Machine Learning Department
School of Computer Science
Carnegie Mellon University

Linear Regression

Matt Gormley
Lecture 7
Feb. 6, 2023

Reminders

- **Homework 3: KNN, Perceptron, Lin.Reg.**
  - Out: Fri, Feb. 3
  - Due: Fri, Feb. 10 at 11:59pm
  - (only two grace/late days permitted)
- **Exam conflicts form**

Q&A

Q: I have a medical emergency or family emergency or disability or other compelling reason and am unable to attend office hours in-person this week. Can an exception be made so I can attend office hours remotely?

A: Yes. Please email the Education Associate(s) and request a period of remote office hours. We will reply with instructions on how to utilize them during the approved time period.

Q&A

Q: How do we build Decision Trees with real-valued features?

A: Great question! I made a 7 minute video about that.

Q: Is there a more formal statement of the Perceptron Mistake Bound?

A: Great question! I'm going to make a 5 minute video about that *and* we'll cover it in Recitation.

Q: How do we prove the Perceptron Mistake Bound?

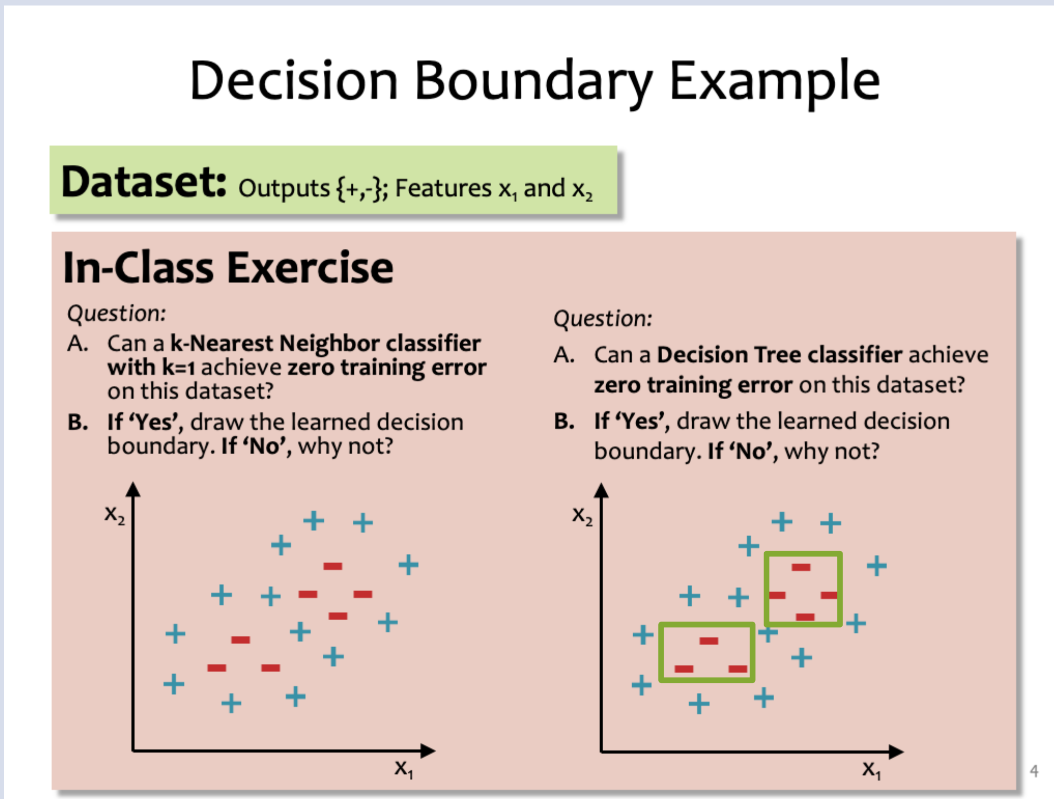
A: Great question! I'm going to make a 10 minute video about that.

DECISION TREES WITH REAL-VALUED FEATURES

Q&A

Q: How do we learn a Decision Tree with real-valued features?

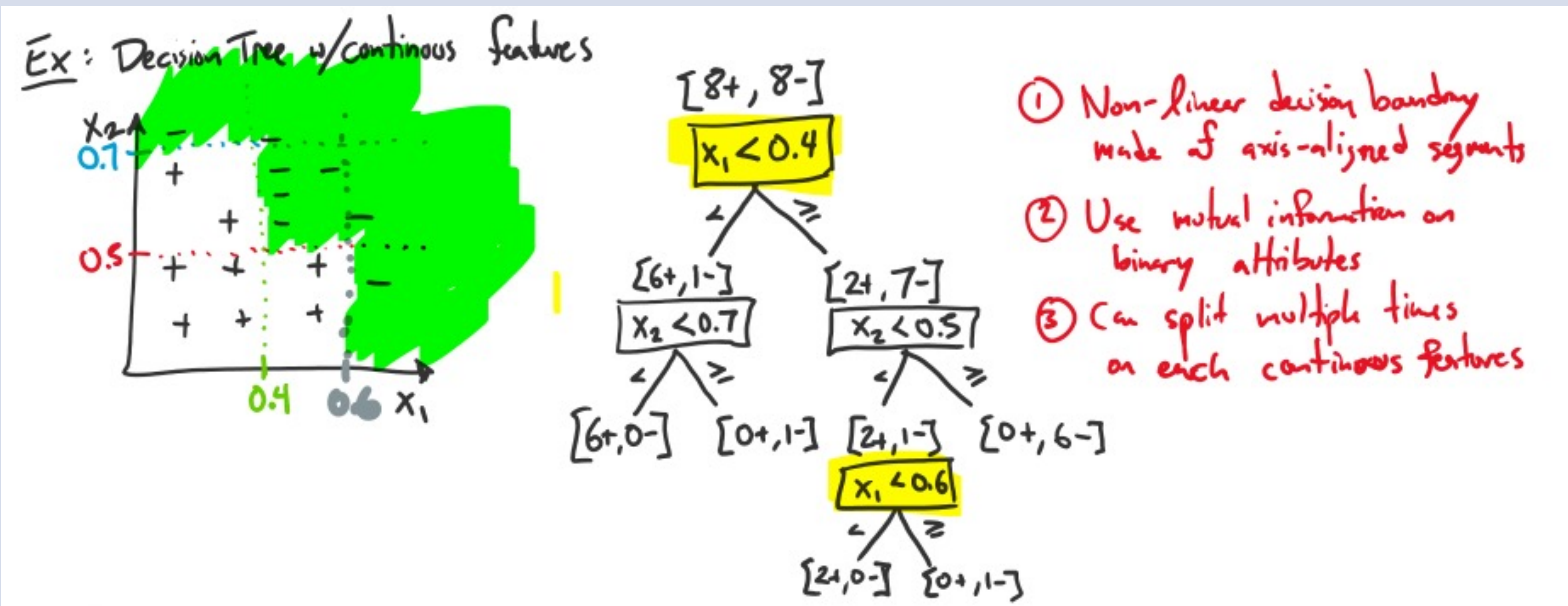
A:



Q&A

Q: How do we learn a Decision Tree with real-valued features?

A: Make new discrete features out of the real-valued features and then learn the Decision Tree as normal! Here's an example…



Perceptron Exercise

Question:

Unlike Decision Trees and K-Nearest Neighbors, the Perceptron algorithm ***does not suffer from overfitting*** *because it does not have any hyperparameters that could be over-tuned on the training data.*

A. *True*
B. *False*
C. *True and False*

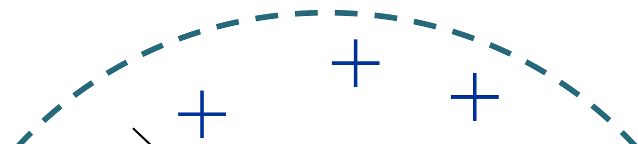
Answer:

PERCEPTRON MISTAKE BOUND

Perceptron Mistake Bound

Guarantee: if some data has margin γ and all points lie inside a ball of radius R rooted at the origin, then the online Perceptron algorithm makes $\leq (R/\gamma)^2$ mistakes

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes! The algorithm is invariant to scaling.)



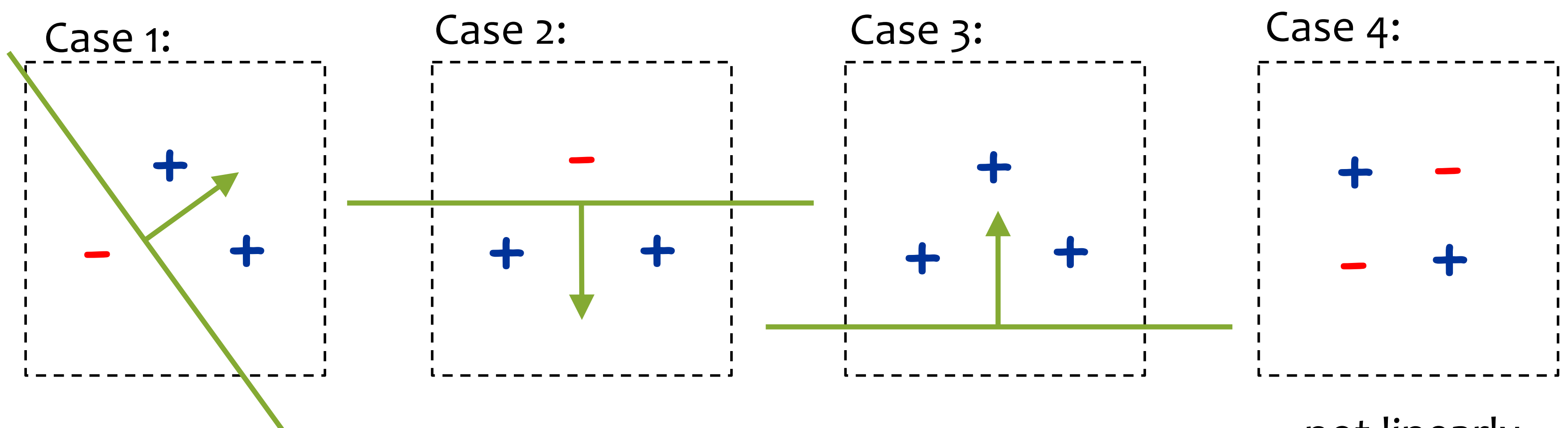
Def: We say that the (batch) perceptron algorithm has **converged** if it stops making mistakes on the training data (perfectly classifies the training data).

Main Takeaway: For **linearly separable** data, if the perceptron algorithm cycles repeatedly through the data, it will **converge** in a finite # of steps.

Slide adapted from Nina Balcan

Linear Separability

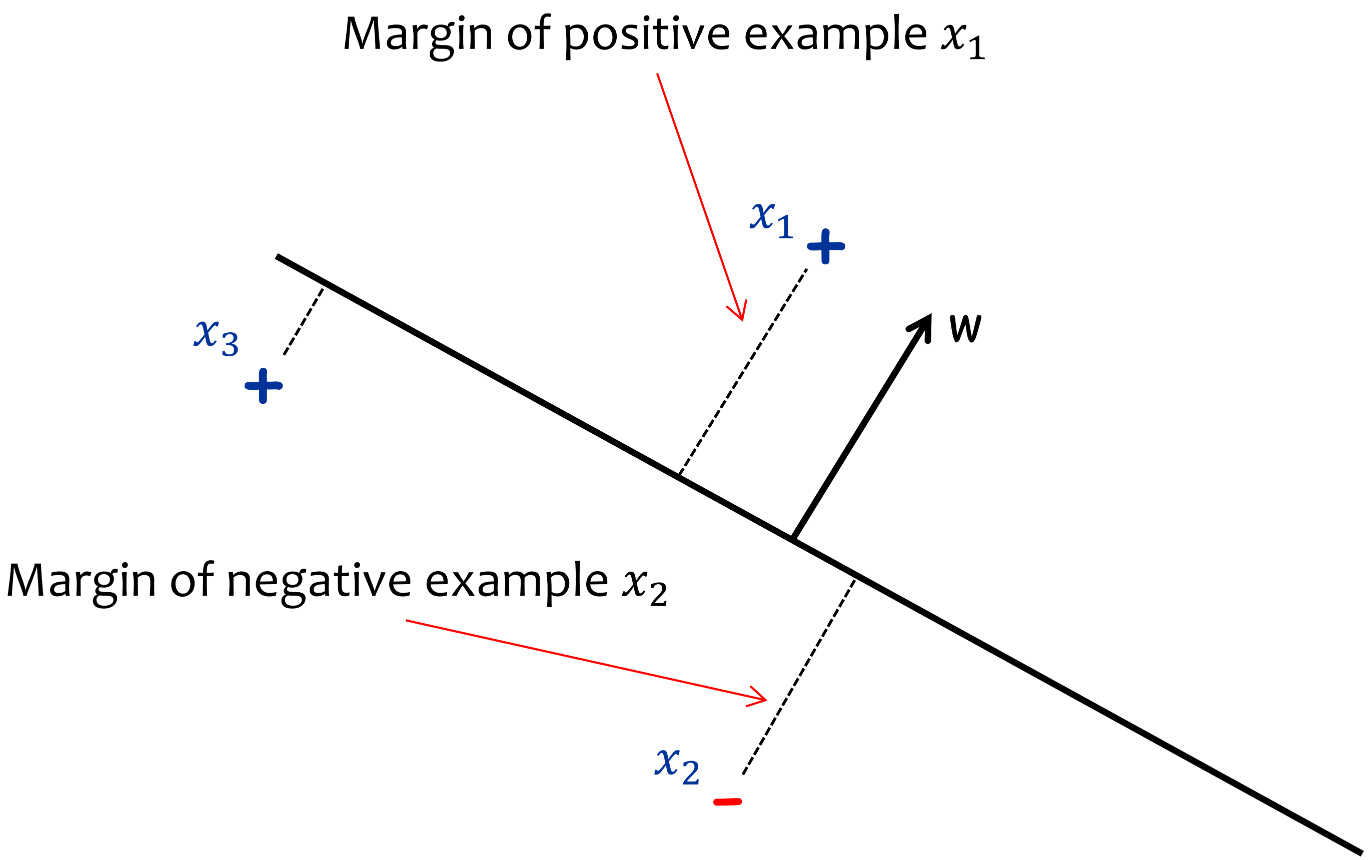
Def: For a **binary classification** problem, a set of examples S is **linearly separable** if there exists a linear decision boundary that can separate the points



not linearly separable

Geometric Margin

Definition: The margin of example x w.r.t. a linear separator w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

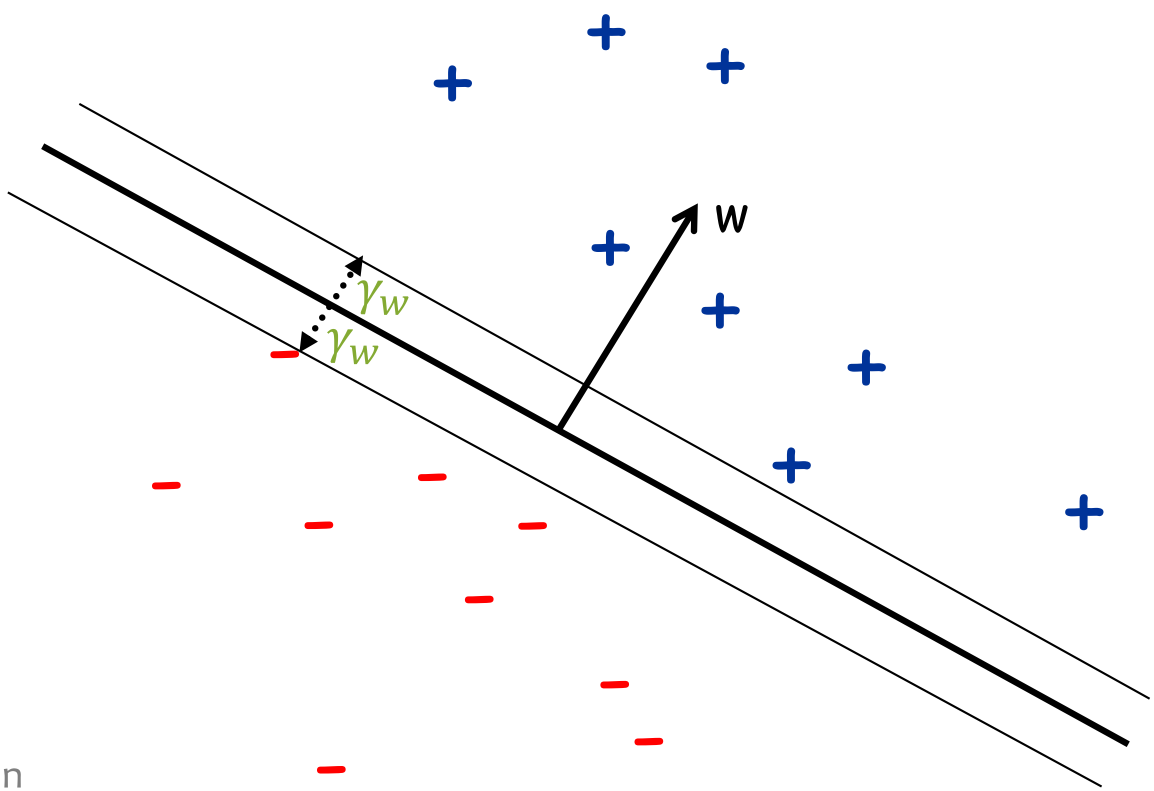


Slide from Nina Balcan

Geometric Margin

Definition: The margin of example x w.r.t. a linear separator w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

Definition: The margin γ_w of a set of examples S w.r.t. a linear separator w is the smallest margin over points $x \in S$.

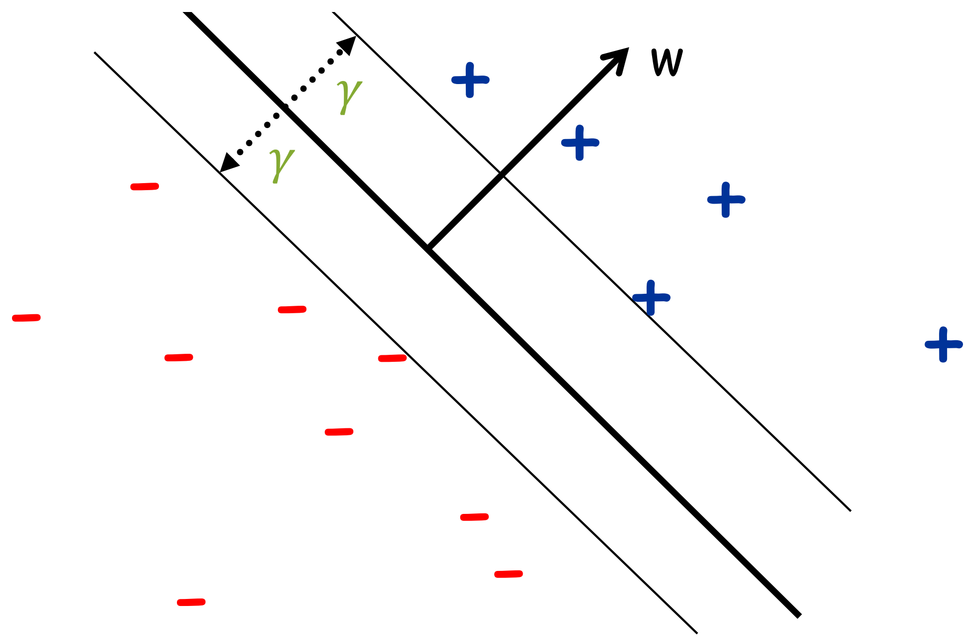


Geometric Margin

Definition: The margin of example x w.r.t. a linear separator w is the distance from x to the plane $w \cdot x = 0$ (or the negative if on wrong side)

Definition: The margin γ_w of a set of examples S w.r.t. a linear separator w is the smallest margin over points $x \in S$.

