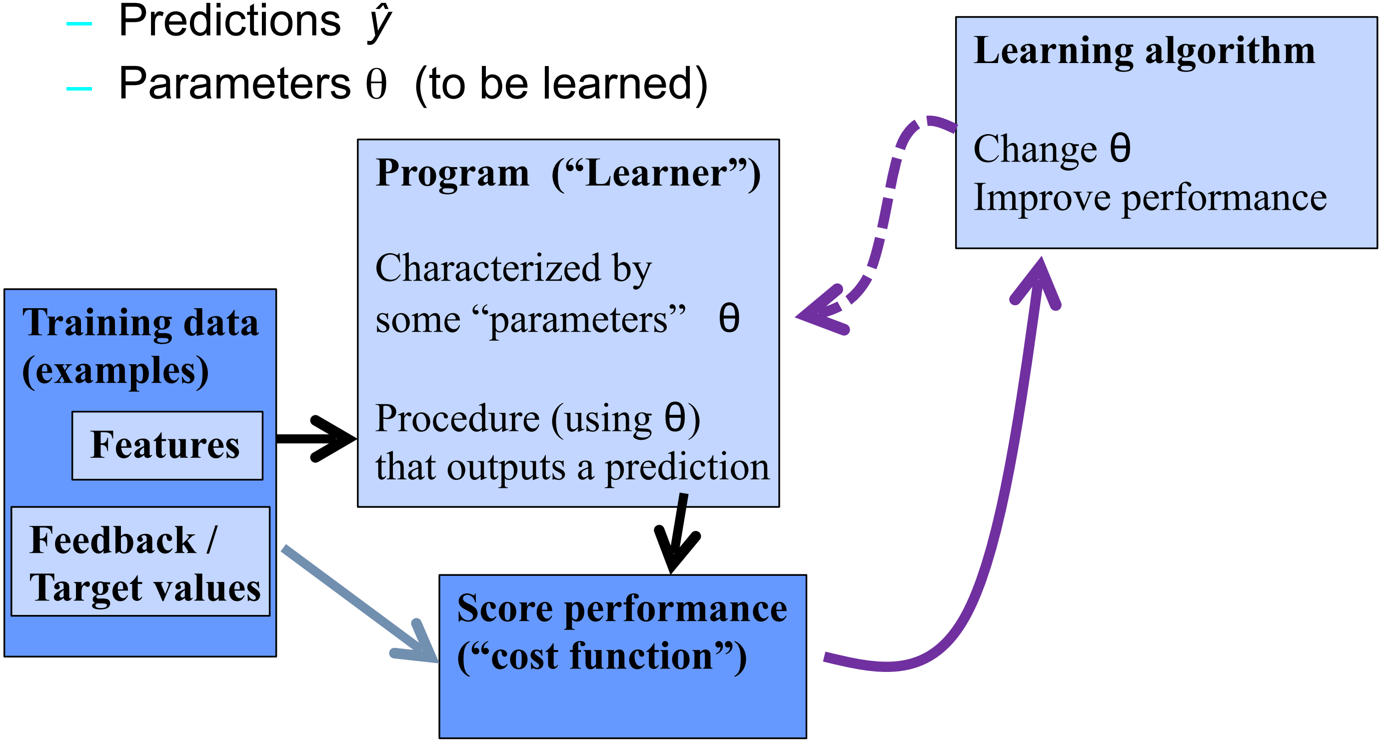
Linear regression

Hung-Hsuan Chen

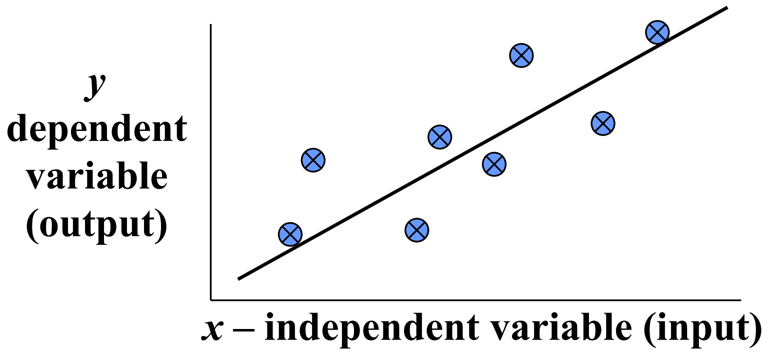
Adapted from Alexander Ihler (UCI) and Tony R. Martinez (BYU)

Parametric supervised learning

- Notation
  - Features x
  - Targets y
  - Predictions $\hat{y}$
  - Parameters θ (to be learned)



Linear regression

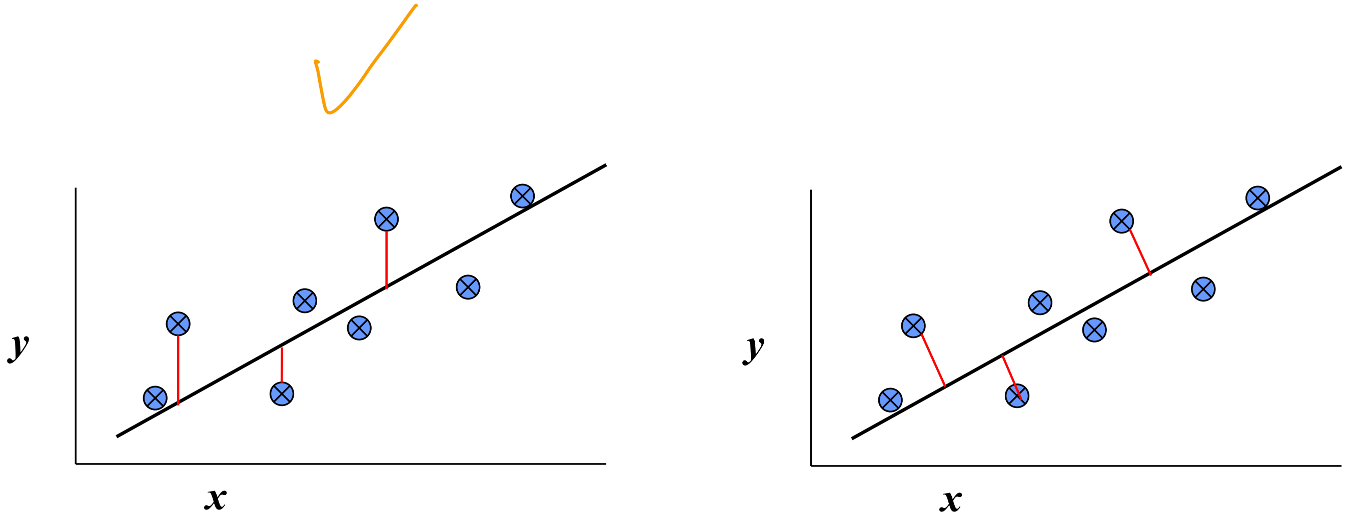


"Predictor":

