Regression

陳弘軒 Hung-Hsuan Chen
https://www.ncu.edu.tw/~hhchen
中央大學資訊工程學系

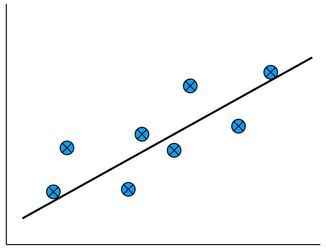
「版權聲明頁」

Notations

- Features $\boldsymbol{x}$
- Target y
- Prediction $\hat{y}$
- Parameters θ (to be learned)

Linear regression

y **dependent variable (output)**



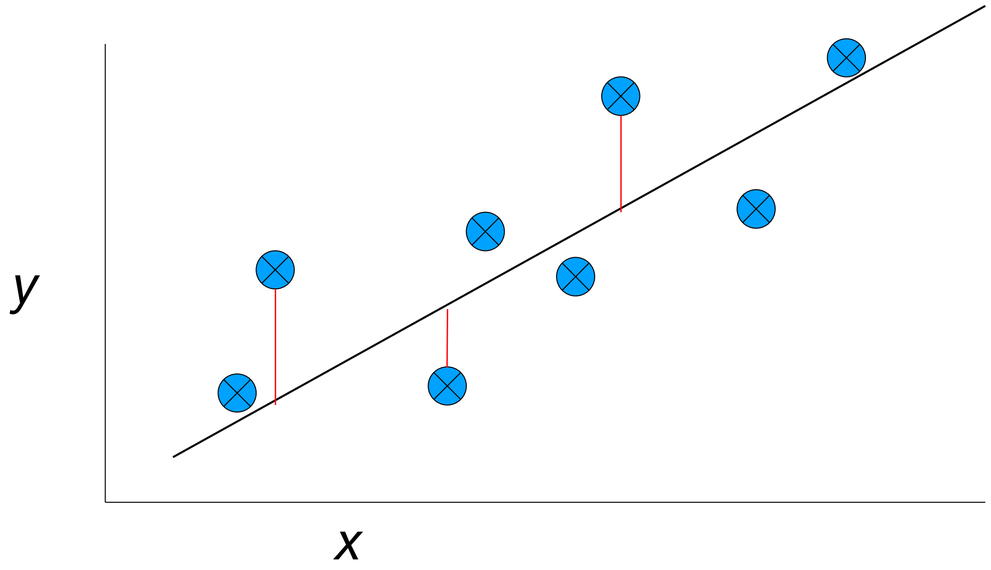
x **– independent variable (input)**

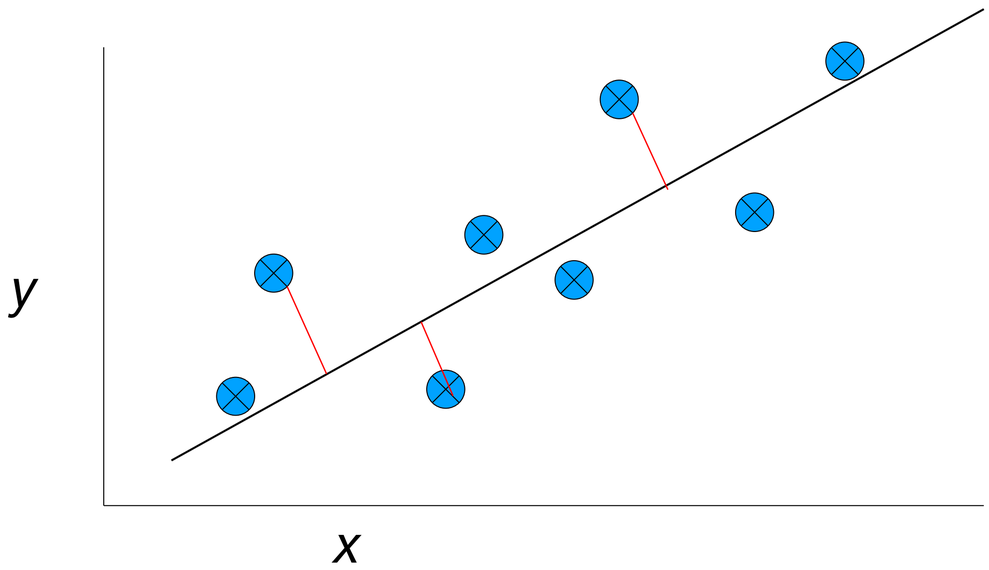
"Predictor": $\hat{y} = \theta_0 + \theta_1 x$

- $\boldsymbol{x} = [1 \quad x]^T$
- $\boldsymbol{\theta} = [\theta_0 \quad \theta_1]^T$

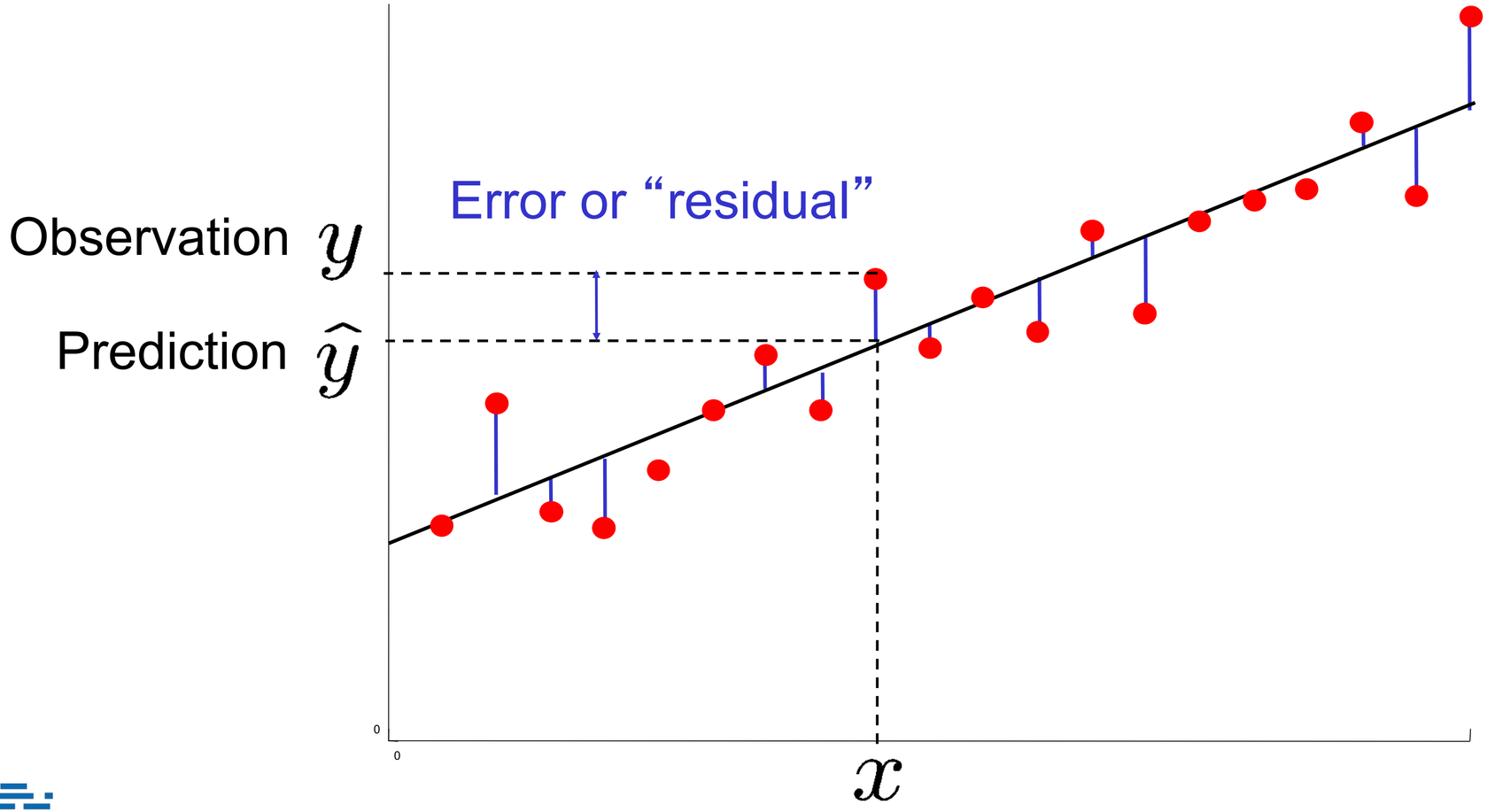
- Define the form of the function f(***x***) explicitly
  - i.e., $\hat{y} = \theta_0 + \theta_1 x$ in this case
  - If you believe that *x* and *y* have a **non-linear** relationship, you should assume a non-linear f(***x***)
- Find a good f(***x***) within that family
  - i.e. find good θ_0 and θ_1 such that $\hat{y} \approx y$ in this case

Quiz: which one should be an error measurement?