Definition: The margin γ of a set of examples S is the maximum γ_w over all linear separators w.

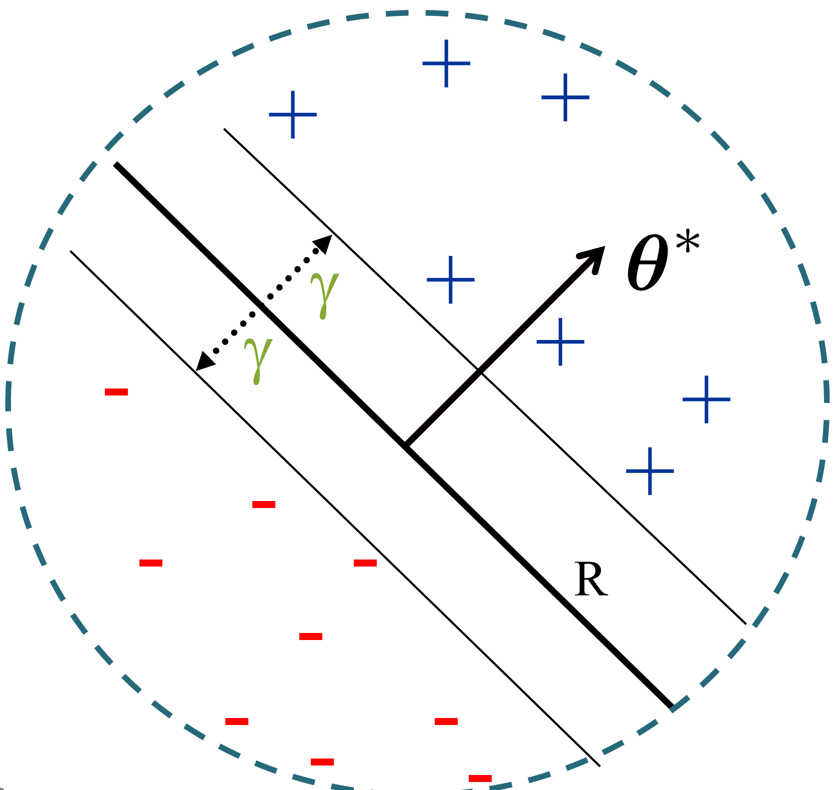


Slide from Nina Balcan

Perceptron Mistake Bound

Guarantee: if some data has margin γ and all points lie inside a ball of radius R rooted at the origin, then the online Perceptron algorithm makes $\leq (R/\gamma)^2$ mistakes

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes! The algorithm is invariant to scaling.)

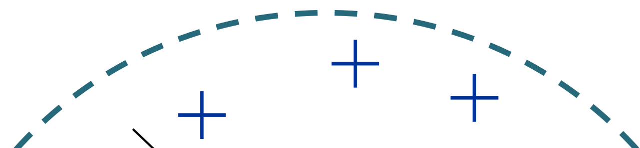


Slide adapted from Nina Balcan

Perceptron Mistake Bound

Guarantee: if some data has margin γ and all points lie inside a ball of radius R rooted at the origin, then the online Perceptron algorithm makes $\leq (R/\gamma)^2$ mistakes

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes! The algorithm is invariant to scaling.)



Def: We say that the (batch) perceptron algorithm has **converged** if it stops making mistakes on the training data (perfectly classifies the training data).

Main Takeaway: For **linearly separable** data, if the perceptron algorithm cycles repeatedly through the data, it will **converge** in a finite # of steps.

PROOF OF THE MISTAKE BOUND

Analysis: Perceptron

Perceptron Mistake Bound

Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.
Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ *s.t.* $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$ *and some* $\gamma > 0$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$

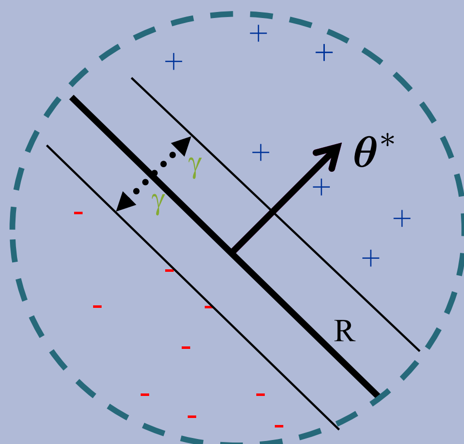


Figure from Nina Balcan

Analysis: Percept

Common *Misunderstanding*:

The **radius** is **centered at the origin,** not at the center of the points.

Perceptron Mistake Boun

Theorem 0.1 (Block (1962), Novikoff (19

Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$

Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ *s.t.* $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$ *and some* $\gamma > 0$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$

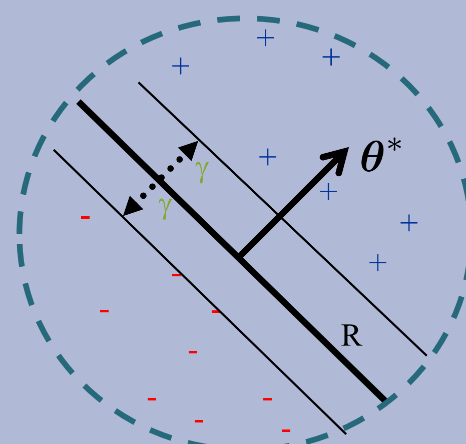


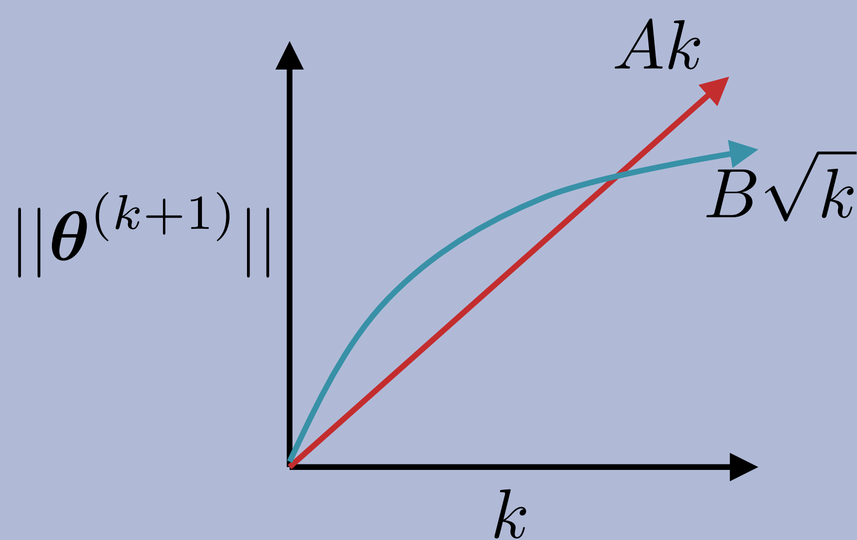
Figure from Nina Balcan

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

We will show that there exist constants A and B s.t.

$$Ak \leq ||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$$



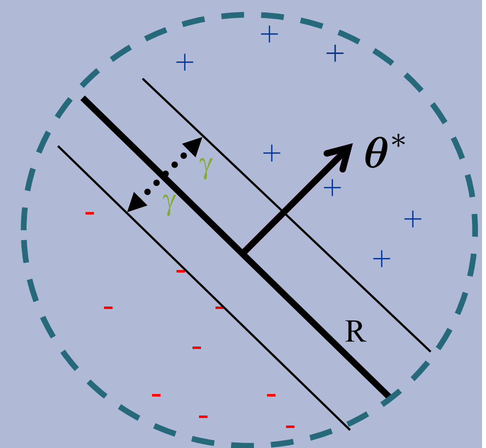
Analysis: Perceptron

Theorem 0.1 (Block (1962), Novikoff (1962)).
Given dataset: $\mathcal{D} = \{(\mathbf{x}^{(i)}, y^{(i)})\}_{i=1}^{N}$.
Suppose:

1. *Finite size inputs:* $||x^{(i)}|| \leq R$
2. *Linearly separable data:* $\exists \boldsymbol{\theta}^*$ *s.t.* $||\boldsymbol{\theta}^*|| = 1$ *and* $y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)}) \geq \gamma, \forall i$ *and some* $\gamma > 0$

Then: The number of mistakes made by the Perceptron algorithm on this dataset is

$$k \leq (R/\gamma)^2$$



Algorithm 1 Perceptron Learning Algorithm (Online)

procedure PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots\}$)
    $\boldsymbol{\theta} \leftarrow \mathbf{0}$, $k = 1$ ▷ Initialize parameters
    for $i \in \{1, 2, \ldots\}$ **do** ▷ For each example
        if $y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)}) \leq 0$ **then** ▷ If mistake
            $\boldsymbol{\theta}^{(k+1)} \leftarrow \boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)}$ ▷ Update parameters
            $k \leftarrow k + 1$
    return $\boldsymbol{\theta}$

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 1: for some A, $Ak \leq ||\boldsymbol{\theta}^{(k+1)}||$

$$\boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^* = (\boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)})\boldsymbol{\theta}^*$$

by Perceptron algorithm update

$$= \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^* + y^{(i)}(\boldsymbol{\theta}^* \cdot \mathbf{x}^{(i)})$$

$$\geq \boldsymbol{\theta}^{(k)} \cdot \boldsymbol{\theta}^* + \gamma$$

by assumption

$$\Rightarrow \boldsymbol{\theta}^{(k+1)} \cdot \boldsymbol{\theta}^* \geq k\gamma$$

by induction on k since $\theta^{(1)} = \mathbf{0}$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}|| \geq k\gamma$$

since $||\mathbf{w}|| \times ||\mathbf{u}|| \geq \mathbf{w} \cdot \mathbf{u}$ and $||\theta^*|| = 1$

Cauchy-Schwartz inequality

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 2: for some B, $||\boldsymbol{\theta}^{(k+1)}|| \leq B\sqrt{k}$

$$||\boldsymbol{\theta}^{(k+1)}||^2 = ||\boldsymbol{\theta}^{(k)} + y^{(i)}\mathbf{x}^{(i)}||^2$$

by Perceptron algorithm update

$$= ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||\mathbf{x}^{(i)}||^2 + 2y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)})$$

$$\leq ||\boldsymbol{\theta}^{(k)}||^2 + (y^{(i)})^2||\mathbf{x}^{(i)}||^2$$

since kth mistake $\Rightarrow y^{(i)}(\boldsymbol{\theta}^{(k)} \cdot \mathbf{x}^{(i)}) \leq 0$

$$= ||\boldsymbol{\theta}^{(k)}||^2 + R^2$$

since $(y^{(i)})^2||\mathbf{x}^{(i)}||^2 = ||\mathbf{x}^{(i)}||^2 = R^2$ by assumption and $(y^{(i)})^2 = 1$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}||^2 \leq kR^2$$

by induction on k since $(\theta^{(1)})^2 = 0$

$$\Rightarrow ||\boldsymbol{\theta}^{(k+1)}|| \leq \sqrt{k}R$$

Analysis: Perceptron

Proof of Perceptron Mistake Bound:

Part 3: Combining the bounds finishes the proof.

$$k\gamma \leq ||\boldsymbol{\theta}^{(k+1)}|| \leq \sqrt{k}R$$

$$\Rightarrow k \leq (R/\gamma)^2$$

The total number of mistakes must be less than this

Analysis: Perceptron

What if the data is *not* linearly separable?

1. Perceptron will **not converge** in this case (it can't!)
2. However, Freund & Schapire (1999) show that by projecting the points (hypothetically) into a higher dimensional space, we can achieve a similar bound on the number of mistakes made on **one pass** through the sequence of examples

Theorem 2. *Let* $\langle(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\rangle$ *be a sequence of labeled examples with* $\|\mathbf{x}_i\| \leq R$. *Let* $\mathbf{u}$ *be any vector with* $\|\mathbf{u}\| = 1$ *and let* $\gamma > 0$. *Define the deviation of each example as*

$$d_i = \max\{0, \gamma - y_i(\mathbf{u} \cdot \mathbf{x}_i)\},$$

and define $D = \sqrt{\sum_{i=1}^{m} d_i^2}$. *Then the number of mistakes of the online perceptron algorithm on this sequence is bounded by*

$$\left(\frac{R + D}{\gamma}\right)^2.$$

Summary: Perceptron

- Perceptron is a **linear classifier**
- **Simple learning algorithm**: when a mistake is made, add / subtract the features
- Perceptron will converge if the data are **linearly separable**, it will **not** converge if the data are **linearly inseparable**
- For linearly separable and inseparable data, we can **bound the number of mistakes** (geometric argument)
- **Extensions** support nonlinear separators and structured prediction

Perceptron Learning Objectives

You should be able to…

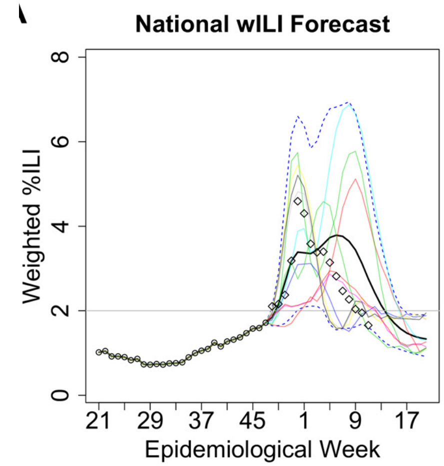
- Explain the difference between online learning and batch learning
- Implement the perceptron algorithm for binary classification [CIML]
- Determine whether the perceptron algorithm will converge based on properties of the dataset, and the limitations of the convergence guarantees
- Describe the inductive bias of perceptron and the limitations of linear models
- Draw the decision boundary of a linear model
- Identify whether a dataset is linearly separable or not
- Defend the use of a bias term in perceptron

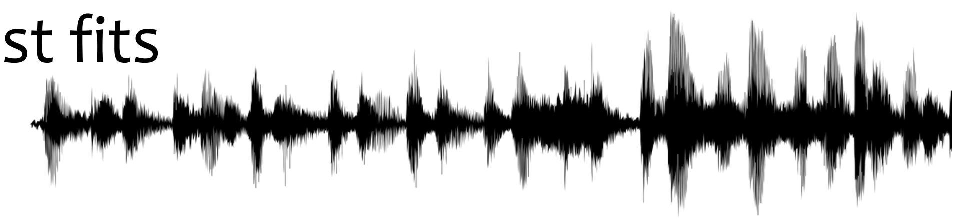
REGRESSION

Regression

Goal:

- Given a training dataset of pairs (**x**,y) where
  - **x** is a vector
  - y is a scalar
- Learn a function (aka. curve or line) y' = h(x) that best fits the training data



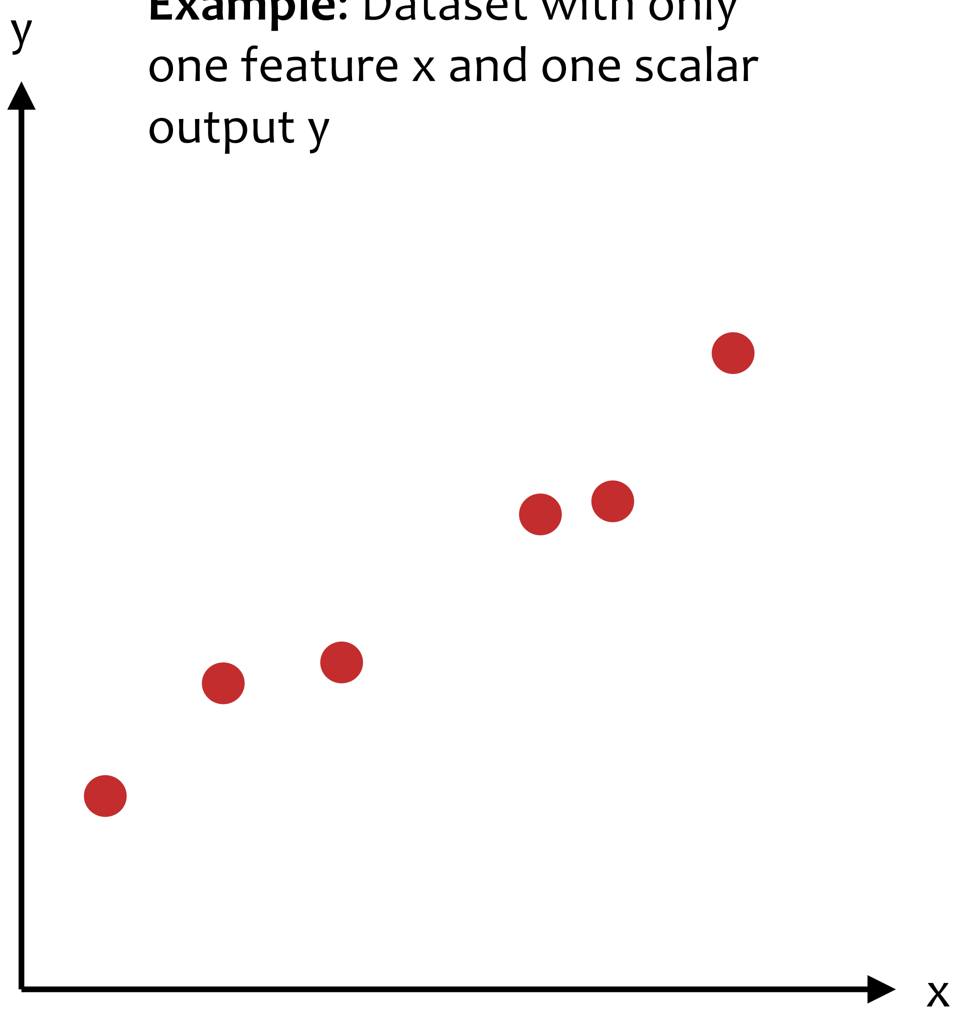


Example Applications:

- Stock price prediction
- Forecasting epidemics
- Speech synthesis
- Generation of images (e.g. *Deep Dream*)