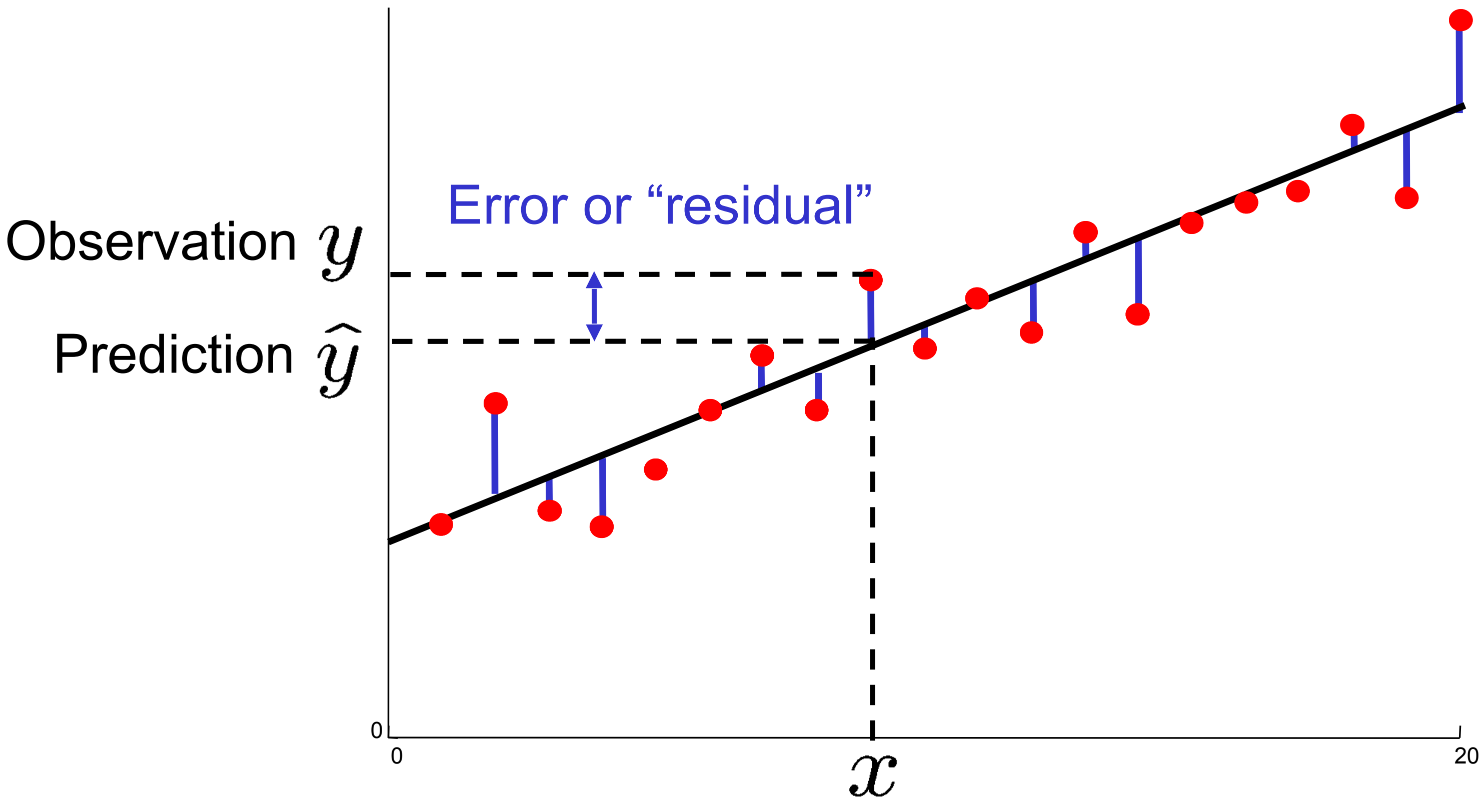
$$\hat{y} = \theta_0 + \theta_1 x$$

- 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

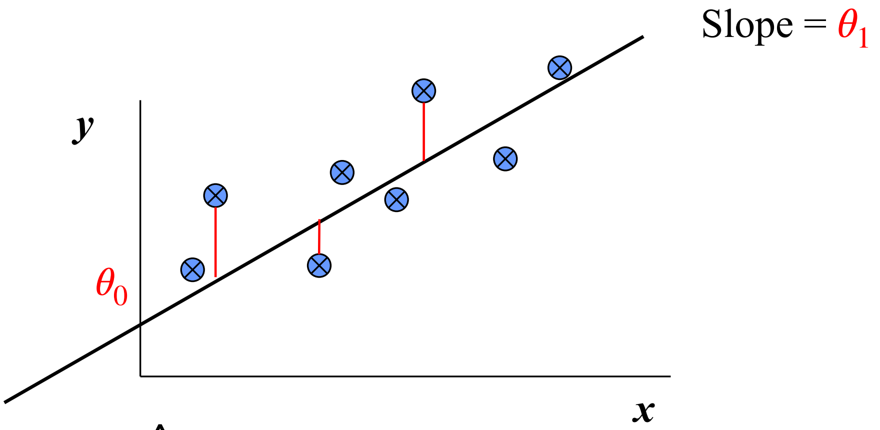
- For now, assume just one (input) independent variable *x*, and one (output) dependent variable *y*
  - Multiple linear regression assumes an input vector **x**
- We will "fit" the points with a line
  - In a high dimensional space, we will fit the points with a hyper-plane
- Which line should we use?
  - Choose an objective function
  - Typically, we choose sum squared error (SSE) (or its variants)
    - $$\frac{1}{n}\sum\left(\hat{y}_i - y_i\right)^2 = \frac{1}{n}\sum\left(\mathrm{residual_i}\right)^2$$
    - Choices with the same result: $\frac{\mathbf{1}}{2\boldsymbol{n}}\sum\left(\hat{y}_i - y_i\right)^2$ or $\frac{\mathbf{1}}{2}\sum\left(\hat{y}_i - y_i\right)^2$
  - You may choose other objective functions
    - More to come in future lectures

why $\frac{1}{2}$? 微分後
好看

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:
  - $\theta_1 = \dfrac{n \sum x_i y_i - \sum x_i \sum y_i}{n \sum x_i^2 - \left(\sum x_i \right)^2}$
  - $\theta_0 = \dfrac{\sum y_i - \theta_1 \sum x_i}{n}$
  - *n*: number of training instances

Visualzing θ_0 and θ_1



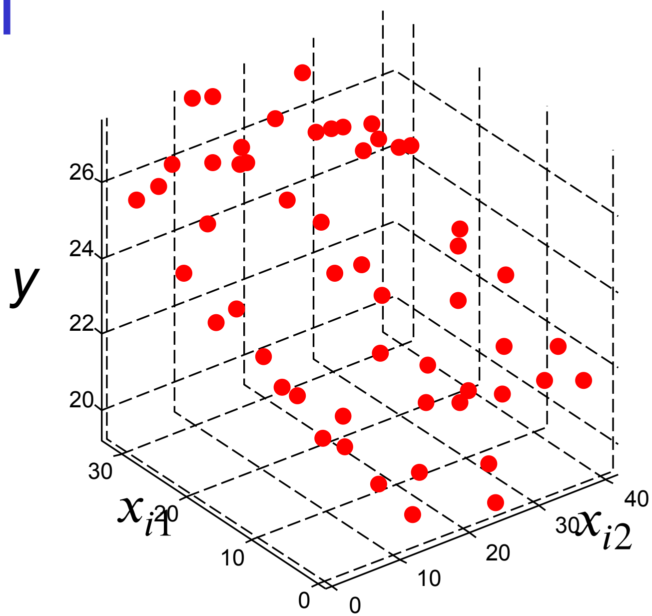
$$\hat{y} = \theta_0 + \theta_1 x$$

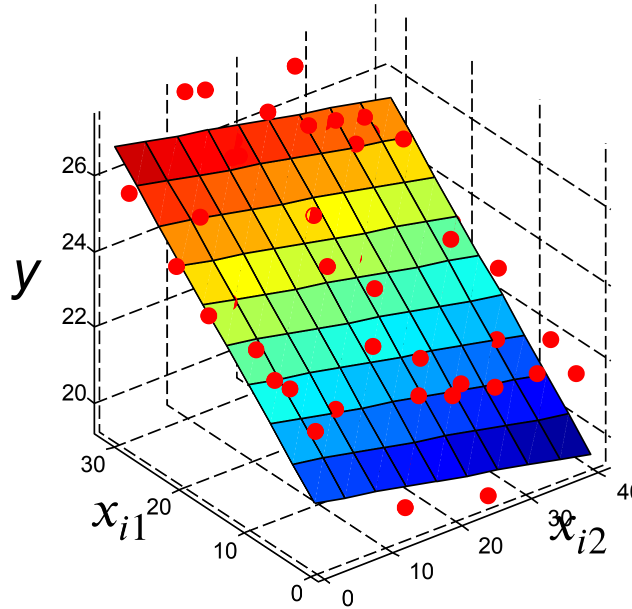
How to derive the value of the coefficients?

- $$J(\theta_0, \theta_1) = \frac{1}{2n} \sum \left(\hat{y}_i - y_i \right)^2$$
  $$= \frac{1}{2n} \sum \left((\theta_0 + \theta_1 x_i) - y_i \right)^2,$$
- 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?





$$\hat{y}(\mathbf{x}_i) = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \ \ldots + \theta_d x_{i,d} = \boldsymbol{\theta}^T \mathbf{x}_i, \text{ where}$$

$$\mathbf{x}_i = \left[1, x_{i,1}, x_{i,2}, \ \ldots, x_{i,d}\right]^T$$

$$\boldsymbol{\theta} = \left[\theta_0, \ \theta_1, \theta_2, \ \ldots, \ \theta_d\right]^T$$

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:
  - $\boldsymbol{X} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix}, \boldsymbol{y} = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}$
  - (We assume no coefficient parameter here)
- Find $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 \\ \vdots \\ \theta_d \end{bmatrix}$ such that $\left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right)^T \left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right)$ is minimized,

  where
  - $\hat{\boldsymbol{y}} = \boldsymbol{X\theta}$
- The solution is $\boldsymbol{\theta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$

How to derive the value of the coefficient vector?

