https://introml.mit.edu/

6.390 Intro to Machine Learning

Lecture 3: Gradient Descent Methods

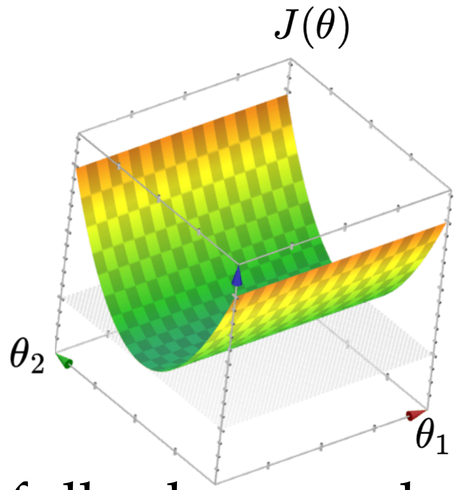
Shen Shen

Feb 17, 2026

3pm, Room 10-250

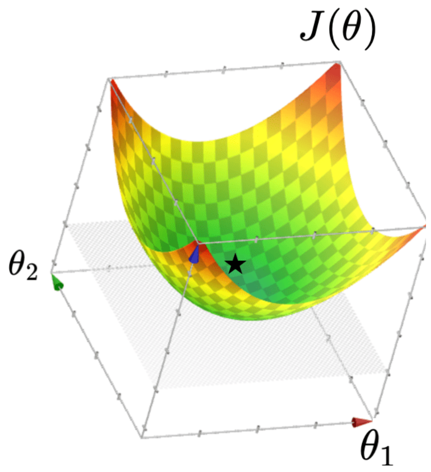
Slides and Lecture Recording

Recall:



When X is not full column rank

- $J(\theta)$ has a "flat" bottom
- This 👉 formula is not well-defined
- Infinitely many optimal hyperplanes



Typically, X is full column rank

- $J(\theta)$ "curves up" everywhere
- $\theta^* = \left(X^\top X\right)^{-1} X^\top Y$
- unique optimal hyperplane

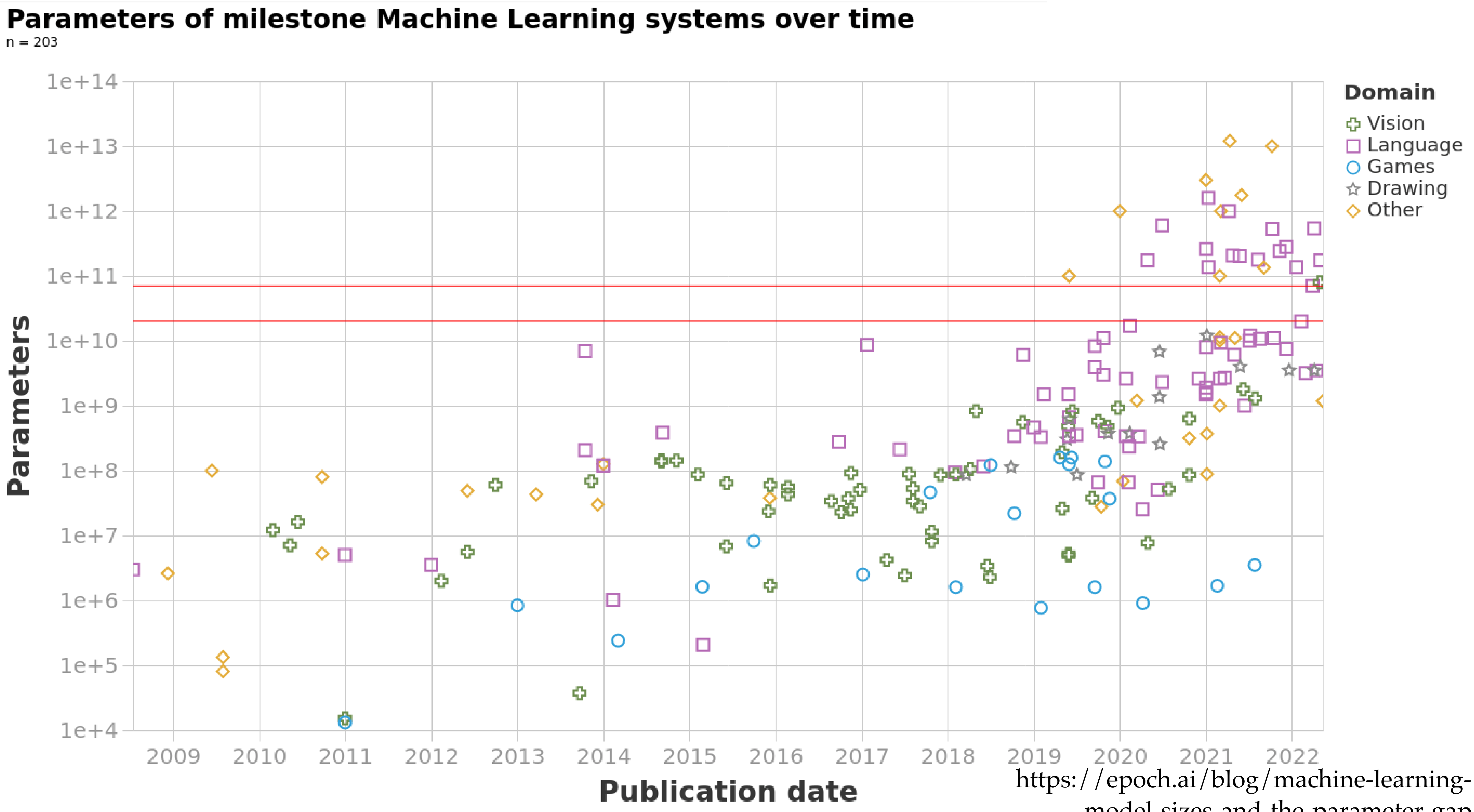
θ^*

No way yet to get any θ^*

numerically sensitive

θ^* can be costly to compute

(lab2 Q2.7)



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

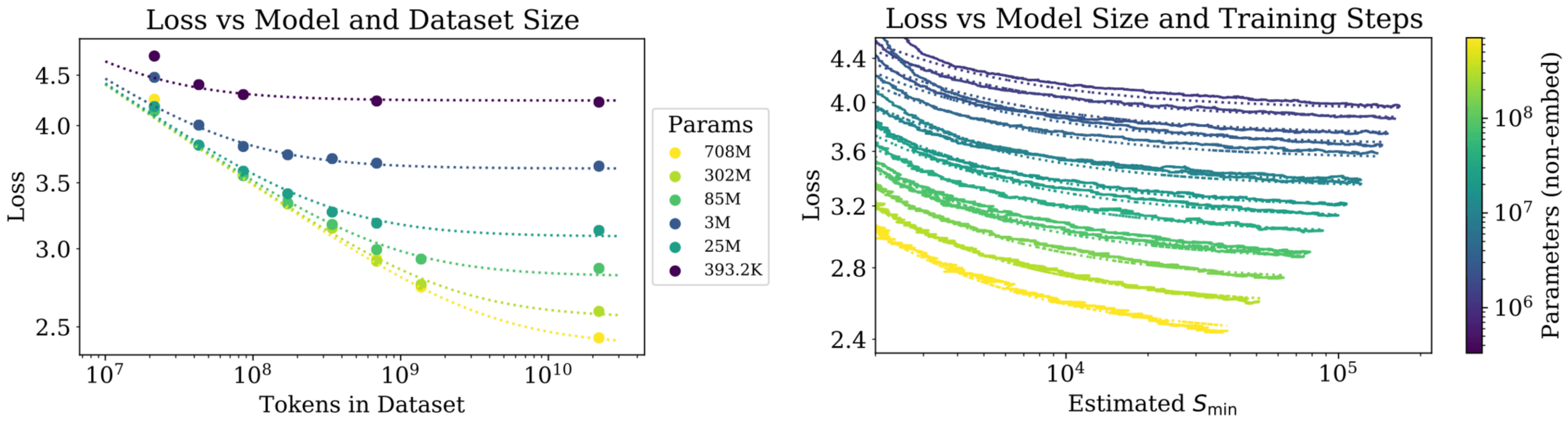


Figure 4 **Left**: The early-stopped test loss $L(N, D)$ varies predictably with the dataset size D and model size N according to Equation (1.5). **Right**: After an initial transient period, learning curves for all model sizes N can be fit with Equation (1.6), which is parameterized in terms of S_{min},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

https://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 our ML system
=> gradient descent methods

Outline

- Gradient descent (GD)
  - The gradient vector
  - GD algorithm
  - Gradient descent properties
- Stochastic gradient descent (SGD)

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.

The gradient may not always exist or be well-behaved.

Today, we have nice gradients unless otherwise specified.

3. The gradient can be symbolic or numerical.

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

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

its *symbolic* gradient:

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

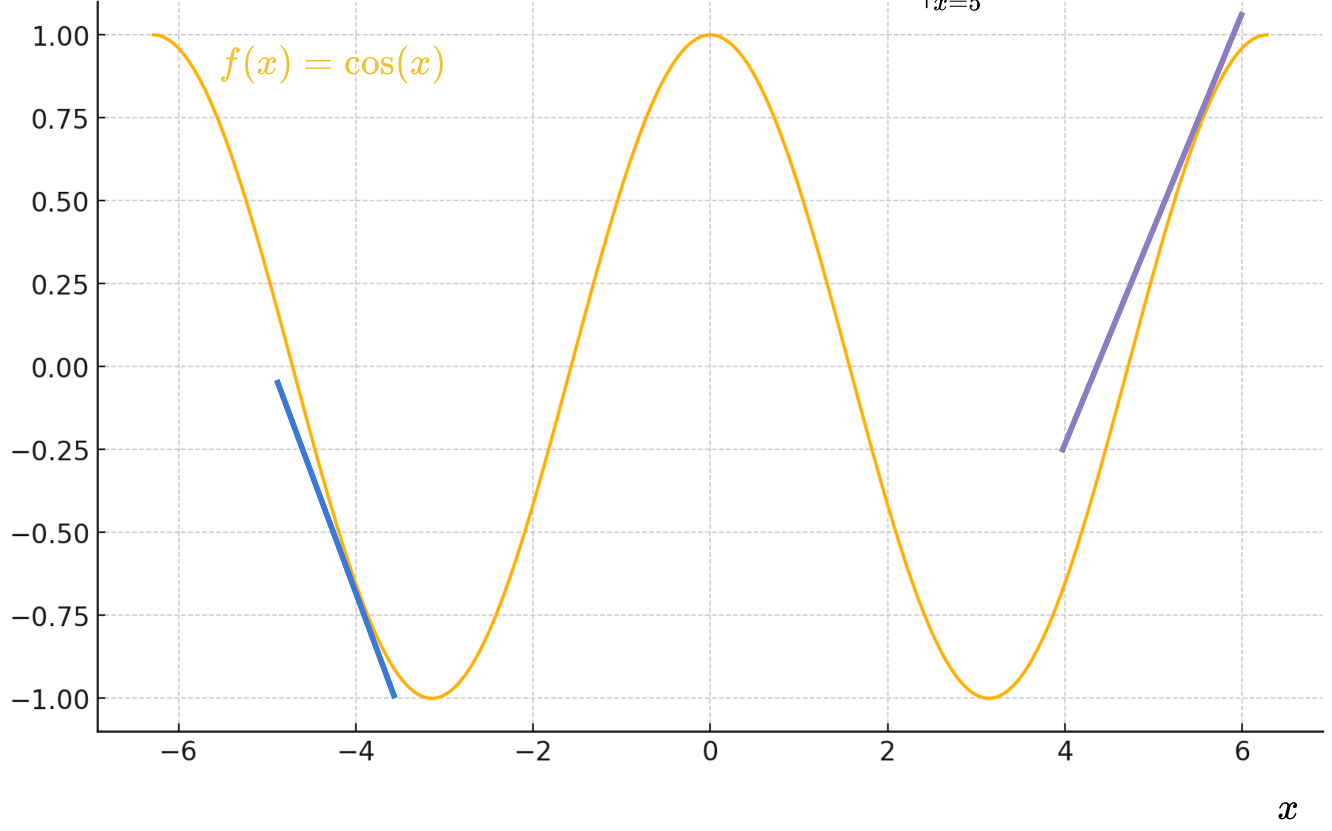
evaluating the symbolic gradient at a point gives a *numerical* gradient:

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

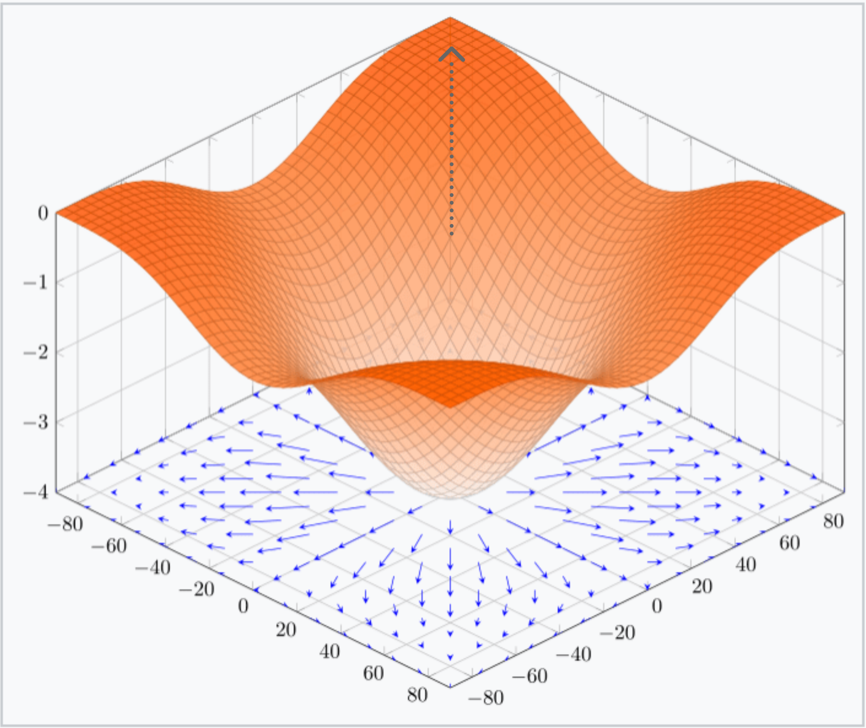
just like a derivative can be a function or a number.

