DS 5220

Supervised Machine Learning and Learning Theory

Alina Oprea
Associate Professor, CCIS
Northeastern University

September 25 2019

Outline

- Brief review
- Gradient descent
  - Batch algorithm
  - Line search optimization
- Gradient descent for linear regression
- Regularization
  - Ridge and Lasso regression
  - Gradient descent for ridge regression

Review

- Regression
  - Linear regression
  - MSE loss
  - Closed-form solution
  - Simple and multiple regression
  - Bias-variance tradeoff
- Classification
  - Linear models
  - Perceptron
  - Generative models: Linear Discriminant Analysis (LDA)

Training algorithms

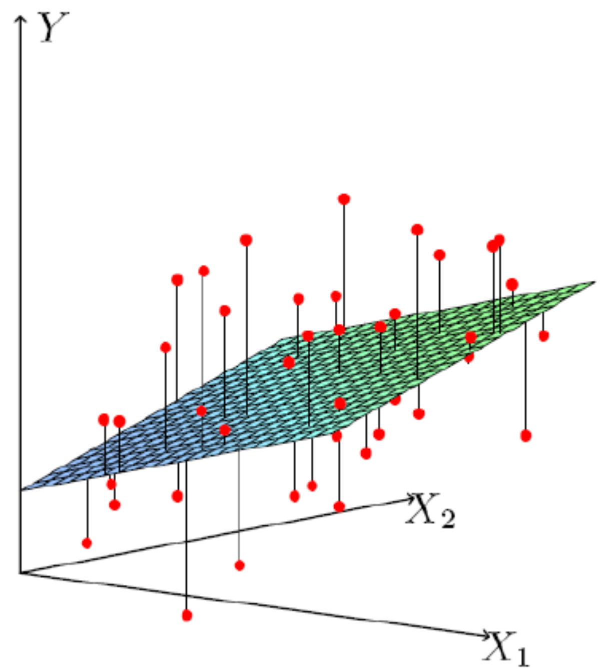
- Linear Regression
  – Minimize MSE
  – Compute closed-form solution (linear model that minimizes the MSE)
- LDA
  – Estimating probabilities of each class using Bayes Theorem
  – Learn normal distribution of data in each class
  – Estimate mean vector and covariance matrix

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

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

What are the drawbacks of computing the closed-form solution?



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
  - Strictly convex function (unique minimum)
- General method to optimize a multi-variate function
  - Practical (low asymptotic complexity)
  - Convergence guarantees to global minimum

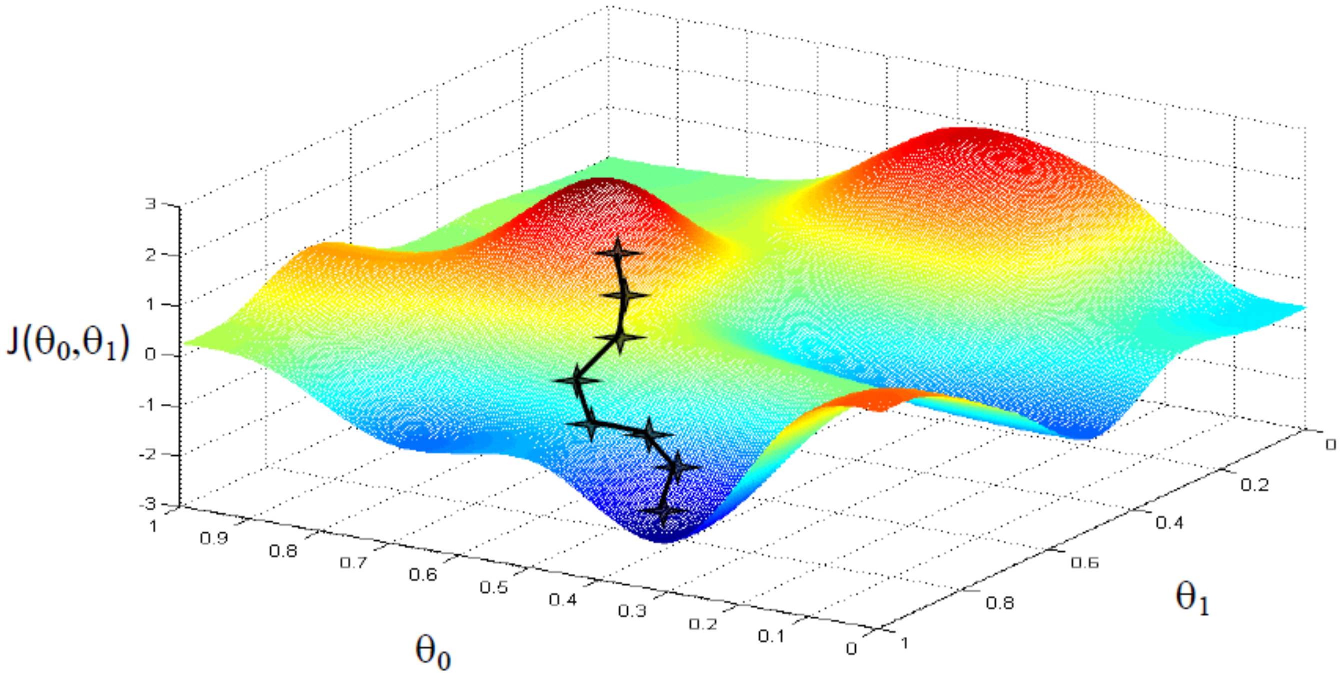
What Strategy to Use?

Follow the Slope

Follow the direction of steepest descent!

How to optimize $J(\theta)$?

- Choose initial value for $\boldsymbol{\theta}$
- Until we reach a minimum:
  - Choose a new value for $\boldsymbol{\theta}$ to reduce $J(\boldsymbol{\theta})$ → Direction of steepest descent!