Regression

Example: Dataset with only one feature x and one scalar output y

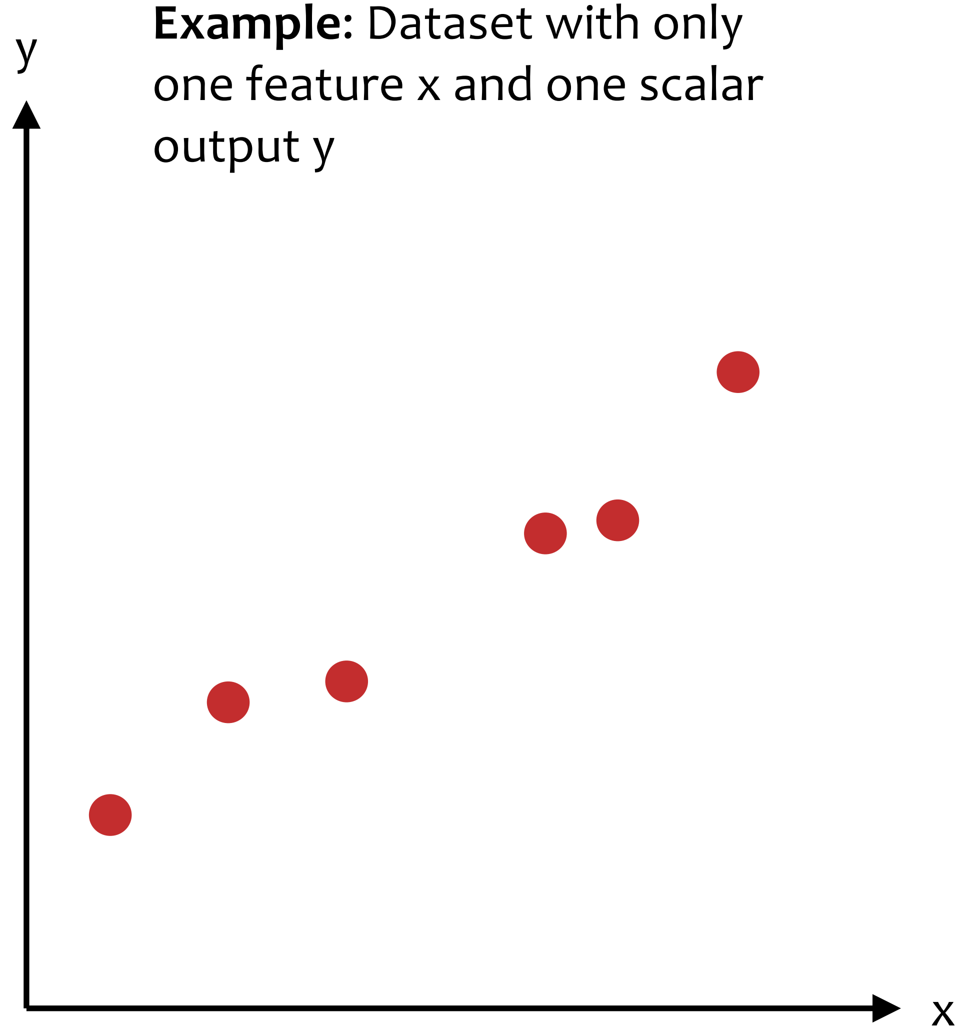


Q: What is the function that best fits these points?

K-NEAREST NEIGHBOR REGRESSION

k-NN Regression

Example: Dataset with only one feature x and one scalar output y



Algorithm 1: k=1 Nearest Neighbor Regression

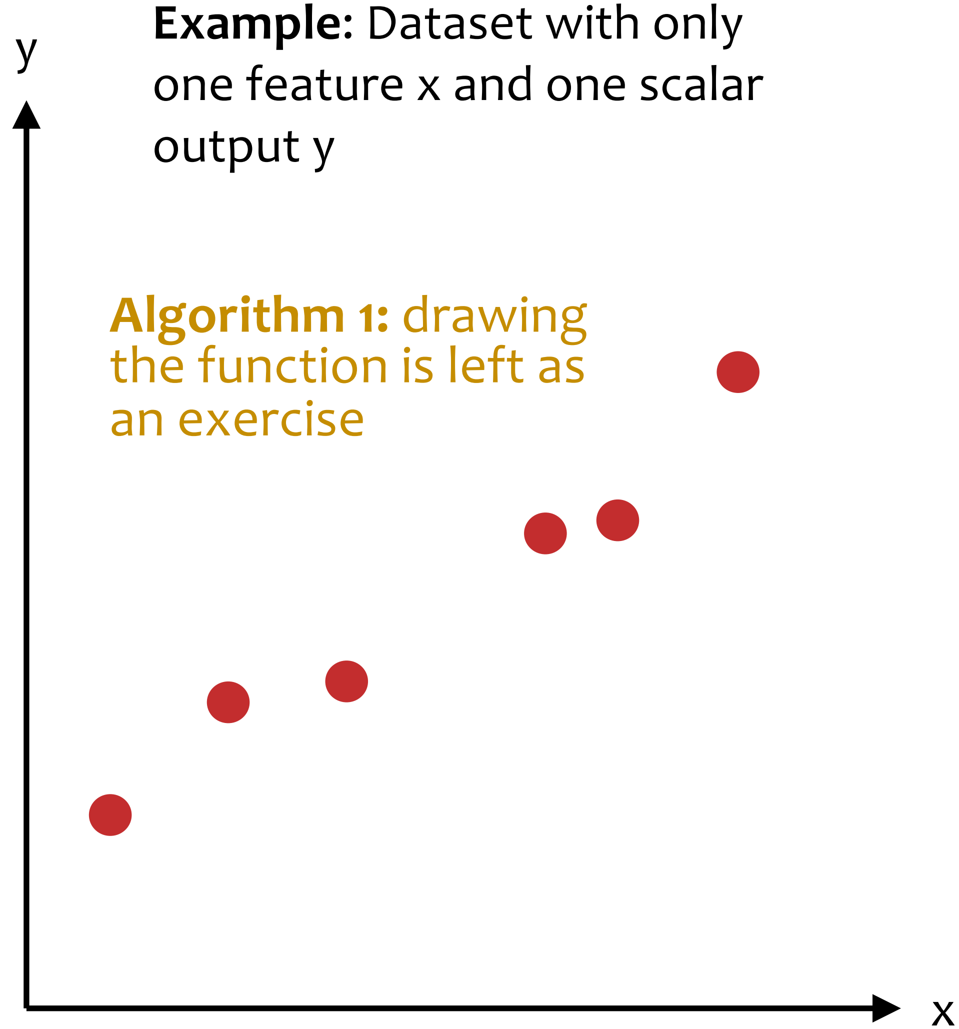
- *Train*: store all (x, y) pairs
- *Predict*: pick the nearest x in training data and return its y

Algorithm 2: k=2 Nearest Neighbors Distance Weighted Regression

- *Train*: store all (x, y) pairs
- *Predict*: pick the nearest two instances $x^{(n1)}$ and $x^{(n2)}$ in training data and return the weighted average of their y values

k-NN Regression

Example: Dataset with only one feature x and one scalar output y



Algorithm 1: drawing the function is left as an exercise

Algorithm 1: k=1 Nearest Neighbor Regression

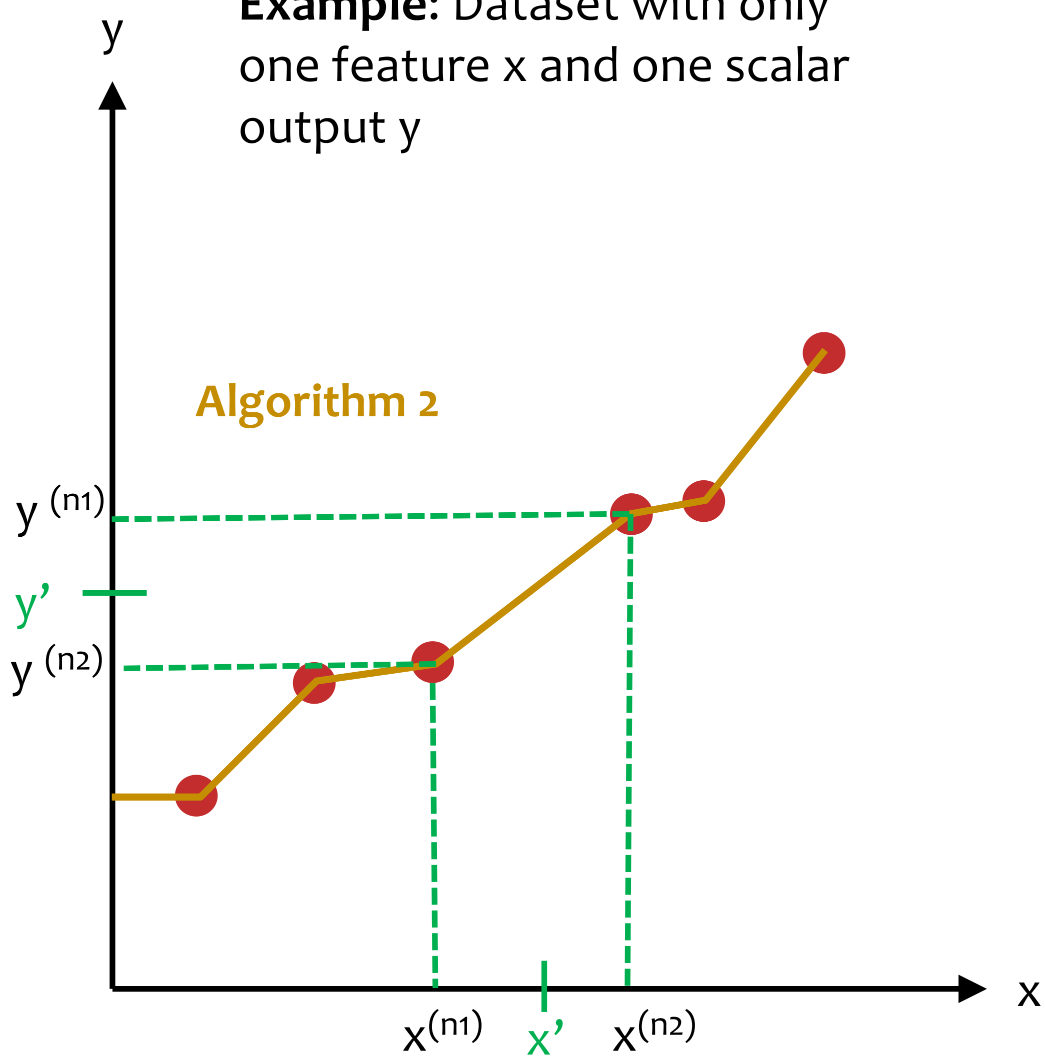
- *Train*: store all (x, y) pairs
- *Predict*: pick the nearest x in training data and return its y

Algorithm 2: k=2 Nearest Neighbors Distance Weighted Regression

- *Train*: store all (x, y) pairs
- *Predict*: pick the nearest two instances $x^{(n1)}$ and $x^{(n2)}$ in training data and return the weighted average of their y values

k-NN Regression

Example: Dataset with only one feature x and one scalar output y



Algorithm 1: k=1 Nearest Neighbor Regression

- *Train*: store all (x, y) pairs
- *Predict*: pick the nearest x in training data and return its y

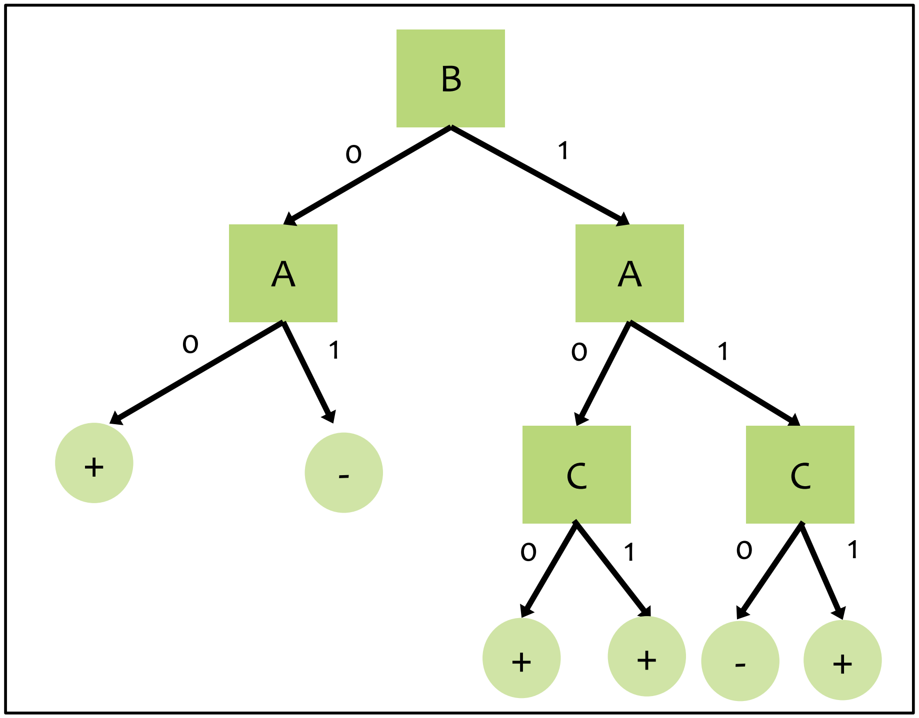
Algorithm 2: k=2 Nearest Neighbors Distance Weighted Regression

- *Train*: store all (x, y) pairs
- *Predict*: pick the nearest two instances $x^{(n1)}$ and $x^{(n2)}$ in training data and return the weighted average of their y values

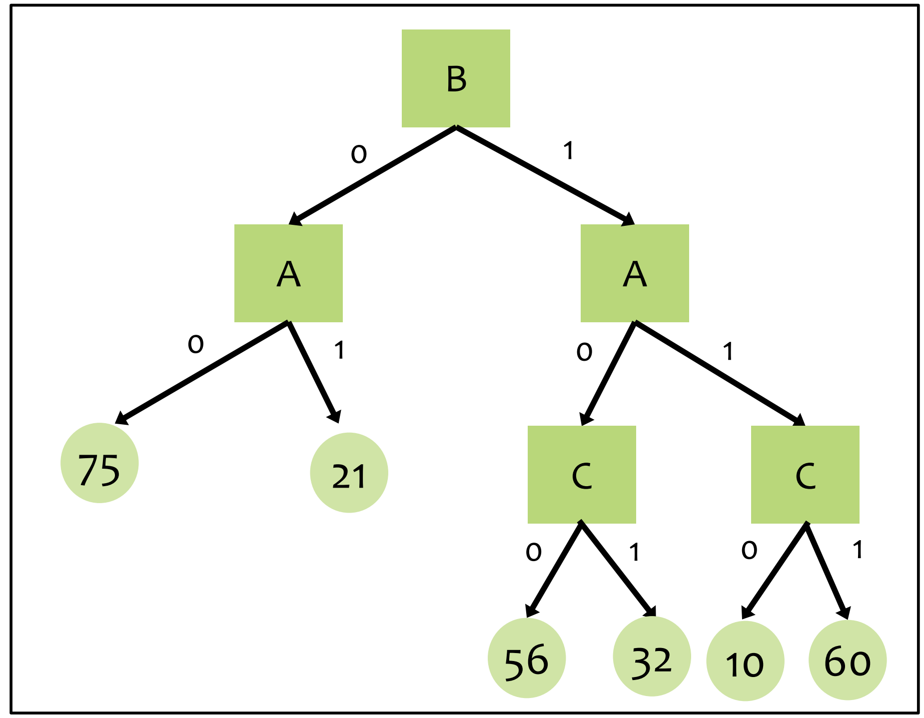
DECISION TREE REGRESSION

Decision Tree Regression

Decision Tree for Classification



Decision Tree for Regression

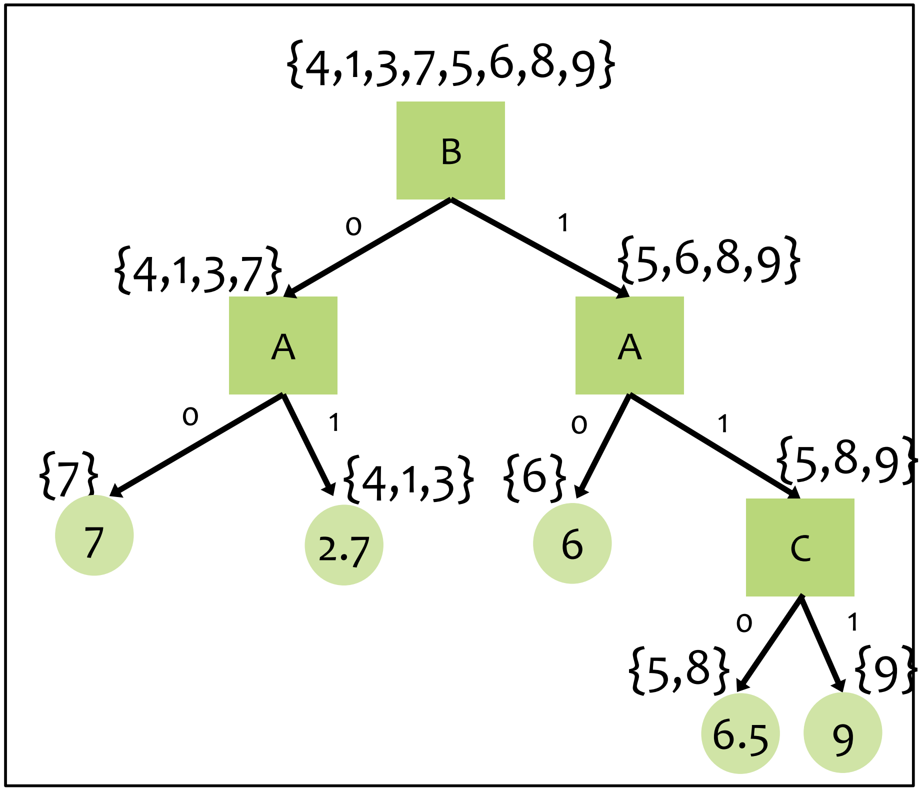


Decision Tree Regression

Dataset for Regression

Y	A	B	C
4	1	0	0
1	1	0	1
3	1	0	0
7	0	0	1
5	1	1	0
6	0	1	1
8	1	1	0
9	1	1	1

Decision Tree for Regression



During learning, choose the attribute that minimizes an appropriate splitting criterion (e.g. mean squared error, mean absolute error)

