Introduction to Statistical Machine Learning CSC/DSCC 265/465

Lecture 5: Supervised Learning – Part II

Cantay Caliskan

Notes and updates

Notes and updates

- The deadline for the 2nd Problem Set is **Friday, February 4, 11:59 PM!**
- Summer School opportunity
- Questions?

SICSS Summer School (May 9 – May 20)

- **Summer Institute in Computational Social Sciences** ***(May 9 – May 20)*** at the University of Rochester

- Opportunity to learn, to network, and to listen to exciting speakers, but most importantly: to finish a CSS project in a team (and there is no tuition)

- Topics: DL, ML, image as data, NLP, data visualization, network analysis with applications in CSS

- Adv. UGs interested in PhD, Master's students, PhD students, postdoctoral students (from different fields) are welcome!

- Check the link for more information and application (**Deadline: March 18**): https://sicss.io/2022/rochester/

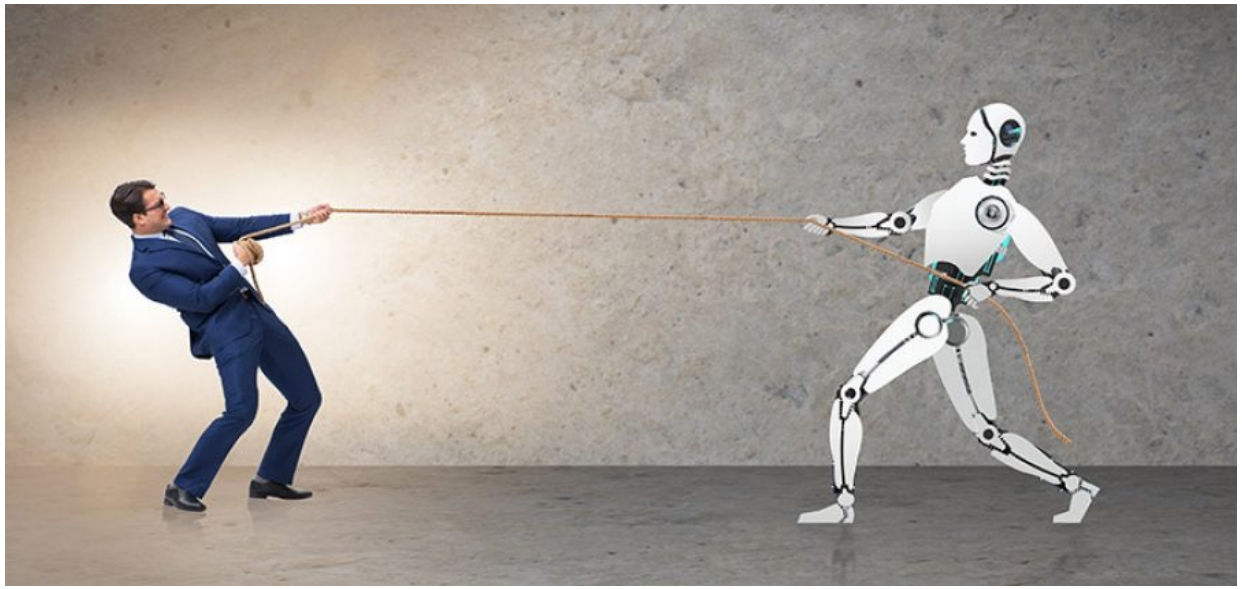
Plan for today

- ***Supervised Learning: Multivariate linear regression***
- **Direct Solution for Linear Regression**
- ***Gradient Descent Solution for Linear Regression***
- ***Maximum Likelihood for Linear Regression***

Plan for today

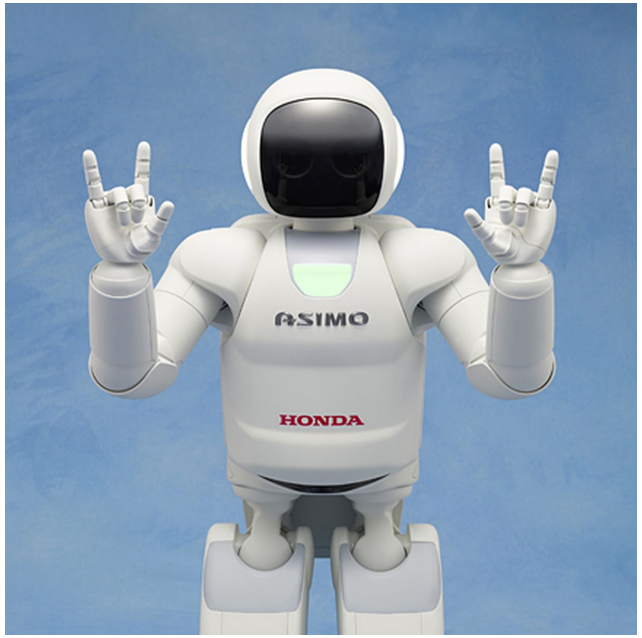
- ***Supervised Learning: Multivariate linear regression***
- **Direct Solution for Linear Regression**
- ***Gradient Descent Solution for Linear Regression***
- ***Maximum Likelihood for Linear Regression***

Supervised Learning



Supervised Learning

- Provide a ***training set*** to your algorithm
  - Your set needs to have ***features*** and ***labels***
- Your training performance is tested with a ***test set***
- Outputs can be:
  - Categorical (***classification***)
  - Continuous (***regression***)



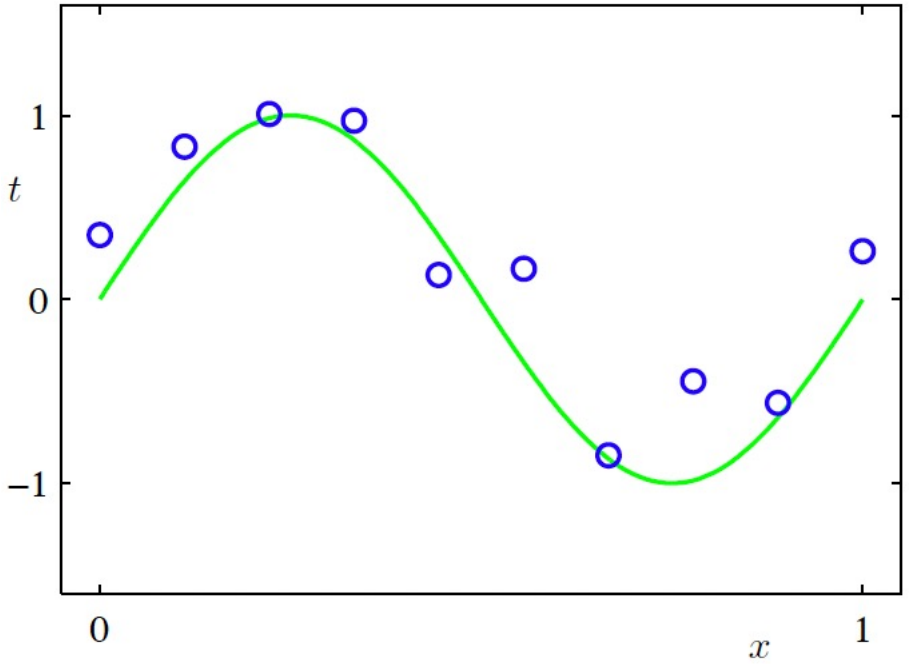
Examples: Supervised Learning

- **Recognize digits**
  - MNIST dataset!
  - What is input, what is output?

