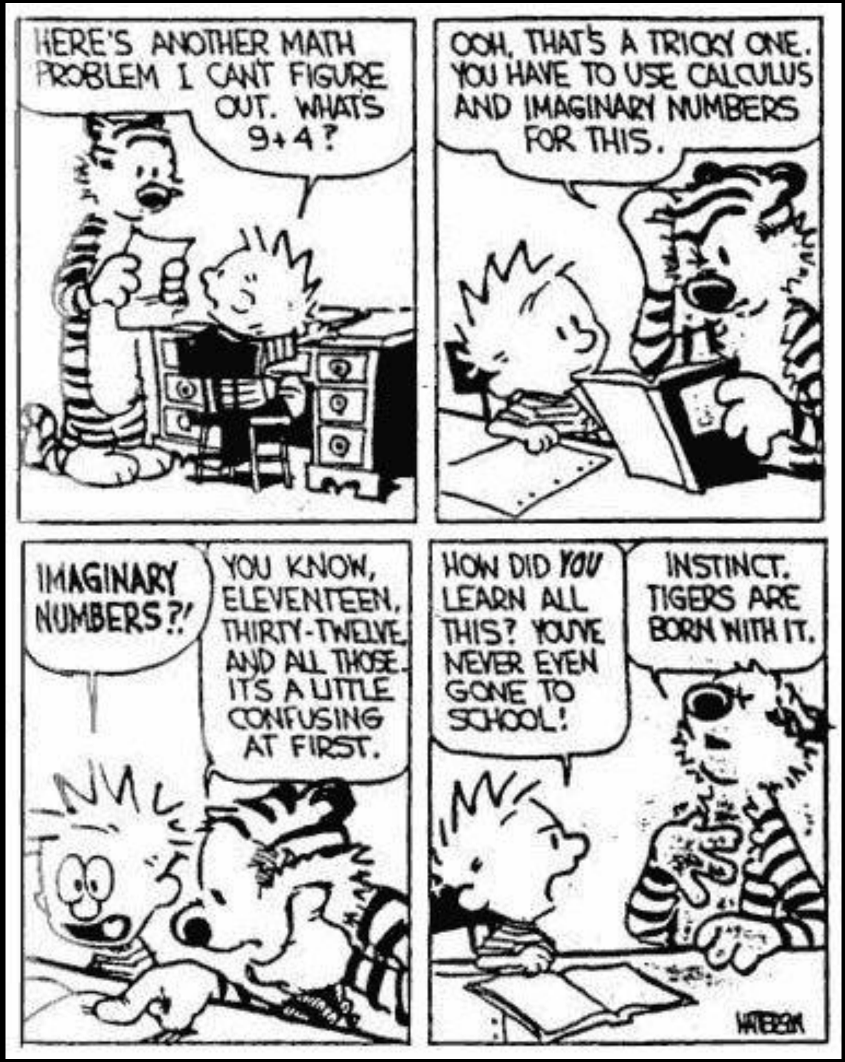


INFO 251: Applied Machine Learning

Gradient Descent

A pen and paper might be useful for today's lecture (to sketch graphs)

Course Outline

- Causal Inference and Research Design
  - Experimental methods
  - Non-experiment methods
- **Machine Learning**
  - Design of Machine Learning Experiments
  - **Linear Models and Gradient Descent**
  - Non-linear models
  - Fairness and Bias in ML
  - Neural models
  - Deep Learning
  - Practicalities
  - Unsupervised Learning
- Special topics

Key Concepts (previous lecture)

- Decision boundaries
- Voronoi diagrams
- (*K*-)Nearest Neighbors
- Similarity and Distance metrics
- Normalization and Standardization
- Feature weighting

Key Concepts (today's lecture)

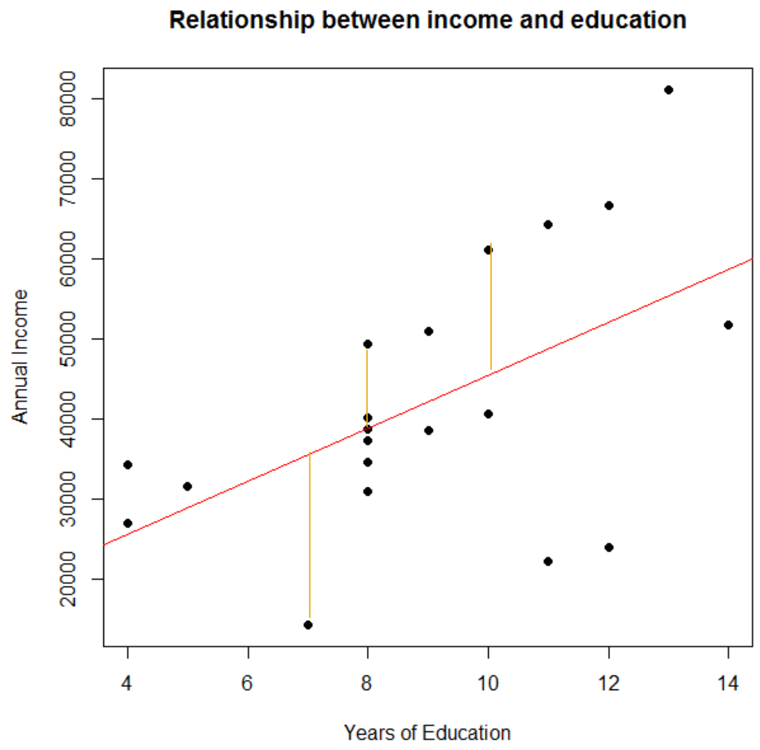
- Cost Functions
- Gradient Descent
- Local and global minima
- Convex functions
- Incremental vs. Batch GD
- Learning rates
- Feature scaling

Cost minimization

- In general:
  - We make a prediction of Y using some function f(X)
  - To choose the best model:
    - Define a loss function J(Y, f(X))
    - Minimize the expected loss of J
- With linear regression:
  - f is a linear function (e.g., $\alpha + \beta X$)
  - OLS regression minimizes squared-error loss $E(Y-f(X))^2$

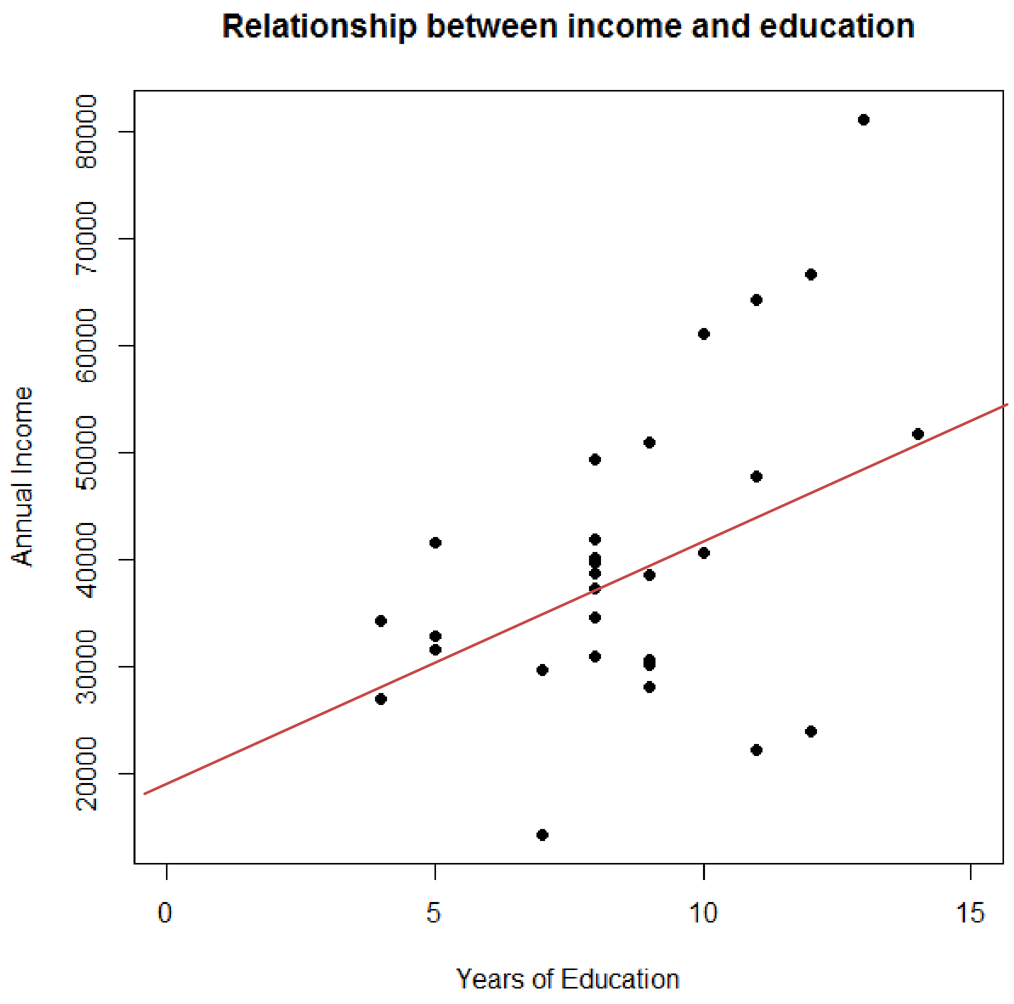
Linear Regression

- OLS as Maximum Likelihood Estimation:
  - $Y_i = \alpha + \beta X_i + \epsilon_i$
  - Idea: Choose α and β so that $\alpha + \beta X_i$ is "as close as possible" to Y_i for training data
- In other words
  - $\min_{\alpha,\beta} \sum_{i=1}^{N} (\alpha + \beta X_i - Y_i)^2$
- In general, we are minimizing a **Cost Function** *J*
  - $\min_{\alpha,\beta} J(\alpha, \beta)$
  - In the case of OLS, we use a "squared error" cost function
  - $J(\alpha, \beta) = \frac{1}{2N} \sum_{i=1}^{N} (\alpha + \beta X_i - Y_i)^2$



