Announcements

- **HW 0** due today 8 pm
- **HW 1** (on linear regression) will be released this afternoon.

- **Office hour** starting tomorrow.
    - Time and location (in-person & remote) will be posted on course website & canvas.

Lecture 4: Linear Regression (Part 3)

CIS 4190/5190

Spring 2025

Last Lecture

- Train/Test Split Protocol for Measuring Underfitting / Overfitting
- Bias and variance as functions of a model class
  - Tuning them by selecting hypothesis spaces / feature maps
  - Tuning them by modifying the loss function
    - $L_{\text{new}}(\beta; Z) = L(\beta; Z) + \lambda \cdot R(\beta)$
- Train/Val/Test Split Protocol for Hyperparameter tuning.
  - K-fold cross validation for small datasets.

Last Lecture

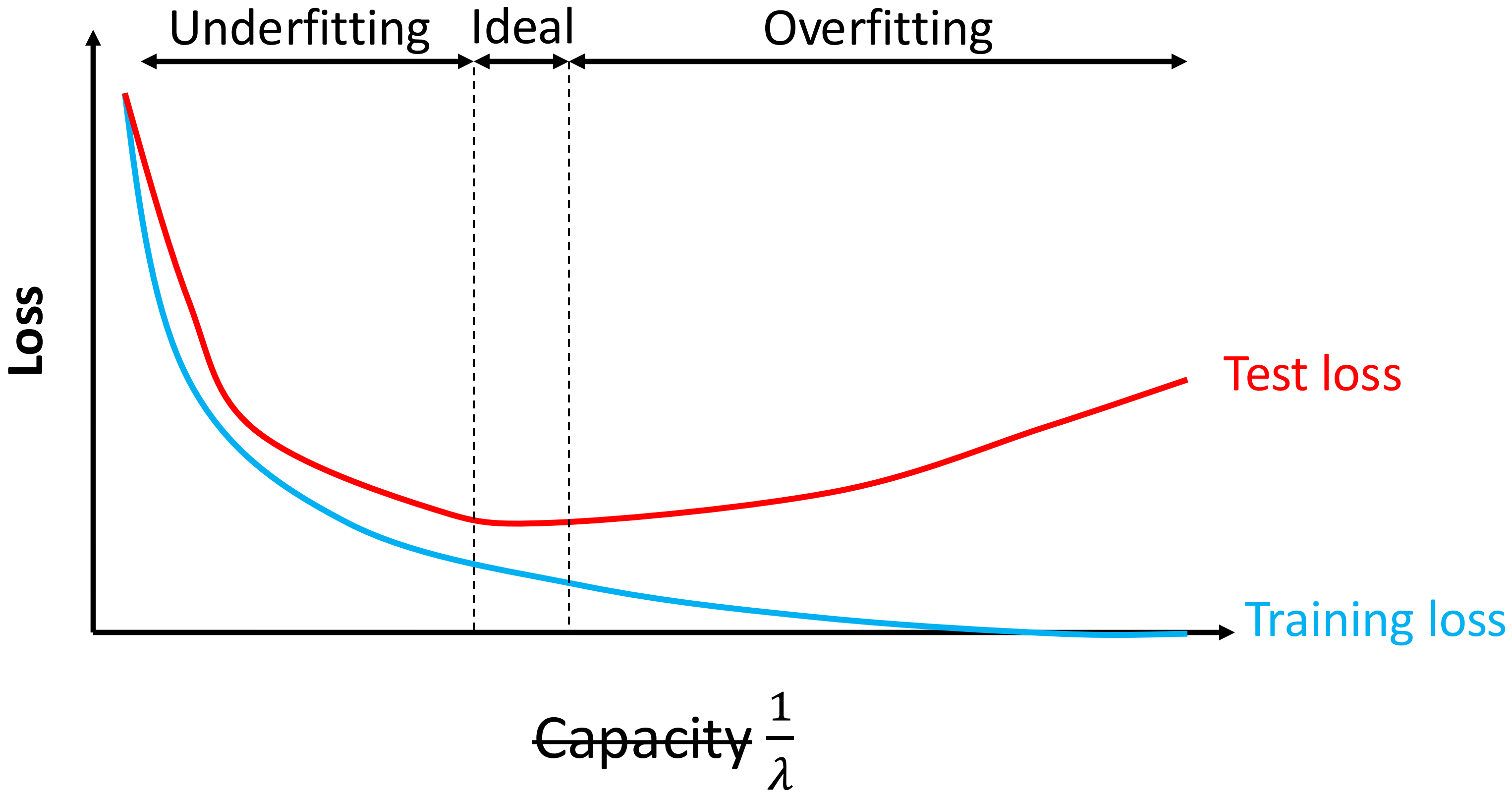
- **Original MSE loss** + **regularization**:

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \cdot \|\beta\|_2^2$$

- With intercept term ($\phi(x) = [1 \quad x_1 \quad \cdots \quad x_d]^\top$), no penalty on β_1:

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \sum_{j=2}^{d}\beta_j^2$$

Last Lecture



Today

- **Minimizing the MSE Loss**
    - Closed-form solution
    - Stochastic gradient descent

Minimizing the MSE Loss

- Recall that linear regression minimizes the loss

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2$$

- **Closed-form solution:** Compute using matrix operations
- **Optimization-based solution:** Search over candidate β

Vectorizing Linear Regression

Vectorizing Linear Regression

$$\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix}$$

Vectorizing Linear Regression

$$\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix} = \begin{bmatrix} \beta^\top x_1 \\ \vdots \\ \beta^\top x_n \end{bmatrix}$$

Vectorizing Linear Regression

$$\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix} = \begin{bmatrix} \beta^\top x_1 \\ \vdots \\ \beta^\top x_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{d} \beta_j x_{1,j} \\ \vdots \\ \sum_{j=1}^{d} \beta_j x_{n,j} \end{bmatrix}$$

Vectorizing Linear Regression

$$\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix} = \begin{bmatrix} \beta^\top x_1 \\ \vdots \\ \beta^\top x_n \end{bmatrix} = \begin{bmatrix} \boxed{\sum_{j=1}^{d} \beta_j x_{1,j}} \\ \vdots \\ \sum_{j=1}^{d} \beta_j x_{n,j} \end{bmatrix} = \begin{bmatrix} \multicolumn{3}{c}{\boxed{x_{1,1} \quad \cdots \quad x_{1,d}}} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix} \boxed{\begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}}$$

Vectorizing Linear Regression

$$\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix} = \begin{bmatrix} \beta^\top x_1 \\ \vdots \\ \beta^\top x_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{d} \beta_j x_{1,j} \\ \vdots \\ \boxed{\sum_{j=1}^{d} \beta_j x_{n,j}} \end{bmatrix} = \left[\begin{array}{ccc} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ \multicolumn{3}{c}{\boxed{x_{n,1} \quad \cdots \quad x_{n,d}}} \end{array}\right] \boxed{\begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}}$$

Vectorizing Linear Regression

$$\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix} = \begin{bmatrix} \beta^\top x_1 \\ \vdots \\ \beta^\top x_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{d} \beta_j x_{1,j} \\ \vdots \\ \sum_{j=1}^{d} \beta_j x_{n,j} \end{bmatrix} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix} = X\beta$$

Vectorizing Linear Regression

$$
\begin{array}{c}
\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix} = \begin{bmatrix} \beta^\top x_1 \\ \vdots \\ \beta^\top x_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{d} \beta_j x_{1,j} \\ \vdots \\ \sum_{j=1}^{d} \beta_j x_{n,j} \end{bmatrix} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix} = X\beta \\
\wr\wr \\
\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}
\end{array}
$$

Vectorizing Linear Regression

$$\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix} = \begin{bmatrix} \beta^\top x_1 \\ \vdots \\ \beta^\top x_n \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{d} \beta_j x_{1,j} \\ \vdots \\ \sum_{j=1}^{d} \beta_j x_{n,j} \end{bmatrix} = \begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix} \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix} = X\beta$$

$$\wr\wr$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = Y$$

Summary: $Y \approx X\beta$

Vectorizing Linear Regression

$$Y \approx X\beta$$

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \qquad X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,d} \\ \vdots & \ddots & \vdots \\ x_{n,1} & \cdots & x_{n,d} \end{bmatrix} \qquad \beta = \begin{bmatrix} \beta_1 \\ \vdots \\ \beta_d \end{bmatrix}$$

Vectorizing Mean Squared Error

Vectorizing Mean Squared Error

$L(\beta; Z)$

Vectorizing Mean Squared Error

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2$$

Vectorizing Mean Squared Error

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 = \frac{1}{n}\|\overset{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}{\overset{\downarrow}{Y}} - \overset{\begin{bmatrix} f_\beta(x_1) \\ \vdots \\ f_\beta(x_n) \end{bmatrix}}{\overset{\downarrow}{X\beta}}\|_2^2$$

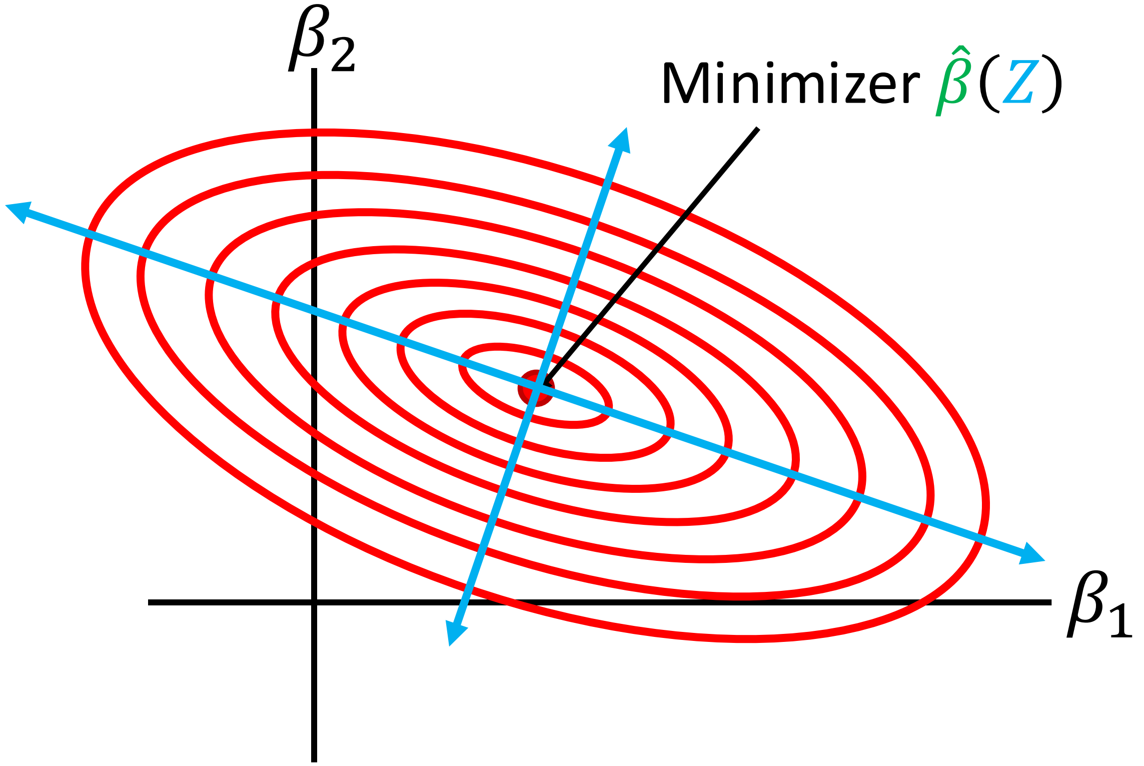
Intuition on Vectorized Linear Regression