LINEAR FUNCTIONS, RESIDUALS, AND MEAN SQUARED ERROR

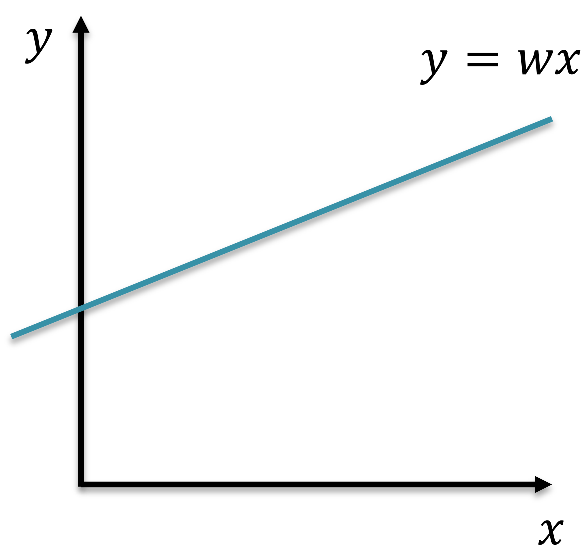
Linear Functions

<u>*Def*</u>: Regression is predicting real-valued outputs

$$\mathcal{D} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n} \text{ with } \mathbf{x}^{(i)} \in \mathbb{R}^M, y^{(i)} \in \mathbb{R}$$

Common Misunderstanding:

Linear functions ≠ Linear decision boundaries



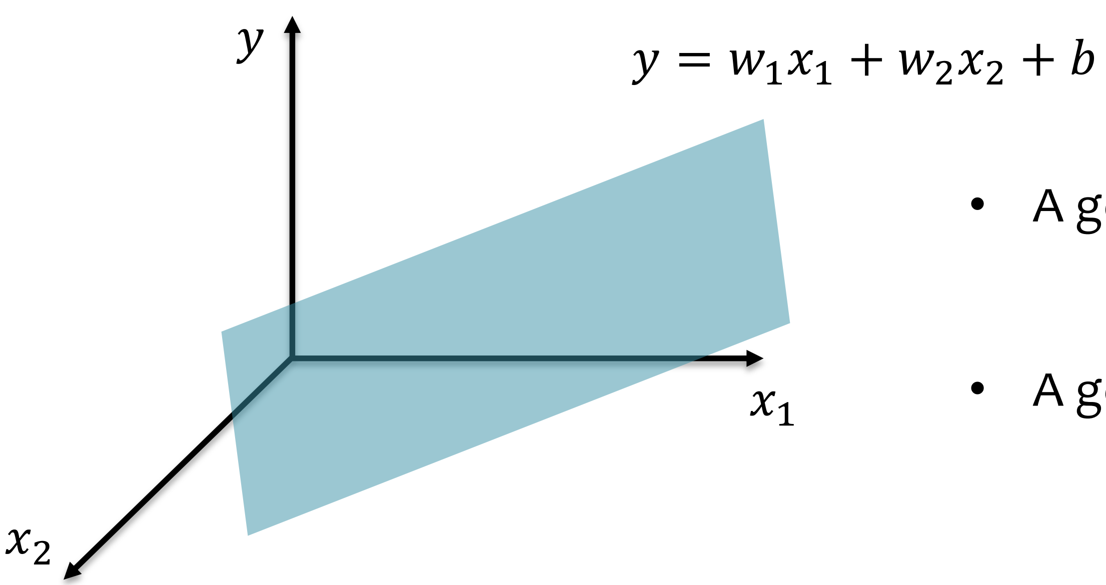
Linear Functions

Def: Regression is predicting real-valued outputs

$$\mathcal{D} = \left\{\left(\mathbf{x}^{(i)}, y^{(i)}\right)\right\}_{i=1}^{n} \text{ with } \mathbf{x}^{(i)} \in \mathbb{R}^M, y^{(i)} \in \mathbb{R}$$

Common Misunderstanding:

Linear functions ≠ Linear decision boundaries



- A general linear function is
$$y = \mathbf{w}^T\mathbf{x} + b$$
- A general linear decision boundary is
$$y = \text{sign}(\mathbf{w}^T\mathbf{x} + b)$$

Regression Problems

Chalkboard

- Residuals
- Mean squared error

The Big Picture

OPTIMIZATION FOR ML

Unconstrained Optimization

- *Def*: In **unconstrained optimization**, we try minimize (or maximize) a function with *no constraints* on the inputs to the function

Given a function $J(\boldsymbol{\theta}), J : \mathbb{R}^M \rightarrow \mathbb{R}$

Our goal is to find $\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta} \in \mathbb{R}^M}{\text{argmin}}\, J(\boldsymbol{\theta})$

For ML, these are the parameters

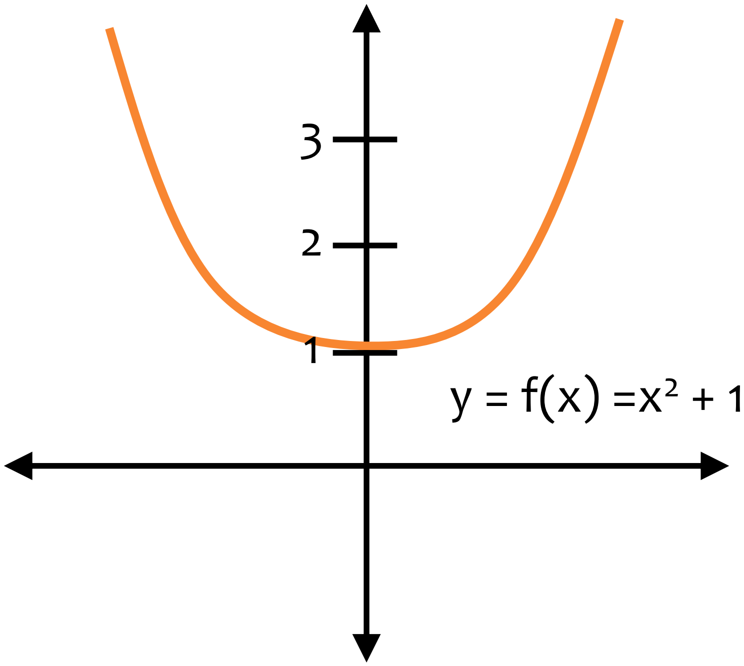
For ML, this is the objective function

Optimization for ML

Not quite the same setting as other fields…

- Function we are optimizing might not be the true goal
  (e.g. likelihood vs generalization error)
- Precision might not matter
  (e.g. data is noisy, so optimal up to 1e-16 might not help)
- Stopping early can help generalization error
  (i.e. "early stopping" is a technique for regularization – discussed more next time)

min vs. argmin



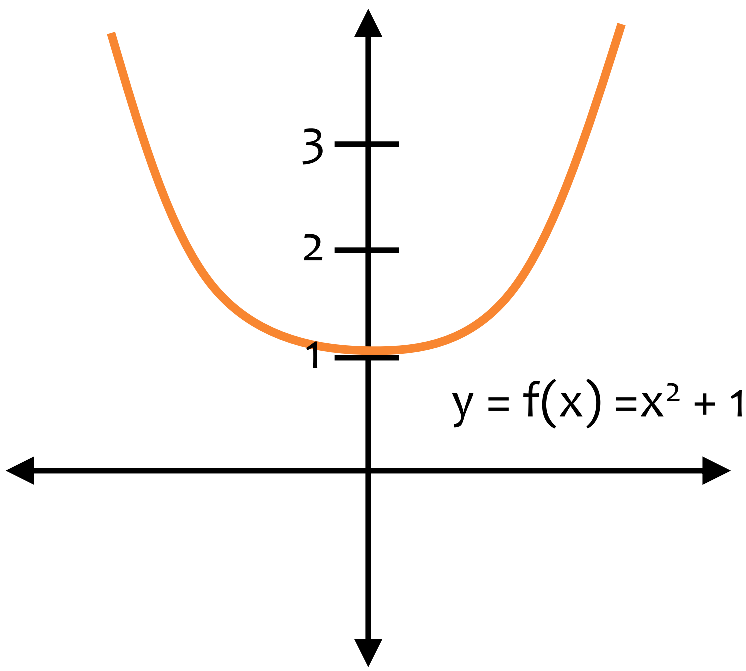
$v^* = \min_x f(x)$

$x^* = \text{argmin}_x f(x)$

1. Question: What is v*?

2. Question: What is x*?

min vs. argmin



$v^* = \min_x f(x)$

$x^* = \text{argmin}_x f(x)$

1. Question: What is v*?

   $v^* = 1$, the minimum value of the function

2. Question: What is x*?

   $x^* = 0$, the argument that yields the minimum value

OPTIMIZATION METHOD #0: RANDOM GUESSING

Notation Trick: Folding in the Intercept Term

$$\mathbf{x}' = [1, x_1, x_2, \ldots, x_M]^T$$

$$\boldsymbol{\theta} = [b, w_1, \ldots, w_M]^T$$

Notation Trick: fold the bias *b* and the weights ***w*** into a single vector **θ** by prepending a constant to ***x*** and increasing dimensionality by one!

$$h_{\mathbf{w},b}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$$

$$h_{\boldsymbol{\theta}}(\mathbf{x}') = \boldsymbol{\theta}^T\mathbf{x}'$$

This convenience trick allows us to more compactly talk about linear functions as a simple dot product (without explicitly writing out the intercept term every time).

Linear Regression as Function Approximation

$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^{N}$
where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \mathbb{R}$

1. Assume $\mathcal{D}$ generated as:

$$\mathbf{x}^{(i)} \sim p^*(\cdot)$$
$$y^{(i)} = h^*(\mathbf{x}^{(i)})$$

2. Choose hypothesis space, $\mathcal{H}$:
   all linear functions in M-dimensional space

$$\mathcal{H} = \{h_{\boldsymbol{\theta}} : h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}, \boldsymbol{\theta} \in \mathbb{R}^M\}$$

3. Choose an objective function:
   mean squared error (MSE)

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^{N} e_i^2$$
$$= \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)^2$$
$$= \frac{1}{N} \sum_{i=1}^{N} \left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)^2$$

4. Solve the unconstrained optimization problem via favorite method:
   - *gradient descent*
   - *closed form*
   - *stochastic gradient descent*
   - ...

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmin}}\, J(\boldsymbol{\theta})$$

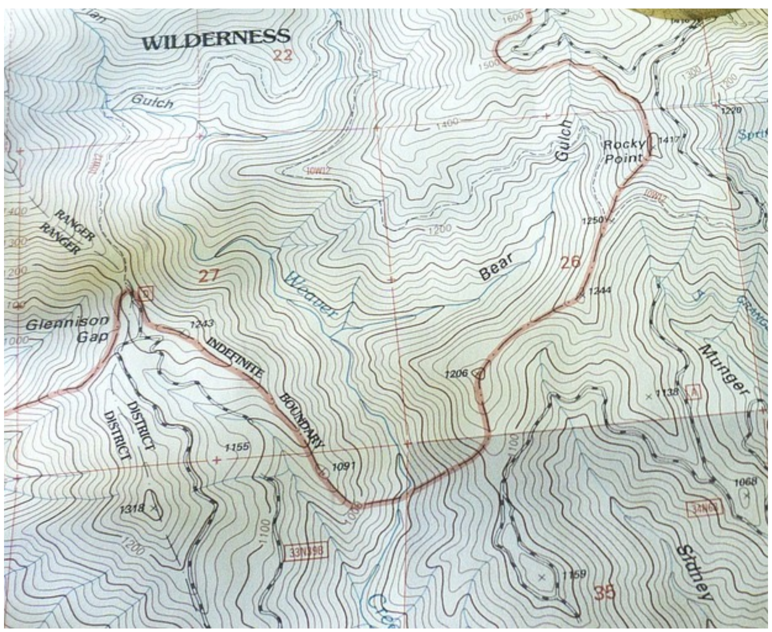
5. Test time: given a new $\mathbf{x}$, make prediction $\hat{y}$

$$\hat{y} = h_{\hat{\boldsymbol{\theta}}}(\mathbf{x}) = \hat{\boldsymbol{\theta}}^T \mathbf{x}$$

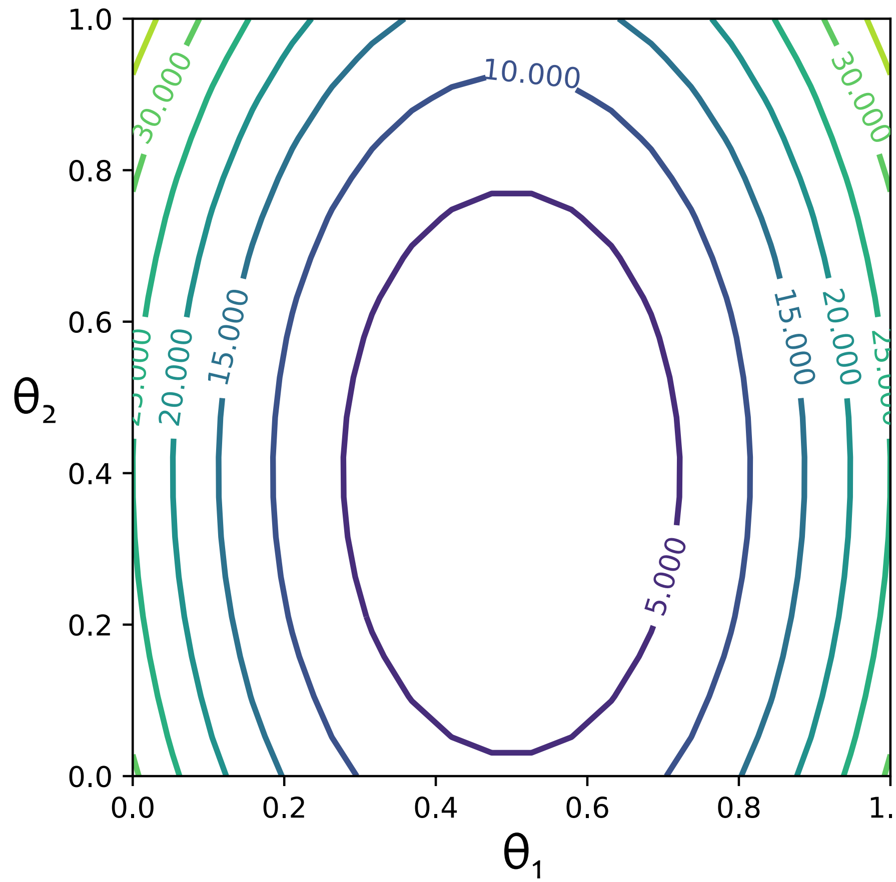
Contour Plots

Contour Plots

1. Each level curve labeled with value
2. Value label indicates the value of the function for all points lying on that level curve
3. Just like a topographical map, but for a function



$J(\boldsymbol{\theta}) = J(\theta_1, \theta_2) = (10(\theta_1 - 0.5))^2 + (6(\theta_1 - 0.4))^2$

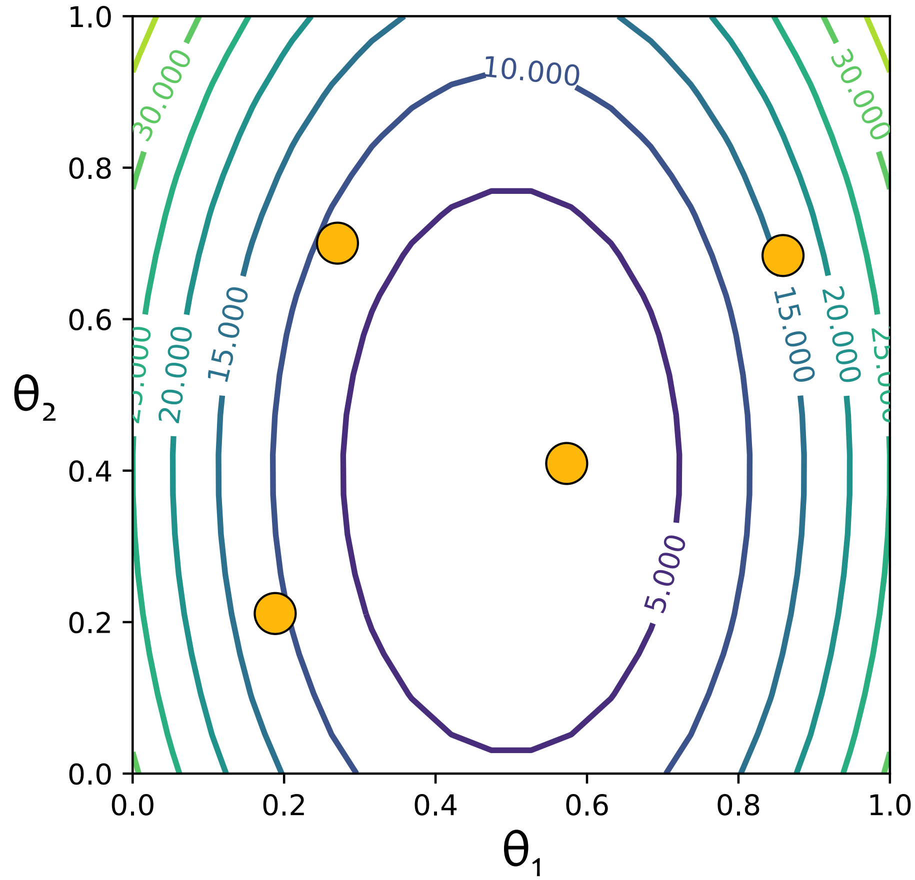


Optimization by Random Guessing

Optimization Method #0: Random Guessing

1. Pick a random $\boldsymbol{\theta}$
2. Evaluate $J(\boldsymbol{\theta})$
3. Repeat steps 1 and 2 many times
4. Return $\boldsymbol{\theta}$ that gives smallest $J(\boldsymbol{\theta})$

$J(\boldsymbol{\theta}) = J(\theta_1, \theta_2) = (10(\theta_1 - 0.5))^2 + (6(\theta_1 - 0.4))^2$



t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.2	0.2	10.4
2	0.3	0.7	7.2
3	0.6	0.4	1.0
4	0.9	0.7	16.2