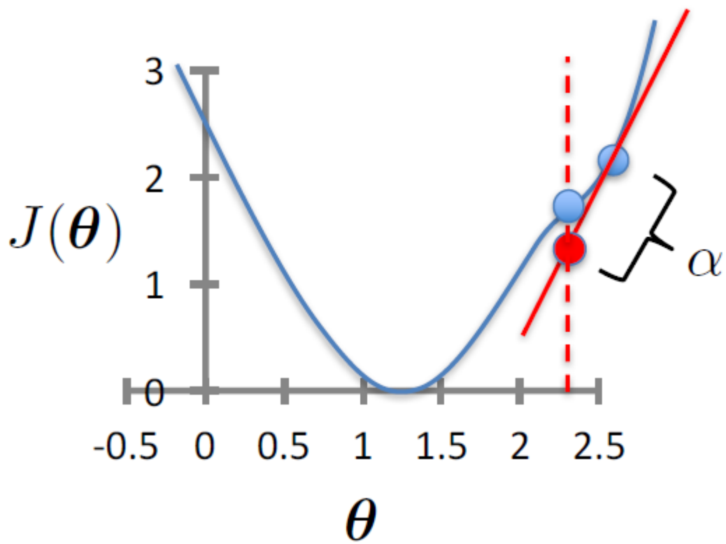
Batch Gradient Descent

- 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 = slope of line tangent to curve
- Function decreases faster in negative direction of gradient
- Step is proportional to learning rate

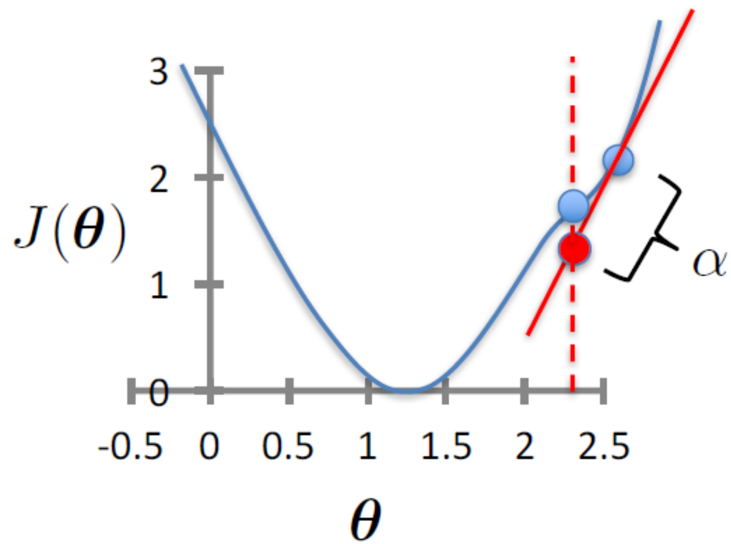
Batch Gradient Descent

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

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

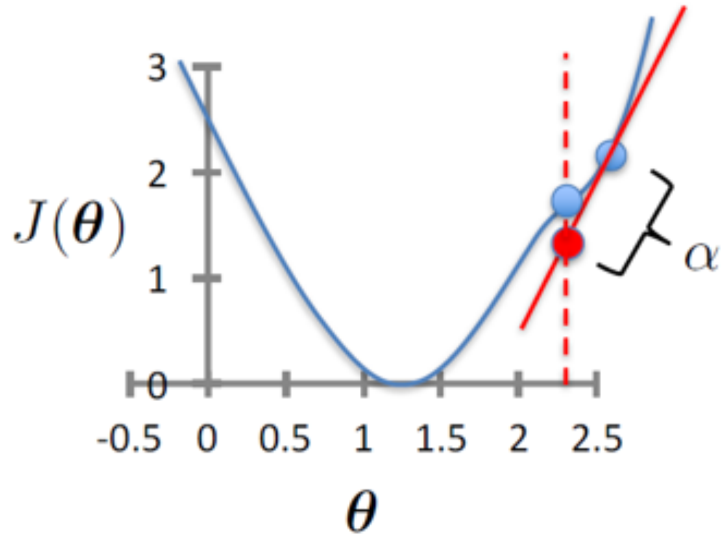
simultaneous update
for j = 0 ... d

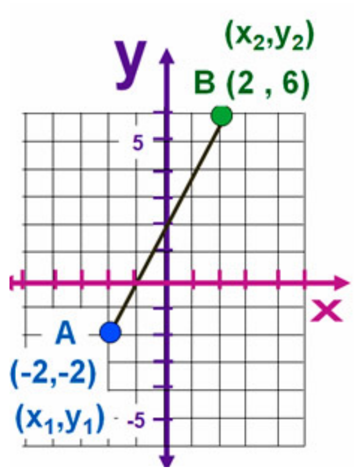
learning rate (small)
e.g., α = 0.05



Vector update rule: $\theta \leftarrow \theta - \frac{\partial J(\theta)}{\partial \theta}$

Gradient Descent





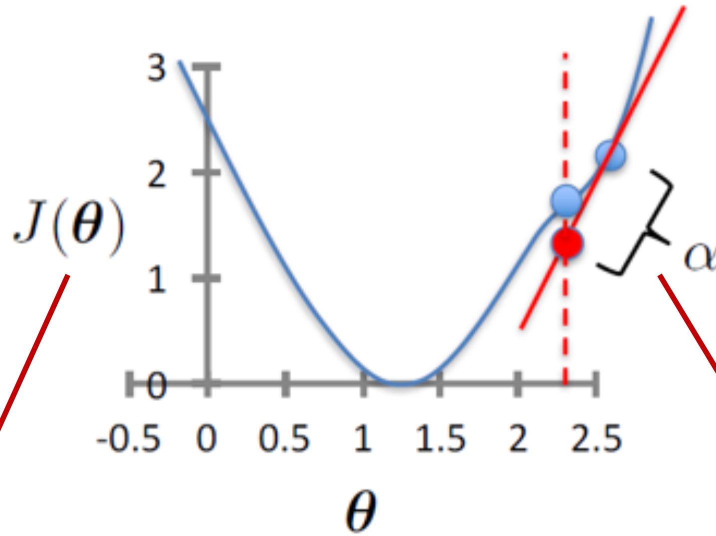
The Gradient "m" is:

$$m = \frac{y_2 - y_1}{x_2 - x_1} = \frac{\Delta Y}{\Delta X}$$

$$m = \frac{6 - {}^{-}2}{2 - {}^{-}2}$$

$$m = 8 / 4 = 2 \checkmark$$

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

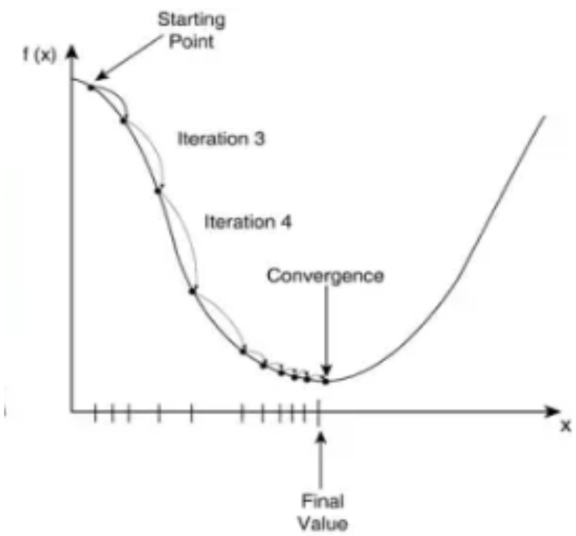
Gradient Descent

- 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



- As approach minimum, slope gets smaller (GD takes smaller steps)

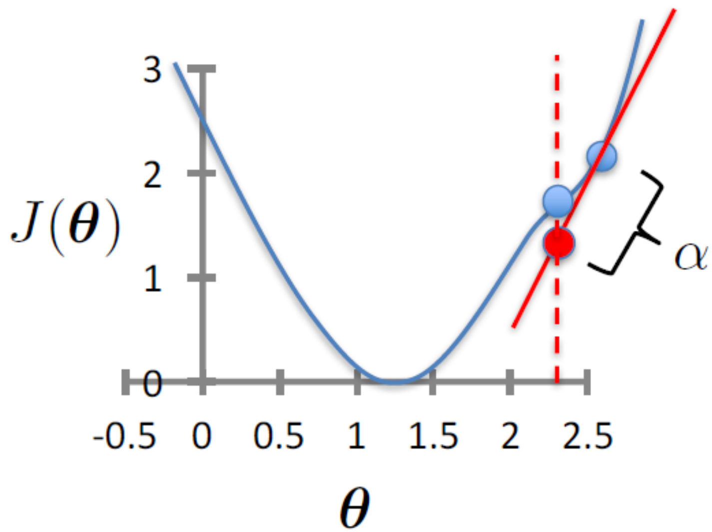
Gradient Descent

- 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?
- The slope is 0, and gradient descent converges!
- Strictly convex functions only have global minimum

