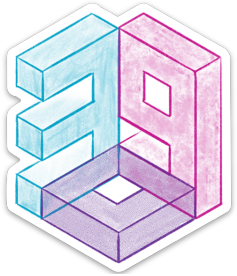


https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 3: Gradient Descent Methods

Shen Shen

Sept 18, 2025

11am, Room 10-250

Interactive Slides and Lecture Recording

Logistics

1. There is a student in our class who needs copies of class notes as an approved accommodation. If you're interested in serving as a paid note taker, please reach out to DAS, at 617-253-1674 or das-student@mit.edu.

2. **Midterm 1: October 8, 730pm-9pm**. It covers all the materials up to and including week 4 (linear classification). If you need to take the conflict or accommodation exam, please get in touch with us at 6.390-personal@mit.edu by **Sept 24**.

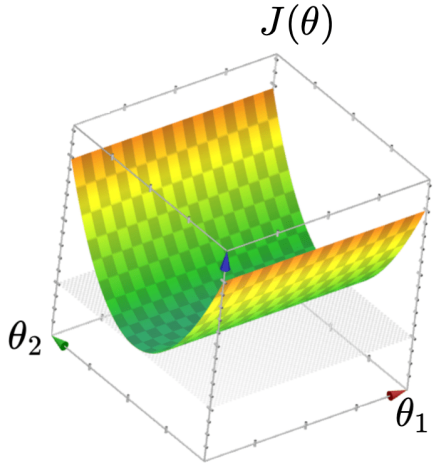
3. Heads-up: Midterm 2 is November 12, 730pm-9pm. Final is December 15, 9am-12pm.

More details on introML homepage

Outline

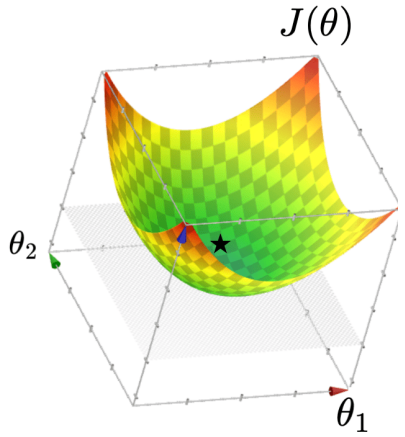
- Gradient descent (GD)
  - The gradient vector
  - GD algorithm
  - Gradient decent properties
- Stochastic gradient descent (SGD)
  - SGD algorithm and setup
  - SGD vs. GD

`Recall:`



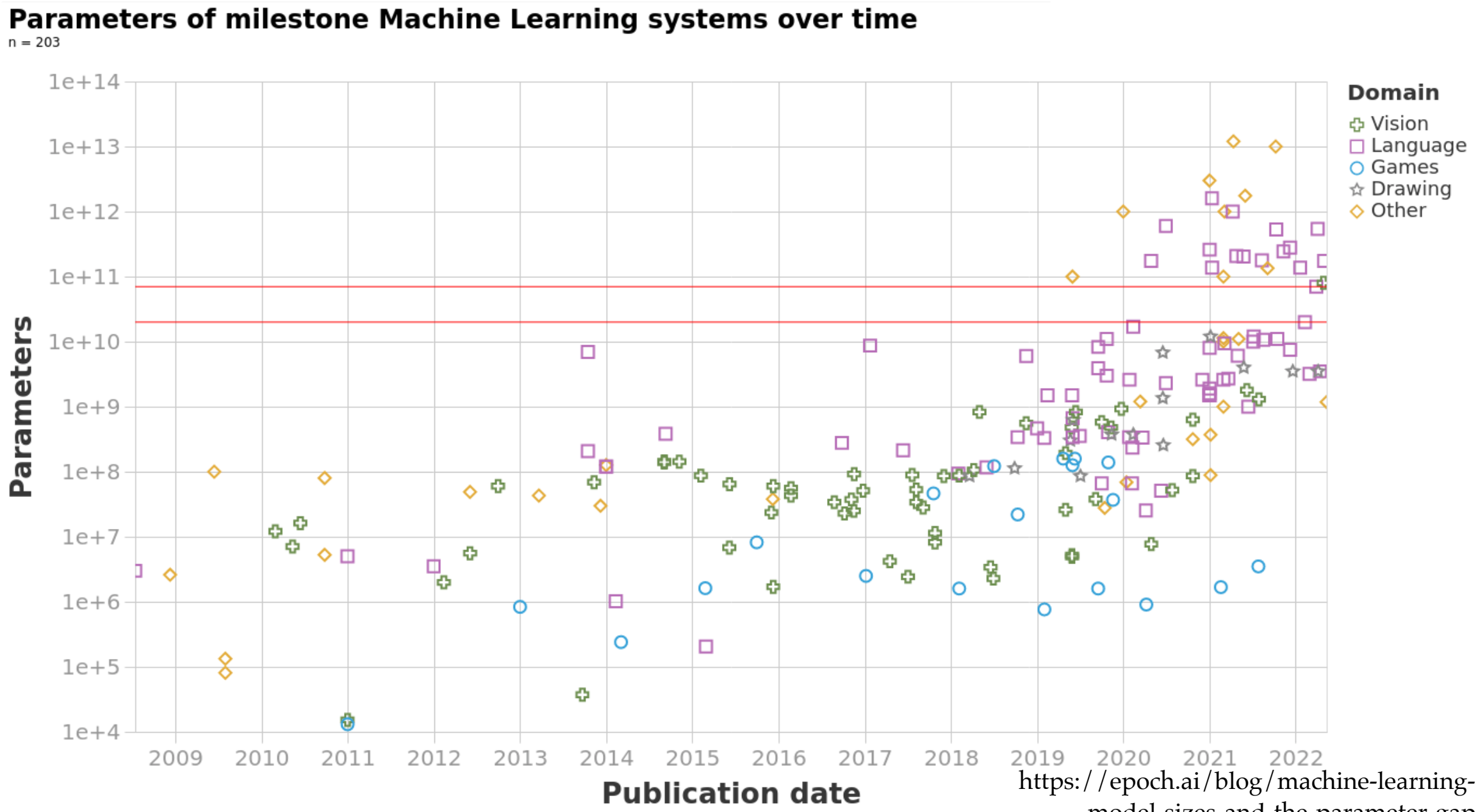
When X is not full column rank

- $J(\theta)$ has a "flat" bottom, like a half pipe
- This 👉 formula is not well-defined
- Infinitely many optimal hyperplanes
- No way yet to obtain an optimal parameter



Typically, X is full column rank

- $J(\theta)$ "curves up" everywhere
- $\theta^* = \left(X^\top X\right)^{-1} X^\top Y$
- θ^* gives the unique optimal hyperplane
- θ^* can be costly to compute (lab2, Q2.7)



https://epoch.ai/blog/machine-learning-model-sizes-and-the-parameter-gap

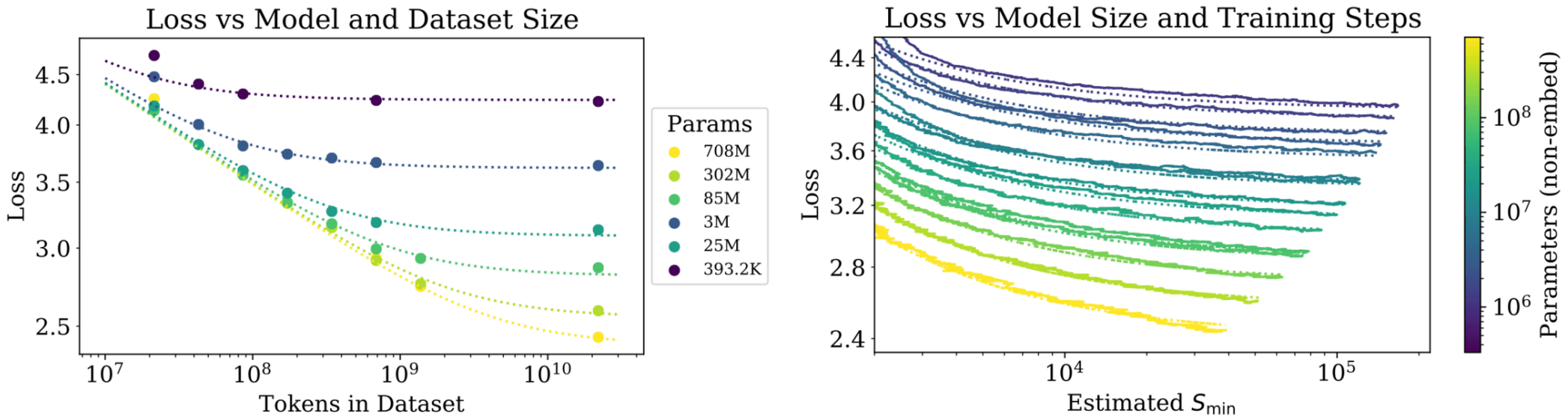


Figure 4 **Left**: The early-stopped test loss $L(N, D)$ varies predictably with the dataset size D and model size N according to Equation (1.5). **Right**: After an initial transient period, learning curves for all model sizes N can be fit with Equation (1.6), which is parameterized in terms of S_{min}, the number of steps when training at large batch size (details in Section 5.1).

https://arxiv.org/pdf/2001.08361

https://losslandscape.com/gallery/

In the real world,

- the number of parameters is huge
- the number of training data points is huge
- hypothesis class is typically highly nonlinear
- loss function is rarely as simple as squared error

Need a more **efficient** and **general** algorithm to train
=> gradient descent methods

Outline

- Gradient descent algorithm (GD)
  - The gradient vector
  - GD algorithm
  - Gradient decent properties
- Stochastic gradient descent (SGD)
  - SGD algorithm and setup
  - SGD vs. GD

For $f : \mathbb{R}^m \to \mathbb{R}$, its ***gradient*** $\nabla f : \mathbb{R}^m \to \mathbb{R}^m$ is defined at the point $p = (x_1, \ldots, x_m)$ as:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_m}(p) \end{bmatrix}$$

1. The gradient generalizes the concept of a derivative to multiple dimensions.
2. By construction, the gradient's dimensionality always matches the function input.

Sometimes the gradient is undefined or ill-behaved, but today it is well-behaved.

3. The gradient can be symbolic or numerical.

example: $f(x, y, z) = x^2 + y^3 + z$

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_m}(p) \end{bmatrix}$$

its *symbolic* gradient:

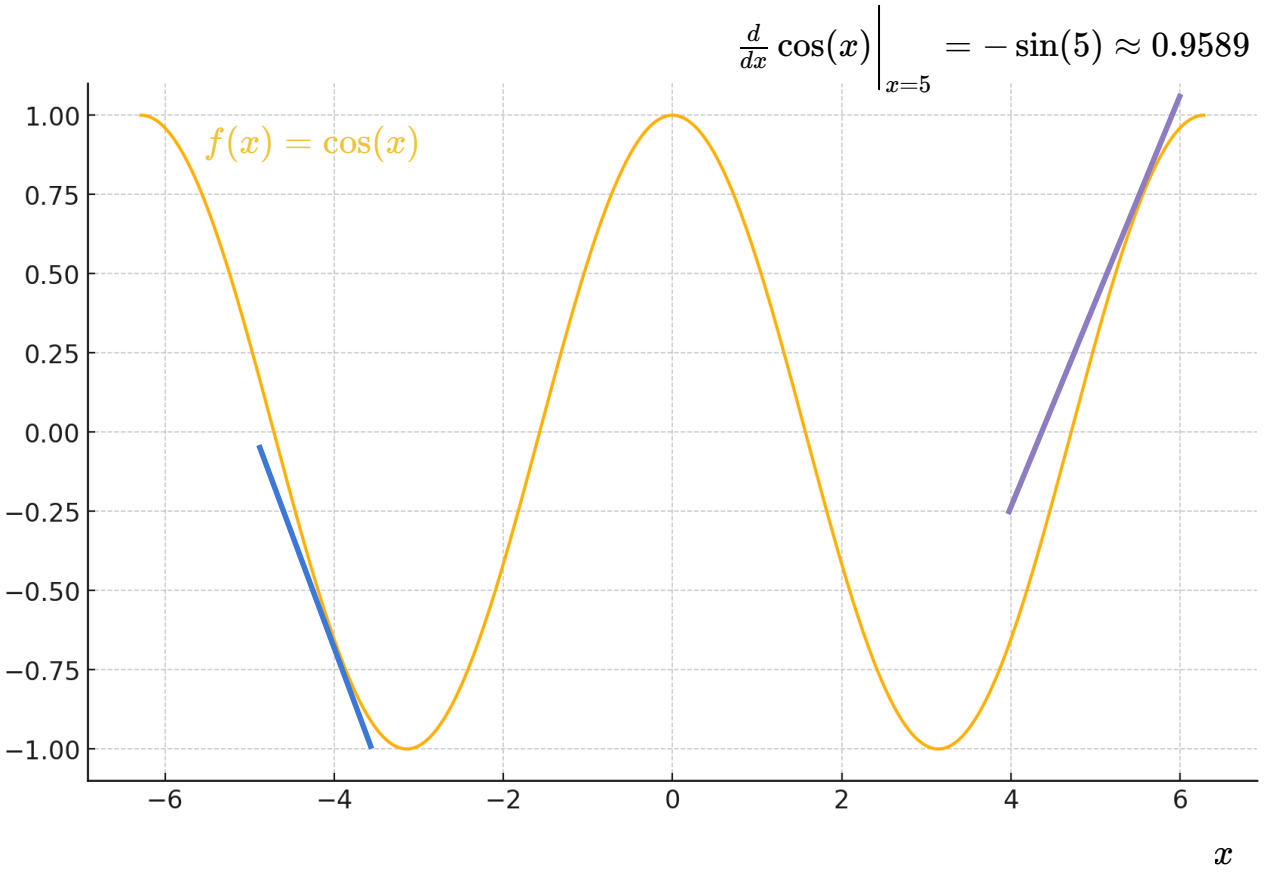
$$\nabla f(x, y, z) = \begin{bmatrix} 2x \\ 3y^2 \\ 1 \end{bmatrix}$$

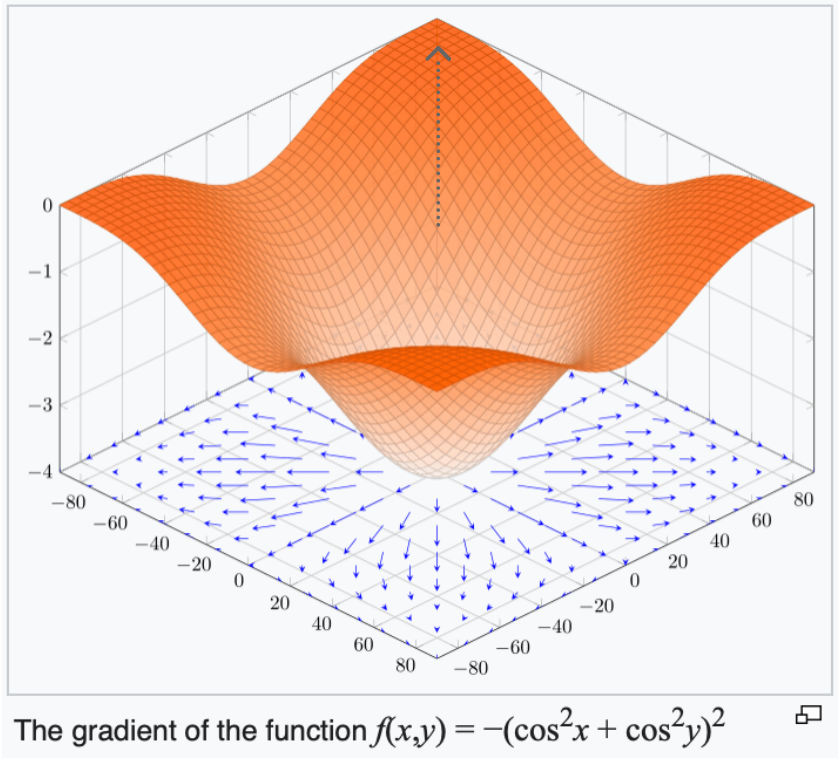
evaluating the symbolic gradient at a point gives a *numerical* gradient:

$$\nabla f(3, 2, 1) = \nabla f(x, y, z)\Big|_{(x,y,z)=(3,2,1)} = \begin{bmatrix} 6 \\ 12 \\ 1 \end{bmatrix}$$

just like a derivative can be a function or a number.

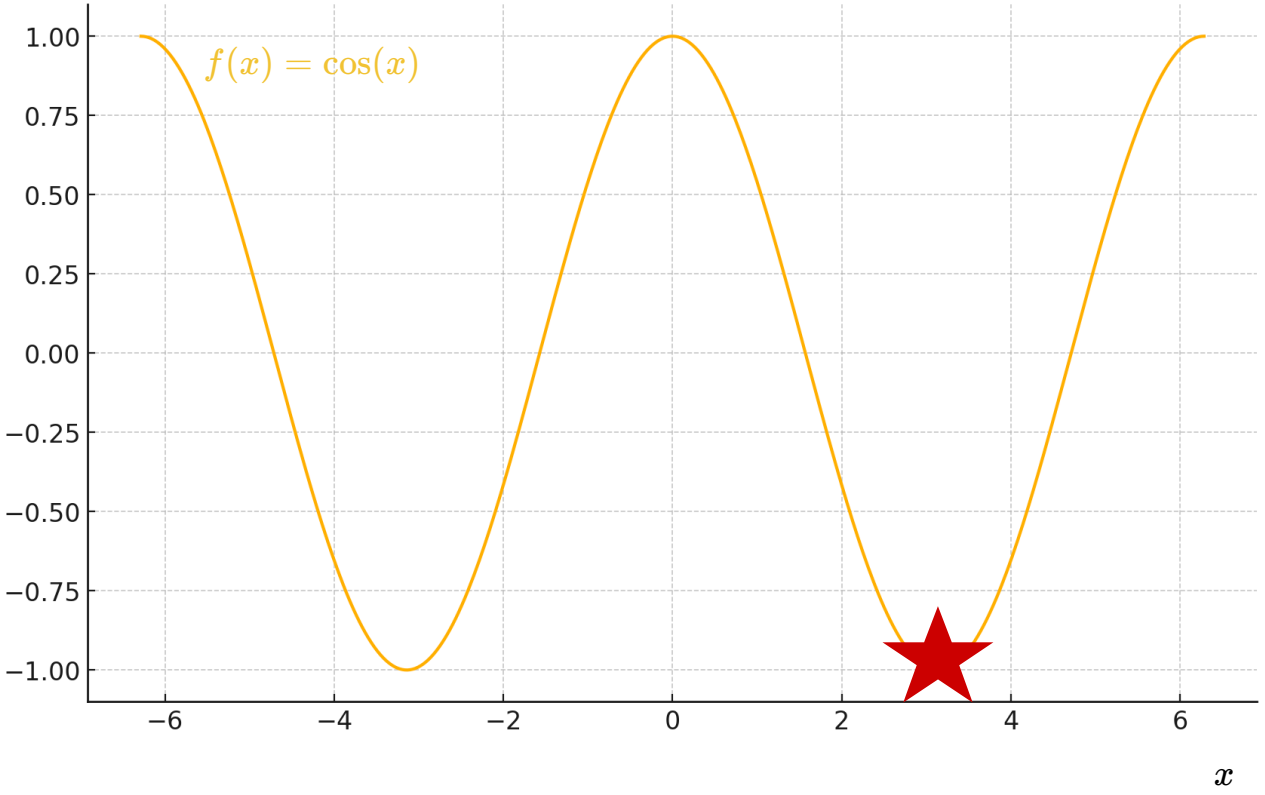
4. The gradient points in the direction of the (steepest) *increase* in the function value.

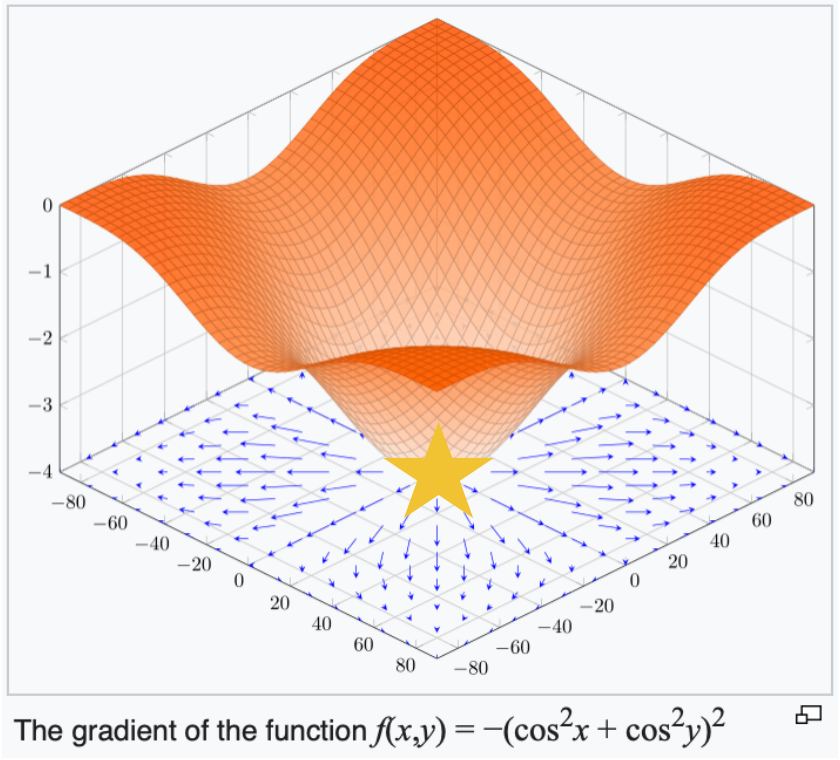




The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$

5. The gradient at the function minimizer is *necessarily* zero.





The gradient of the function $f(x,y) = -(\cos^2 x + \cos^2 y)^2$

For $f : \mathbb{R}^m \to \mathbb{R}$, its ***gradient*** $\nabla f : \mathbb{R}^m \to \mathbb{R}^m$ is defined at the point $p = (x_1, \ldots, x_m)$ as:

$$\nabla f(p) = \begin{bmatrix} \frac{\partial f}{\partial x_1}(p) \\ \vdots \\ \frac{\partial f}{\partial x_m}(p) \end{bmatrix}$$

1. The gradient generalizes the concept of a derivative to multiple dimensions.
2. By construction, the gradient's dimensionality always matches the function input.
3. The gradient can be symbolic or numerical.
4. The gradient points in the direction of the (steepest) *increase* in the function value.
5. The gradient at the function minimizer is *necessarily* zero.