Optimization by Random Guessing

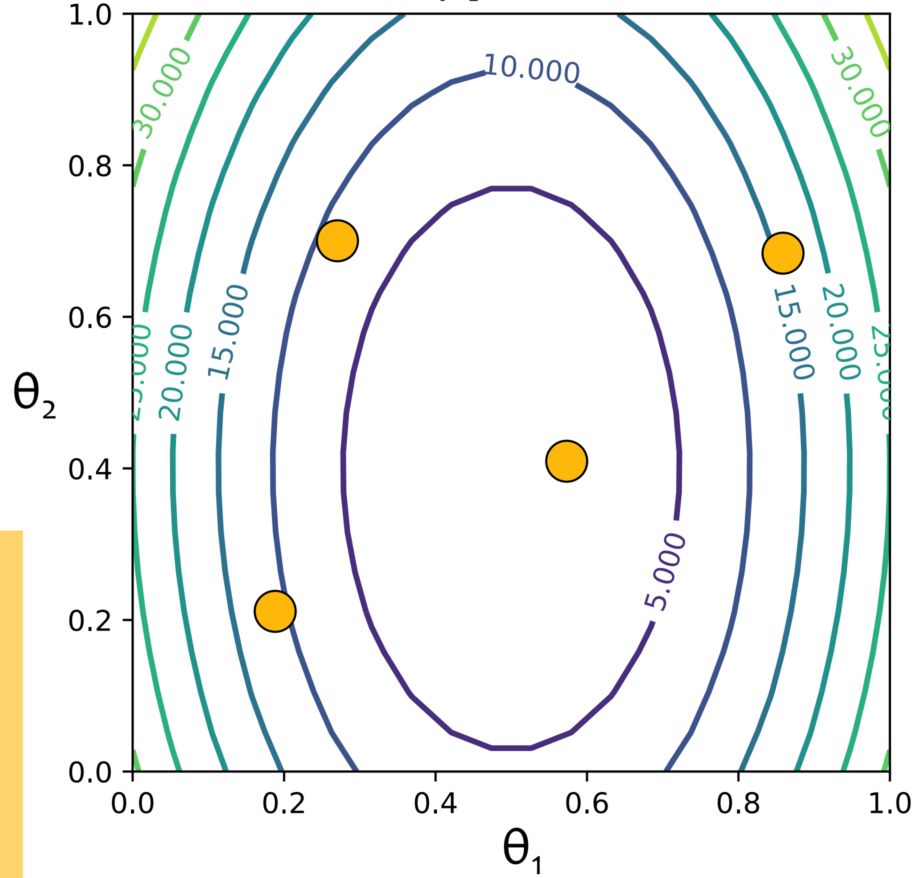
Optimization Method #0: Random Guessing

1. Pick a random $\boldsymbol{\theta}$
2. Evaluate $J(\boldsymbol{\theta})$
3. Repeat steps 1 and 2 many times
4. Return $\boldsymbol{\theta}$ that gives smallest $J(\boldsymbol{\theta})$

For Linear Regression:

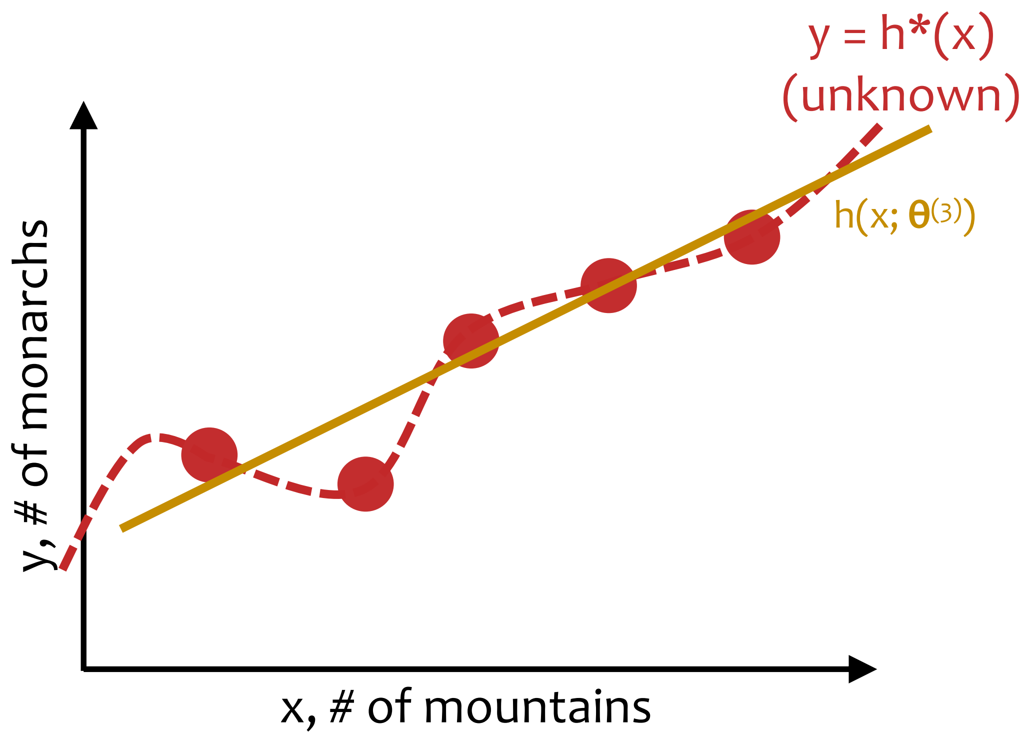
- **objective function** is Mean Squared Error (MSE)
- MSE $= J(w, b)$
  $= J(\theta_1, \theta_2) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)^2$
- contour plot: each line labeled with MSE – **lower means a better fit**
- **minimum** corresponds to parameters $(w,b) = (\theta_1, \theta_2)$ that **best fit** some training dataset

$$J(\boldsymbol{\theta}) = J(\theta_1, \theta_2) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)^2$$



t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.2	0.2	10.4
2	0.3	0.7	7.2
3	0.6	0.4	1.0
4	0.9	0.7	16.2

Counting Butterflies

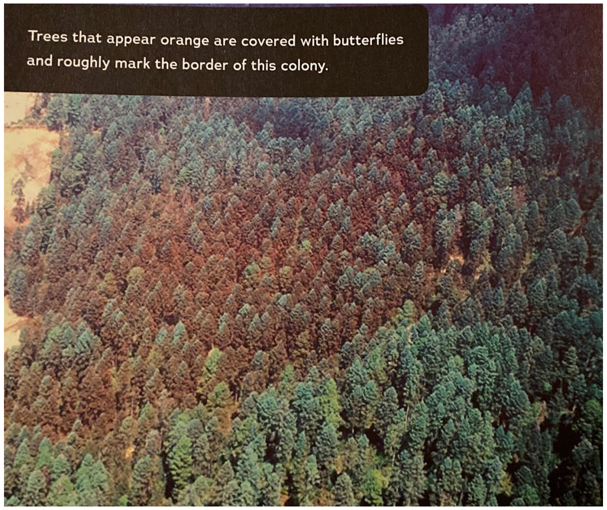


Linear Regression in High Dimensions

- In our discussions of linear regression, we will always assume there is just one output, y
- But our inputs will usually have many features:

$$\mathbf{x} = [x_1, x_2, \ldots, x_M]^T$$

- For example:
  - suppose we had a drone take pictures of each section of forest
  - each feature could correspond to a pixel in this image such that $x_m = 1$ if the pixel is orange and $x_m = 0$ otherwise
  - the output y would be the number of butterflies in each picture

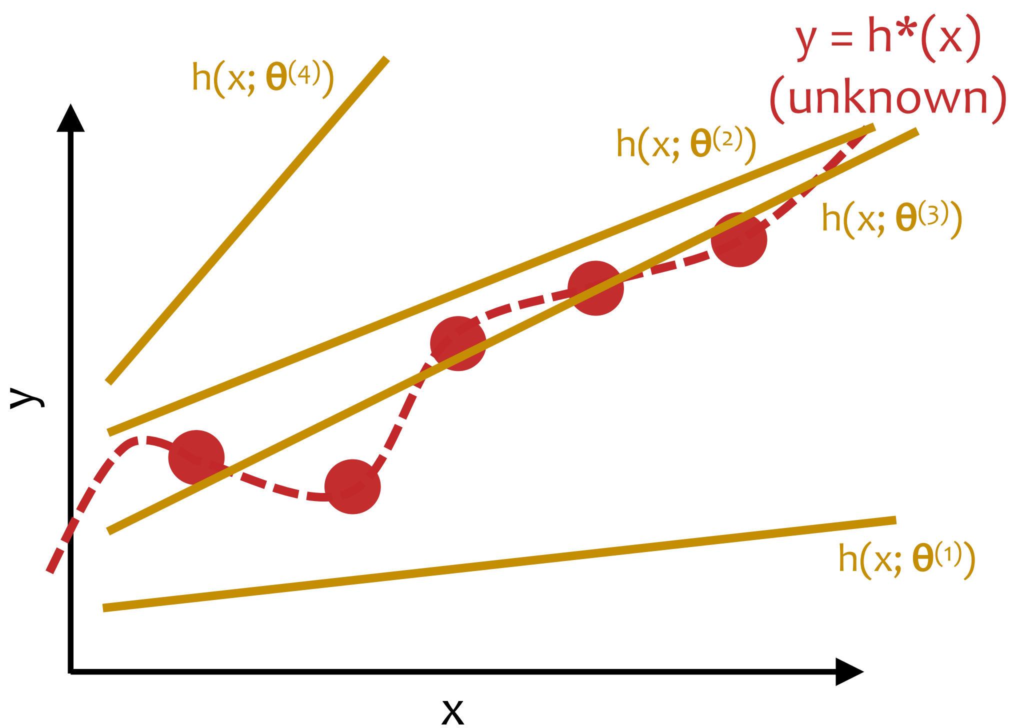


Q: How would you obtain ground truth data?

Linear Regression by Rand. Guessing

Optimization Method #0: Random Guessing

1. Pick a random $\boldsymbol{\theta}$
2. Evaluate $J(\boldsymbol{\theta})$
3. Repeat steps 1 and 2 many times
4. Return $\boldsymbol{\theta}$ that gives smallest $J(\boldsymbol{\theta})$



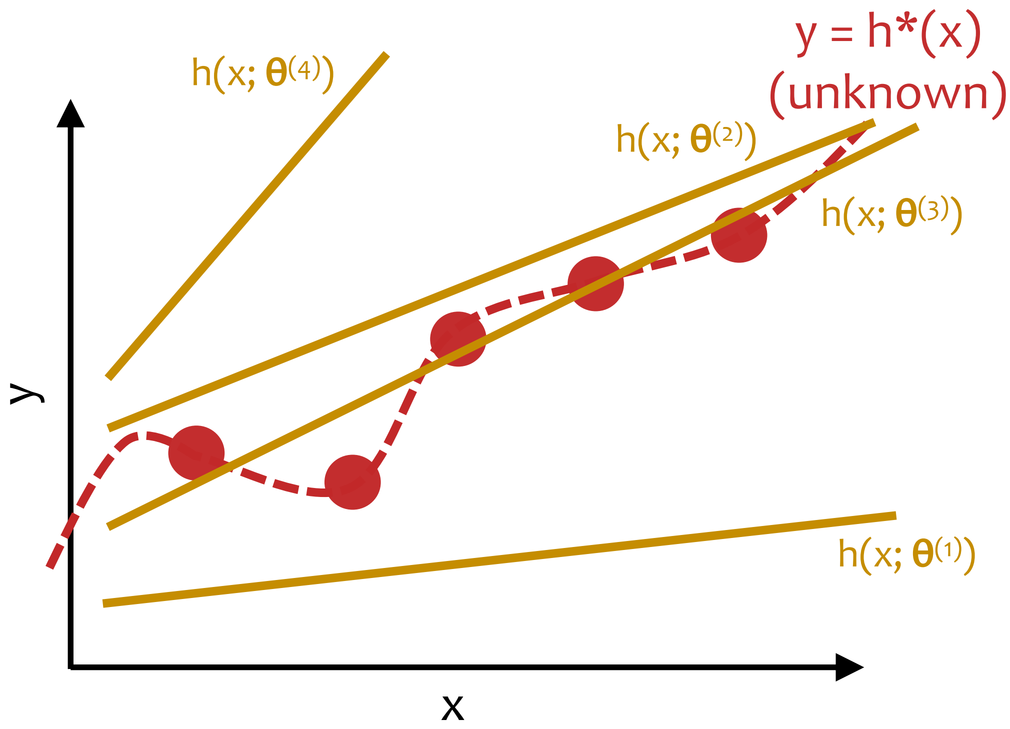
For Linear Regression:

- target function $h^*(x)$ is **unknown**
- only have access to $h^*(x)$ through **training examples** $(x^{(i)},y^{(i)})$
- want $h(x; \boldsymbol{\theta}^{(t)})$ that **best approximates** $h^*(x)$
- **enable generalization** w/inductive bias that restricts hypothesis class to **linear functions**

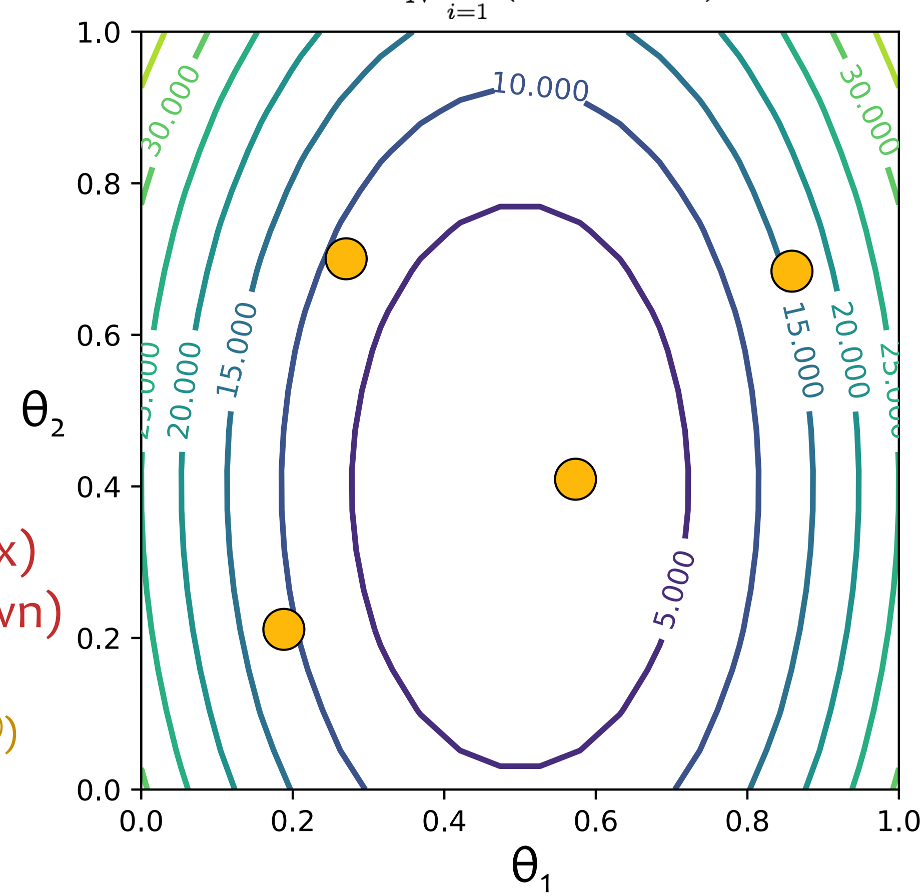
Linear Regression by Rand. Guessing

Optimization Method #0: Random Guessing

1. Pick a random $\boldsymbol{\theta}$
2. Evaluate $J(\boldsymbol{\theta})$
3. Repeat steps 1 and 2 many times
4. Return $\boldsymbol{\theta}$ that gives smallest $J(\boldsymbol{\theta})$



$J(\boldsymbol{\theta}) = J(\theta_1, \theta_2) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{\theta}^T\mathbf{x}^{(i)}\right)^2$



t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.2	0.2	10.4
2	0.3	0.7	7.2
3	0.6	0.4	1.0
4	0.9	0.7	16.2