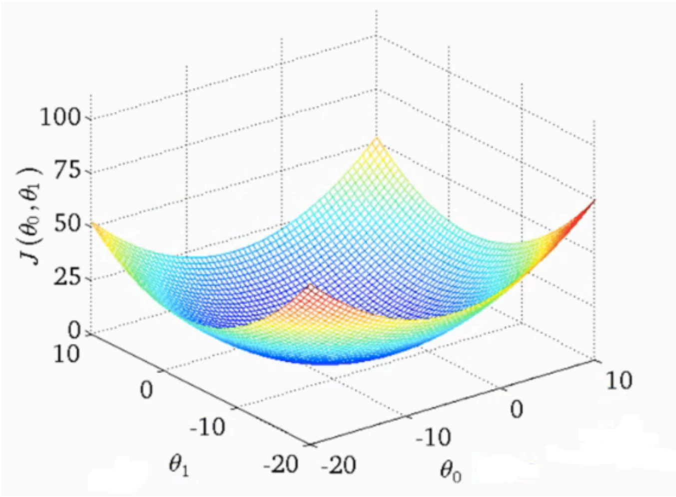
- $$J(\theta) = \frac{1}{2}\left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right)^T\left(\hat{\boldsymbol{y}} - \boldsymbol{y}\right)$$
  $$= \frac{1}{2}\left(\boldsymbol{X}\theta - \boldsymbol{y}\right)^T\left(\boldsymbol{X}\theta - \boldsymbol{y}\right)$$
- Set
  - $\nabla J(\theta) = \mathbf{0}$

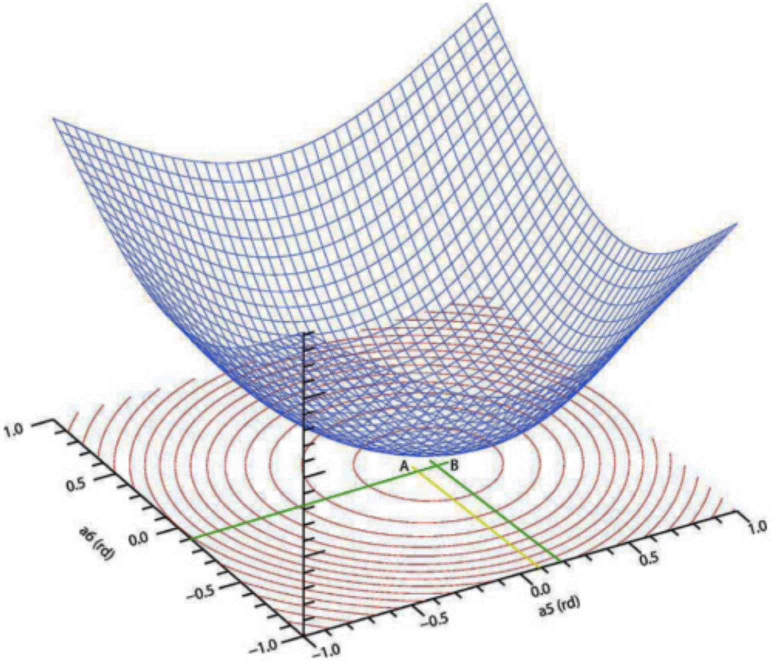
  to solve $\boldsymbol{\theta}$

Another way to find $\boldsymbol{\theta}$ for the linear regression problem

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

Plotting the objective function (3D and contour plots)



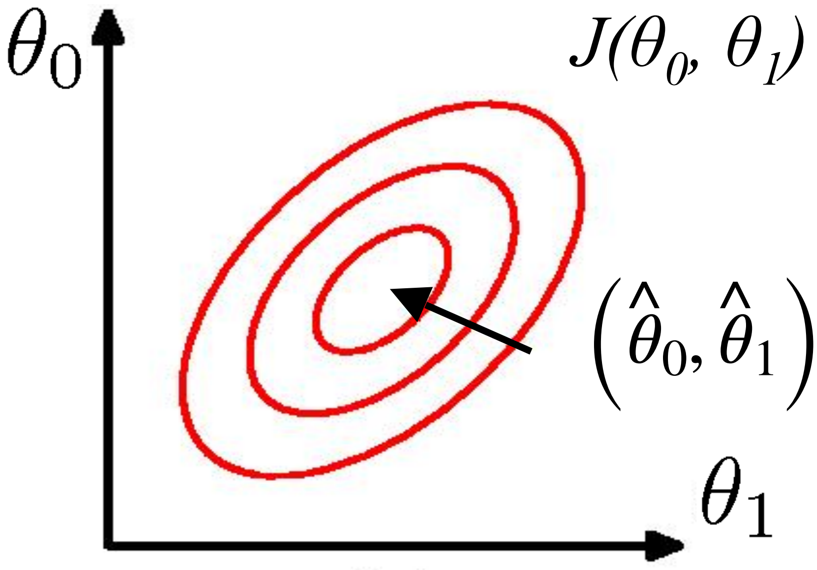


Keep moving ”downward” to reach the minimum

Finding good parameters

- Want to find parameters which minimize our error…

- Think of a cost "surface": error residual for that θ…



$$\left(\hat{\theta}_0, \hat{\theta}_1\right) = \underset{\theta_0, \theta_1}{\operatorname{argmin}} J(\theta_0,\ \theta_1)$$

Gradient descent

$\theta_0^{(k)} = 5$

$\theta_1^{(k)} = -4$

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

$J = \left(\underbrace{y - (\theta_0 + \theta_1 x)}\right)^2$

$\frac{\partial J}{\partial \theta_0} = 2(\quad)(-1)$

$\frac{\partial J}{\partial \theta_1} = 2(\quad)(-x)$

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

α is a small positive number

$$\frac{\partial}{\partial \theta_i} J(\theta) = \frac{1}{n} \sum_j \left(\hat{y}_j - y_j\right) x_{ji},$$

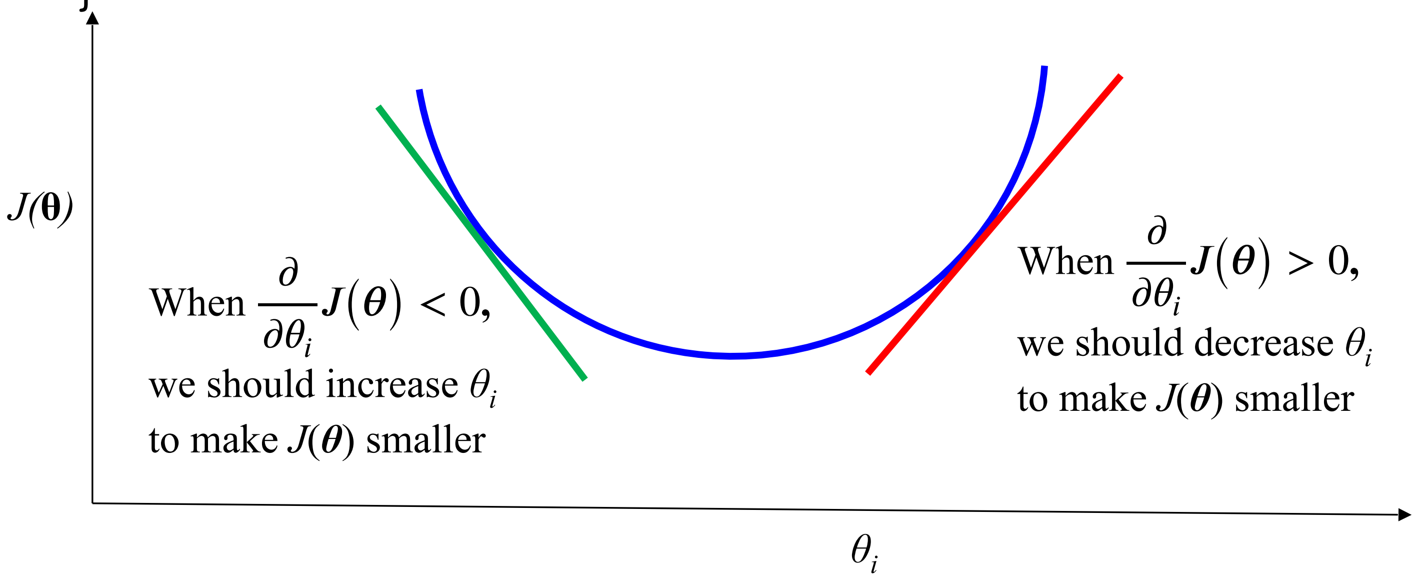
$\frac{1}{n}$ is usually ignored

Why gradient descent work?