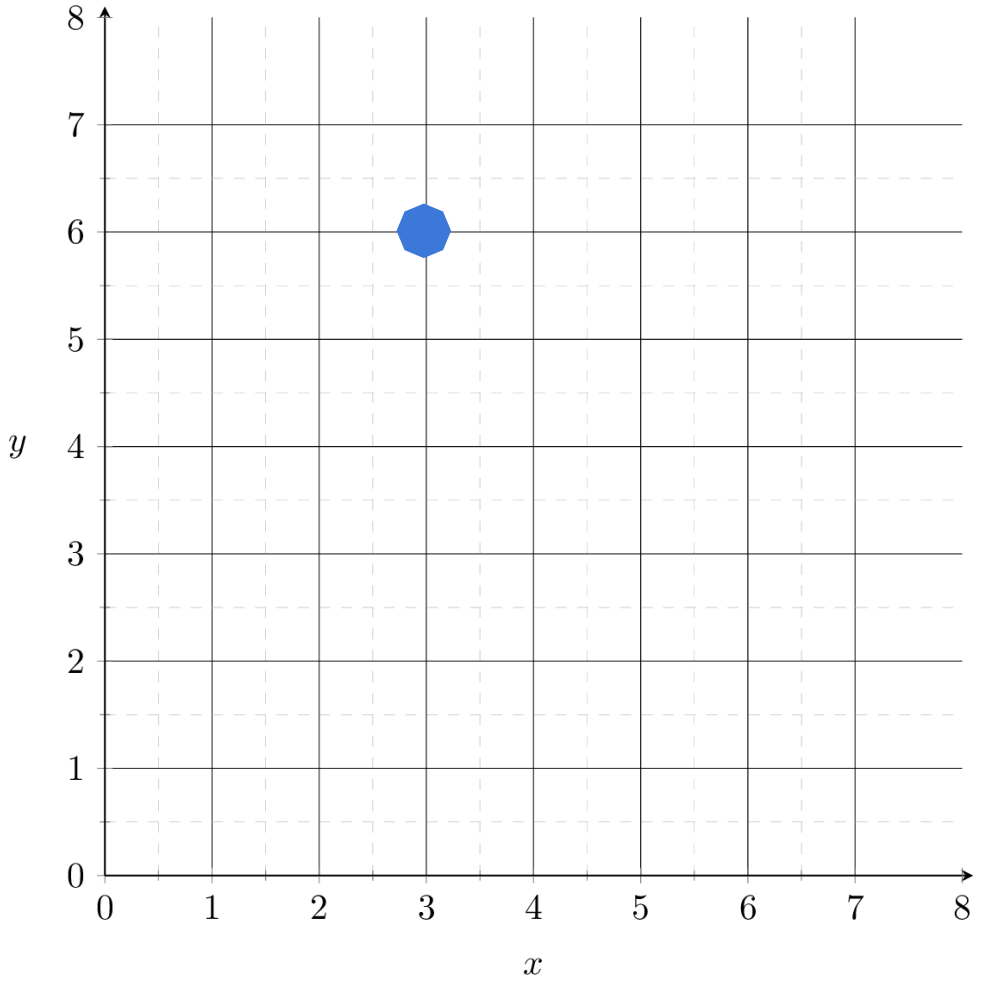
Sometimes the gradient is undefined or ill-behaved, but today it is well-behaved.

Outline

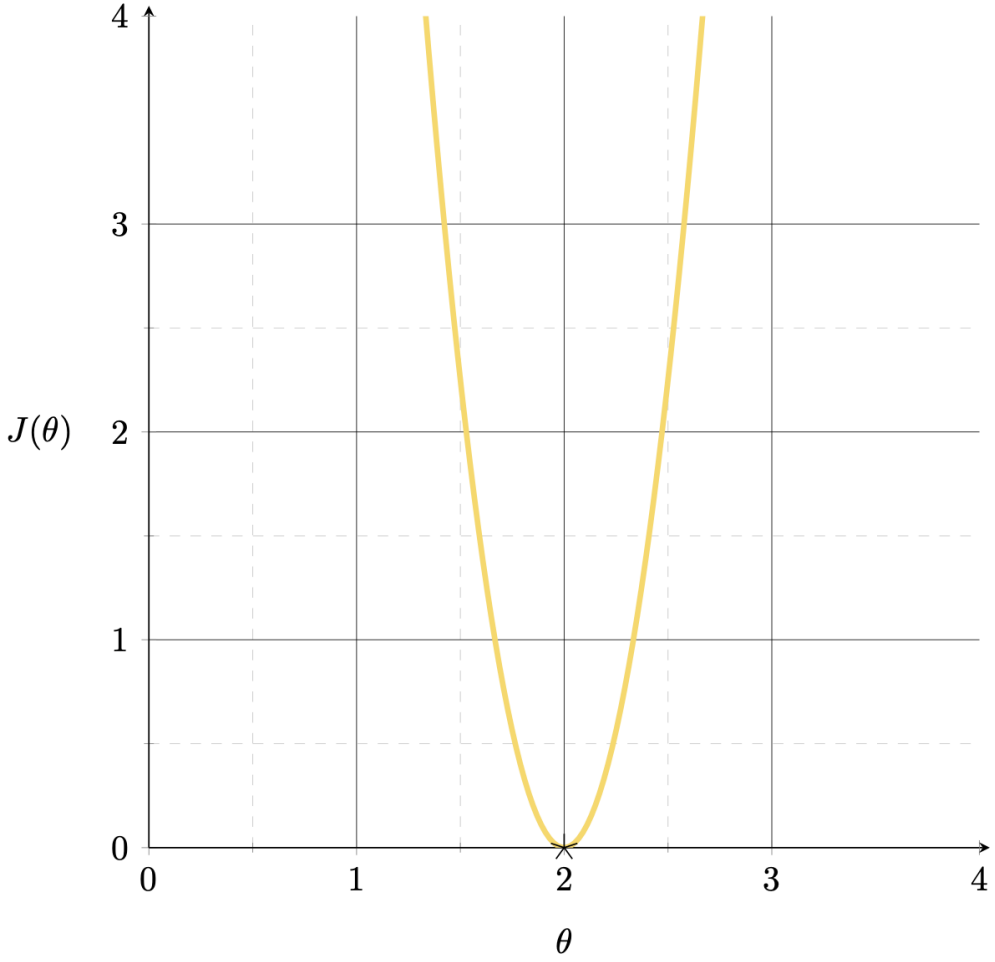
- Gradient descent algorithm (GD)
  - The gradient vector
  - GD algorithm
  - Gradient decent properties
- Stochastic gradient descent (SGD)
  - SGD algorithm and setup
  - SGD vs. GD

A single training data point

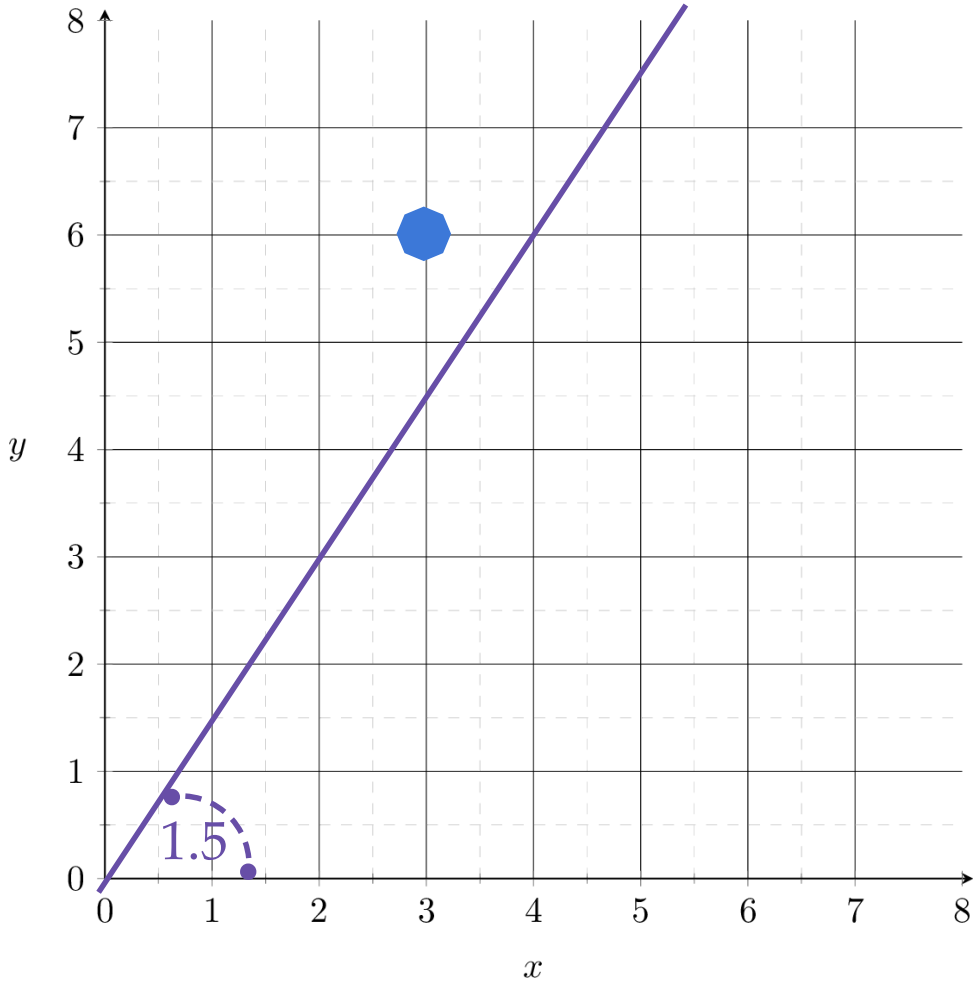
$(x, y) = (3, 6)$



Want to fit a line (without offset) to minimize the MSE: $J(\theta) = (3\theta - 6)^2$

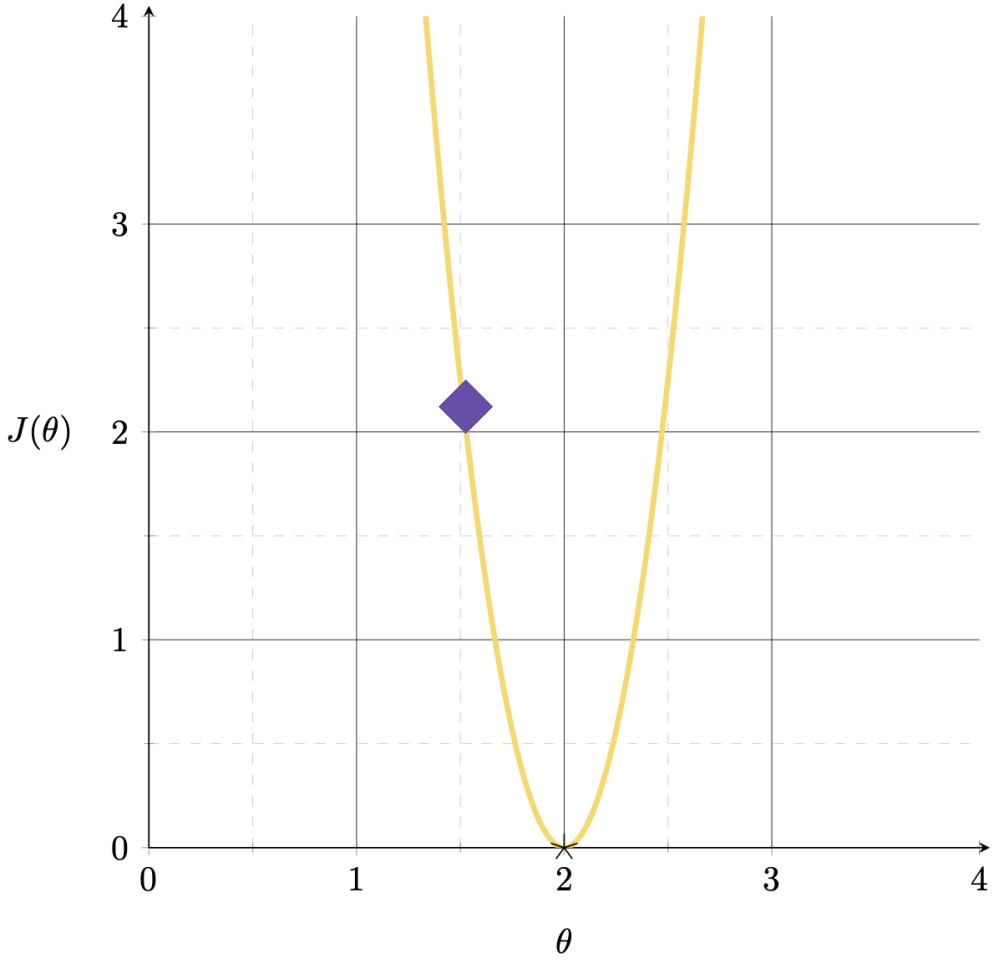


Suppose we fit a line $y = 1.5x$

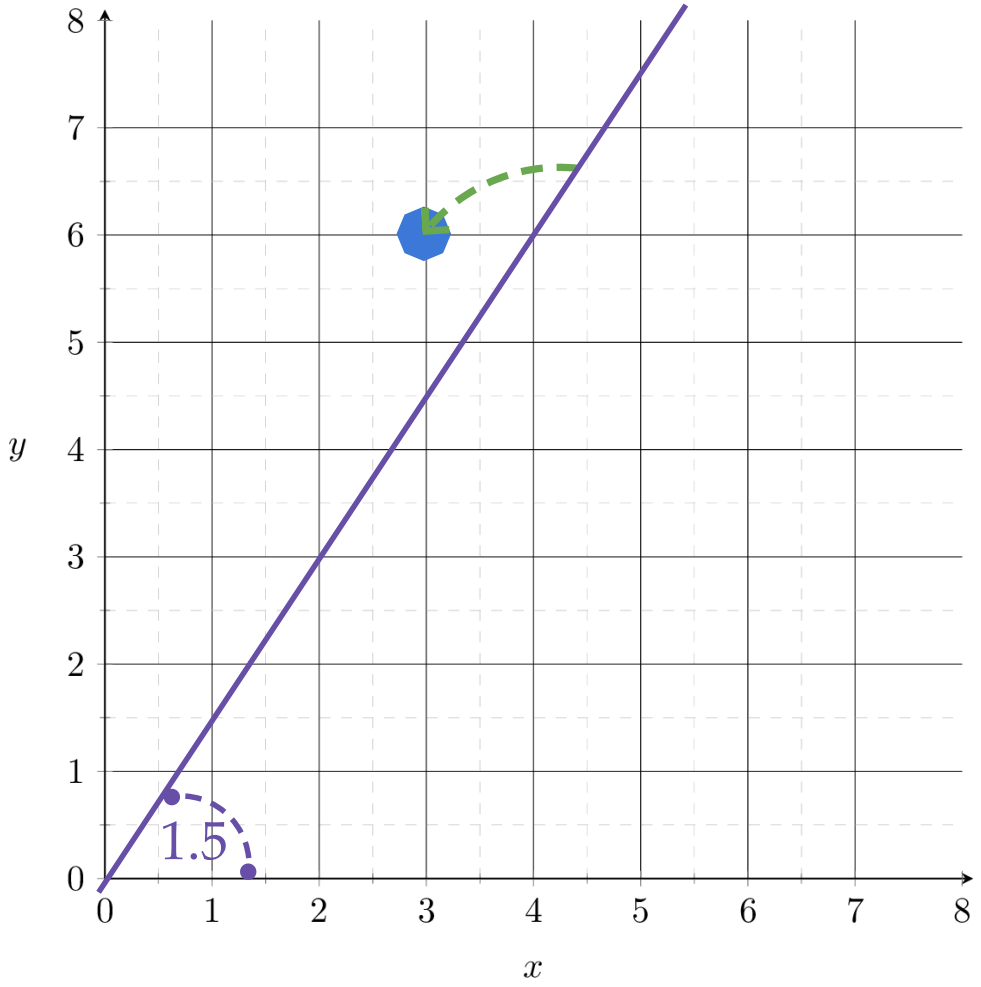


MSE could get better.

How to formalize this?

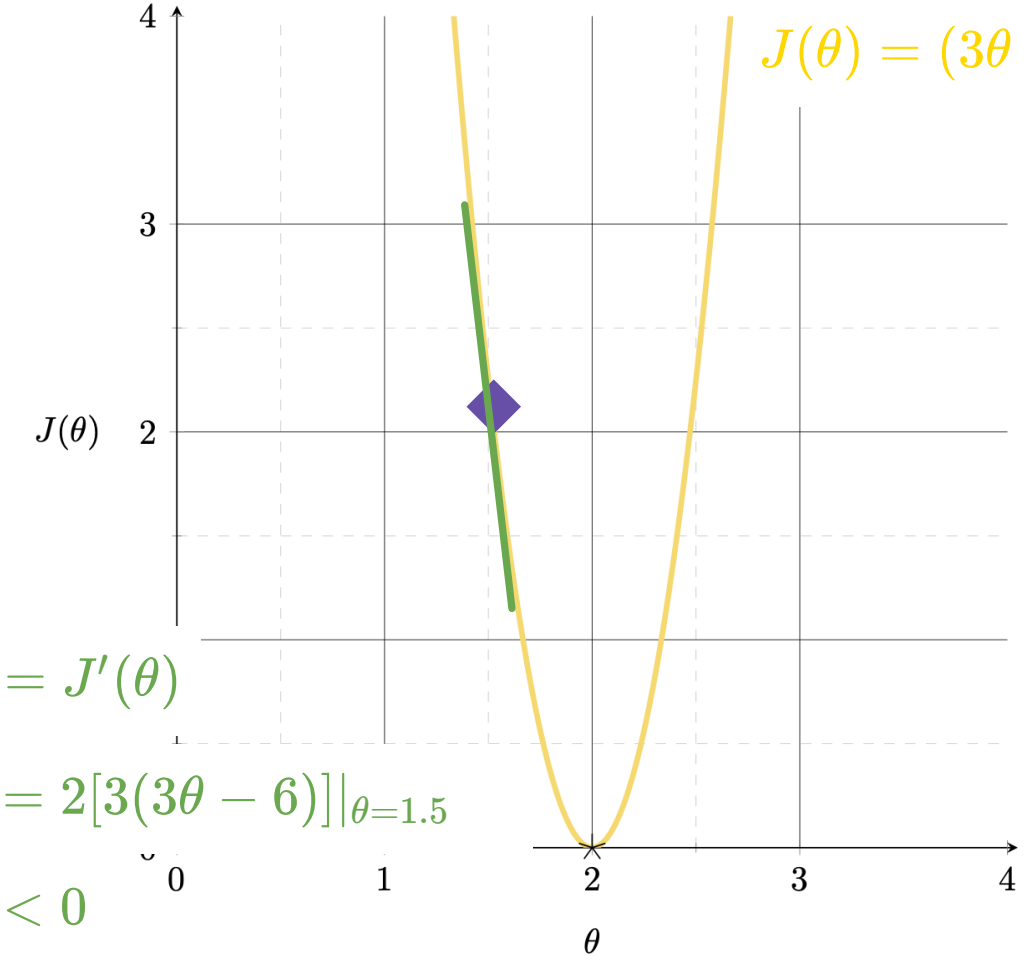


Suppose we fit a line $y = 1.5x$



MSE could get better. How to?

Leveraging the gradient.