- Rewriting the vectorized loss:

$$\begin{aligned} n \cdot L(\beta; Z) = \|Y - X\beta\|_2^2 &= \|Y\|_2^2 - 2Y^\top X\beta + \|X\beta\|_2^2 \\ &= \|Y\|_2^2 - 2Y^\top X\beta + \beta^\top (X^\top X)\beta \end{aligned}$$

- Quadratic function of β with leading “coefficient” $X^\top X$
    - In one dimension, “width” of parabola $ax^2 + bx + c$ is a^{-1}
    - In multiple dimensions, “width” along direction v_i is λ_i^{-1}, where v_i is an eigenvector of $X^\top X$ with eigenvalue λ_i

Intuition on Vectorized Linear Regression



Directions/magnitudes are given by eigenvectors/eigenvalues of $X^\top X$

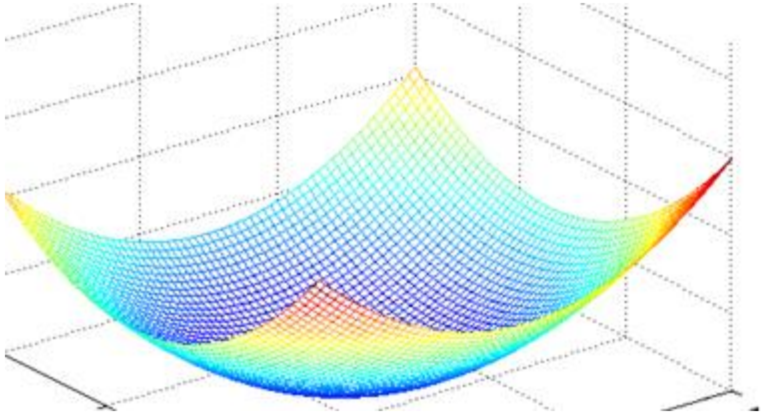
Strategy 1: Closed-Form Solution

- Recall that linear regression minimizes the loss

$$L(\beta; Z) = \frac{1}{n} \|Y - X\beta\|_2^2$$

- Minimum solution has gradient equal to zero:

$$\nabla_\beta L(\hat{\beta}; Z) = 0$$



Strategy 1: Closed-Form Solution

- The gradient is

$$\nabla_{\beta} L(\beta; Z)$$

Strategy 1: Closed-Form Solution

- The gradient is

$$\nabla_{\beta} L(\beta; Z) = \nabla_{\beta} \frac{1}{n} \|Y - X\beta\|_2^2$$

Strategy 1: Closed-Form Solution

- The gradient is

$$
\begin{aligned}
\nabla_{\beta} L(\beta; Z) = \nabla_{\beta} \frac{1}{n} \|Y - X\beta\|_2^2 &= \nabla_{\beta} \frac{1}{n} (Y - X\beta)^{\top} (Y - X\beta) \\
&= \frac{2}{n} \left[\nabla_{\beta} (Y - X\beta)^{\top} \right] (Y - X\beta) \\
&= -\frac{2}{n} X^{\top} (Y - X\beta) \\
&= -\frac{2}{n} X^{\top} Y + \frac{2}{n} X^{\top} X\beta
\end{aligned}
$$

Strategy 1: Closed-Form Solution

- The gradient is

$$\nabla_{\beta} L(\beta; Z) = \nabla_{\beta} \frac{1}{n} \|Y - X\beta\|_2^2 = -\frac{2}{n} X^{\top} Y + \frac{2}{n} X^{\top} X \beta$$

- Setting $\nabla_{\beta} L\left(\hat{\beta}; Z\right) = 0$, we have $X^{\top} X \hat{\beta} = X^{\top} Y$

Strategy 1: Closed-Form Solution

- Setting $\nabla_\beta L(\hat{\beta}; Z) = 0$, we have $X^\top X\hat{\beta} = X^\top Y$

- Assuming $X^\top X$ is invertible, we have

$$\hat{\beta}(Z) = (X^\top X)^{-1} X^\top Y$$

Note on Invertibility

- Closed-form solution only **unique** if $X^\top X$ is invertible
  - Otherwise, **multiple solutions exist** to $X^\top X \hat{\beta} = X^\top Y$
  - **Intuition:** Underconstrained system of linear equations

When Can this Happen?

- **Case 1**
  - Fewer data examples than feature dimension (i.e., $n < d$)
  - **Solution:** Remove features so $d \leq n$
  - **Solution:** Collect more data until $d \leq n$

- **Case 2:** Some feature is a linear combination of the others
  - **Special case (duplicated feature):** For some j and j', $x_{i,j} = x_{i,j'}$ for all i
  - **Solution:** Remove linearly dependent features
  - **Solution:** Use L_2 regularization

Shortcomings of Closed-Form Solution

- Computing $\hat{\beta}(Z) = (X^\top X)^{-1} X^\top Y$ can be challenging

- **Computing $(\boldsymbol{X}^\top \boldsymbol{X})^{-1}$ is $\boldsymbol{O}(\boldsymbol{d}^3)$**
  - $d = 10^4$ features → $O(10^{12})$
  - Even storing $X^\top X$ requires a lot of memory

- **Numerical accuracy issues due to "ill-conditioning"**
  - $X^\top X$ is "barely" invertible
  - Then, $(X^\top X)^{-1}$ has large variance along some dimension
  - Regularization helps (more on this later)

Today

- **Minimizing the MSE Loss**
    - Closed-form solution
    - Stochastic gradient descent

Iterative Optimization Algorithms

- Recall that linear regression minimizes the loss

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2$$

- Iteratively optimize β
  - Initialize $\beta_1 \leftarrow \mathrm{Init}(\ldots)$
  - For some number of iterations T, update $\beta_t \leftarrow \mathrm{Step}(\ldots)$
  - Return β_T

Iterative Optimization Algorithms

- **Global search**: Try random values of β and choose the best
  - I.e., β_t independent of β_{t-1}
  - Very unstructured, can take a long time (especially in high dimension d)!

- **Local search**: Start from some initial β and make local changes
  - I.e., β_t is computed based on β_{t-1}
  - What is a “local change”, and how do we find good one?

Strategy 2: Gradient Descent

- **Gradient descent:** Update β based on **gradient** $\nabla_\beta L(\beta; Z)$ of $L(\beta; Z)$:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- **Intuition:** The gradient is the direction along which $L(\beta; Z)$ changes most quickly as a function of β

- $\alpha \in \mathbb{R}$ is a hyperparameter called the **learning rate**
  - More on this later

Strategy 2: Gradient Descent

- Choose initial value for β
- Until we reach a minimum:
  - Choose a new value for β to reduce $L(\beta; Z)$

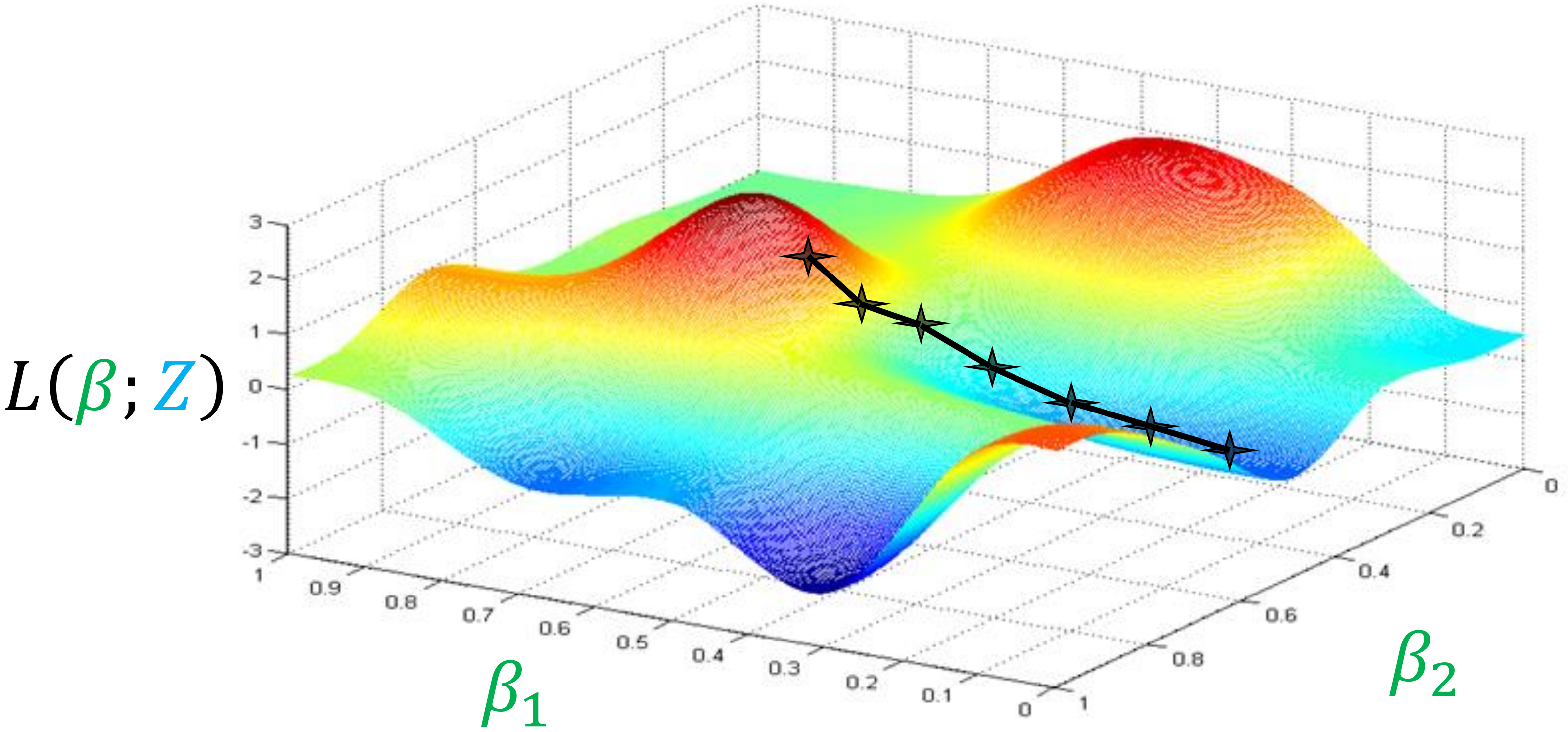


Figure by Andrew Ng

Strategy 2: Gradient Descent

- Choose initial value for β
- Until we reach a minimum:
    - Choose a new value for β to reduce $L(\beta; Z)$