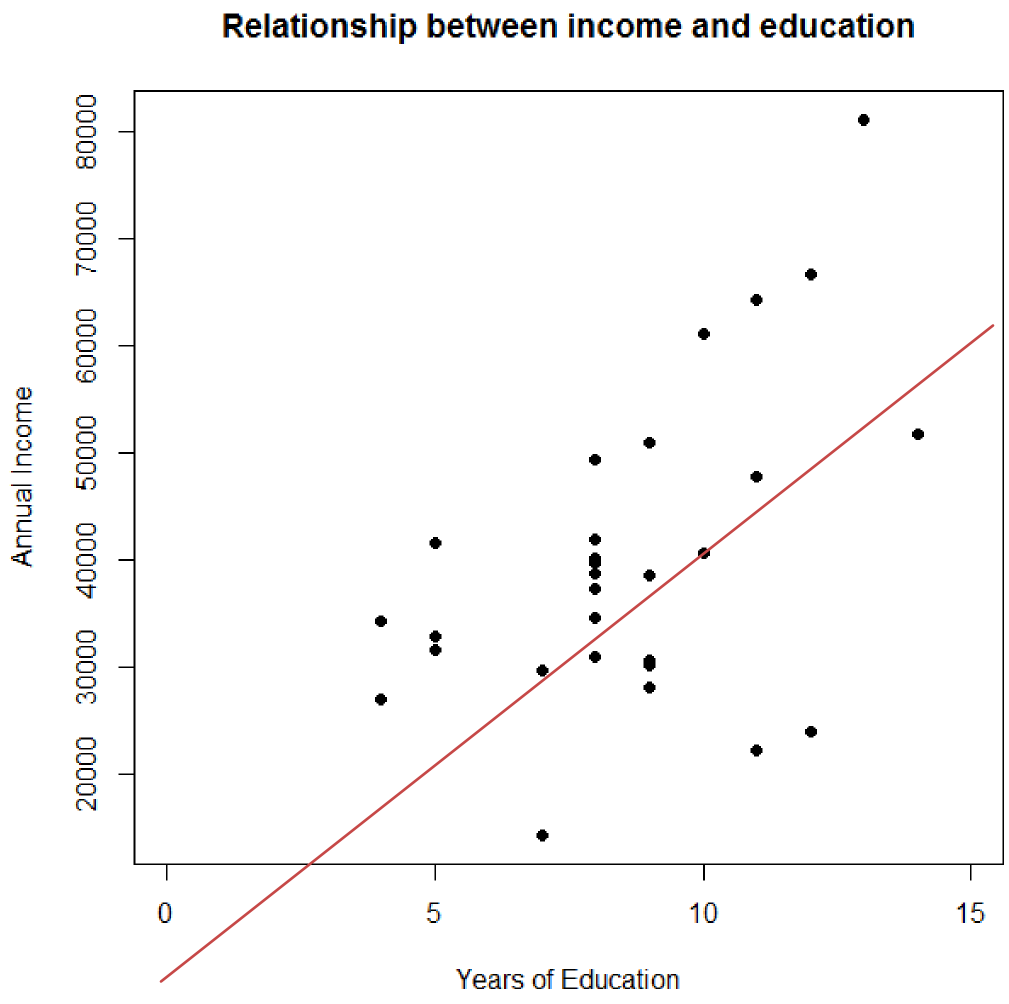
General formulation (OLS)

- Model ("hypothesis")
  - $Y_i = \alpha + \beta X_i$
- Parameters
  - α, β
- Cost Function
  - $J(\alpha, \beta) = \frac{1}{2N} \sum_{i=1}^{N} (\alpha + \beta X_i - Y_i)^2$
- Objective
  - $\min_{\alpha,\beta} J(\alpha, \beta)$



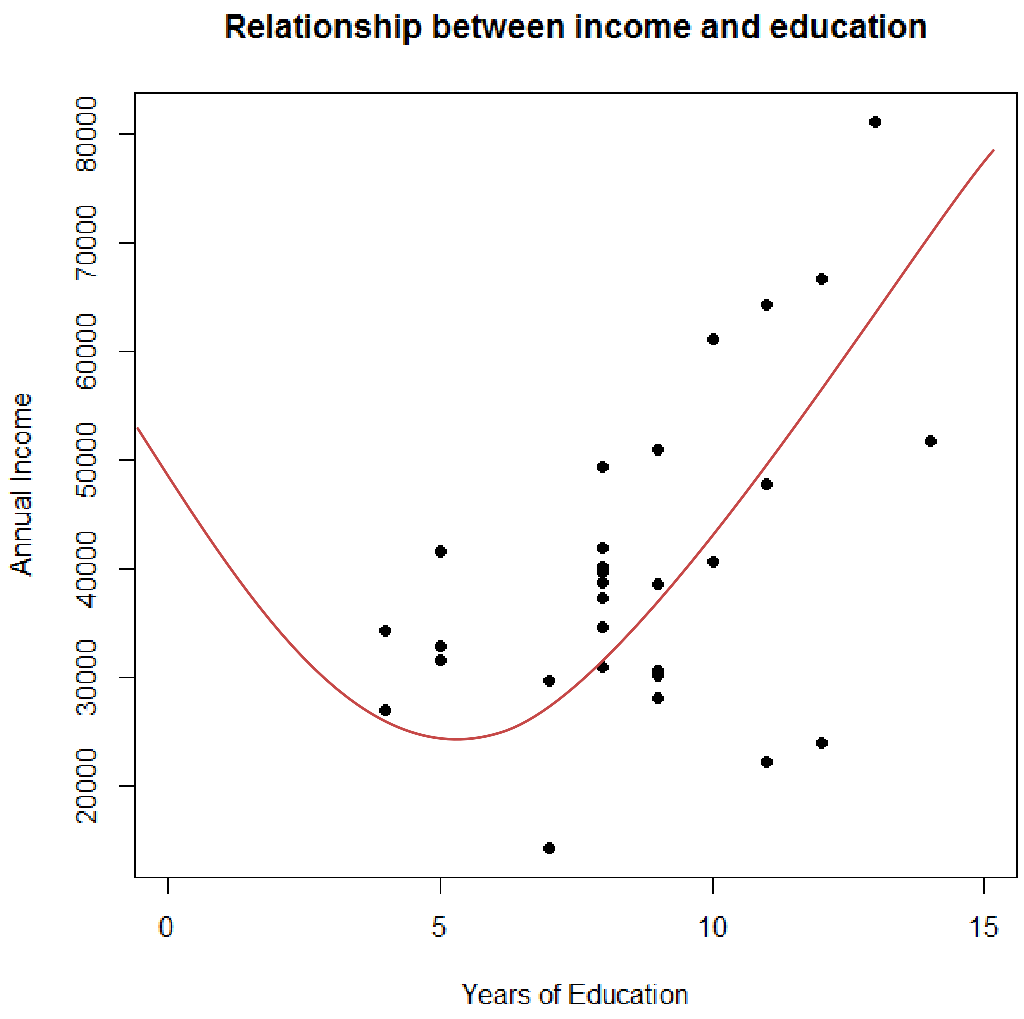
OLS with no intercept

- Model ("hypothesis")
  - $Y_i = \beta X_i$
- Parameters
  - β
- Cost Function
  - $J(\beta) = \frac{1}{2N}\sum_{i=1}^{N}(\beta X_i - Y_i)^2$
- Objective
  - $\min_{\beta} J(\beta)$



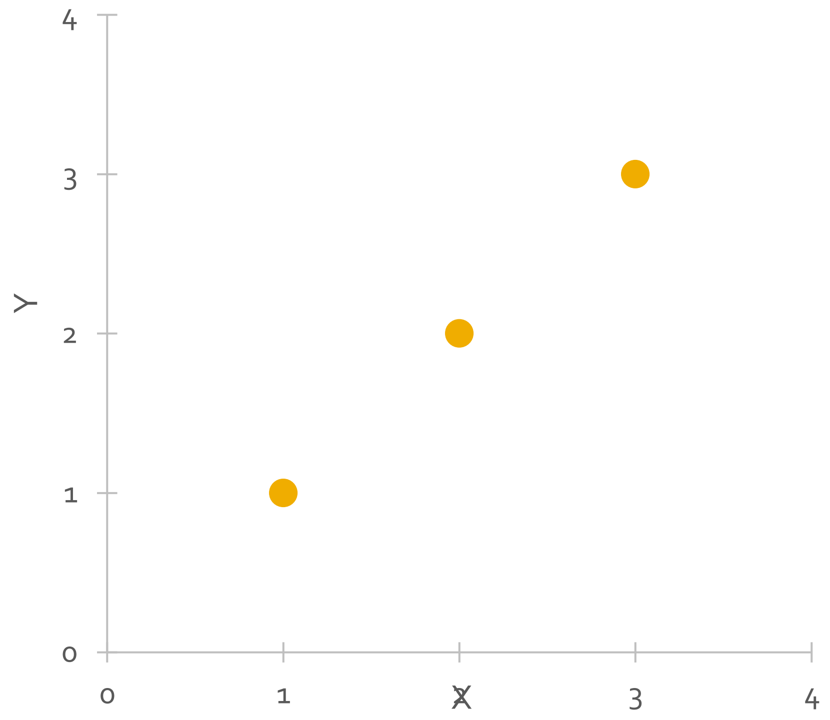
Fill in the blanks

- Model ("hypothesis")
  - Income is a linear function of education and also eduction², i.e. nonlinearities exist
  - $Y_i = \alpha + \beta X_i + \gamma X_i^2$
- Parameters
  - α, β, γ
- Cost Function
  - Use "absolute error" cost function
  - $J(\alpha,\beta,\gamma) = \frac{1}{N}\sum_{i=1}^{N} |\alpha + \beta X_i + \gamma X_i^2 - Y_i|$
- Objective
  - $\min_{\alpha,\beta,\gamma} J(\alpha,\beta,\gamma)$