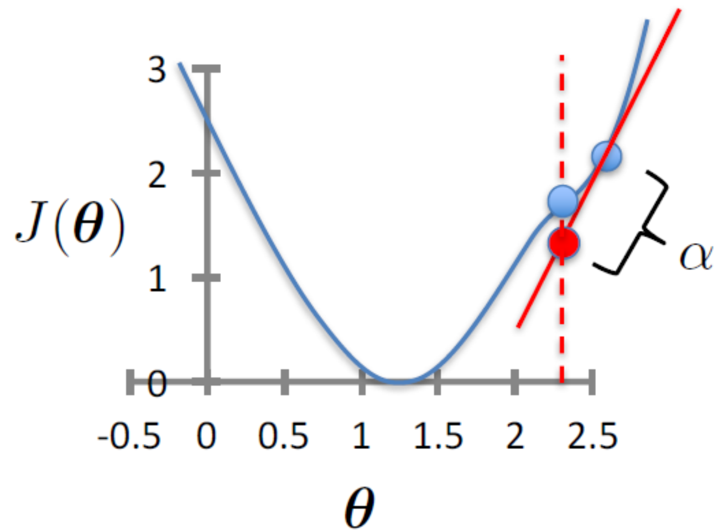
Stopping Condition

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

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

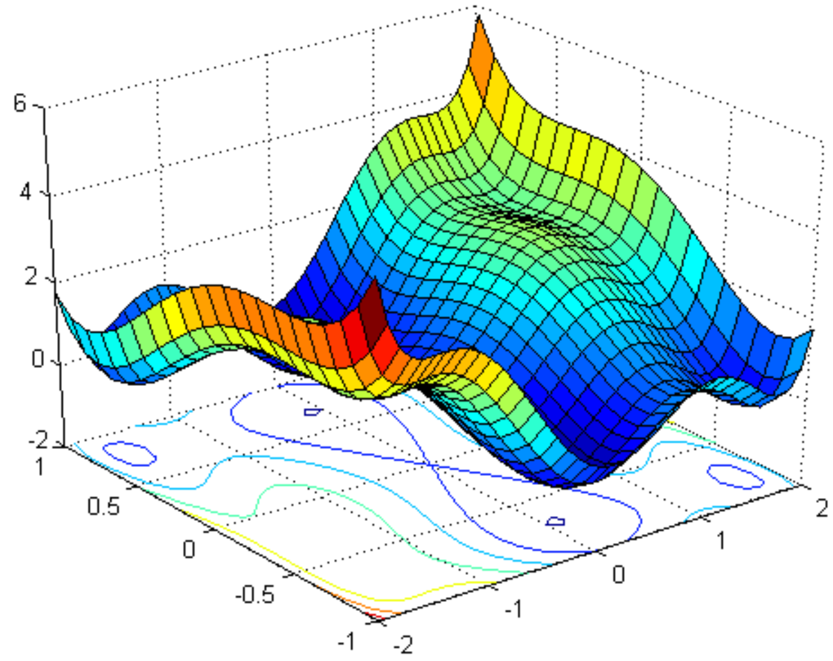
simultaneous update
for j = 0 ... d

learning rate (small)
e.g., α = 0.05



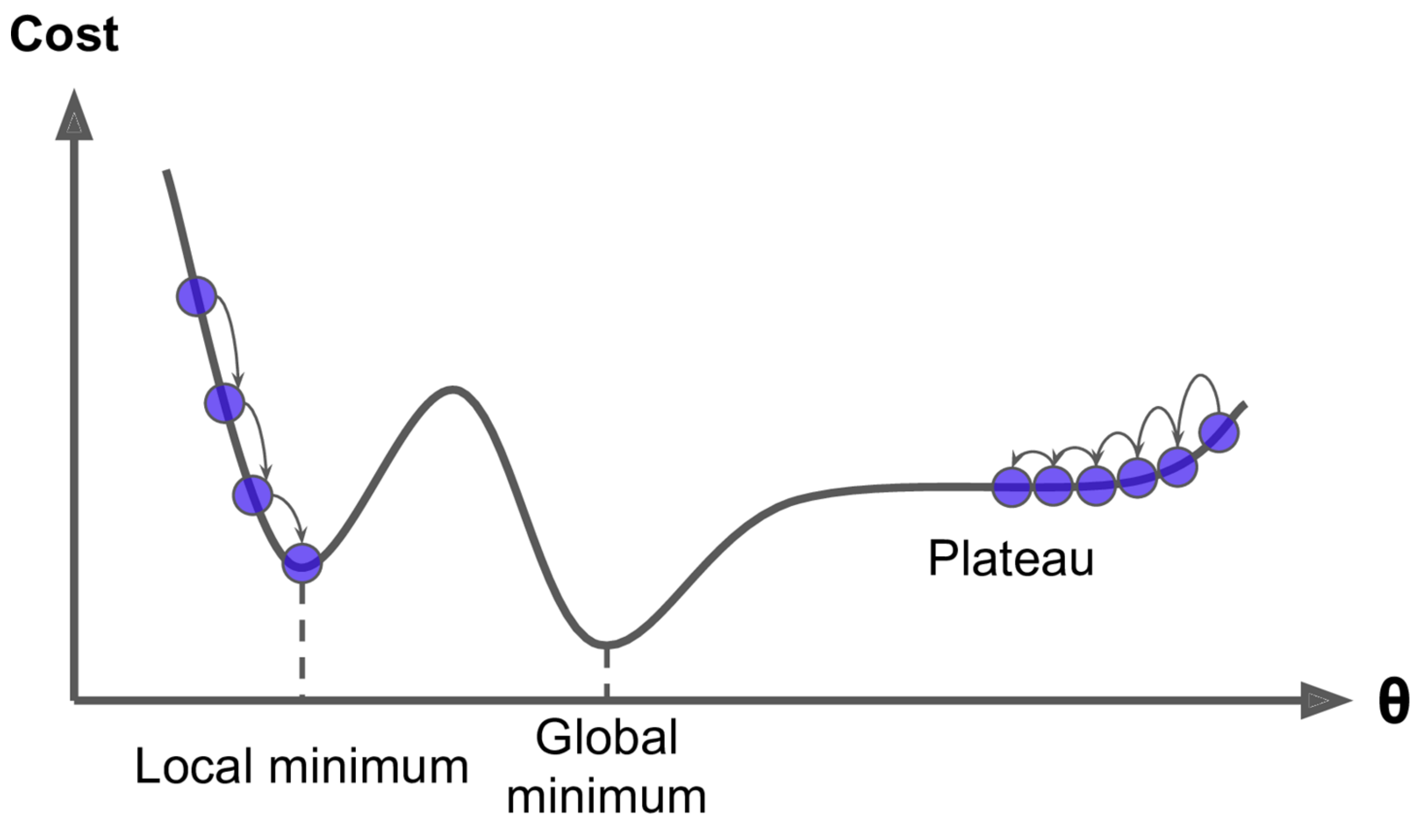
- When should the algorithm stop?
- When the update in θ is below some threshold

Complex loss function



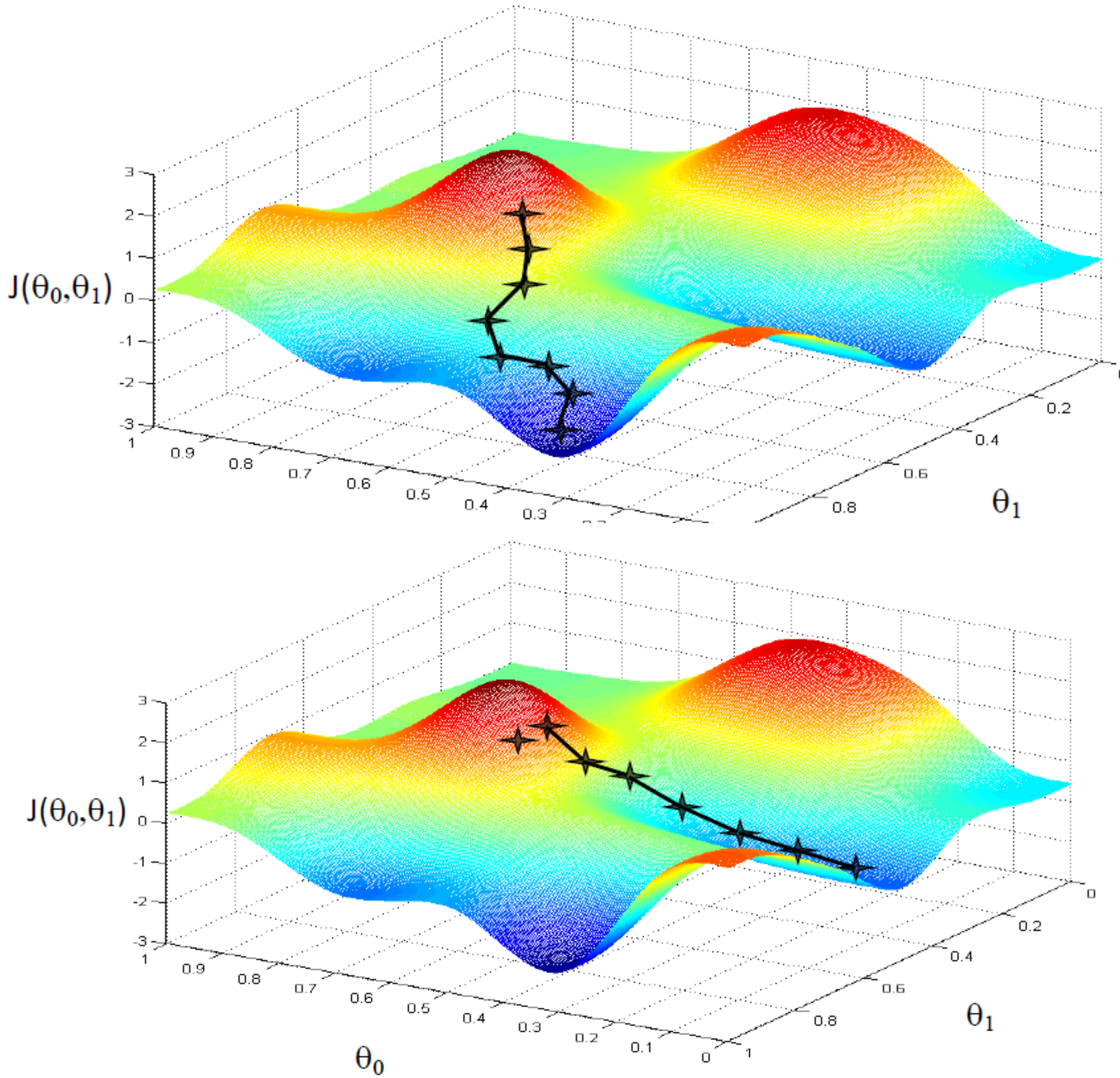
- Complex loss functions are more difficult to optimize