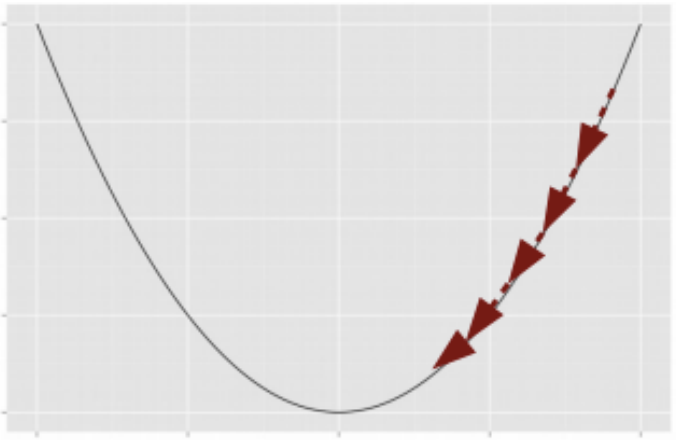
```
Repeat until converge {
  for (i=1,2, ..., d){
```

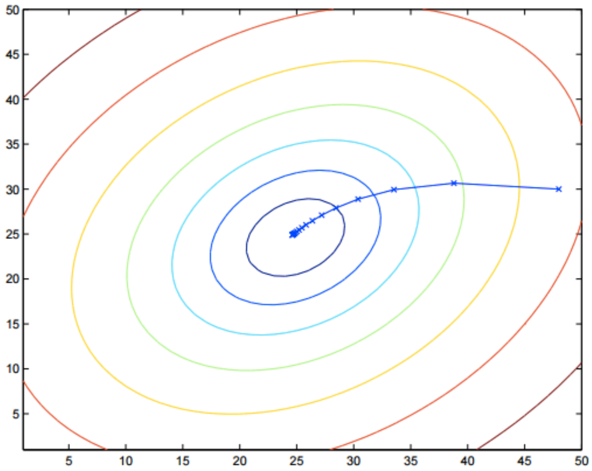
$$\theta_i := \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\boldsymbol{\theta}) = \theta_i - \alpha \sum_j \left(\hat{y}_j - y_j \right) x_{ji}$$

```
  }
}
```

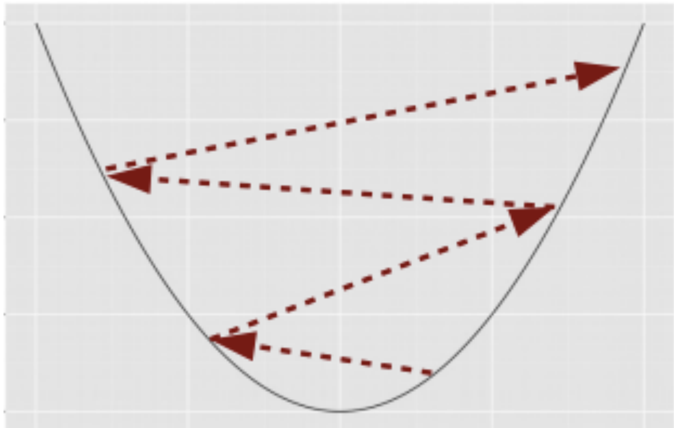


Gradient descent (one feature and two features)





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 $\alpha \in [0.001,\ 1]$
  - Shrink α as k becomes larger

We now have two tools to solve linear regression problem

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

- Gradient descent

```
Repeat until converge {
  for (i=1,2, ..., d){
```
$$\theta_i = \theta_i - \alpha \sum_j \left(\hat{y}_j - y_j\right)x_{ji}$$
```
  }
}
```

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 X is sparse, $(X^TX)^{-1}$ may not be sparse
  - If n=1M, d=100K, $(X^TX)^{-1}$'s dimension is $(100k)^2 = 10^{10}$
    - If 8 bytes per entry, storing $(X^TX)^{-1}$ requires 80GB ➔ typically infeasible on a single machine
- Computation time:
  - Computing matrix multiplication X^TX takes $O(nd^2)$
  - Let $\left(\boldsymbol{X}^T\boldsymbol{X}\right) = \boldsymbol{P}$, computing P^{-1} takes $O(d^{2.373})$ to $O(d^3)$
    - Depending on the inversion algorithm
  - Let $(X^TX)^{-1} = Q$, computing QX^T takes $O(nd^2)$
  - Let $\left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T = \boldsymbol{R}$, computing Ry takes O(nd)
  - Total: $O(nd^2+d^{2.373}+nd^2)$
- If d is small, close form solution is probably acceptable

n : number of Instance

d : number of Features

Consider using Gradient Descent

```
Repeat until converge {
   for (j=1, 2, …, n){
```
$$z_j = \left(\boldsymbol{\theta}^T \mathbf{x}_j - y_j\right)$$
```
   }
   for (i=1, 2, ..., d){
```
$$\theta_i = \theta_i - \alpha \sum_j z_j x_{ji}$$
```
   }
}
```

- Required space:
  - **X** = [x_{ij}], ***y***, **θ**, and **z**
  - If n=1M, d=100K, and X is sparse
    - If 8 bytes per entry, we need: (nnz(X)+1M+100K+1M)*8bytes
    - Much more efficient than 80GB
- Computation time:
  - $z_j = \left(\boldsymbol{\theta}^T \mathbf{x}_j - y_j\right)$ 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 in both memory usage and computing time
- 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 ☺️
  - More to come in future lectures

Quiz

?

- If the features are all categorical
  - Can you apply linear regression?
  - Can you apply decision tree?
  - Can you apply knn?
- If all features are numerical, which one runs faster during "test" phase?
  - Linear regression? Decision tree? KNN?

Lg little faster than Dt

When to stop updating?

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

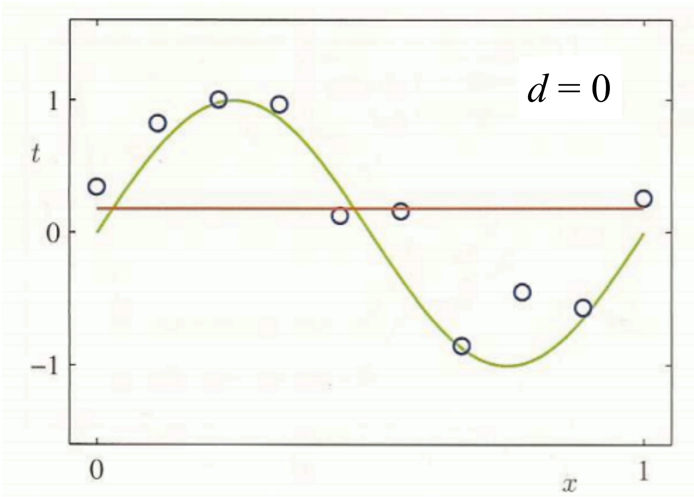
Fitting non-linear data to linear model

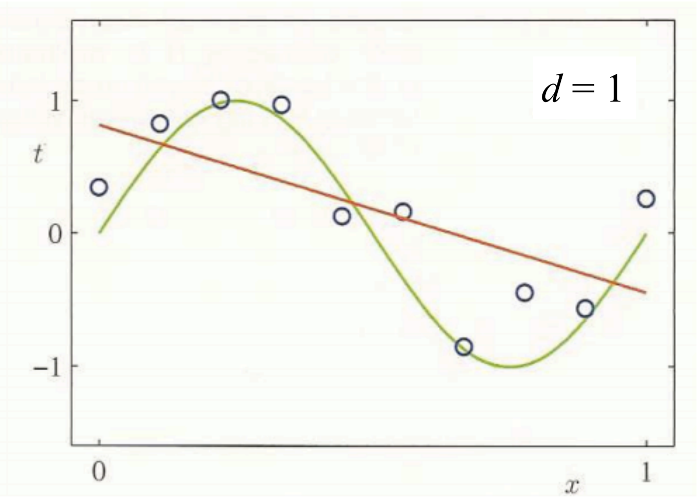
- $\hat{y}(\mathbf{x_i}) = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \ \ldots + \theta_d x_{i,d}$
  - Linear model
- We may generate the higher degree terms as the new features
  - $\hat{y}(\mathbf{x_i}) = \left(\theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \ldots + \theta_d x_{i,d}\right)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

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

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



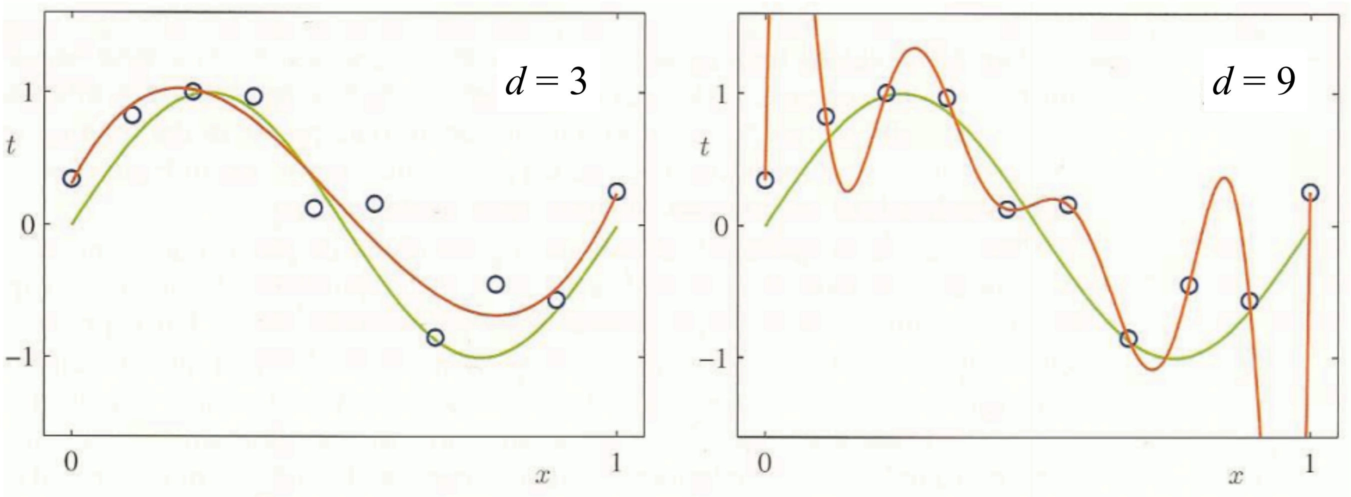


$\hat{y}_i = \theta_0$

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

Target: $\sin(2\pi x)$ + noise (overfitting)