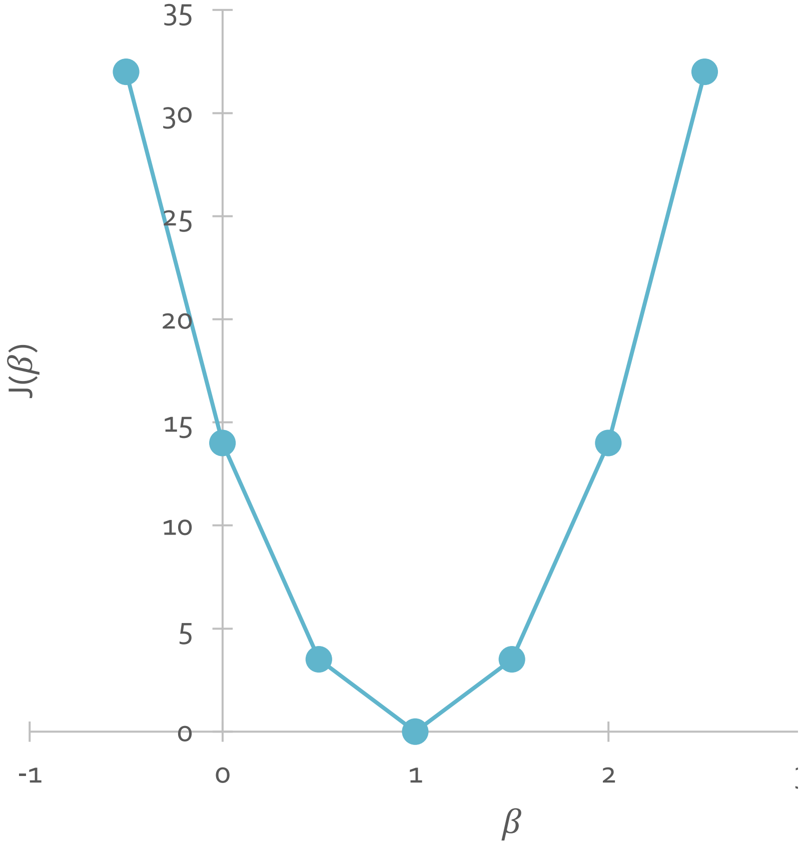
Exercise: Computing Cost

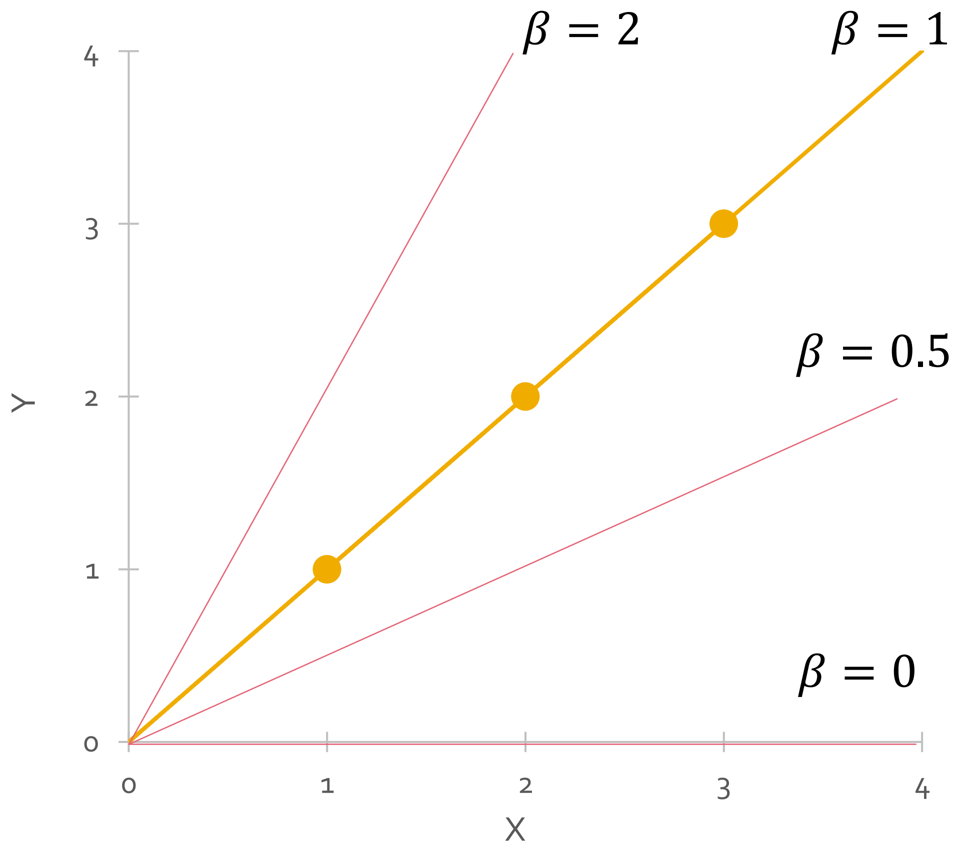
- Assume our data look like this ($N = 3$)
  - $X_1 = 1, \;\; Y_1 = 1$
  - $X_2 = 2, \;\; Y_2 = 2$
  - $X_3 = 3, \;\; Y_3 = 3$
- Our model is $Y_i = \beta X_i$
  - Our cost function is squared error: $J(\beta) = \sum_{i=1}^{N} (\beta X_i - Y_i)^2$
- Your task is to compute $J(\beta)$, given these 3 points, for:
  - $\beta = 1$
  - $\beta = 0$
  - $\beta = 2$
  - $\beta = 0.5$ (you might need a calculator)
- Draw a plot of $J(\beta)$ as a function of β



Visualizing Cost (1 parameter)

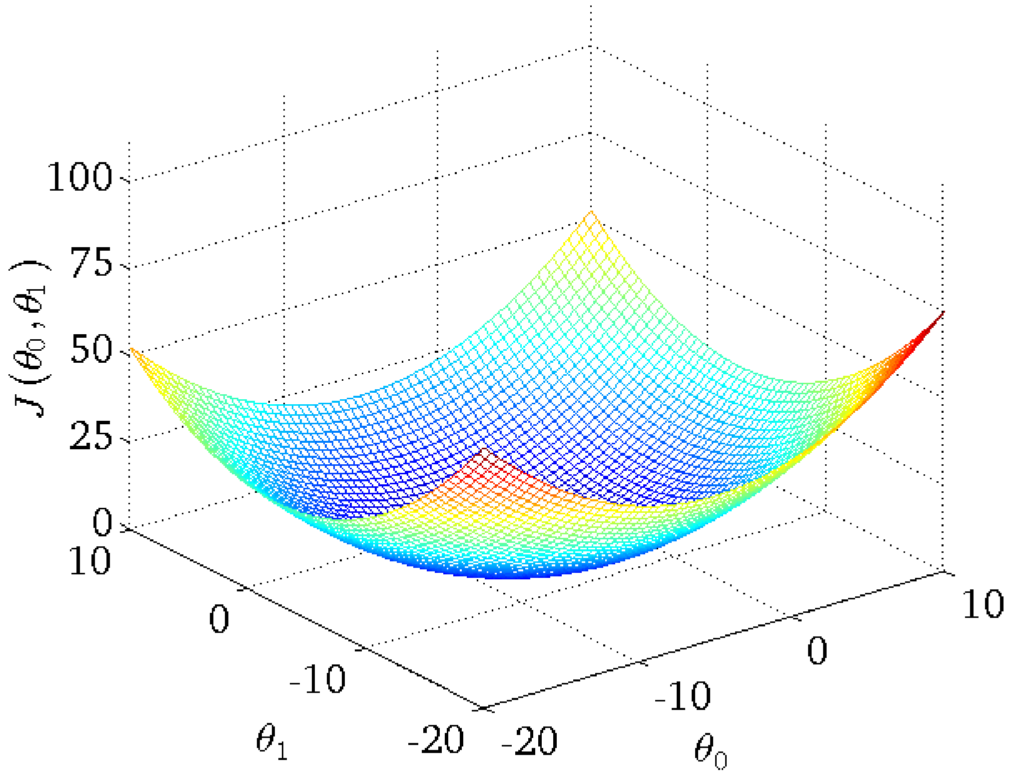
- Where is $J(\beta)$ minimized?