OPTIMIZATION METHOD #1: GRADIENT DESCENT

Optimization for ML

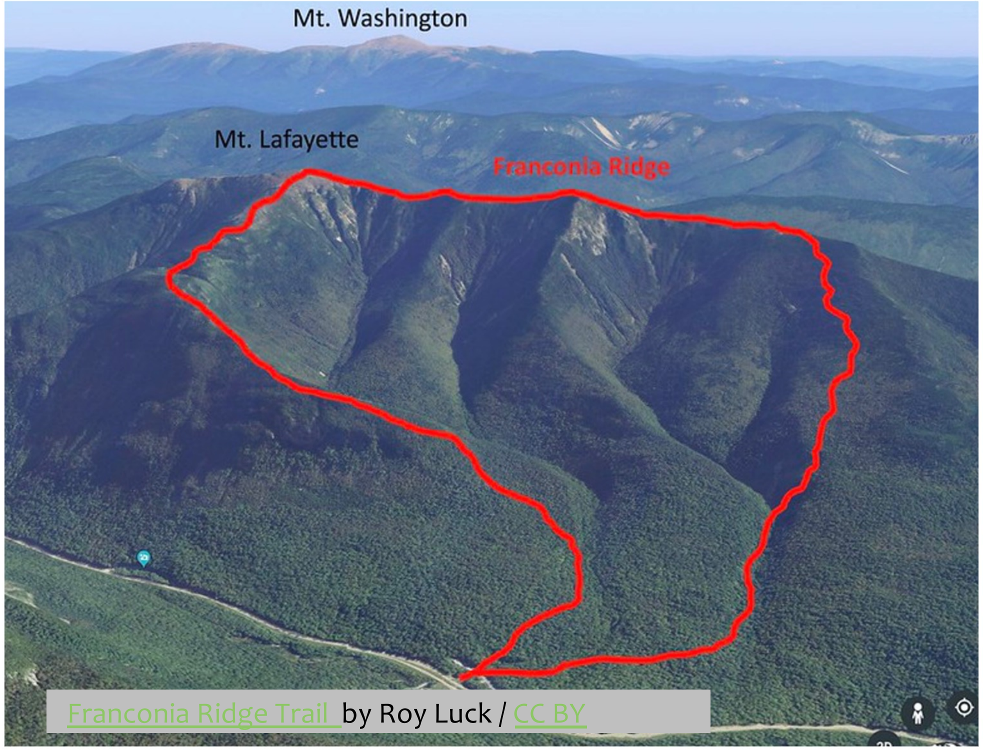
Chalkboard

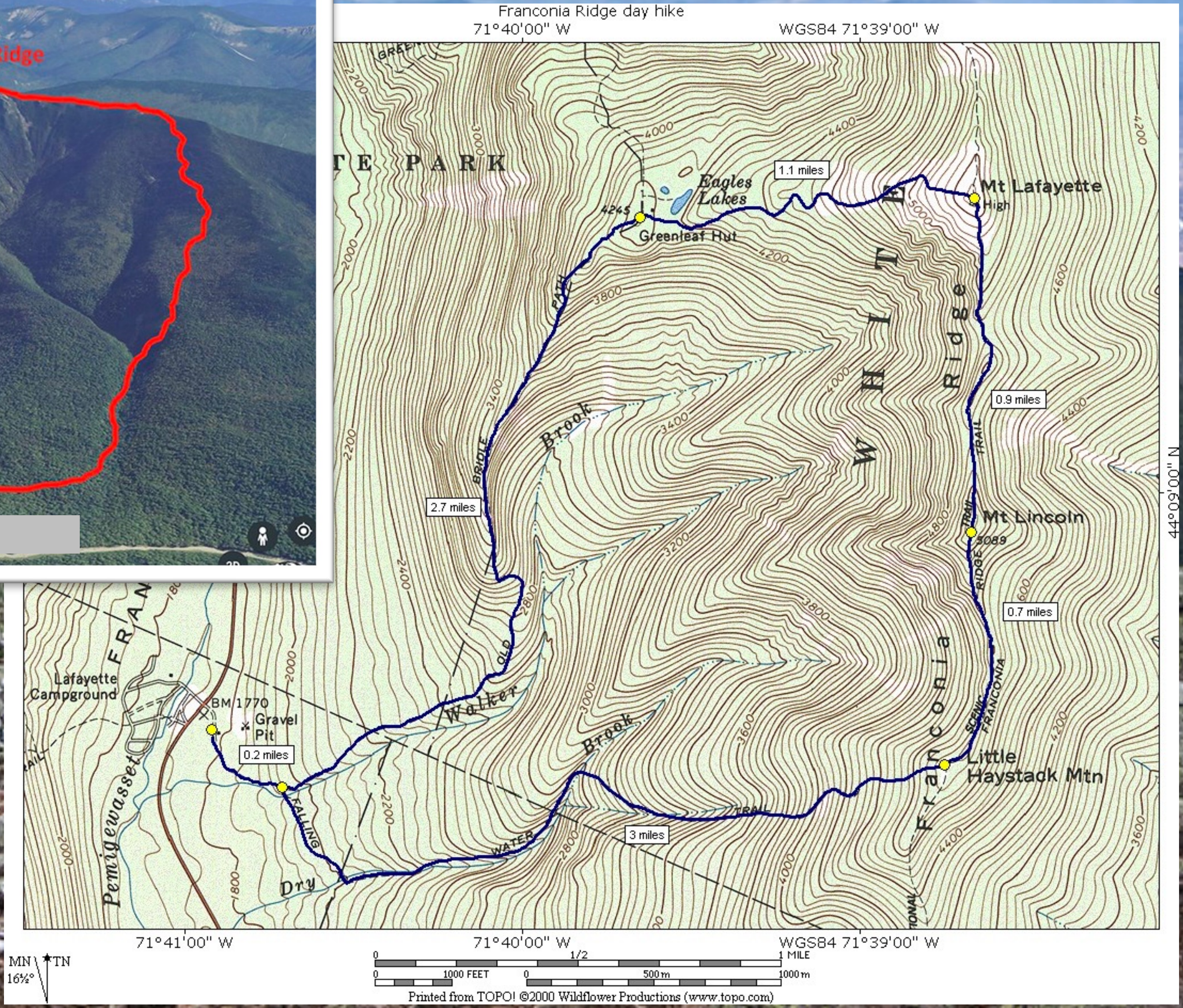
- Derivatives
- Gradient

Topographical Maps

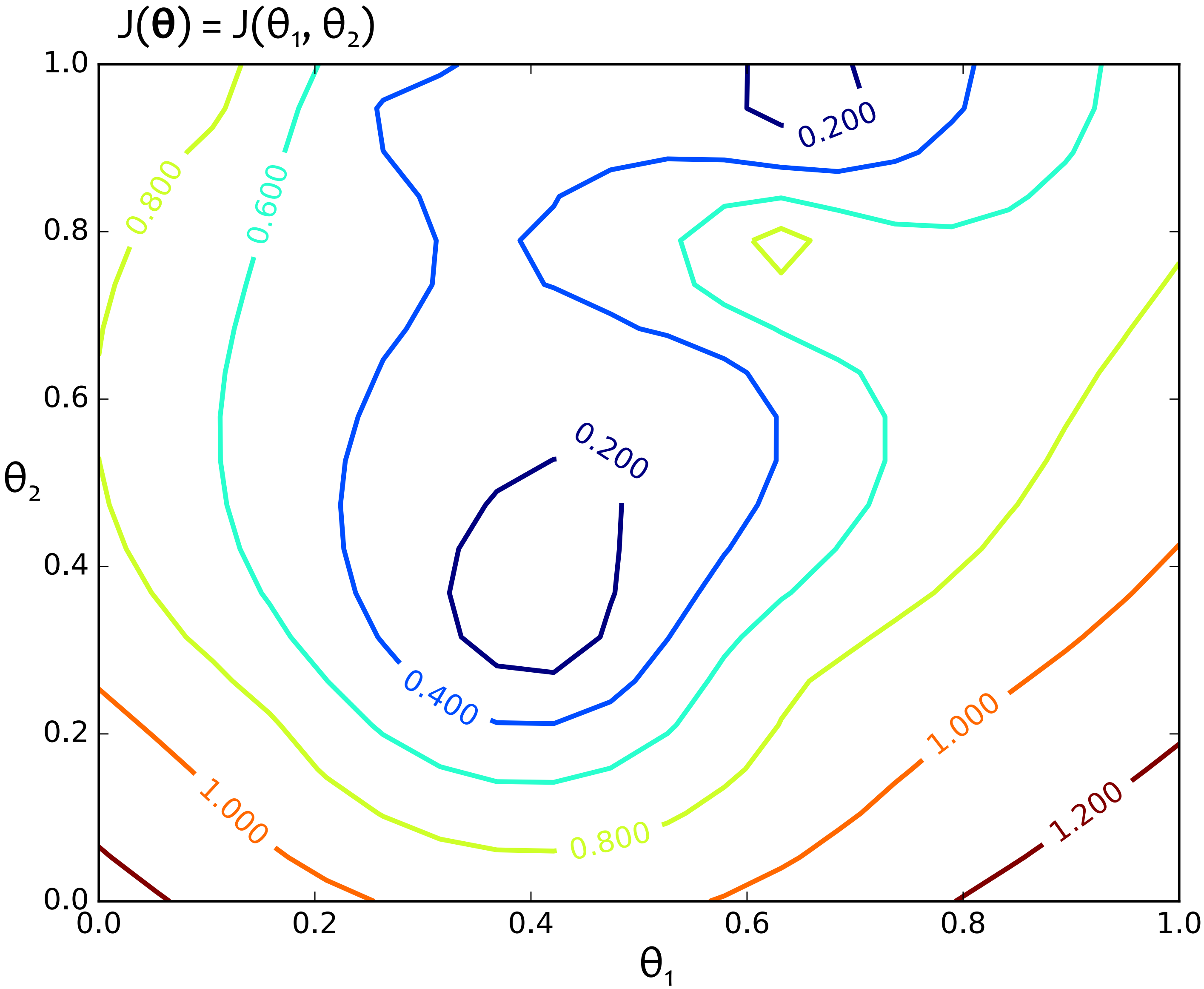
Franconia Ridge by Jeff P /

Topographical Maps



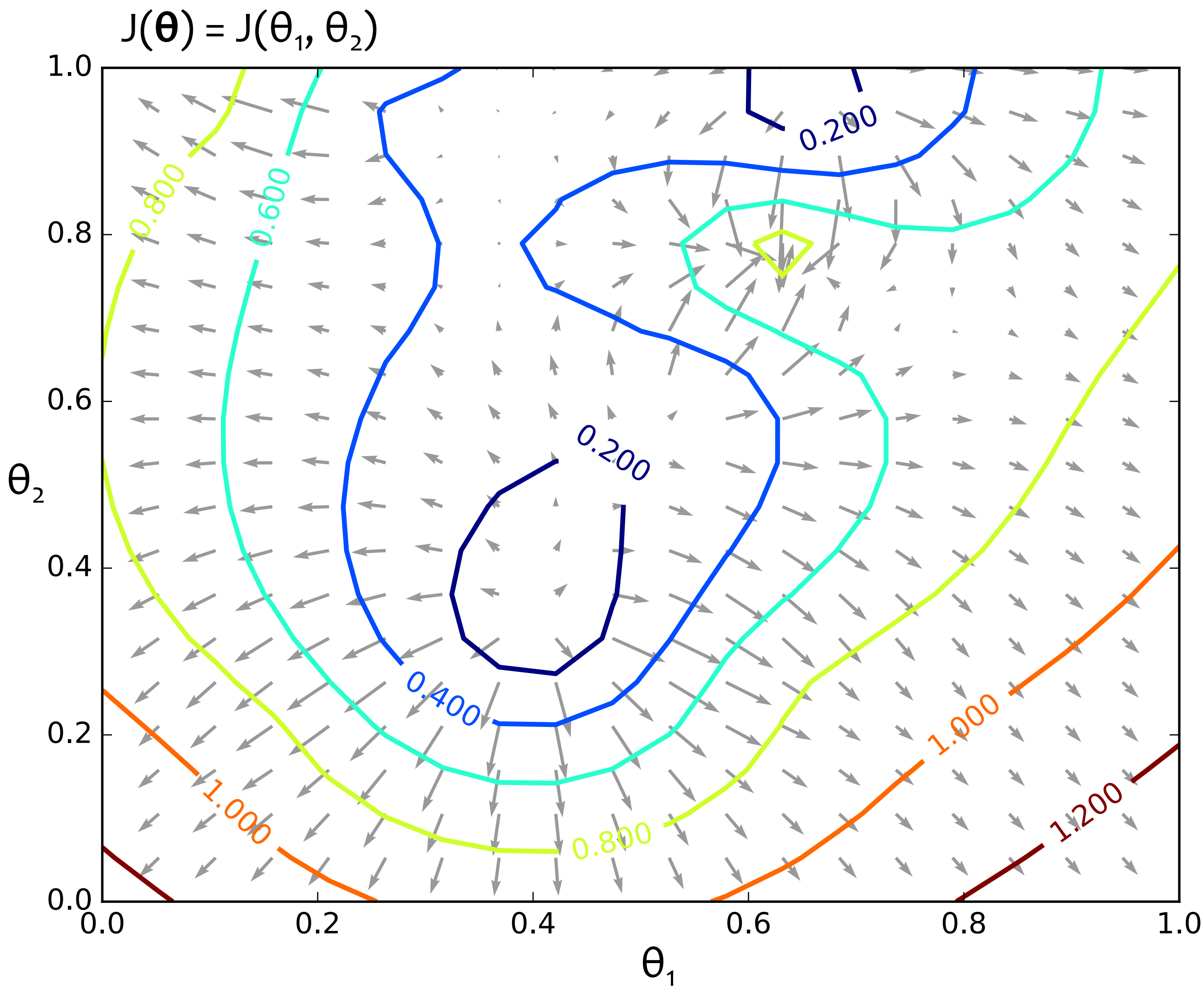


Gradients



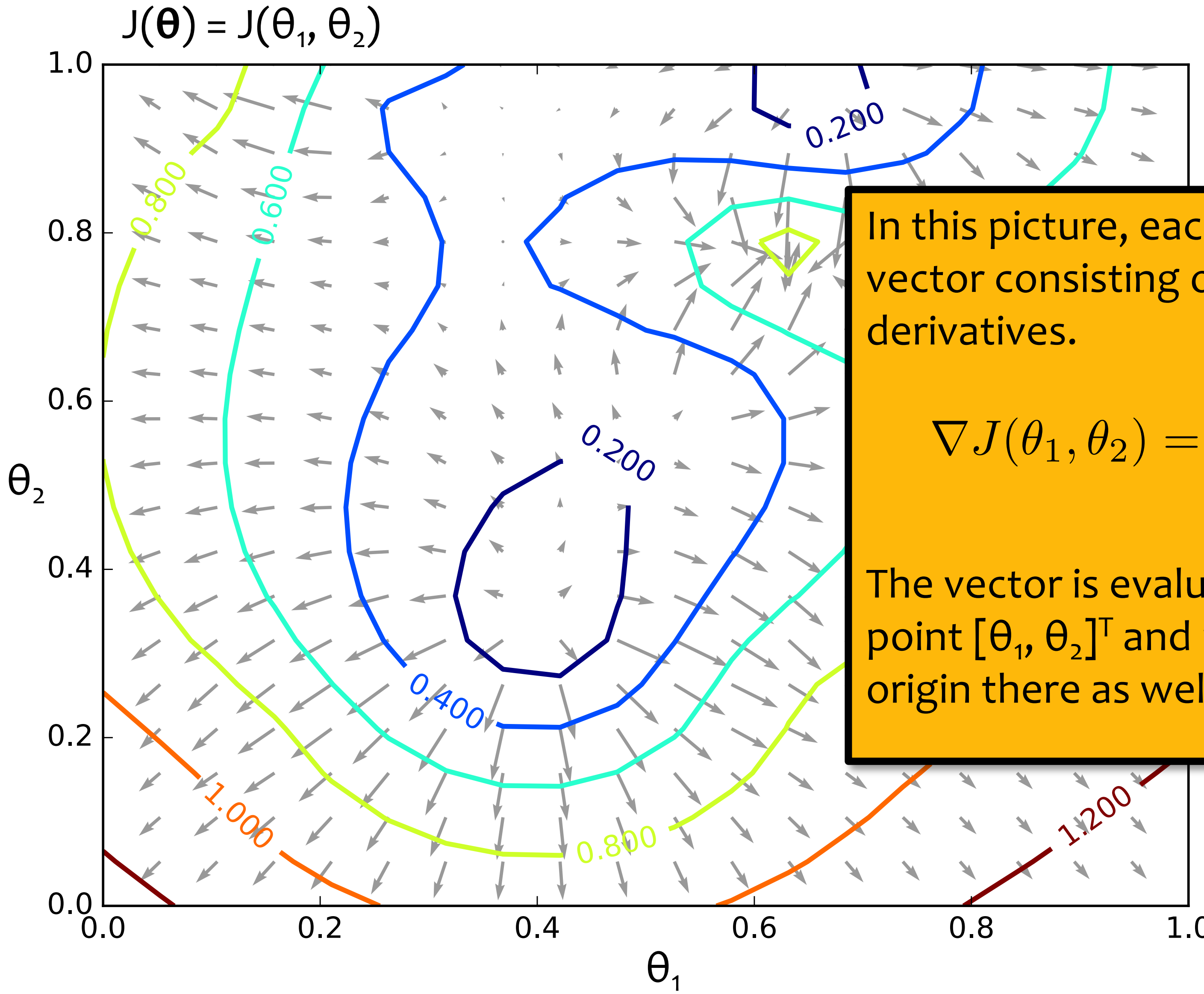
J(**θ**) = J(θ1, θ2)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
θ2
θ1
0.200
0.200
0.400
0.600
0.800
0.800
1.000
1.000
1.200

Gradients



These are the **gradients** that
Gradient **Ascent** would follow.

Gradients



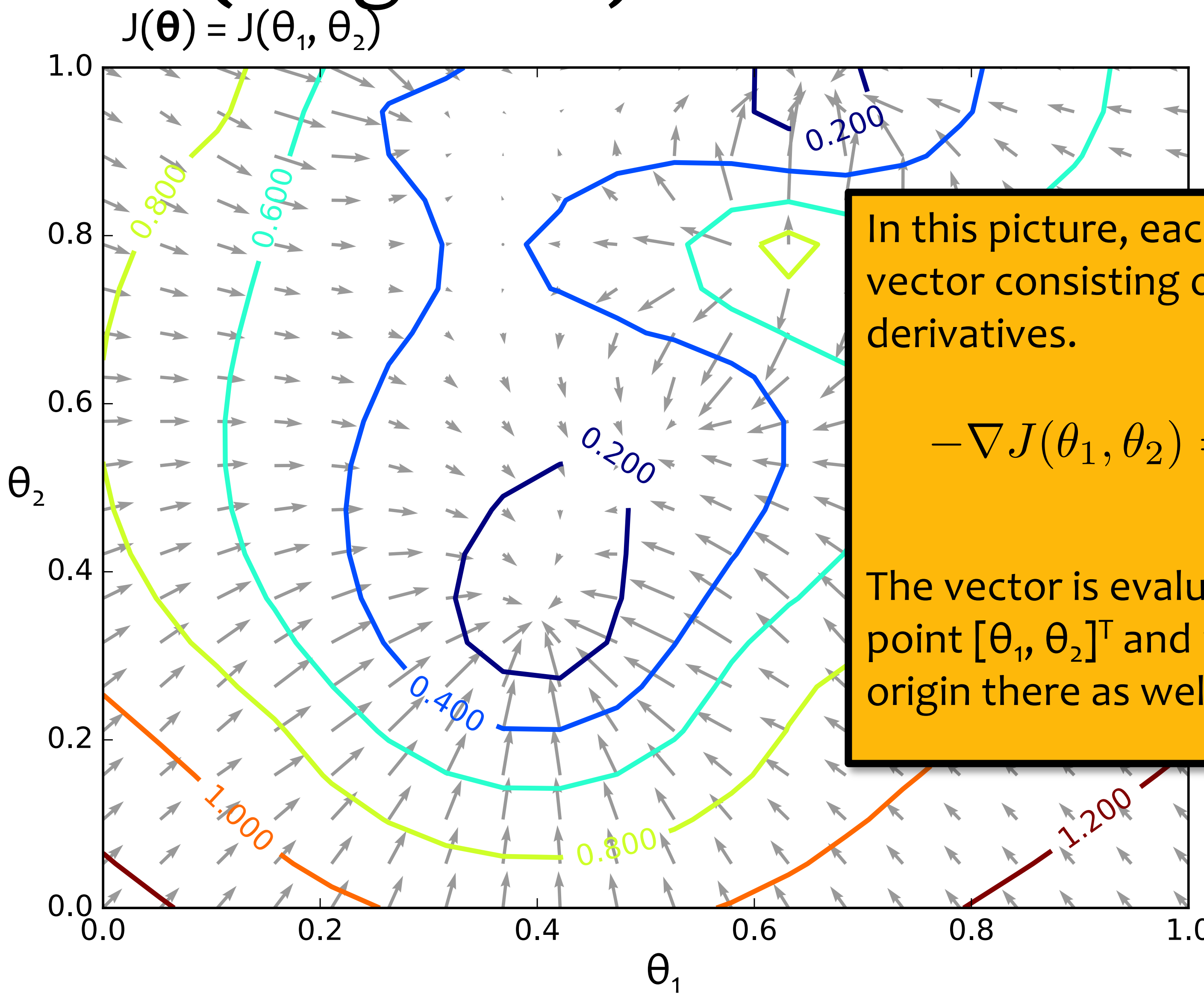
In this picture, each arrow is a 2D vector consisting of two partial derivatives.

$$\nabla J(\theta_1, \theta_2) = \begin{bmatrix} \frac{\partial J}{\partial \theta_1} \\ \\ \frac{\partial J}{\partial \theta_2} \end{bmatrix}$$

The vector is evaluated at the point $[\theta_1, \theta_2]^T$ and plotted with its origin there as well.

These are the **gradients** that Gradient **Ascent** would follow.