$\hat{y}_i = \theta_0 + \sum_{i=1}^{9} \theta_i x^i$



$\hat{y}_i = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

Observation

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

	$d = 0$	$d = 1$	$d = 3$	$d = 9$
	0.19	0.82	0.31	0.35
		-1.27	7.99	232.37
			-25.43	-5321.83
			17.37	48568.31
				231639.30
				640042.26
				-1061800.52
				1042400.18
				-557682.99
				125201.43

Regularization to avoid overfitting

- Introduce a penalty term for the size of the weights
- Un-regularized regression (the original objective function)
  - $J(\boldsymbol{\theta}) = \frac{1}{2}\sum\left(\hat{y}_i - y_i\right)^2 = \frac{1}{2}\sum\left(\boldsymbol{\theta}^T\mathbf{x}_i - y_i\right)^2$
  - $\boldsymbol{\theta} := \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{2}\sum\left(\hat{y}_i - y_i\right)^2$
- Regularized regression (enforce the solution to have low L2-norm of $\boldsymbol{\theta}$)
  - $\boldsymbol{\theta} := \operatorname*{argmin}_{\boldsymbol{\theta}} \frac{1}{2}\sum\left(\hat{y}_i - y_i\right)^2$ such that $\left\|\boldsymbol{\theta}\right\|^2 \le K$

$\theta_1^2 + \theta_2^2 \cdots \theta_n^2$

Regularization to avoid overfitting

- Original objective function: minimizing the training error
  - $J(\boldsymbol{\theta}) = \frac{1}{2}\sum\left(\hat{y}_i - y_i\right)^2 = \frac{1}{2}\sum\left(\boldsymbol{\theta}^T\mathbf{x}_i - y_i\right)^2$
- New target
  - $\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \frac{1}{2}\sum\left(\hat{y}_i - y_i\right)^2$ such that $\left\|\boldsymbol{\theta}\right\|^2 \leq K$
- This is equivalent to the following problem with some λ
  - $\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\frac{1}{2}\sum\left(\hat{y}_i - y_i\right)^2 + \underbrace{\frac{\lambda}{2}\left\|\boldsymbol{\theta}\right\|^2}\right]$ → hyperparameter
- New objective function: minimizing training error and the L2-norm of $\boldsymbol{\theta}$ simultaneously
  - Solution: $\boldsymbol{\theta} = \left(\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}\right)^{-1}\mathbf{X}^T\mathbf{y}$

* if λ大

得到的θ小

Regularized linear regression

- Regularized linear regression
  - $\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\frac{1}{2} \sum \left(\hat{y}_i - y_i \right)^2 + R(\boldsymbol{\theta}) \right]$
  - $R(\boldsymbol{\theta})$: regularization
- L2-regularization (**Ridge regression**)
  - $\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\frac{1}{2} \sum \left(\hat{y}_i - y_i \right)^2 + \lambda \|\boldsymbol{\theta}\|^2 \right]$ $\rightarrow \theta_1^2 + \theta_2^2 \ldots$
- L1-regularization (**Lasso**)
  - $\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\frac{1}{2} \sum \left(\hat{y}_i - y_i \right)^2 + \lambda \|\boldsymbol{\theta}\|_1 \right]$ $\rightarrow |\theta_1| + |\theta_2| + \ldots$

Combining Ridge and Lasso

- Elastic net regularization
  - $$\boldsymbol{\theta} := \operatorname*{argmin}_{\boldsymbol{\theta}} \left[\sum \left(\hat{y}_i - y_i \right)^2 + \lambda_1 \left\| \boldsymbol{\theta} \right\|_1 + \lambda_2 \left\| \boldsymbol{\theta} \right\|^2 \right]$$
  - When $\lambda_1 = 0$ and $\lambda_2 > 0$ ➔ Ridge regression
  - When $\lambda_1 > 0$ and $\lambda_2 = 0$ ➔ Lasso

How to select λ?

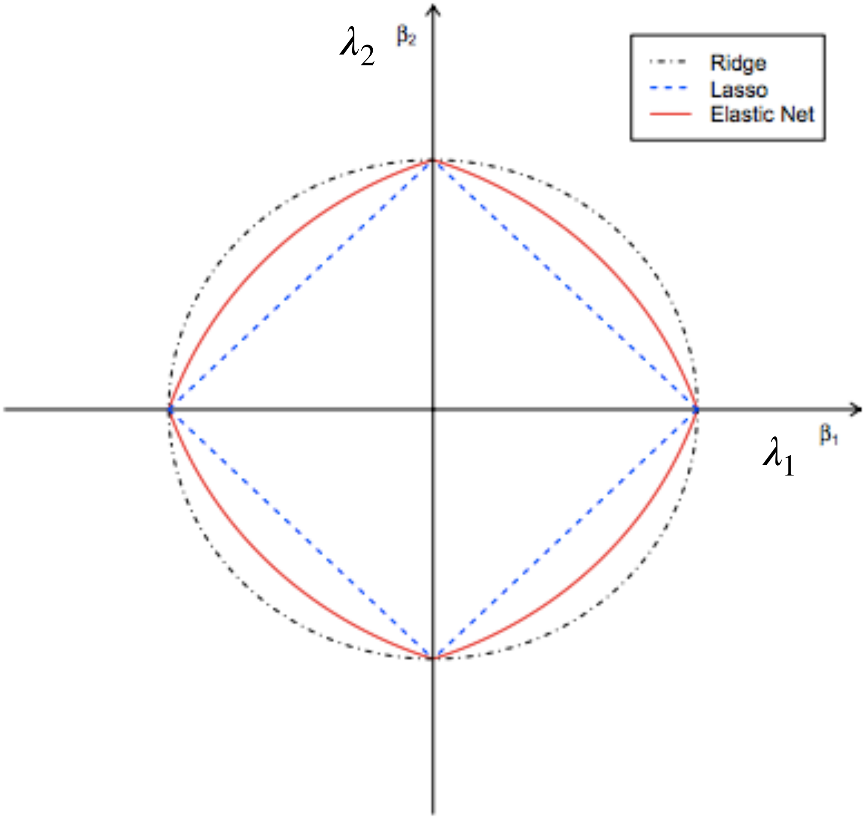
- Split the data into training and test datasets
  - Based on the training data, train the models by different λ's
  - Based on the test data, calculate the test performance of all the models (of different λ's)
  - Select the λ with the best test performance
- Cross validation
  - More to come in future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

Properties of Ridge and Lasso

- Ridge 每个 feature 皆有 effect
  - Good if many features have small/medium sized effects
- Lasso 只有特定 feature 有 effect
  - Good if only a few features with a medium/ large effect