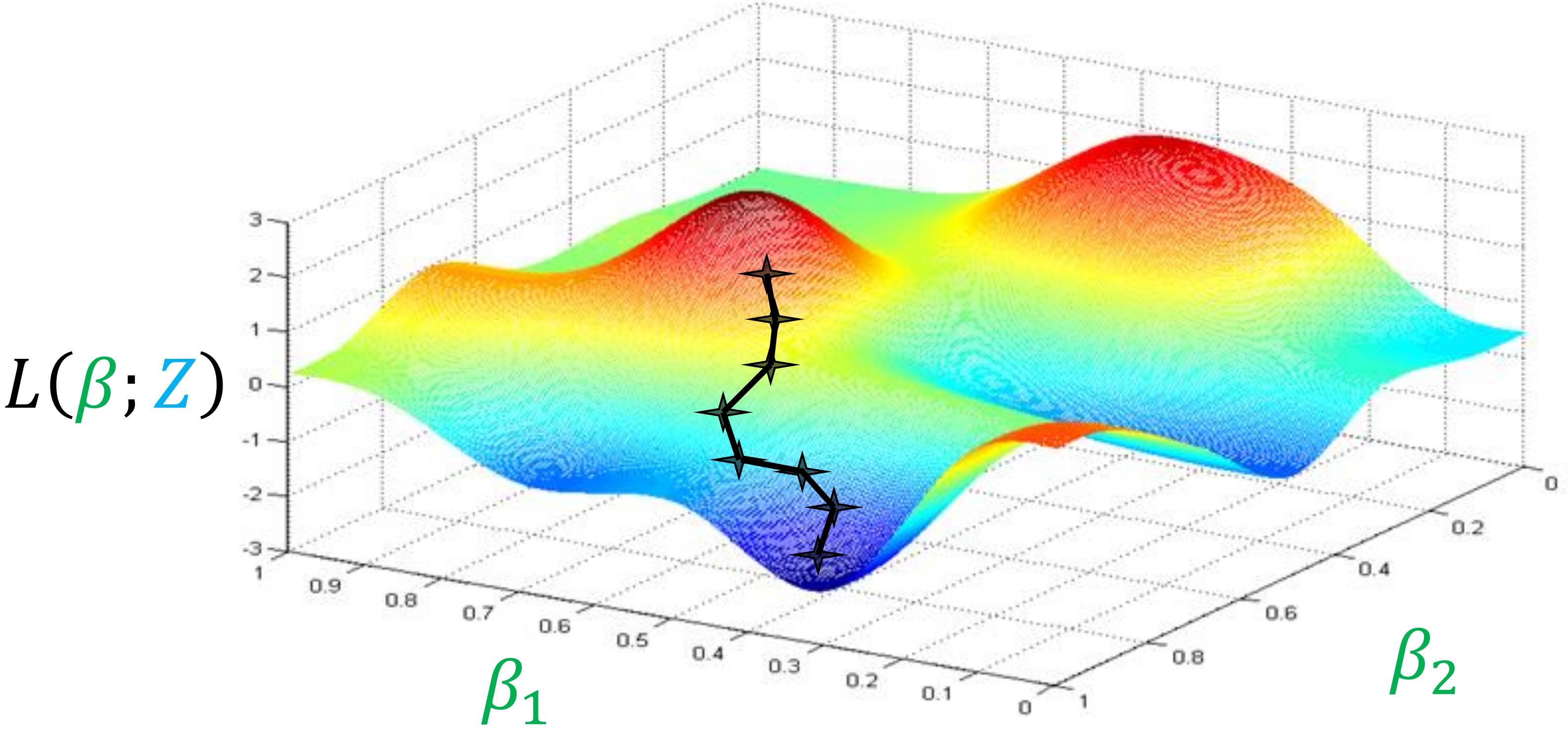
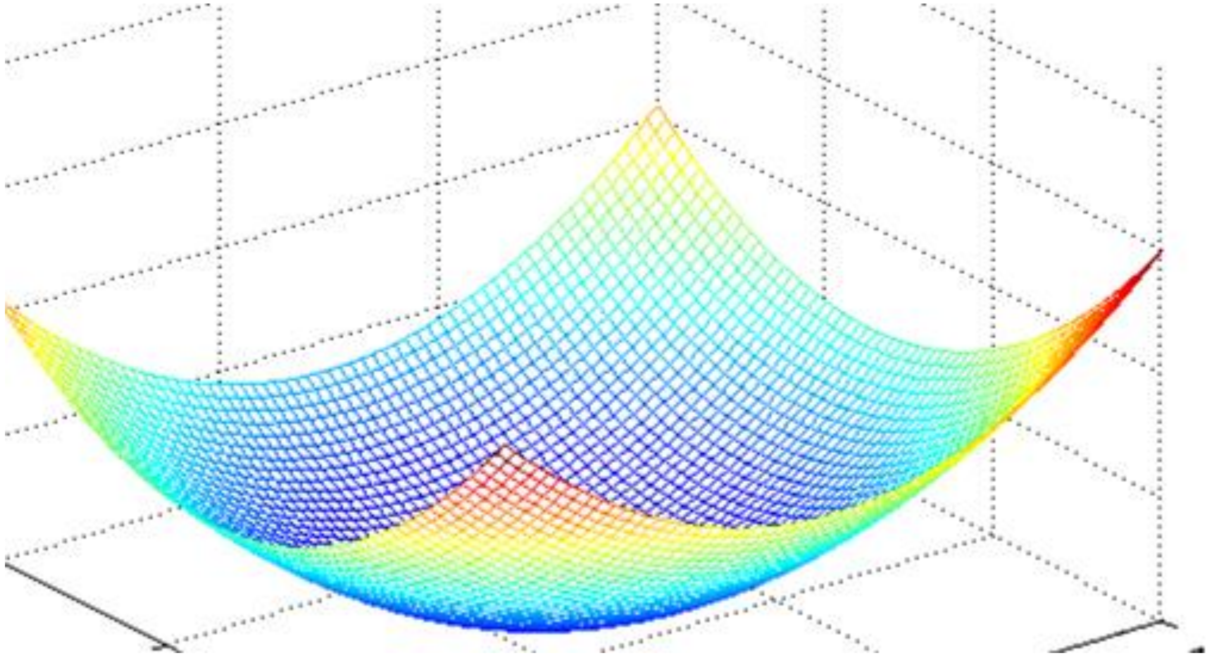


Figure by Andrew Ng

Strategy 2: Gradient Descent

- Choose initial value for β
- Until we reach a minimum:
  - Choose a new value for β to reduce $L(\beta; Z)$



Linear regression loss is convex, so no local minima

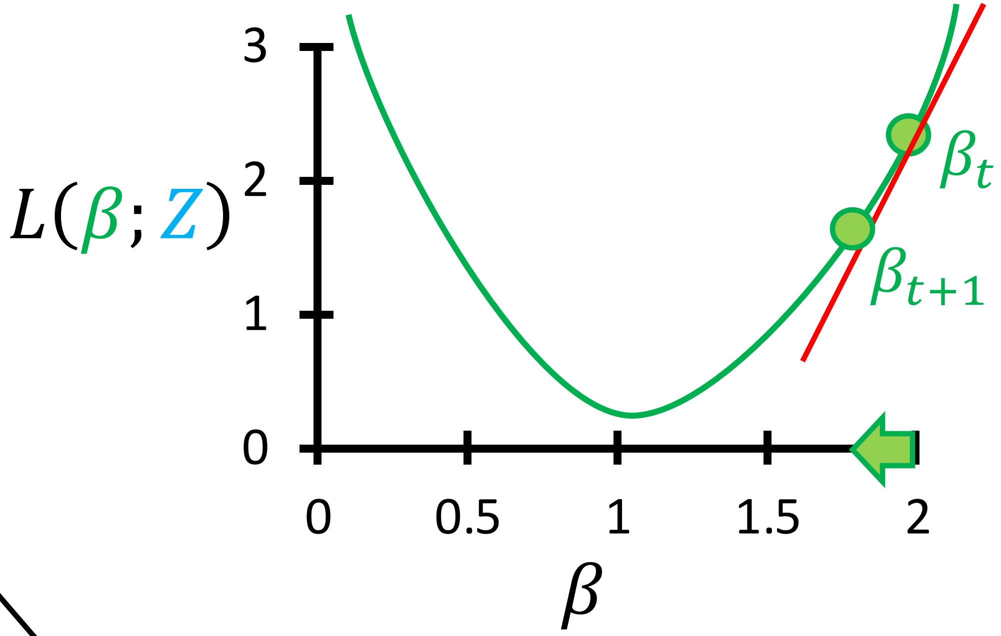
Figure by Andrew Ng

Strategy 2: Gradient Descent

- Initialize $\beta_1 = \vec{0}$
- Repeat until convergence:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1



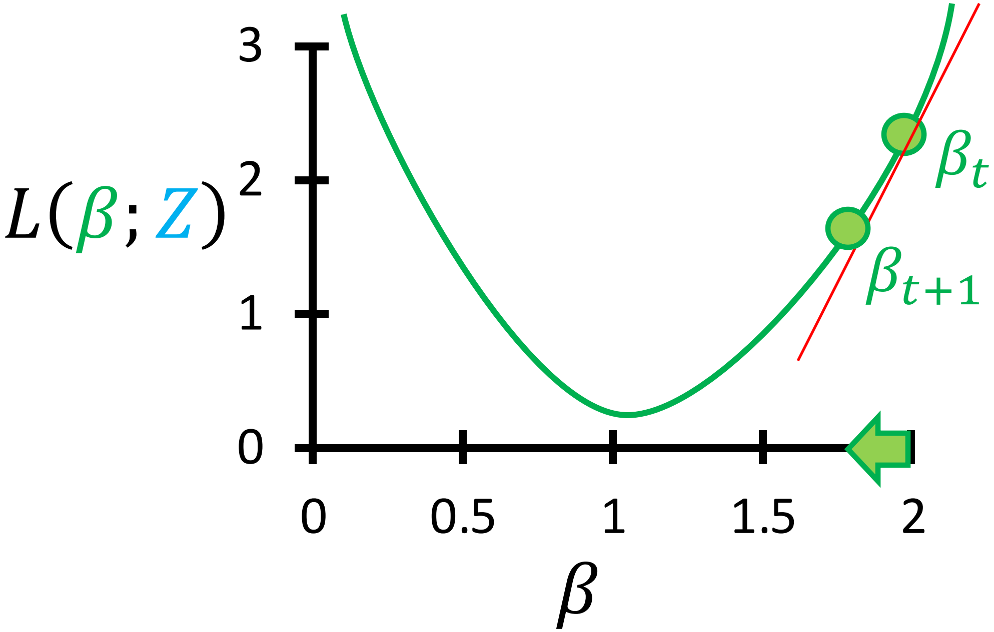
For in-place updates $\beta \leftarrow \beta - \alpha \cdot \nabla_\beta L(\beta; Z)$, compute all components of $\nabla_\beta L(\beta; Z)$ before modifying β