Visualizing Cost (2 parameters)

- Generalizing to a multi-dimensional loss surface
  - Model ("hypothesis"): $Y_i = \theta_1 + \theta_2 X_i$
  - Objective: $\min_{\theta_1,\theta_2} J(\theta_1, \theta_2)$



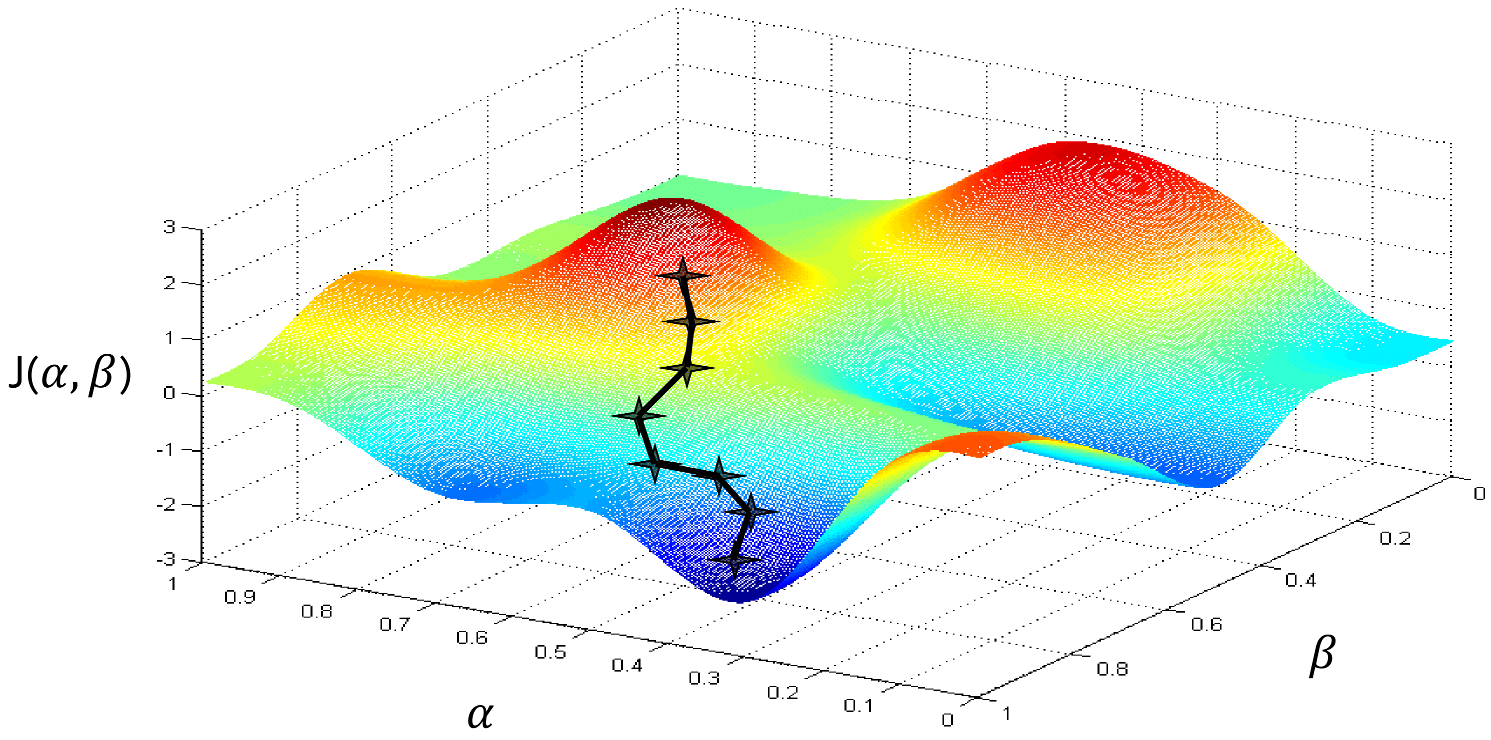
Outline

- Cost functions
- **Gradient descent**
- Feature scaling

Gradient Descent

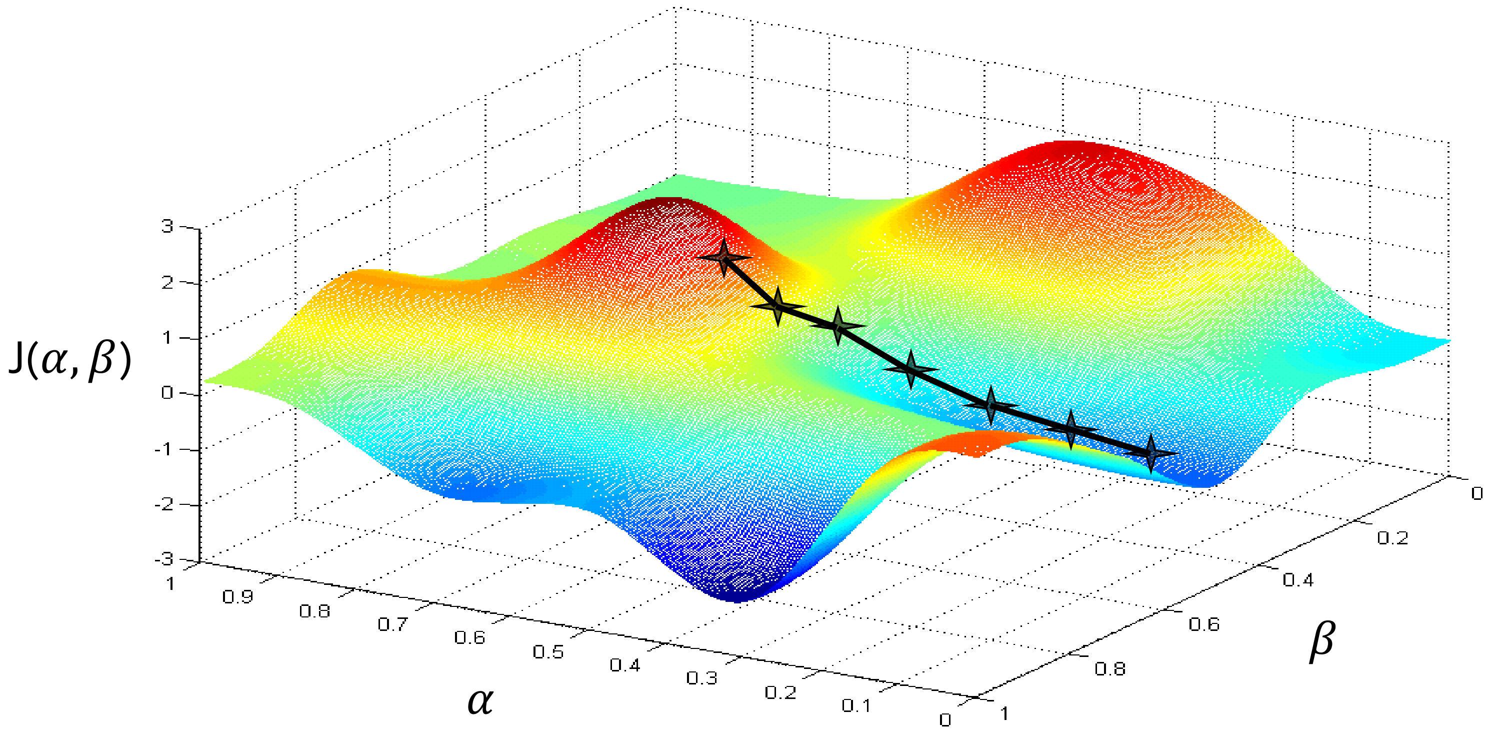
- Gradient descent provides a principled method/algorithm to minimize the cost function J
- Idea: to solve $\min_{\alpha,\beta} J(\alpha, \beta)$
  - Initialize α, β
  - Change α, β in some way that reduces $J(\alpha, \beta)$
  - Eventually we will end up at a minimum
- What about analytic solution: $(\mathrm{X}'\mathrm{X})^{-1}\mathrm{X}'\mathrm{Y}$
  - Sometimes not practically feasible (too much data, multicollinearity)

Gradient Descent: Visualization



Andrew Ng

Local Minima



Andrew Ng

Gradient Descent Algorithm (incremental)

- In pseudo-code:

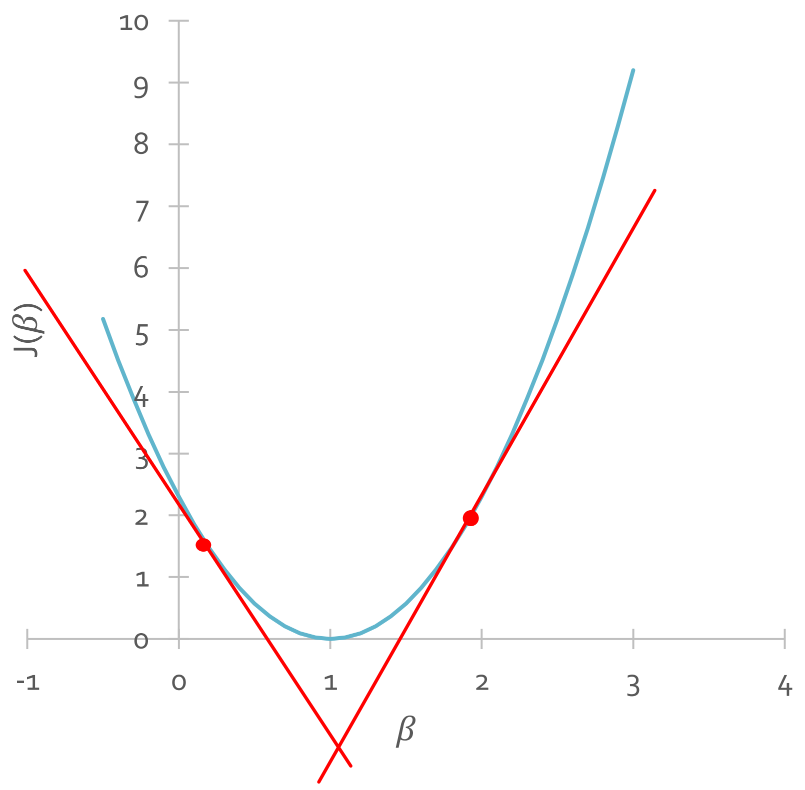
```
Choose an initial vector of parameters α, β
Choose learning rate R
Repeat until convergence (i.e., until an approximate minimum is obtained):
   For each example i in training set:
```

$$\left.\boxed{\begin{aligned} \alpha &\leftarrow \alpha - R\frac{\partial}{\partial\alpha}J(\alpha,\beta) \\ \beta &\leftarrow \beta - R\frac{\partial}{\partial\beta}J(\alpha,\beta) \end{aligned}}\;\right\}\ \text{Simultaneous update}$$

- With multiple predictors/regressors…
  - $Y_i = \alpha + \beta_1 X_{i1} + \beta_2 X_{i2} + \ \dots + \beta_k X_{ik}$

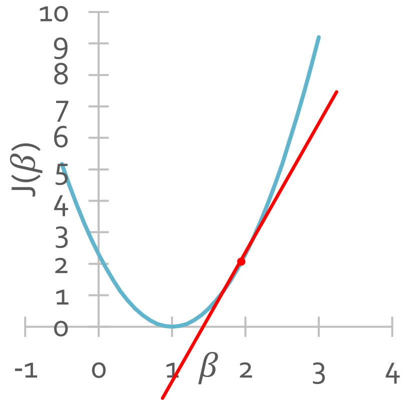
Gradient Descent: Derivative

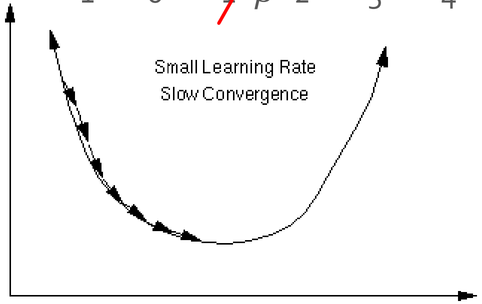
- In 1-Dimension
- Update Rule:
  - β <- β − R $\frac{\partial}{\partial \beta} J(\alpha, \beta)$
- Initialize β at 1.9
  - What's the derivative $\frac{\partial}{\partial \beta} J(\alpha, \beta)$?
  - How does β update?
- Initialize β at 0.2
  - What's the derivative $\frac{\partial}{\partial \beta} J(\alpha, \beta)$?
  - How does β update?