Geometry of Ridge, Lasso, and Elastic net

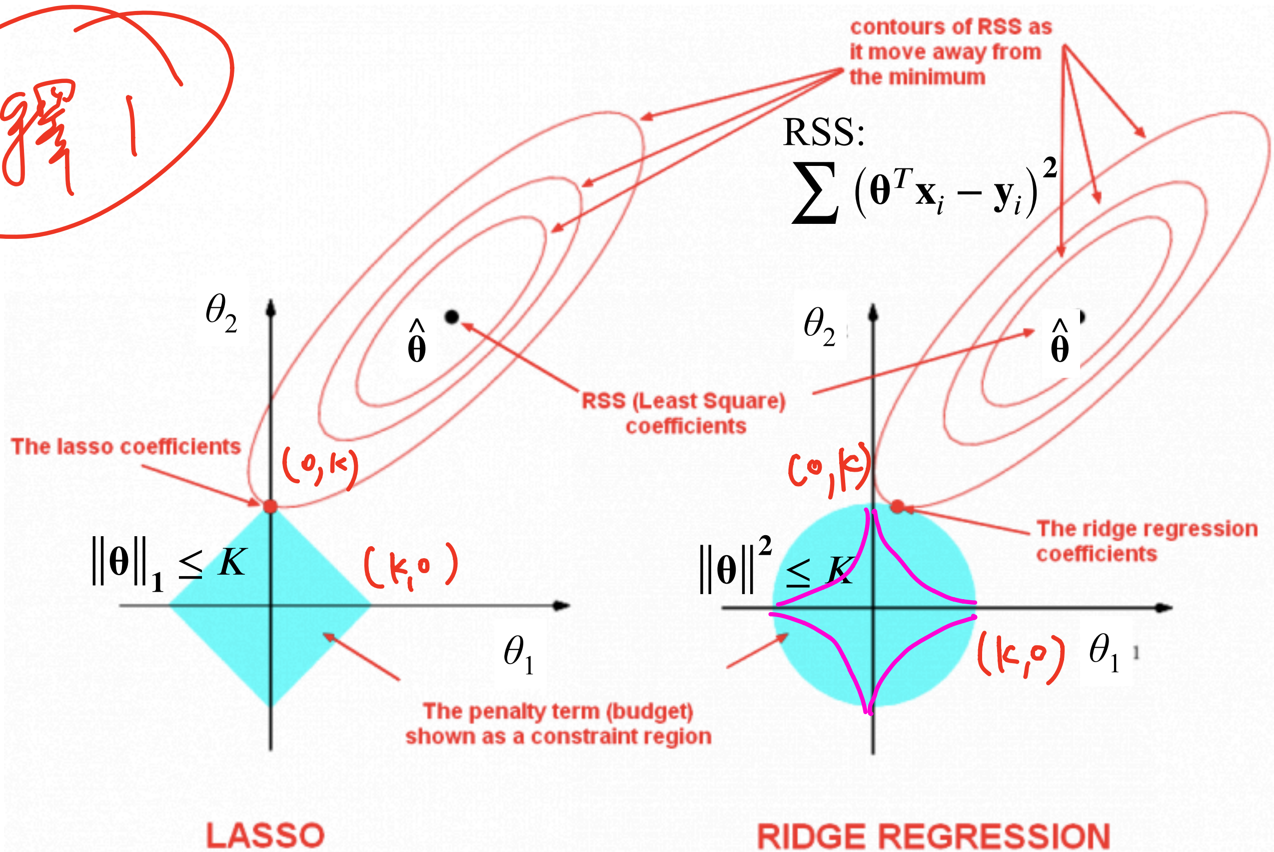
Elastic net: $\lambda_1 = \lambda_2 = 0.5$



ex: $|\theta_1|^{0.01} + |\theta_2|^{0.01} \le k$

Why Lasso zeros coefficients

解釋1



$|\theta_1| + |\theta_2| \le k$

$\theta_1^2 + \theta_2^2 \le k$

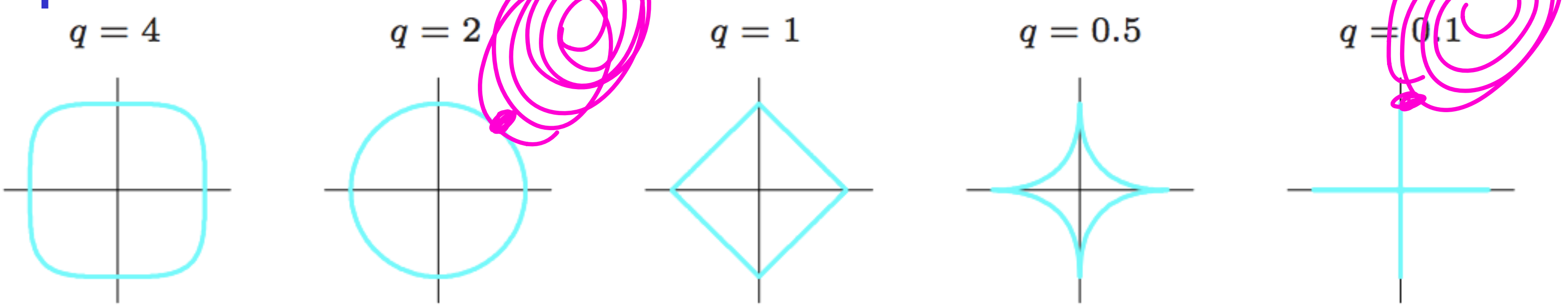
Why Lasso zeros coefficients? – a numerical explanation

解釋 2

- Ridge
  - $$\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\frac{1}{2} \sum \left(\hat{y}_i - y_i \right)^2 + \lambda \left\| \boldsymbol{\theta} \right\|^2 \right]$$
    - Changing θ_j from 2 to 1 reduces the cost by $\lambda(22 - 12) = 3\lambda$
    - Changing θ_j from 1 to 0 reduces the cost by $\lambda(12 - 02) = \lambda$
- Lasso
  - $$\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\frac{1}{2} \sum \left(\hat{y}_i - y_i \right)^2 + \lambda \left\| \boldsymbol{\theta} \right\|_1 \right]$$
    - 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



- $$\boldsymbol{\theta} := \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left[\frac{1}{2} \sum \left(\hat{y}_i - y_i \right)^2 + \lambda \|\boldsymbol{\theta}\|^q \right]$$

  $q \geq 0$

- 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
  - $J(\boldsymbol{\theta}) = \frac{1}{2}\left(\hat{\mathbf{y}} - \mathbf{y}\right)^T\left(\hat{\mathbf{y}} - \mathbf{y}\right) + \frac{\lambda}{2}\|\boldsymbol{\theta}\|^2$

    $= \frac{1}{2}(\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) + \frac{\lambda}{2}\boldsymbol{\theta}^T\boldsymbol{\theta}$
  - $\nabla_{\theta} J(\boldsymbol{\theta}) = (\mathbf{X}\boldsymbol{\theta} - \mathbf{y})^T\mathbf{X} + \lambda\boldsymbol{\theta}^T$
  - $\nabla_{\theta} J(\boldsymbol{\theta})^T = \mathbf{X}^T(\mathbf{X}\boldsymbol{\theta} - \mathbf{y}) + \lambda\boldsymbol{\theta}$

Gradient descent for ridge regression (pseudo code)

k := 0

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

while not converge:

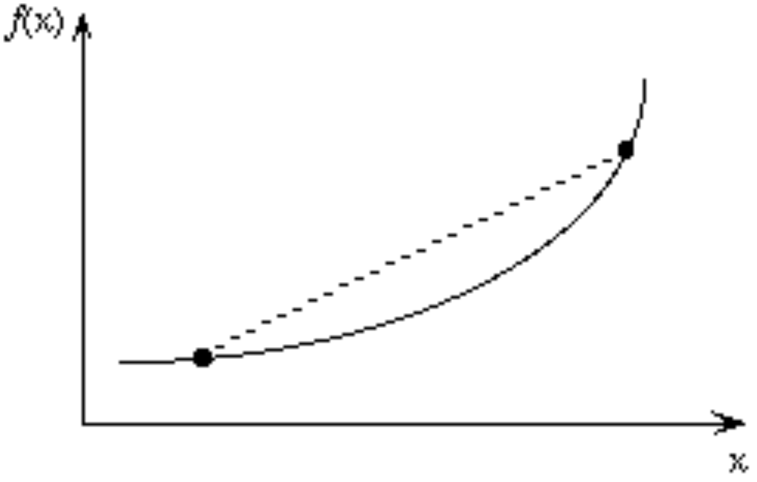
$\quad \boldsymbol{g} := \mathbf{X}^T\left(\mathbf{X}\boldsymbol{\theta}^{(k)} - \mathbf{y}\right) + \lambda\boldsymbol{\theta}^{(k)}$

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

$\quad k$:= k+1

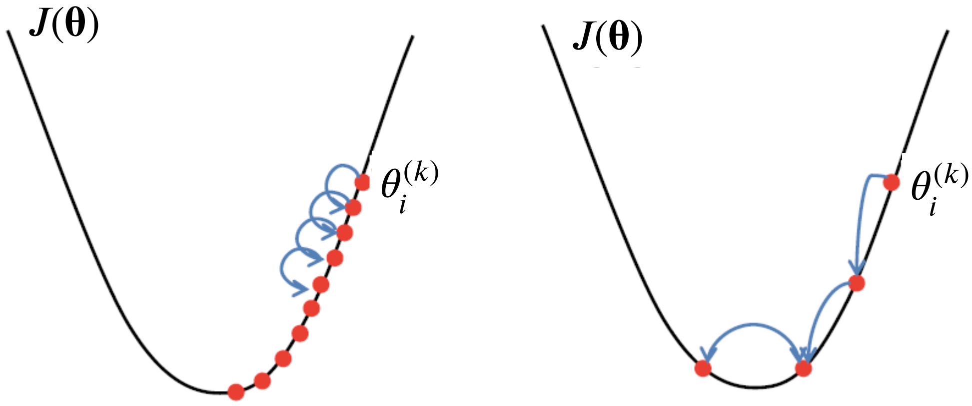
Convex function

- Convex function
  - Local minimum is global minimum
- The least-square linear regression objective and the Ridge regression objective are both convex functions
  - Gradient descent will find the unique minimum (or very close, depending on the step size α