Strategy 2: Gradient Descent

- Initialize $\beta_1 = \vec{0}$
- Repeat until convergence:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1



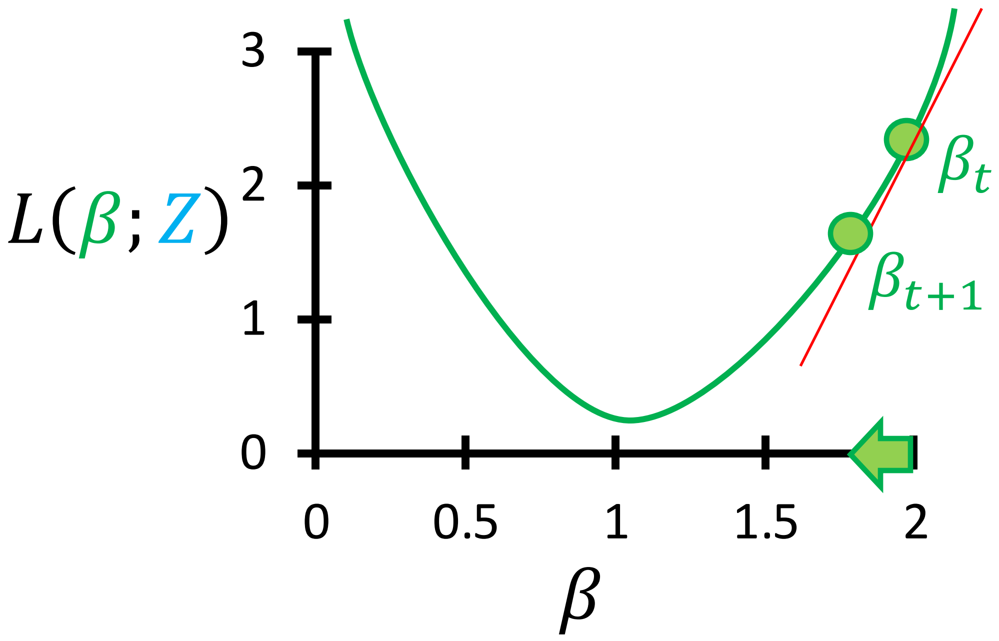
Strategy 2: Gradient Descent

Hyperparameter defining convergence

- Initialize $\beta_1 = \vec{0}$
- Repeat until $\|\beta_t - \beta_{t+1}\|_2 \leq \epsilon$:

$$\beta_{t+1} \leftarrow \beta_t - \alpha \cdot \nabla_\beta L(\beta_t; Z)$$

- For linear regression, know the gradient from strategy 1



Aside: Gradient As Sum of Sample-Wise Gradients

$$\boxed{-\frac{2}{n}X^\top Y + \frac{2}{n}X^\top X\beta}$$

(Equivalent to our earlier matrix expression of gradient)

- By linearity of the gradient, we have

$$\nabla_\beta L(\beta; Z) = \sum_{i=1}^{n} \nabla_\beta (y_i - \beta^\top x_i)^2 = \sum_{i=1}^{n} 2(y_i - \beta^\top x_i)x_i$$

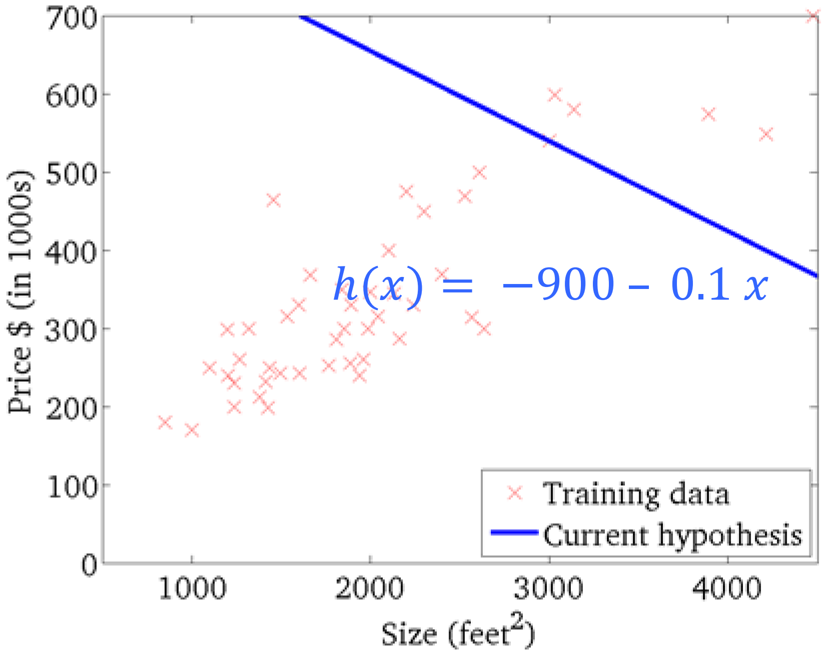
- The gradient term induced by a single training data sample is:

$$\nabla_\beta (y_i - \beta^\top x_i)^2 = 2(y_i - \beta^\top x_i)x_i$$

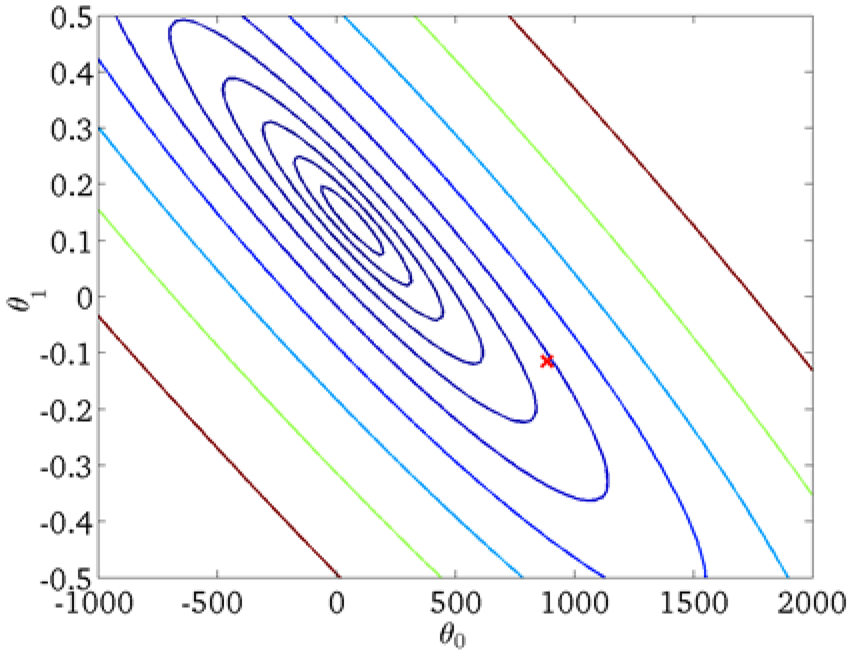
- I.e., the current error $y_i - \beta^\top x_i$ times the feature vector x_i

"Large error samples induce large changes to β, proportional to their feature values."