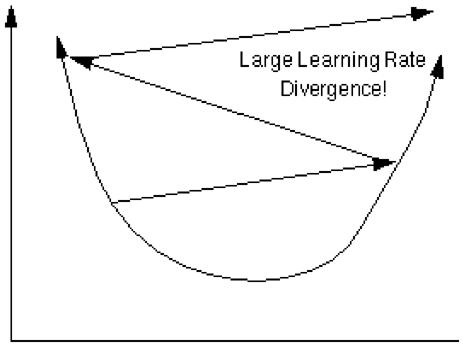


Gradient Descent: Learning Rate

- $\beta \;\texttt{<-}\; \beta \;-\; R\frac{\partial}{\partial \beta}J(\alpha,\beta)$
- What does R do?
- Small R:
  - Gradient descent can be slow
- Large R:
  - Can overshoot the minimum
  - May fail to converge
  - May diverge!







Gradient Descent: Convergence

- Do we need to change the learning rate?

```
Choose an initial vector of parameters α, β
Choose learning rate R
Repeat until convergence:

    For each example i:
```

$$\alpha \leftarrow \alpha - R\frac{\partial}{\partial\alpha}J(\alpha,\beta)$$

$$\beta \leftarrow \beta - R\frac{\partial}{\partial\beta}J(\alpha,\beta)$$

- Not typically.Gradient descent can converge to a local minimum, even with the learning rate fixed
  - As we approach a local minimum, gradient descent takes smaller steps
  - But adaptive learning rates can help speed up convergence, prevent overshooting

Gradient Descent: Regression

- Gradient Descent

```
Repeat until convergence:
```

$$\alpha \leftarrow \alpha - R \frac{\partial}{\partial \alpha} J(\alpha, \beta)$$

$$\beta \leftarrow \beta - R \frac{\partial}{\partial \beta} J(\alpha, \beta)$$

- Regression cost function
  - $J(\alpha, \beta) = \frac{1}{2N} \sum_{i=1}^{N} (\alpha + \beta X_i - Y_i)^2$
- The missing pieces: $\frac{\partial}{\partial \alpha} J(\alpha, \beta)$ and $\frac{\partial}{\partial \beta} J(\alpha, \beta)$
  - $\frac{\partial}{\partial \alpha} J(\alpha, \beta) = \frac{\partial}{\partial \alpha} \frac{1}{2N} \sum_{i=1}^{N} (\alpha + \beta X_i - Y_i)^2$
  - $\frac{\partial}{\partial \beta} J(\alpha, \beta) = \frac{\partial}{\partial \beta} \frac{1}{2N} \sum_{i=1}^{N} (\alpha + \beta X_i - Y_i)^2$

Gradient Descent: Regression

- Partial derivatives:

$$\frac{\partial}{\partial\alpha}J(\alpha,\beta) = \frac{\partial}{\partial\alpha}\frac{1}{2N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)^2$$
$$= \frac{\partial}{\partial\alpha}\frac{1}{2N}\sum_{i=1}^{N}(\alpha + \beta X_i - Y_i)^2$$
$$= \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial}{\partial\alpha}(\alpha + \beta X_i - Y_i)^2$$
$$= \frac{1}{2N}\sum_{i=1}^{N}2(\alpha + \beta X_i - Y_i)\frac{\partial}{\partial\alpha}(\alpha + \beta X_i - Y_i)$$
$$= \frac{1}{N}\sum_{i=1}^{N}(\alpha + \beta X_i - Y_i)$$
$$= \frac{1}{N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)$$

$$\frac{\partial}{\partial\beta}J(\alpha,\beta) = \frac{\partial}{\partial\beta}\frac{1}{2N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)^2$$
$$= \frac{1}{2N}\sum_{i=1}^{N}\frac{\partial}{\partial\beta}(\alpha + \beta X_i - Y_i)^2$$
$$= \frac{1}{2N}\sum_{i=1}^{N}2(\alpha + \beta X_i - Y_i)\frac{\partial}{\partial\beta}(\alpha + \beta X_i - Y_i)$$
$$= \frac{1}{N}\sum_{i=1}^{N}(\alpha + \beta X_i - Y_i)(X_i)$$
$$= \frac{1}{N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)X_i$$

Gradient Descent: Regression

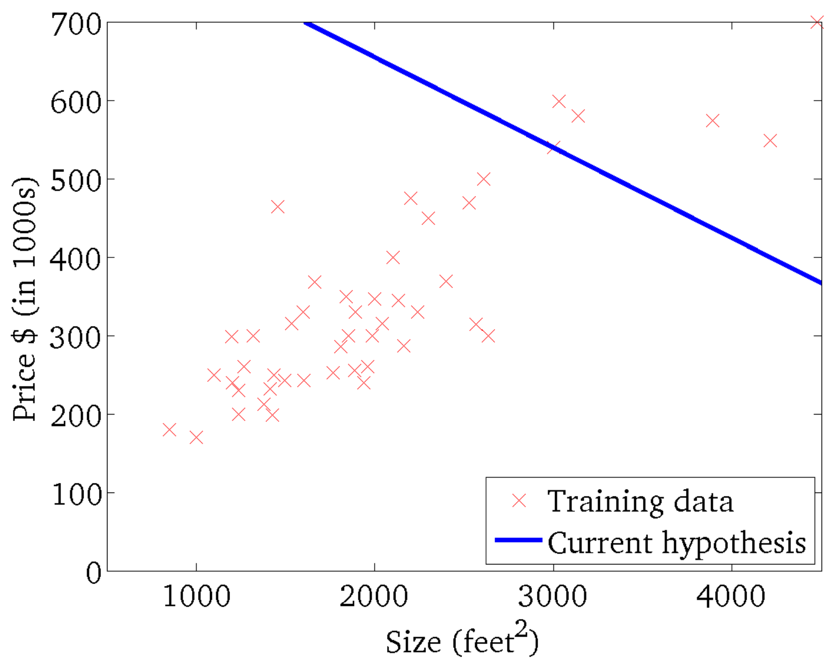
- Gradient Descent Algorithm (linear regression)

```
Repeat until convergence:
```

$\alpha \ \texttt{<-} \ \alpha \ - \ \mathrm{R}\frac{1}{N}\sum_{i=1}^{N}(\alpha + \beta X_i - Y_i)$

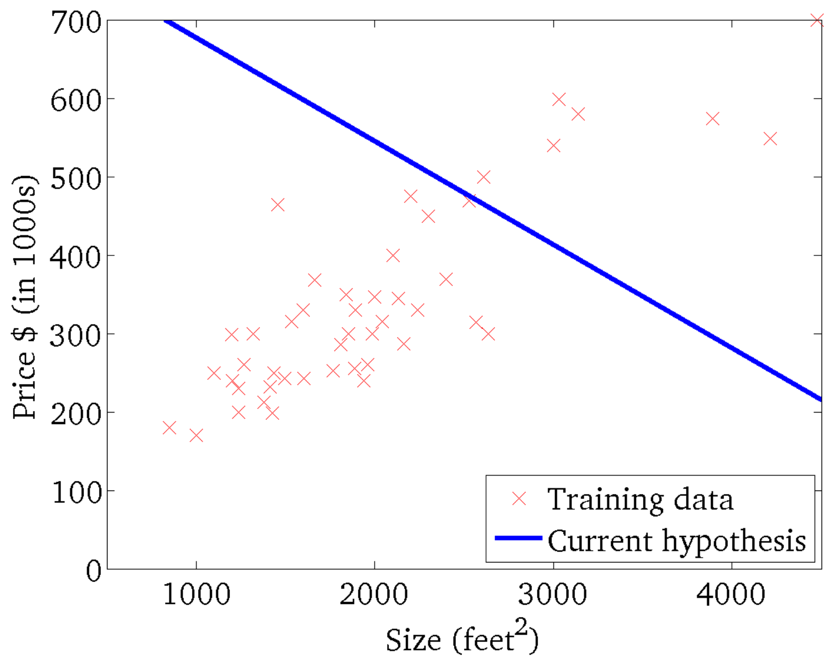
$\beta \ \texttt{<-} \ \beta \ - \ \mathrm{R}\frac{1}{N}\sum_{i=1}^{N}(\alpha + \beta X_i - Y_i)X_i$