Effect of the step size α

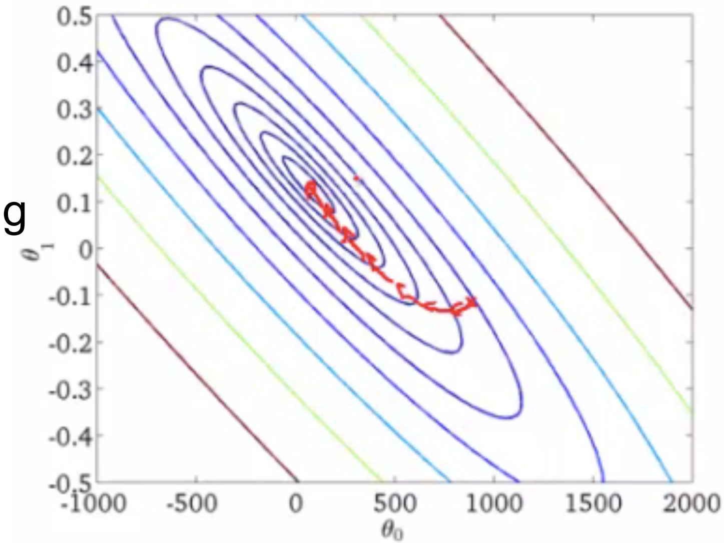
- Small α: Long computation time
- Large α: may not reach the minimum



- A common practice
  - Gradually shrink the value of the step size

Stochastic gradient descent (SGD) – motivation

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



k := 0
Initialize $\boldsymbol{\theta}^{(k)}$
while not converge:
    $g := \mathbf{X}^T\left(\mathbf{X}\boldsymbol{\theta}^{(k)} - \mathbf{y}\right) + \lambda\boldsymbol{\theta}^{(k)}$
    $\boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \alpha g$
    k := k+1

Stochastic gradient descent

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

```
k := 0
Initialize θ^(k)
while not converge:
    g := x_i (x_i^T θ^(k) − y_i) + λθ^(k)
    θ^(k+1) := θ^(k) − αg
    k := k+1
    Get next data instance i
```

$$k := 0$$
$$\text{Initialize } \boldsymbol{\theta}^{(k)}$$
$$\text{while not converge:}$$
$$\boldsymbol{g} := \mathbf{x}_i\left(\mathbf{x}_i^T\boldsymbol{\theta}^{(k)} - \mathbf{y}_i\right) + \lambda\boldsymbol{\theta}^{(k)}$$
$$\boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \alpha\boldsymbol{g}$$
$$k := k+1$$
$$\text{Get next data instance } i$$

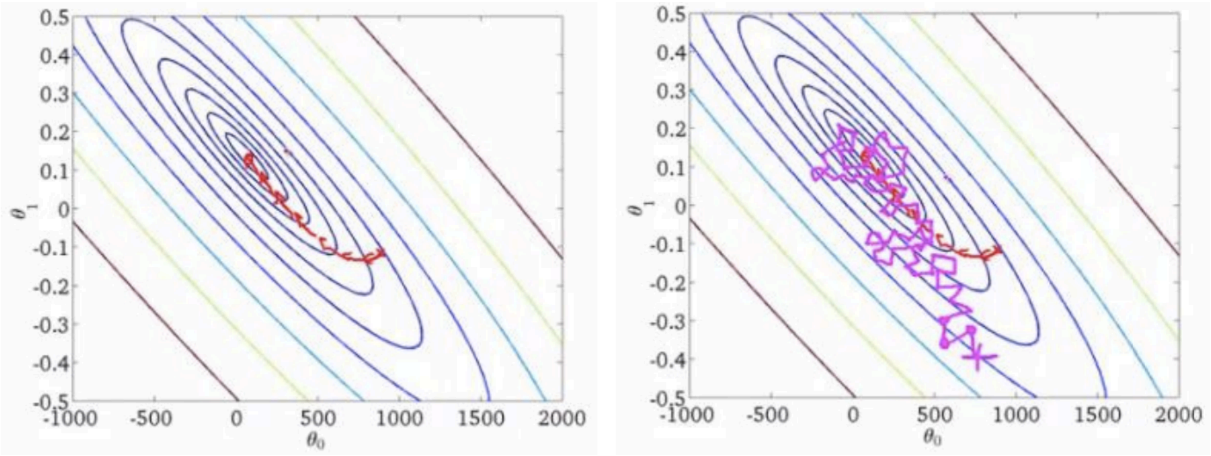
Only one training instance is examined at each step

- Why does SGD work?

$$E\left(\nabla J_i(\boldsymbol{\theta})\right) = \frac{1}{n}\sum \nabla J_i(\boldsymbol{\theta}) = \nabla J(\boldsymbol{\theta})$$

Gradient descent (GD) vs stochastic gradient descent (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
- Cons
  - Tends to bouncing around the minimum

Mini-batch gradient descent

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

$k := 0$

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

while not converge:

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

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

$\quad k := k+1$

$\quad$ Get next batch $(i, i+1, \ldots, j)$

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*)

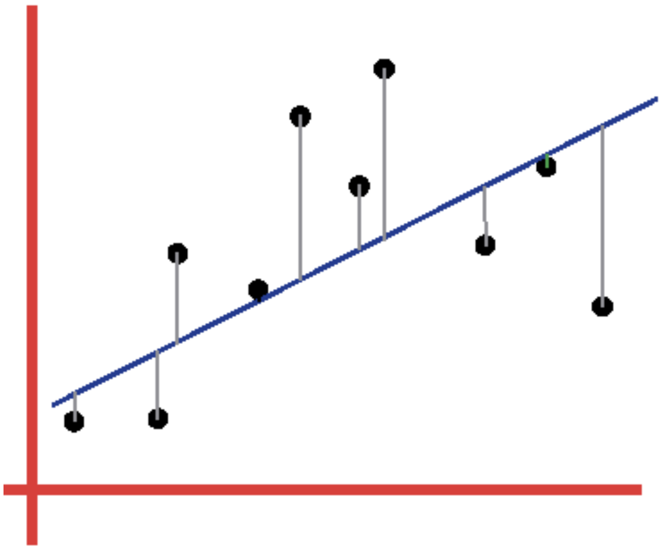
Locally weighted linear regression

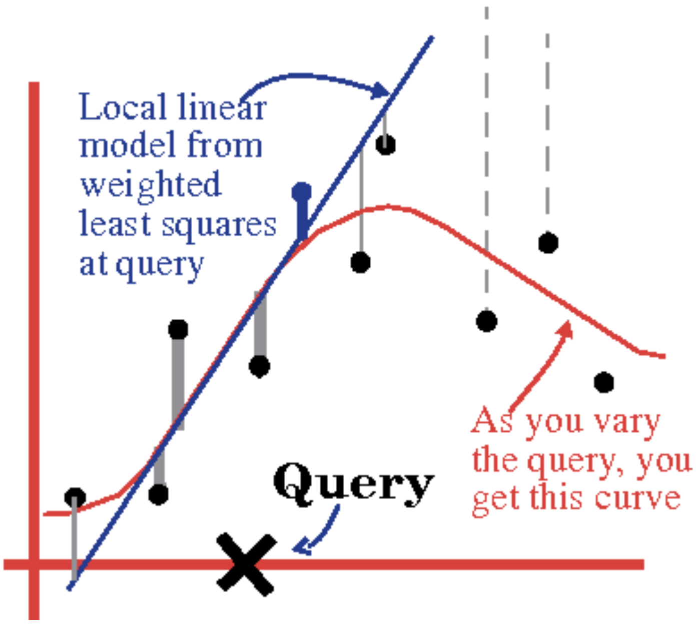
- Linear regression
  - Method
    - Assume $\hat{y} = \theta_0 + \theta_1 x$
    - Find θ_0 and θ_1 to minimize $\sum_i \left(y_i - \hat{y}_i\right)^2$
    - Given a new instance x^*, we predict the corresponding y^* as $\theta_0 + \theta_1 x^*$
  - Once training finished, we get θ_0 and θ_1, and training data can be removed
  - However, can only model linear relationship
- Locally weighted linear regression
  - Model the ”nonlinear” data by weighting errors differently