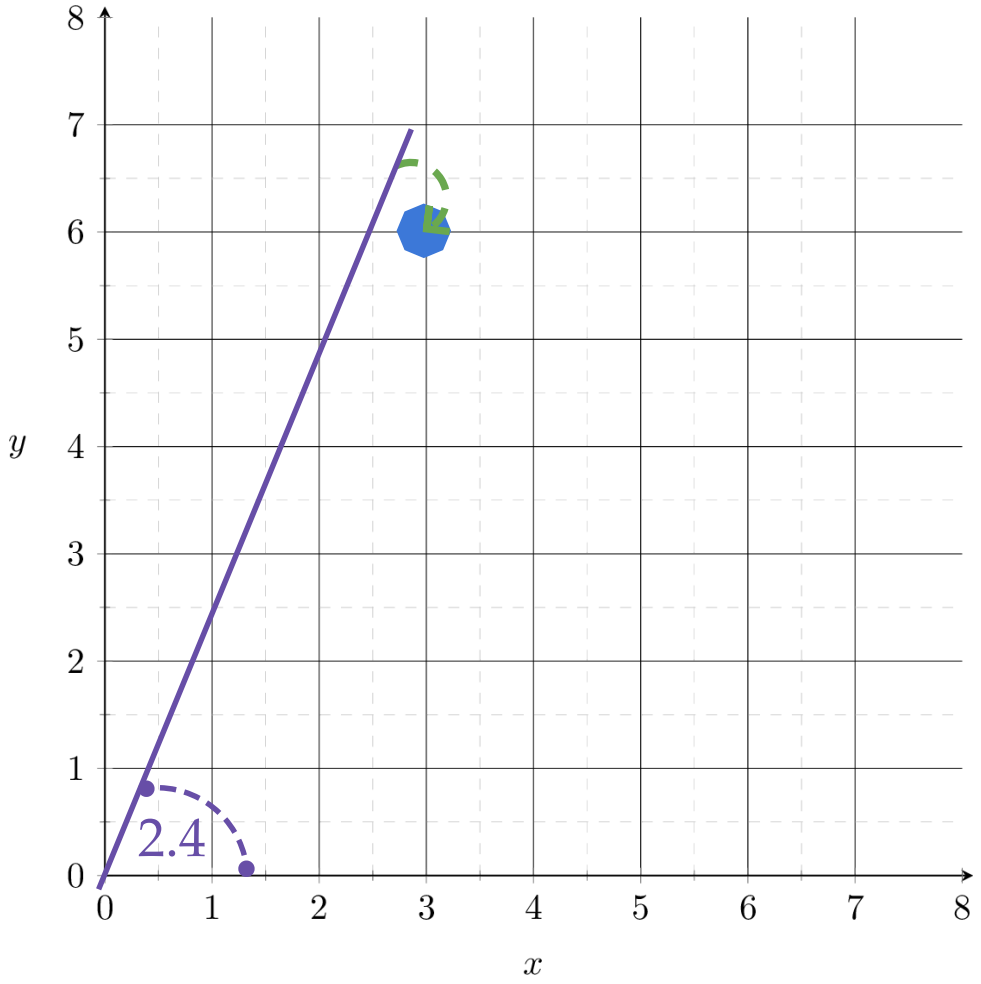


$$\nabla_\theta J = J'(\theta)$$

$$= 2[3(3\theta - 6)]|_{\theta=1.5}$$

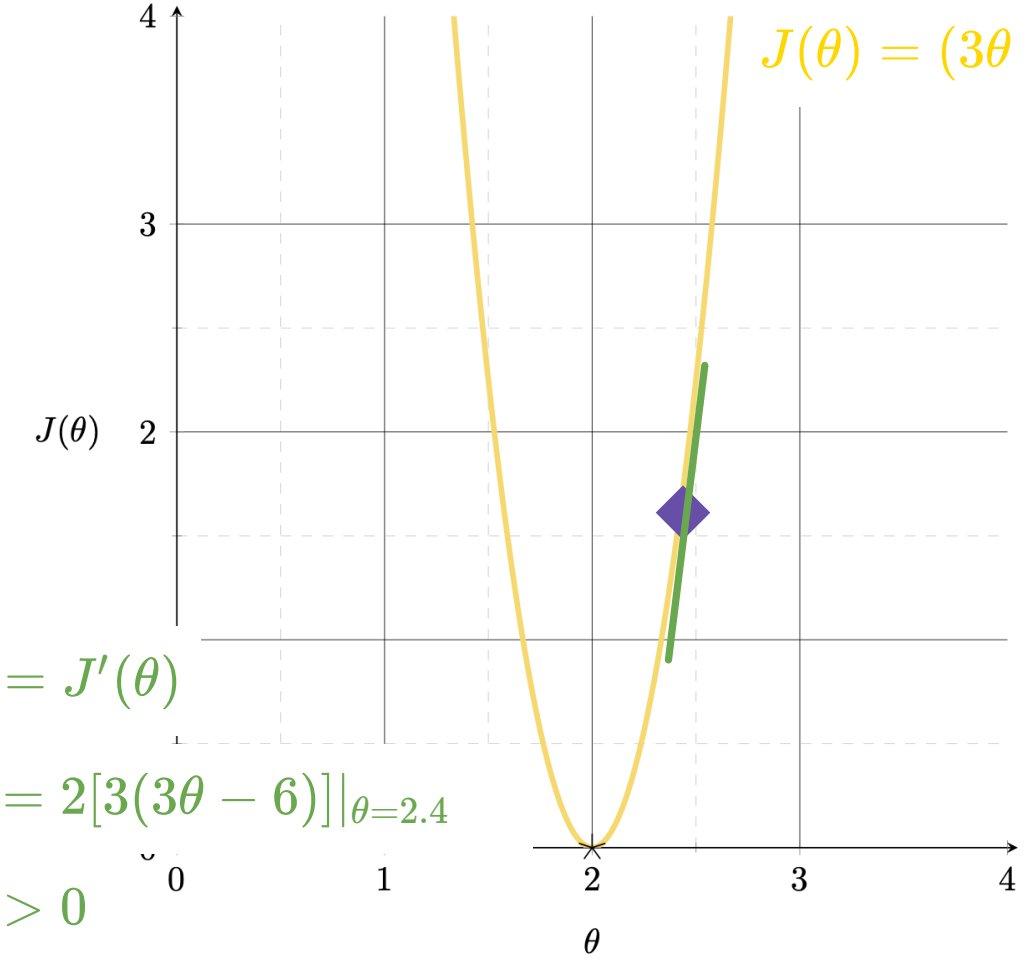
$$< 0$$

Suppose we fit a line $y = 2.4x$



MSE could get better. How to?

Leveraging the gradient.



$$\nabla_\theta J = J'(\theta)$$

$$= 2[3(3\theta - 6)]|_{\theta=2.4}$$

$$> 0$$

initial guess

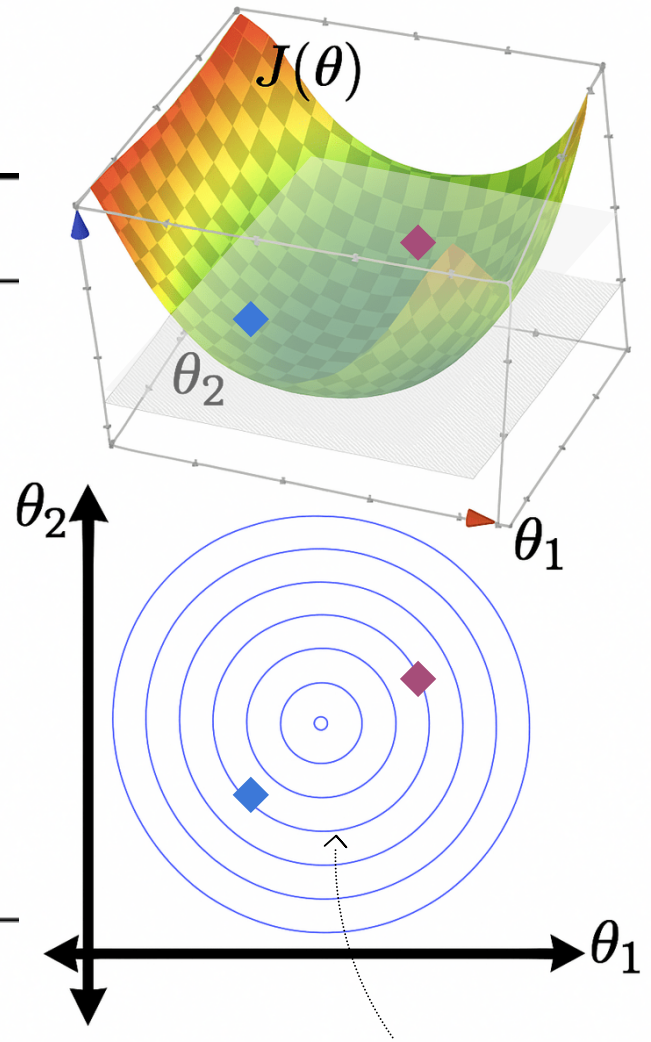
hyperparameters of parameters learning rate precision

Algorithm 1 Gradient Descent($\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon$)

1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$

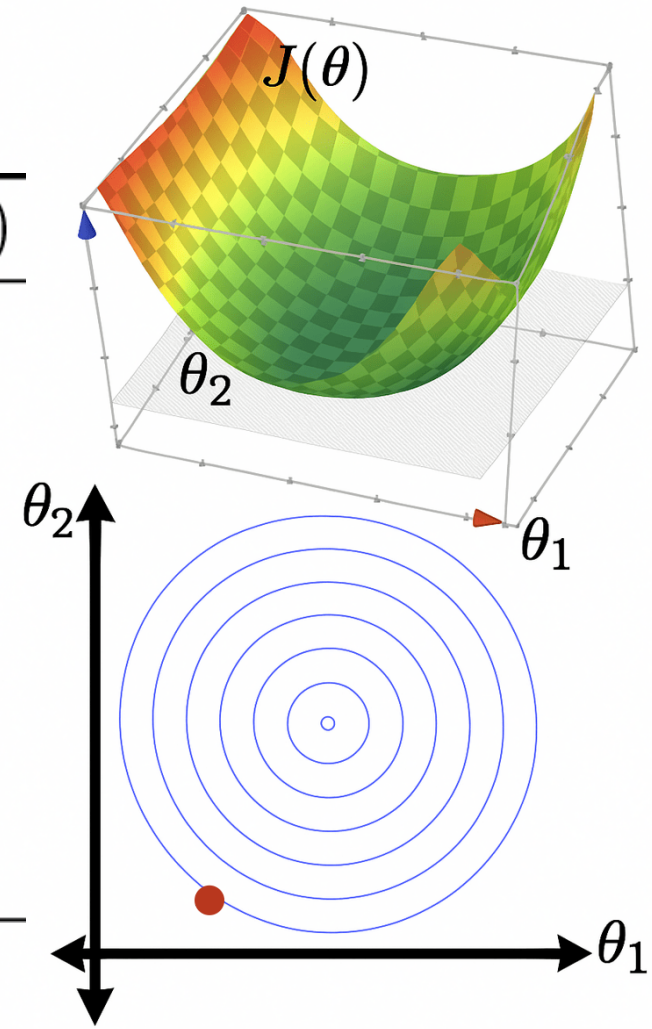
Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$



Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

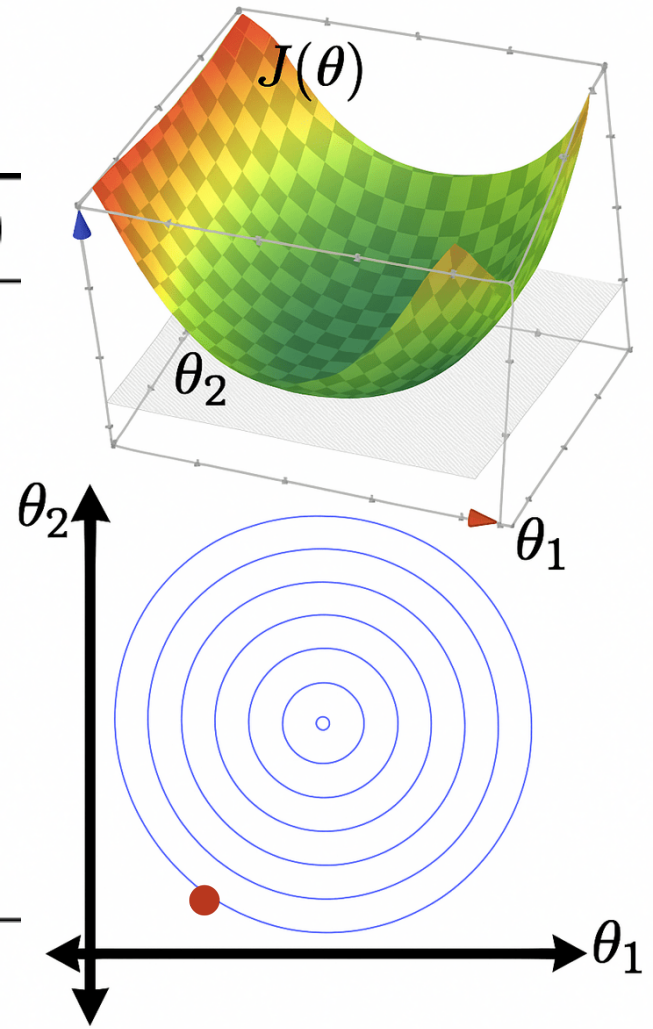
Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
    $t = t + 1$
    $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
return $\theta^{(t)}$



Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

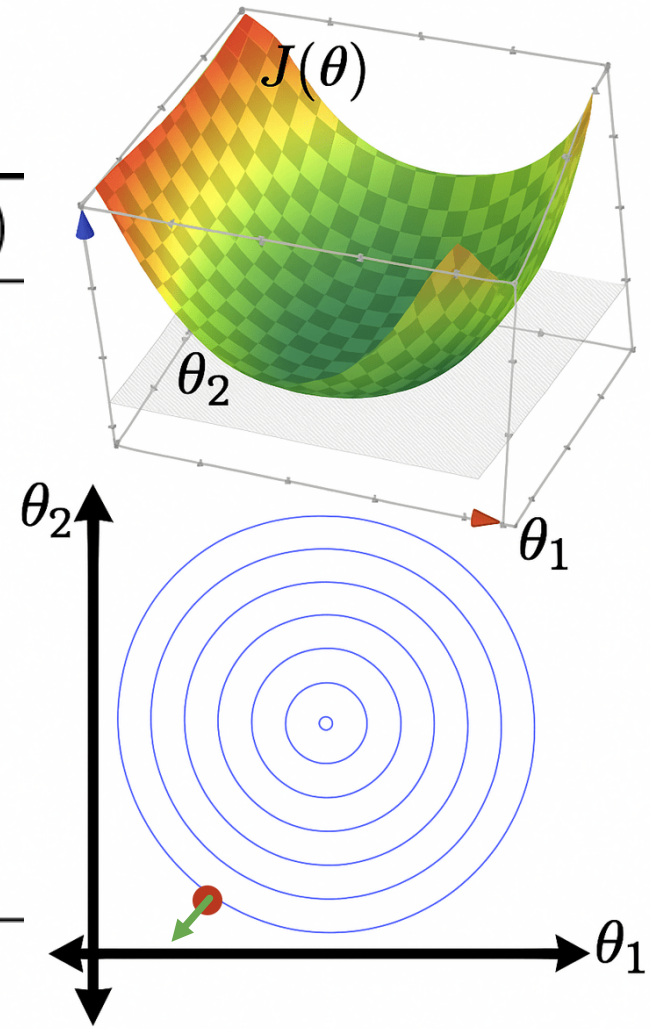
Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
    $t = t + 1$
    $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
return $\theta^{(t)}$

iteration counter



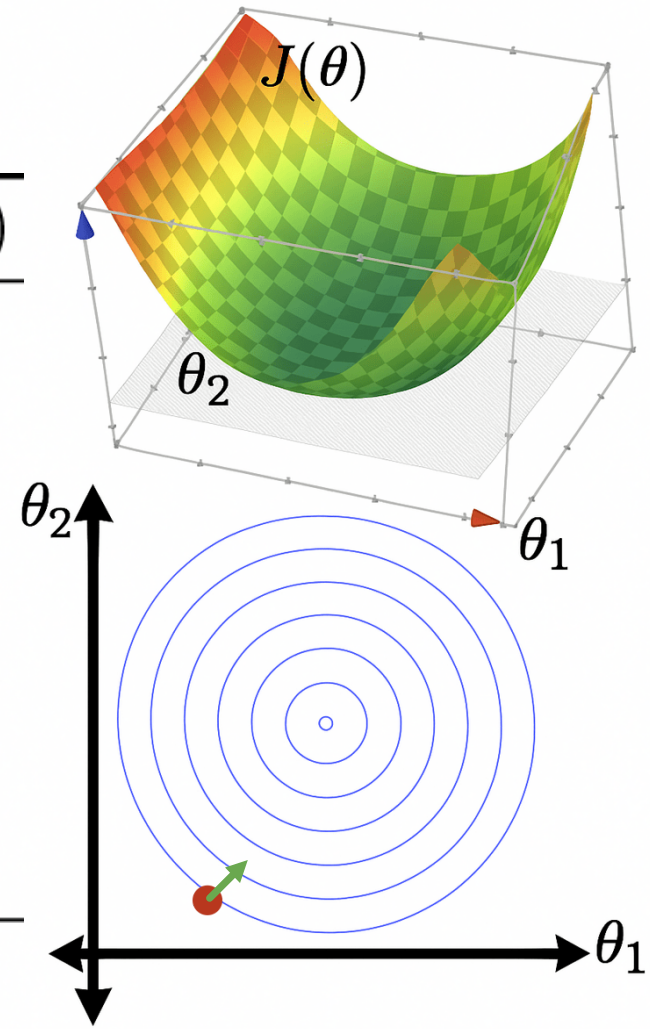
Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$



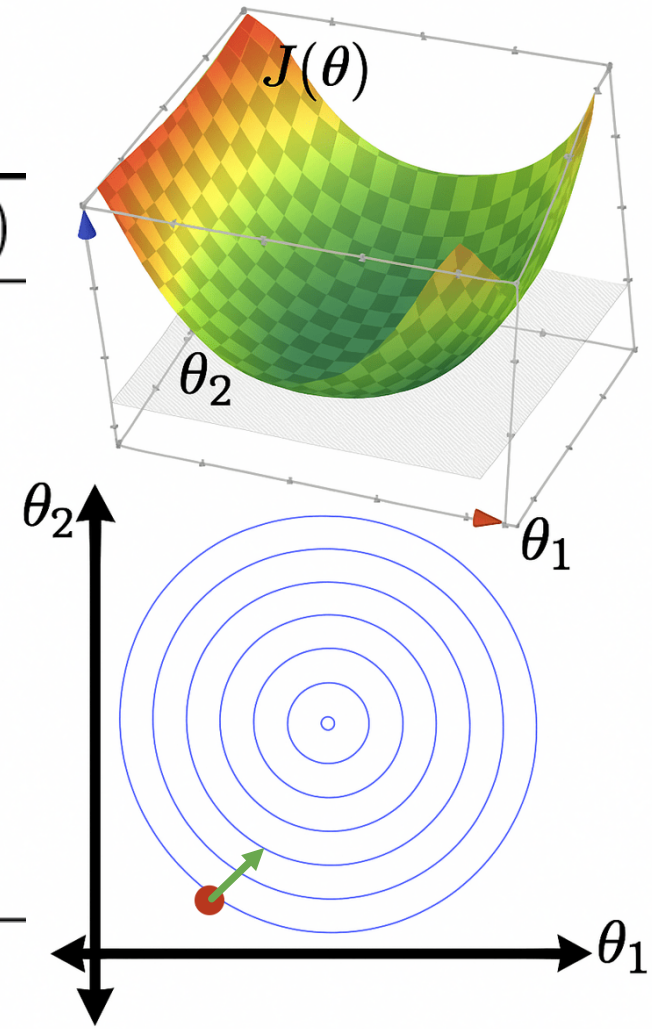
Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$



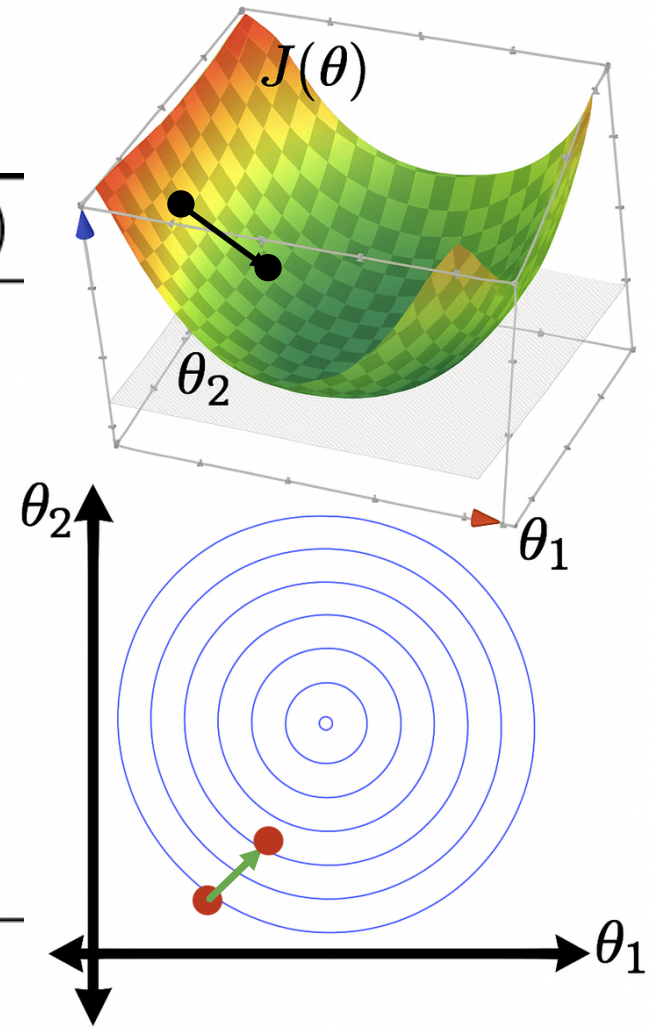
Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$



Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

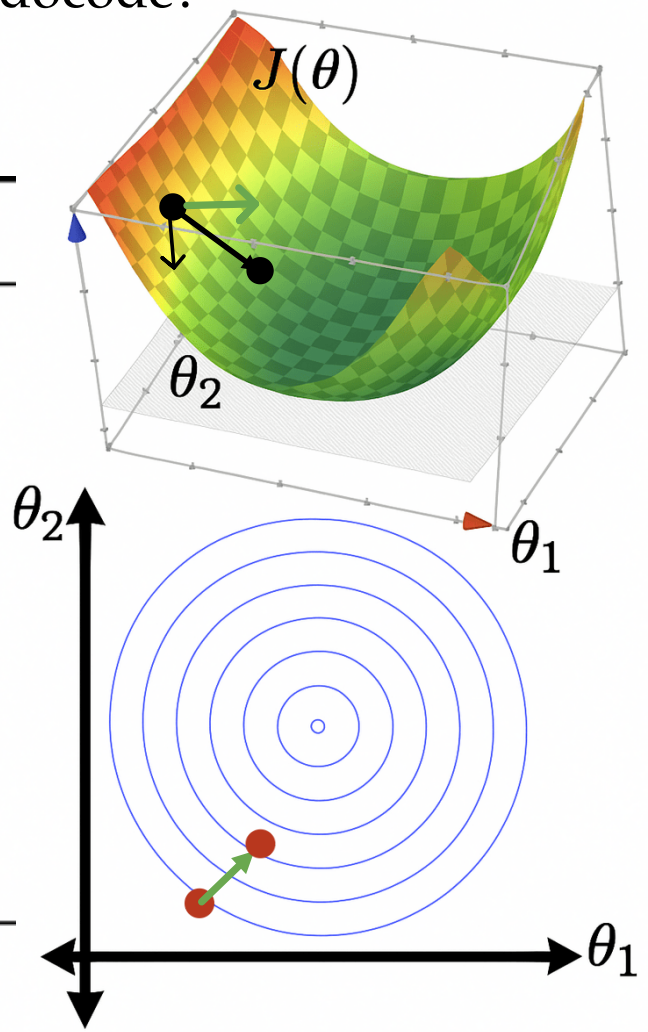
1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$



- What does this 3d vector ↘ represent? anything in the psuedocode?

Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

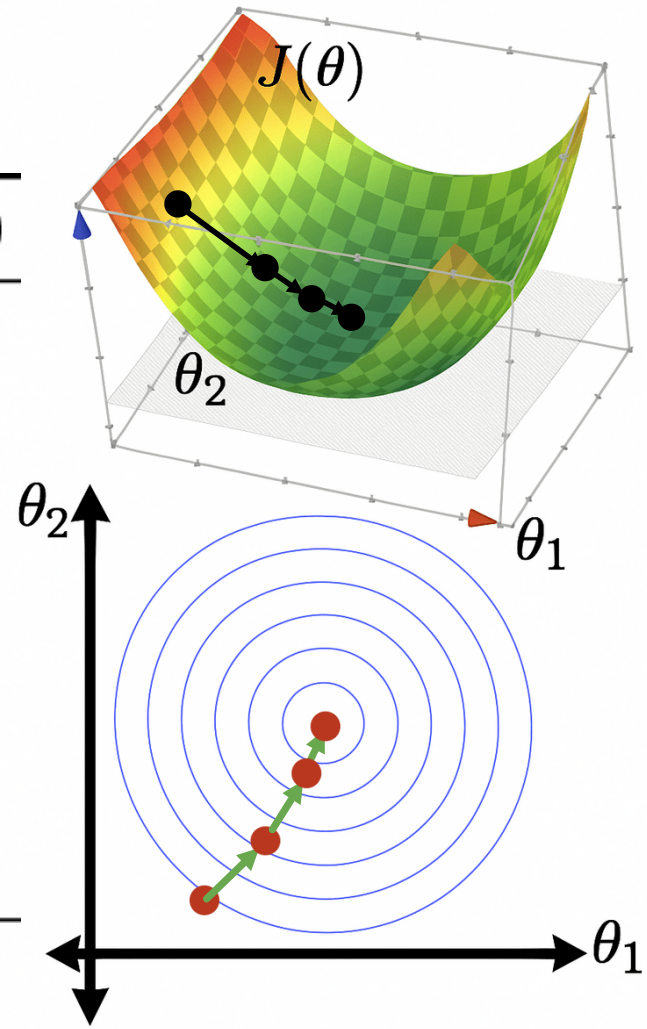
1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$



- What does this 2d vector ↗ represent? anything in the psuedocode?

Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

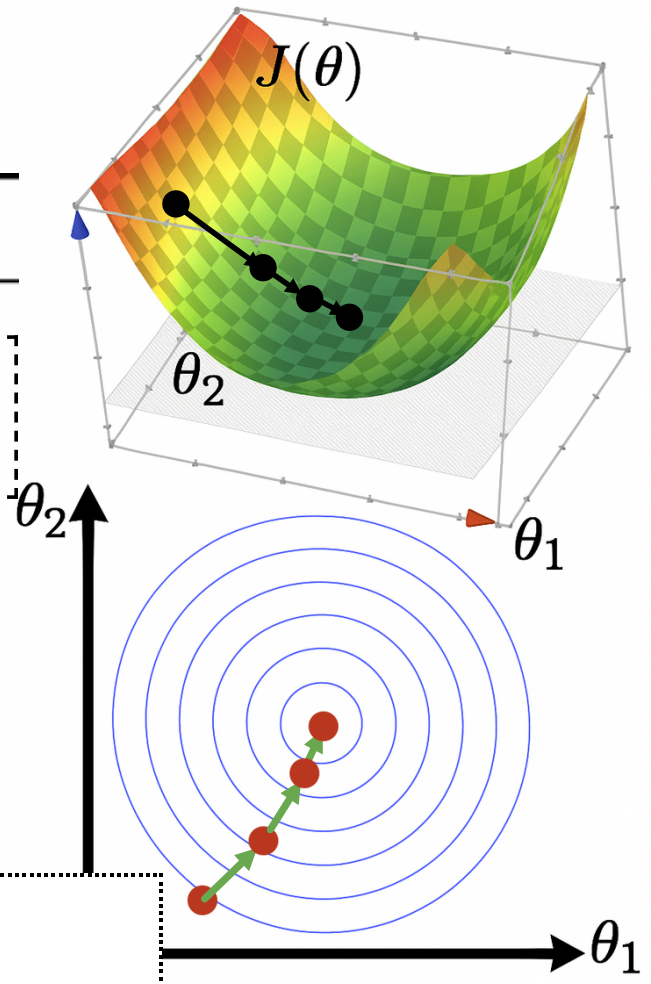
1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$



Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
6: **until** $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
7: **return** $\theta^{(t)}$

objective improvement is nearly zero.



Other possible stopping criterion for line 6:

- Small parameter change: $\|\theta^{(t)} - \theta^{(t-1)}\| < \epsilon$, or
- Small gradient norm: $\|\nabla_\theta J(\theta^{(t-1)})\| < \epsilon$

Outline

- Gradient descent algorithm (GD)
  - The gradient vector
  - GD algorithm
  - Gradient decent properties
- Stochastic gradient descent (SGD)
  - SGD algorithm and setup
  - SGD vs. GD

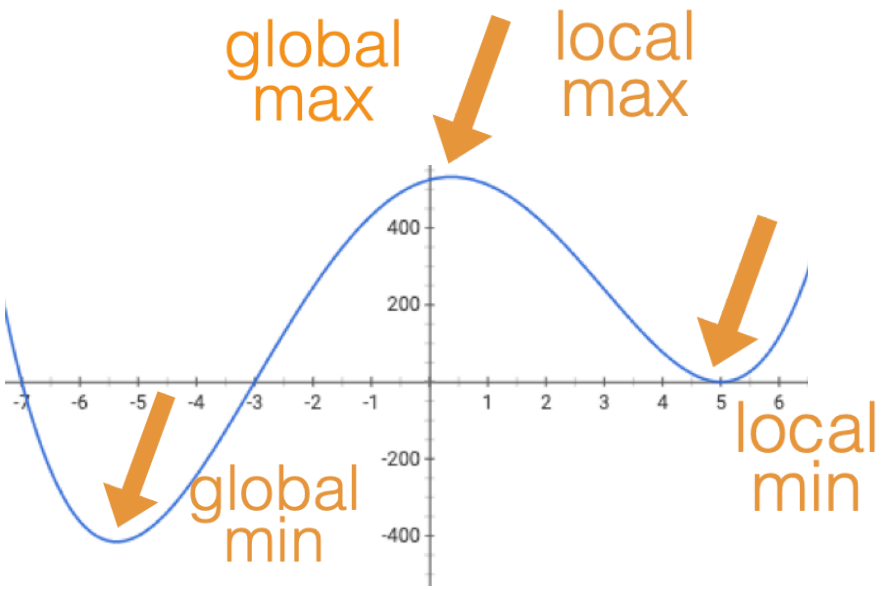
When minimizing a function, we aim for a global minimizer.

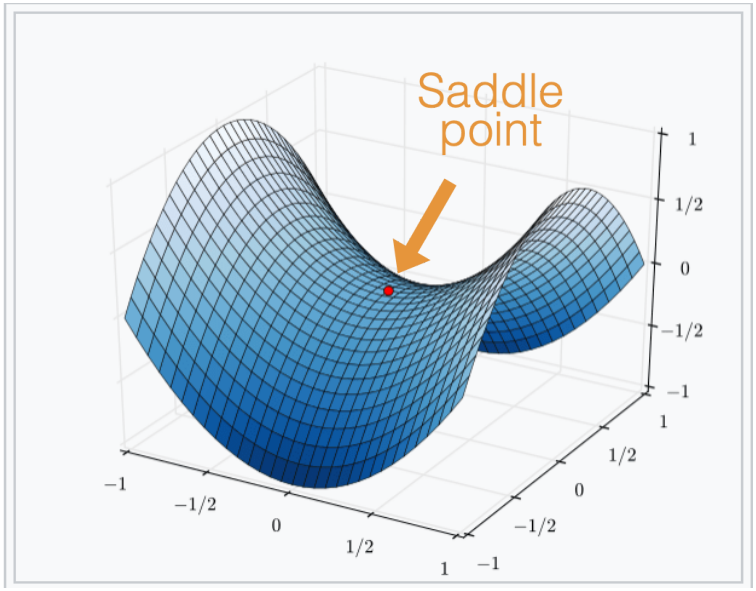
$\Rightarrow$

At a global minimizer the gradient vector is zero

gradient descent can achieve this (to arbitrary precision)

$\not\Leftarrow$





When minimizing a function, we aim for a global minimizer.

At a global minimizer $\Leftarrow$ { the gradient vector is zero ; the objective function is convex

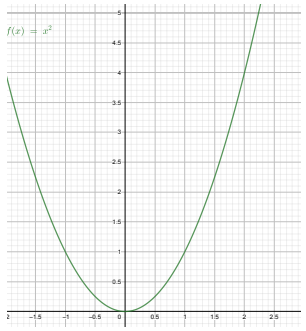
A function f is ***convex*** if any line segment connecting two points of the graph of f lies above or on the graph.

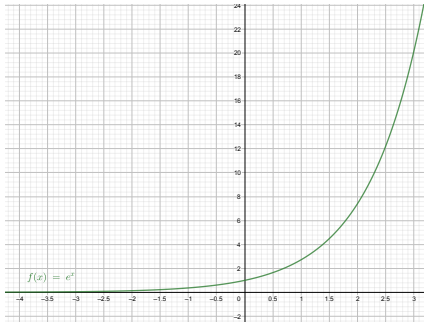
- f is concave if $-f$ is convex.
- Convex functions are the largest well-understood class of functions where optimization theory guarantees convergence and efficiency

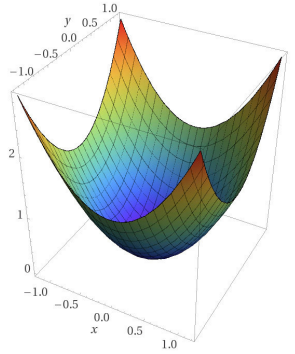
https://shenshen.mit.edu/demos/convex.html

Some examples

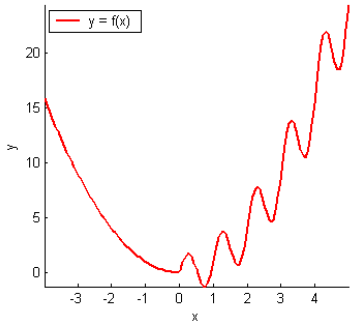
Convex functions

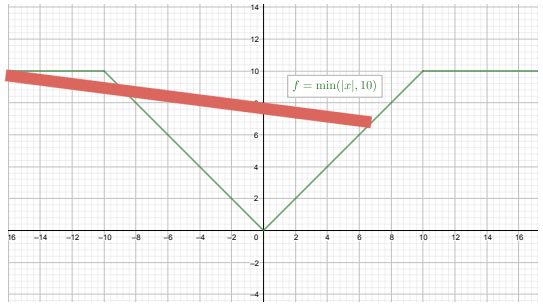


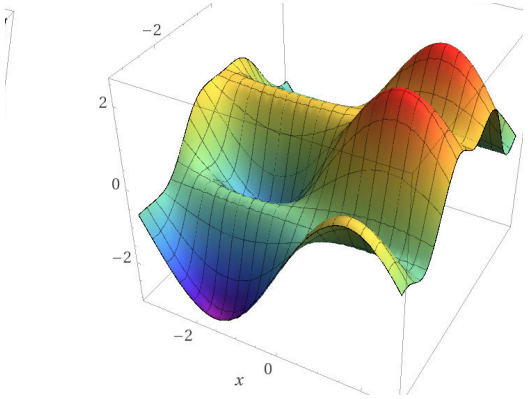




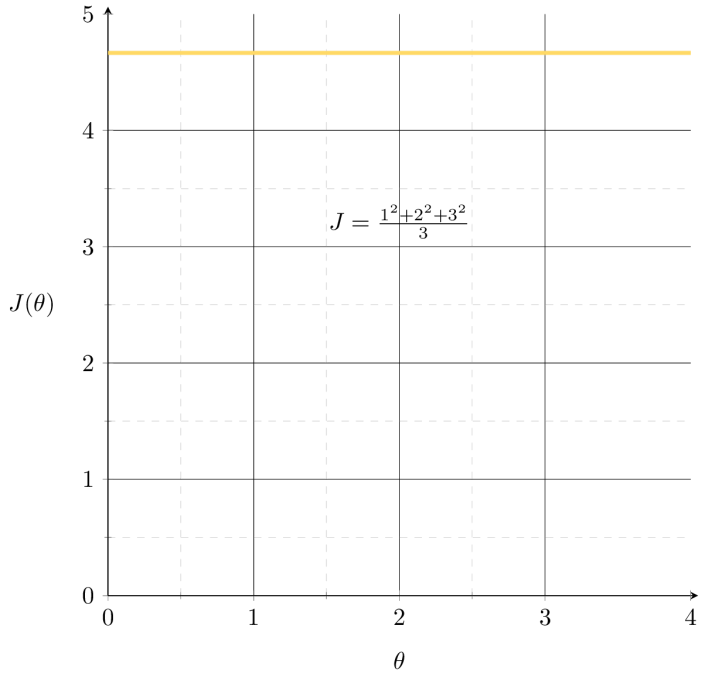
Non-convex functions

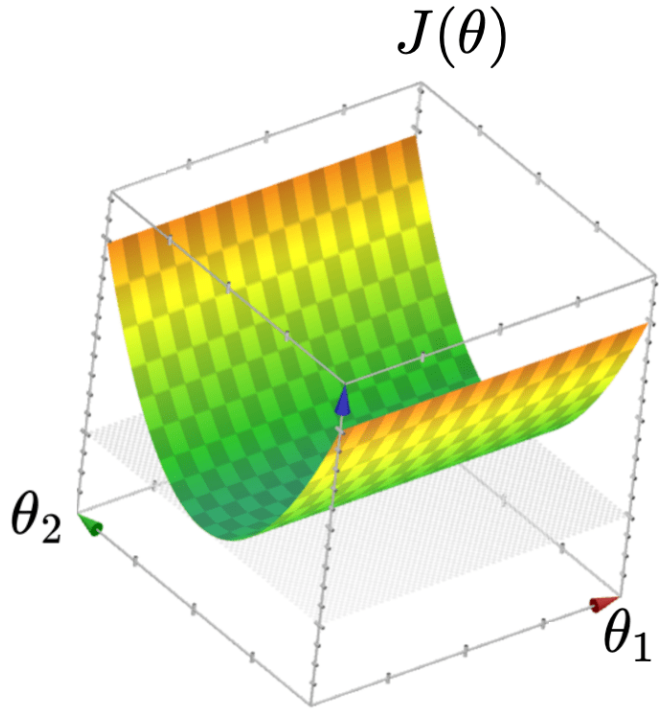


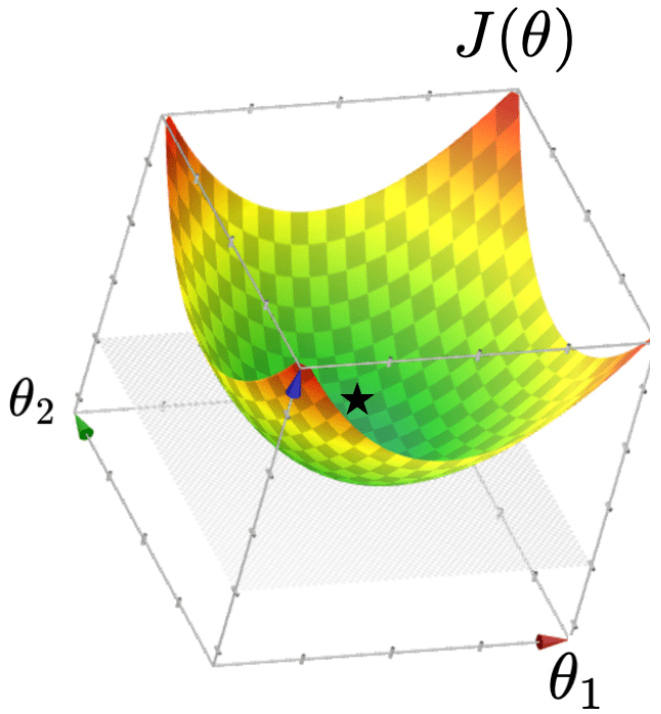




- MSE: $J(\theta) = \frac{1}{n}(X\theta - Y)^\top (X\theta - Y)$ is always convex



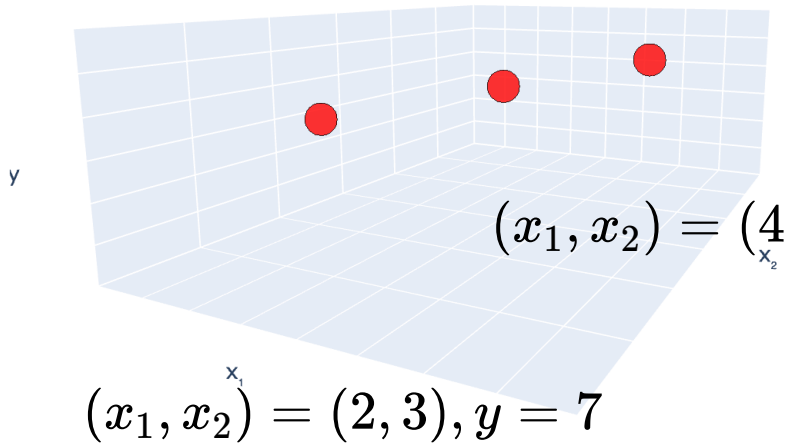




convexity is why we can claim the point whose gradient is zero is a global minimizer.

case (c) training data set again

$(x_1, x_2) = (6, 9), y = 9$



$(x_1, x_2) = (4, 6), y = 8$

$(x_1, x_2) = (2, 3), y = 7$

- Ridge objective with $\lambda > 0$ is always (strongly) convex

convexity is why we can claim the point whose gradient is zero is a global minimizer.

Gradient Descent Performance

- Assumptions:
  - f is sufficiently "smooth"
  - f is convex
  - f has at least one global minimum
  - Run gradient descent for sufficient iterations
  - η is sufficiently small
- Conclusion:
  - Gradient descent converges arbitrarily close to a global minimizer of f.

Gradient Descent Performance

- Assumptions:
  - f is sufficiently "smooth"
  - f is convex
  - f has at least one global minimum
  - Run gradient descent for sufficient iterations
  - η is sufficiently small

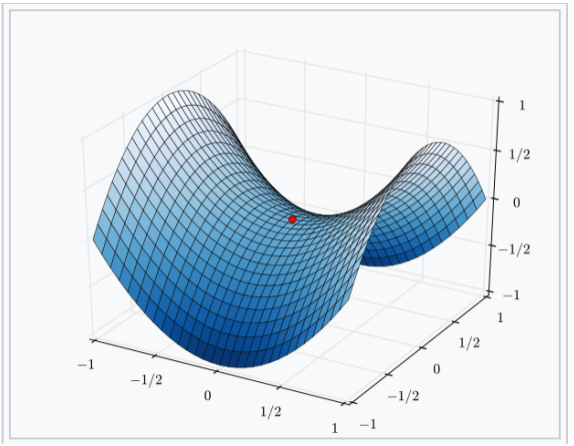
if violated, may not have gradient, can't run gradient descent

- Conclusion:
  - Gradient descent converges arbitrarily close to a global minimizer of f.