Gradient Descent: In Action



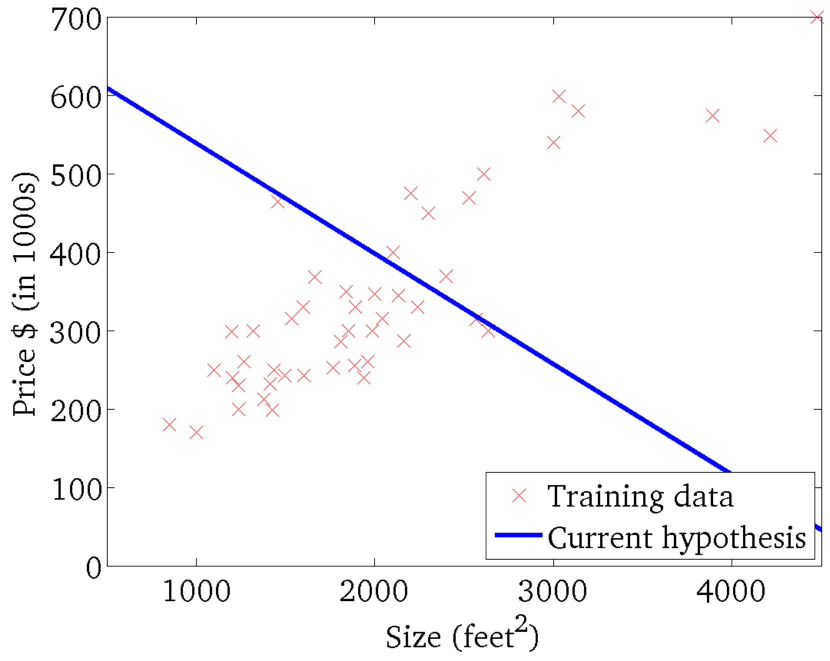
Andrew Ng

Gradient Descent: In Action



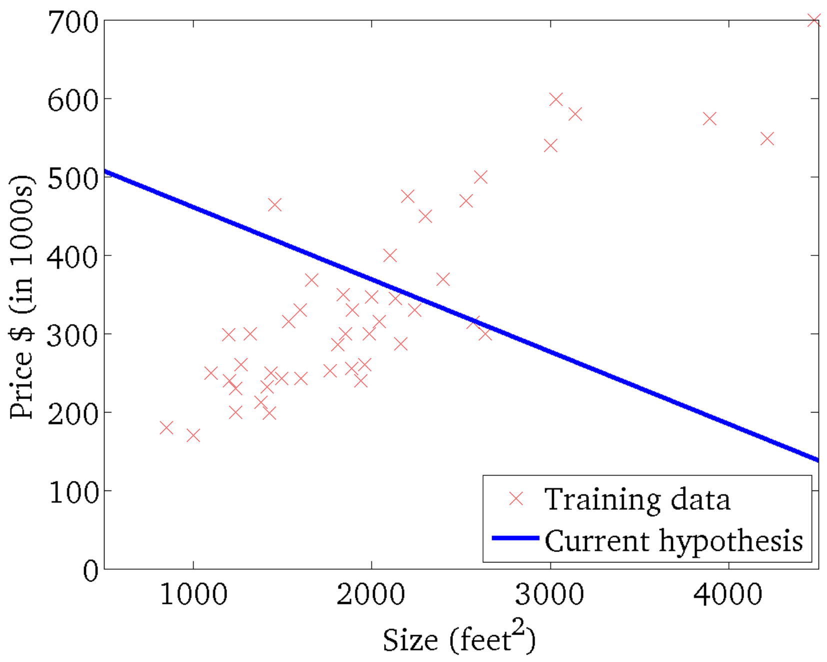
Andrew Ng

Gradient Descent: In Action



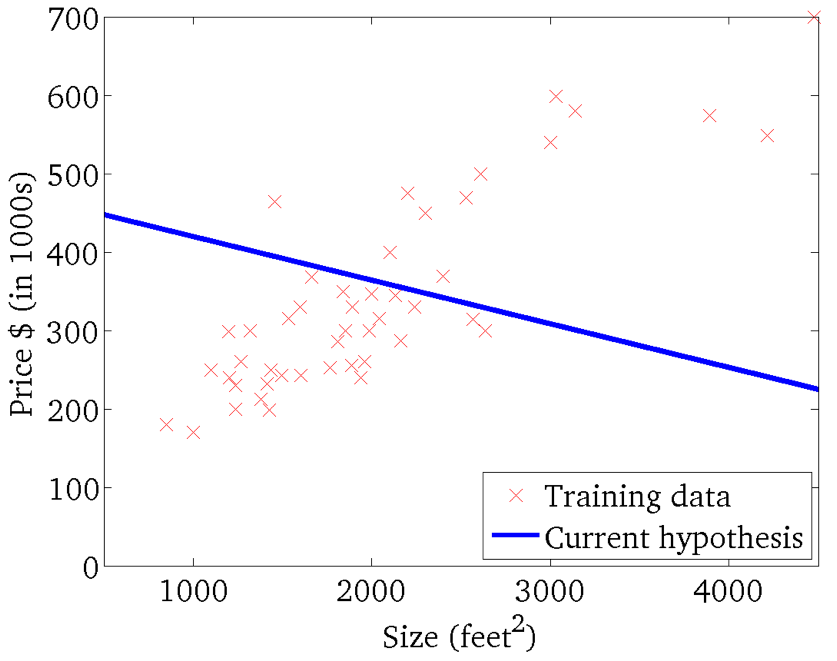
Andrew Ng

Gradient Descent: In Action



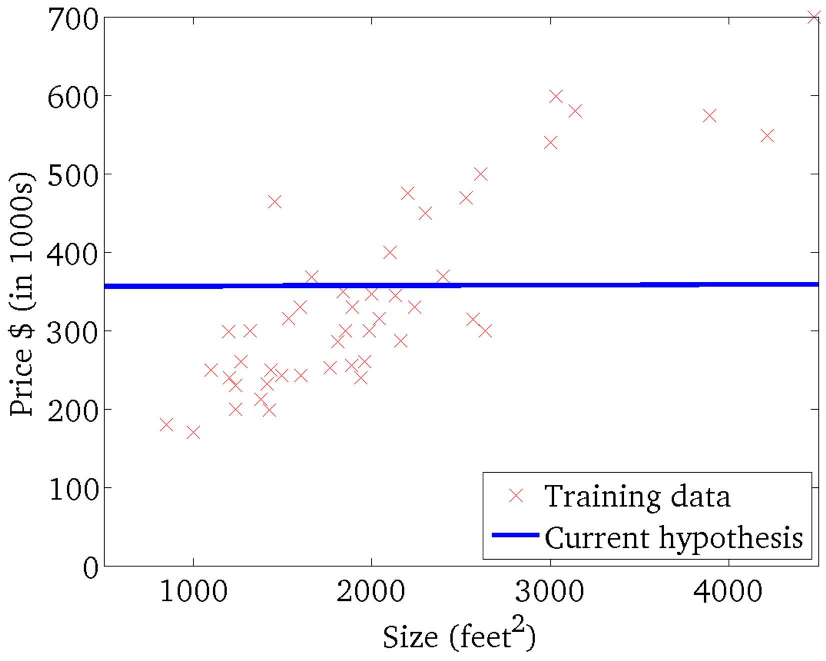
Andrew Ng

Gradient Descent: In Action



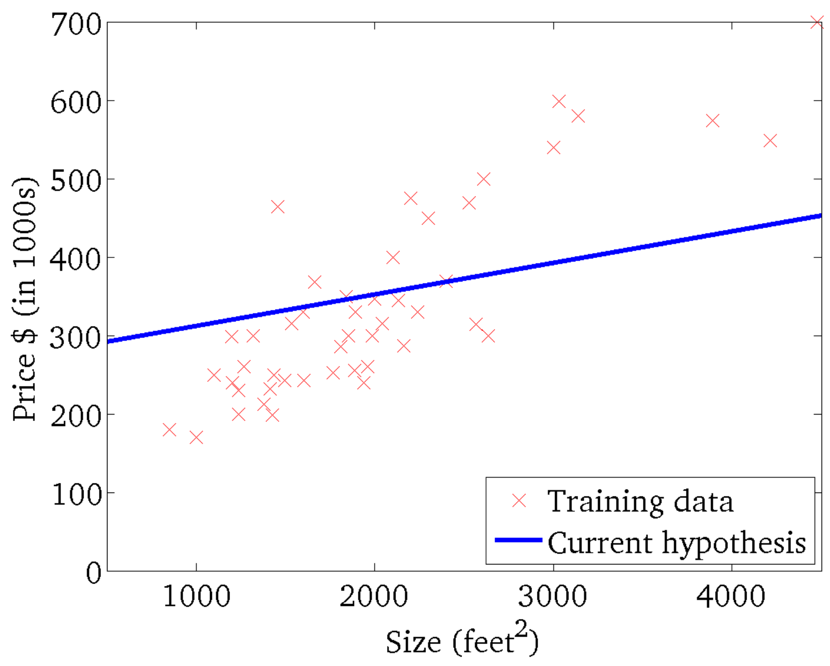
Andrew Ng

Gradient Descent: In Action



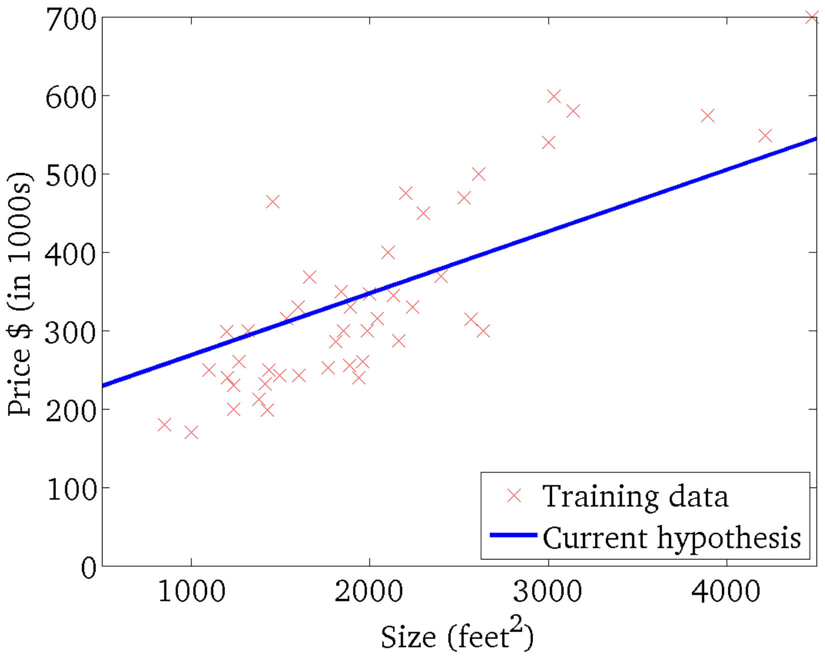
Andrew Ng

Gradient Descent: In Action



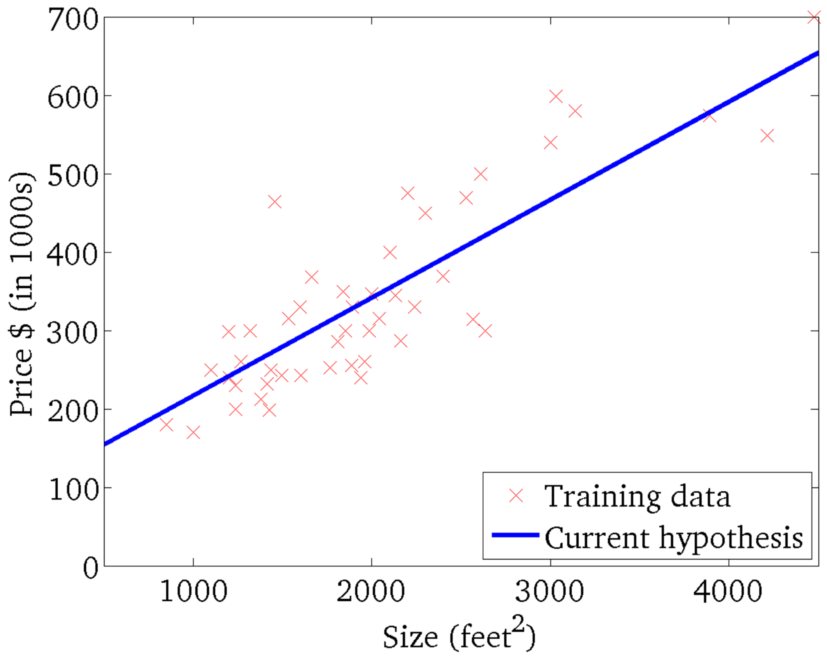
Andrew Ng

Gradient Descent: In Action



Andrew Ng

Gradient Descent: In Action

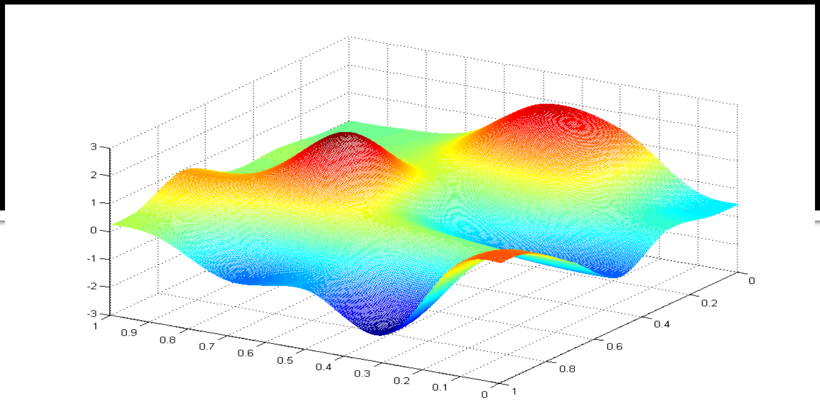


Andrew Ng

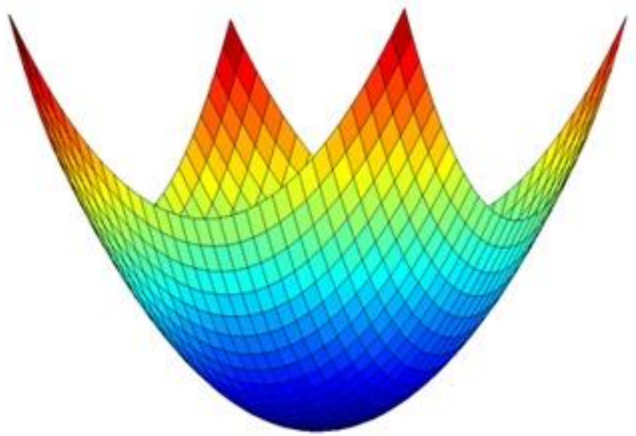
Outline

- Cost functions
- Gradient descent
  - **Local Minima**
  - Batch and Incremental versions
- Feature scaling

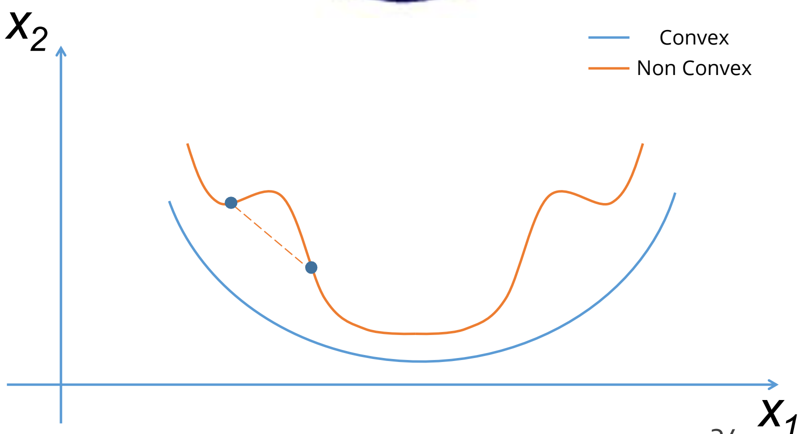
Local Minima?



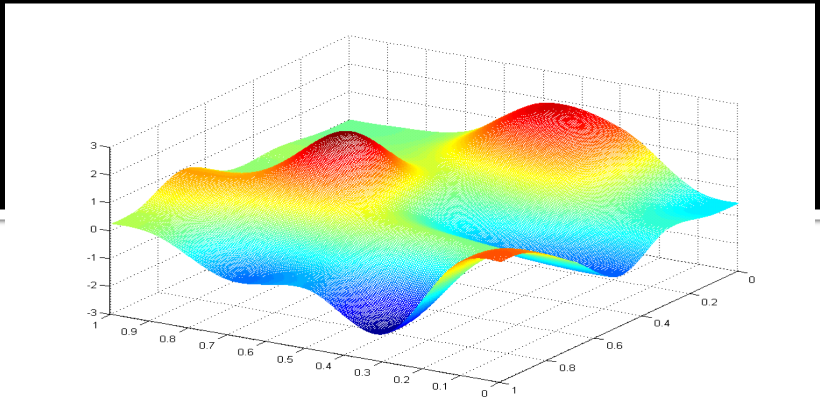
- What about local minima in gradient descent for regression?



- No problem!
- Cost function in regression is **convex**
  - Convex: second derivative is non-negative
  - More intuitively: a continuous function where the midpoint of any interval doesn't exceed the mean of the endpoints



Local Minima?