Measuring error



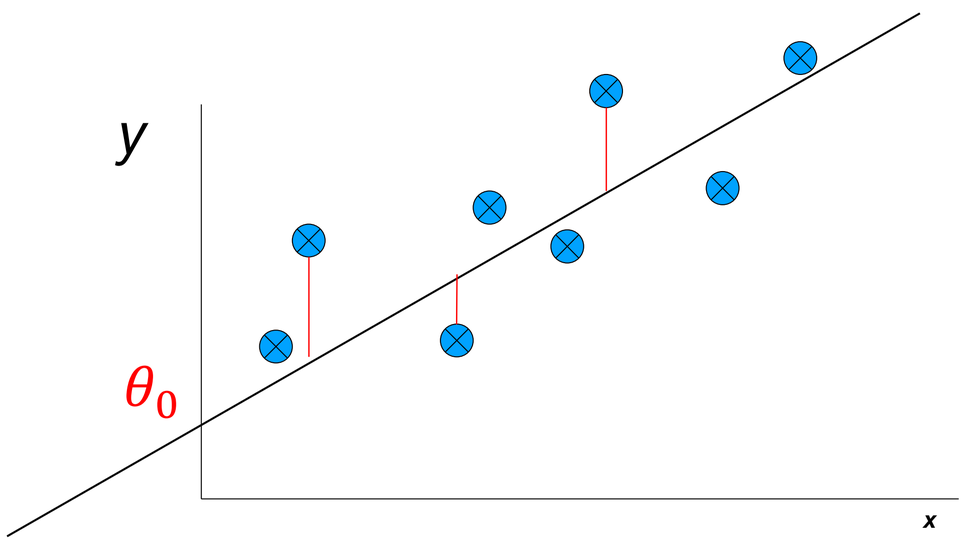
Simple linear regression

- For now, assume just one (input) independent variable *x*, and one (output) dependent variable *y*
- ”Fit" the points with a line, which line should we select?
  - Choose an objective function to minimize
  - Typically, we choose sum squared error (SSE)
    - $\frac{1}{n}\sum(\hat{y}_i - y_i)^2 = \frac{1}{n}\sum(\text{residual}_\text{i})^2$
    - Choices with the same result: $\frac{\mathbf{1}}{2\boldsymbol{n}}\sum(\hat{y}_i - y_i)^2$ or $\frac{\mathbf{1}}{2}\sum(\hat{y}_i - y_i)^2$

How to “learn” the parameters?

- For the 2-*d* problem there are coefficients for the bias and the independent variable (*y*-intercept and slope)
  - $\hat{y} = \theta_0 + \theta_1 x$
- The value of the coefficients which minimize the objective function:
  - $\theta_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2}$
  - $\theta_0 = \frac{\sum y_i - \theta_1 \sum x_i}{n}$
  - *n*: number of training instances

Visualzing θ_0 and θ_1



Slope = θ_1

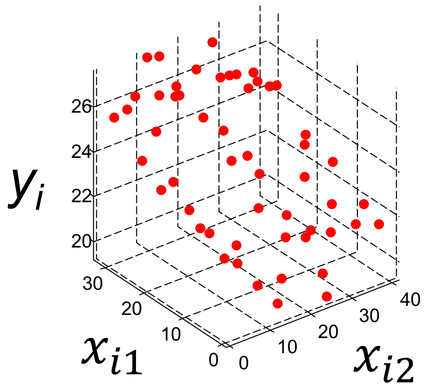
$$\hat{y} = \theta_0 + \theta_1 x$$

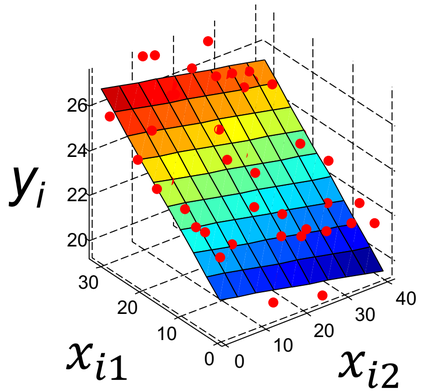
Derive the value of the coefficients

- $J(\theta_0, \theta_1) = \frac{1}{2n}\sum(\hat{y}_i - y_i)^2 = \frac{1}{2n}\sum((\theta_0 + \theta_1 x_i) - y_i)^2$,
- Set
  - $\frac{\partial J(\theta_0,\theta_1)}{\partial \theta_0} = 0$ and
  - $\frac{\partial J(\theta_0,\theta_1)}{\partial \theta_1} = 0$

  to solve θ_0 and θ_1

More dimensions?





- $\hat{y}(\mathbf{x}_i) = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \ \ldots + \theta_d x_{i,d} = \boldsymbol{\theta}^T \boldsymbol{x}_i$, where
  - $\boldsymbol{x}_i = \left[1, x_{i,1}, x_{i,2}, \ldots, x_{i,d}\right]^T$
  - $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2, \ldots, \theta_d]^T$
- d: # of features, $x_{i,j}$: the ith training instance's jth feature

Multiple linear regression

- *n*: the number of training instances
- *d*: the number of features
- Training instances:
  - $\boldsymbol{X} = \begin{bmatrix} 1 & x_{1,1} & \cdots & x_{1,d} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n,1} & \cdots & x_{n,d} \end{bmatrix}$
  - $\boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$

- Find $\boldsymbol{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$ such that $(\hat{\boldsymbol{y}} - \boldsymbol{y})^T(\hat{\boldsymbol{y}} - \boldsymbol{y})$ is minimized, where
  - $\hat{\boldsymbol{y}} = \boldsymbol{X\theta}$
- The solution is $\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$

How to derive the solution?

- $J(\boldsymbol{\theta}) = \frac{1}{2}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T(\hat{\boldsymbol{y}} - \boldsymbol{y})$

$$= \frac{1}{2}(\boldsymbol{X\theta} - \boldsymbol{y})^T(\boldsymbol{X\theta} - \boldsymbol{y})$$

- Set

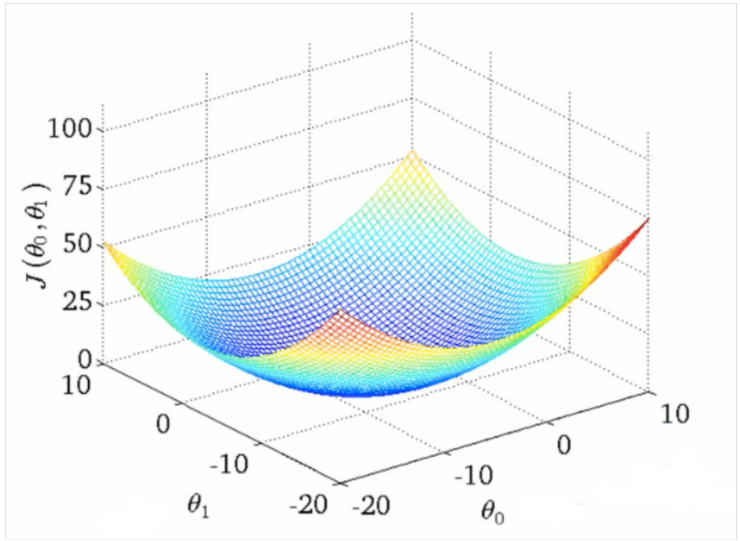
$$\nabla J(\boldsymbol{\theta}) = \mathbf{0}$$