- **Predict the future prices of Tesla stock**
  - Yahoo Finance!
  - What is input, what is output?

Curve Fitting

- A training dataset with **N = 10** (samples) shown as **blue circles**
  - Each have an input (or series of input) ***x***
  - And a target variable ***t***



Goal: Predict the value ***t*** for some new ***x***

Basic form:

- $y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + w_1 x_1 + \ldots + w_d x_d$ ⟸ **d : Dimension**
- $x = (x_1, \ldots, x_d)^T$ is the ***input vector***
- $w = (w_1, \ldots, w_d)^T$ is the ***weight vector*** (parameters to be estimated)

Linear Regression

- More general form of linear regression:

$$y(\boldsymbol{x}, \boldsymbol{w}) = w_0 + \sum_{j=1}^{M-1} w_j \, \phi_j(\boldsymbol{x})$$

- where $\phi_j(\boldsymbol{x})$'s are called basis functions

- Parameter w_0 is called a ***basis parameter*** or a ***constant term***.
- If we add $\phi_0(\boldsymbol{x}) = 1$, we get:

$$y(\boldsymbol{x}, \boldsymbol{w}) = \sum_{j=0}^{M-1} \boldsymbol{w}^T \, \phi_j(\boldsymbol{x})$$

Quick Note: Basis Functions

- **Polynomial functions:**

$$\phi_j(x) = x^j$$

- **Gaussian functions:**

$$\phi_j(\mathbf{x}) = \exp\left\{-\frac{(\mathbf{x}-\boldsymbol{\mu}_j)^T(\mathbf{x}-\boldsymbol{\mu}_j)}{2s^2}\right\}$$

- **Sigmoid basis functions:**

$$\phi_j(\mathbf{x}) = \sigma\left(\frac{\mathbf{b}^T\mathbf{x}-\mu_j}{s}\right)$$

- **Wavelets:**
  - Continuous wavelet transform, fractional Fourier transform etc.

Question: Why do we need different basis functions?

Multidimensional Data

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Notation:

n = Number of observations

k = Number of features (number of independent variables)

$x^{(i)}$ = Input features of the i^{th} training example

$x_j^{(i)}$ = Value of feature j in the i^{th} training example

Cost Function

- Cost function for linear regression:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{i=1}^{m} \left(h_\theta(x^{(i)}) - y^{(i)} \right)^2$$

- Goal: Finding the lowest cost possible
  - Lowest cost gives you the best set of parameters

- **Question: What is this cost function?**
  - Sum of Squared Errors

- **Question: How can we estimate the lowest cost?**
  - Direct Minimization
  - Gradient Descent

Cost Minimization

- **Goal:**

$$\min_{\theta} J(\theta; x^{(1)}, y^{(1)}, \ldots, x^{(m)}, y^{(m)})$$

- **Direct minimization:**
  - Take the first derivative set to zero
  - Take the second derivative, check if min. or max.
  - Not possible for the 'most interesting' cost functions
    - Question: Why?

- **Gradient descent:**
  - Start with a guess for θ
  - Change θ until $J(\theta)$ decreases
  - Repeat until you reach a minimum
    - Question: Does this always work for every supervised algorithm?

Plan for today

- *Supervised Learning: Multivariate linear regression*
- **Direct Solution for Linear Regression**
- *Gradient Descent Solution for Linear Regression*
- *Maximum Likelihood for Linear Regression*

Direct minimization

Minimizing SSE

- **Minimize:**

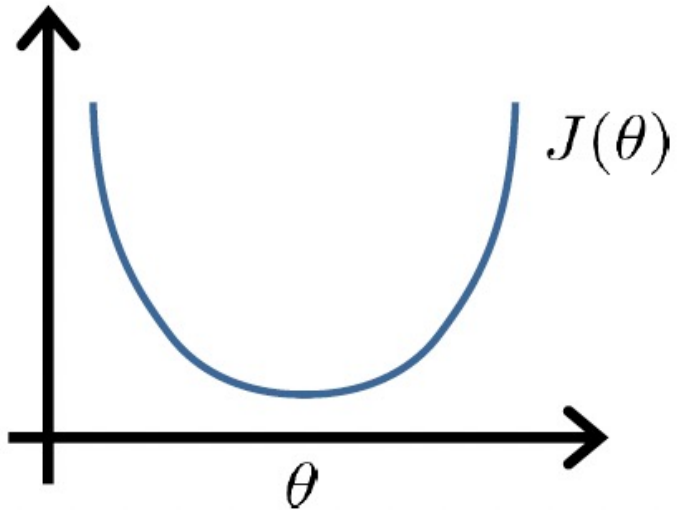
$$J(\theta_0, \theta_1, \ldots, \theta_m) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

- **Find minima of the following function:**

$$\theta \in \mathbb{R}^{n+1}$$

$$\frac{\partial}{\partial \theta_j} J(\theta) = \cdots = 0 \quad \text{(for every } j\text{)}$$

Solve for $\theta_0, \theta_1, \ldots, \theta_n$



This is a **convex** function

Minimizing SSE

- **Re-write SSE using vector-matrix notation:**

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

- **Where:**

$$X = \begin{bmatrix} — & (x^{(1)})^T & — \\ — & (x^{(2)})^T & — \\ & \vdots & \\ — & (x^{(m)})^T & — \end{bmatrix} \quad \vec{y} = \begin{bmatrix} y^{(1)} \\ y^{(2)} \\ \vdots \\ y^{(m)} \end{bmatrix}$$

What is ***X*** and ***y***?