- What if cost function is not convex?

- Several options
  - Use multiple initialization points
  - Or use "smart" starting points (e.g., Xavier, He initialization)
  - Momentum can help
    - E.g., Nesterov Accelerated Gradient (NAG), RMSprop, Adam
  - Adaptive learning rates can help

Stopping conditions

```
Choose an initial vector of parameters α, β
Choose learning rate R
Repeat until convergence (i.e., until an approximate minimum is obtained):
```

$$\alpha \leftarrow \alpha - R\frac{\partial}{\partial\alpha}J(\alpha,\beta)$$

$$\beta \leftarrow \beta - R\frac{\partial}{\partial\beta}J(\alpha,\beta)$$

- How to know a minimum has been obtained?
  - Look for small changes in the gradient
  - Look for small improvements in cost
  - Look for no changes in parameters
  - Set a stopping condition!

Outline

- Cost functions
- Gradient descent
  - Local Minima
  - **Batch and Incremental versions**
  - Other issues
- Feature scaling

Incremental vs. Batch Gradient Descent

- In "Batch" gradient descent

```
Repeat until convergence:
```

Compute $\nabla\alpha = \frac{\partial}{\partial\alpha}J(\alpha,\beta) = \frac{1}{N}\sum_{i=1}^{N}(\alpha + \beta X_i - Y_i)$

Compute $\nabla\beta = \frac{\partial}{\partial\beta}J(\alpha,\beta) = \frac{1}{N}\sum_{i=1}^{N}(\alpha + \beta X_i - Y_i)X_i$