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

$$\frac{d}{dx}\cos(x)\Big|_{x=5} = -\sin(5) \approx 0.9589$$

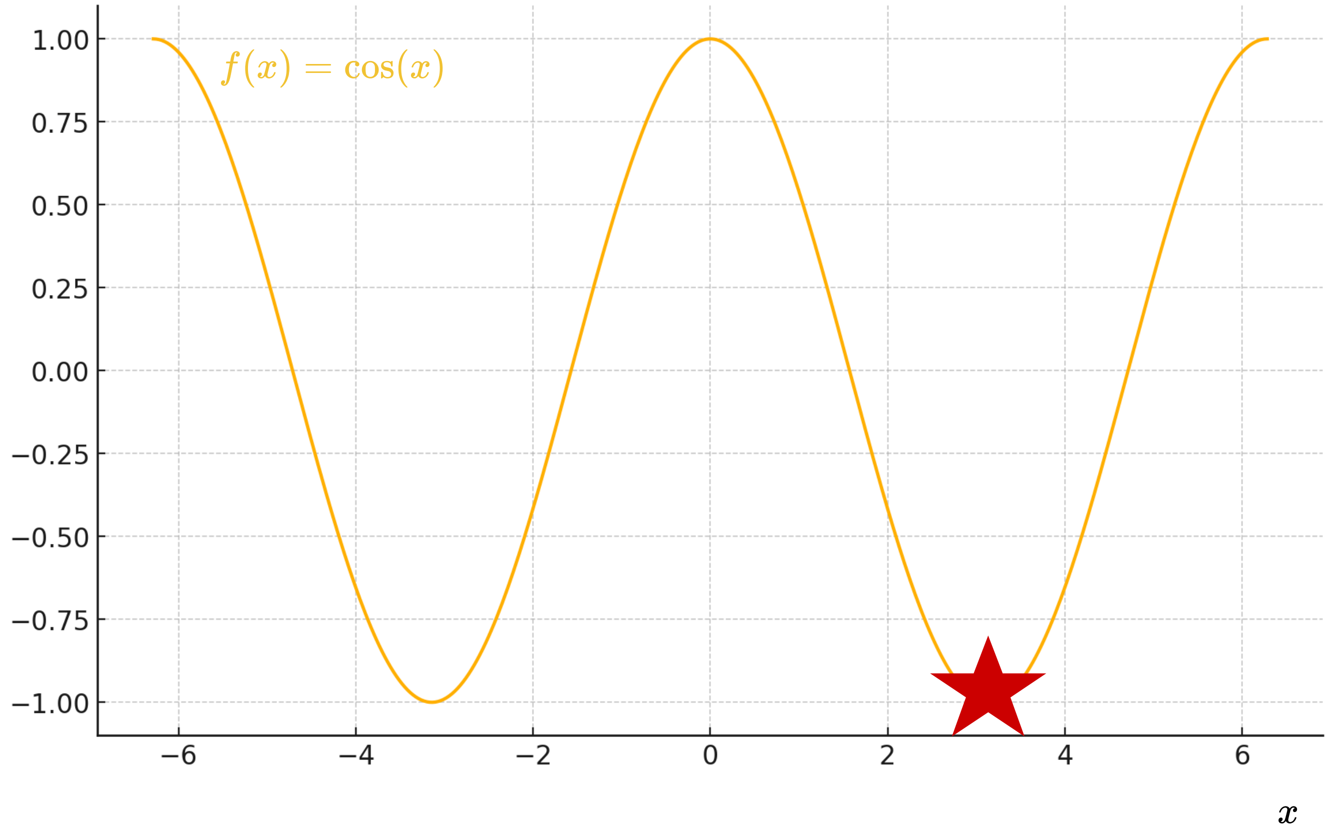


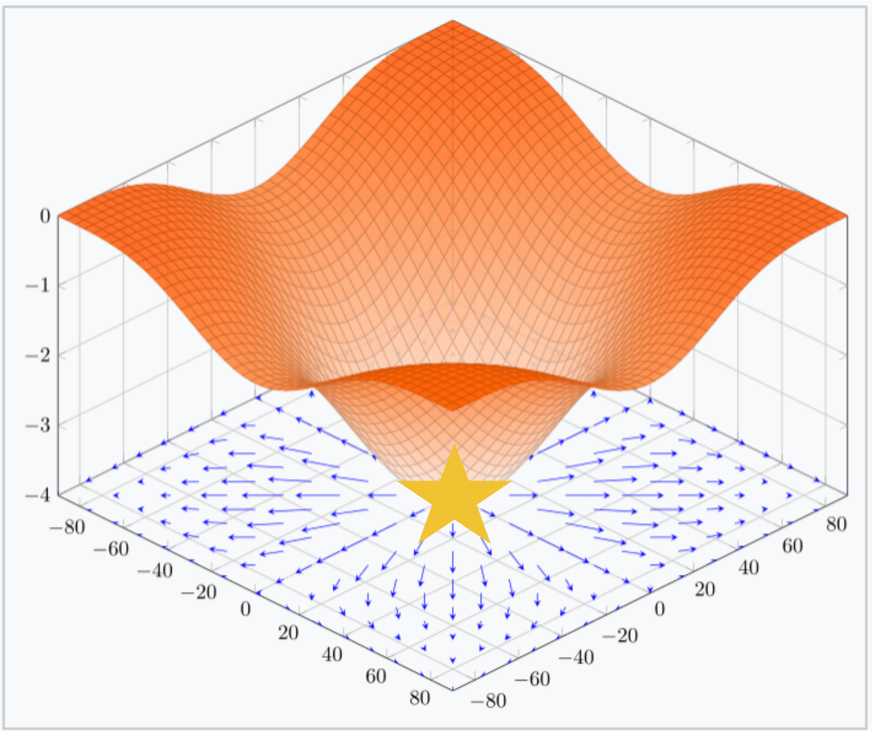
$$\frac{d}{dx}\cos(x)\Big|_{x=-4} = -\sin(-4) \approx -0.7568$$



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 $f(x,y) = -(\cos^2 x + \cos^2 y)^2$

assuming the function is unconstrained (domain $\mathbb{R}^d$)

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.

The gradient may not always exist or be well-behaved.

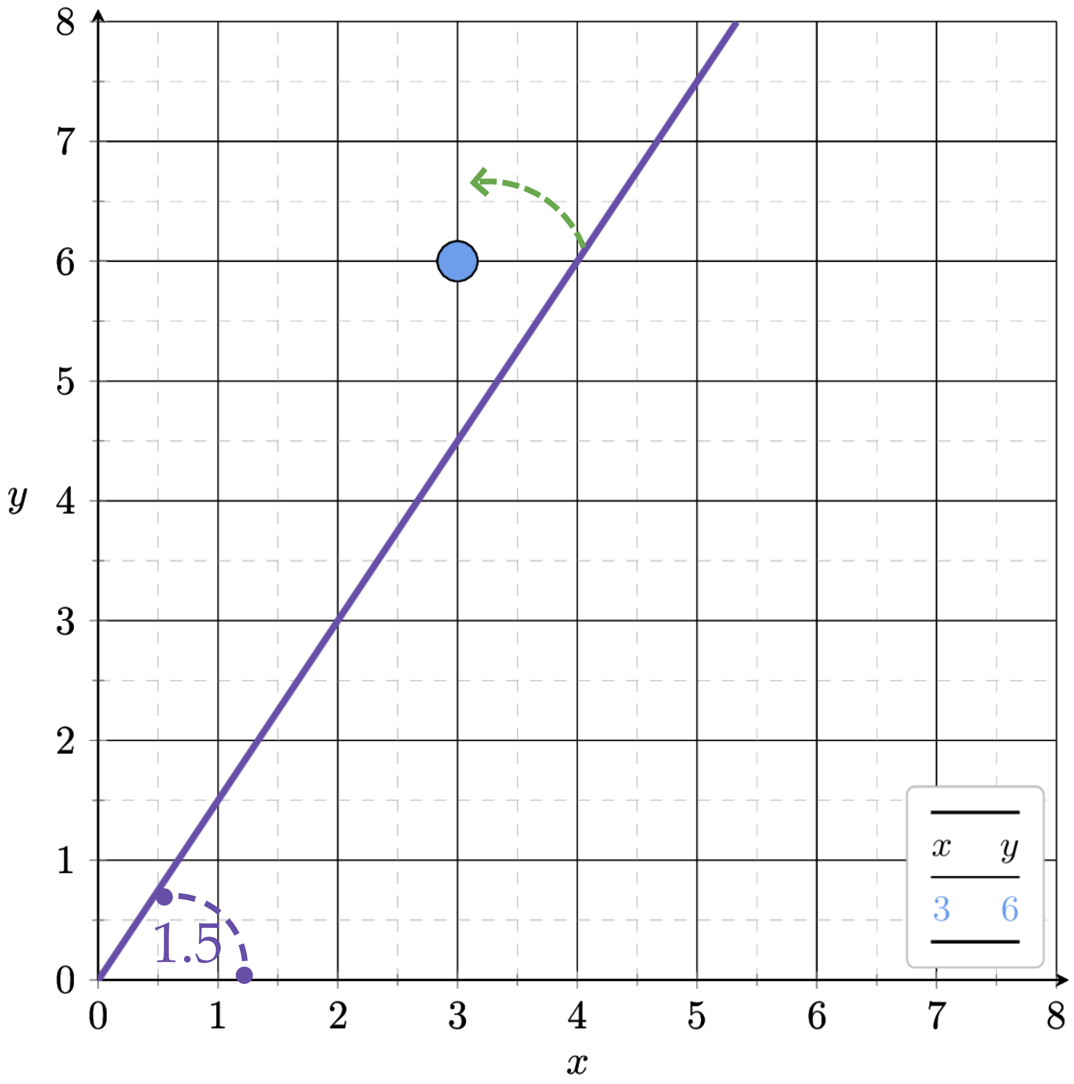
In our context, J plays the role of f, θ plays p.