Locally weighted linear regression

- Method
  - Given a new instance x^*, we define the weight of each training error by
    - $$w_i = \exp\left(\frac{\left(x_i - x^*\right)^2}{2\sigma^2}\right)$$
      - σ is a hyperparameter; we set to 1 for simplicity
      - If $x_i \approx x^*,\ w_i \approx 1$
      - If x_i and x^* is far, $w_i \approx 0$
  - Find θ_0 and θ_1 to minimize
    - $$\sum_i w_i\left(y_i - \hat{y}_i\right)^2$$, where $\hat{y}_i = \theta_0 + \theta_1 x_i$
- Adv: can fit nonlinear curve
- Disadv: need to "train" the model for every prediction

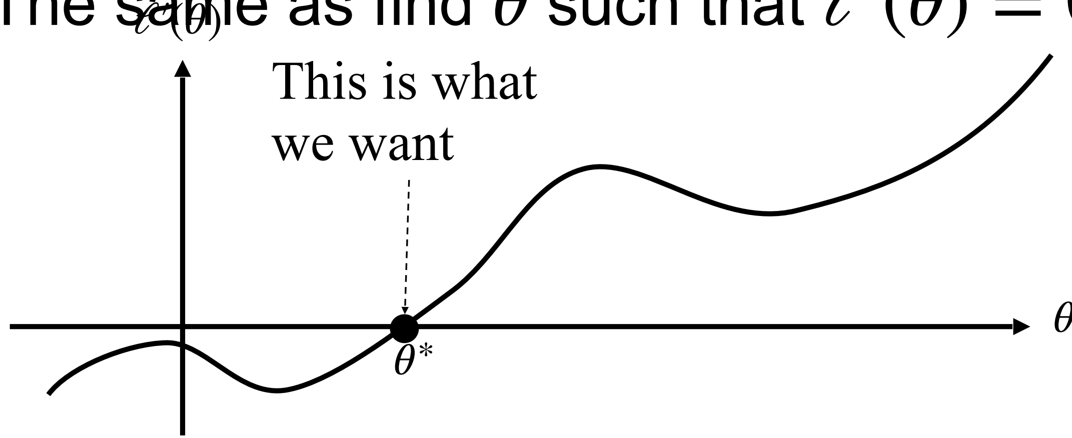
LR vs locally weighted LR





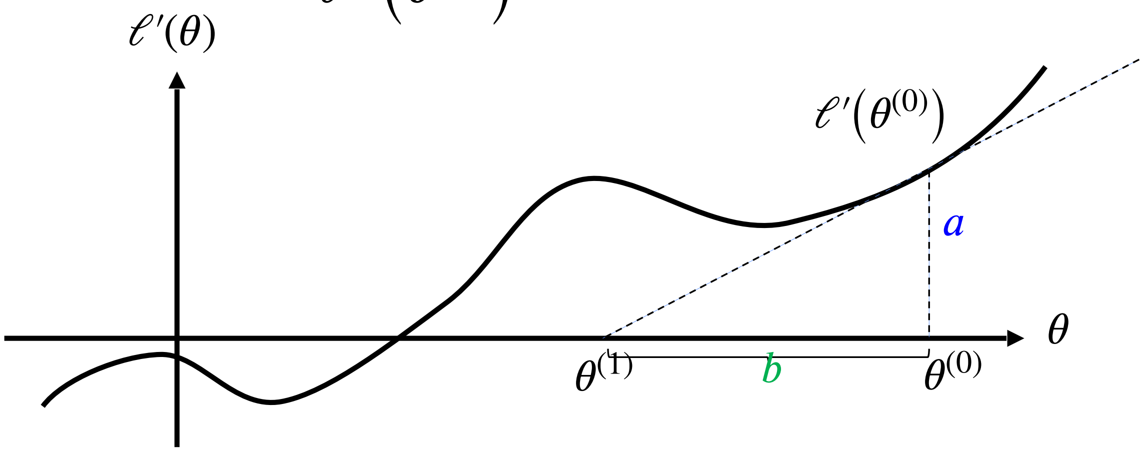
Newton's method for optimization (1/2)

- Let $\ell(\theta) = \sum_i \left(y_i - \hat{y}_i \right)^2$, we want to find θ to minimize $\ell(\theta)$
  - The same as find θ such that $\ell'(\theta) = 0$



Newton's method for optimization (2/2)

- Initial guess: $\theta^{(0)}$
- The slope of the tangent line at $(\theta^{(0)},\ \ell'(\theta^{(0)})$ is $\ell''\left(\theta^{(0)}\right)$
  - I.e., $\ell''\left(\theta^{(0)}\right) = \frac{a}{b} = \frac{\ell'\left(\theta^{(0)}\right)}{\theta^{(0)} - \theta^{(1)}}$
- So, $\theta^{(1)} = \theta^{(0)} - \frac{\ell'\left(\theta^{(0)}\right)}{\ell''\left(\theta^{(0)}\right)}$



Newton’s mthod vs SGD

- Newton’s method usually requires fewer steps
- However, the cost of each step is usually large
  - If θ is a scalar, using Newton’s method requires to compute $f''(\theta)$
  - If $\boldsymbol{\theta}$ is a vector, using Newton’s method requires to compute the Hessian matrix

Evaluation (regression)

Evaluating regression models

- Mean Square Error (MSE)
  - $$MSE(\mathbf{y},\ \hat{\mathbf{y}}) = \frac{\sum \left(y_i - \hat{y}_i\right)^2}{n}$$
  - Similar metric: Root Mean Squared Error (RMSE)
  - Criticism:
    - Can only be compared between models whose errors are measured in the same unit
    - 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:
    - Can only be compared between models whose errors are measured in the same unit

Evaluating regression models

- 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:
    - Can only be compared between models whose errors are measured in the same unit
- R^2 score (coefficient of determination)
  - $$R^2(\mathbf{y}, \hat{\mathbf{y}}) = 1 - \frac{\sum (y_i - \hat{y}_i)^2}{\sum (y_i - \bar{y})^2}$$
  - The scale of the model is more intuitive
    - Best result: 1
    - Can be arbitrarily worse (very negative)

A closer look of R^2 score (coefficient of determination)

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

R (correlation coefficient) vs R^2 (coefficient of determination)

- $\left[R\left(\boldsymbol{y},\ \hat{\boldsymbol{y}}\right)\right]^2$ equals $R^2\left(\boldsymbol{y},\ \hat{\boldsymbol{y}}\right)$ when
  - Applying the linear regression model, i.e., $\hat{\boldsymbol{y}} = \boldsymbol{X\theta}$, where $\boldsymbol{\theta} = \left(\boldsymbol{X}^T\boldsymbol{X}\right)^{-1}\boldsymbol{X}^T\boldsymbol{y}$, and
  - Evaluating $R\left(\boldsymbol{y},\ \hat{\boldsymbol{y}}\right)$ and $R^2\left(\boldsymbol{y},\ \hat{\boldsymbol{y}}\right)$ on the training data
- In this case, $0 \le R^2\left(\boldsymbol{y},\ \hat{\boldsymbol{y}}\right) \le 1$

Conclusion (1/3)

- Linear regression
  - $\hat{y}_i = \theta_0 + \theta_1 x_{i,1} + \theta_2 x_{i,2} + \ldots + \theta_d x_{i,d}$
  - Find good $\theta_0, \theta_1, \ldots, \theta_d$ such that $\hat{y}_i \approx y_i \; \forall i$
    - Commonly used distance between $\hat{y}_i$ and y_i: RSS
- 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

Conclusion (2/3)

- Convex function
  - Local minimum equals global minimum
  - Iteratively adjust $\boldsymbol{\theta}$ to reduce $J(\boldsymbol{\theta})$
    - $\boldsymbol{\theta}^{(k+1)} := \boldsymbol{\theta}^{(k)} - \alpha \boldsymbol{g}$
- For ridge regression
  - Gradient descent
    - $\boldsymbol{g} := \mathbf{X}^T\left(\mathbf{X}\boldsymbol{\theta}^{(k)} - \mathbf{y}\right) + \lambda\boldsymbol{\theta}^{(k)}$
  - Stochastic gradient descent
    - $\boldsymbol{g} := \mathbf{x}_i\left(\mathbf{x}_i^T\boldsymbol{\theta}^{(k)} - \mathbf{y}_i\right) + \lambda\boldsymbol{\theta}^{(k)}$
  - Mini-batch gradient descent
    - $\boldsymbol{g} := \mathbf{x}_{i:j}^T\left(\mathbf{x}_{i:j}\boldsymbol{\theta}^{(k)} - \mathbf{y}_{i:j}\right) + \lambda\boldsymbol{\theta}^{(k)}$

Conclusion (3/3)

- Common metrics to evaluate regression models
  - Mean square error
  - R^2 score

Quiz

- If the training data contain millions of features, should we use gradient descent or closed form solution for training?
- Can gradient descent get stuck in a local minimum (which is not a global minimum) when training a linear regression model?
- Why would one use Ridge regression instead of plain linear regression?
- If the R^2 score on test data is negative, what does this mean?
- If the R^2 score on training data is negative, what does this mean?