Strategy 2: Gradient Descent



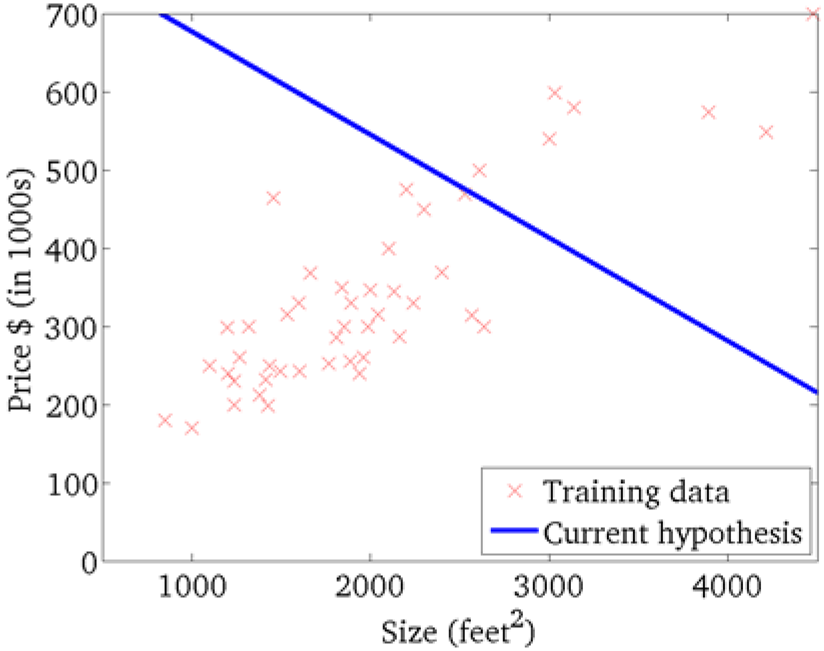
$f_\beta(x)$



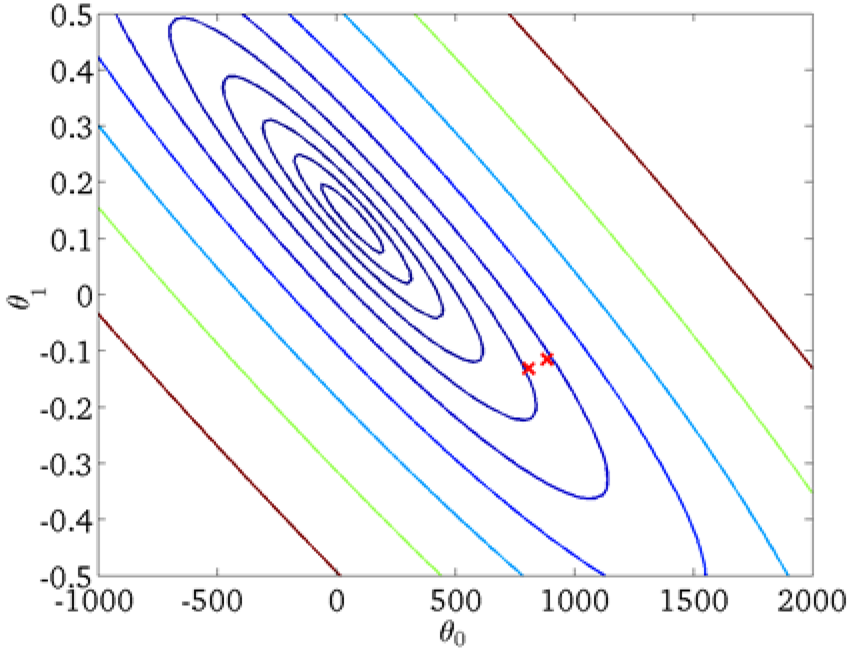
$L(\beta; Z)$

Slide by Andrew Ng

Strategy 2: Gradient Descent



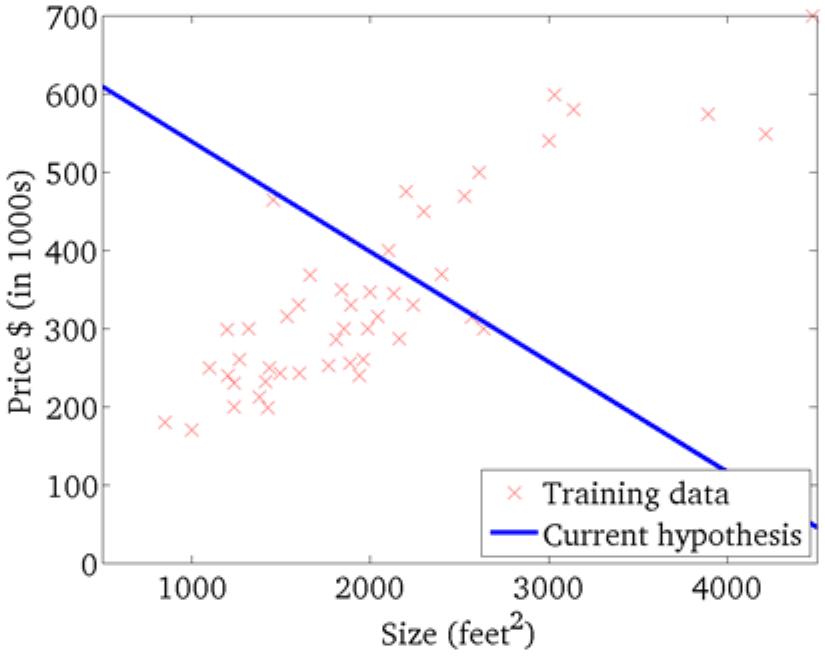
$f_\beta(x)$



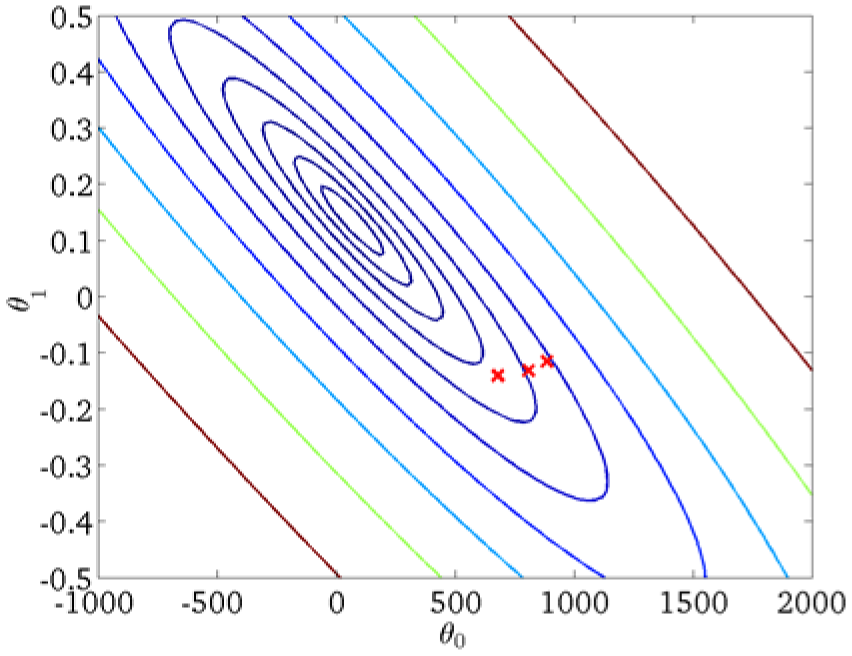
$L(\beta; Z)$

Slide by Andrew Ng

Strategy 2: Gradient Descent



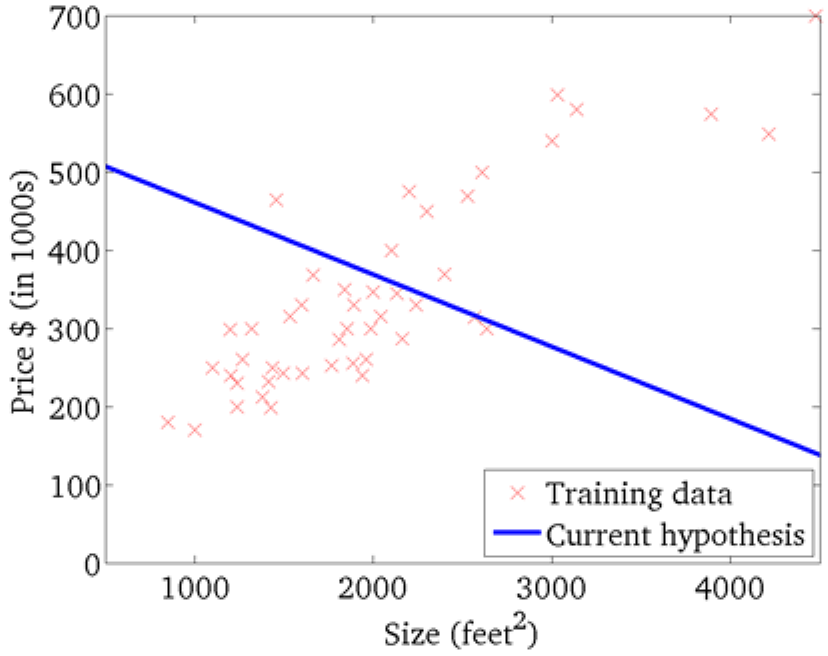
$f_\beta(x)$



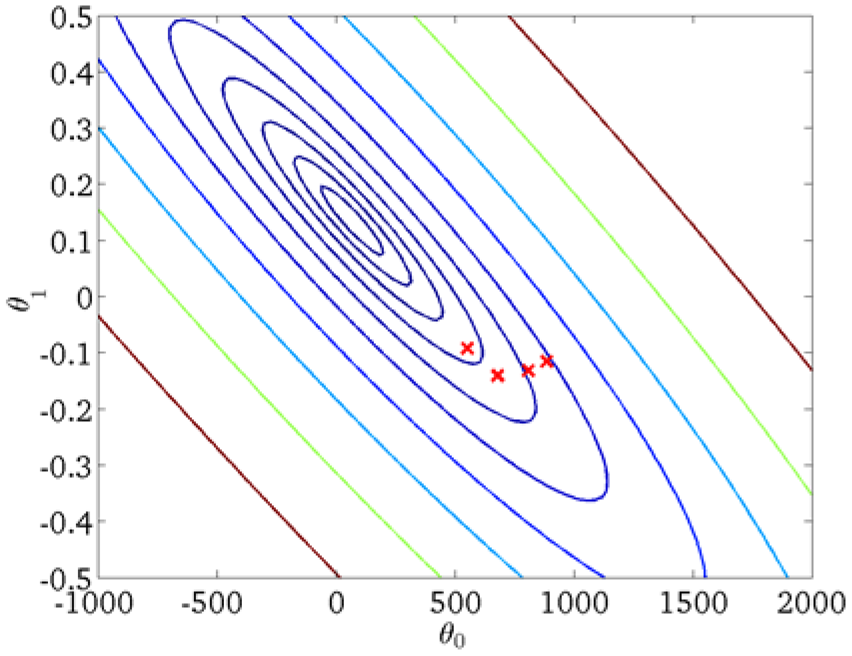
$L(\beta; Z)$

Slide by Andrew Ng

Strategy 2: Gradient Descent



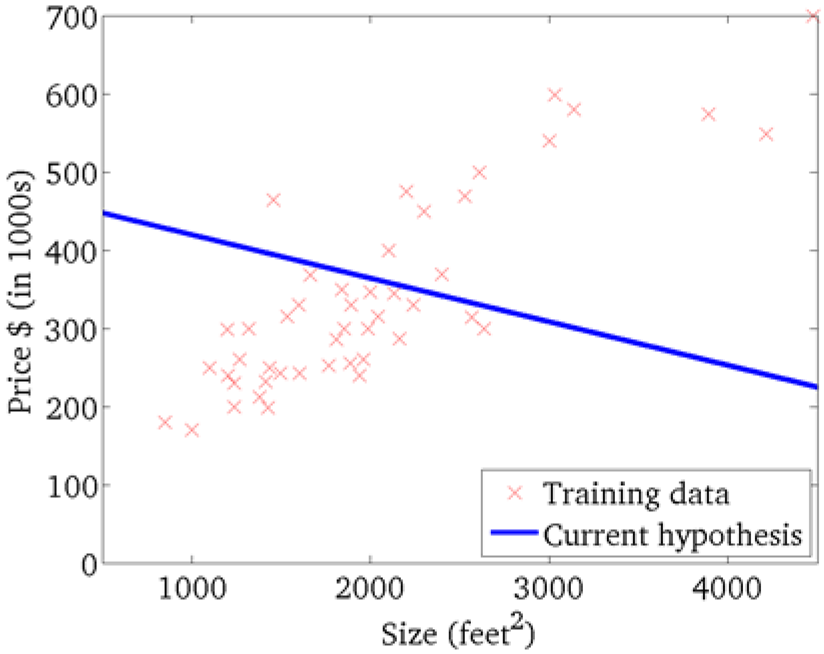
$f_\beta(x)$



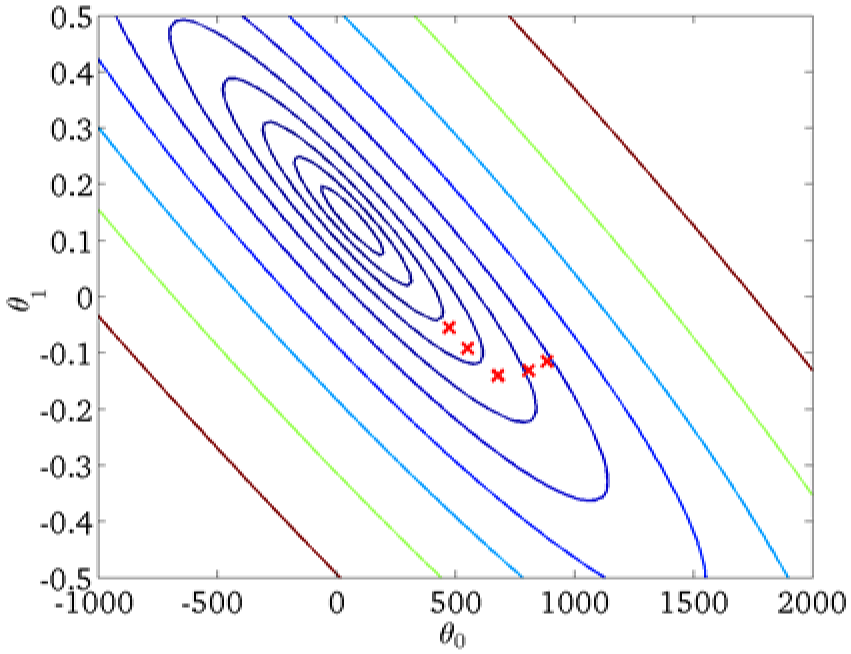
$L(\beta; Z)$

Slide by Andrew Ng

Strategy 2: Gradient Descent



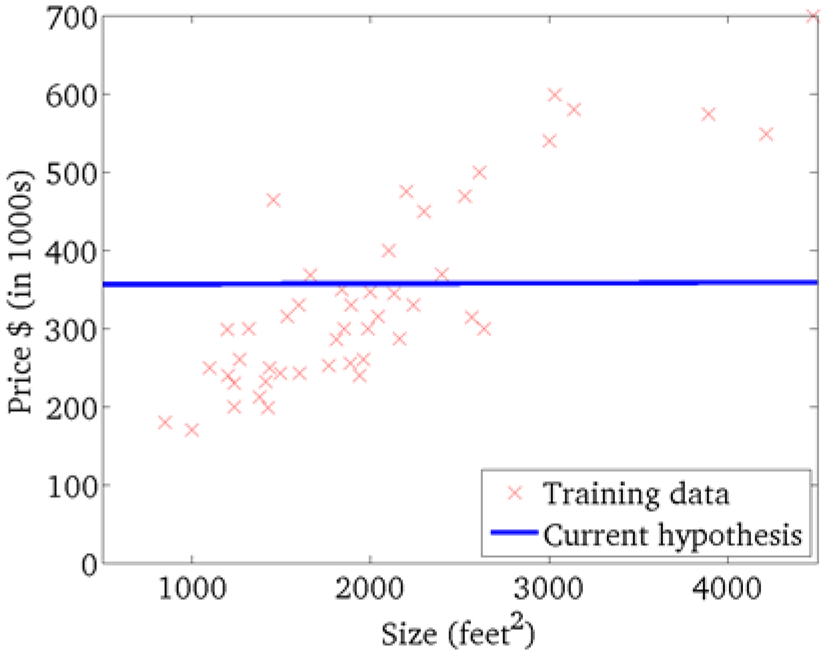
$f_\beta(x)$



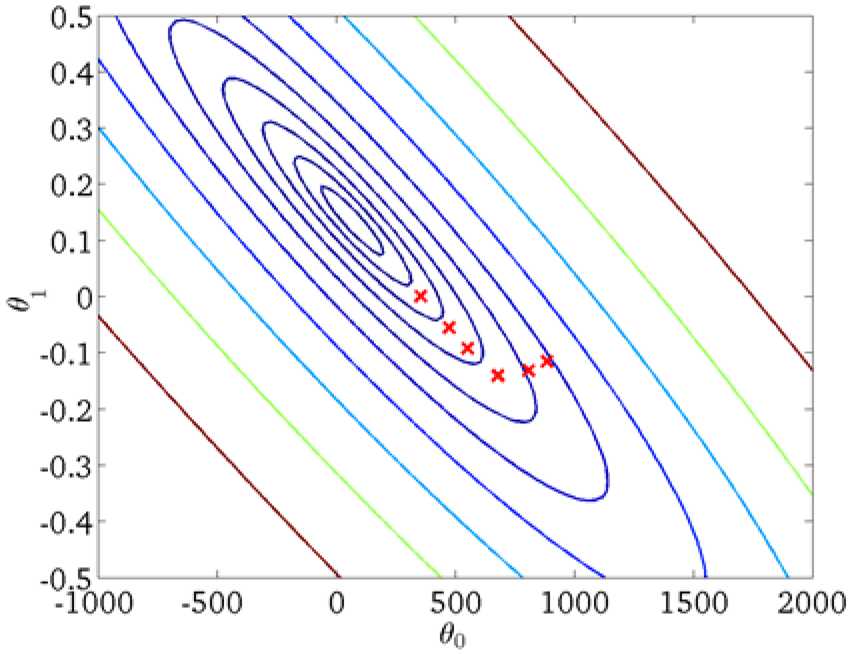
$L(\beta; Z)$

Slide by Andrew Ng

Strategy 2: Gradient Descent



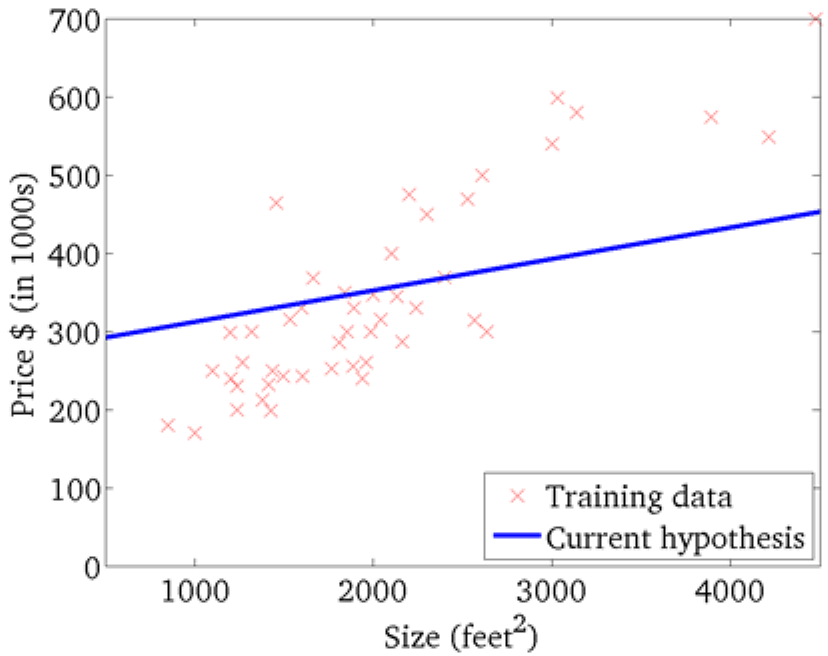
$f_\beta(x)$



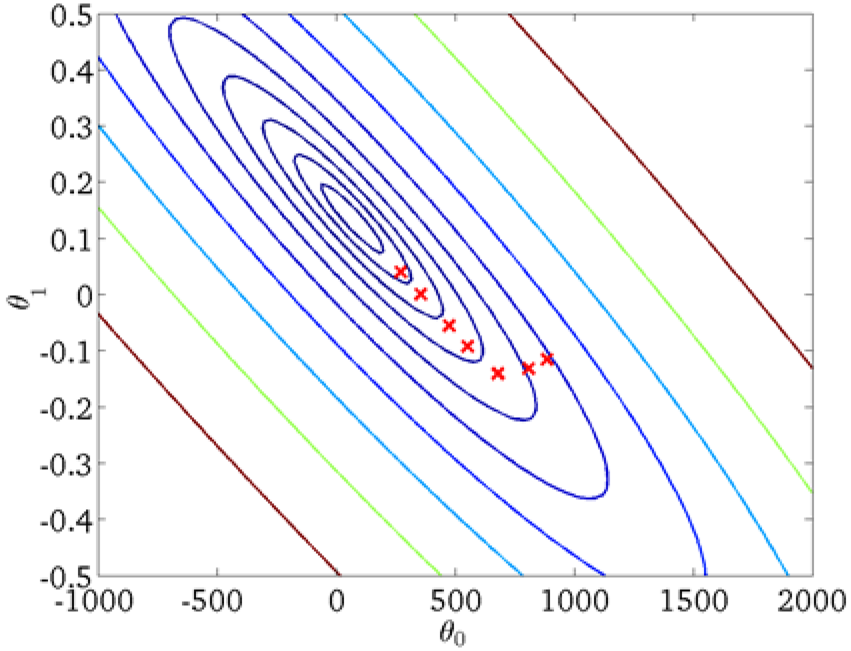