Remember: Our multidimensional dataset

$$X = \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \quad y = \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

- **Solution:**

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

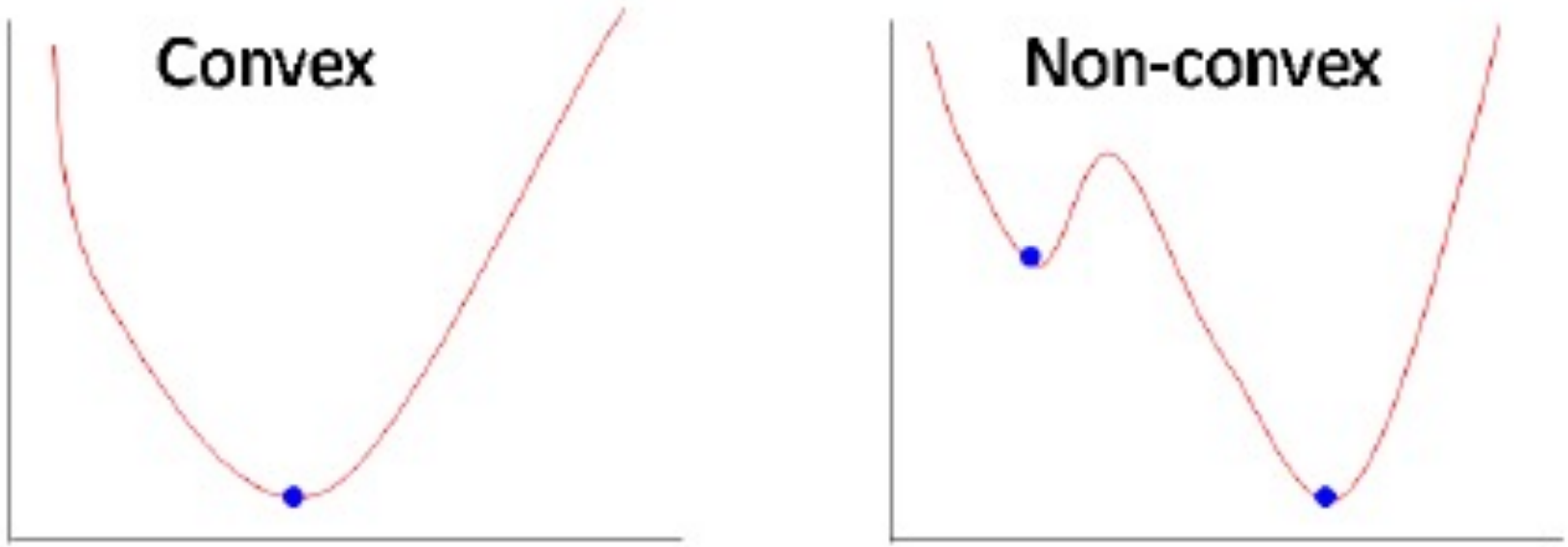
Beware: We need non-singular matrices

Plan for today

- *Supervised Learning: Multivariate linear regression*
- Direct Solution for Linear Regression
- ***Gradient Descent Solution for Linear Regression***
- *Maximum Likelihood for Linear Regression*

Gradient Descent

Reminder: Convex and non-convex functions



- The ***loss functions*** of some models can be shown as convex functions
  - Example: Linear regression
- The ***loss functions*** of other models are non-convex
  - Example: Neural Networks

What is a gradient?

- A gradient measures how much the output of a function changes if you change the inputs ‘a little bit’

- Definition: A vector-valued function that represents the slope of the tangent of the function graph, pointing the direction of the greatest rate of increase in the function

What is gradient descent?

Another definition:

An **optimization** technique used for computing the model parameters (coefficients and bias) for algorithms like linear regression, logistic regression, neural networks etc.

- One of the most important concepts used in the training of **non-linear** clustering/classification algorithms:
  - Examples: Logistic regression, support vector machines, neural networks…

What does it (practically) do?

Using *gradient descent*, we repeatedly iterate through the training set and update the model parameters in accordance with the gradient of error with respect to the training test

Mathematical Intuition

- Suppose, we have some loss function that has some parameters and that we want to minimize:

$$J(\Theta_0, \Theta_1 \ldots \ldots \Theta_n)$$

- Example: In case of 'simple' linear regression, we only have two parameters:

$$J(\Theta_0, \Theta_1)$$

- We start with some random initial values, for instance $\Theta_0 = 0, \Theta_1 = 0$
- We will keep changing the parameters until we minimize the error

- Depending on where we start, we may find different global minima!

Mathematical Intuition

- We "simultaneously update" our parameters until convergence:

$$\Theta_j = \Theta_j - \alpha \frac{d}{d\Theta_j} J(\Theta_0 \ldots\ldots \Theta_n)$$

- Thus, if we have <u>two parameters</u> only:

$$\Theta_0 = \Theta_0 - \alpha \frac{d}{d\Theta_0} J(\Theta_0, \Theta_1)$$

$$\Theta_1 = \Theta_1 - \alpha \frac{d}{d\Theta_1} J(\Theta_0, \Theta_1)$$

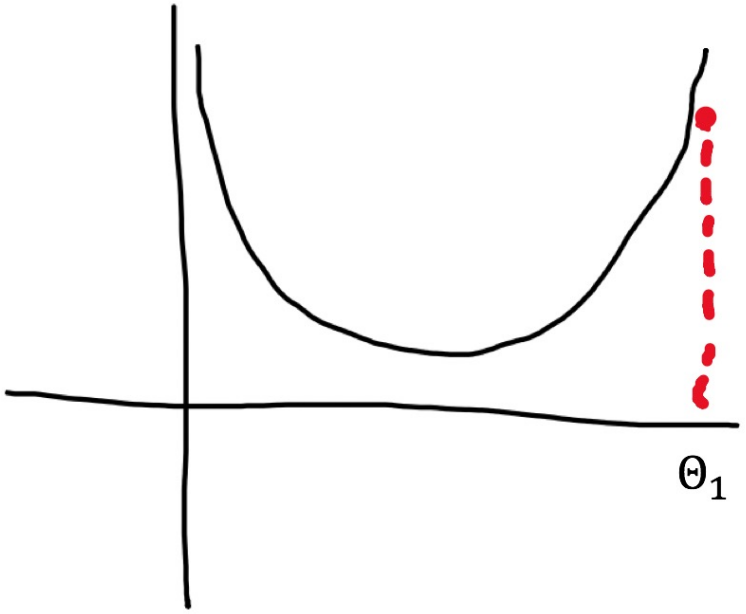
<u>Question</u>: How many parameters can we have?

Alpha (α) is the **learning rate**

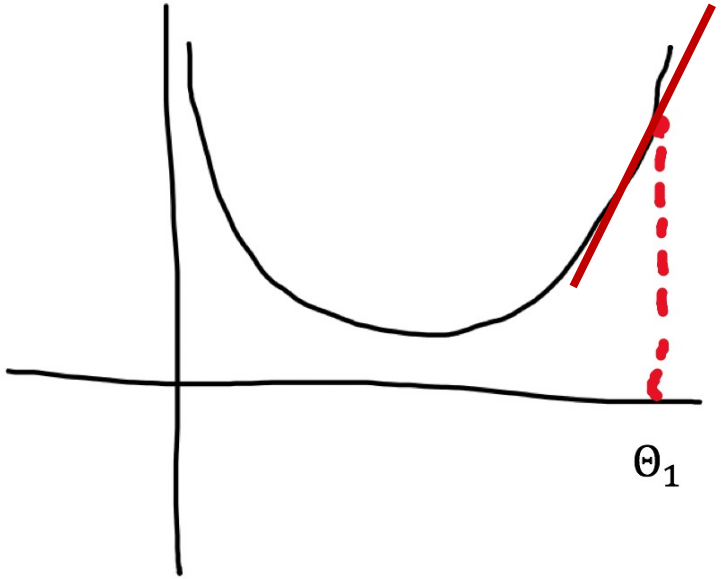
$\frac{d}{d\Theta_1} J(\Theta_0, \Theta_1)$ is the derivative (or the **gradient**)