Global gradient, computed over all training data

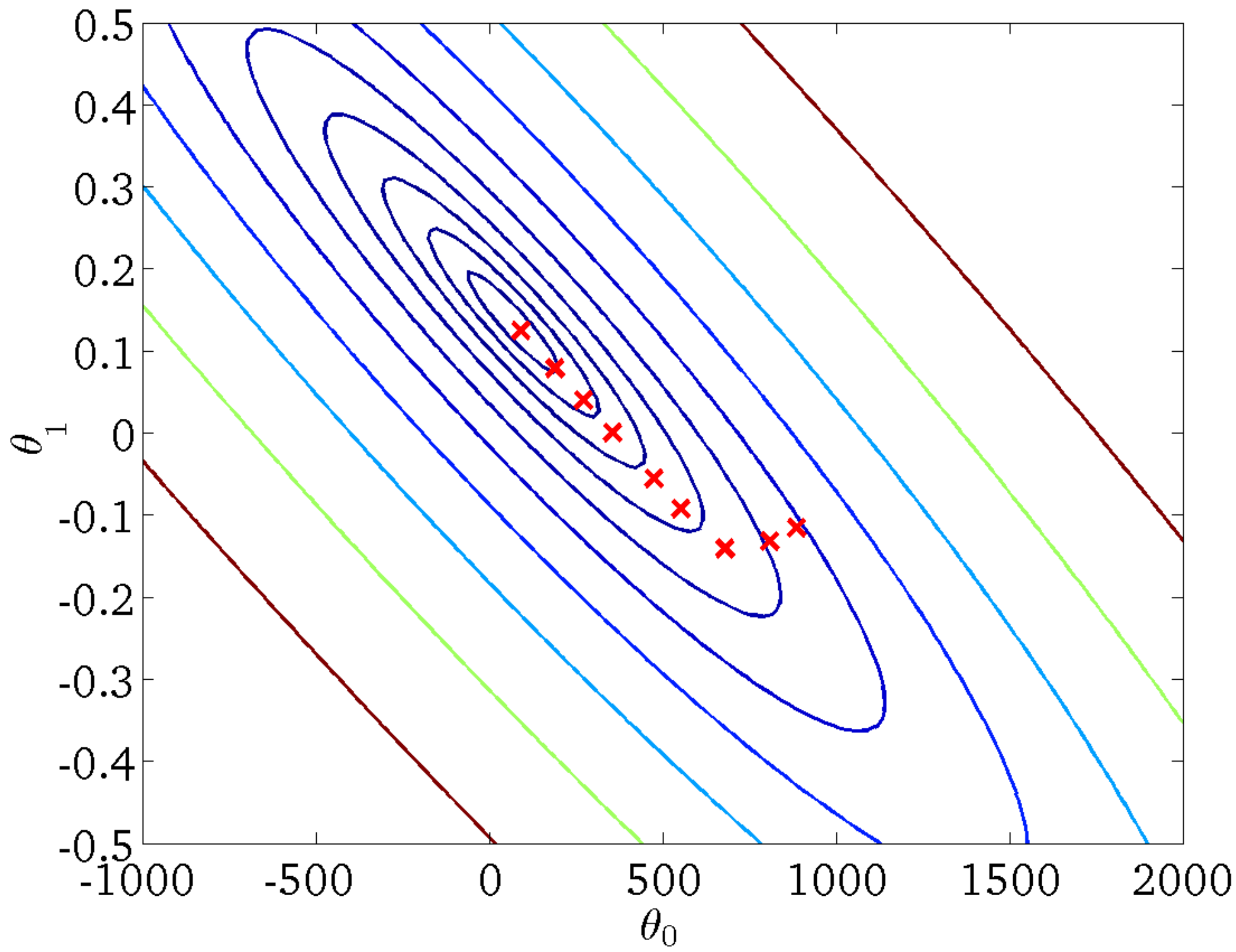
$\alpha \leftarrow \alpha - R\nabla\alpha$

$\beta \leftarrow \beta - R\nabla\beta$

Single, simultaneous update

- Note: each step uses all training examples!

Batch Gradient Descent



Incremental vs. Batch Gradient Descent

- "Iterative" (stochastic) version of gradient descent:

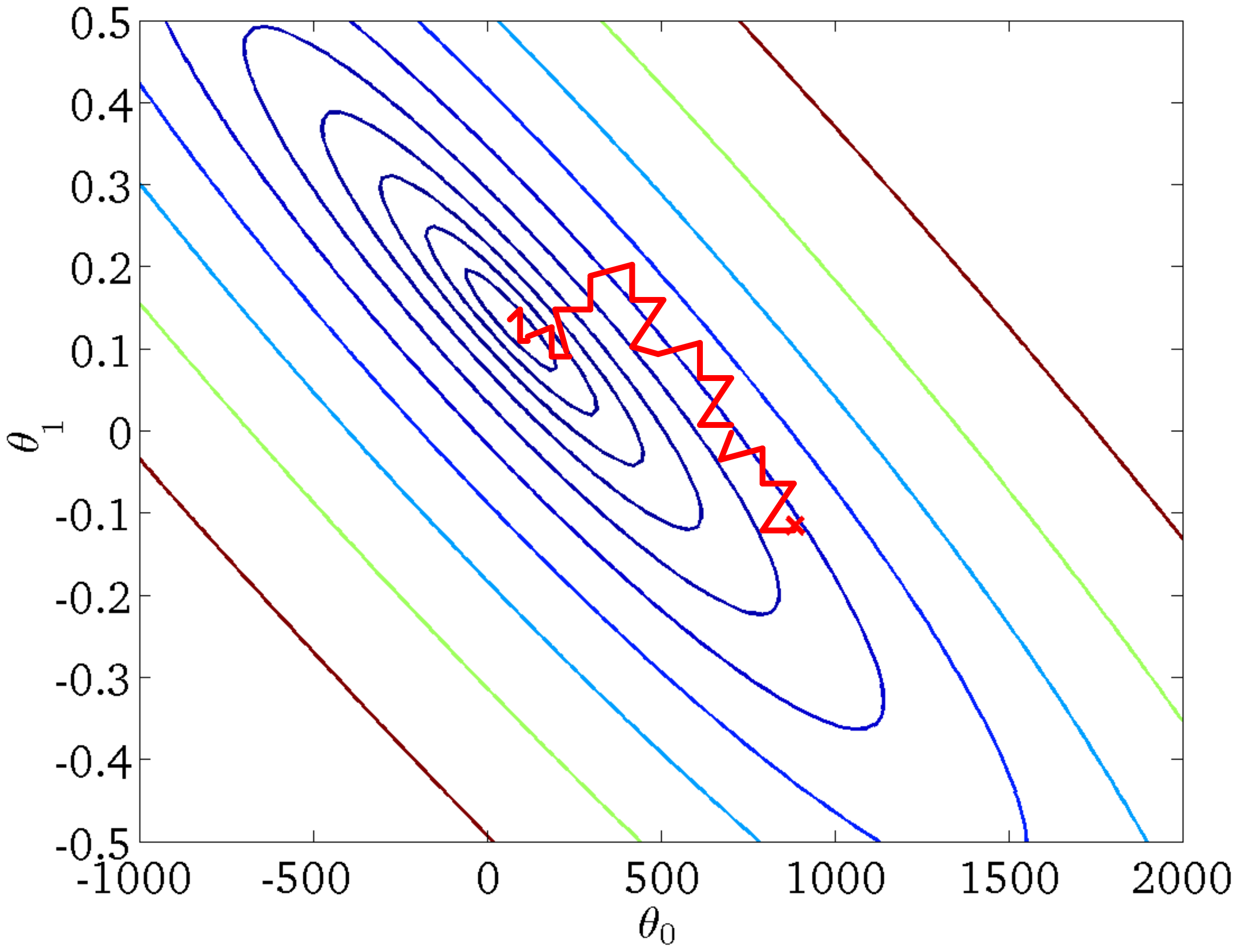
```
Repeat until an approximate minimum is obtained:
    Randomly shuffle examples in the training set
    For each example i:
```

$\alpha \leftarrow \alpha - R\frac{\partial}{\partial\alpha}J(\alpha,\beta)$ // evaluate $\frac{\partial}{\partial\alpha}J(\alpha,\beta)$ at x_i and update α

$\beta \leftarrow \beta - R\frac{\partial}{\partial\beta}J(\alpha,\beta)$ // evaluate $\frac{\partial}{\partial\alpha}J(\alpha,\beta)$ at x_i and update β

- The parameters are adjusted with each training instance, iteratively
- Also: Mini-batch

Stochastic Gradient Descent



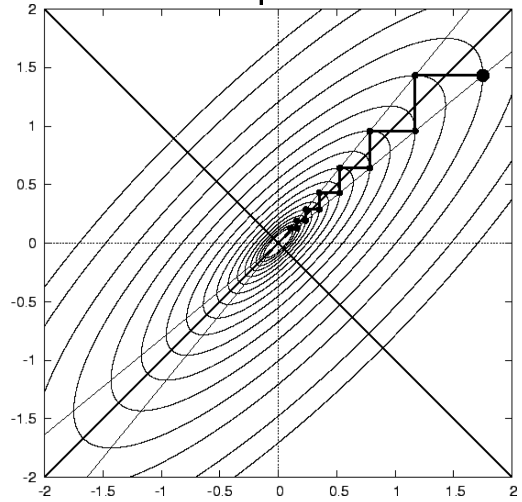
Outline

- Cost functions
- Gradient descent
- **Feature scaling**

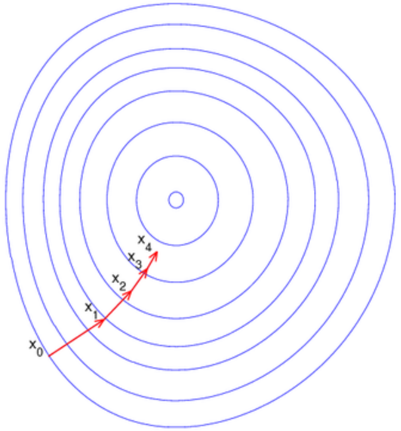
Feature scaling

- In gradient descent, we "take a step" in the direction where the decrease in cost is greatest
- When some features (axes) are on different scales, gradient descent can be inefficient
  - Putting different features on same scale can make gradient descent much faster

Not Proportional



Proportional



Feature scaling

- Feature scaling is an important pre-processing step for many common machine learning algorithms
  - Standardization: (mean 0 standard dev. 1)

$$x'_{ik} = \frac{x_{ik} - \bar{x}_k}{s_k}$$

    - s_k is typically standard deviation of x_k, or range (max-min) of x_k
  - Also common: force feature to be roughly between -1 and 1:

$$x'_{ik} = \frac{x_{ik}}{\max(|x_k|)}$$

  - Note: normalization parameters should be learned **on training data**

Key Concepts (today's lecture)

- Cost Functions
- Gradient Descent
- Local and global minima
- Convex functions
- Incremental vs. Batch GD
- Learning rates
- Feature scaling