(Negative) Gradients



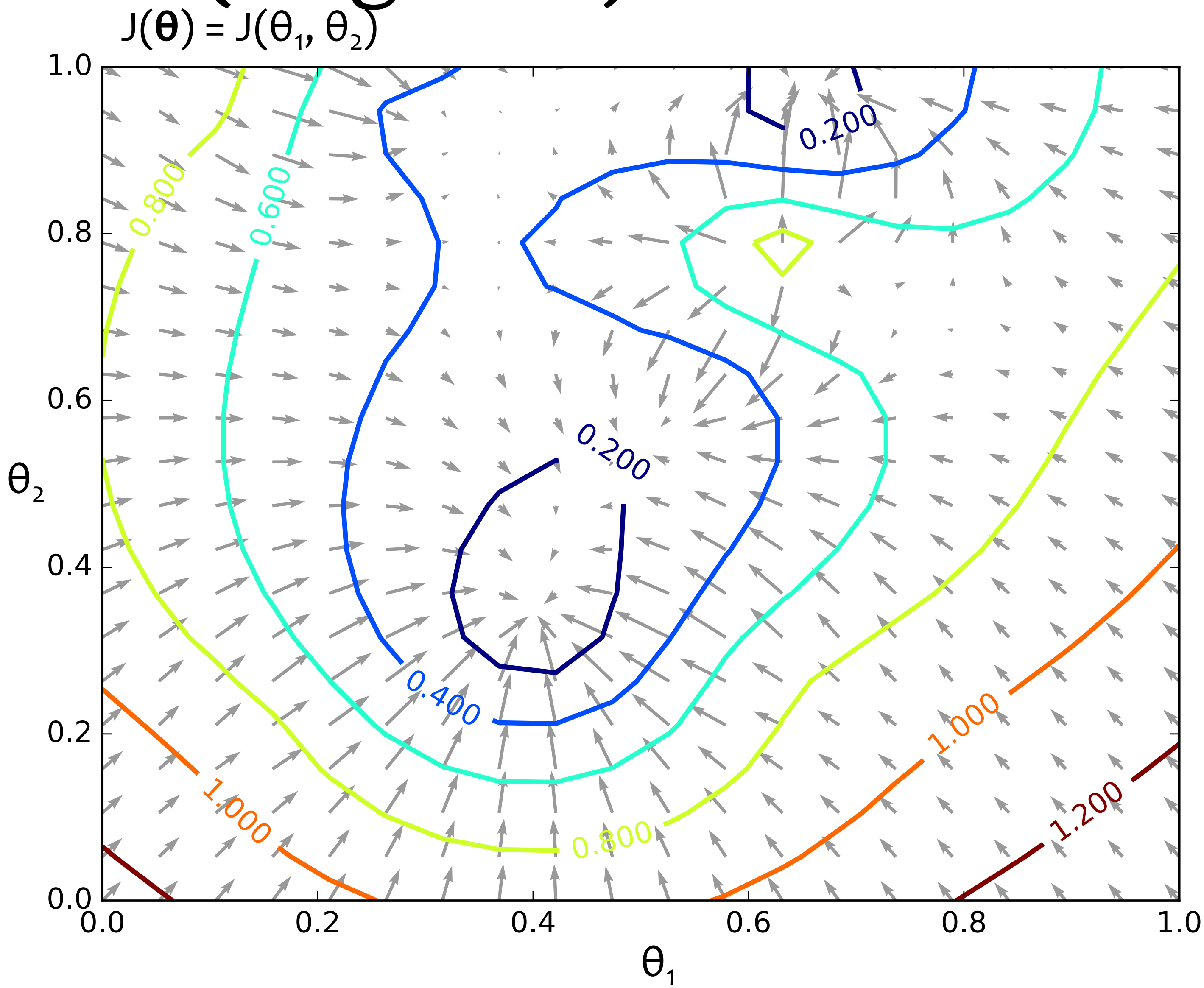
In this picture, each arrow is a 2D vector consisting of two partial derivatives.

$$-\nabla J(\theta_1, \theta_2) = \begin{bmatrix} -\frac{\partial J}{\partial \theta_1} \\ -\frac{\partial J}{\partial \theta_2} \end{bmatrix}$$

The vector is evaluated at the point $[\theta_1, \theta_2]^T$ and plotted with its origin there as well.

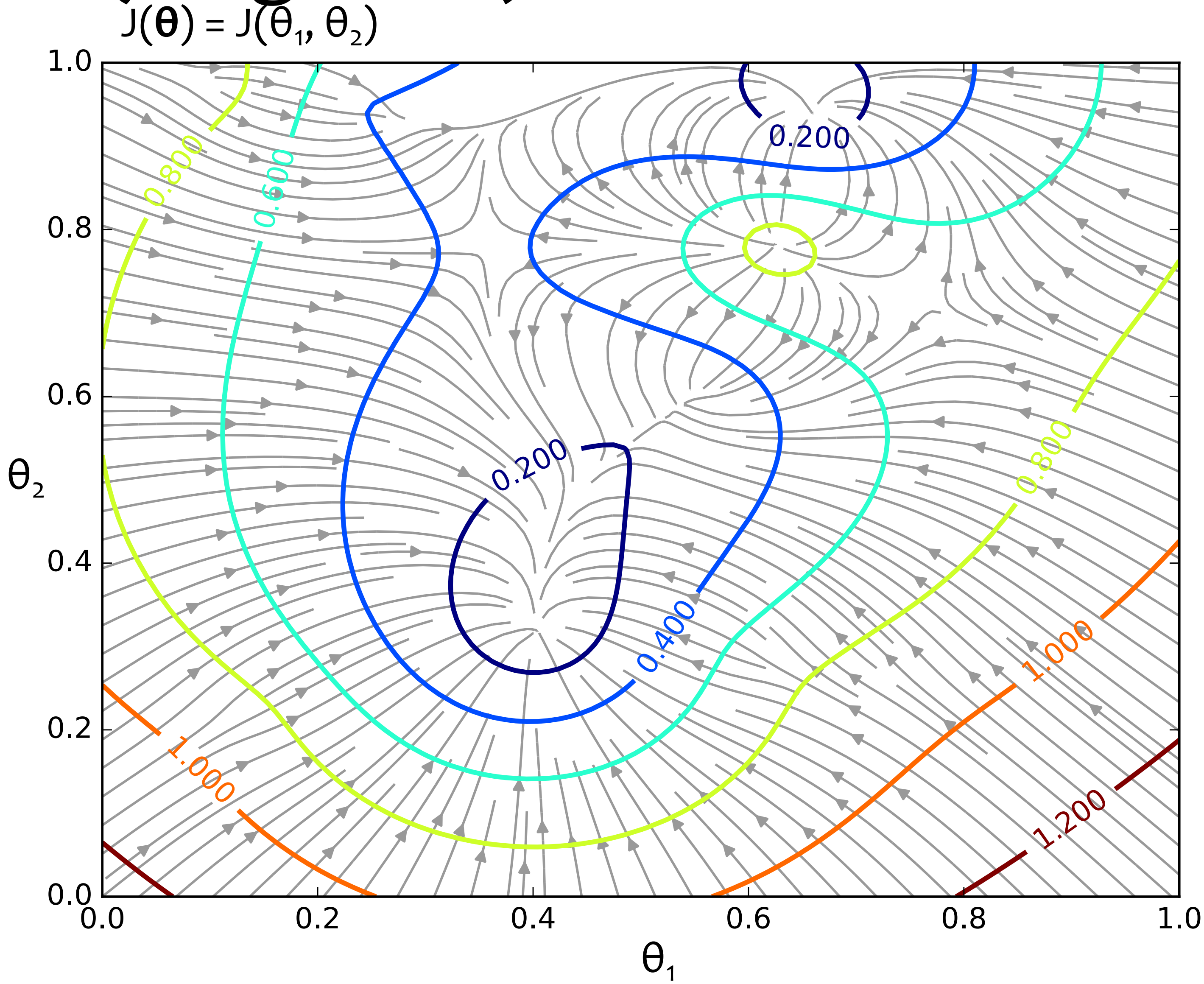
These are the **negative** gradients that Gradient **Descent** would follow.

(Negative) Gradients



These are the **negative** gradients that Gradient **Descent** would follow.

(Negative) Gradient *Paths*



Shown are the **paths** that Gradient Descent would follow if it were making **infinitesimally small steps.**

Gradient Descent

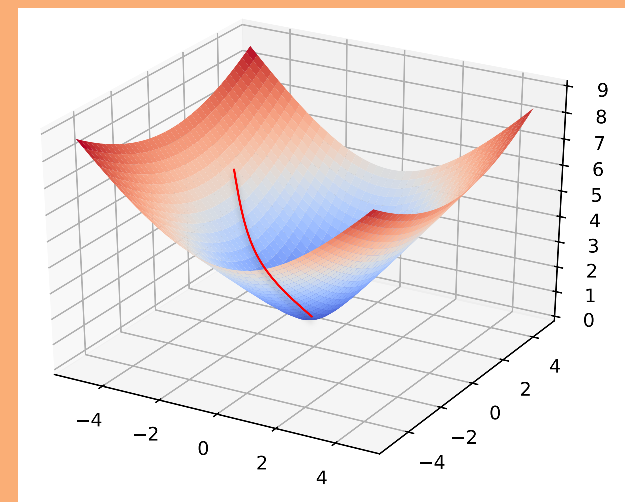
Chalkboard

- Gradient Descent Algorithm
- Details: starting point, stopping criterion, line search

Gradient Descent

Algorithm 1 Gradient Descent

procedure $\text{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)})$
    $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$
    while not converged **do**
        $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$
    return $\boldsymbol{\theta}$



In order to apply GD to Linear Regression all we need is the **gradient** of the objective function (i.e. vector of partial derivatives).

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1} J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2} J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_M} J(\boldsymbol{\theta}) \end{bmatrix}$$

Gradient Descent

Algorithm 1 Gradient Descent

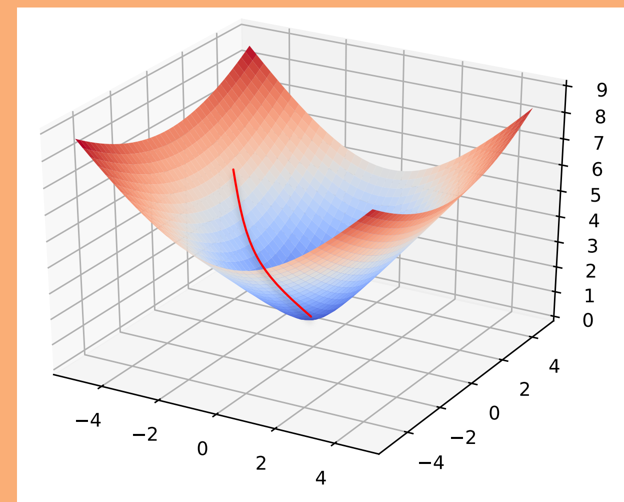
1: **procedure** $\text{GD}(\mathcal{D}, \boldsymbol{\theta}^{(0)})$

2: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$

3: **while** not converged **do**

4: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$

5: **return** $\boldsymbol{\theta}$



There are many possible ways to detect **convergence.** For example, we could check whether the L2 norm of the gradient is below some small tolerance.

$$||\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})||_2 \leq \epsilon$$

Alternatively we could check that the reduction in the objective function from one iteration to the next is small.

GRADIENT DESCENT FOR LINEAR REGRESSION

Linear Regression as Function Approximation

$\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N$
where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \mathbb{R}$

1. Assume $\mathcal{D}$ generated as:

$$\mathbf{x}^{(i)} \sim p^*(\cdot)$$
$$y^{(i)} = h^*(\mathbf{x}^{(i)})$$

2. Choose hypothesis space, $\mathcal{H}$:
all linear functions in M-dimensional space

$$\mathcal{H} = \{h_{\boldsymbol{\theta}} : h_{\boldsymbol{\theta}}(\mathbf{x}) = \boldsymbol{\theta}^T \mathbf{x}, \boldsymbol{\theta} \in \mathbb{R}^M\}$$

3. Choose an objective function:
mean squared error (MSE)

$$J(\boldsymbol{\theta}) = \frac{1}{N} \sum_{i=1}^N e_i^2$$
$$= \frac{1}{N} \sum_{i=1}^N \left(y^{(i)} - h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})\right)^2$$
$$= \frac{1}{N} \sum_{i=1}^N \left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)^2$$

4. Solve the unconstrained optimization problem via favorite method:
   - *gradient descent*
   - *closed form*
   - *stochastic gradient descent*
   - ...

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\text{argmin}}\, J(\boldsymbol{\theta})$$

5. Test time: given a new $\mathbf{x}$, make prediction $\hat{y}$

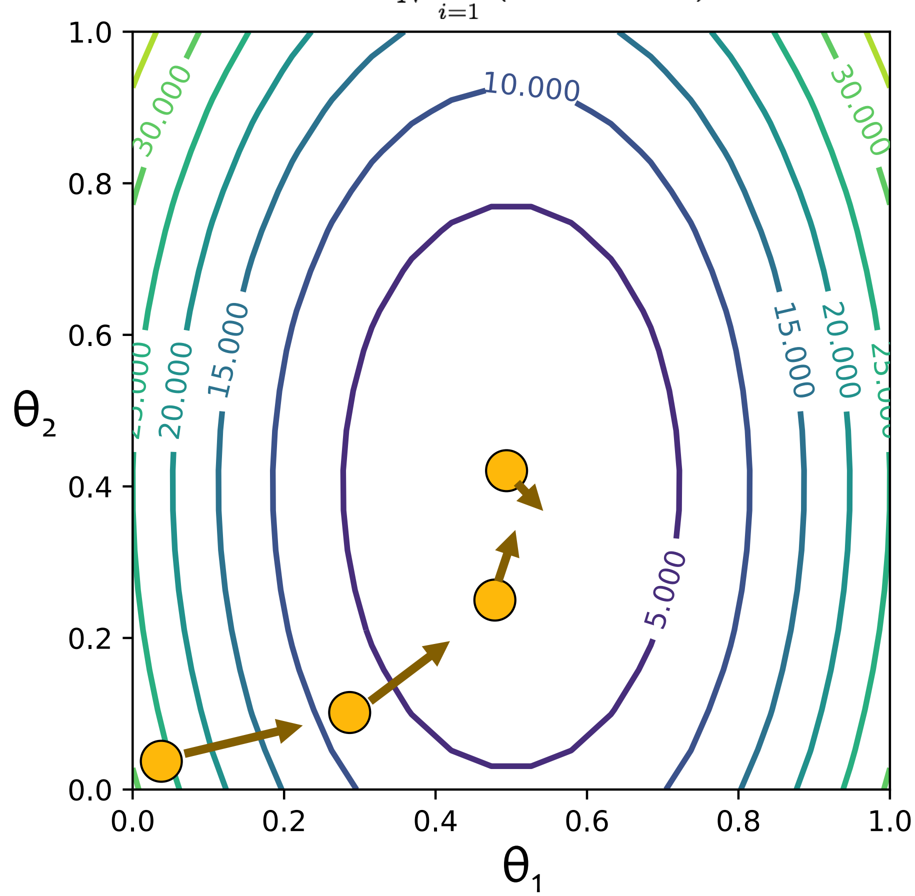
$$\hat{y} = h_{\hat{\boldsymbol{\theta}}}(\mathbf{x}) = \hat{\boldsymbol{\theta}}^T \mathbf{x}$$

Linear Regression by Gradient Desc.

Optimization Method #1: Gradient Descent

1. Pick a random $\boldsymbol{\theta}$
2. Repeat:
   a. Evaluate gradient $\nabla J(\boldsymbol{\theta})$
   b. Step opposite gradient
3. Return $\boldsymbol{\theta}$ that gives smallest $J(\boldsymbol{\theta})$

$$J(\boldsymbol{\theta}) = J(\theta_1, \theta_2) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)^2$$

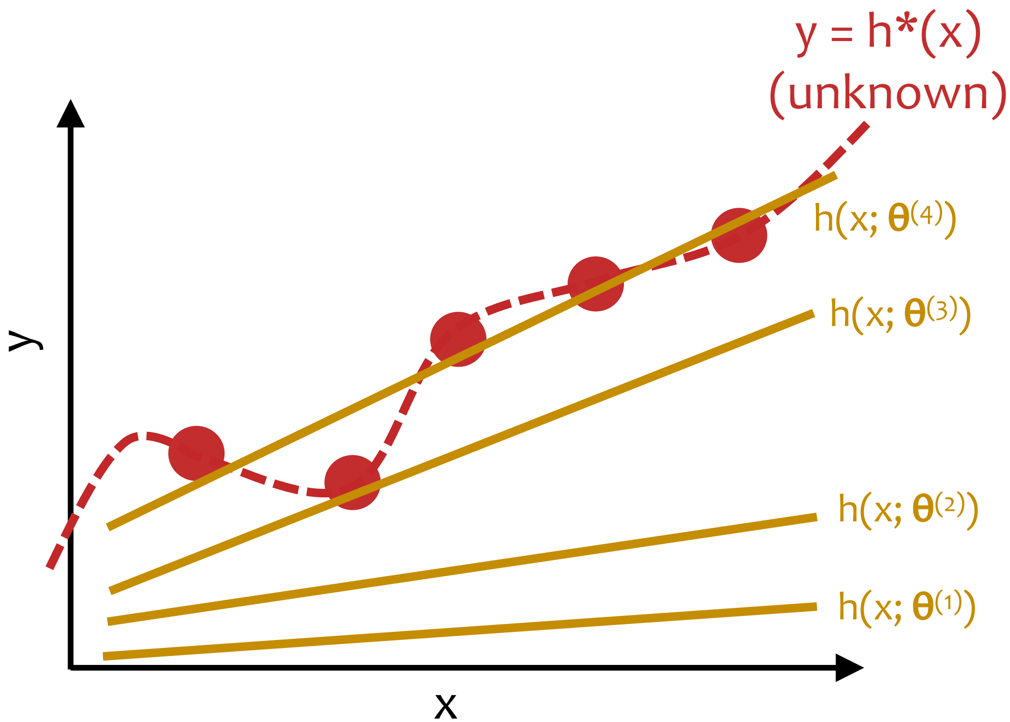


t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.01	0.02	25.2
2	0.30	0.12	8.7
3	0.51	0.30	1.5
4	0.59	0.43	0.2

Linear Regression by Gradient Desc.