Mathematical Intuition

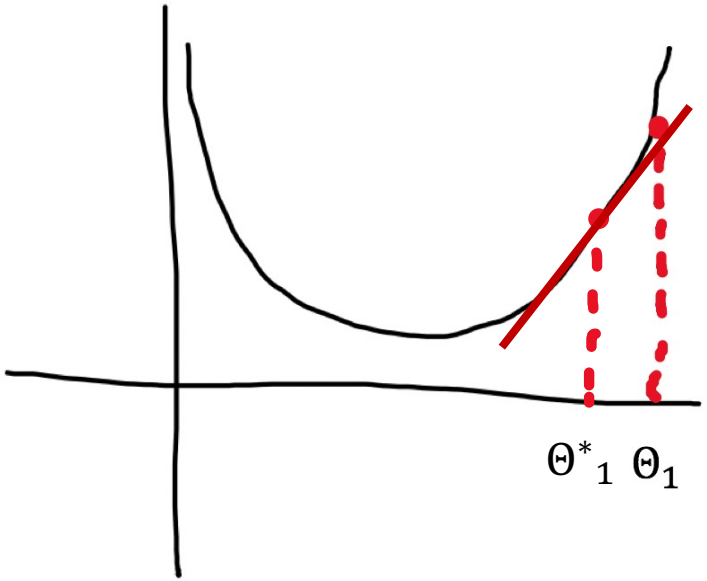
Let's say we have a ***quadratic function*** that looks like:



If we take the derivative at point Θ_1, we get a slope:



So, the descent will look like:

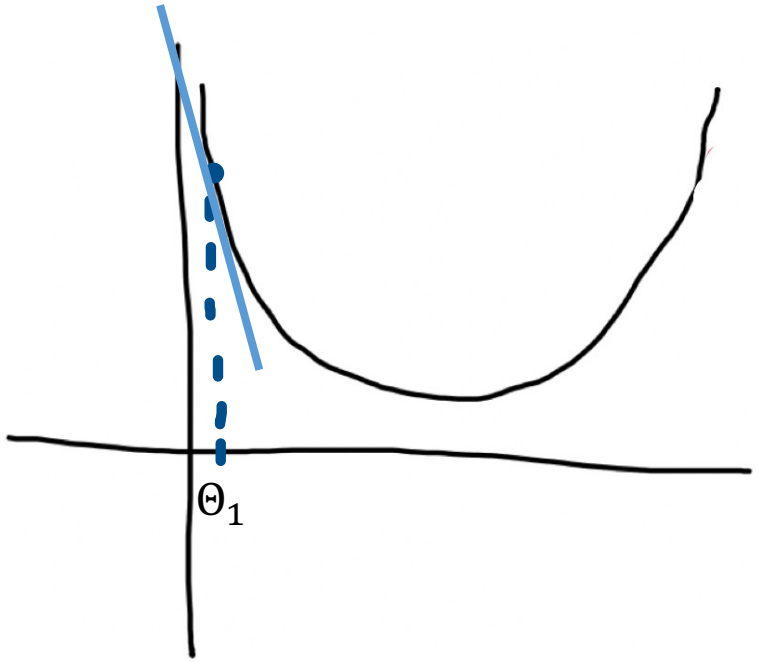


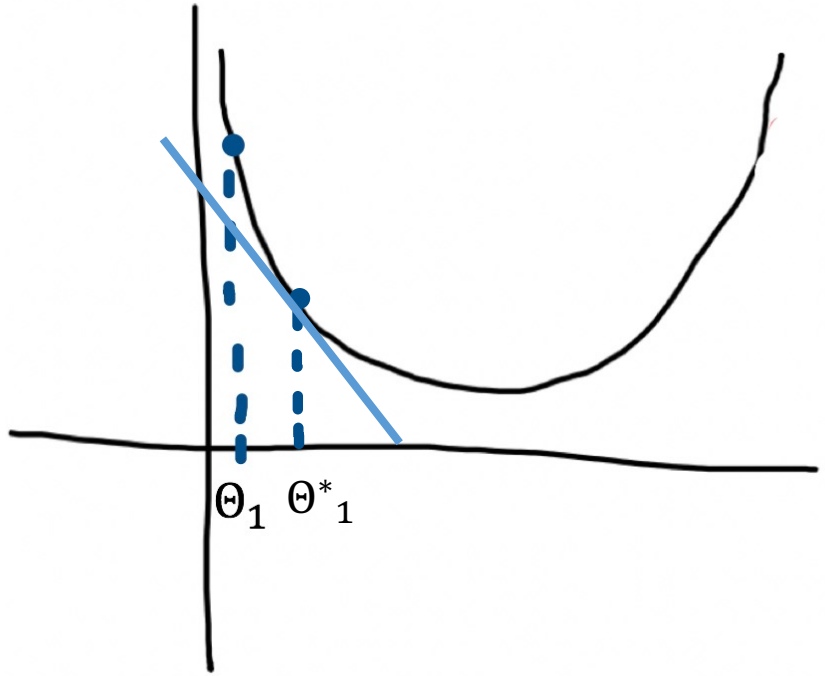
This line has a positive slope, so it has a positive derivative: $\Theta_1 = \Theta_1 - \alpha \frac{d}{d\Theta_1} J(\Theta_1)$

where the red part of the equation to left is positive

Mathematical Intuition

What happens, if we approach from the left:



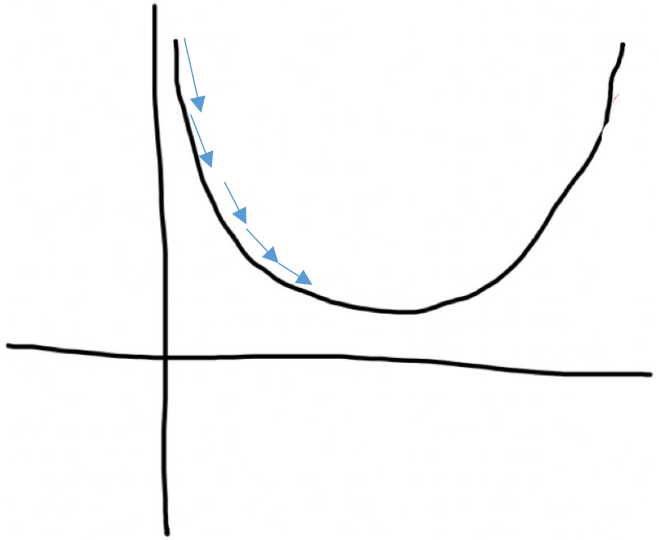


This line has a negative slope, so it has a negative derivative: $\Theta_1 = \Theta_1 - \alpha \frac{d}{d\Theta_1} J(\Theta_1)$