Outline

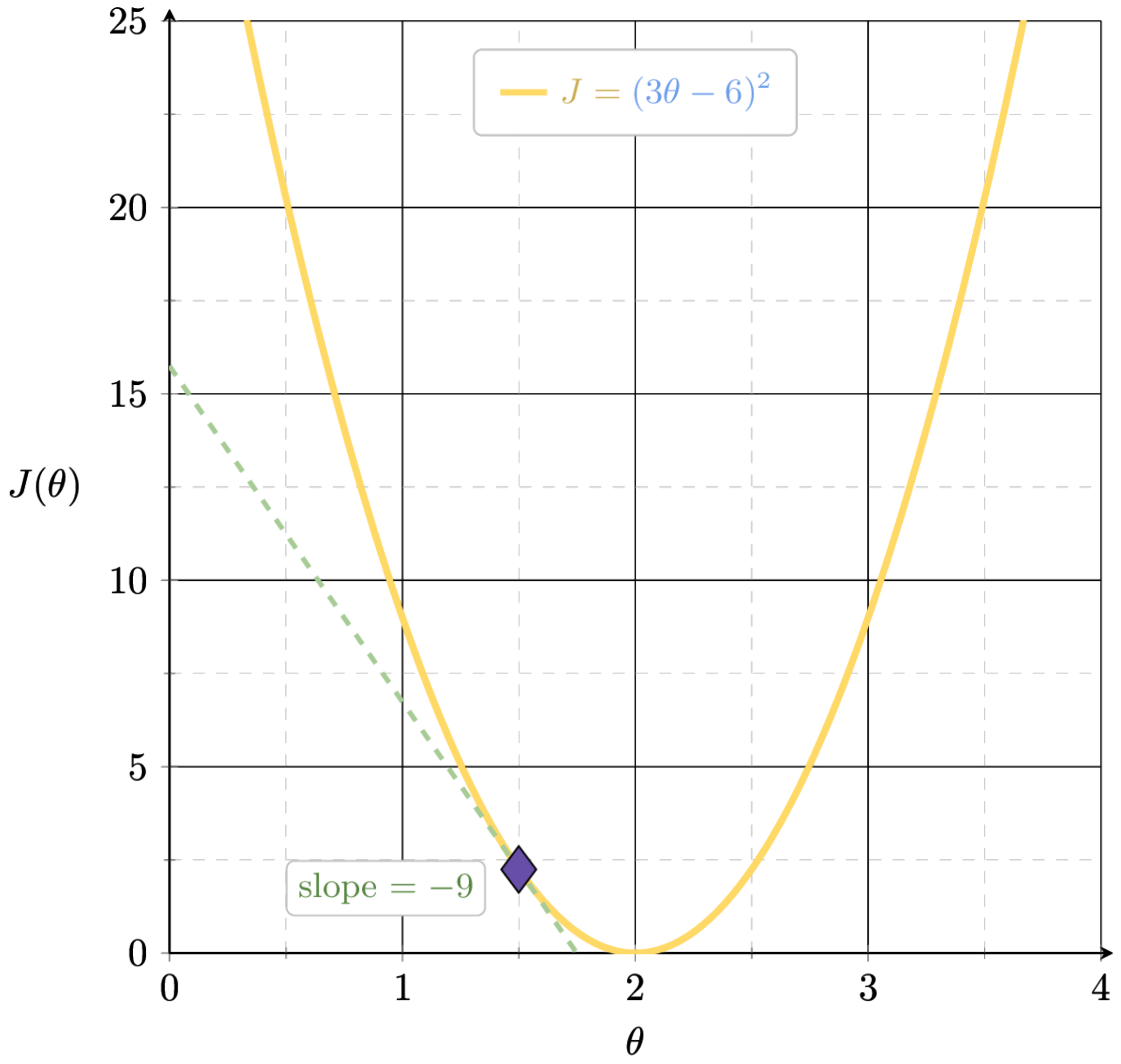
- Gradient descent (GD)
  - The gradient vector
  - GD algorithm
  - Gradient descent properties
- Stochastic gradient descent (SGD)

Example 1: fit a line (no offset) to minimize MSE

Suppose we fit $h = 1.5x$

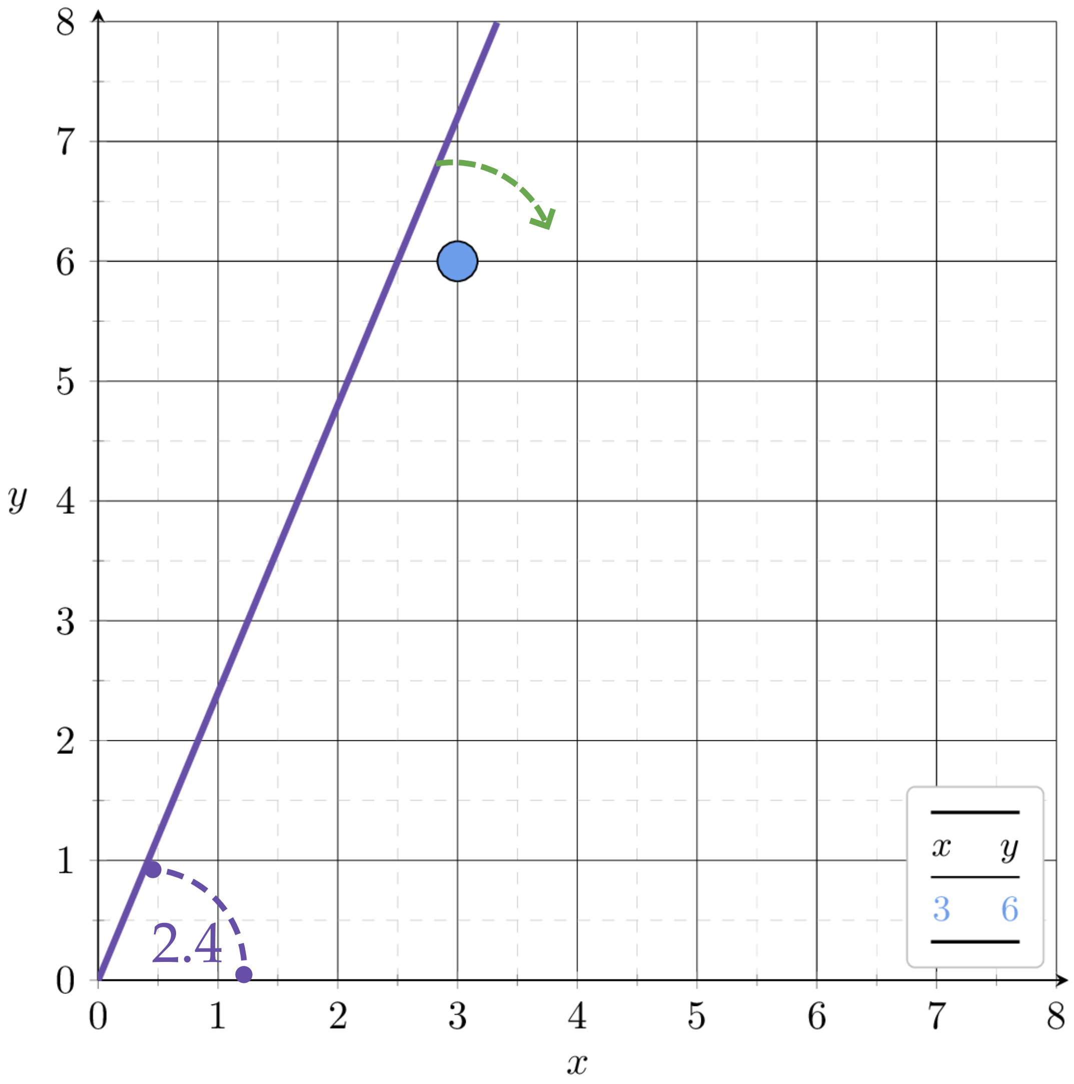


MSE could get better, by leveraging the gradient

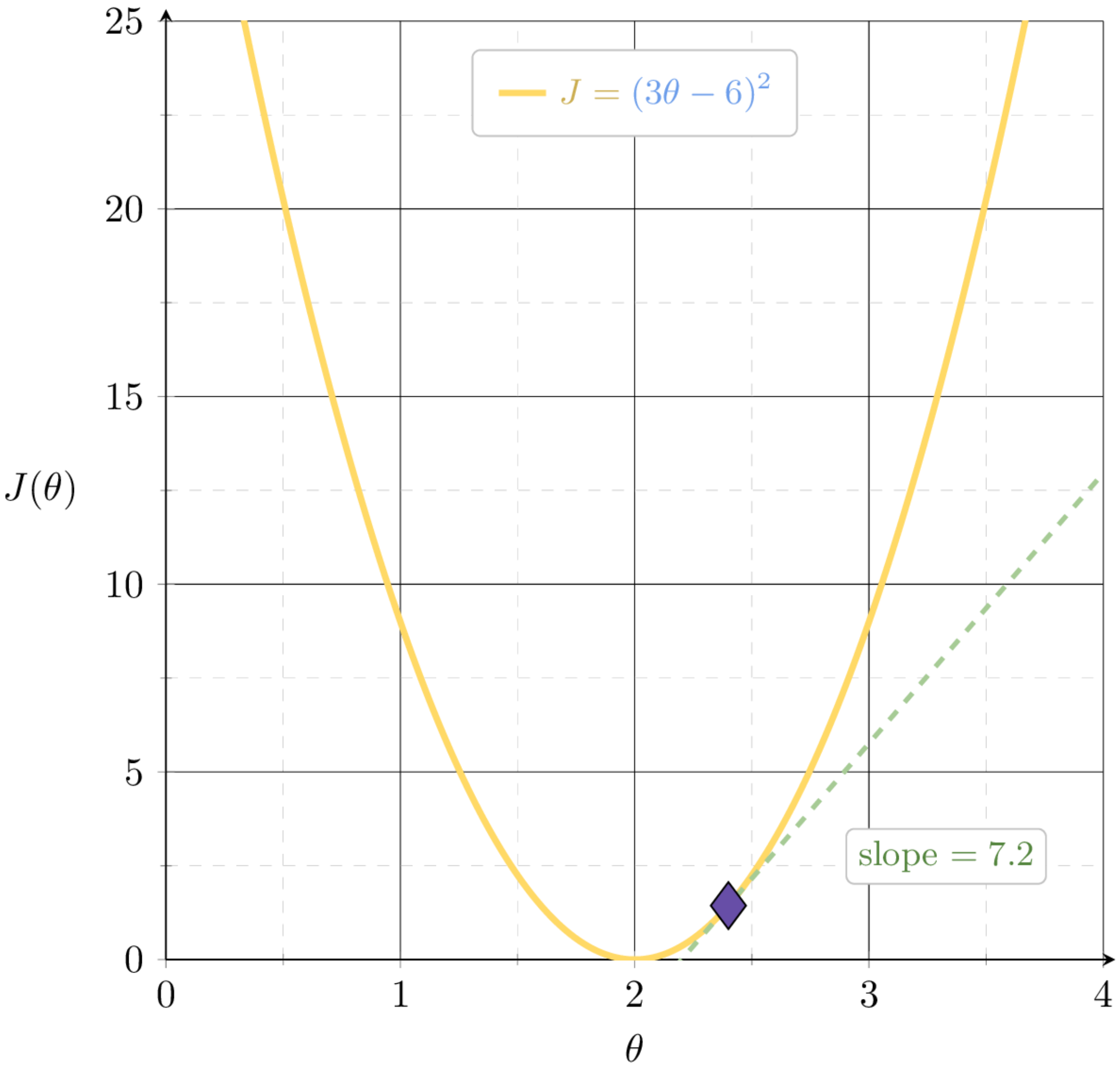


Example 1: fit a line (no offset) to minimize MSE

Suppose we fit $h = 2.4x$



MSE could get better, by leveraging the gradient



hyperparameters

initial guess of parameters

precision

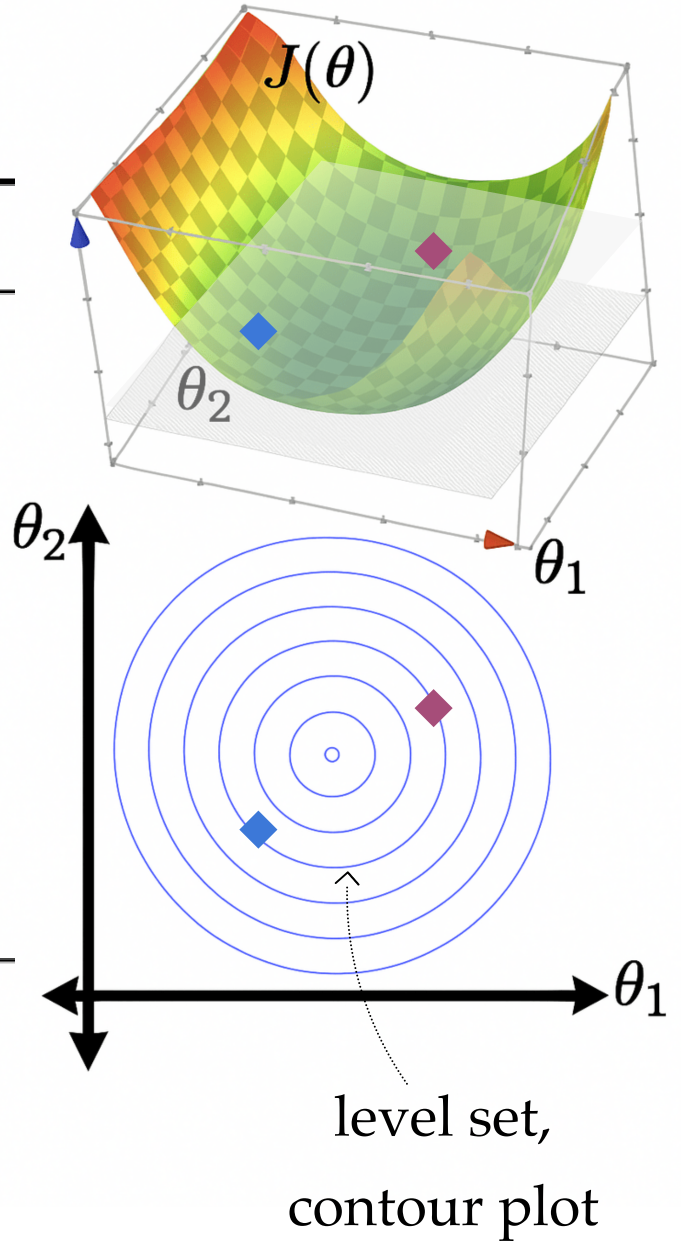
Algorithm 1 Gradient Descent(θ_{init}, η, J, ϵ)