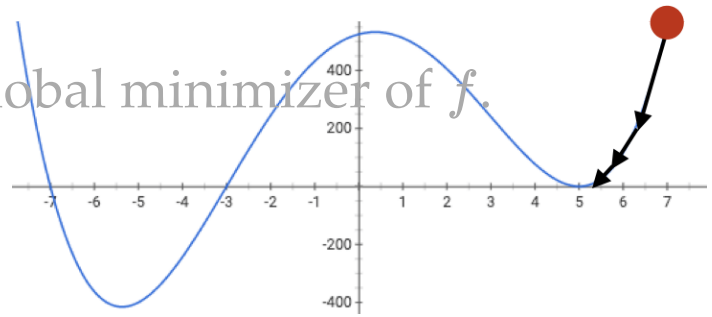
Gradient Descent Performance

- Assumptions:
  - f is sufficiently "smooth"
  - f is convex
  - f has at least one global minimum
  - Run gradient descent for sufficient iterations
  - η is sufficiently small
- Conclusion:
  - Gradient descent converges arbitrarily close to a global minimizer of f.

if violated, may get stuck at a saddle point



or a local minimum

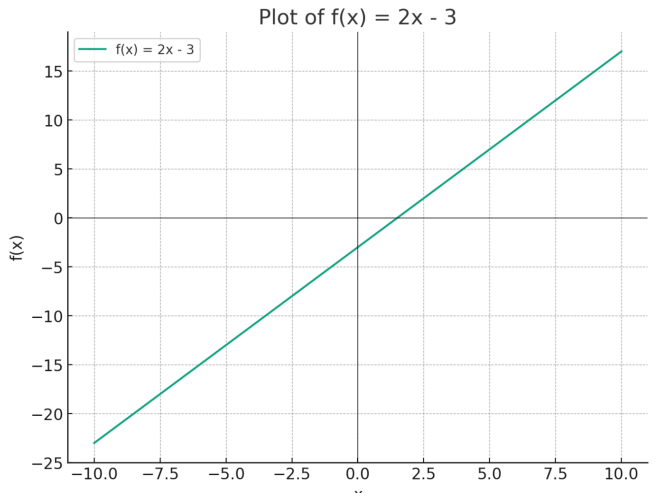


Gradient Descent Performance

- Assumptions:
  - f is sufficiently "smooth"
  - f is convex
  - f has at least one global minimum
  - Run gradient descent for sufficient iterations
  - η is sufficiently small
- Conclusion:
  - Gradient descent converges arbitrarily close to a global minimizer of f.

if violated:

may not terminate/no minimum to converge to

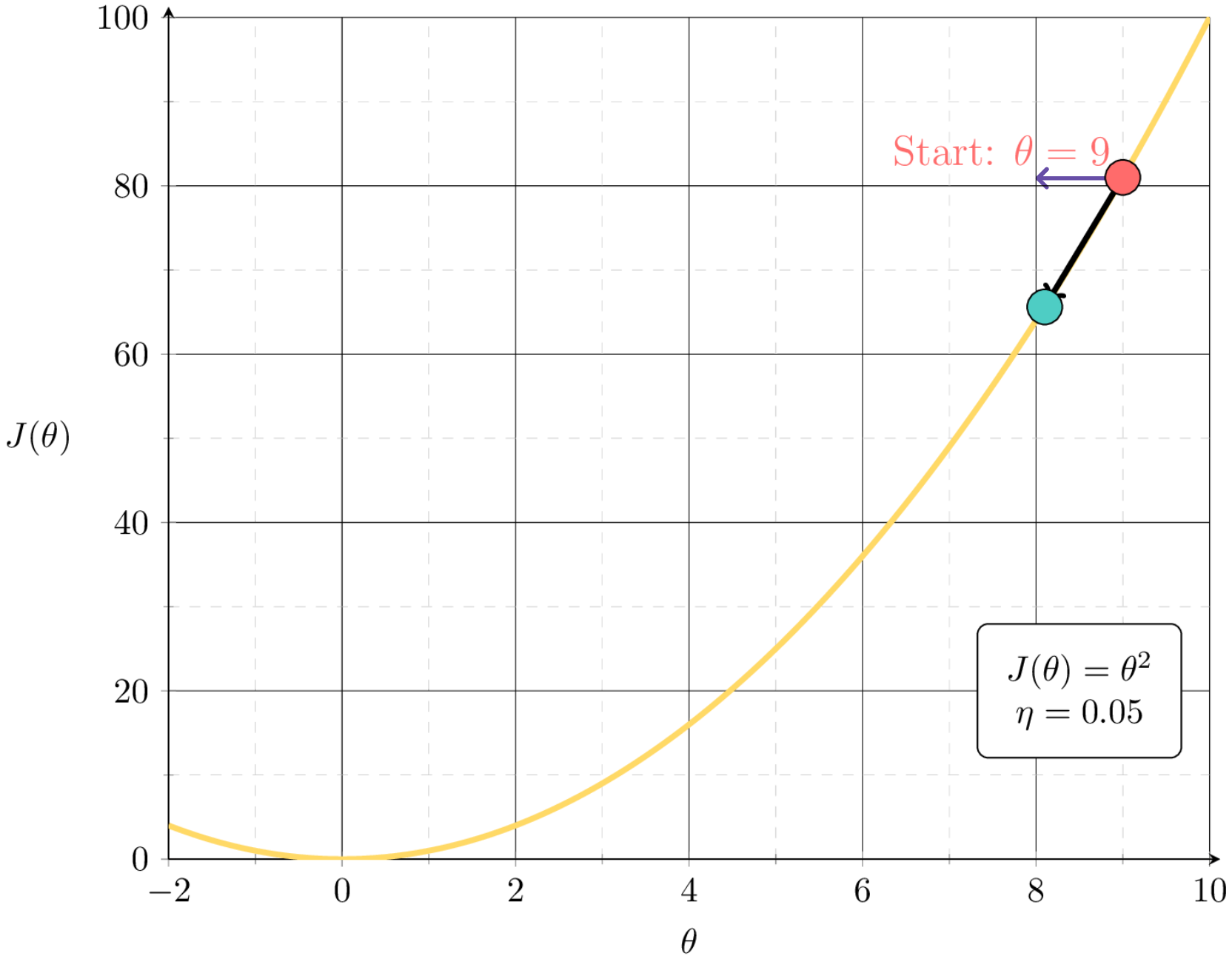


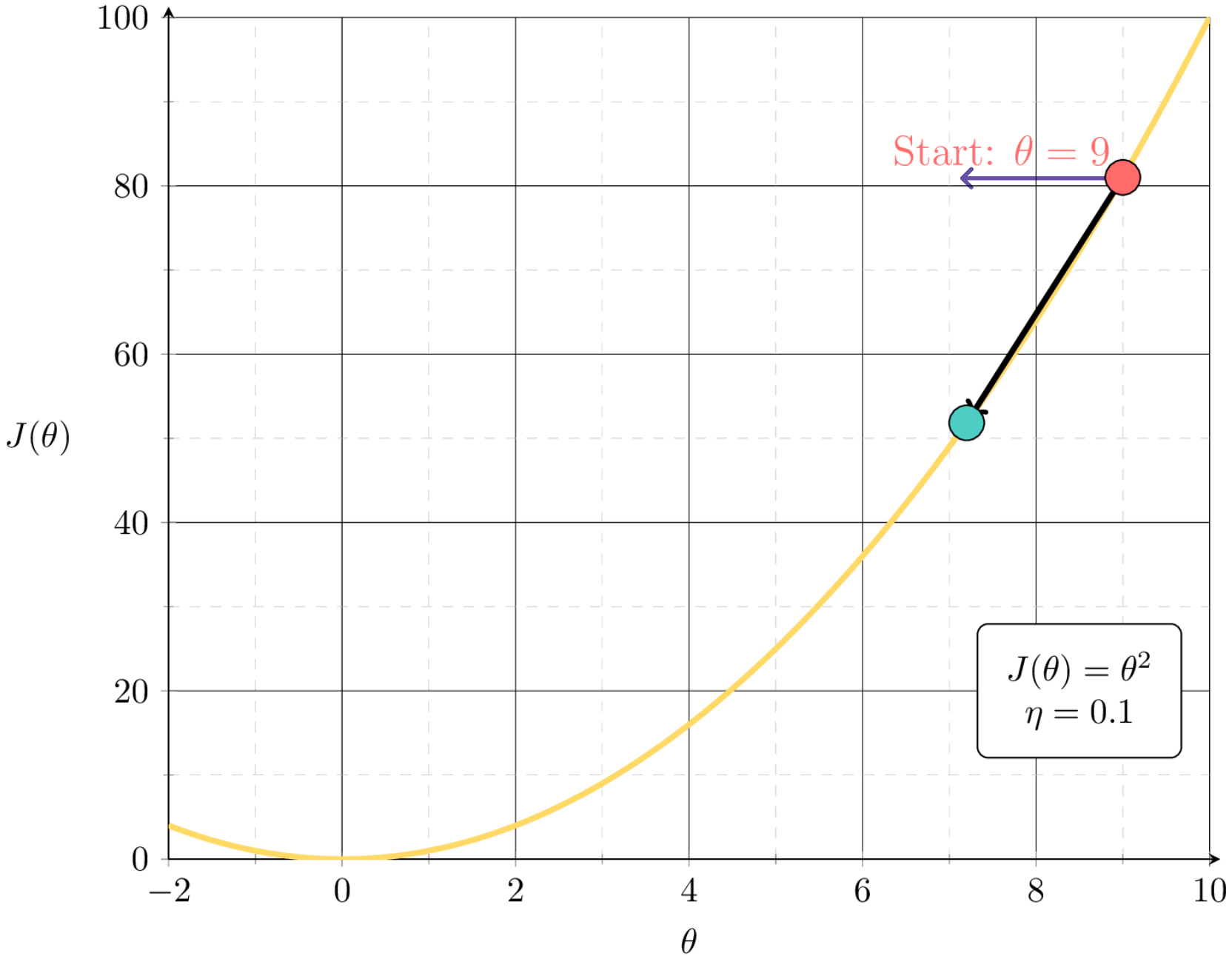
Gradient Descent Performance

- Assumptions:
  - f is sufficiently "smooth"
  - f is convex
  - f has at least one global minimum
  - Run gradient descent for sufficient iterations
  - η is sufficiently small
- Conclusion:
  - Gradient descent converges arbitrarily close to a global minimizer of f.

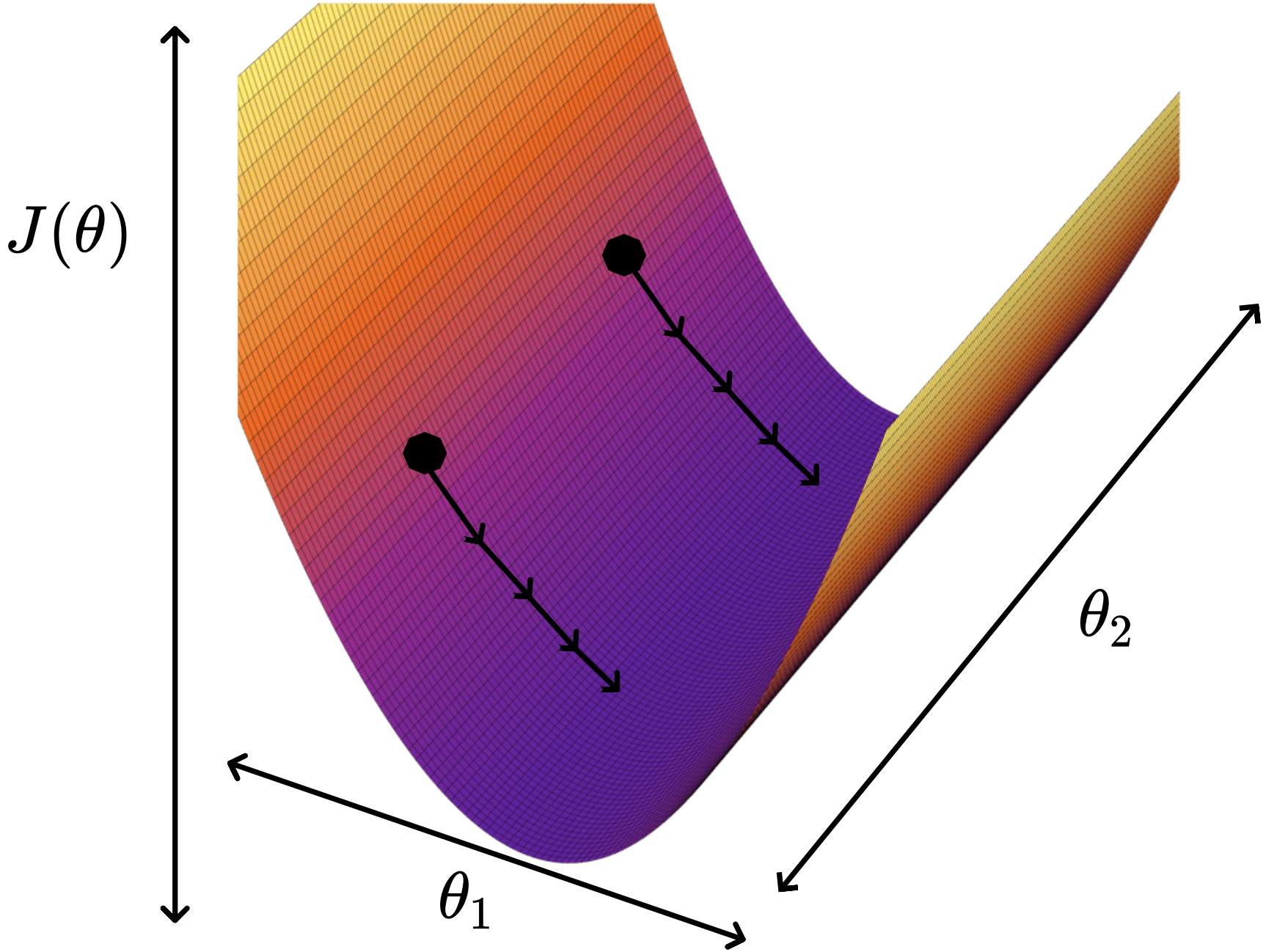
if violated:

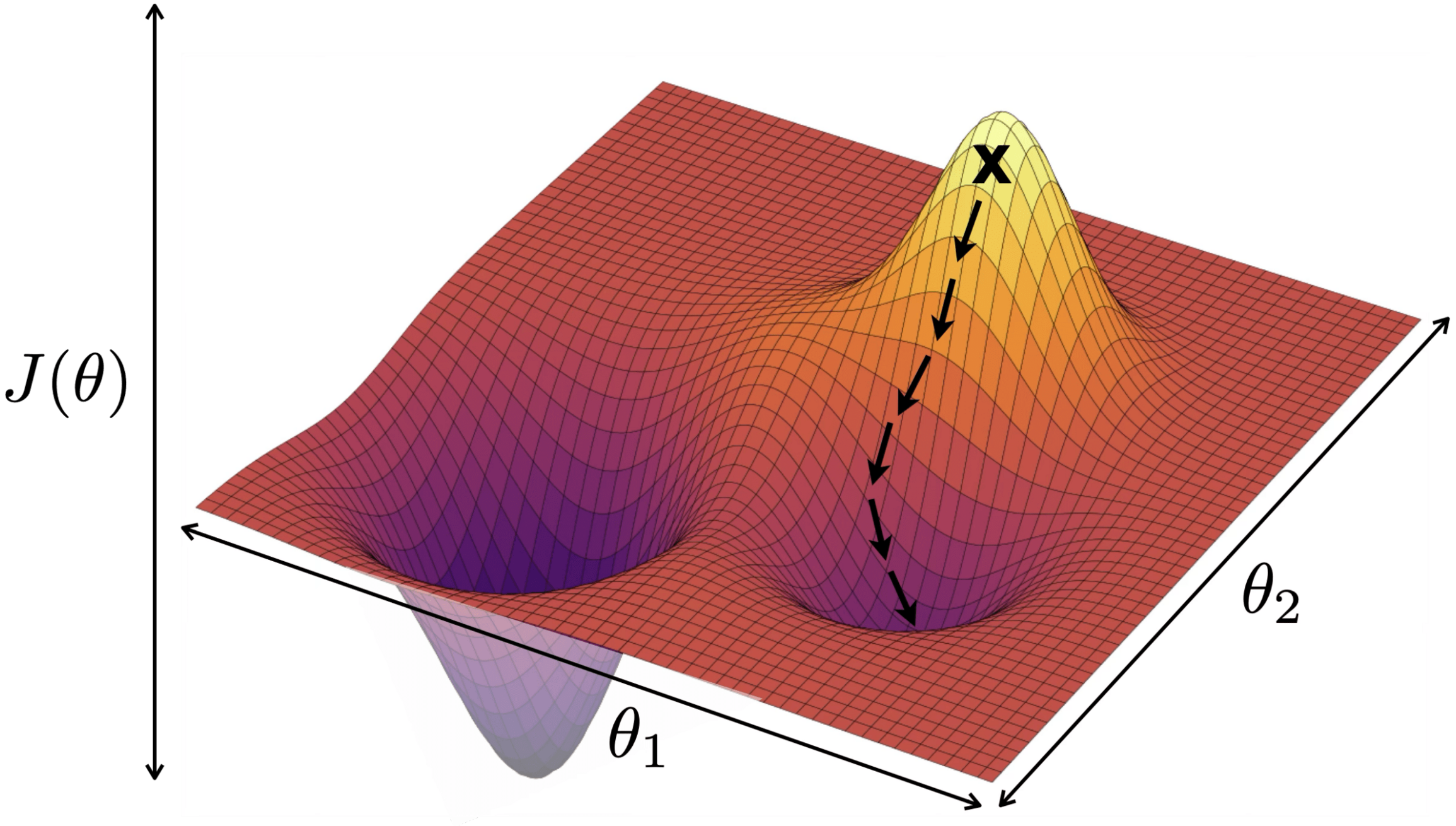
see demo on next slide, also lab/hw


100
80
60
40
20
0
$J(\theta)$
−2
0
2
4
6
8
10
θ
Start: $\theta = 9$
$J(\theta) = \theta^2$
$\eta = 0.05$


100
80
60
40
20
0
$J(\theta)$
−2
0
2
4
6
8
10
θ
Start: $\theta = 9$
$J(\theta) = \theta^2$
$\eta = 0.1$

https://shenshen.mit.edu/demos/gd.html


$J(\theta)$
θ_1
θ_2

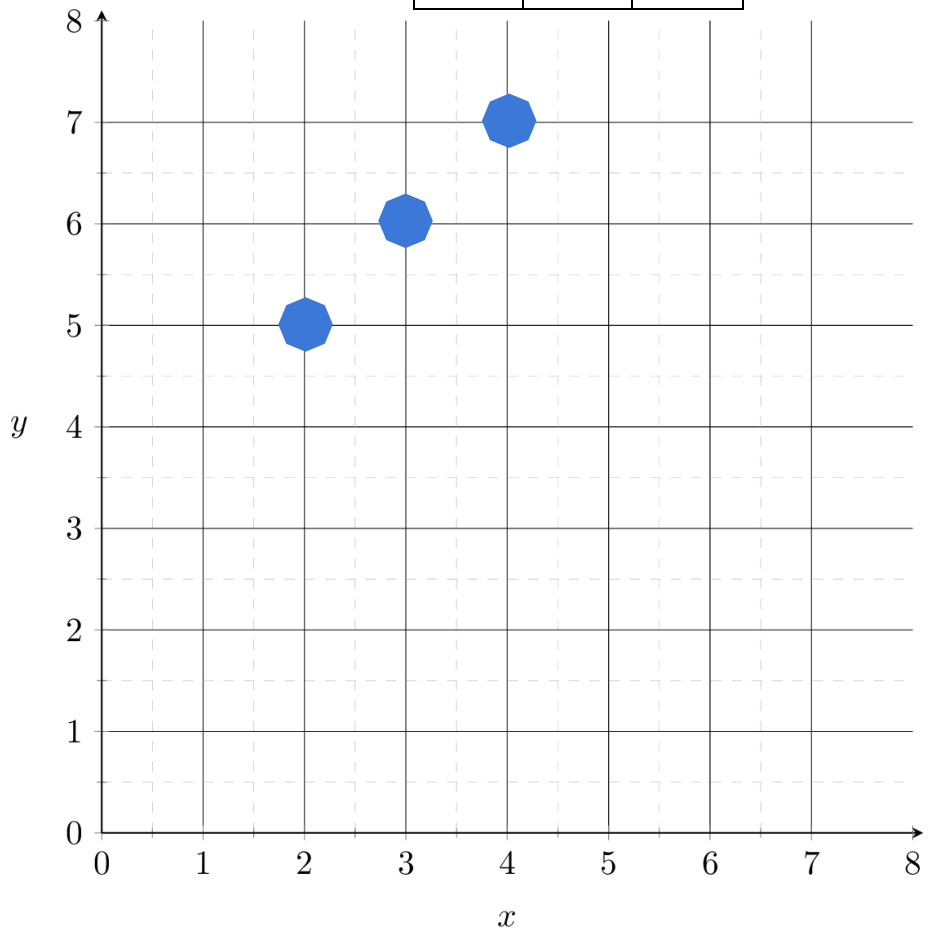

$J(\theta)$
x
θ_1
θ_2

Outline

- Gradient descent algorithm (GD)
  - The gradient vector
  - GD algorithm
  - Gradient decent properties
- Stochastic gradient descent (SGD)
  - SGD algorithm and setup
  - SGD vs. GD