where the blue part of the equation to left is negative

Learning rate

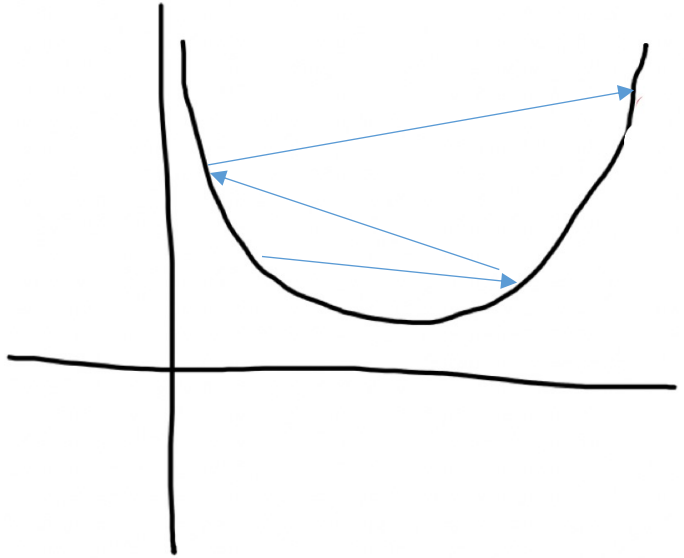
What if ***learning rate*** is <u>too small</u>?



Vanishing gradient problem

The size/speed of updates are becoming smaller at every iteration -> the sizes of the arrows are becoming smaller. **Convergence takes a lot of time.**

What if ***learning rate*** is <u>too large</u>?



Exploding gradient problem

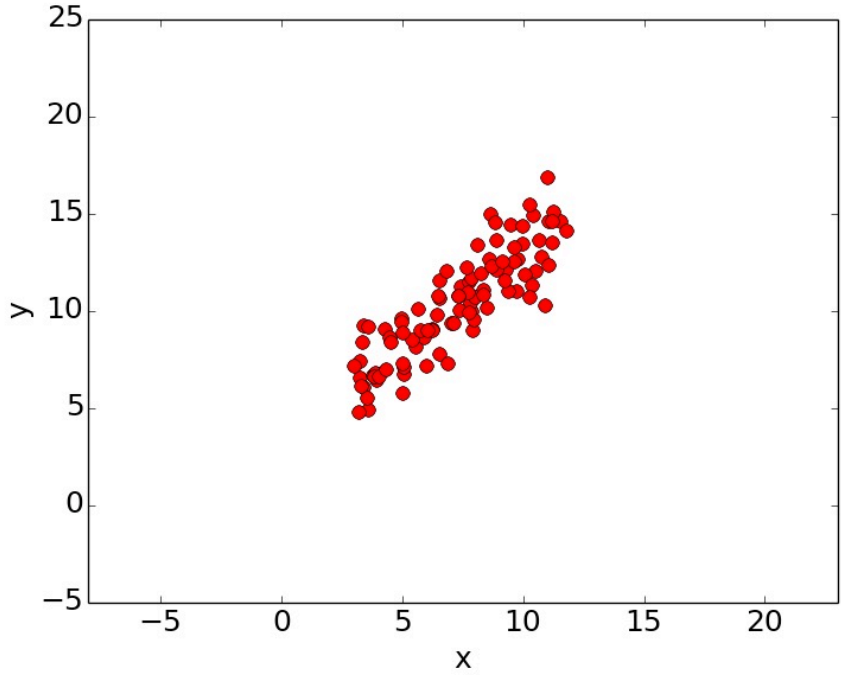
If the shape of the function allows it: the **algorithm converges**

If not: It **never converges**.

Example

Gradient descent in linear regression

Let's assume that we have the following data and we want to fit a regression line:

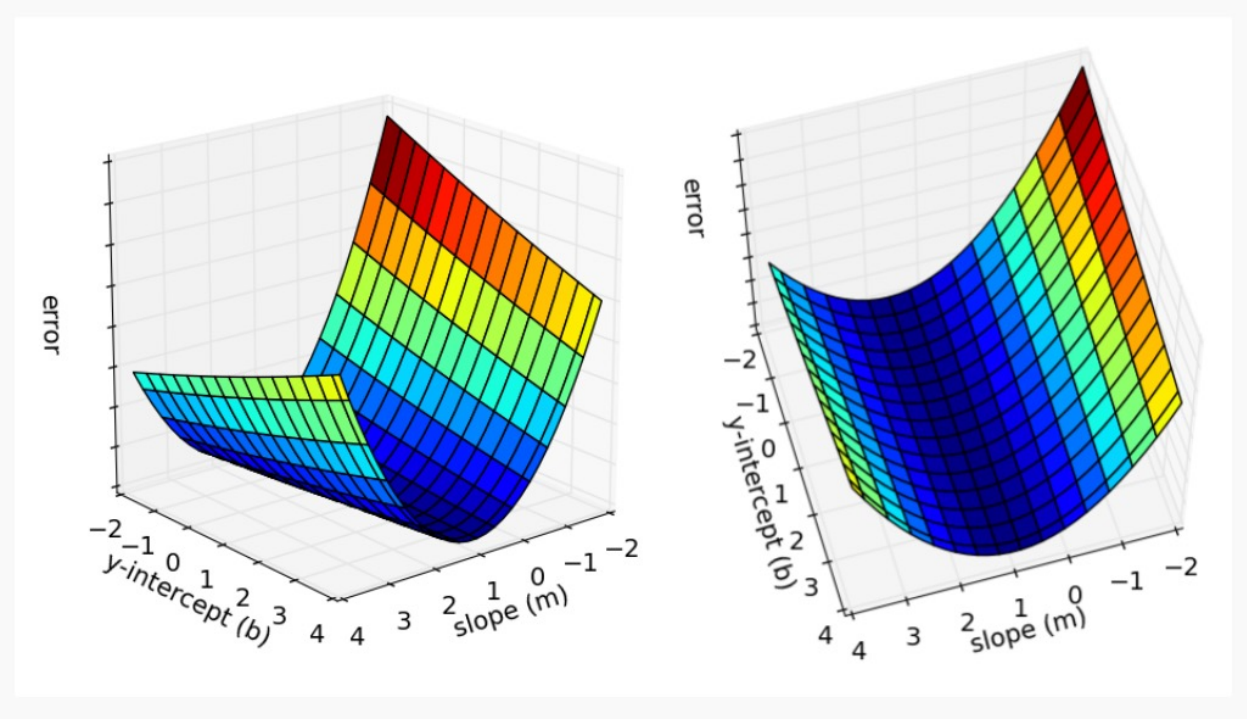


We need to set up a line **y = mx + b**

- So, we need to find the optimal **b** and the optimal **m**
- <u>Standard approach:</u>
  - Define an error function that measures how "good" a given line is
  - This function will take in potential values for (m, b) pair and find the the lowest SSE by calculating it

Gradient descent in linear regression

- Lines that fit the data better will result in lower error values
- If we minimize the SSE, we will get the best line for our data



- The height of the function at each point is the error value for that line

- For gradient descent:
  - Start from some location on this surface and move downhill to find the lowest error

Computing the **gradient**

- Gradient shows the direction for descent
  - But, we have two variables (*m* and *b*), what should we do?
- We need to differentiate our error function with regard to *m* and *b*
- So, we take the first derivative:

$$\frac{\partial}{\partial \mathrm{m}} = \frac{2}{N}\sum_{i=1}^{N} - x_i(y_i - (mx_i + b))$$

$$\frac{\partial}{\partial \mathrm{b}} = \frac{2}{N}\sum_{i=1}^{N} - (y_i - (mx_i + b))$$

- Now, we need to think about:
  - What will be our initial values for *m* and *b*?
  - Each iteration will update m and b to a line that yields a ***slightly lower error*** than the previous iteration.