$L(\beta; Z)$

Slide by Andrew Ng

Strategy 2: Gradient Descent



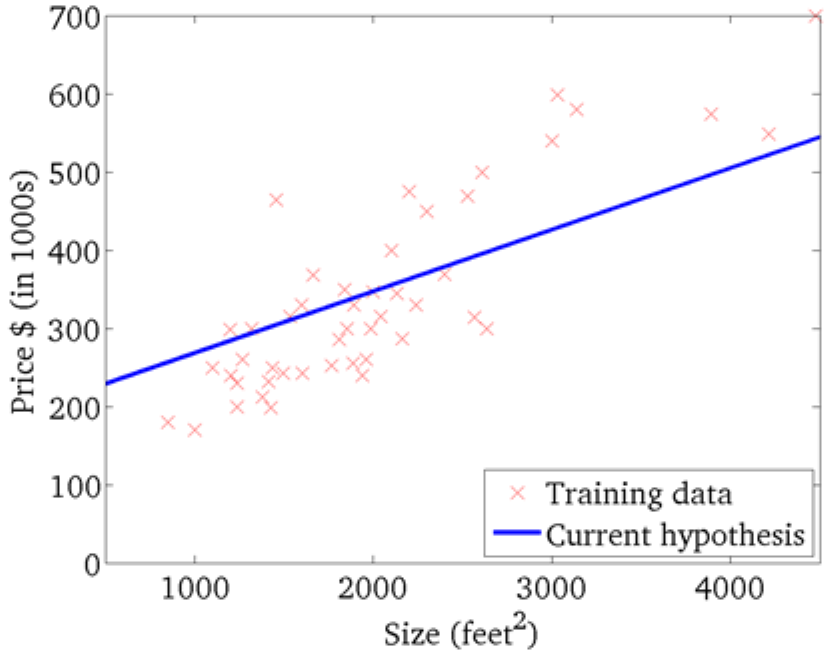
$f_\beta(x)$



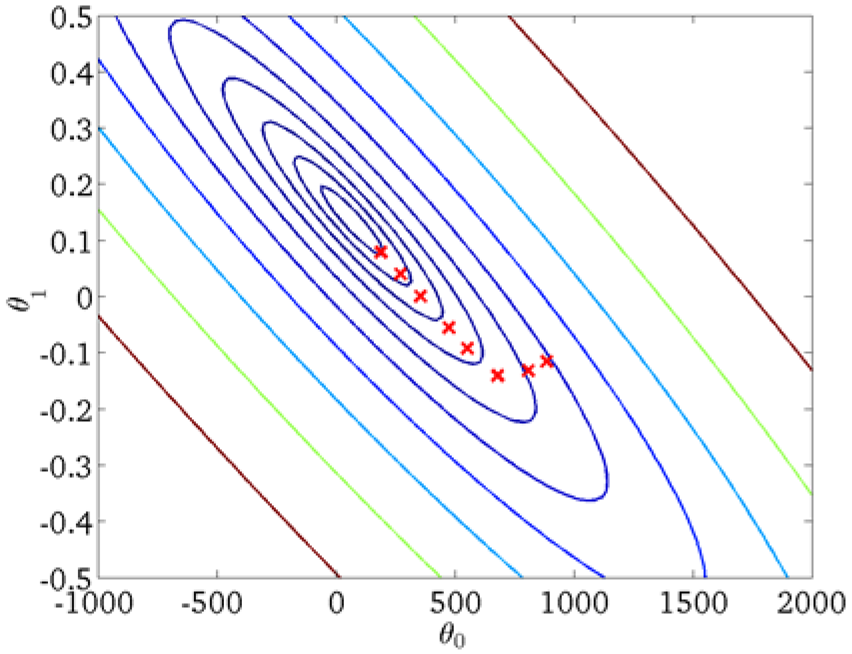
$L(\beta; Z)$

Slide by Andrew Ng

Strategy 2: Gradient Descent



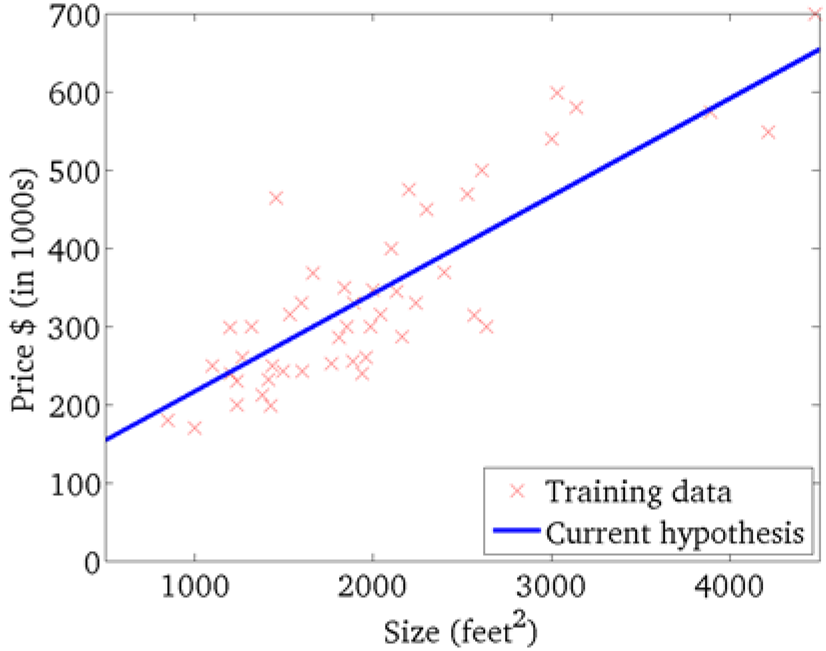
$f_\beta(x)$



$L(\beta; Z)$

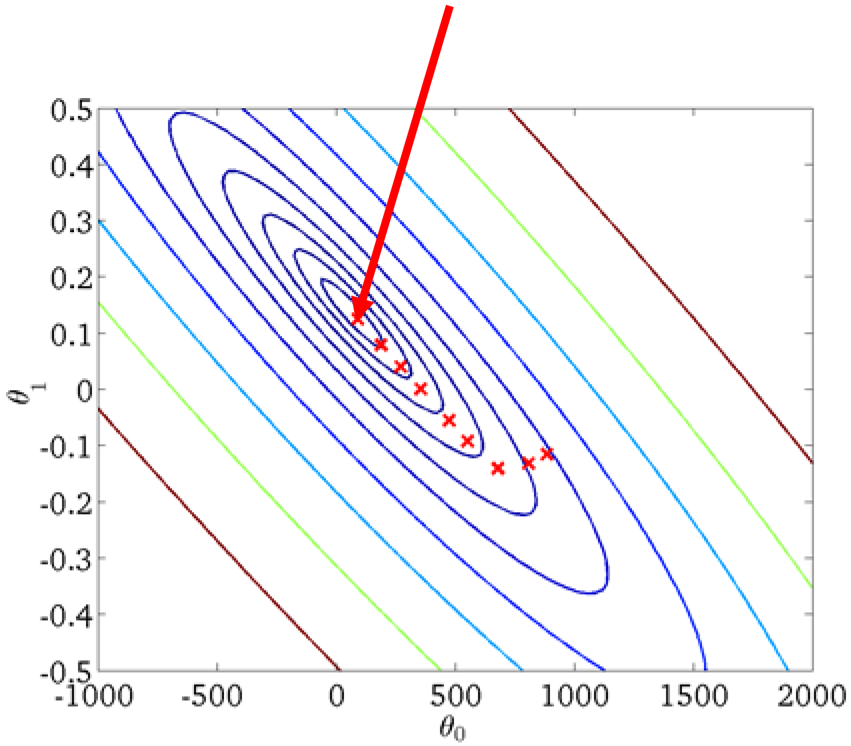
Slide by Andrew Ng

Strategy 2: Gradient Descent



$f_\beta(x)$

Minimizer of loss function



$L(\beta; Z)$

Slide by Andrew Ng

Stochastic Gradient Descent

What if we just used the single-sample gradient of a randomly drawn sample as a noisy approximation to the mean of gradients?

Stochastic Gradient Descent

Batch Gradient Descent

```
Initialize $\beta$
Repeat T times till convergence {
    $\beta_j \leftarrow \beta_j - \alpha \sum_{i=1}^{N} 2(y_i - \beta^\top x_i)x_i$
}
```

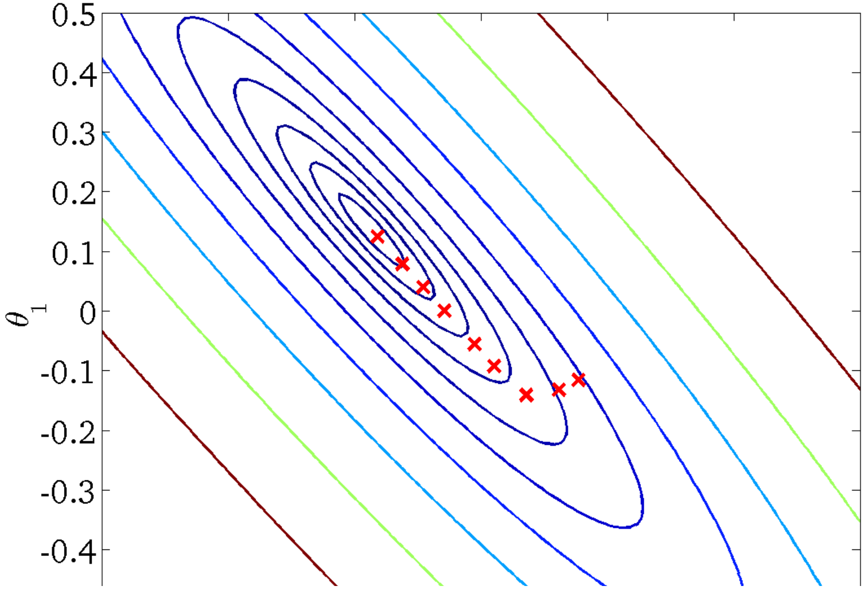
We are descending the original loss function $L(\beta; Z)$.