GD Convergence Issues



- Local minimum: Gradient descent stops
- Plateau: Almost flat region where slope is small

Solution: start from multiple random locations



$J(\theta_0,\theta_1)$
θ_1
$J(\theta_0,\theta_1)$
θ_0
θ_1

Simple Linear Regression

- Dataset $x_i \in R, y_i \in R, h_\theta(x) = \theta_0 + \theta_1 x$
- $J(\theta) = \frac{1}{N}\sum_{i=1}^{n}(\theta_0 + \theta_1 x_i - y_i)^2$

MSE / Loss

- Solution of min loss

$$-\ \theta_0 = \bar{y} - \theta_1 \bar{x}$$

$$-\ \theta_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

Variance of x

Co-variance of x and y

GD for Simple 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

- $J(\theta) = \frac{1}{N}\sum_{i=1}^{N}(\theta_0 + \theta_1 x_i - y_i)^2$
- $\frac{\partial J(\theta)}{\partial \theta_0} = \frac{2}{N}\sum_{i=1}^{N}(\theta_0 + \theta_1 x_i - y_i) = \frac{2}{N}\sum_{i=1}^{N}(h_\theta(x_i) - y_i)$
- $\frac{\partial J(\theta)}{\partial \theta_1} = \frac{2}{N}\sum_{i=1}^{N}(h_\theta(x_i) - y_i)\, x_i$

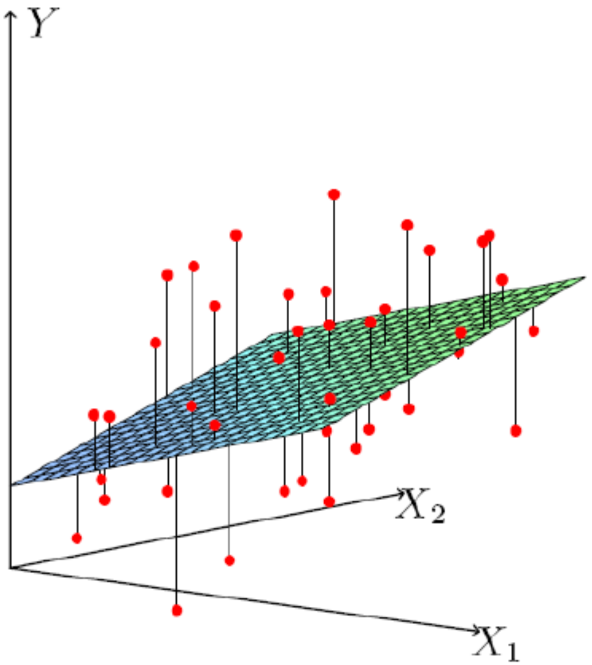
Batch: Update of each parameter
component depends on all training data

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

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

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

- $J(\theta) = \frac{1}{N} \sum_{i=1}^{N} \left(\sum_k \theta_k x_{ik} - y_i \right)^2$
- $\frac{\partial J(\theta)}{\partial \theta_j} = \frac{2}{N} \sum_{i=1}^{N} \left(\sum_k \theta_k x_{ik} - y_i \right) \frac{\partial \left(\sum_k \theta_k x_{ik} - y_i \right)}{\partial \theta_j}$

$$= \frac{2}{N} \sum_{i=1}^{N} \left(h_\theta(x_i) - y_i \right) x_{ij}$$

GD for Linear Regression

- Initialize θ
- Repeat until convergence

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

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

simultaneous update for j = 0 ... d

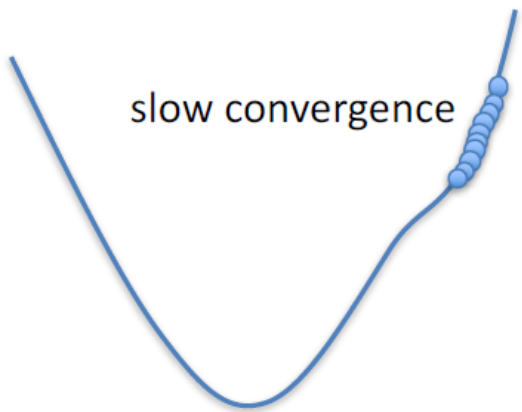
- Assume convergence when $\|\boldsymbol{\theta}_{new} - \boldsymbol{\theta}_{old}\|_2 < \epsilon$