learning rate

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** $\theta^{(t)}$

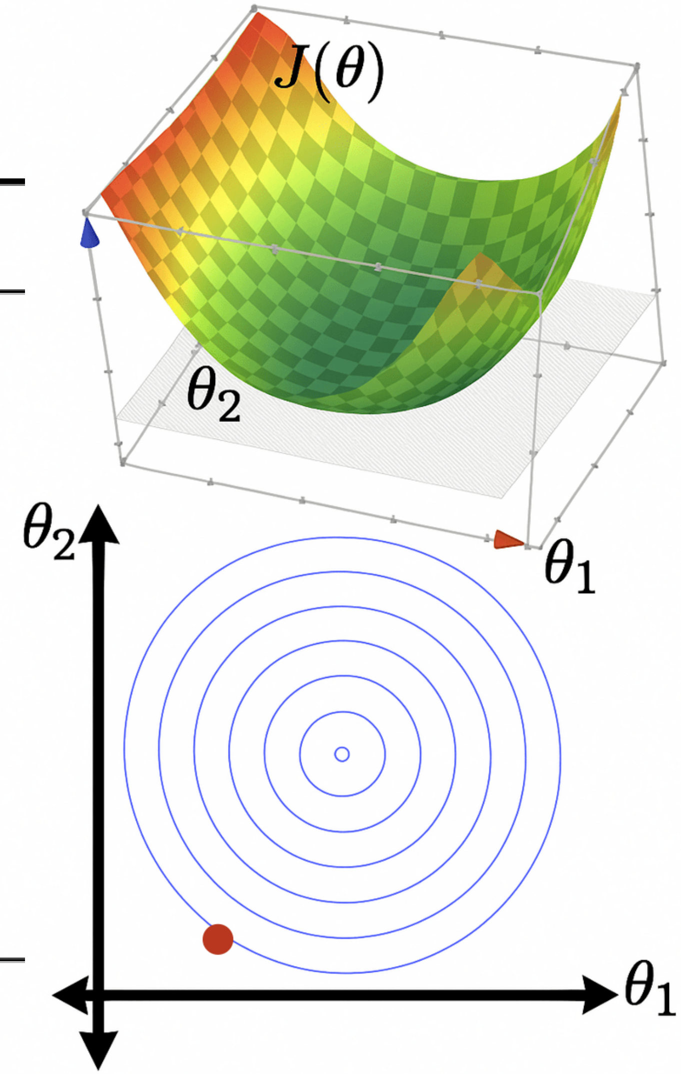
Algorithm 1 Gradient Descent($\theta_{\text{init}}, \eta, J, \epsilon$)

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$(\theta_{\text{init}}, \eta, J, \epsilon)$

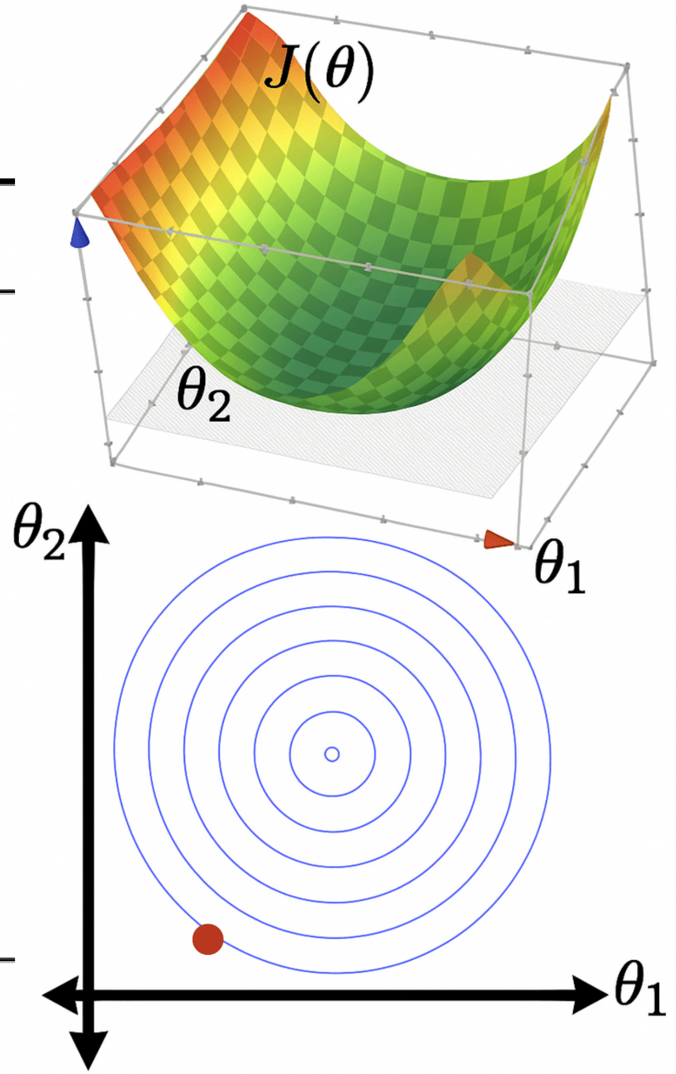
Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
    $t = t + 1$
    $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
return $\theta^{(t)}$



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

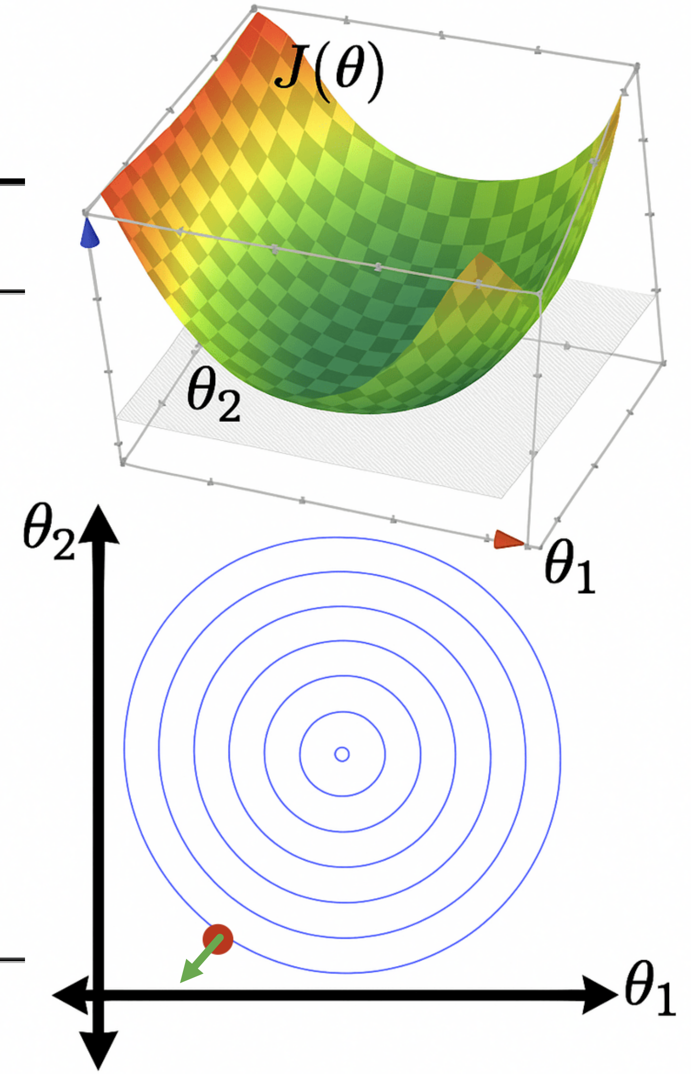
Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
    $t = t + 1$
    $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
return $\theta^{(t)}$

iteration counter



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

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$(\theta_{\text{init}}, \eta, J, \epsilon)$

Initialize $\theta^{(0)} = \theta_{\text{init}}$

Initialize $t = 0$

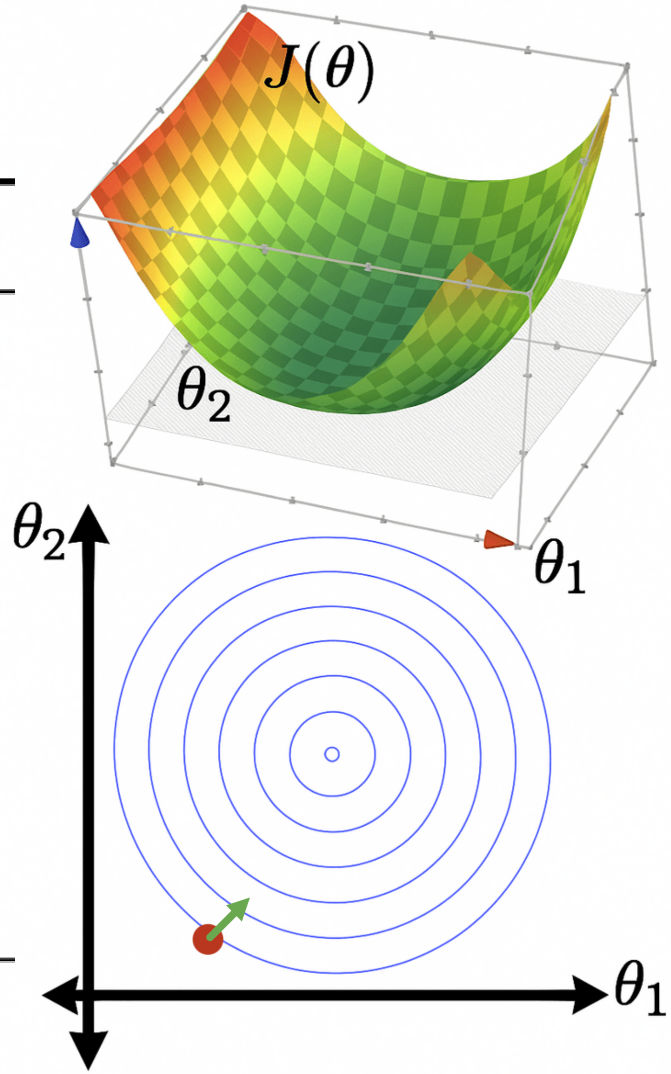
repeat

  $t = t + 1$

  $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$

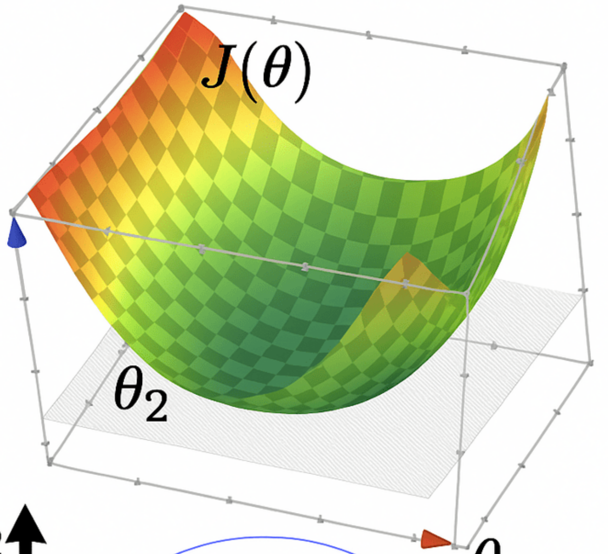
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$

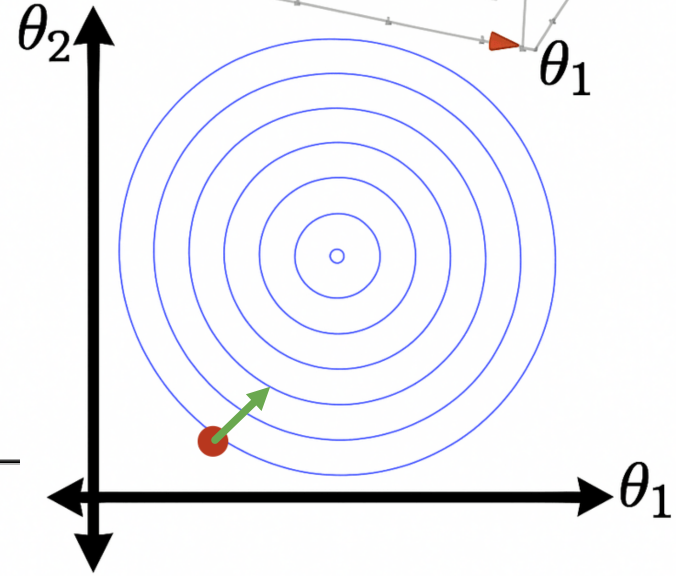
return $\theta^{(t)}$



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

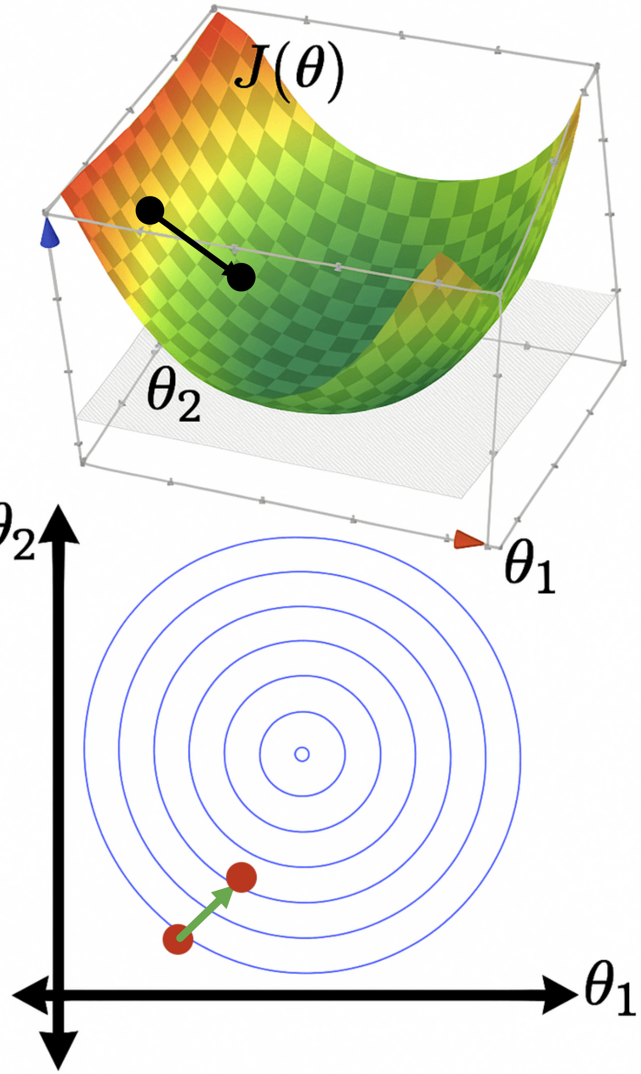
Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
    $t = t + 1$
    $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
return $\theta^{(t)}$