Stochastic Gradient Descent

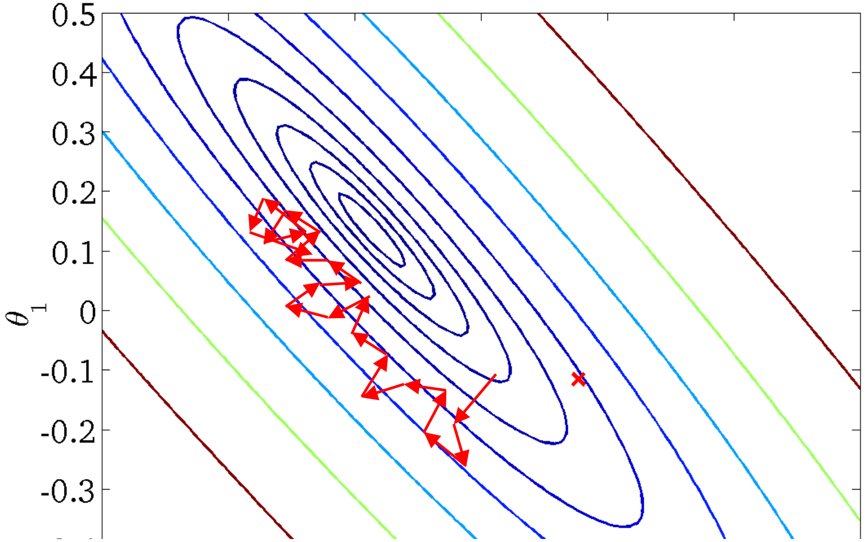
```
Initialize $\beta$
Randomly shuffle dataset
Repeat T' times until convergence {
    For i = 1...N, do
        $\beta_j \leftarrow \beta_j - \alpha 2(y_i - \beta^\top x_i)x_i$
}
```

At each step, we are descending a different loss function specific to the chosen sample $L(\beta; Z_i = \{(x_i, y_i)\})$.

Noisy Gradients in SGD


Full Dataset / “Batch” GD

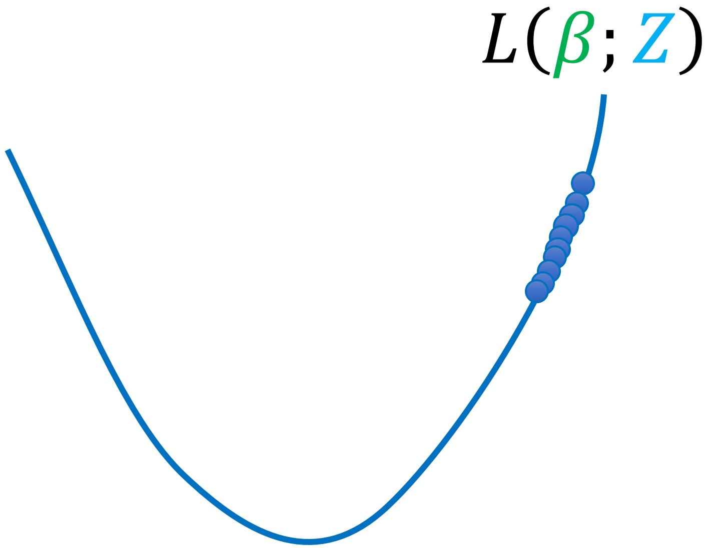
Walking down a hill steadily


Stochastic GD

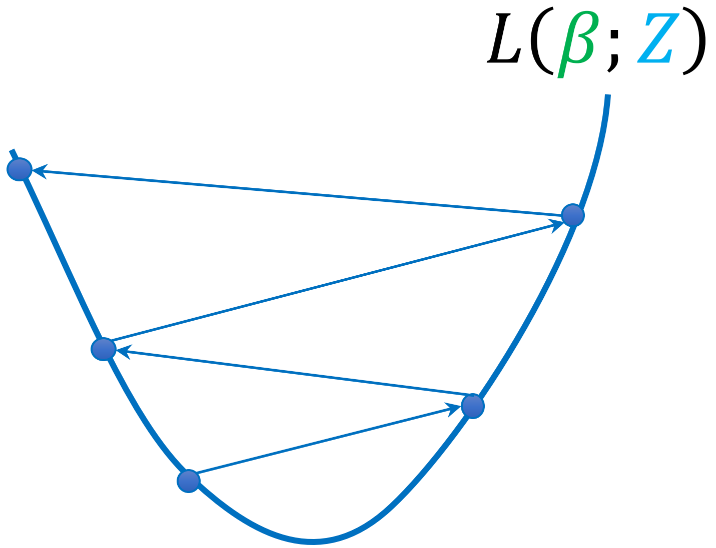
Walking down a slightly perturbed version of the hill at each step

- Learning rate α is typically held constant
- One heuristic is to decrease α over time to force $\boldsymbol{\theta}$ to converge: $\alpha_t = \frac{constant1}{iterationNumber\ t + constant2}$

Choice of Learning Rate



Problem: α too small
- $L(\beta; Z)$ decreases slowly



Problem: α too large
- $L(\beta; Z)$ increases!