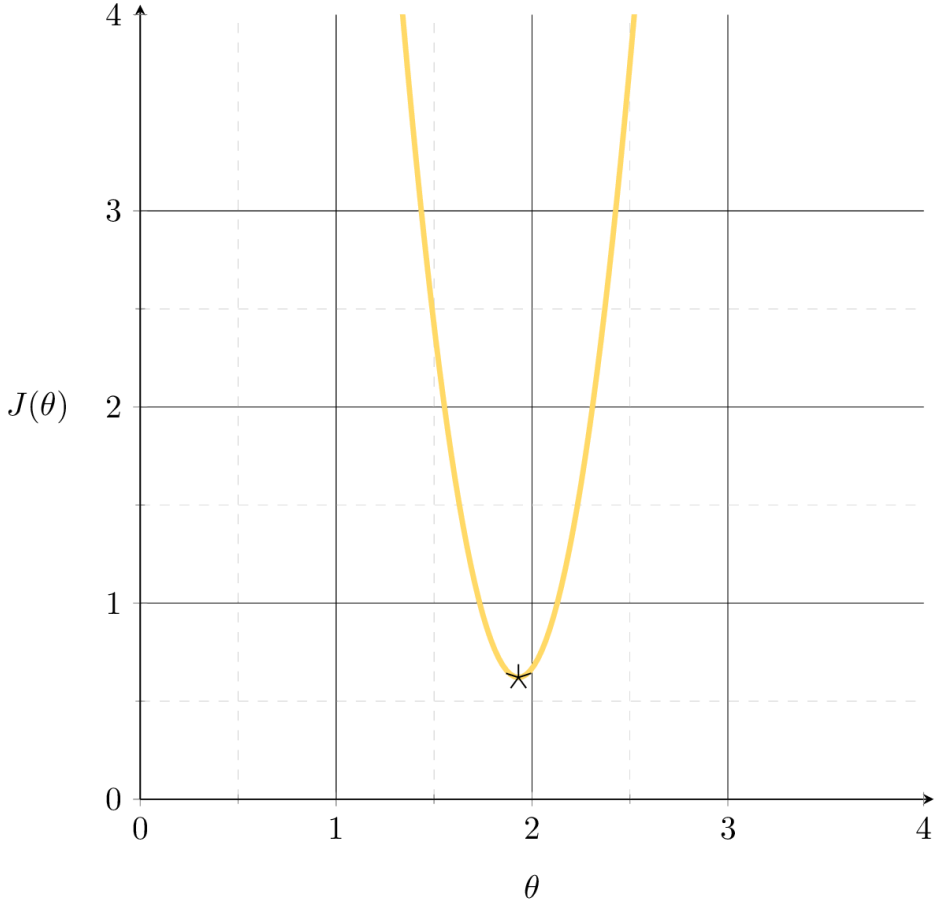
training data

	x	y
p1	2	5
p2	3	6
p3	4	7



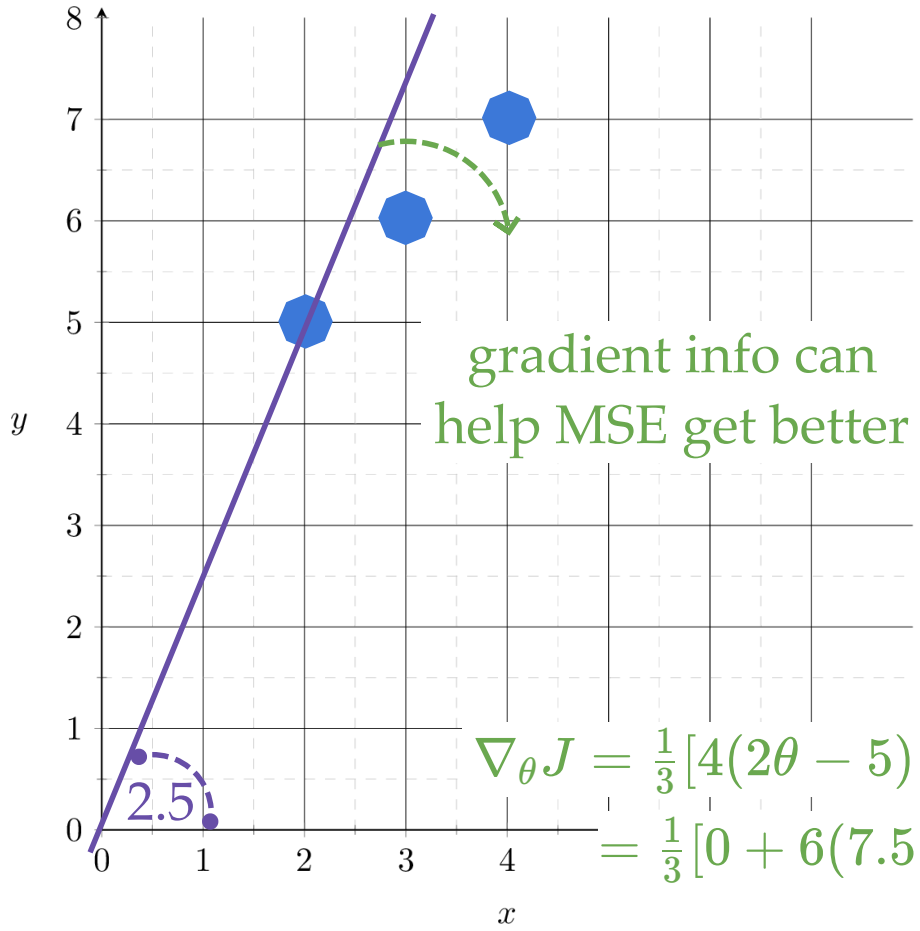
Fit a line (without offset) to the dataset, the MSE:

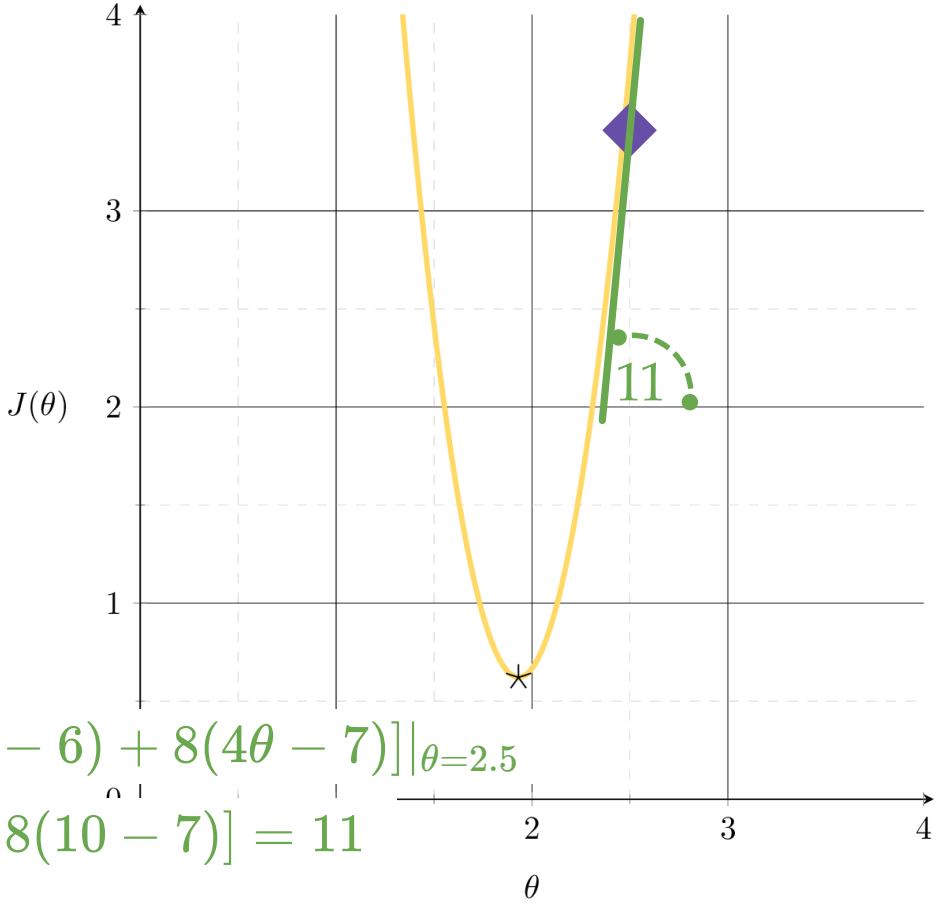
$$J(\theta) = \frac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$



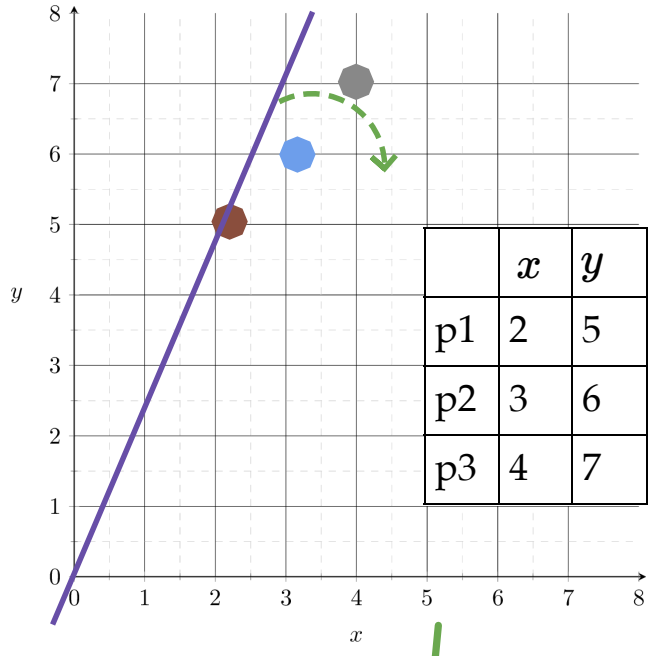
Suppose we fit a line $y = 2.5x$

$$J(\theta) = \frac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

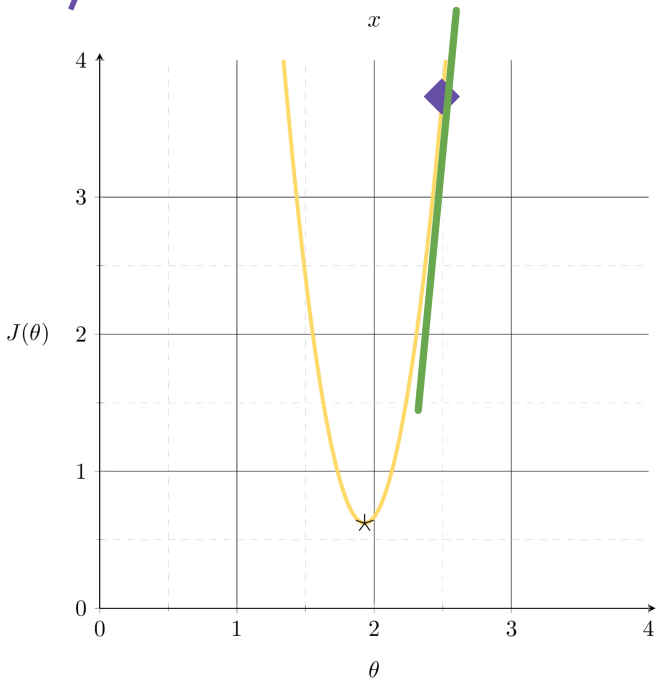




$$\nabla_\theta J = \frac{1}{3}[4(2\theta - 5) + 6(3\theta - 6) + 8(4\theta - 7)]|_{\theta=2.5}$$
$$= \frac{1}{3}[0 + 6(7.5 - 6) + 8(10 - 7)] = 11$$



	x	y
p1	2	5
p2	3	6
p3	4	7



- the MSE of a linear hypothesis:

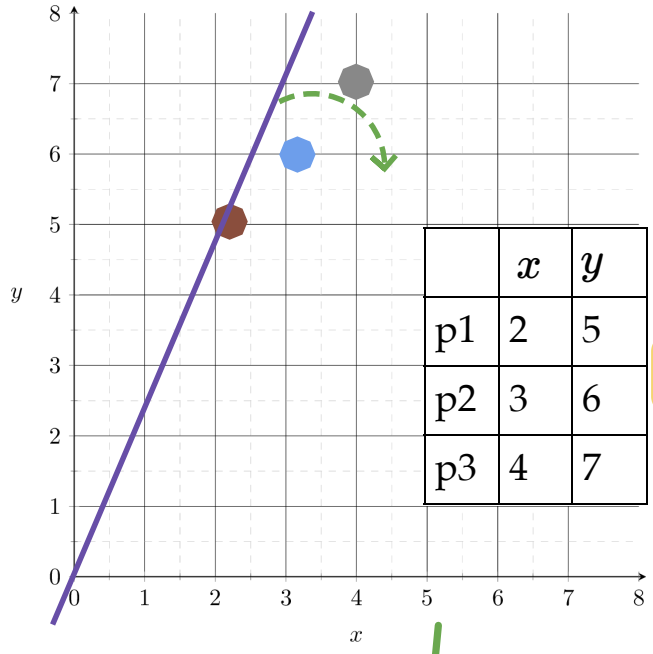
$$J(\theta) = \frac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

$$= \frac{1}{3}\left[\quad J_1 \quad + \quad J_2 \quad + \; J_3\;\right]$$

- and its gradient w.r.t. θ:

$$\nabla_\theta J = \frac{1}{3}\left[4(2\theta - 5) + 6(3\theta - 6) + 8(4\theta - 7)\right]$$

$$= \frac{1}{3}\left[\; \nabla_\theta J_1 \; + \; \nabla_\theta J_2 \; + \; \nabla_\theta J_3 \quad\right]$$

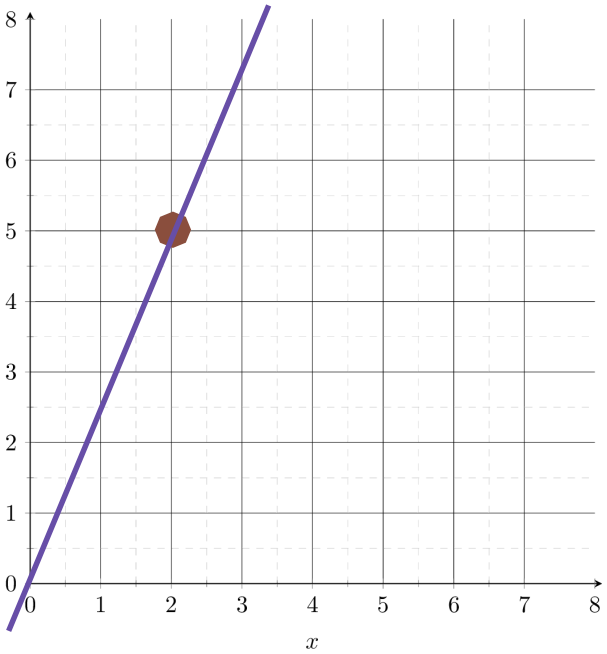


	x	y
p1	2	5
p2	3	6
p3	4	7

- the MSE of a linear hypothesis:

$$J(\theta) = \frac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

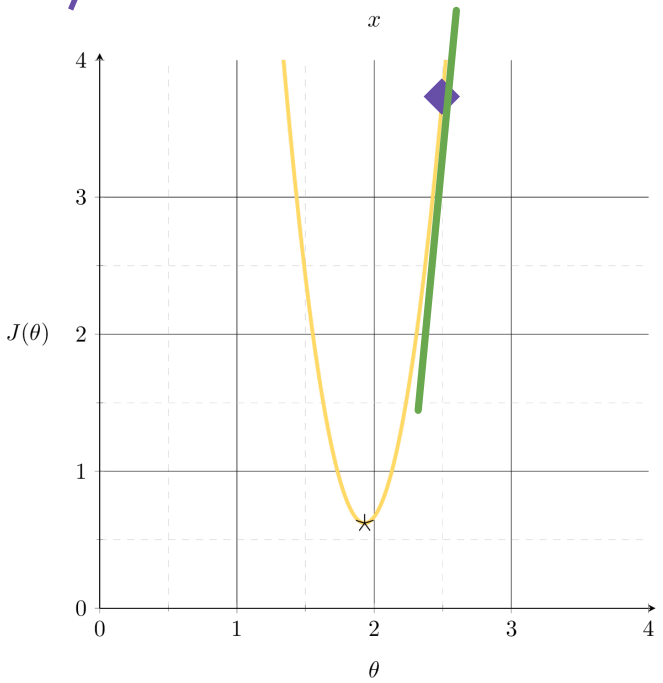
$$= \frac{1}{3}\left[\quad J_1 \quad + \quad J_2 \quad + \; J_3\right]$$

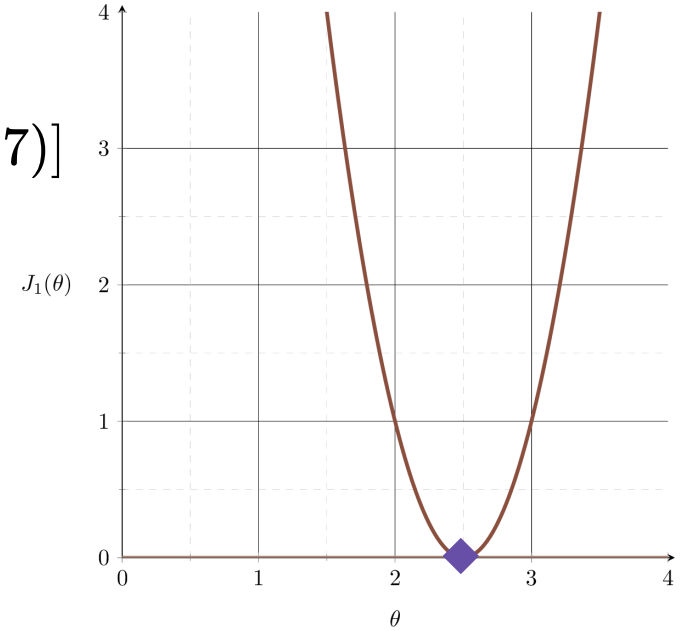


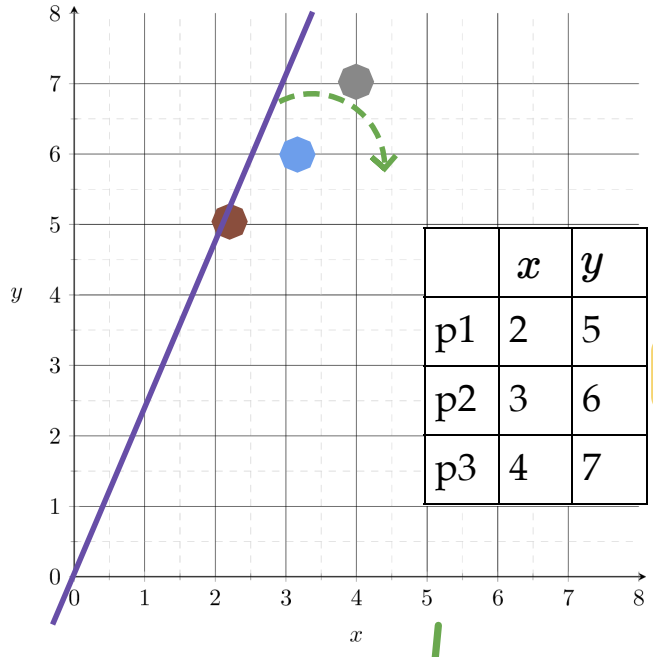
- and its gradient w.r.t. θ:

$$\nabla_\theta J = \frac{1}{3}\left[4(2\theta - 5) + 6(3\theta - 6) + 8(4\theta - 7)\right]$$

$$= \frac{1}{3}\left[\quad \nabla_\theta J_1 \quad + \quad \nabla_\theta J_2 \quad + \; \nabla_\theta J_3 \quad\right]$$





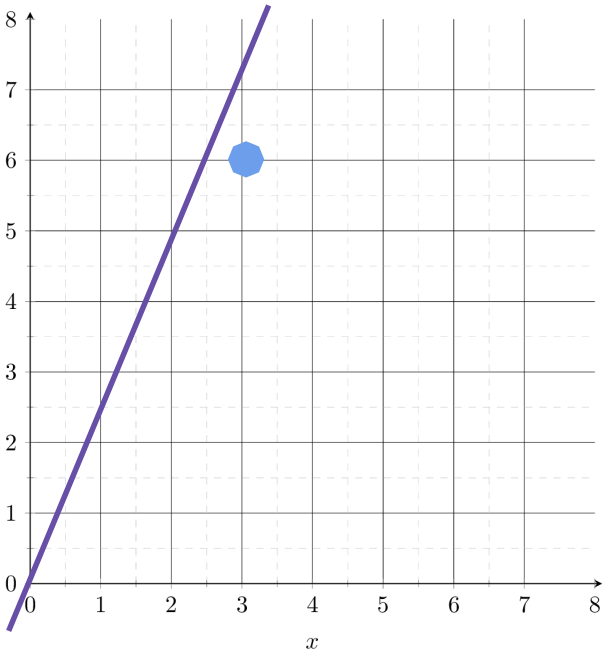


	x	y
p1	2	5
p2	3	6
p3	4	7

- the MSE of a linear hypothesis:

$$J(\theta) = \frac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

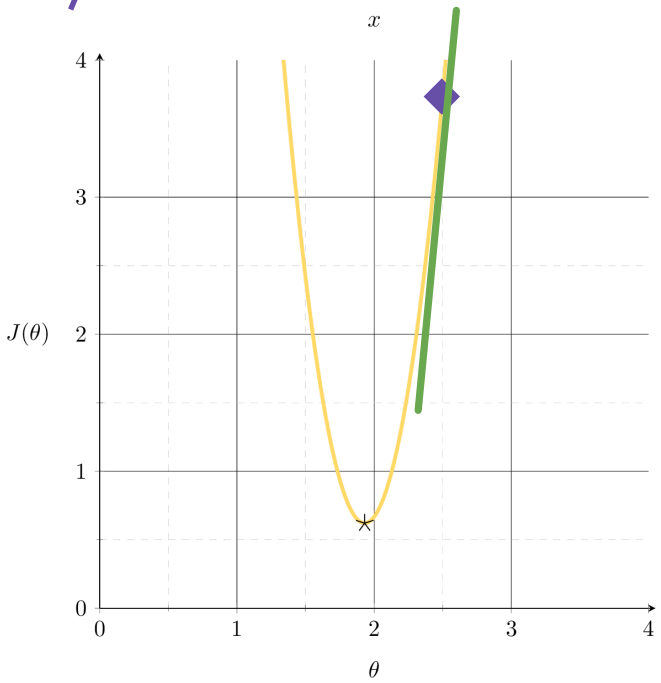
$$= \frac{1}{3}\left[\quad J_1 \quad + \quad J_2 \quad + \quad J_3\right]$$