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

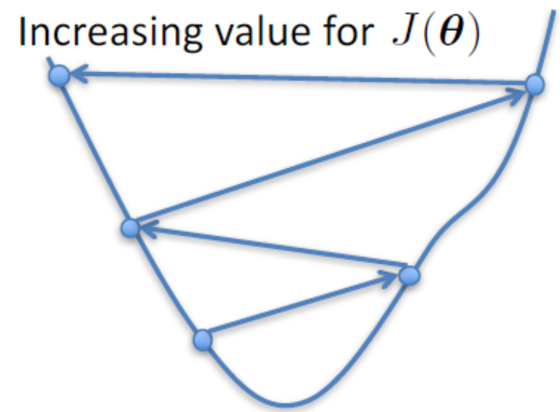
Can also bound number of iterations

Choosing learning rate

α too small



α too large

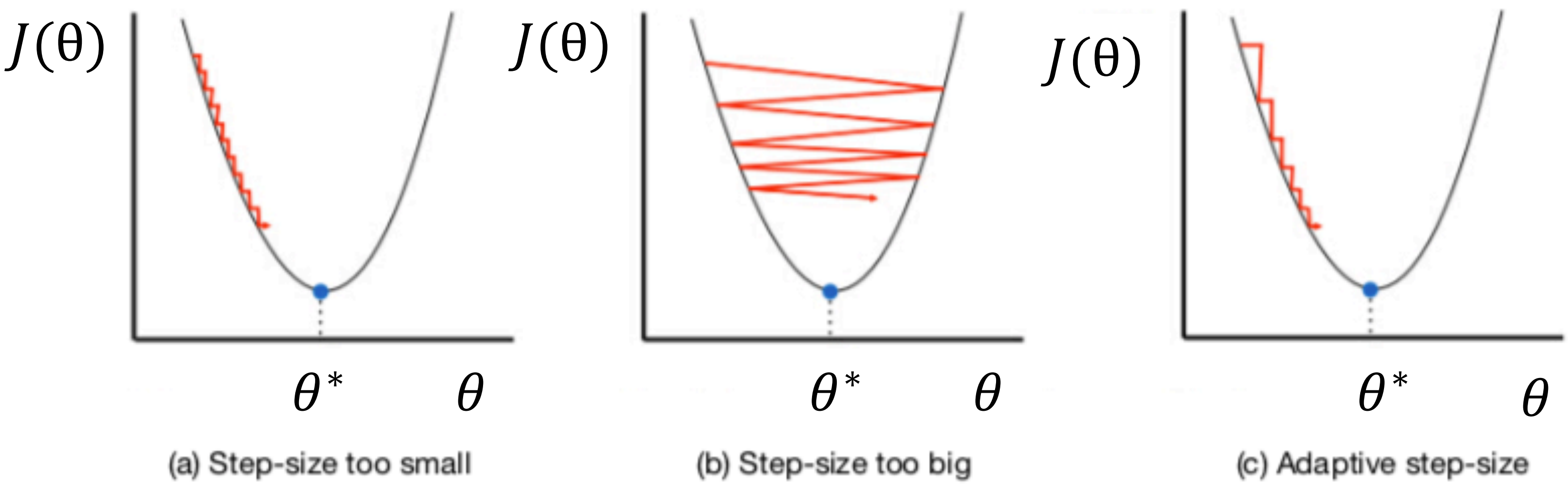


- May overshoot the minimum
- May fail to converge
- May even diverge

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



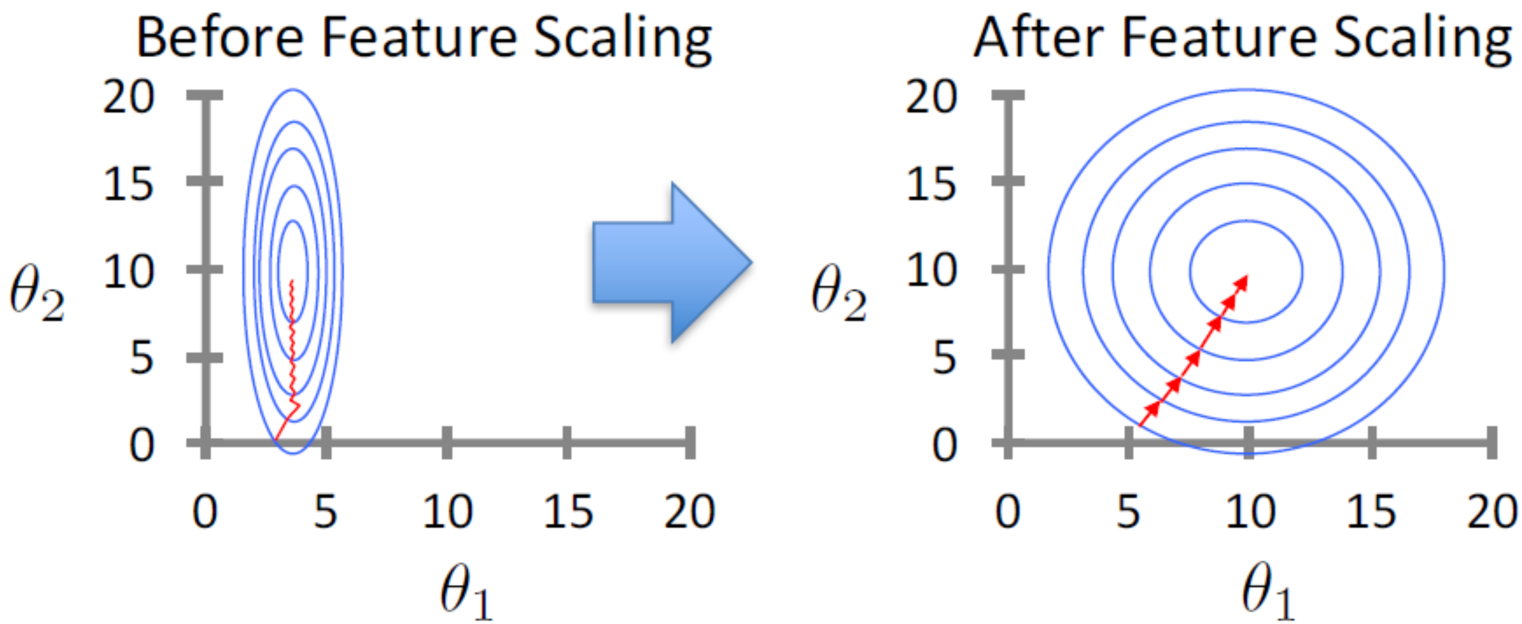
- Start with large step size and reduce over time, adaptively
- Measure how objective decreases