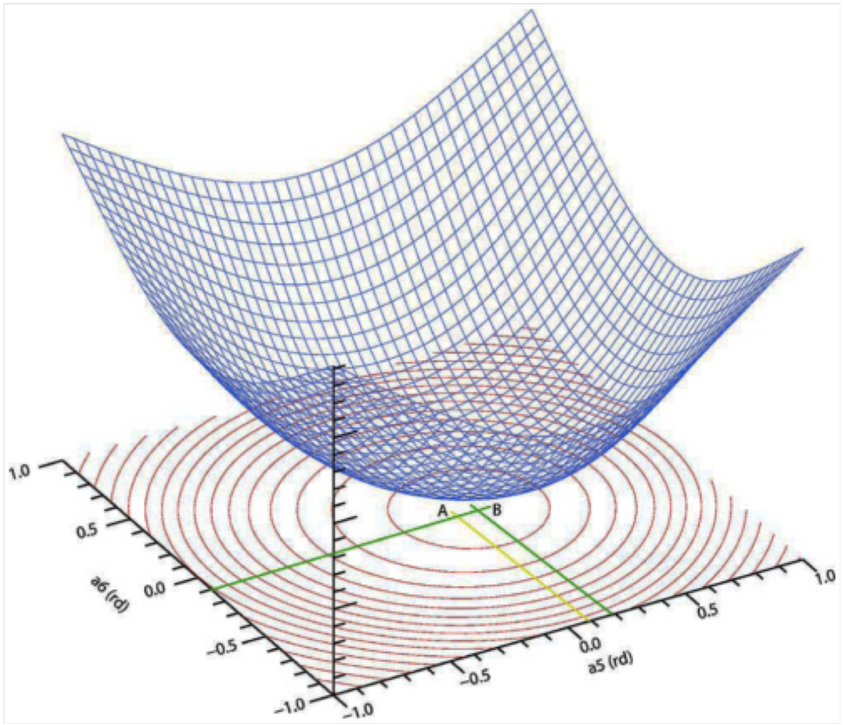
to solve $\boldsymbol{\theta}$

Another way to find $\boldsymbol{\theta}$

- Goal: find θ_0 and θ_1 to minimize
  - $J(\theta_0, \theta_1) = \frac{1}{2n}\sum(\hat{y}_i - y_i)^2 = \frac{1}{2n}\sum((\theta_0 + \theta_1 x_i) - y_i)^2$
- Procedure
  - Start with $\theta_0 = r_0$ (a random number) and $\theta_1 = r_1$ (another random number)
  - Slightly move θ_0 and θ_1 to reduce $J(\theta_0, \theta_1)$
  - Keep doing step 2 until converged
- Question: how to move θ_0 and θ_1?

Plotting the objective function





- Keep moving ”downward” to reach the minimum

Gradient descent

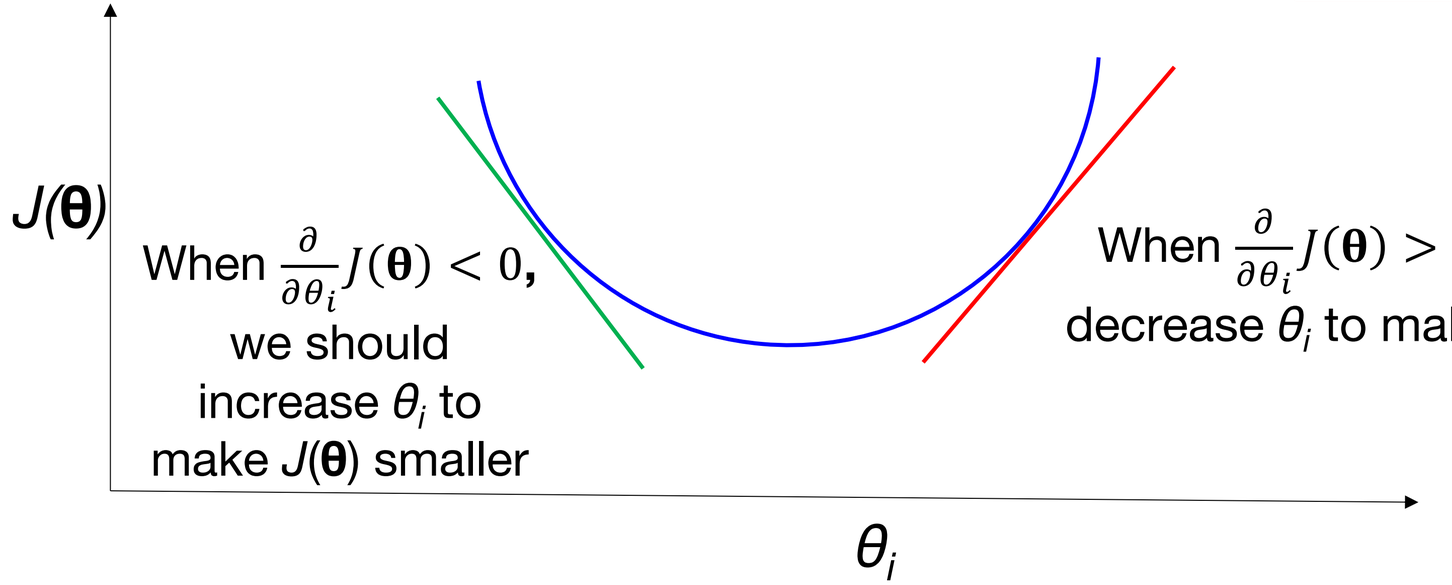
- Procedure
  - Start with random values
    - $\boldsymbol{\theta} = \boldsymbol{\theta}^{(0)} = (\theta_0^{(0)}, \theta_1^{(0)}, \ldots, \theta_d^{(0)})$
  - Slightly move $\theta_0, \ldots \theta_d$ to reduce $J(\boldsymbol{\theta})$
    - $\theta_i^{(k+1)} = \theta_i^{(k)} - \alpha \left. \frac{\partial J(\boldsymbol{\theta})}{\partial \theta_i} \right|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(k)}}$
    - $k = k + 1$
  - Keep doing step 2 until converges

α **is a small positive number**

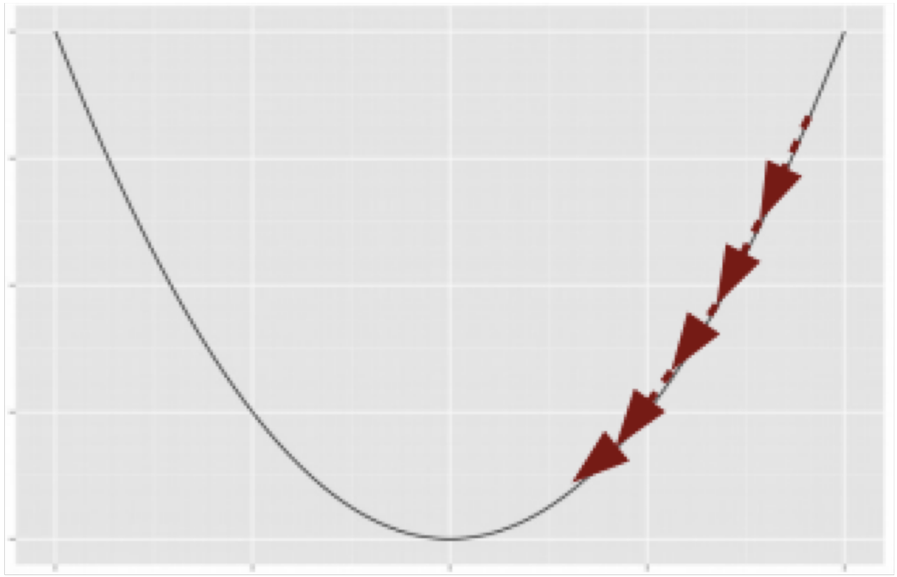
Why gradient descent work?

```
Repeat until converge {
   for (i=1,2, ..., d){
      θ_i := θ_i − α ∂/∂θ_i J(θ) = θ_i − α Σ_j (ŷ_j − y_j) x_ji
   }
}
```