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

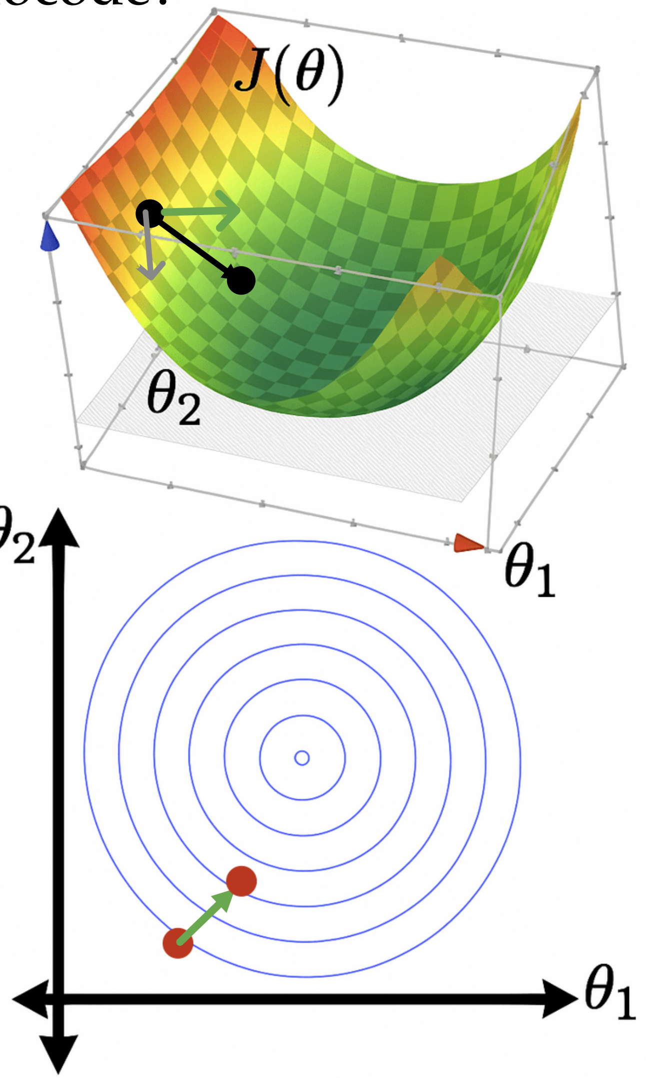
Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
    $t = t + 1$
    $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
return $\theta^{(t)}$



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

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

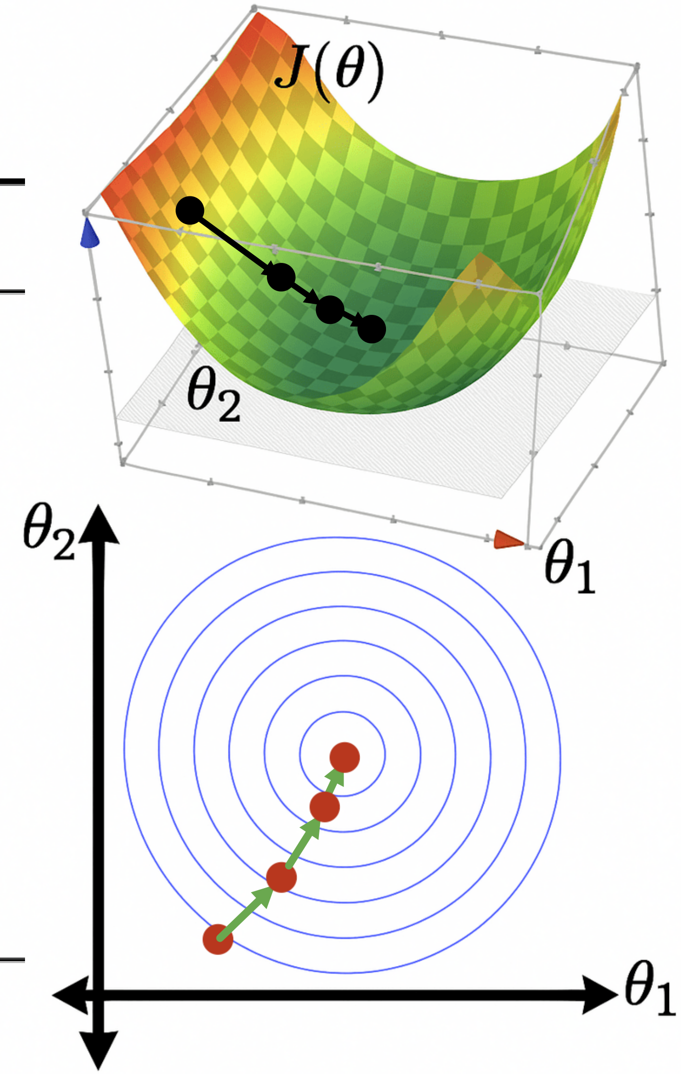
Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
    $t = t + 1$
    $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_\theta J(\theta^{(t-1)})$
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
return $\theta^{(t)}$



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

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

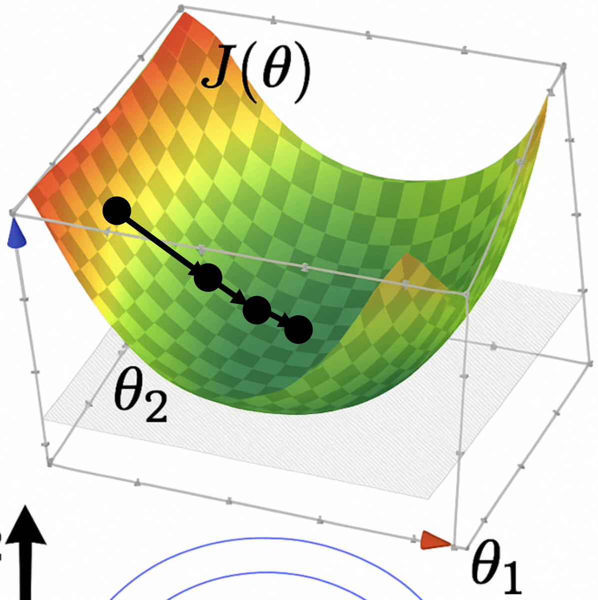
Initialize $\theta^{(0)} = \theta_{\text{init}}$
Initialize $t = 0$
repeat
    $t = t + 1$
    $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla_{\theta} J(\theta^{(t-1)})$
until $\left| J(\theta^{(t)}) - J(\theta^{(t-1)}) \right| < \epsilon$
return $\theta^{(t)}$

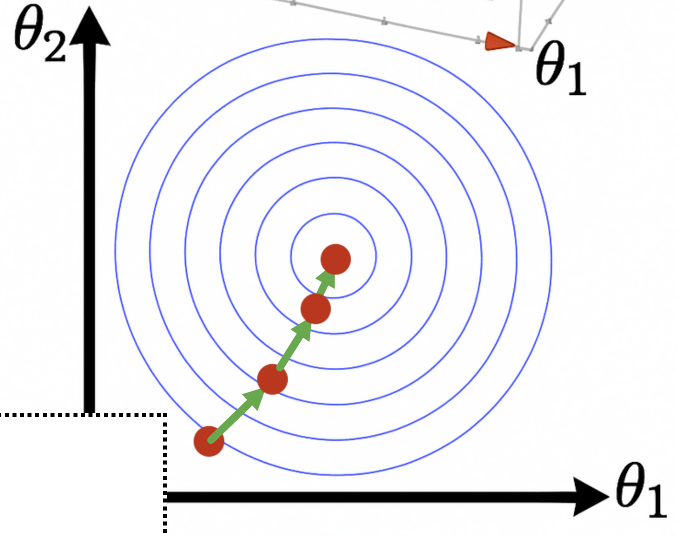


Algorithm 1 Gradient Descent$(\theta_{\text{init}}, \eta, J, \epsilon)$

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: $\|\nabla_\theta J(\theta^{(t-1)})\| < \epsilon$

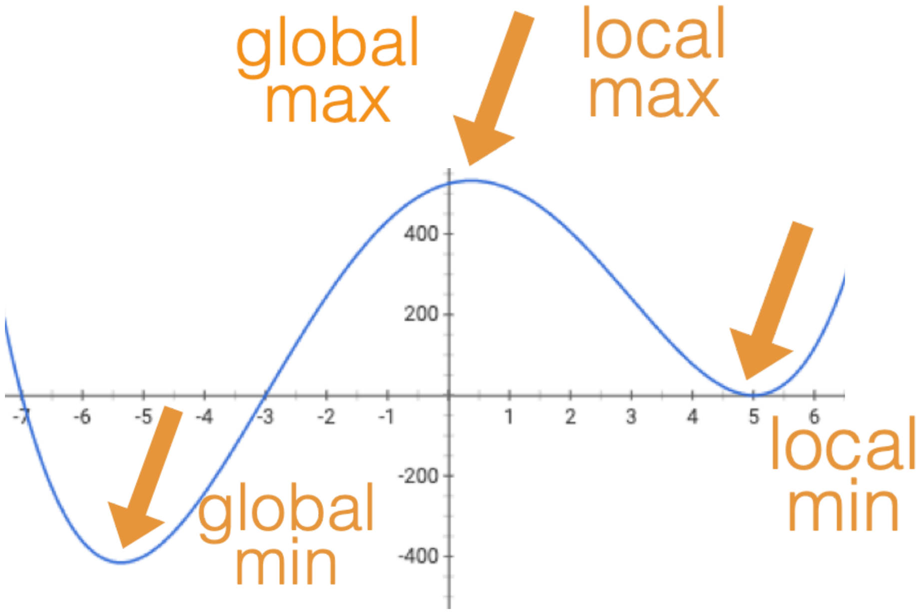
Outline

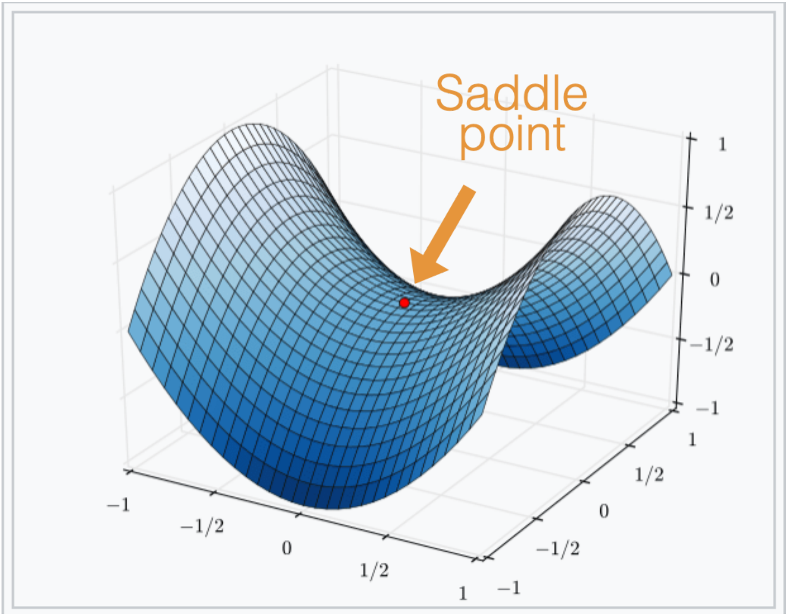
- Gradient descent (GD)
  - The gradient vector
  - GD algorithm
  - Gradient descent properties
- Stochastic gradient descent (SGD)

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

gradient descent can achieve this (to arbitrary precision)

$$\text{At a global minimizer} \quad \underset{\not\Leftarrow}{\Rightarrow} \quad \text{the gradient vector is zero}$$





