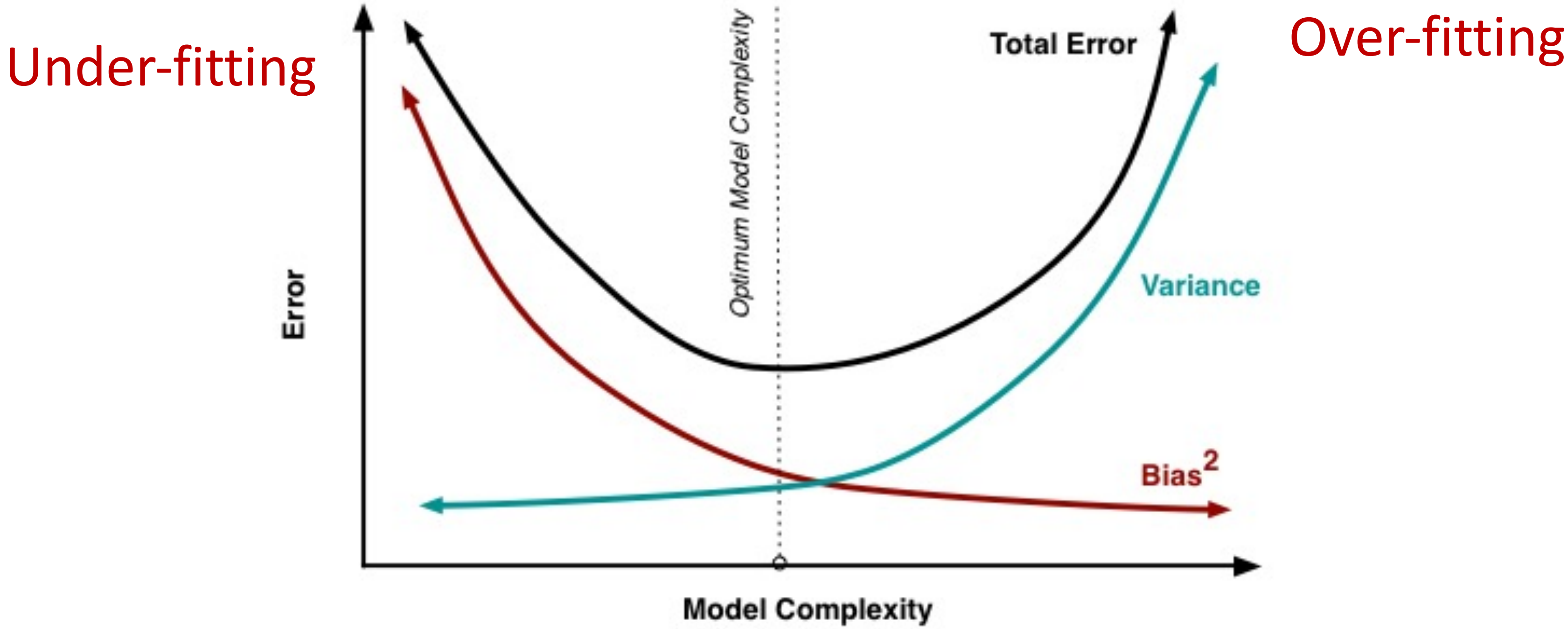


Simple model

Complex model

- Goal is to generalize well on new testing data
- Risk of overfitting to training data

Bias-Variance Tradeoff



- Bias = Difference between estimated and true models
- Variance = Model difference on different training sets

MSE is proportional to Bias + Variance

Regularization

- A method for automatically controlling the complexity of the learned hypothesis

- **Idea**: penalize for large values of θ_j
  - Can incorporate into the cost function
  - Works well when we have a lot of features, each that contributes a bit to predicting the label

- Can also address overfitting by eliminating features (either manually or via model selection)

Reduce model complexity

Reduce model variance