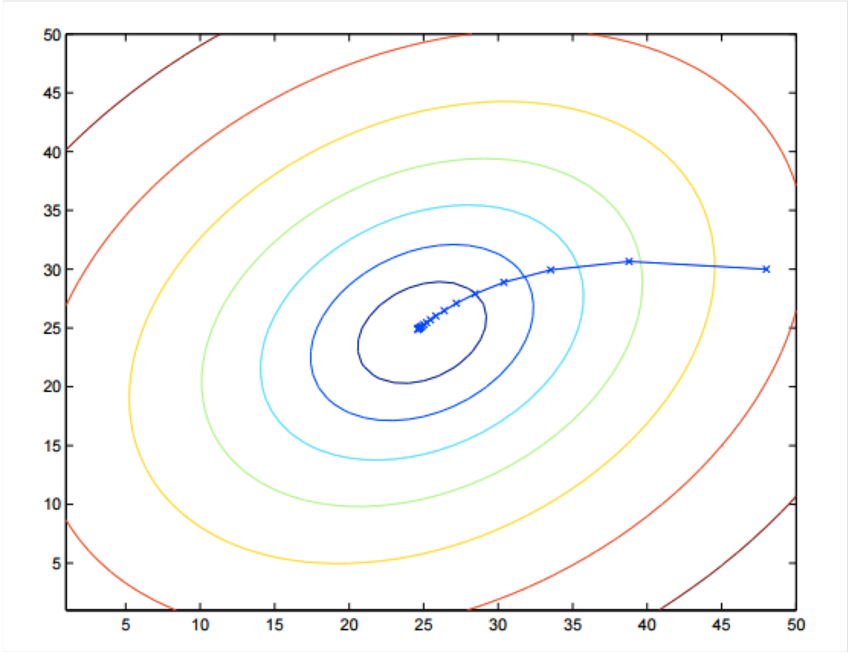
$\frac{\partial}{\partial \theta_i} J(\theta) = \frac{1}{n} \sum_j (\hat{y}_j - y_j)\, x_{ji}$,
$\frac{1}{n}$ **is usually ignored**



Visualize gradient descent steps

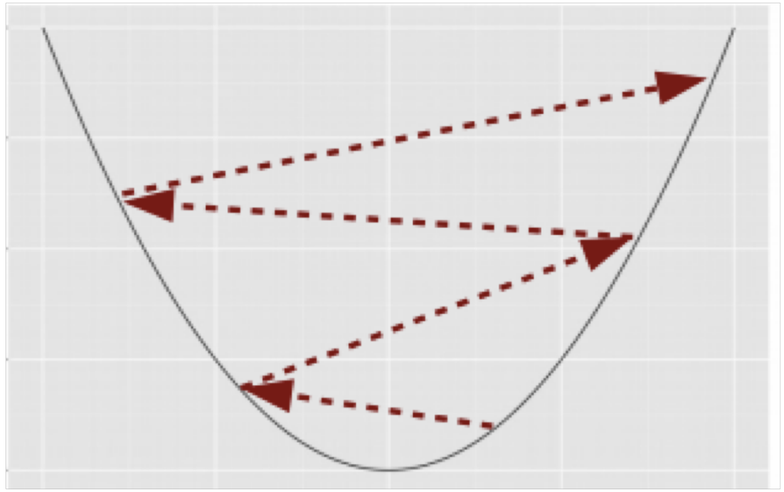


One parameter to learn



Two parameters to learn

Overly large learning rate may not lead to converge



- $\theta_i^{(k+1)} := \theta_i^{(k)} - \alpha \frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}^{(k)})$
  - α: learning rate
  - Often $\alpha \in [0, 1]$
  - Shrink α as k becomes larger

Possible termination conditions

- Improvement drops (e.g., < 0)
- Reached small error
- Achieved predefined # of iterations
- No time to train anymore

We have two tools for linear regression

- Close form solution
  - $\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}$

- Gradient descent

```
Repeat until converge
{
   for (i=1,2, ..., d){
      θ_i = θ_i − α Σ_j (ŷ_j − y_j) x_ji
   }
}
```

Which one should we use?

Consider using close form solution

- $\boldsymbol{\theta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$
- Required space:
  - Even if $\boldsymbol{X}$ is sparse, $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ may not be sparse
  - If n=1M, d=100K, $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$'s dimension is $(100\text{k})^2 = 10^{10}$
    - If 8 bytes per entry, storing $(\boldsymbol{X}^T\boldsymbol{X})^{-1}$ requires 80GB ➔ typically infeasible on a single machine

- Computation time:
  - Computing matrix multiplication $\boldsymbol{X}^T\boldsymbol{X}$ takes $O(nd^2)$
  - Let$(\boldsymbol{X}^T\boldsymbol{X}) = \boldsymbol{P}$, computing $\boldsymbol{P}^{-1}$ takes $O(d^{2.373})$ to $O(d^3)$
  - Let $(\boldsymbol{X}^T\boldsymbol{X})^{-1} = \boldsymbol{Q}$, computing $\boldsymbol{Q}\boldsymbol{X}^T$ takes $O(nd^2)$
  - Let $(\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T = \boldsymbol{R}$, computing $\boldsymbol{R}\boldsymbol{y}$ takes $O(nd)$
  - Total: $O(nd^2+d^{2.373}+nd^2)$

Consider using Gradient Descent

```
Repeat until converge {
   for (j=1, 2, …, n){
      z_j = (θ^T x_j − y_j)
   }
   for (i=1, 2, ..., d){
      θ_i = θ_i − α Σ_j z_j x_ji
   }
}
```

- Required space:
  - $\boldsymbol{X} = [x_{ij}]$, $\boldsymbol{y}$, $\boldsymbol{\theta}$, and $\boldsymbol{z}$
  - If n=1M, d=100K, and $\boldsymbol{X}$ is sparse
    - If 8 bytes per entry, we need: (nnz($\boldsymbol{X}$)+1M+100K+1M)*8bytes
    - Much more efficient than 80GB
- Computation time:
  - $z_j = (\boldsymbol{\theta}^T \boldsymbol{x}_j - y_j)$ for all j: O(nd)
  - $\theta_i := \theta_i - \alpha \sum_j z_j x_{ji}$ for all i: O(nd)
  - Outer loop: Assume T iterations
  - Total: O(Tnd)
    - Usually more efficient than $O(nd^2 + d^{2.373} + nd^2)$

Close form vs gradient descent

- If the number of features is small, close form solution is probably acceptable
- However, if the number of features is large, using gradient descent is more efficient
- Moreover, gradient descent is capable of solving more complex optimization problem
  - In many cases, $\frac{\partial J(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}} = \mathbf{0}$ has no closed–form solution
    - But we can still apply gradient descent ☺

Quiz

- If the features are all categorical, can we apply linear regression?

Encoding categorical features

- Consider one-hot encoding categorical data (depending on the learning algorithm)
  - E.g., the values of the ”nationality” feature: “UK”, “Japan”, “Mexico”
  - We may encode UK as “1,0,0”, Japan as “0,1,0”, and “Mexico” as “0,0,1”
  - “Nationality feature” becomes three features: “UK or not”, “Japan or not”, “Mexico or not”

What if we encode categorical features by natural numbers?