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

$$\text{At a global minimizer} \quad \Longleftarrow \begin{cases} \text{the gradient vector is zero} \\ \text{the objective function is convex} \end{cases}$$

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

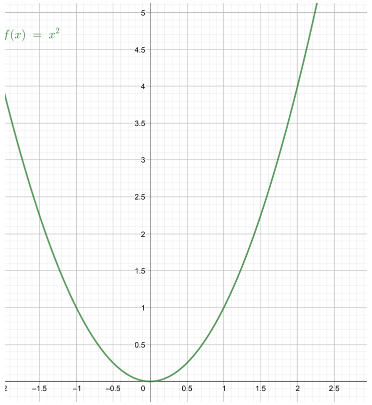
https://shenshen.mit.edu/demos/convex-obj.html?embed

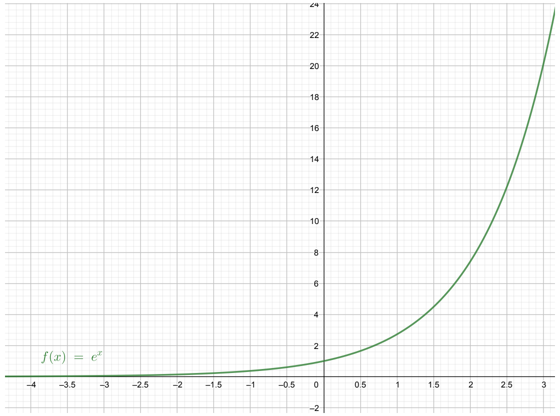
- J_{MSE} is always convex
- J_{ridge} with $\lambda > 0$ is always (strongly) convex

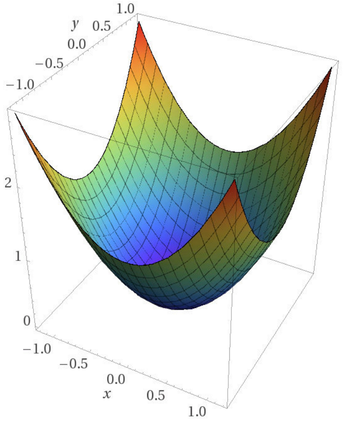
convexity is why we can claim the point whose gradient is zero is a global minimizer.

More examples

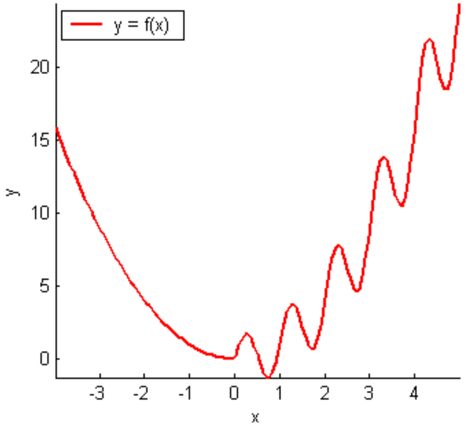
Convex functions

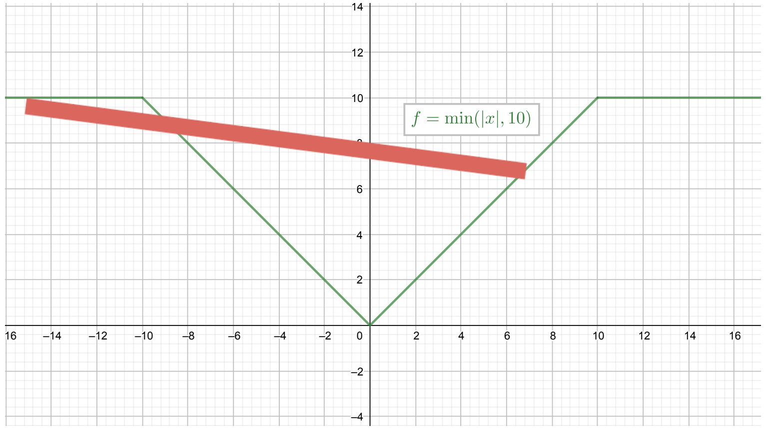


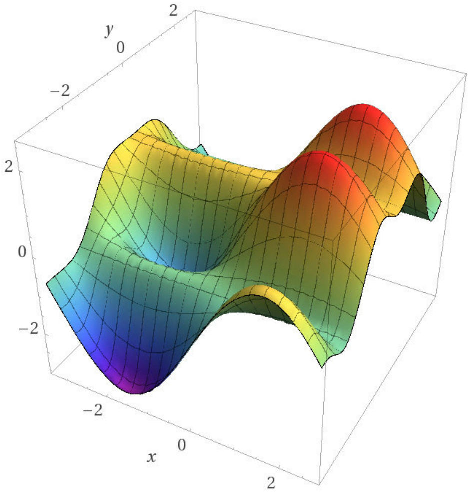




Non-convex functions







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

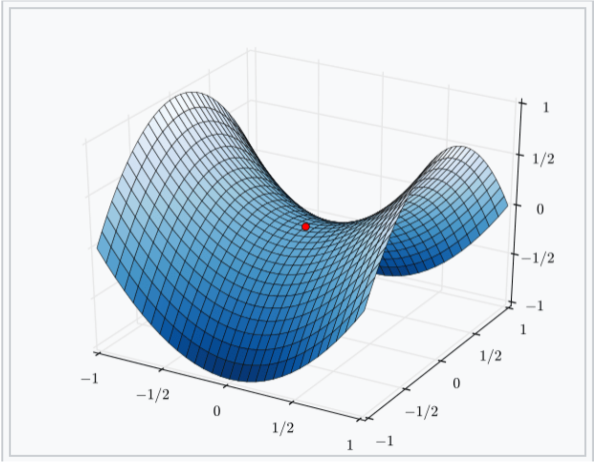
if violated, may not have gradient,

can't run gradient descent

Gradient Descent Performance

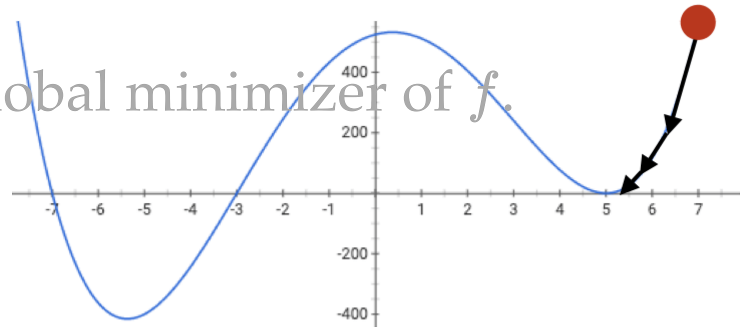
- Assumptions:
  - f is sufficiently "smooth"
  - f is convex
  - f has at least one global minimum
  - Run gradient descent for sufficient iterations
  - η is sufficiently small