Optimization Method #1: Gradient Descent

1. Pick a random $\boldsymbol{\theta}$
2. Repeat:
   a. Evaluate gradient $\nabla J(\boldsymbol{\theta})$
   b. Step opposite gradient
3. Return $\boldsymbol{\theta}$ that gives smallest $J(\boldsymbol{\theta})$



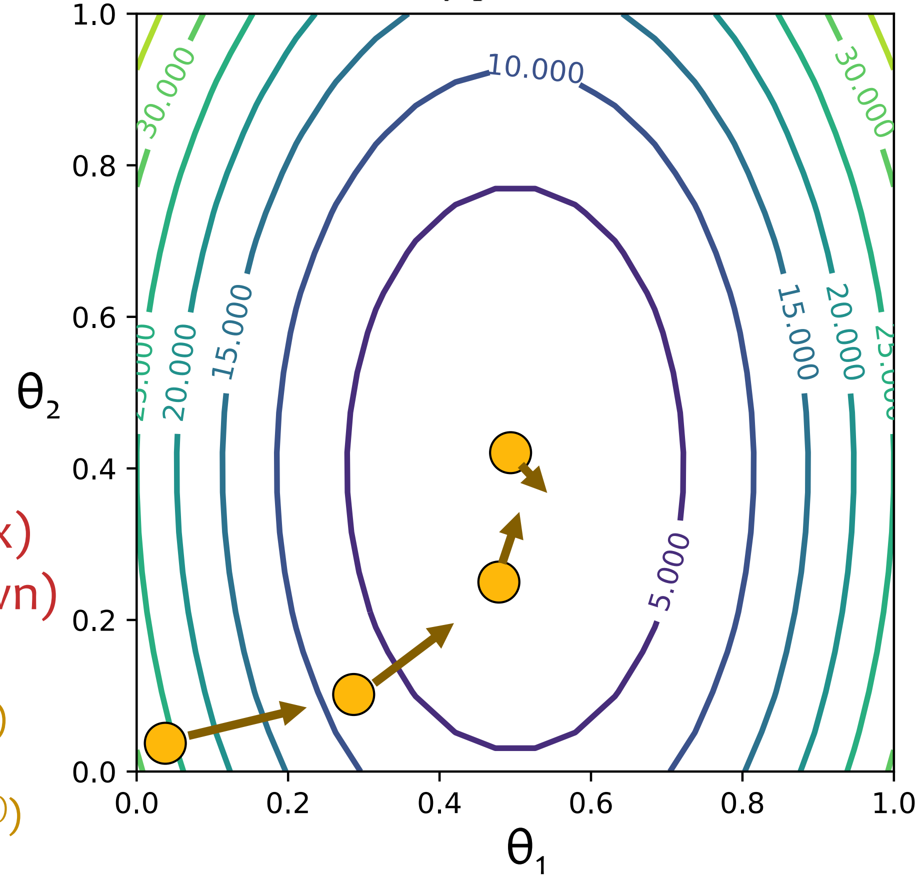
t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.01	0.02	25.2
2	0.30	0.12	8.7
3	0.51	0.30	1.5
4	0.59	0.43	0.2

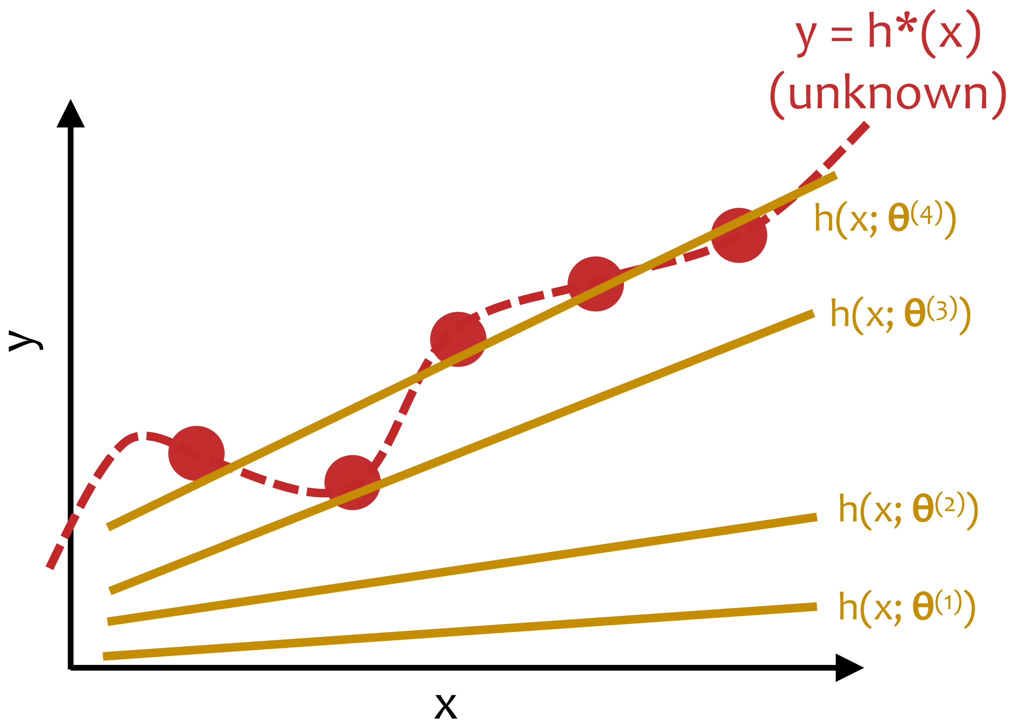
Linear Regression by Gradient Desc.

Optimization Method #1: Gradient Descent

1. Pick a random $\boldsymbol{\theta}$
2. Repeat:
   a. Evaluate gradient $\nabla J(\boldsymbol{\theta})$
   b. Step opposite gradient
3. Return $\boldsymbol{\theta}$ that gives smallest $J(\boldsymbol{\theta})$

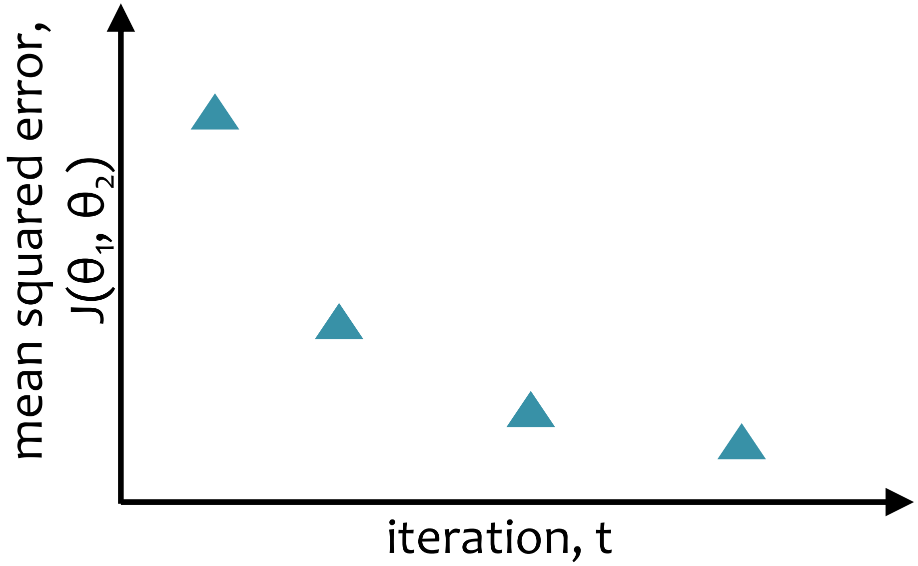
$$J(\boldsymbol{\theta}) = J(\theta_1, \theta_2) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)^2$$

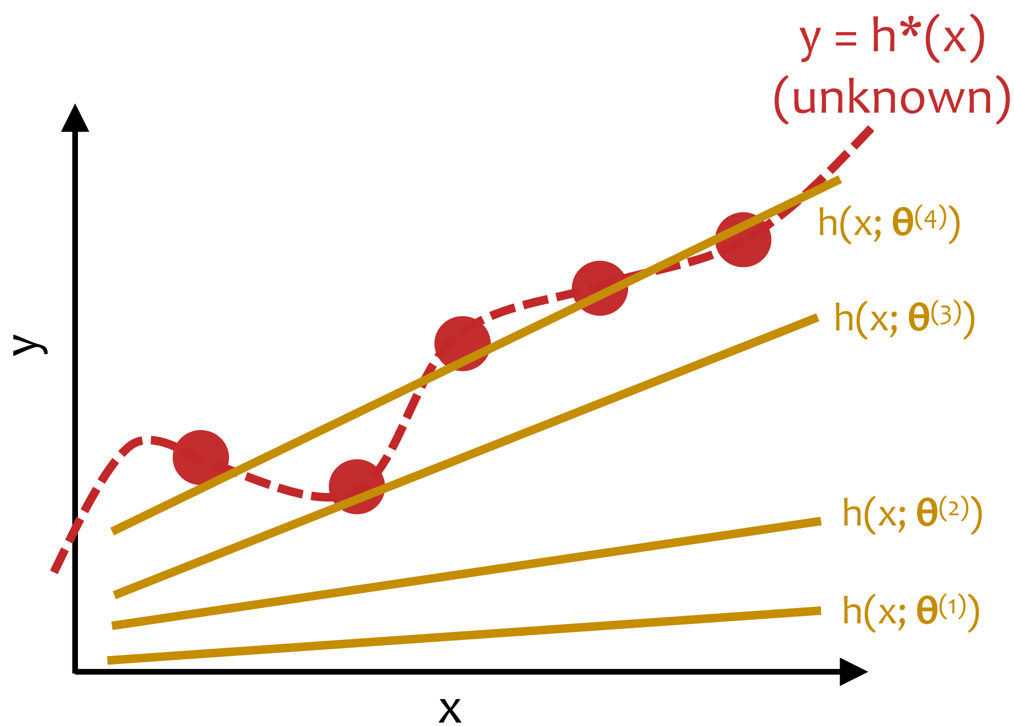




t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.01	0.02	25.2
2	0.30	0.12	8.7
3	0.51	0.30	1.5
4	0.59	0.43	0.2

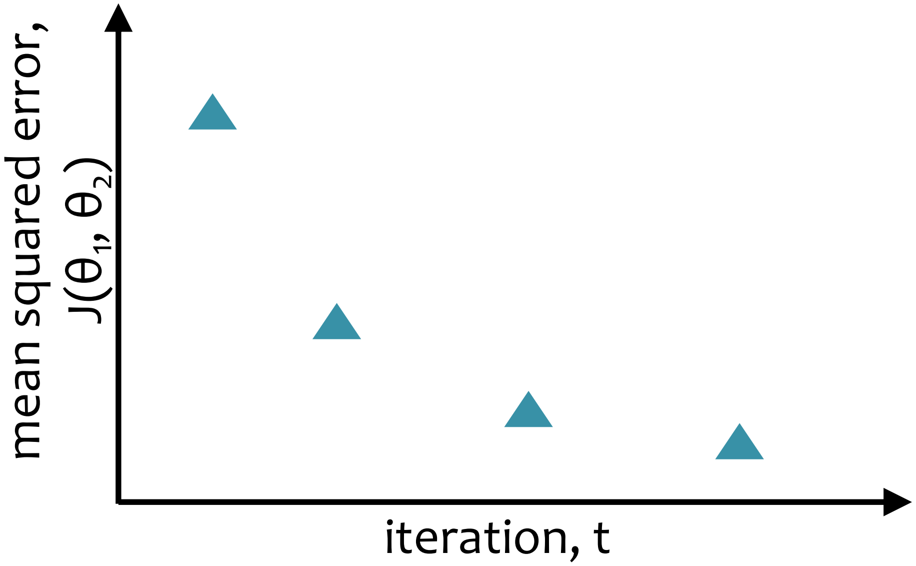
Linear Regression by Gradient Desc.

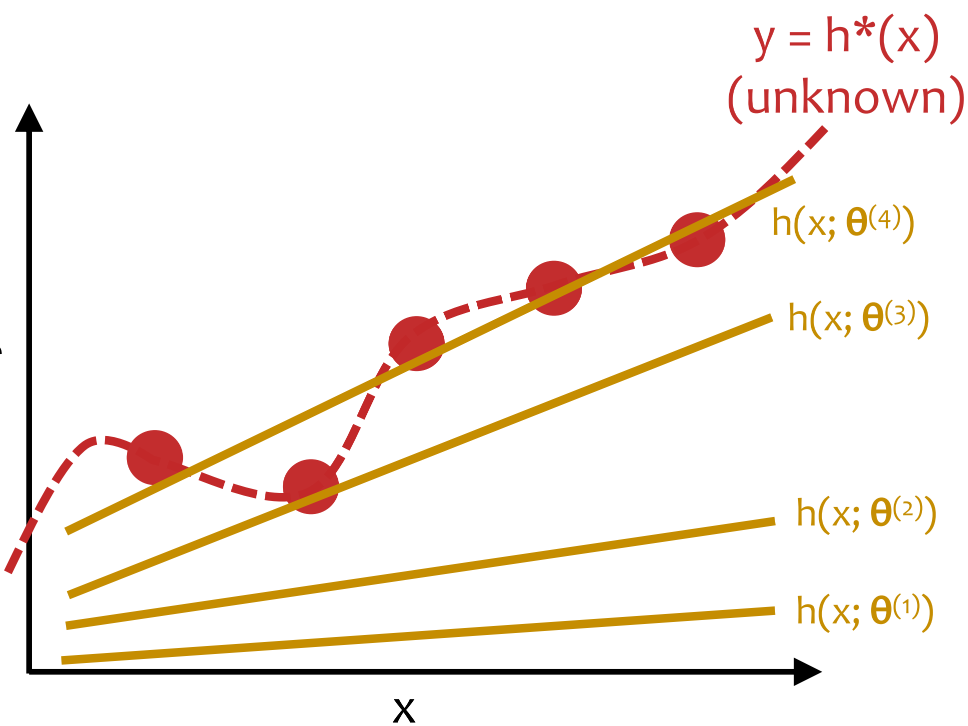




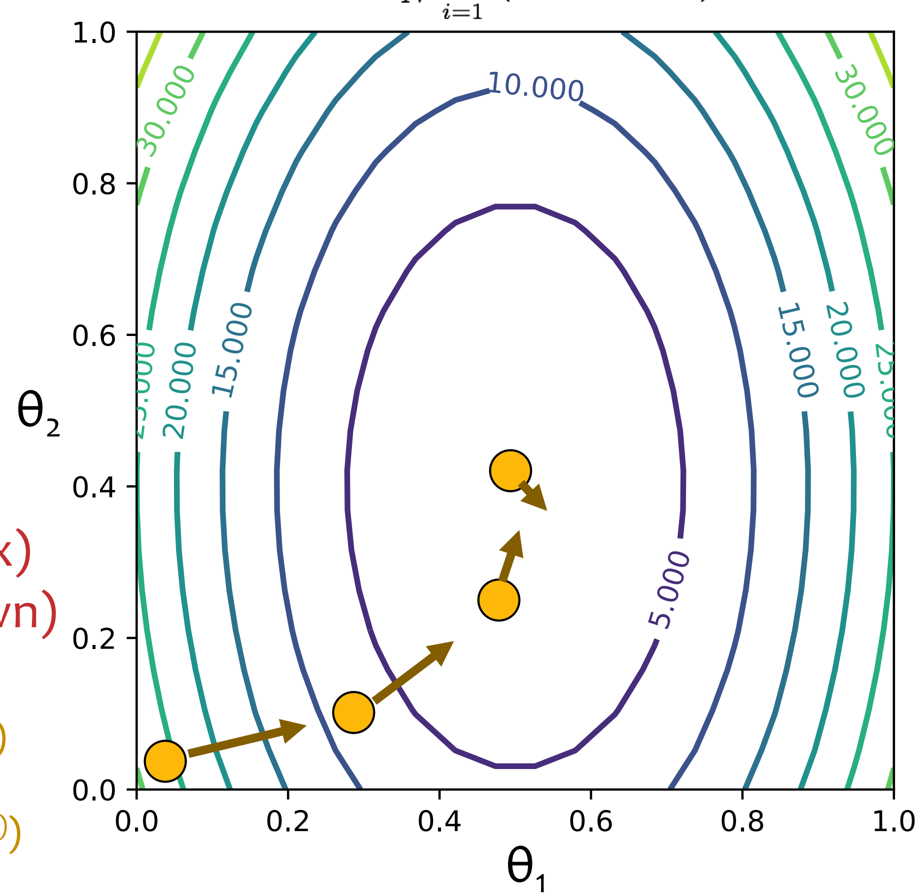
t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.01	0.02	25.2
2	0.30	0.12	8.7
3	0.51	0.30	1.5
4	0.59	0.43	0.2

Linear Regression by Gradient Desc.





$$J(\boldsymbol{\theta}) = J(\theta_1, \theta_2) = \frac{1}{N}\sum_{i=1}^{N}\left(y^{(i)} - \boldsymbol{\theta}^T \mathbf{x}^{(i)}\right)^2$$



t	θ_1	θ_2	$J(\theta_1, \theta_2)$
1	0.01	0.02	25.2
2	0.30	0.12	8.7
3	0.51	0.30	1.5
4	0.59	0.43	0.2

Optimization for Linear Regression

Chalkboard

- Computing the gradient for Linear Regression
- Gradient Descent for Linear Regression

Gradient Calculation for Linear Regression

Derivative of $J^{(i)}(\boldsymbol{\theta})$:

$$\frac{d}{d\theta_k} J^{(i)}(\boldsymbol{\theta}) = \frac{d}{d\theta_k}\frac{1}{2}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})^2$$
$$= \frac{1}{2}\frac{d}{d\theta_k}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})^2$$
$$= (\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})\frac{d}{d\theta_k}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})$$
$$= (\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})\frac{d}{d\theta_k}\left(\sum_{j=1}^{K}\theta_j x_j^{(i)} - y^{(i)}\right)$$
$$= (\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})x_k^{(i)}$$

Derivative of $J(\boldsymbol{\theta})$:

$$\frac{d}{d\theta_k} J(\boldsymbol{\theta}) = \sum_{i=1}^{N}\frac{d}{d\theta_k}J^{(i)}(\boldsymbol{\theta})$$
$$= \sum_{i=1}^{N}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})x_k^{(i)}$$

Gradient of $J(\boldsymbol{\theta})$ [used by Gradient Descent]

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \begin{bmatrix} \frac{d}{d\theta_1}J(\boldsymbol{\theta}) \\ \frac{d}{d\theta_2}J(\boldsymbol{\theta}) \\ \vdots \\ \frac{d}{d\theta_M}J(\boldsymbol{\theta}) \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{N}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})x_1^{(i)} \\ \sum_{i=1}^{N}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})x_2^{(i)} \\ \vdots \\ \sum_{i=1}^{N}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})x_N^{(i)} \end{bmatrix}$$
$$= \sum_{i=1}^{N}(\boldsymbol{\theta}^T\mathbf{x}^{(i)} - y^{(i)})\mathbf{x}^{(i)}$$

GD for Linear Regression

Gradient Descent for Linear Regression repeatedly takes steps opposite the gradient of the objective function

Algorithm 1 GD for Linear Regression

1: **procedure** GDLR($\mathcal{D}$, $\boldsymbol{\theta}^{(0)}$)
2: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta}^{(0)}$ ▷ Initialize parameters
3: **while** not converged **do**
4: $\mathbf{g} \leftarrow \sum_{i=1}^{N} (\boldsymbol{\theta}^T \mathbf{x}^{(i)} - y^{(i)}) \mathbf{x}^{(i)}$ ▷ Compute gradient
5: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \gamma \mathbf{g}$ ▷ Update parameters
6: **return** $\boldsymbol{\theta}$