- If we use linear regression model and naively encode the nationality feature (feature x_i) by numbers (e.g., “UK”, “Japan”, “Mexico” as 0, 1, 2)

$$\hat{y} = \theta_0 + \theta_1 x$$

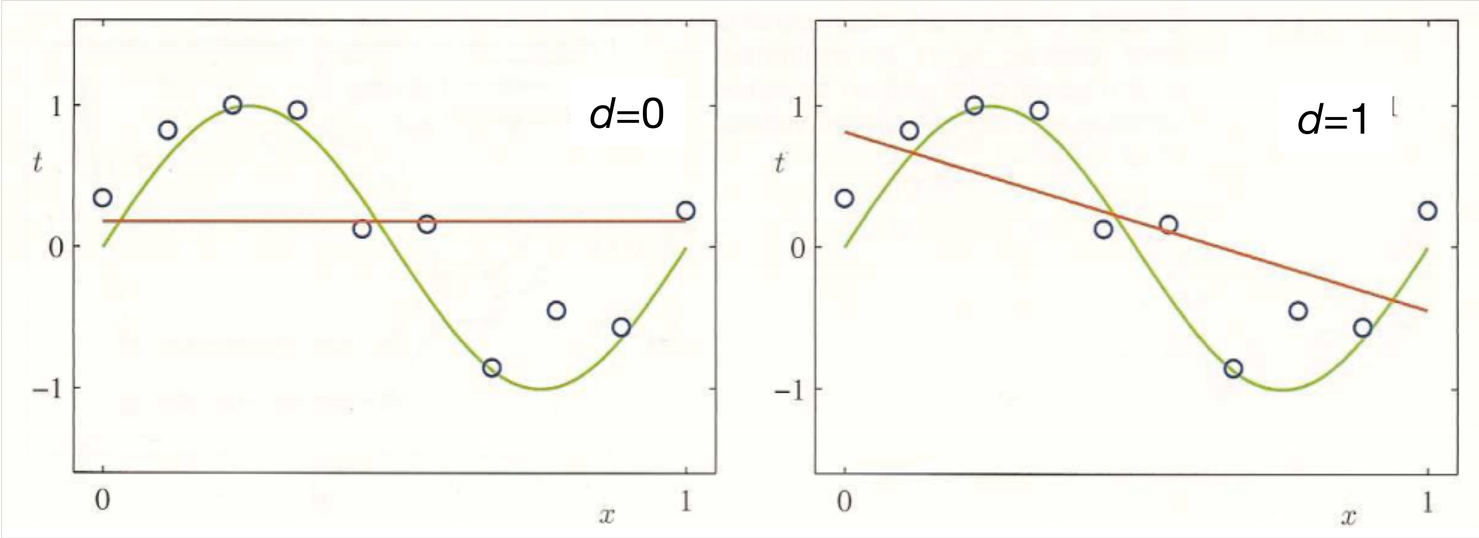
- **The three possible values y’s are**
  - θ_0**(UK)**
  - $\theta_0 + \theta_1$**(Japan)**
  - $\theta_0 + 2\theta_1$**(Mexico)**
- **If** $\theta_1 > 0$**,** $\theta_0 < \theta_0 + \theta_1 < \theta_0 + 2\theta_1$
- **If** $\theta_1 < 0$**,** $\theta_0 > \theta_0 + \theta_1 > \theta_0 + 2\theta_1$

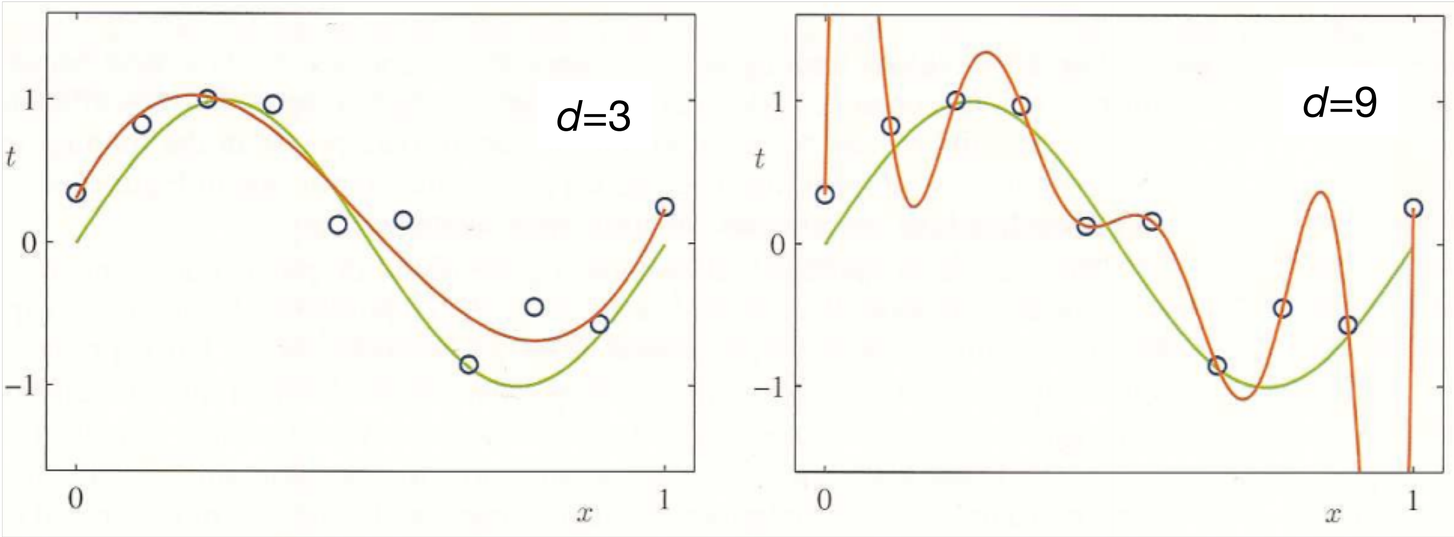
- **The distance between (UK, Mexico) is always larger than (UK, Japan)**
  - **This is NOT decided by the data, but by how we encode the data**

Fitting non–linear data to linear model

- $\hat{y}(\boldsymbol{x_i}) = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \ \ldots + \theta_d x_{i,d}$
  - Linear model
- We may generate the higher degree terms as the new features
  - $\hat{y}(\boldsymbol{x_i}) = \left(\theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \cdots + \theta_d x_{i,d}\right) +$
    $\left(\theta_{1,2} x_{i,1} x_{i,2} + \theta_{1,3} x_{i,1} x_{i,3} + \cdots + \theta_{d-1,d}\, x_{i,d-1} x_{i,d}\right) +$
    $(\theta_{1,1} x_{i,1}^2 + \theta_{2,2} x_{i,2}^2 + \cdots + \theta_{d,d} x_{i,d}^2)$

Target: $\sin(2\pi x) + \text{noise}$





Overfitting

- Overfitting occurs when a model captures idiosyncrasies of the input data, rather than generalizing
  - Too many parameters relative to the amount of training data
- For example, an order-*N* polynomial can perfectly fit to *N*+1 data points

Observation

- In the linear regression model, overfitting is characterized by large parameters

	$d = 0$	$d = 1$	$d = 3$	$d = 9$
θ_0	0.19	0.82	0.31	0.35
θ_1		-1.27	7.99	232.37
θ_2			-25.43	-5321.83
θ_3			17.37	48568.31
θ_4				231639.30
θ_5				640042.26
θ_6				-1061800.52
θ_7				1042400.18
θ_8				-557682.99
θ_9				125201.43

Regularization to avoid overfitting (1/2)

- Introduce a penalty term for the size of the weights
- Un-regularized regression (the original objective function)
  - $J(\boldsymbol{\theta}) = \frac{1}{2}\sum(\hat{y}_i - y_i)^2 = \frac{1}{2}\sum(\boldsymbol{\theta}^T \boldsymbol{x}_i - y_i)^2$
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \frac{1}{2}\sum(\hat{y}_i - y_i)^2$
- Regularized regression (enforce the solution to have low L2-norm of $\boldsymbol{\theta}$)
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \frac{1}{2}\sum(\hat{y}_i - y_i)^2$ such that $\|\boldsymbol{\theta}\|^2 \leq K$

Regularization to avoid overfitting (2/2)