if violated, may get stuck at a saddle point



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

or a local minimum

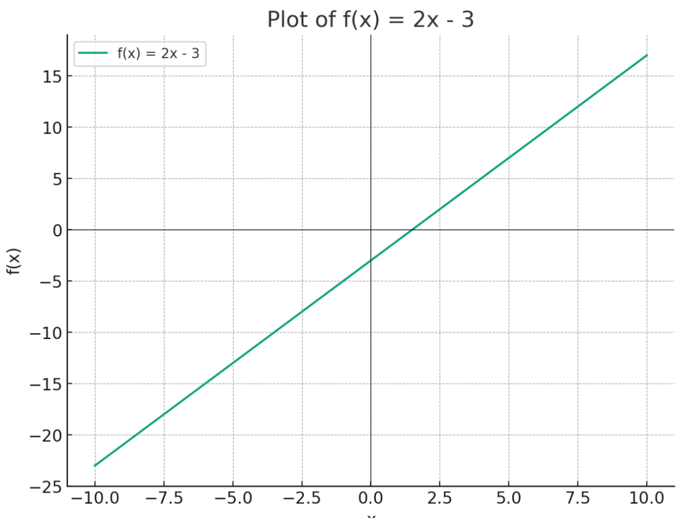


Gradient Descent Performance

- 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



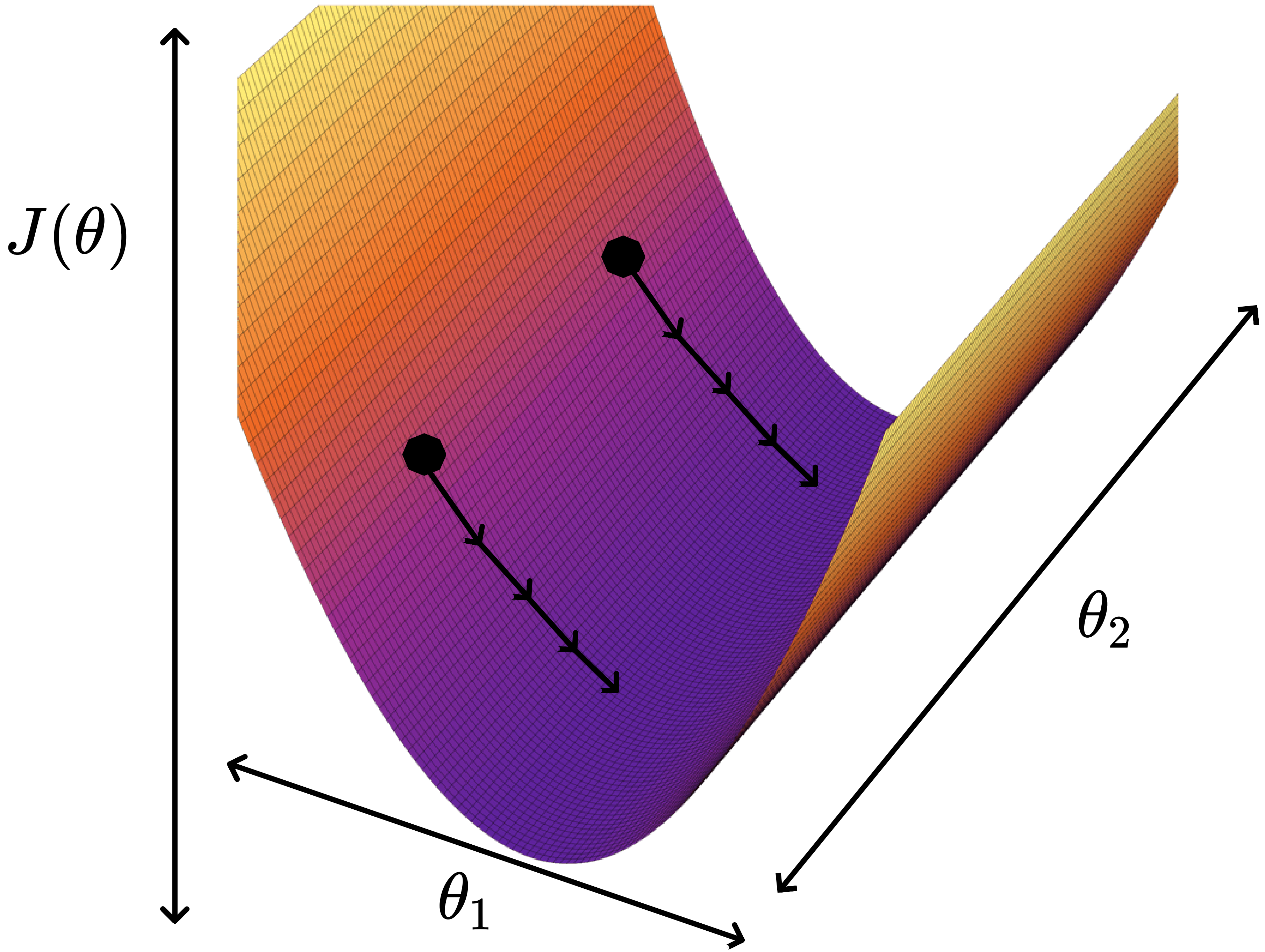
Gradient Descent Performance

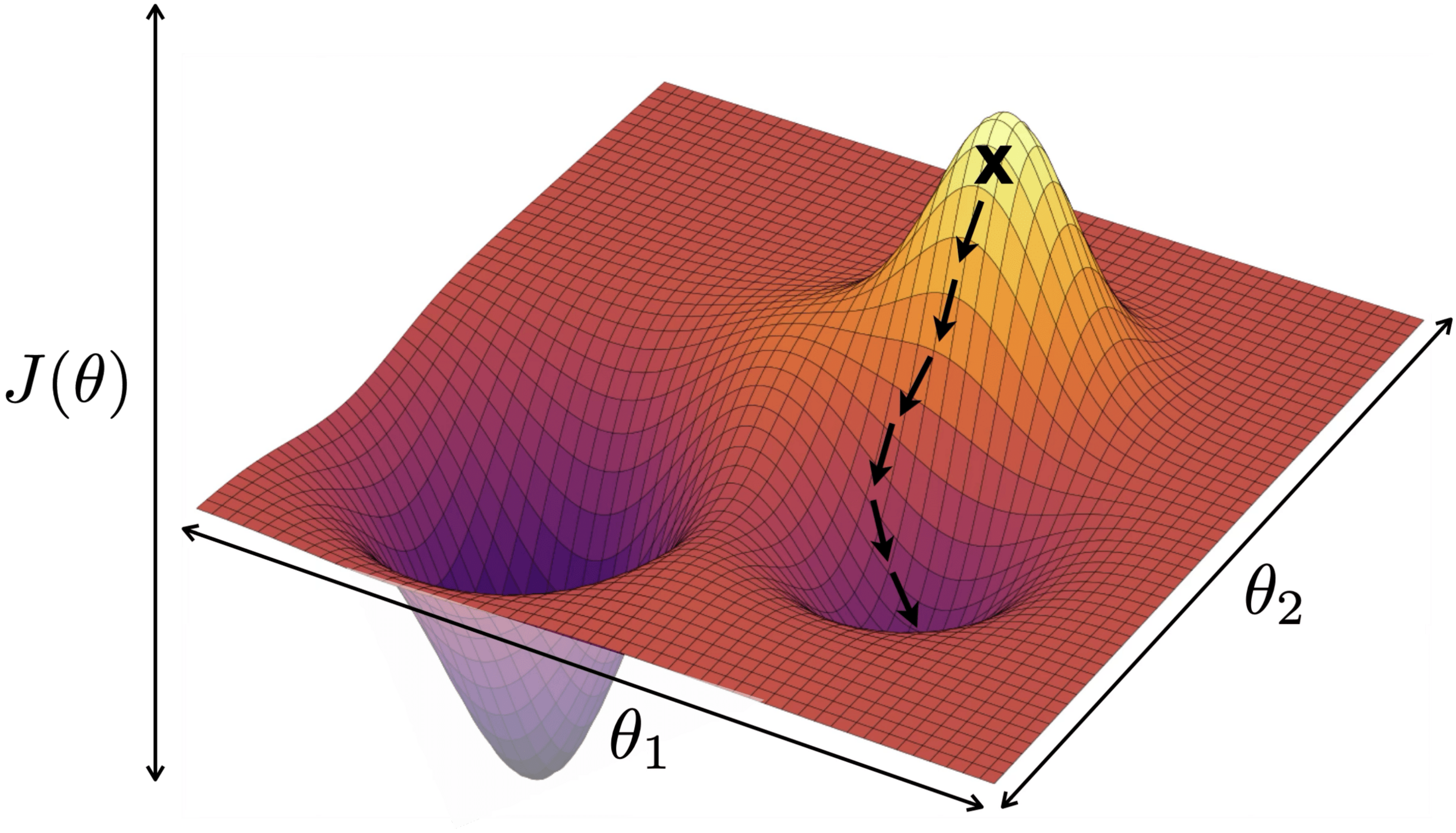
- Assumptions:
  - f is sufficiently "smooth"
  - f is convex
  - f has at least one global minimum
  - Run gradient descent for sufficient iterations
  - η is sufficiently small
- Conclusion:
  - Gradient descent converges arbitrarily close to a global minimizer of f.

if violated:

see demo on next slide, also lab/hw

https://shenshen.mit.edu/demos/gd/gd3d.html

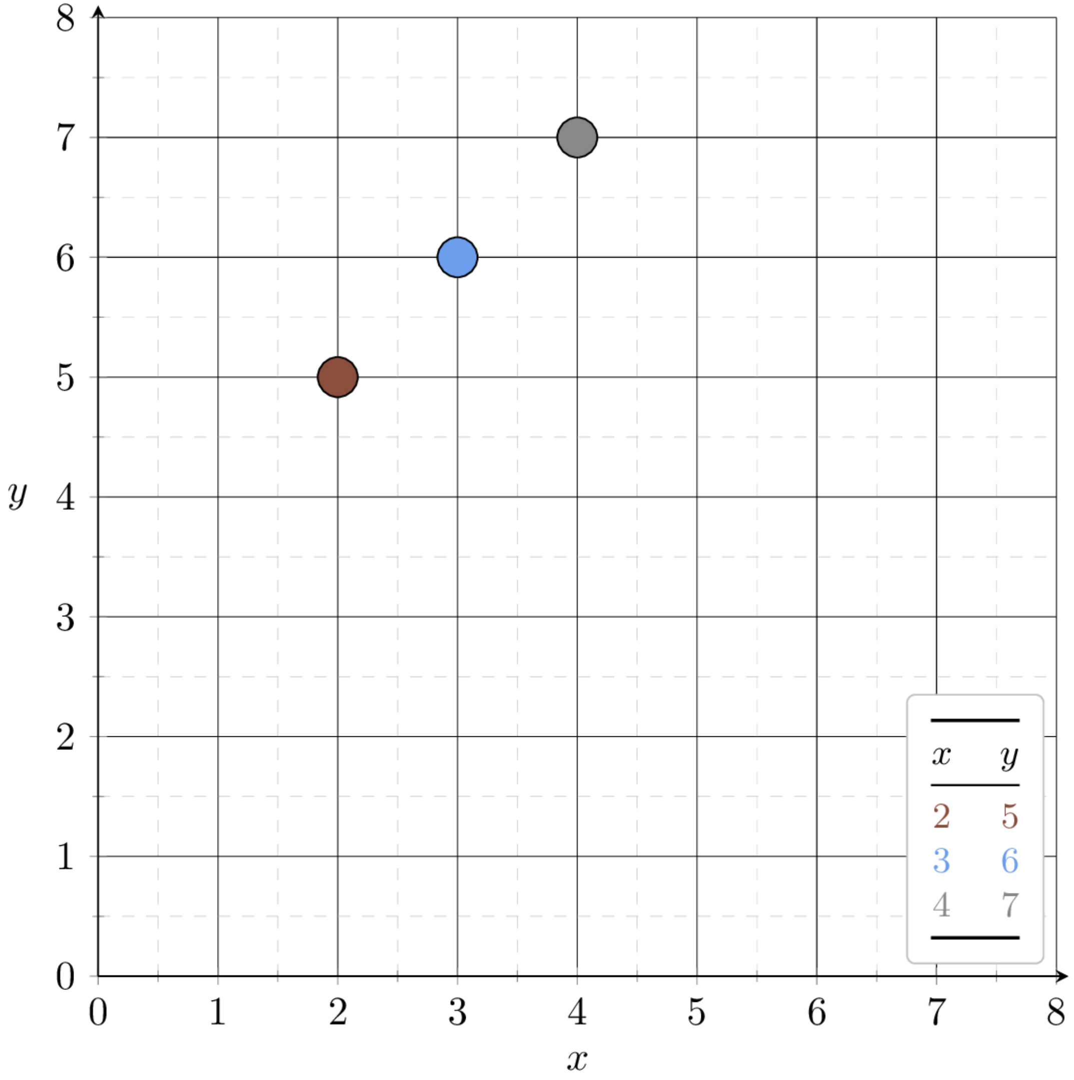

$J(\theta)$
θ_1
θ_2

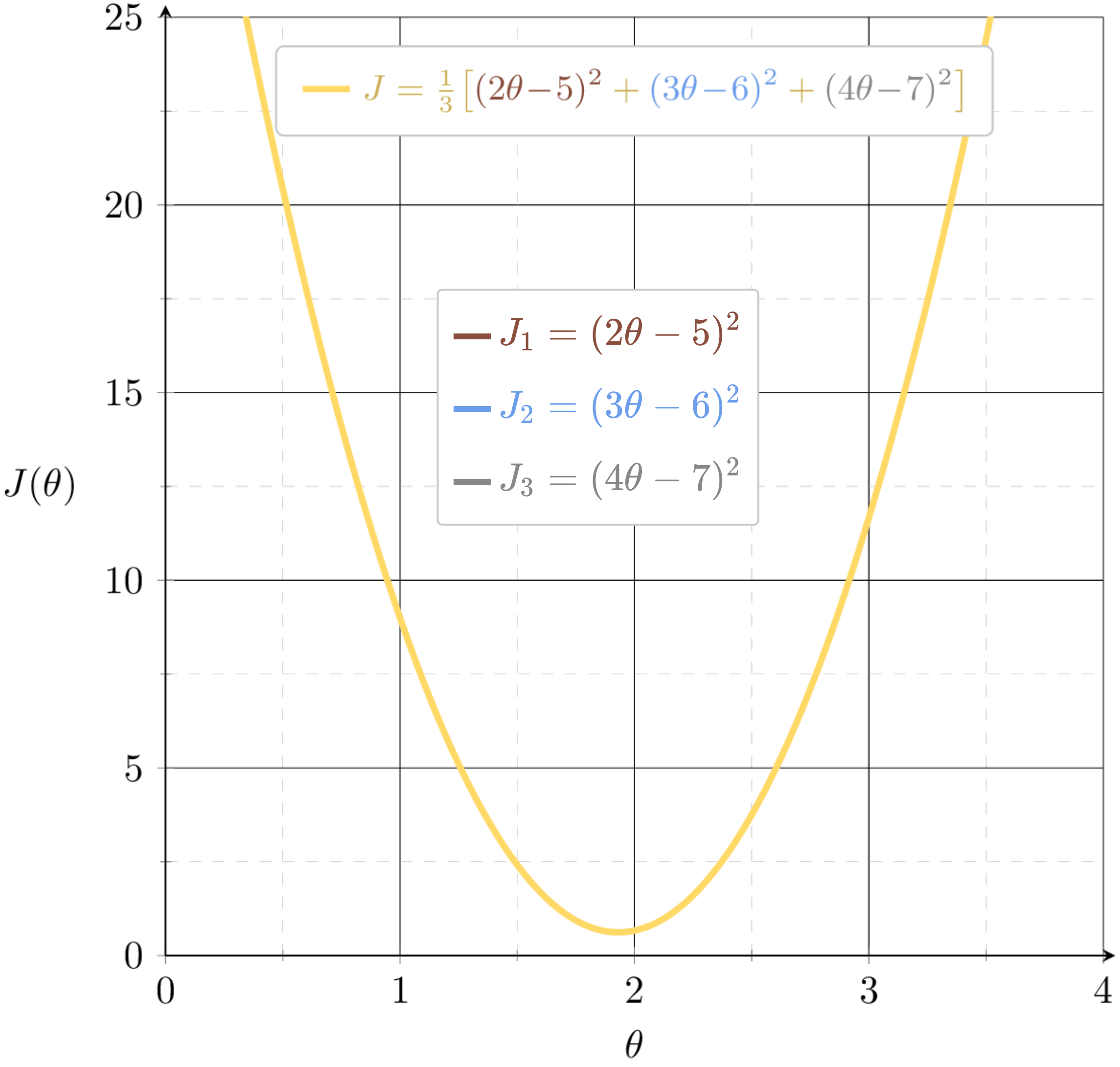

$J(\theta)$
x
θ_1
θ_2

Outline

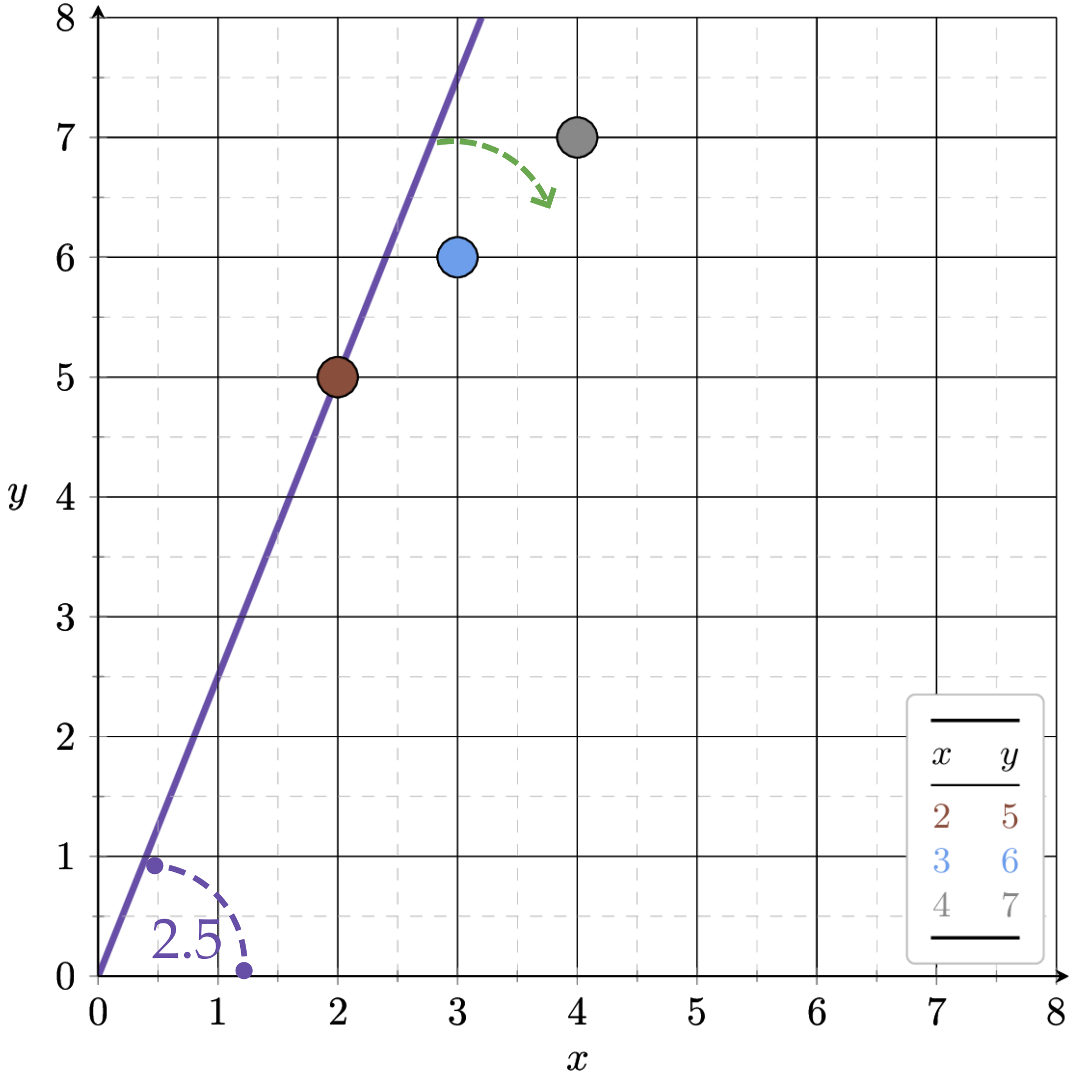
- Gradient descent (GD)
- Stochastic gradient descent (SGD)
  - SGD algorithm and setup
  - SGD vs. GD