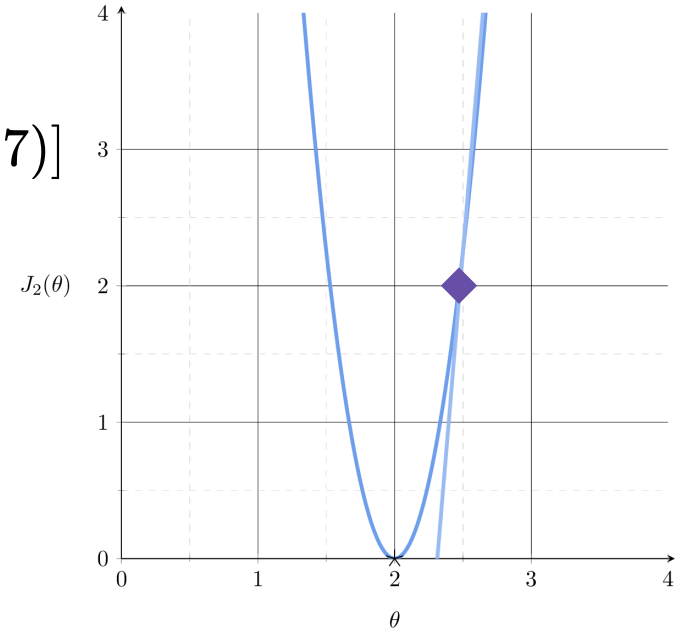


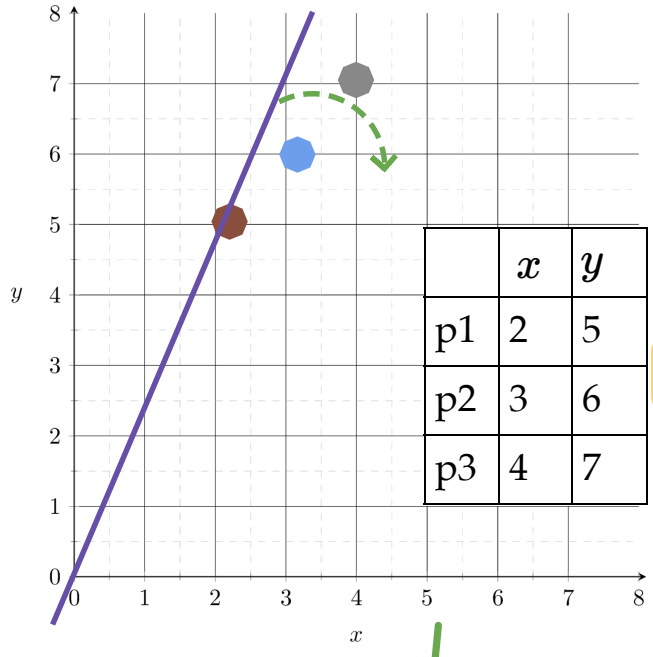
- and its gradient w.r.t. θ:

$$\nabla_\theta J = \frac{1}{3}[4(2\theta - 5) + 6(3\theta - 6) + 8(4\theta - 7)]$$

$$= \frac{1}{3}\left[\quad \nabla_\theta J_1 \quad + \quad \nabla_\theta J_2 \quad + \quad \nabla_\theta J_3 \quad\right]$$







	x	y
p1	2	5
p2	3	6
p3	4	7

- the MSE of a linear hypothesis:

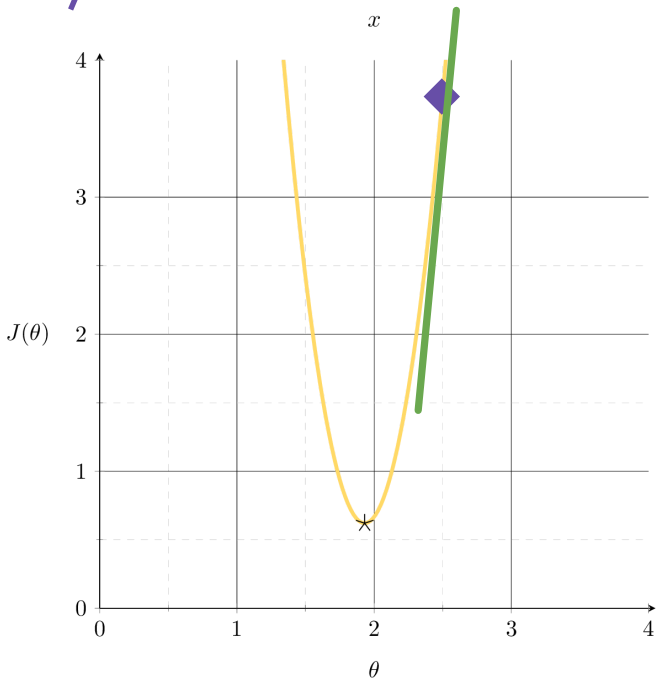
$$J(\theta) = \frac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

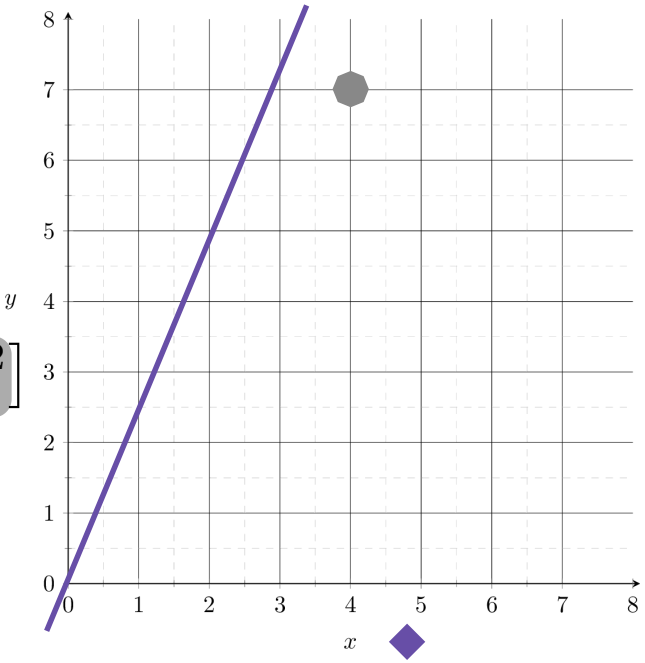
$$= \frac{1}{3}\left[\quad J_1 \quad + \quad J_2 \quad + \; J_3\right]$$

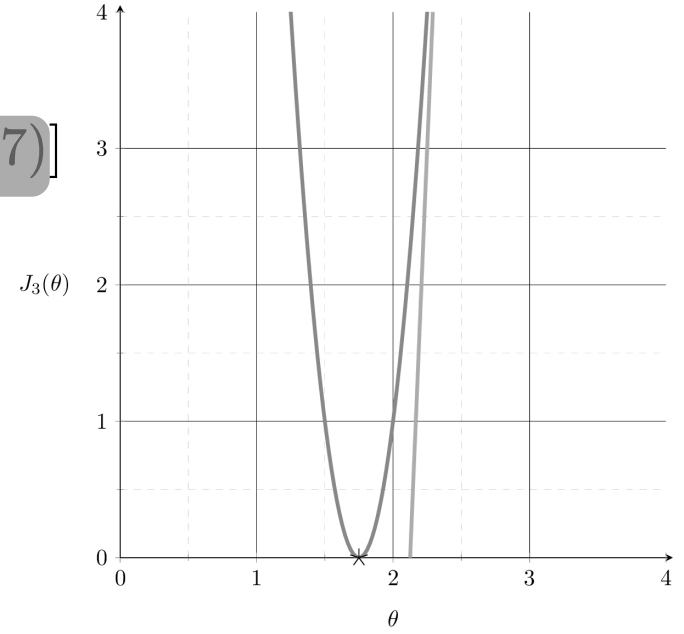
- and its gradient w.r.t. θ:

$$\nabla_\theta J = \frac{1}{3}\left[4(2\theta - 5) + 6(3\theta - 6) + 8(4\theta - 7)\right]$$

$$= \frac{1}{3}\left[\quad \nabla_\theta J_1 \quad + \quad \nabla_\theta J_2 \quad + \; \nabla_\theta J_3 \quad\right]$$







Gradient of an ML objective

Using our example data set,

- the MSE of a linear hypothesis:

$$J(\theta) = \tfrac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

- and its gradient w.r.t. θ:

$$\nabla_\theta J = \tfrac{1}{3}[4(2\theta - 5) + 6(3\theta - 6) + 8(4\theta - 7)]$$

Using any dataset,

- the MSE of a linear hypothesis:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^\top x^{(i)} - y^{(i)}\right)^2$$

- and its gradient w.r.t. θ:

$$\nabla_\theta J(\theta) = \frac{1}{n}\sum_{i=1}^{n} 2\left(\theta^\top x^{(i)} - y^{(i)}\right) x^{(i)}$$

Gradient of an ML objective

- the MSE of a linear hypothesis:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\theta^\top x^{(i)} - y^{(i)}\right)^2$$

- and its gradient w.r.t. θ:

$$\nabla_\theta J(\theta) = \frac{1}{n}\sum_{i=1}^{n} 2\left(\theta^\top x^{(i)} - y^{(i)}\right) x^{(i)}$$

In general,

- An ML objective function is a finite sum

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

- and its gradient w.r.t. θ:

$$\nabla_\theta J(\theta) = \nabla\left(\frac{1}{n}\sum_{i=1}^{n} J_i(\theta)\right) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta J_i(\theta)$$

👆

👋 (gradient of the sum) = (sum of the gradient)

Gradient of an ML objective

In general,

- An ML objective function is a finite sum

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n} J_i(\theta)$$

loss incurred on a single i^{th} data point

- and its gradient w.r.t. θ:

$$\nabla_\theta J(\theta) = \nabla_\theta\left(\frac{1}{n}\sum_{i=1}^{n} J_i(\theta)\right) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta J_i(\theta)$$

need to add n of these, each $\nabla_\theta J_i(\theta) \in \mathbb{R}^d$

gradient info from a single i^{th} data point's loss

Costly in practice!

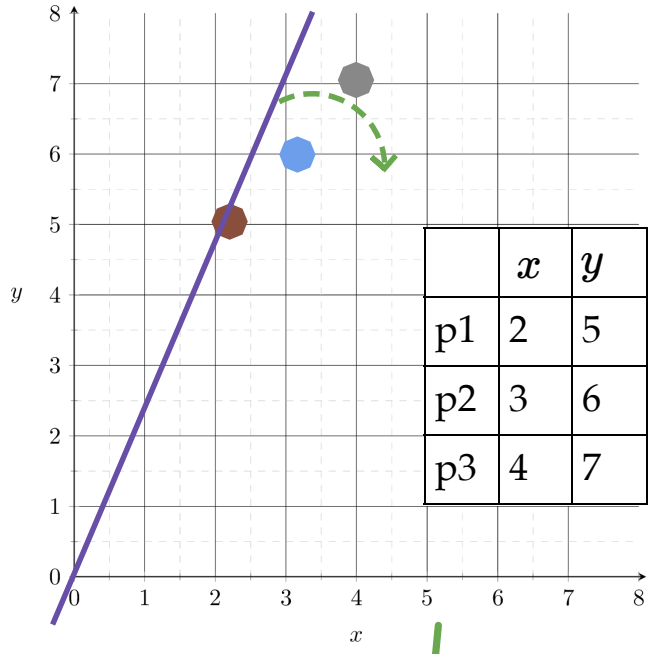
Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
  $t = t + 1$
  $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
until $\left|J(\theta^{(t)}) - J(\theta^{(t-1)})\right| < \epsilon$
return $\theta^{(t)}$

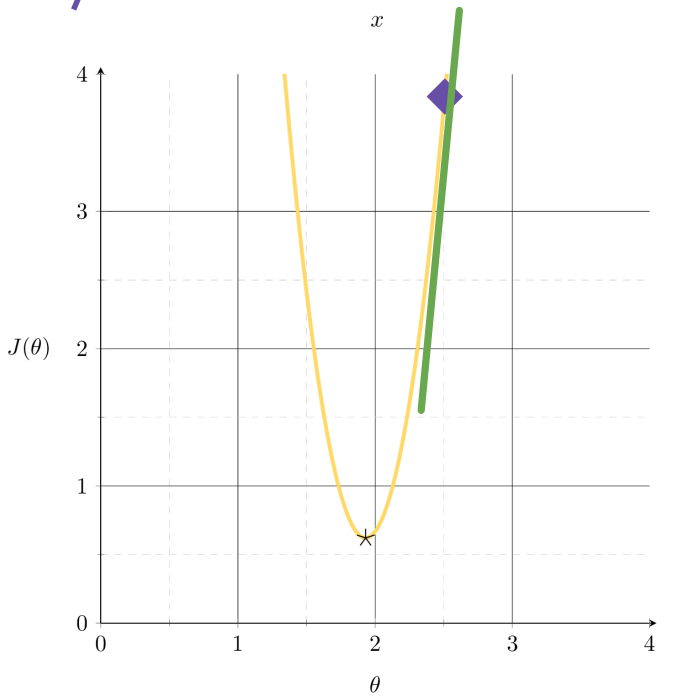
Algorithm 2 Stochastic Gradient Descent$(\theta_{\text{init}}, \eta, J, \nabla_\theta J, \epsilon)$

Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
  $t = t + 1$
  $i =$ randomly sample from $\{1, \ldots, n\}$
  $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J_i(\theta^{(t-1)})$
until $\left|J(\theta^{(t)}) - J(\theta^{(t-1)})\right| < \epsilon$
return $\theta^{(t)}$

$$\nabla_\theta J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta J_i(\theta) \approx \nabla_\theta J_i(\theta)$$



	x	y
p1	2	5
p2	3	6
p3	4	7



- the MSE of a linear hypothesis:

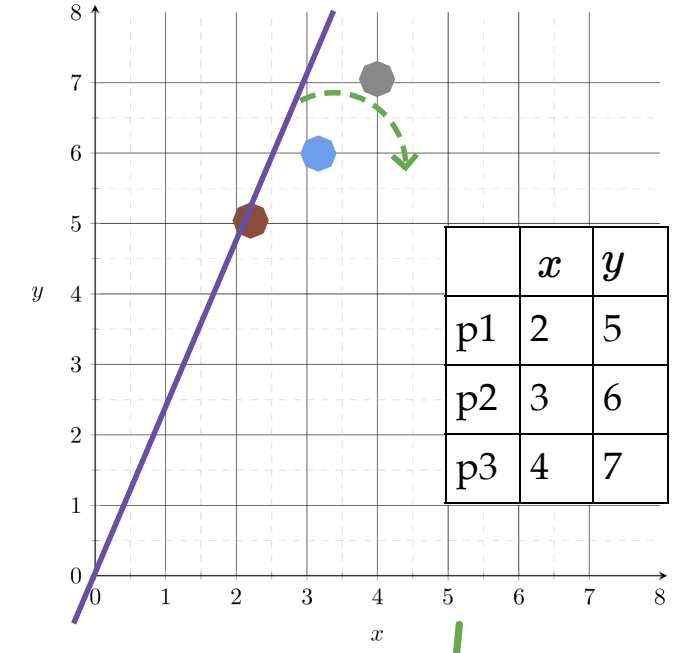
$$J(\theta) = \frac{1}{3}\left[(2\theta - 5)^2 + (3\theta - 6)^2 + (4\theta - 7)^2\right]$$

$$= \frac{1}{3}\left[\quad J_1 \quad + \quad J_2 \quad + \; J_3\;\right]$$

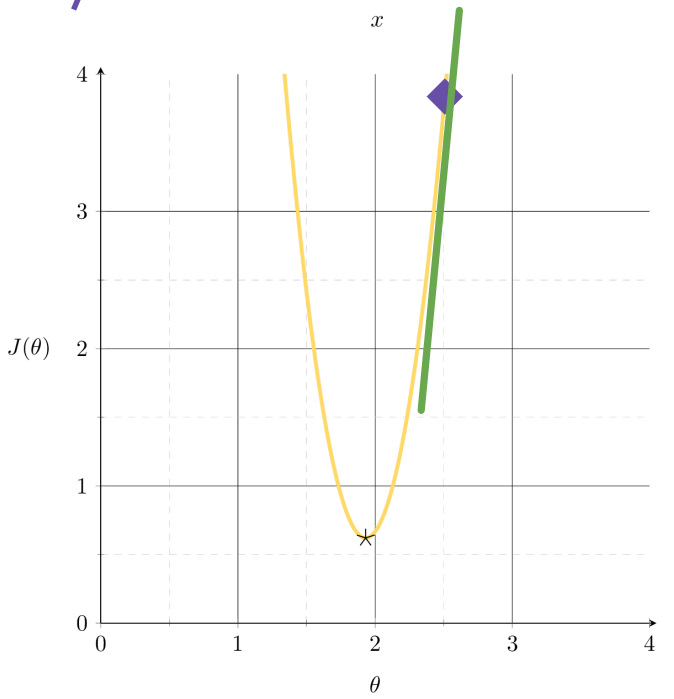
- and its gradient w.r.t. θ:

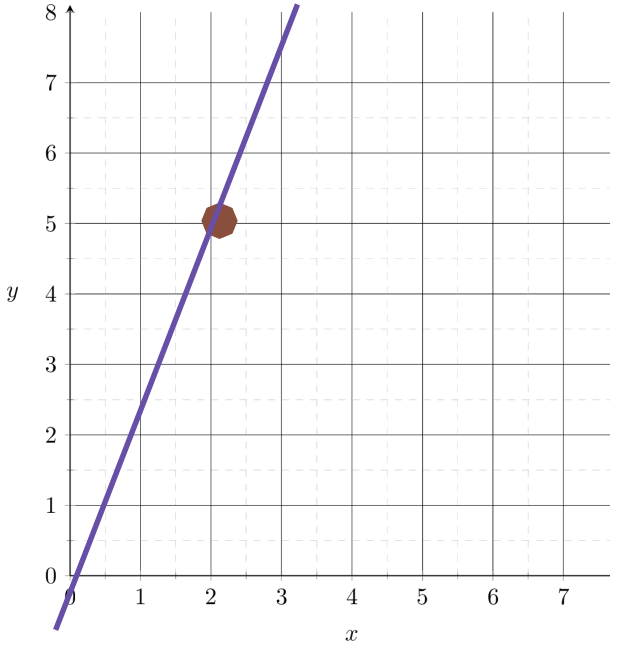
$$\nabla_\theta J = \frac{1}{3}\left[4(2\theta - 5) + 6(3\theta - 6) + 8(4\theta - 7)\right]$$

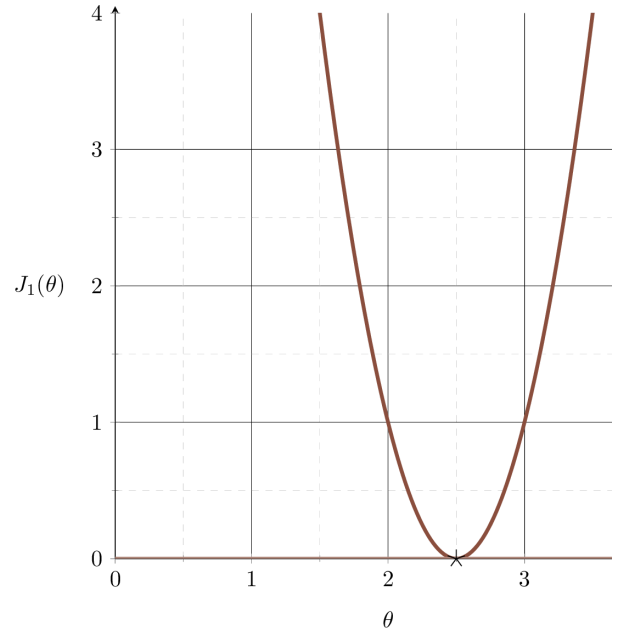
$$= \frac{1}{3}\left[\quad \nabla_\theta J_1 \quad + \quad \nabla_\theta J_2 \quad + \; \nabla_\theta J_3 \quad\right]$$