- New target: $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \frac{1}{2}\sum(\hat{y}_i - y_i)^2$ such that $\|\boldsymbol{\theta}\|^2 \leq K$
- This is equivalent to the following problem with some λ
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{2}\sum(\hat{y}_i - y_i)^2 + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2\right]$
- New objective function: minimizing both training error and the L2-norm of $\boldsymbol{\theta}$
  - Solution: $\boldsymbol{\theta} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}$

Regularized linear regression

- Regularized linear regression
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{2}\sum(\hat{y}_i - y_i)^2 + R(\boldsymbol{\theta})\right]$
  - $R(\boldsymbol{\theta})$: regularization
- L2-regularization (**Ridge regression**)
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{2}\sum(\hat{y}_i - y_i)^2 + \lambda\|\boldsymbol{\theta}\|^2\right]$
- L1-regularization (**Lasso**)
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{2}\sum(\hat{y}_i - y_i)^2 + \lambda\|\boldsymbol{\theta}\|_1\right]$

Combining Ridge and Lasso

- Elastic net regularization
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}}[\sum(\hat{y}_i - y_i)^2 + \lambda_1\|\boldsymbol{\theta}\|_1 + \lambda_2\|\boldsymbol{\theta}\|^2]$
  - When $\lambda_1 = 0$ and $\lambda_2 > 0$ ➔ Ridge regression
  - When $\lambda_1 > 0$ and $\lambda_2 = 0$ ➔ Lasso

How to select λ?

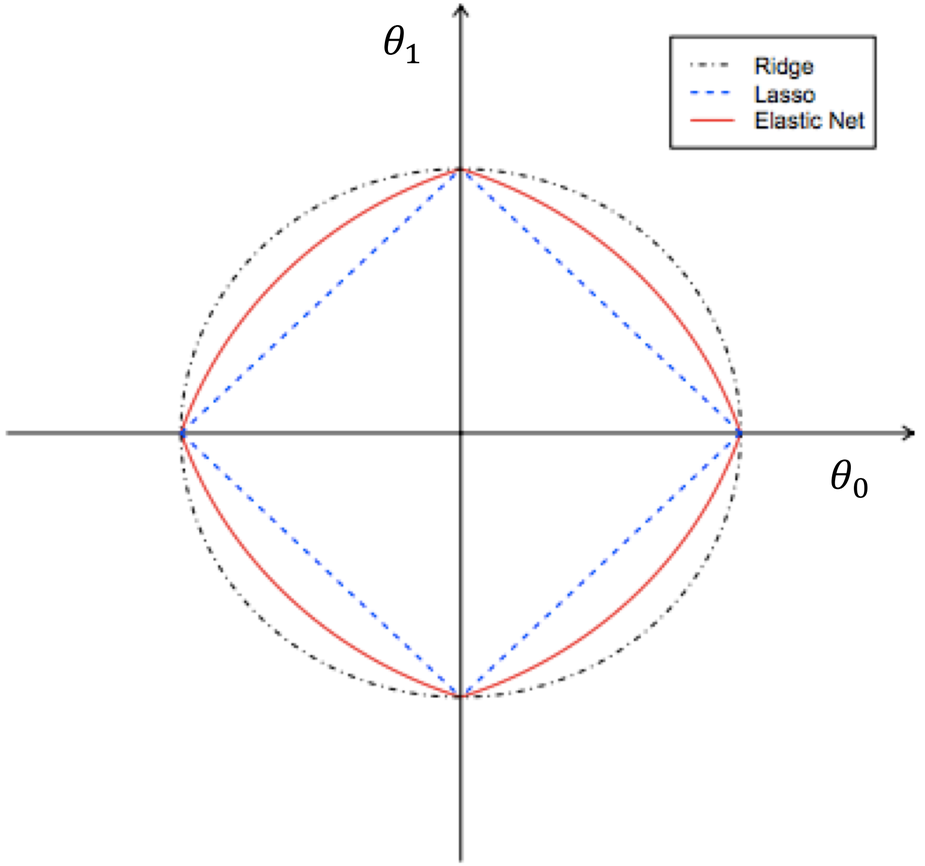
- Split the data into training and test datasets
  - Based on the training data, train the models by different λ's
  - Based on the test data, calculate the test performance of all the models (of different λ's)
  - Select the λ with the best test performance
- Cross validation
  - More to come in future lectures

Ridge, Lasso, and Elastic-net

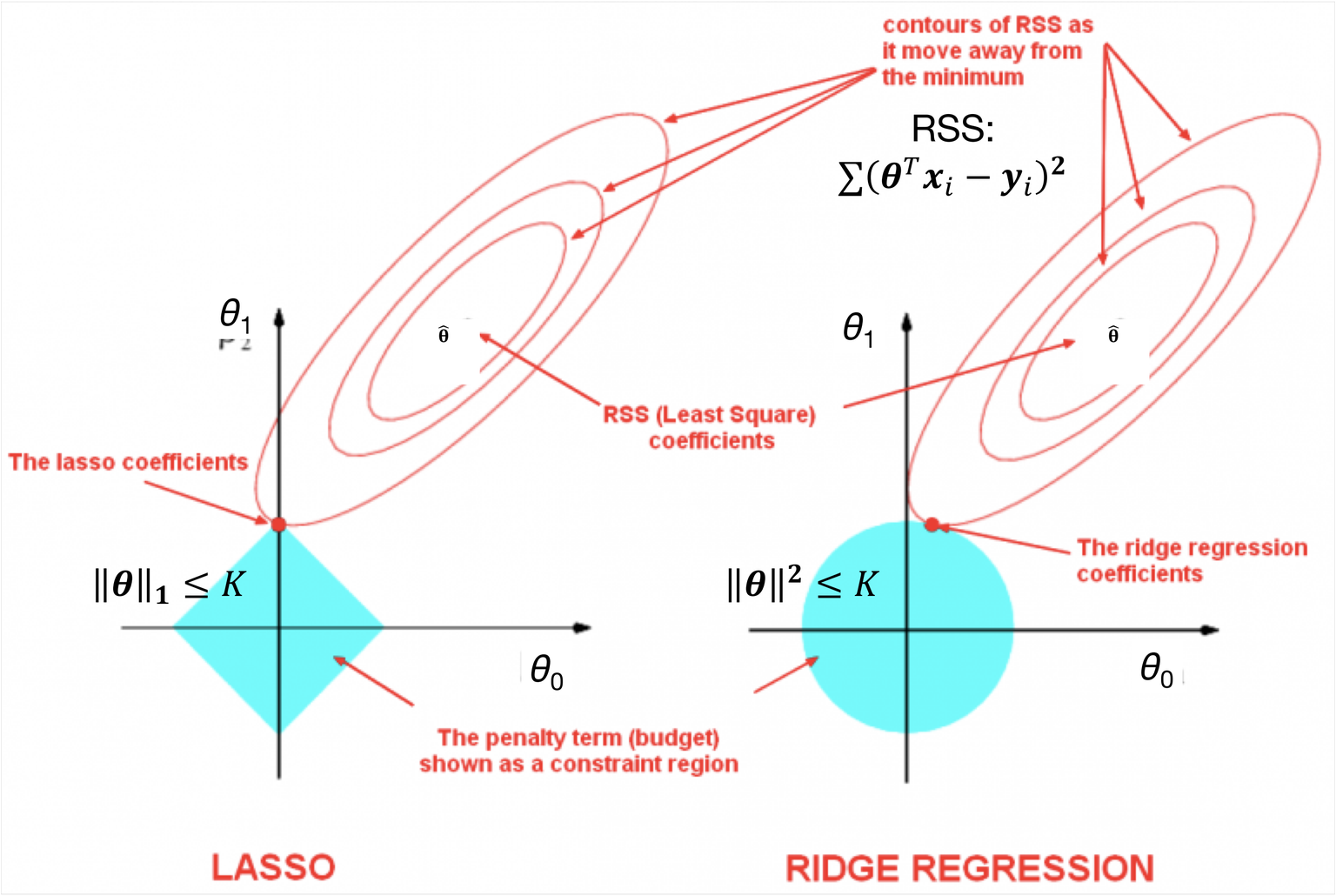
- Ridge
  - Good if many features have small/medium sized effects
- Lasso
  - Good if only a few features with a medium/large effect
- Elastic-net
  - A generalization of Ridge and Lasso, probably combine the strength of both