Summary

Let's say we have a simple linear regression: $y_{pred} = mX + b$

Step 1: Initialize the weights (**m** and **b**) with random values and calculate SSE

Step 2: Calculate the gradient (or the change) at the initial values for **m** and **b**

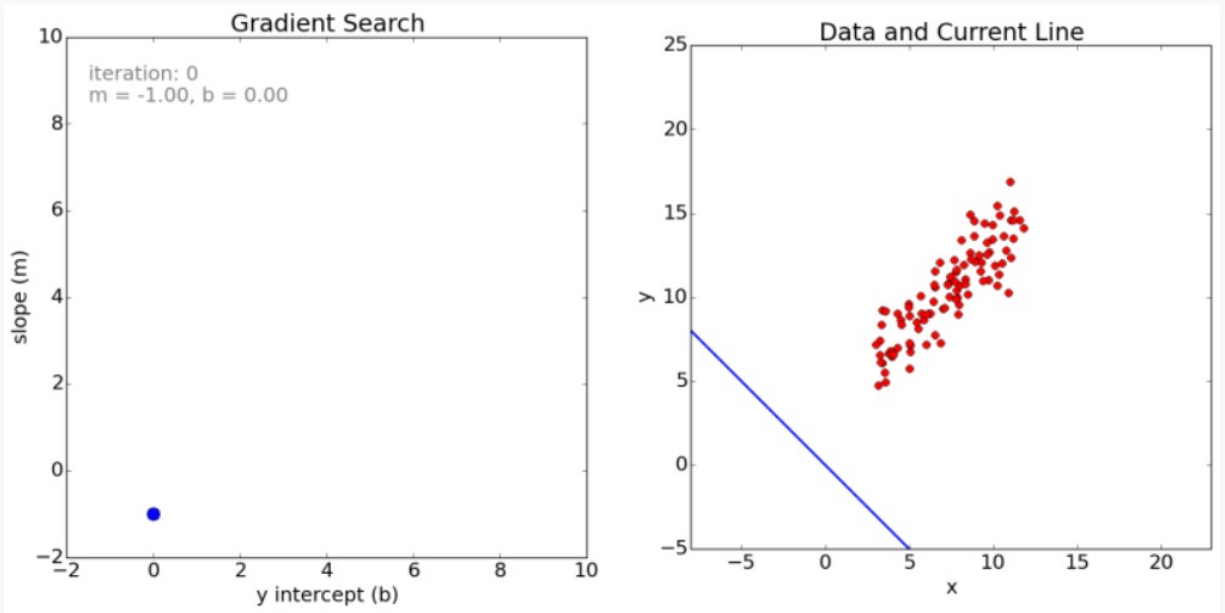
Step 3: Change the initial values of m and b by subtracting the product of learning rate and gradient for each parameter and adjust the weights with the gradients to reach the optimal values (goal: minimizing SSE)

Step 4: Use the new weights for prediction and to calculate the new SSE. If exists, compare the new SSE to the previous SSE

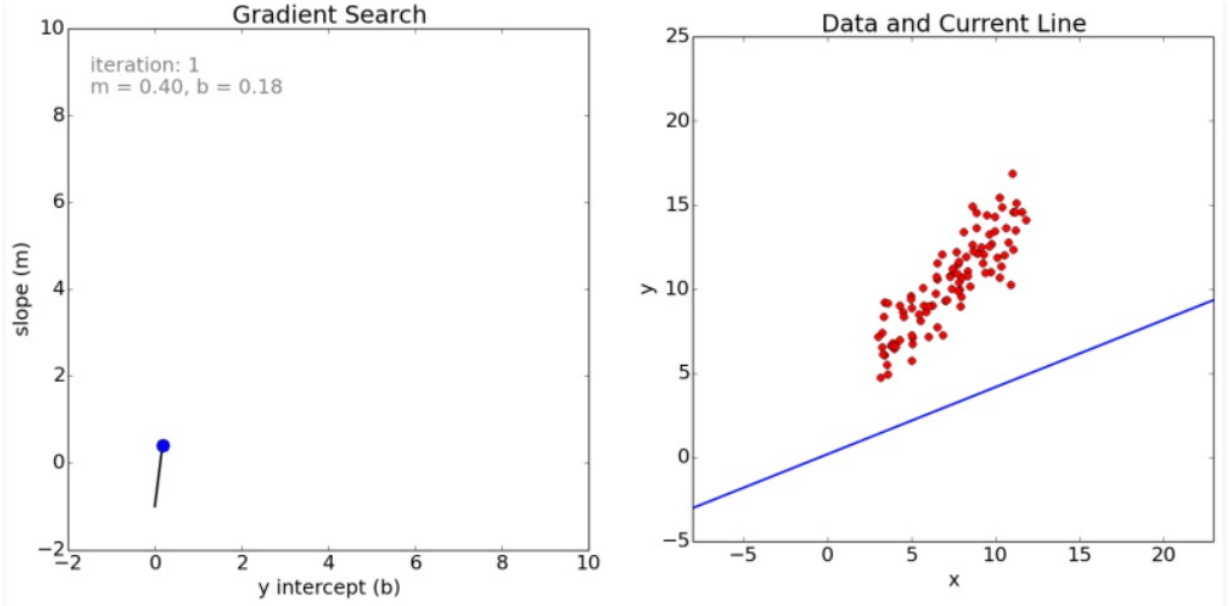
Step 5: Repeat Step 2 and Step 3 until further adjustments to weights does not significantly reduce the error

Example: Gradient Descent

- Let's run a gradient descent algorithm for linear regression
- Let's set the number of iterations to 2000
- Let's set *m = -1* and *b = 0*