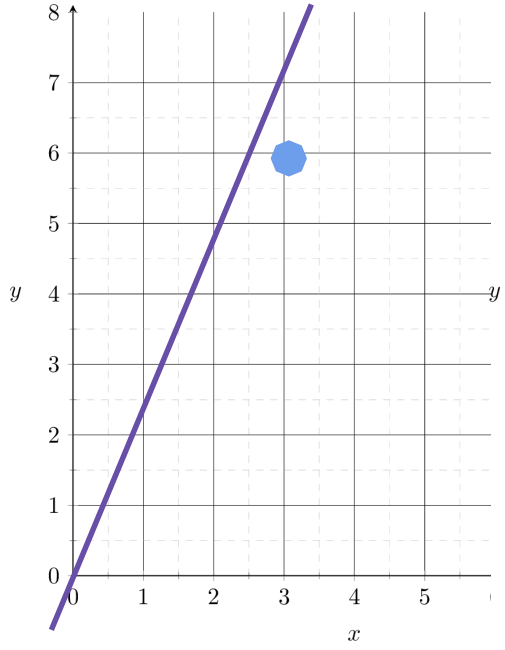


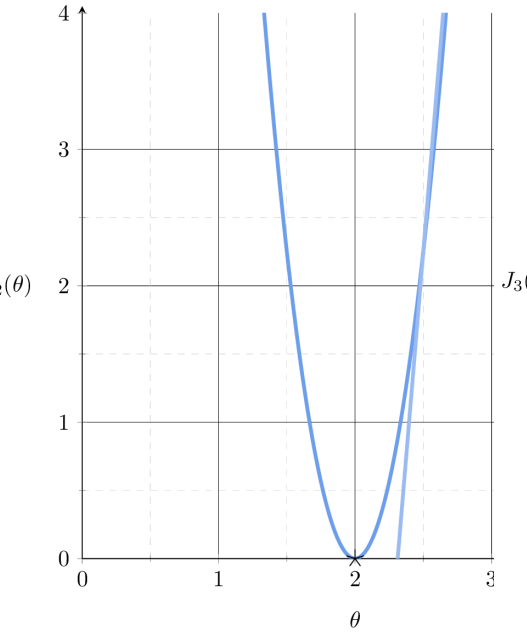
	x	y
p1	2	5
p2	3	6
p3	4	7

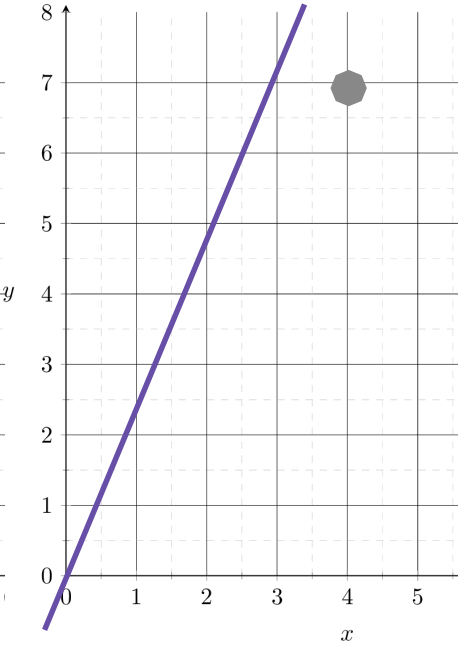


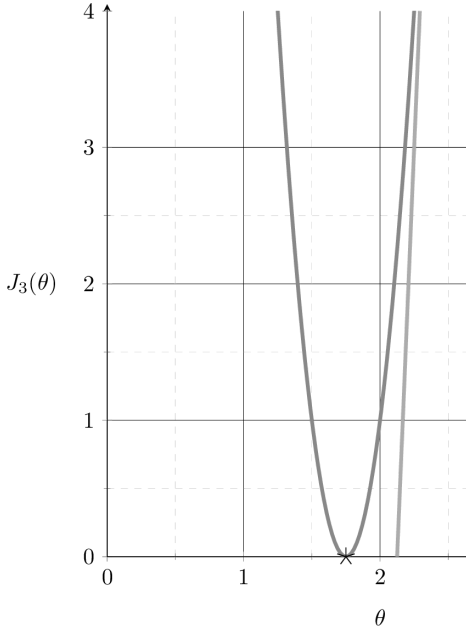






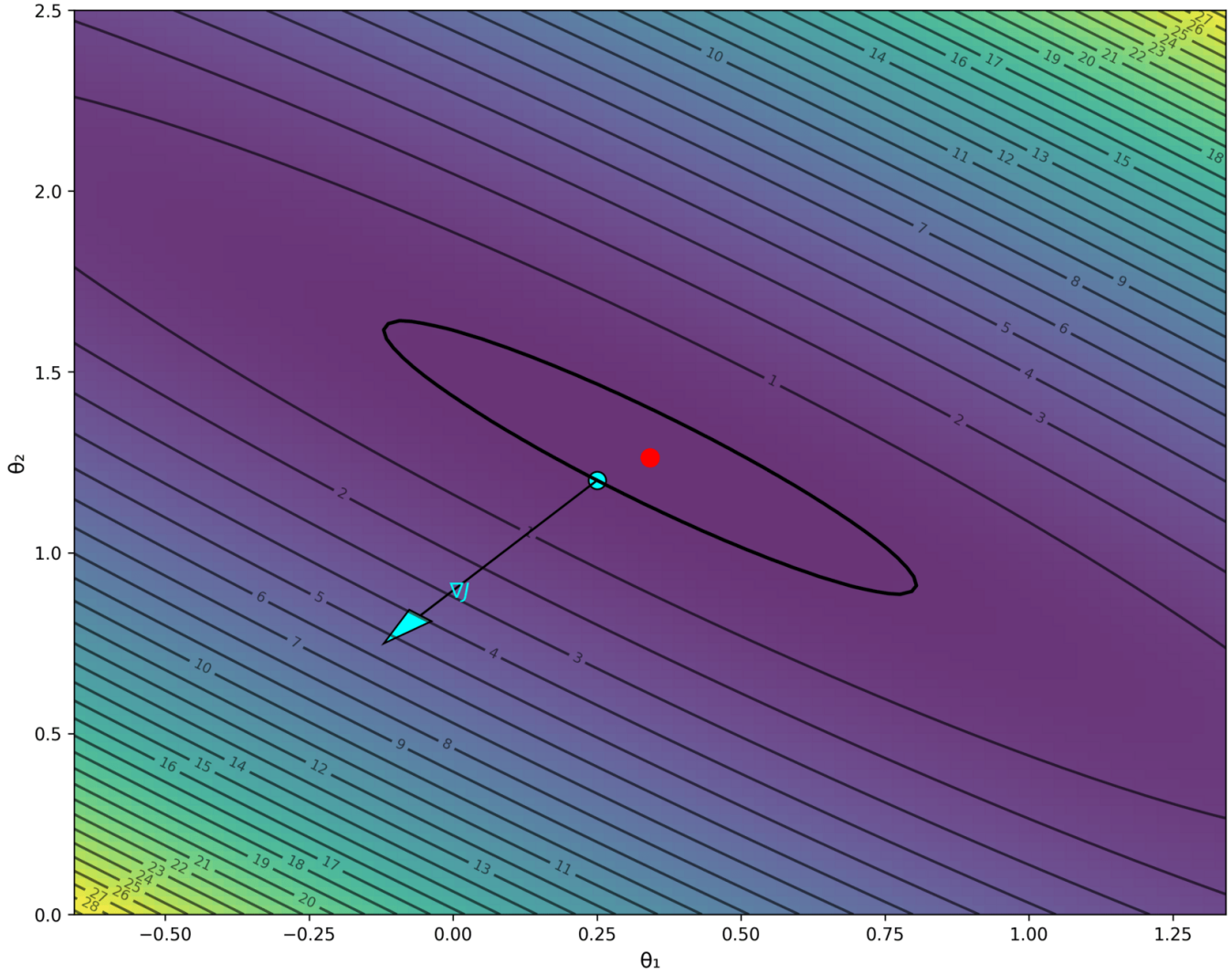






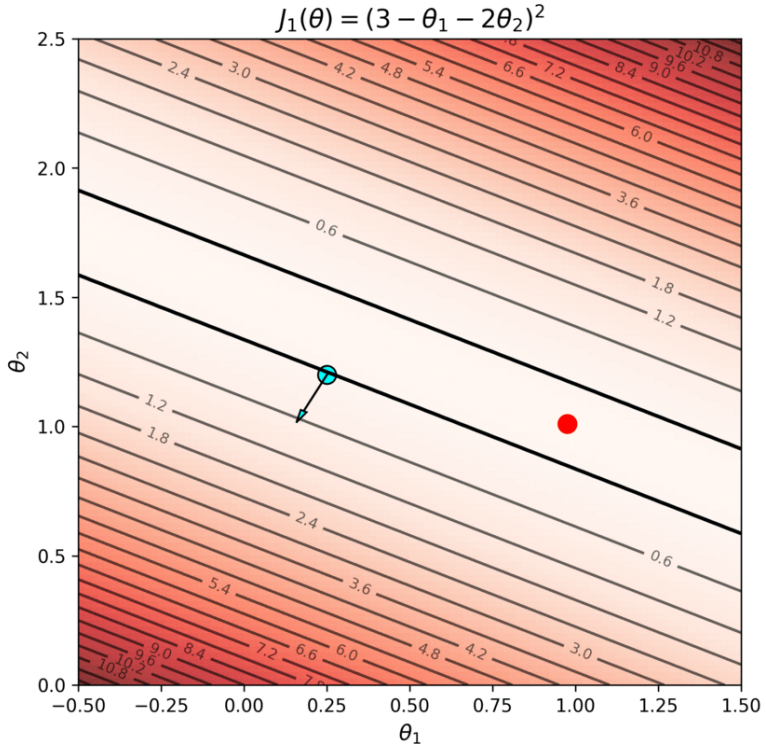
	x1	x2	y
p1	1	2	3
p2	2	1	2
p3	3	4	6

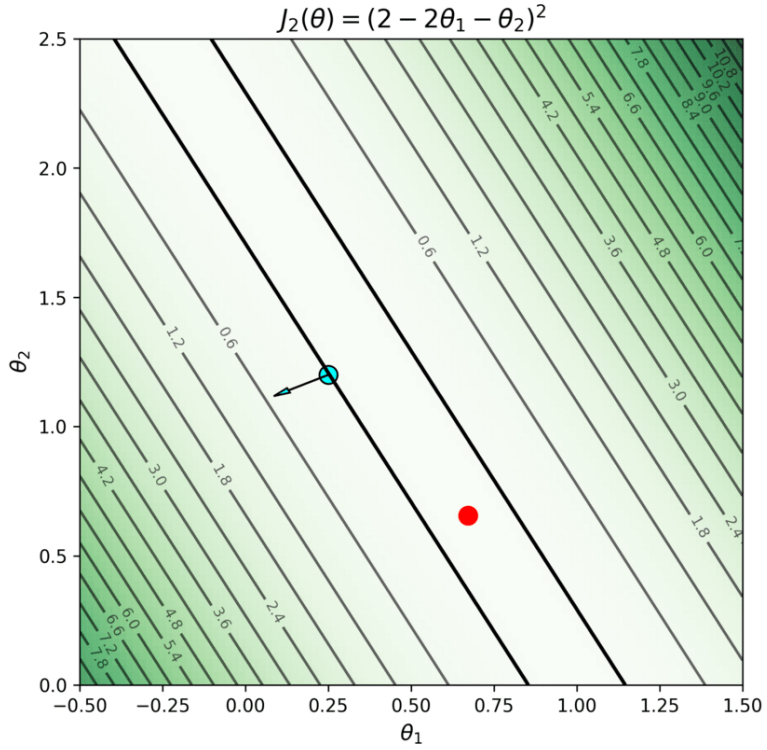
$$J(\theta) = \frac{1}{3}\left[(3 - \theta_1 - 2\theta_2)^2 + (2 - 2\theta_1 - \theta_2)^2 + (6 - 3\theta_1 - 4\theta_2)^2\right]$$

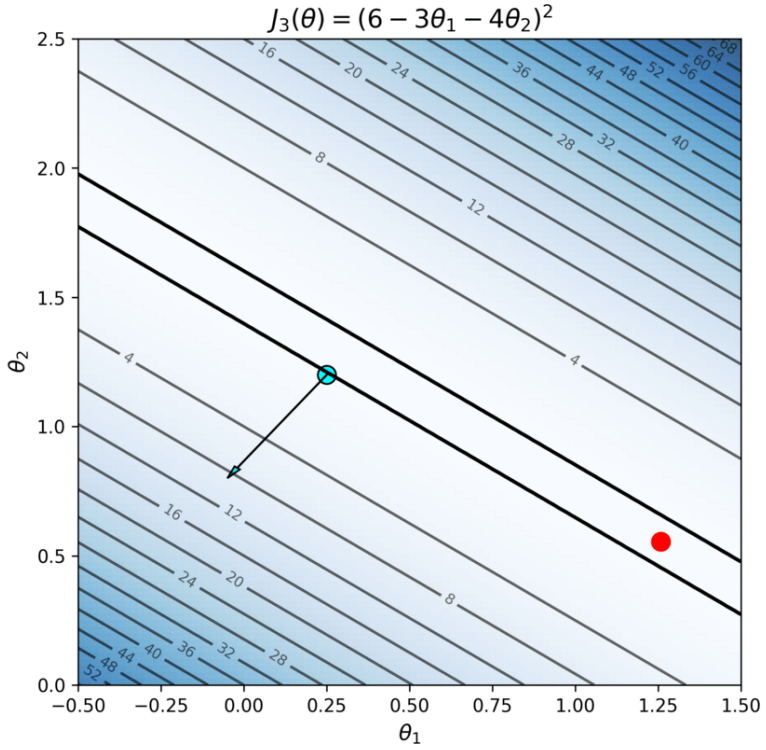


	x1	x2	y
p1	1	2	3
p2	2	1	2
p3	3	4	6

$$J(\theta) = \frac{1}{3}\left[(3-\theta_1-2\theta_2)^2 + (2-2\theta_1-\theta_2)^2 + (6-3\theta_1-4\theta_2)^2\right]$$

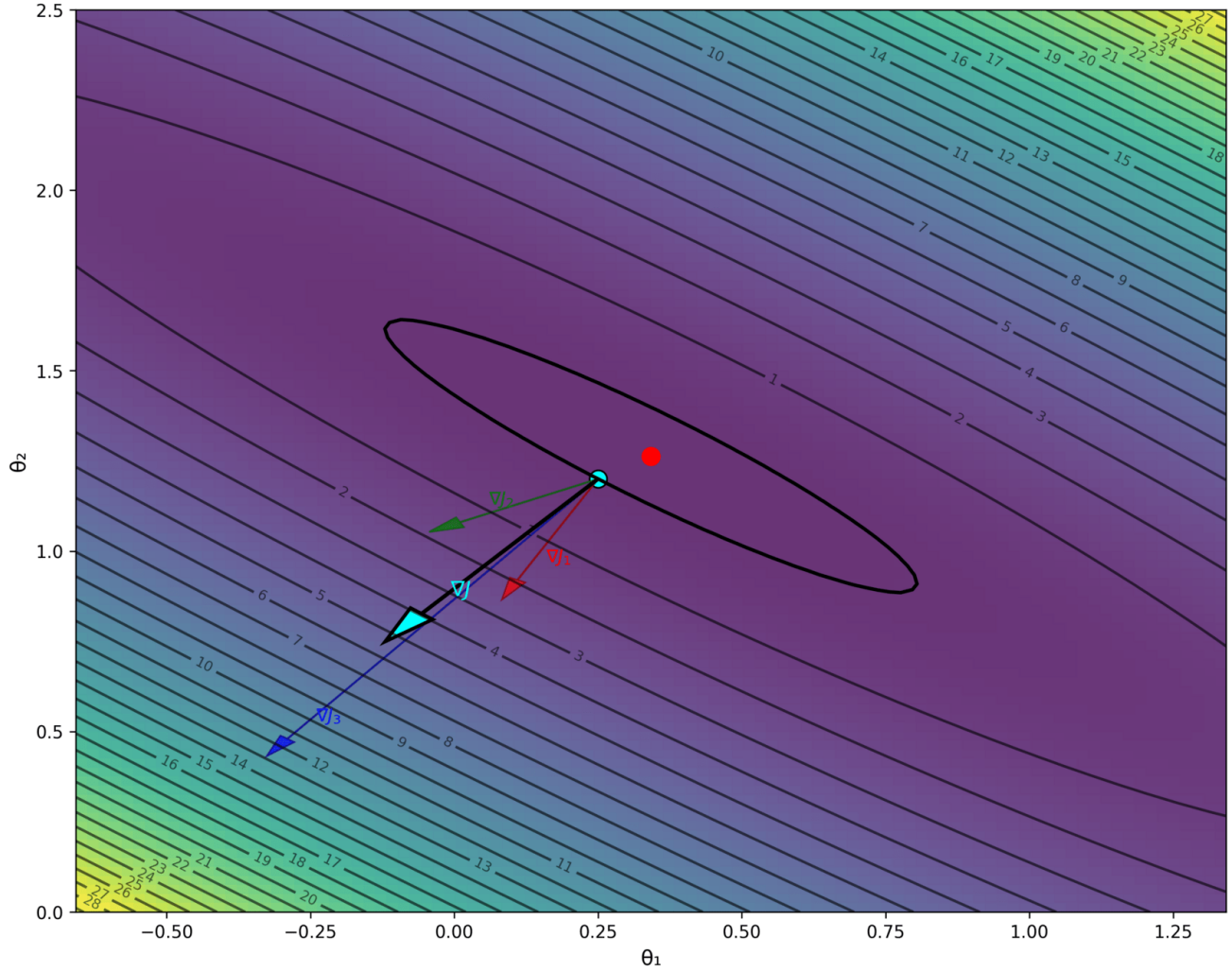






	x1	x2	y
p1	1	2	3
p2	2	1	2
p3	3	4	6

$$J(\theta) = \frac{1}{3}\left[(3-\theta_1-2\theta_2)^2 + (2-2\theta_1-\theta_2)^2 + (6-3\theta_1-4\theta_2)^2\right]$$

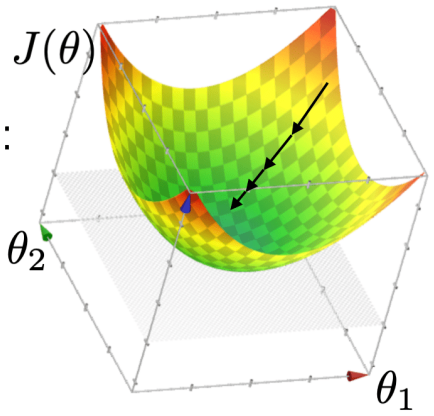


Compared with GD, SGD:

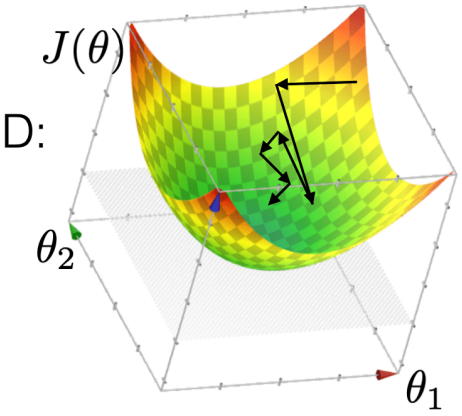
$$\nabla_\theta J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta J_i(\theta) \approx \nabla_\theta J_i(\theta)$$

is more efficient

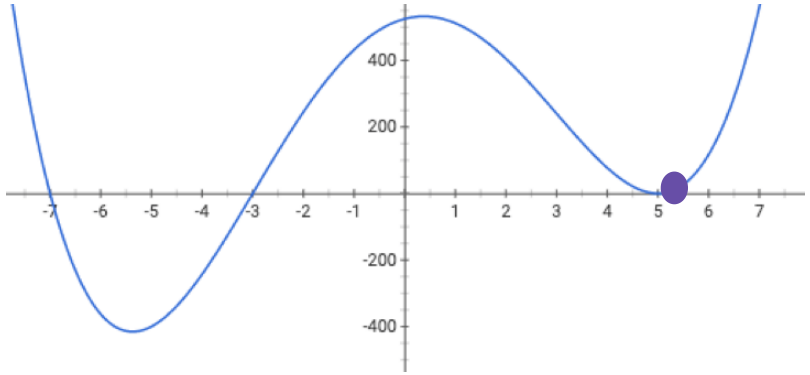
- GD:



- SGD:



is much more "random"



may get us out of a local min

Stochastic gradient descent performance

- Assumptions:
  - f is sufficiently "smooth"
  - f is convex
  - f has at least one global minimum
  - Run gradient descent for sufficient iterations
  - η is sufficiently small and satisfies additional "scheduling" condition
- Conclusion:

$$\sum_{t=1}^{\infty} \eta(t) = \infty \text{ and } \sum_{t=1}^{\infty} \eta(t)^2 < \infty$$

  - Stochastic gradient descent converges arbitrarily close to a global minimum of f with probability 1.

Algorithm 3 Mini-batch Gradient Descent$(\theta_{\text{init}}, \eta, b, J, \nabla_\theta J, \epsilon)$

1: Initialize $\theta^{(0)} = \theta_{\text{init}}$
2: Initialize $t = 0$
3: **repeat**
4: $\quad t = t + 1$
5: $\quad B =$ random mini-batch of size b from $\{1, \ldots, n\}$
6: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J_B(\theta^{(t-1)})$
7: **until** $\left|J(\theta^{(t)}) - J(\theta^{(t-1)})\right| < \epsilon$
8: **return** $\theta^{(t)}$

batch size

SGD

$\nabla_\theta J_i(\theta)$

mini-batch GD

$$\frac{1}{b}\sum_{i=1}^{b} \nabla_\theta J_i(\theta)$$

GD

$$\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta J_i(\theta) = \nabla_\theta J(\theta)$$

🥰more accurate gradient estimate

🥰stronger theoretical guarantee

🥺higher cost per parameter update

Summary

- Most ML methods can be formulated as optimization problems. We won't always be able to solve optimization problems analytically (in closed-form) nor efficiently.
- We can still use numerical algorithms to good effect. Lots of sophisticated ones available. Gradient descent is one of the simplest.
- The GD algorithm, iterative algorithm, keeps applying the parameter update rule.
- Under appropriate conditions (most notably, when objective function is convex, and when learning rate is small enough), GD can guarantee convergence to a global minimum.
- SGD is approximated GD, it uses a single data point to approximate the entire data set, it's more efficient, more random, and less guarantees.
- mini-batch GD is a middle ground between GD and SGD.

https://docs.google.com/forms/d/e/1FAIpQLScj9i83AI8TuhWDZXSjiWzX6gZpnPugjGsH-i3RdrBCtF-opg/viewform?embedded=true

We'd love to hear
your thoughts.

Thanks!