Geometry of Ridge, Lasso, and Elastic net

- Ridge: $\|\boldsymbol{\theta}\|^2 = \sum_i \theta_i^2 \leq K$
- LASSO: $\|\boldsymbol{\theta}\|_1 = \sum_i |\theta_i| \leq K$



Why Lasso zeros coefficients

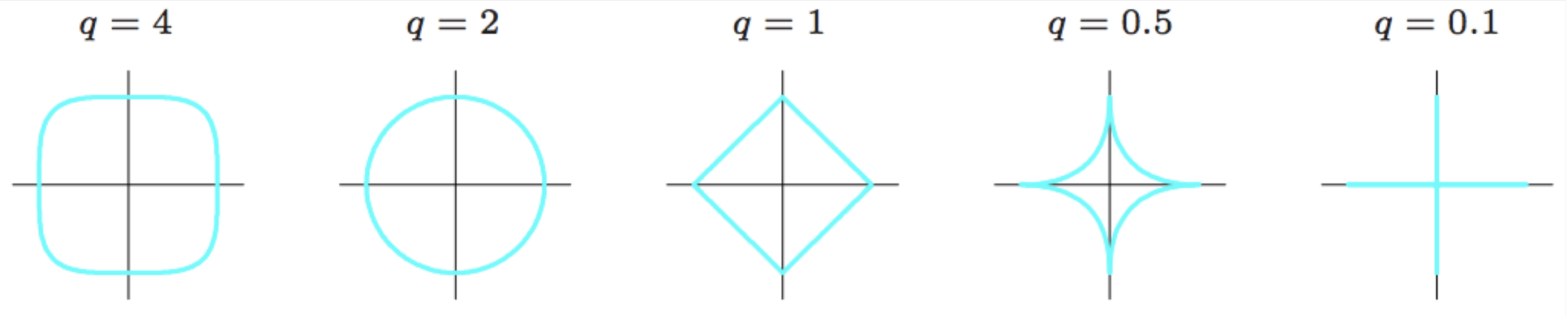


A numerical explanation

- Ridge
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{2}\sum(\hat{y}_i - y_i)^2 + \lambda\|\boldsymbol{\theta}\|^2\right]$
    - Changing θ_j from 2 to 1 reduces the cost by $\lambda(2^2 - 1^2) = 3\lambda$
    - Changing θ_j from 1 to 0 reduces the cost by $\lambda(1^2 - 0^2) = \lambda$
- Lasso
  - $\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{2}\sum(\hat{y}_i - y_i)^2 + \lambda\|\boldsymbol{\theta}\|_1\right]$
    - Changing θ_j from from 2 to 1 reduces the cost by $\lambda(2 - 1) = \lambda$
    - Changing θ_j from from 1 to 0 reduces the cost by $\lambda(1 - 0) = \lambda$

Ridge tends to shrink large coefficients to smaller ones, but not to shrink small coefficients to zero

Generalize the regularization term



$$\boldsymbol{\theta} \coloneqq \arg\min_{\boldsymbol{\theta}} \left[\frac{1}{2} \sum (\hat{y}_i - y_i)^2 + \lambda \|\boldsymbol{\theta}\|^q \right], q \geq 0$$

- q=1: Lasso
- q=2: Ridge regression
- A smaller q tends to shrink the coefficients

Gradient descent for ridge regression

- Gradient descent (GD)
  - $\boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \alpha \nabla J\left(\boldsymbol{\theta}^{(k)}\right)^T$
- Ridge regression
  - $J(\boldsymbol{\theta}) = \frac{1}{2}(\hat{\boldsymbol{y}} - \boldsymbol{y})^T(\hat{\boldsymbol{y}} - \boldsymbol{y}) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$
    $= \frac{1}{2}(\boldsymbol{X\theta} - \boldsymbol{y})^T(\boldsymbol{X\theta} - \boldsymbol{y}) + \frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$
  - $\nabla J(\boldsymbol{\theta}) = (\boldsymbol{X\theta} - \boldsymbol{y})^T\boldsymbol{X} + \lambda\boldsymbol{\theta}^T$
  - $\nabla J(\boldsymbol{\theta})^T = \boldsymbol{X}^T(\boldsymbol{X\theta} - \boldsymbol{y}) + \lambda\boldsymbol{\theta}$

Gradient descent for ridge regression

1 k := 0

2 Initialize $\boldsymbol{\theta}^{(k)}$

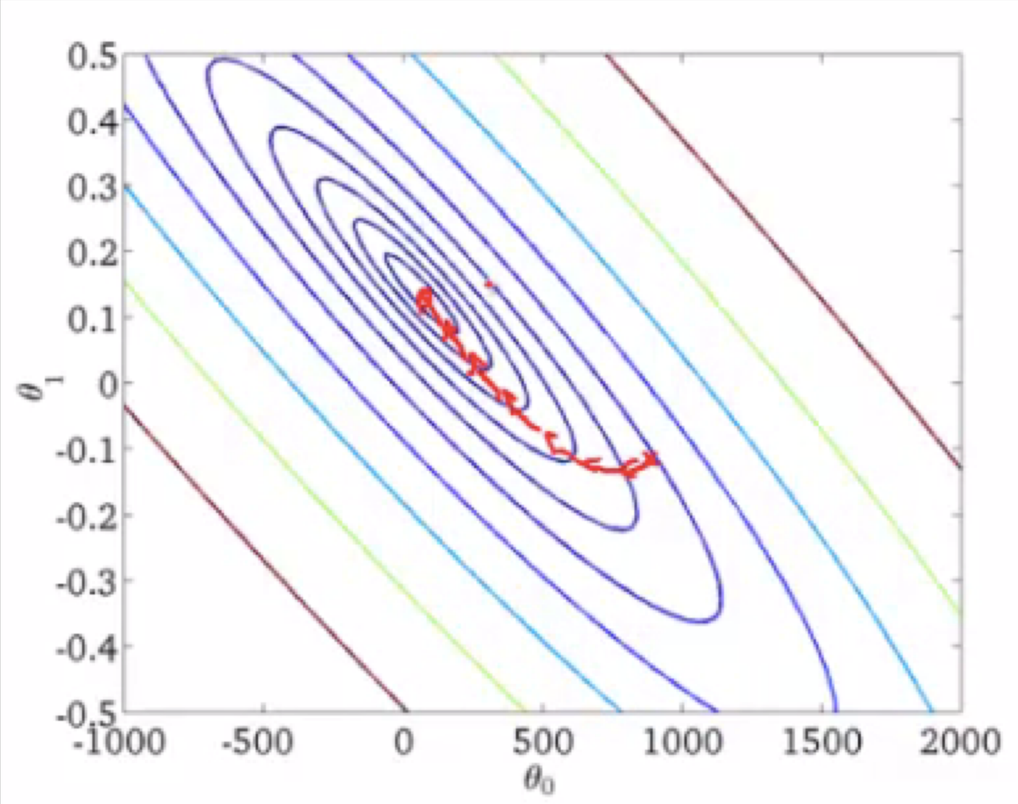
3 while not converge:

4 $\quad \boldsymbol{g} := \boldsymbol{X}^T\left(\boldsymbol{X}\boldsymbol{\theta}^{(k)} - \boldsymbol{y}\right) + \lambda\boldsymbol{\theta}^{(k)}$

5 $\quad \boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \alpha\boldsymbol{g}$

6 $\quad k$:= k+1

Gradient descent on large dataset



- In GD, the gradient is computed by
  - $\boldsymbol{g} \coloneqq \boldsymbol{X}^T\left(\boldsymbol{X}\boldsymbol{\theta}^{(k)} - \boldsymbol{y}\right) + \lambda\boldsymbol{\theta}^{(k)}$
  - ***The entire training set*** is examined at each step
  - ***Slow*** when the # of training instances is large