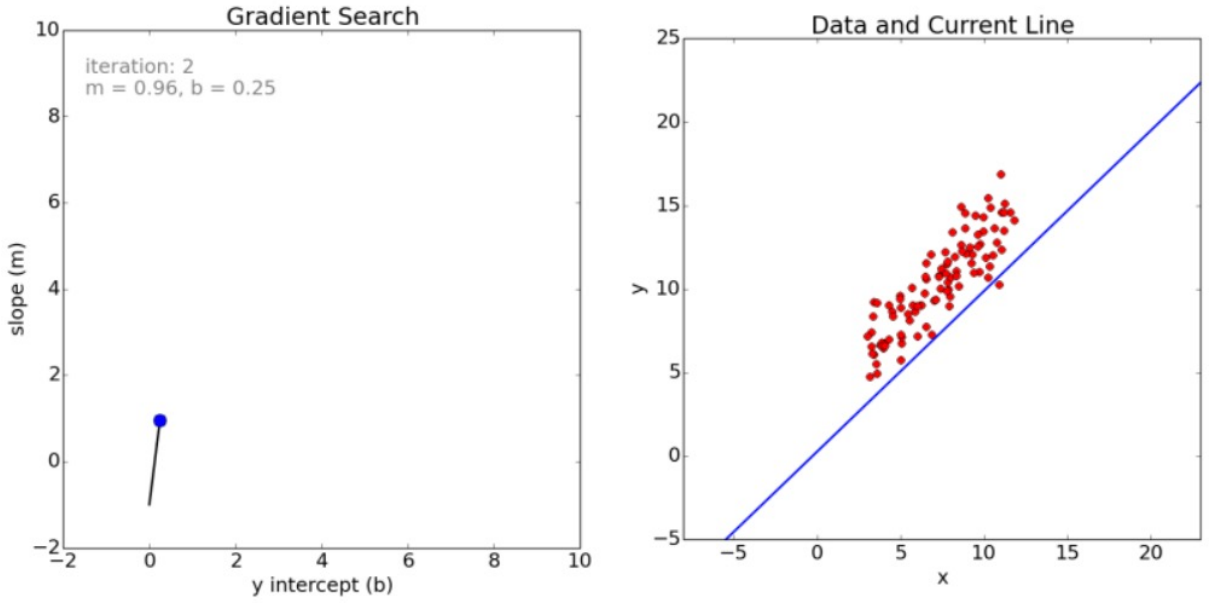
Iteration 0, *m = -1, b = 0*



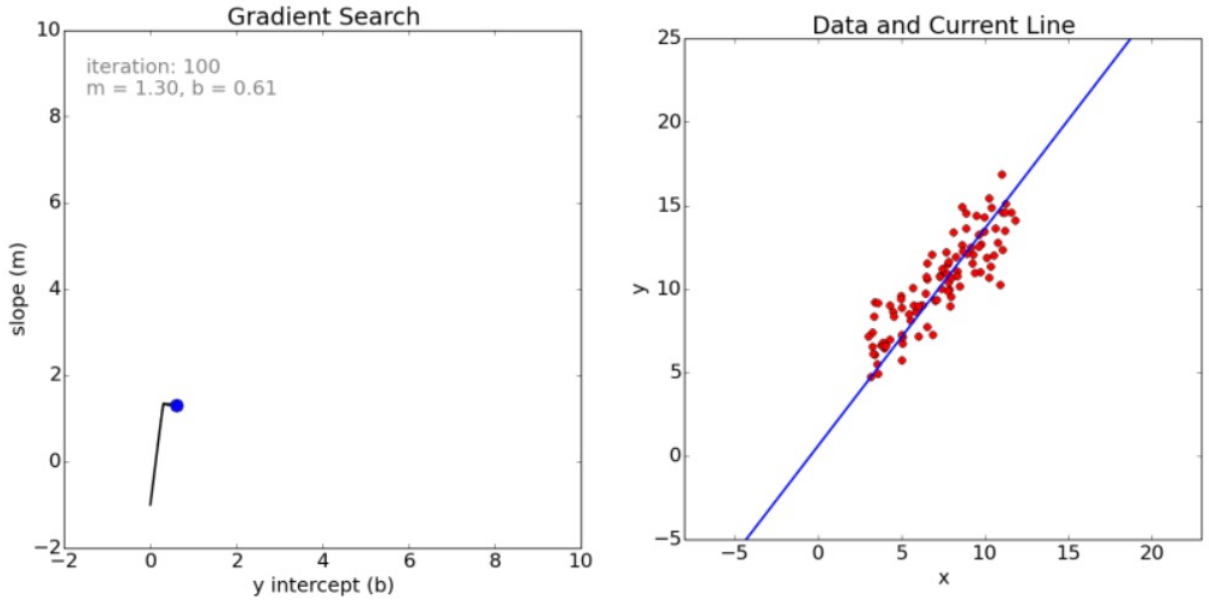
Iteration 1, *m = 0.4, b = 0.18*

Example: Gradient Descent

- The error becomes “a little smaller” with each iteration…



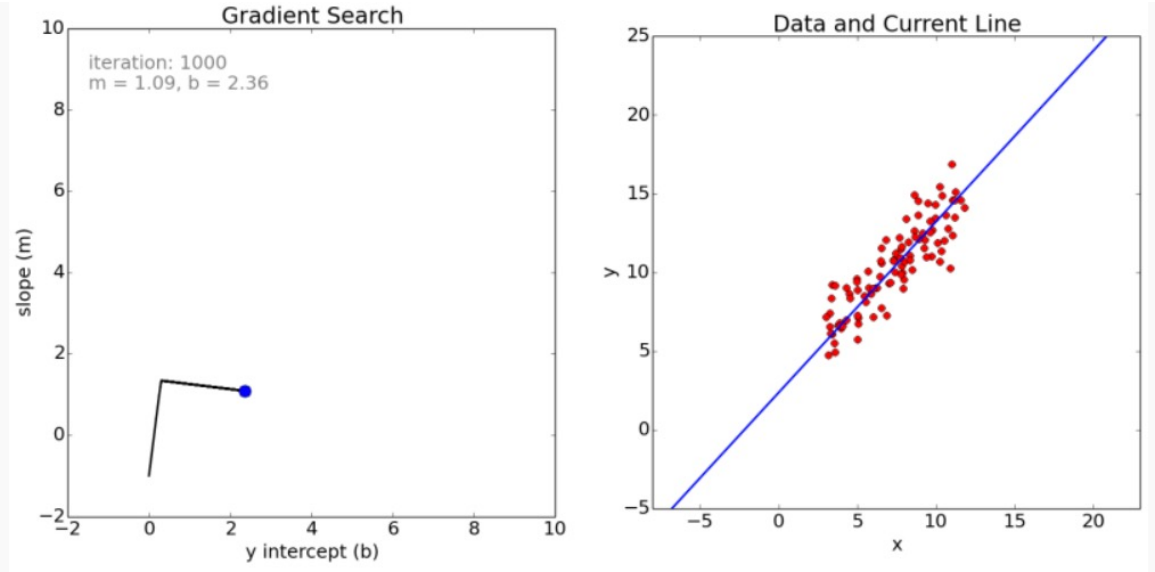
Iteration 2, *m = 0.96, b = 0.25*



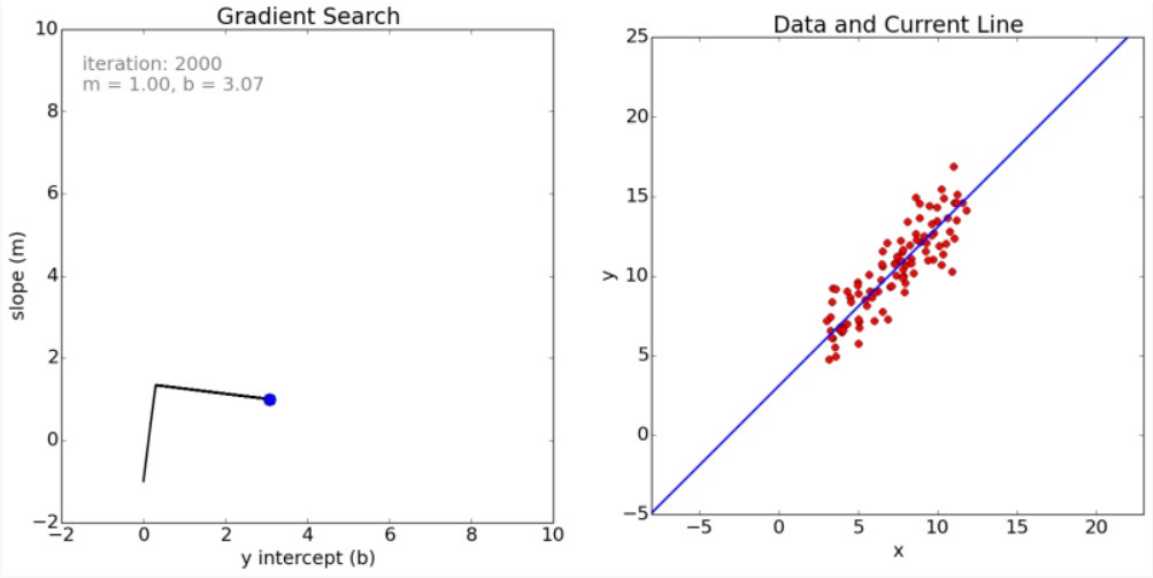
Iteration 100, *m = 1.30, b = 0.61*

Example: Gradient Descent

- And the differences between each iteration become smaller and smaller…



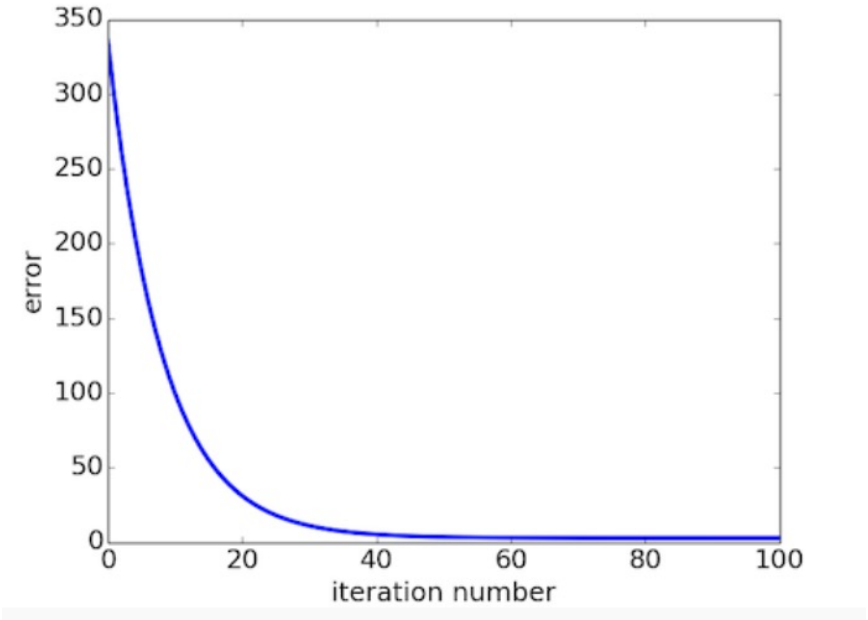
Iteration 1000, *m = 1.09, b = 0.25*



Iteration 2000, *m = 1.00, b = 3.07*

Convergence: Gradient Descent

- The difference between each iteration becomes smaller and smaller…
- You can set a ”convergence mechanism”. This is typically done by looking for small changes in error iteration