Gradient Descent with Line Search

- Input: α_{max} max. learning rate
  $\tau \in [0.5, 0.9], \epsilon$ (tolerance), T(backtrack)
- Initialize θ with random value
- $\alpha \leftarrow \alpha_{max}$ // Set at maximum learning rate
- Repeat until convergence
  - $\theta_{try} \leftarrow \theta$
  - Repeat max T times // Maximum number backtrack
  - $\theta_j^{try} \leftarrow \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\boldsymbol{\theta})$ for all j
    - If $J(\theta) - J(\theta_{try}) > \epsilon$: then $\theta \leftarrow \theta_{try}$; break // Improved objective
    - Else $\alpha \leftarrow \tau\alpha$ (reduce step size). // Backtrack to smaller rate
  - If T times is reached, break and start over

Feature Scaling

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



- Makes gradient descent converge *much* faster

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

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

• 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!
  - Much 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

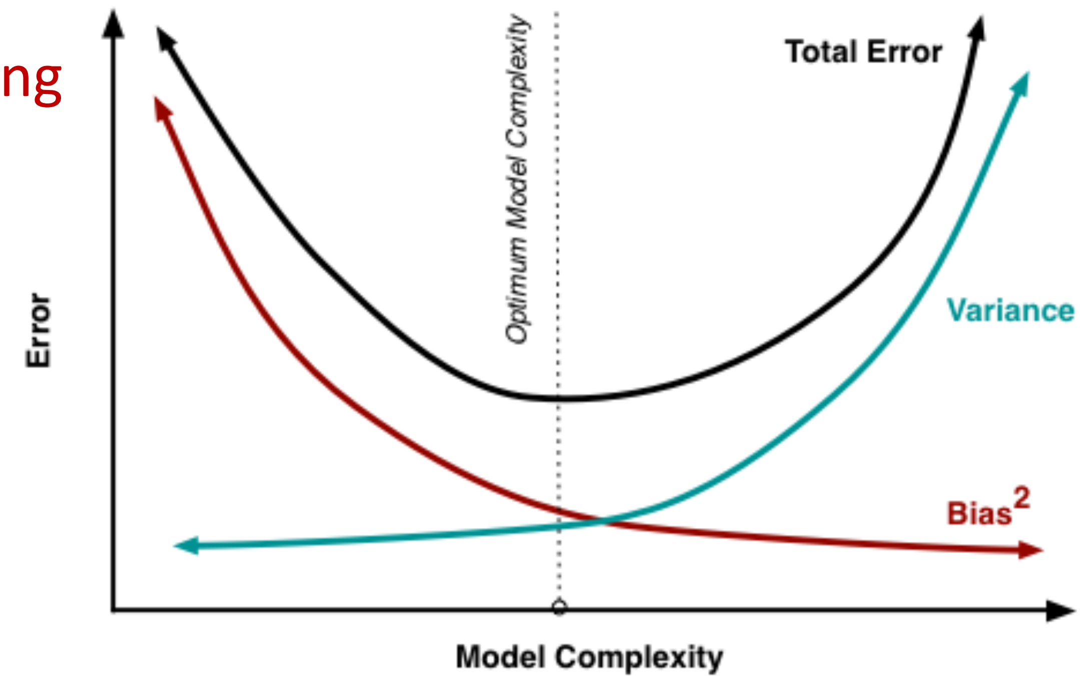
Outline

- Brief review
- Gradient descent
  - Batch algorithm
  - Line search optimization
- Gradient descent for linear regression
- Regularization
  - Ridge and Lasso regression
  - Gradient descent for ridge regression