Stochastic gradient descent

- Optimize ***one example*** at a time

1 k := 0

2 Initialize $\boldsymbol{\theta}^{(k)}$

3 while not converge:

4 $\boldsymbol{g} := \boldsymbol{x}_i\left(\boldsymbol{x}_i^T\boldsymbol{\theta}^{(k)} - \boldsymbol{y}_i\right) + \lambda\boldsymbol{\theta}^{(k)}$

5 $\boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \alpha\boldsymbol{g}$

6 k := k+1

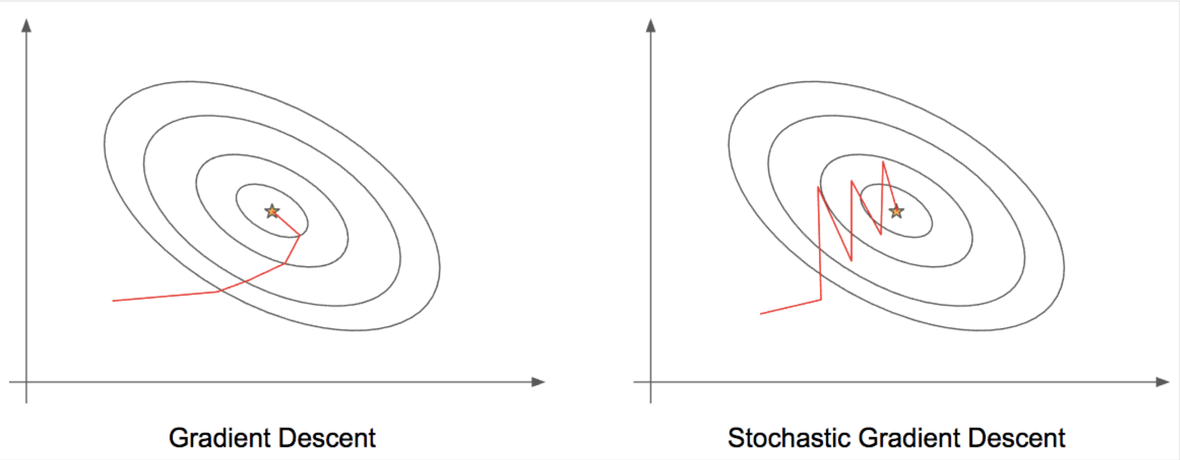
7 Get next data instance i

Only one training instance is examined at each step

- Why does SGD work? ➔ $E\left(\nabla J_i(\boldsymbol{\theta})\right) = \frac{1}{n}\sum\nabla J_i(\boldsymbol{\theta}) = \nabla J(\boldsymbol{\theta})$

GD vs SGD

- # steps
  - GD: fewer steps
  - SGD: more steps
- Computation of each step
  - GD: look through *all* the training instances
  - SGD: look only *one* training instance



Pros and cons of SGD

- Pros
  - When the training data is large with some (near) redundant instances, SGD is usually much faster to converge than GD
  - Supports online learning
  - Sometimes can pass local minimum (if any)
- Cons
  - Tends to bouncing around the minimum

Mini-batch gradient descent

- Optimize ***few examples*** at a time

1 k := 0

2 Initialize $\boldsymbol{\theta}^{(k)}$

3 while not converge:

4 $\boldsymbol{g} := \boldsymbol{x}_{i:j}^{T}\left(\boldsymbol{x}_{i:j}\boldsymbol{\theta}^{(k)} - \boldsymbol{y}_{i:j}\right) + \lambda\boldsymbol{\theta}^{(k)}$

5 $\boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \alpha\boldsymbol{g}$

6 k := k+1

7 Get next batch (i,i+1,…,j)

Training instances (*i*, *i*+1, …, *j*) are examined

Gradient descent/stochastic gradient descent/mini-batch gradient descent

- All of them iteratively update the parameters such that the target function gradually becomes smaller
- If we have *n* training instances
  - (Batch) gradient descent: every parameter update requires seeing ***all*** training instances once
  - Stochastic gradient descent: every parameter update requires seeing ***one*** of *n* training instances
  - Mini-batch: every parameter update requires seeing ***b*** training instances (if batch size = *b*)

Summary (1/4)

- Linear regression
  - $\hat{y}_i = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \cdots + \theta_d x_{i,d}$
  - Find good θ_0, θ_1, …, θ_d such that $\hat{y}_i \approx y_i \ \forall i$
    - Commonly used distance between $\hat{y}_i$ and y_i: RSS
- Non-linear regression can be achieved by generating non-linear features

Summary (2/4)

- Linear regression with constraints on $\boldsymbol{\theta}$
  - Prevent overfitting
  - Limiting $\boldsymbol{\theta}$ by L2-norm: Ridge regression
  - Limiting $\boldsymbol{\theta}$ by L1-norm: Lasso
    - Feature selection
  - Limiting $\boldsymbol{\theta}$ by both L1-norm and L2-norm: Elastic net

Summary (3/4)

- Convex function
  - Local minimum equals global minimum
  - Iteratively adjust $\boldsymbol{\theta}$ by gradient descent to reduce $J(\boldsymbol{\theta})$
    - $\boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \alpha \boldsymbol{g}$

Summary (4/4)

- For ridge regression
  - Gradient descent
    - $\boldsymbol{g} \coloneqq \boldsymbol{X}^T\left(\boldsymbol{X}\boldsymbol{\theta}^{(k)} - \boldsymbol{y}\right) + \lambda\boldsymbol{\theta}^{(k)}$
  - Stochastic gradient descent
    - $\boldsymbol{g} \coloneqq \boldsymbol{x}_i\left(\boldsymbol{x}_i^T\boldsymbol{\theta}^{(k)} - \boldsymbol{y}_i\right) + \lambda\boldsymbol{\theta}^{(k)}$
  - Mini-batch gradient descent
    - $\boldsymbol{g} \coloneqq \boldsymbol{x}_{i:j}^T\left(\boldsymbol{x}_{i:j}\boldsymbol{\theta}^{(k)} - \boldsymbol{y}_{i:j}\right) + \lambda\boldsymbol{\theta}^{(k)}$

Quiz

- What is regularization terms and why do we need them?
- What is Lasso?
- What is Ridge regression?
- Compared to Ridge regression, why does Lasso tend to shrink some parameters to zero?

Evaluation

Mean Square Error (MSE)

$$MSE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum(y_i - \hat{y}_i)^2}{n}$$

- Similar metric: Root Mean Squared Error (RMSE)
- Criticism:
  - Not a normalized measure
  - Tend to be influenced by extreme values

Mean Absolute Error (MAE)

$$MAE(\mathbf{y}, \hat{\mathbf{y}}) = \frac{\sum |y_i - \hat{y}_i|}{n}$$

- Usually smaller than MSE
- Criticism:
  - Not a normalized measure
  - Tend to be influenced by outliers

Median Absolute Error (MedAE)

$$MedAE(\mathbf{y}, \hat{\mathbf{y}}) = \text{median}(|y_1 - \hat{y}_1|, |y_2 - \hat{y}_2|, \ldots, |y_n - \hat{y}_n|)$$

- Robust to the extreme values
- Criticism:
  - Not a normalized measure

R^2 score (coefficient of determination)

$$R^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

- Mean model: a constant function that always outputs $\bar{y}$ the mean of the target variable in the training dataset
- $\sum(y_i - \hat{y}_i)^2$: sum of square error (SSE)
- $\sum(y_i - \bar{y})^2$: sum of square total (SST) i.e., SSE of the mean model
- Max: 1 (when SSE=0, i.e., all predictions are correct)
- Min value: -∞ (SSE can be arbitrarily worse)
- $R^2(\mathbf{y}, \hat{\mathbf{y}}) = 0$: the performance of the prediction is the same as the mean model