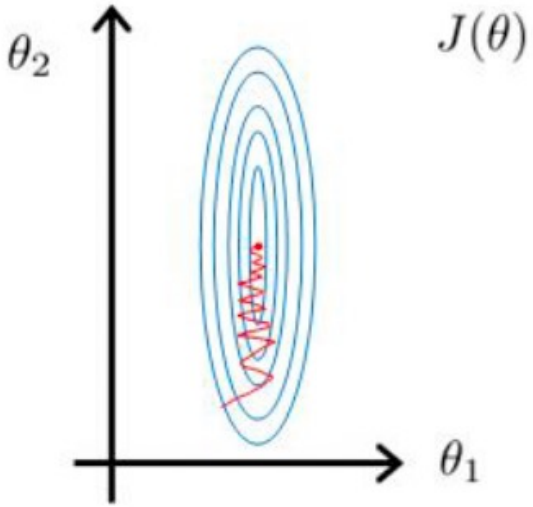


Reminder: Gradient Descent is a generalizable concept and an be applied to all ML and DL applications

Feature Normalization

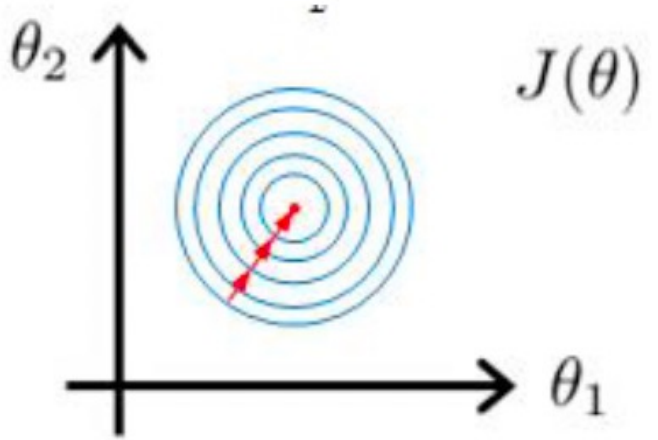
Important note:

- If features have very different scale, Gradient Descent can get "stuck" since x_j affects size of gradient in the direction j^{th} dimension
- Normalizing features to be zero-mean (μ) and same-variance (σ) helps gradient descent converge faster



normalize

$$x_j^{(i)} = \frac{x_j^{(i)} - \mu_j}{\sigma_j}$$



Please do the following until next lecture!

- Review what we have just gone through (supervised learning: linear regression)

- Review ***Chapter 4***

- Continue with your problem set (due date ***Friday, February 4, 11:59 PM***)