Bias-Variance Tradeoff

Under-fitting

Over-fitting



- 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

Ridge regression

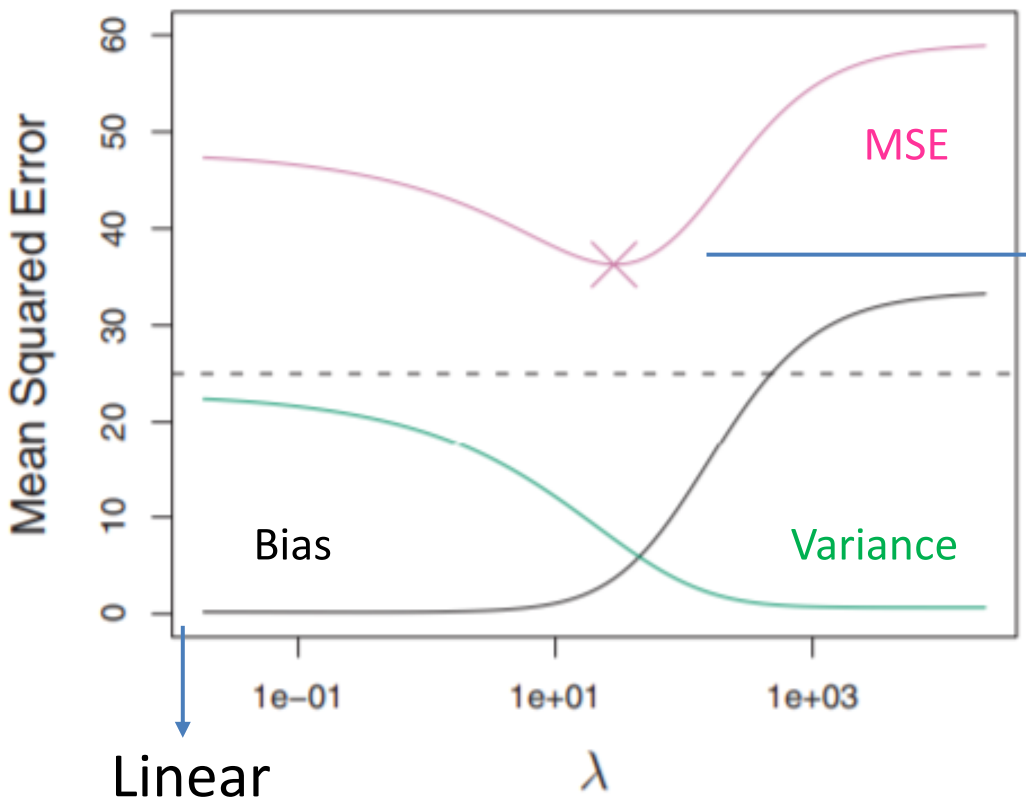
- Linear regression objective function

$$J(\theta) = \underbrace{\frac{1}{2}\sum_{i=1}^{N} (h_\theta(x_i) - y_i)^2}_{\text{model fit to data}} + \underbrace{\frac{\lambda}{2}\sum_{j=1}^{d} \theta_j^2}_{\text{regularization}}$$

  - λ is the regularization parameter ($\lambda \geq 0$)
  - No regularization on θ_0!

    - If λ = 0, we train linear regression
    - If λ is large, the coefficients will shrink close to 0

Bias-Variance Tradeoff

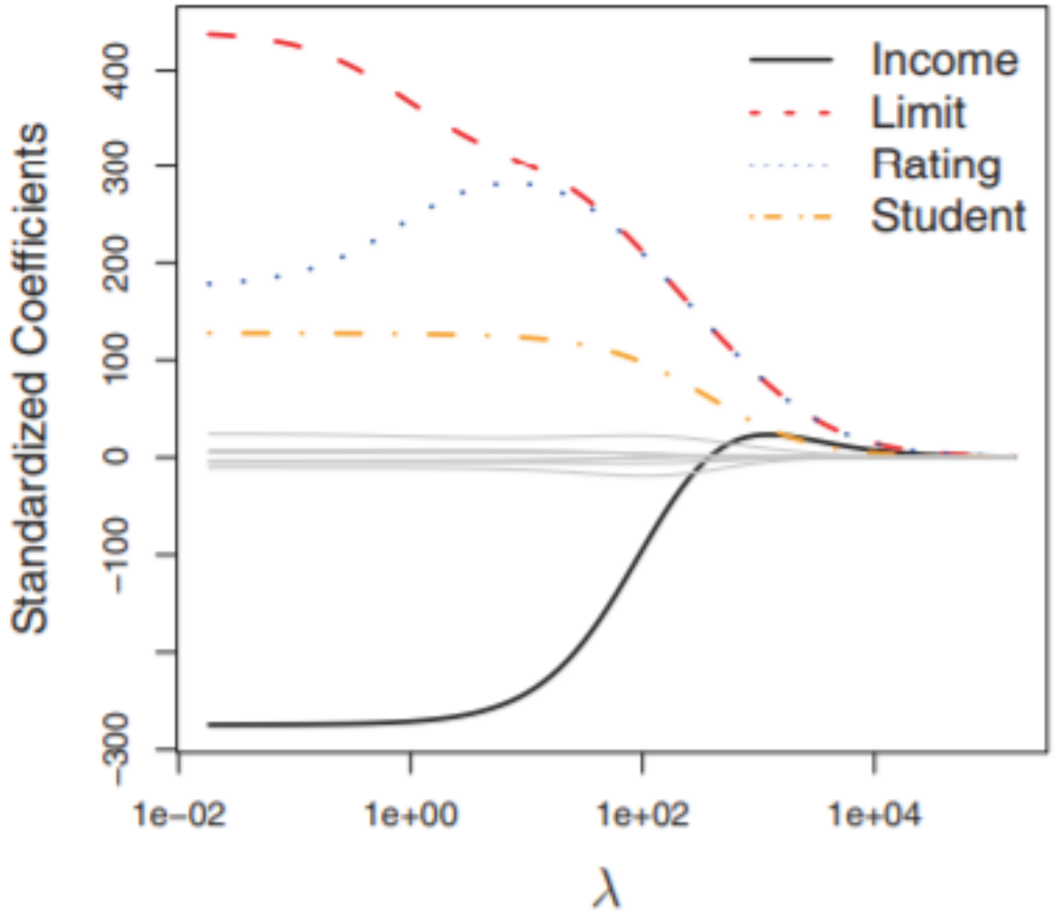


Optimal
Ridge regression

Linear
regression

Reduced model
complexity

Coefficient shrinkage



Predict credit card balance

GD for Ridge Regression

Min MSE

$$\boxed{J(\theta) = \frac{1}{2}\sum_{i=1}^{N} (\, h_\theta(x_i) - y_i \,)^2 + \frac{\lambda}{2}\sum_{j=1}^{d} \theta_i^2}$$

Gradient update: $\theta_0 \leftarrow \theta_0 - \alpha \sum_{i=1}^{N}(h_\theta(x_i) - y_i)$

$$\theta_j \leftarrow \theta_j - \alpha \sum_{i=1}^{N}(h_\theta(x_i) - y_i)x_{ij} - \underbrace{\alpha\lambda\theta_j}_{\text{Regularization}}$$

$$\boxed{\theta_j \leftarrow \theta_j(1 - \alpha\lambda) - \alpha \sum_{i=1}^{N}(h_\theta(x_i) - y_i)x_{ij}}$$

Review

- Gradient descent is a general optimization algorithm that can be applied in training
  - Minimize loss function (error metric)
- Gradient descent converges to local minima
- Requires selection of learning rate
- Bias-Variance tradeoff shows that both bias and variance need to be minimized
  - Regularization is a method to reduce model complexity and avoid overfitting
  - Ridge and Lasso regularization can be added to any loss function
  - Regularized model trained with Gradient Descent

Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
- Thanks!