Plot $L(\beta_t; Z_{\text{train}})$ vs. t to diagnose these problems

Choice of Learning Rate

- α is a hyperparameter for gradient descent that we need to choose
  - Can set just based on training data

- **Rule of thumb**
  - $\boldsymbol{\alpha}$ **too small:** Loss decreases slowly
  - $\boldsymbol{\alpha}$ **too large:** Loss increases!

- Try rates $\alpha \in \{1.0, 0.1, 0.01, \ldots\}$ (can tune further once one works)

Comparison of Strategies

- **Closed-form solution**
  - No hyperparameters
  - Slow if n or d are large
- **Gradient descent**
  - Need to tune α
  - Scales to large n and d
- For linear regression, there are better optimization algorithms, but gradient descent is very general
  - Accelerated gradient descent is an important tweak that improves performance in practice (and in theory)

L_2 Regularized Linear Regression

- Recall that linear regression with L_2 regularization minimizes the loss

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d} \beta_j^2$$

L_2 Regularized Linear Regression

- Recall that linear regression with L_2 regularization minimizes the loss

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda\sum_{j=1}^{d}\beta_j^2 = \frac{1}{n}\|Y - X\beta\|_2^2 + \lambda\|\beta\|_2^2$$

- Gradient is

$$\nabla_\beta L(\beta; Z) = -\frac{2}{n}X^\top Y + \frac{2}{n}X^\top X\beta + 2\lambda\beta$$

Strategy 1: Closed-Form Solution

- Gradient is

$$\nabla_\beta L(\beta; Z) = -\frac{2}{n} X^\top Y + \frac{2}{n} X^\top X\beta + 2\lambda\beta$$

- Setting $\nabla_\beta L\left(\hat{\beta}; Z\right) = 0$, we have $(X^\top X + n\lambda I)\hat{\beta} = X^\top Y$

- Always invertible if $\lambda > 0$, so we have

$$\hat{\beta}(Z) = (X^\top X + n\lambda I)^{-1} X^\top Y$$

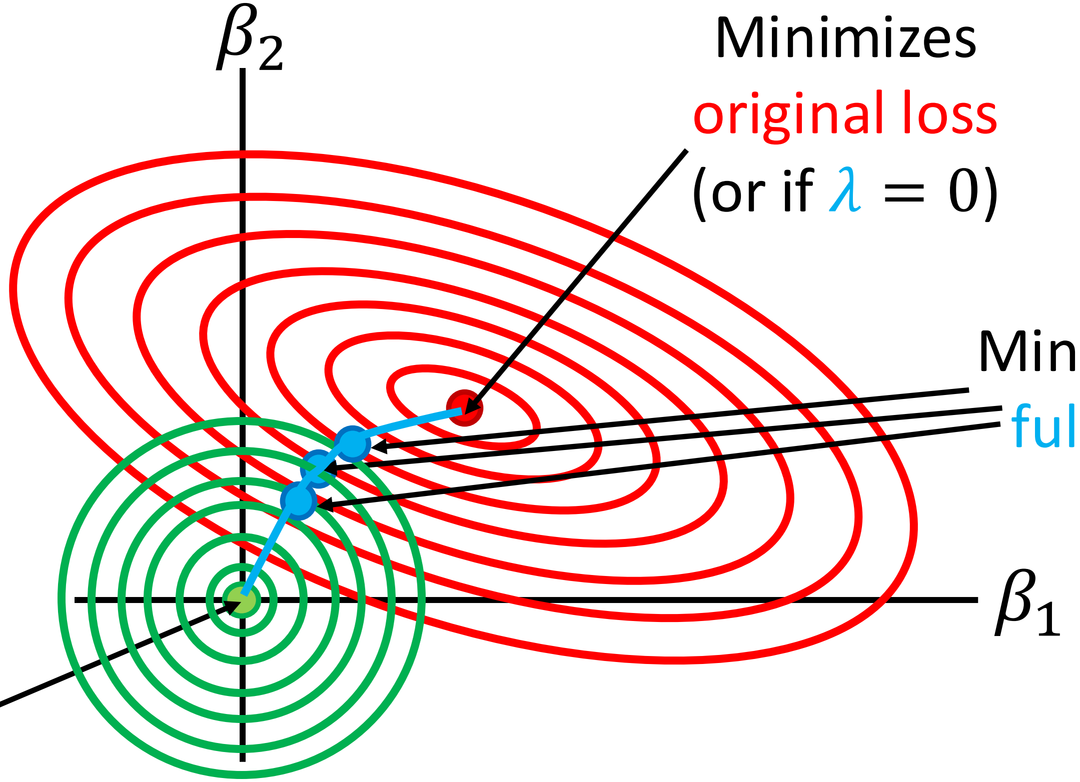
Strategy 2: Gradient Descent

- Gradient is

$$\nabla_{\beta} L(\beta; Z) = -\frac{2}{n} X^{\top} Y + \frac{2}{n} X^{\top} X \beta + 2\lambda\beta$$

- Same algorithm as vanilla linear regression (a.k.a. OLS)
- **Intuition:** The extra term $\lambda\beta$ in the gradient is **weight decay** that encourages β to be small

L_2 Regularization



Minimizes
original loss
(or if $\lambda = 0$)

Minimizes
full loss

Minimizes
regularization term
(or if $\lambda \to \infty$)

- At this point, the gradients are **equal** (with opposite sign)
- Tradeoff depends on choice of λ

$$L(\beta; Z) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d} \beta_j^2$$

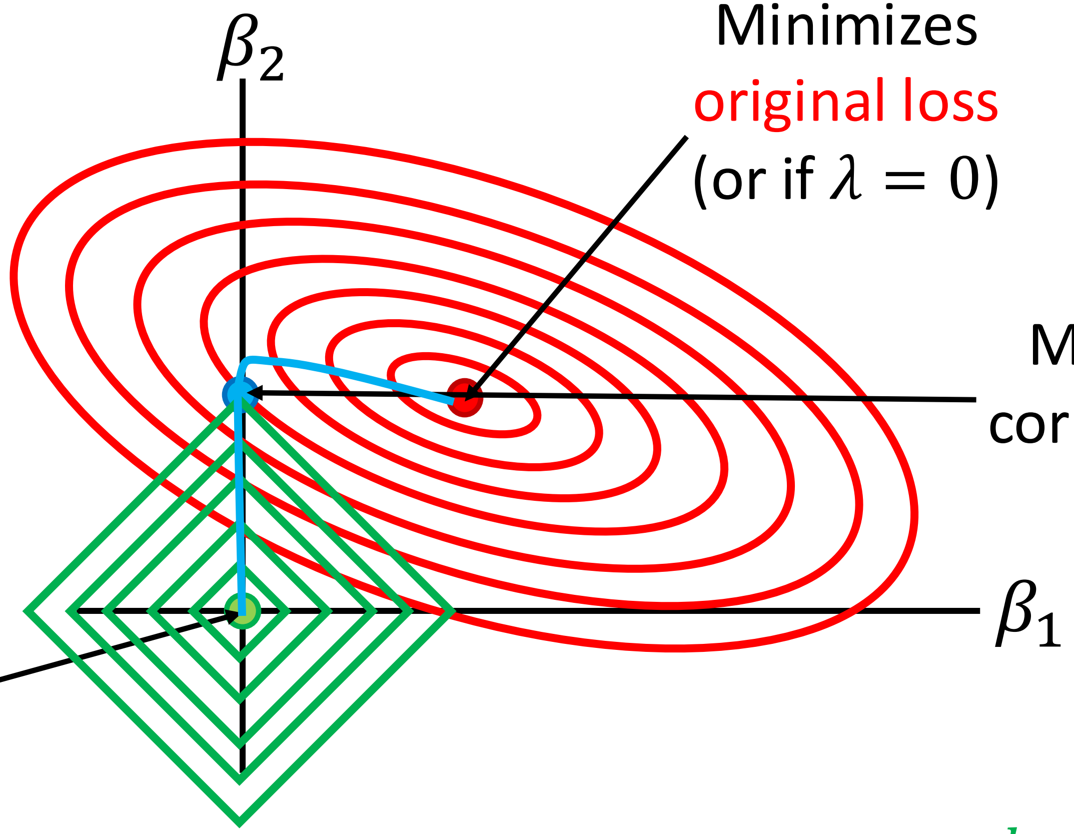
What About L_1 Regularization?

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d}|\beta_j|$$

- Gradient descent still works!

- Specialized algorithms work better in practice
  - **Simple one:** Gradient descent + soft thresholding
  - Basically, if $|\beta_{t,j}| \leq \lambda$, just set it to zero
  - Good theoretical properties

L_1 Regularization



Minimizes
original loss
(or if $\lambda = 0$)

Minimizer of full loss at
corner → sparse ($\beta_1 = 0$)!

Minimizes
regularization term
(or if $\lambda \to \infty$)

$$L(\beta; Z) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d}|\beta_j|$$

Loss Minimization View of ML

- **Two design decisions**
  - **Model family:** What are the candidate models f? (E.g., linear functions)
  - **Loss function:** How to define “approximating”? (E.g., MSE loss)

Loss Minimization View of ML

- Three **design decisions**
  - **Model family:** What are the candidate models f? (E.g., linear functions)
  - **Loss function:** How to define “approximating”? (E.g., MSE loss)
  - **Optimizer:** How do we minimize the loss? (E.g., gradient descent)

This Module: Linear Regression

- Your very first supervised learning algorithm

- Regression with **real value** label $y_i \in \mathbb{R}$

Next Module:

- Classification with **discrete value** $y_i \in \{c_1, \ldots, c_k\}$