Example 2: fit a line (no offset) to minimize MSE (3 data points)

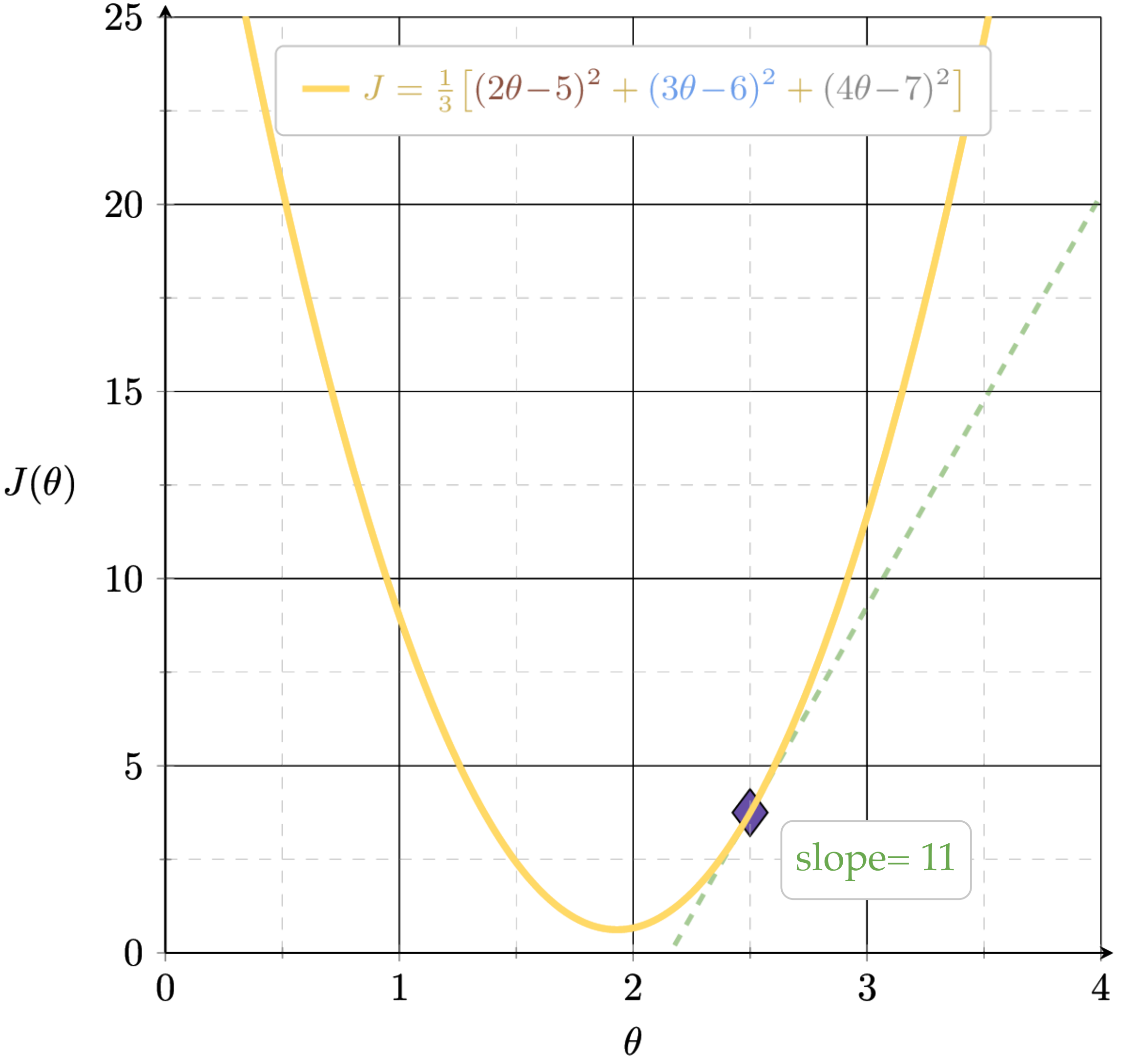




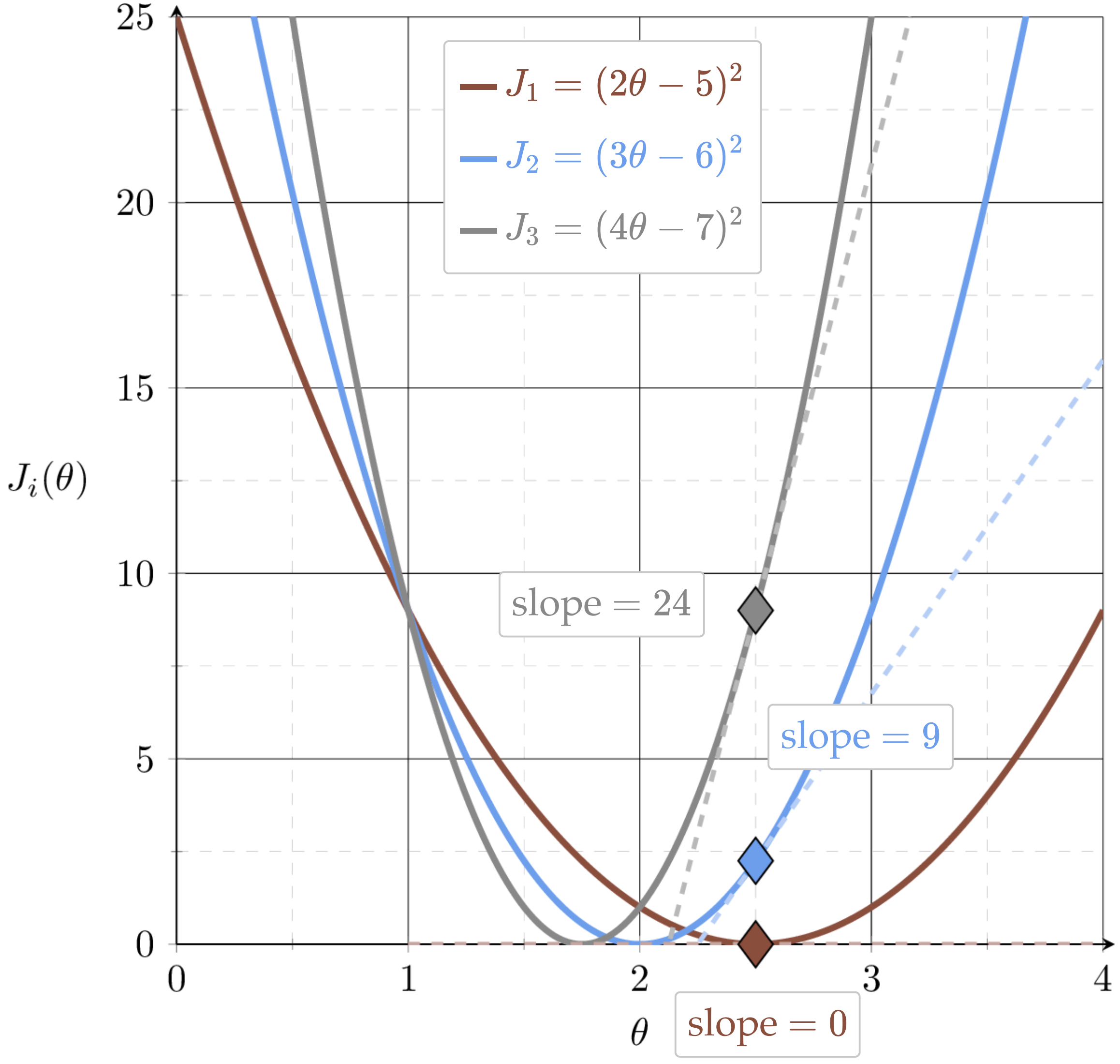
Suppose we fit $h = 2.5x$

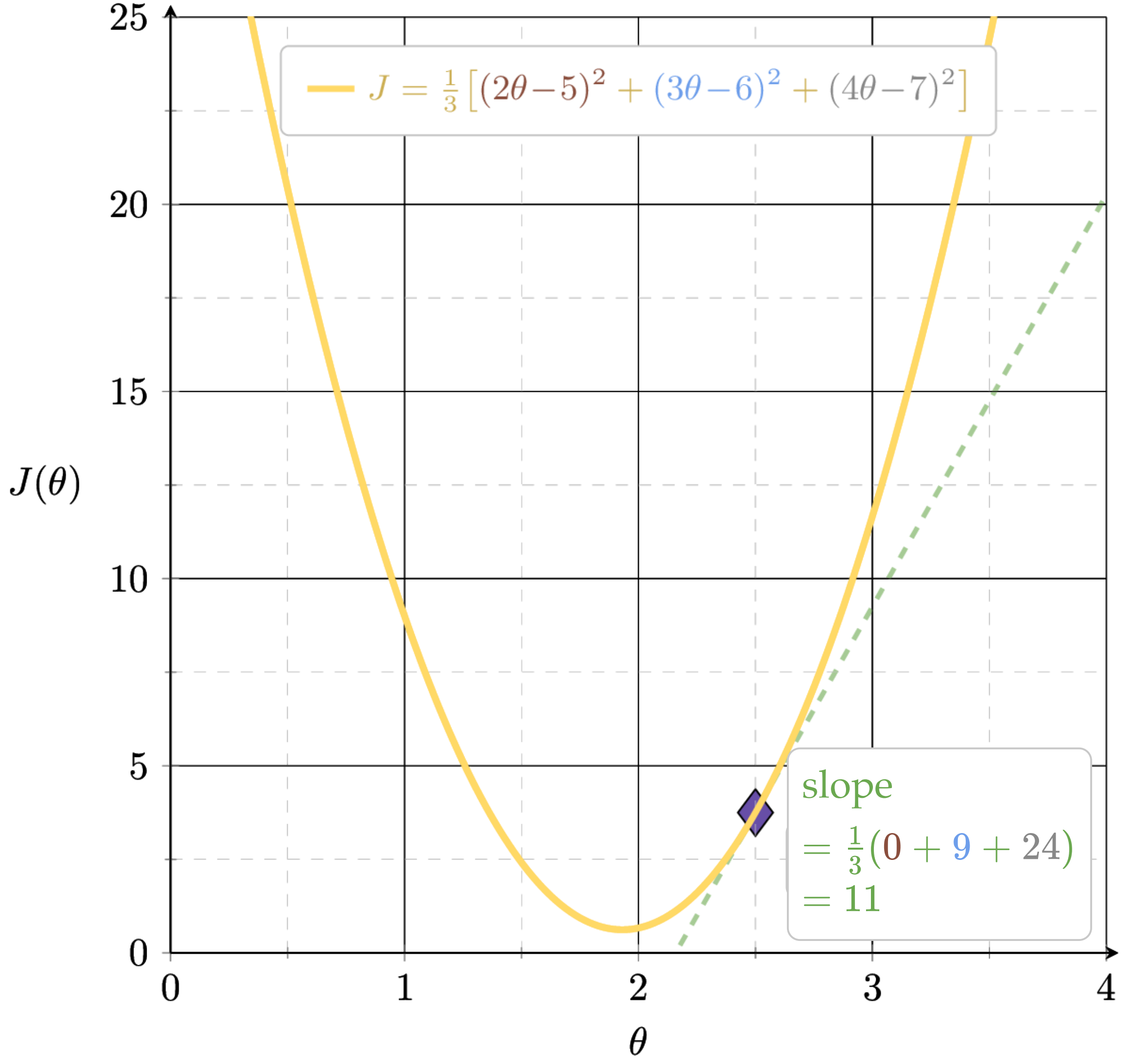


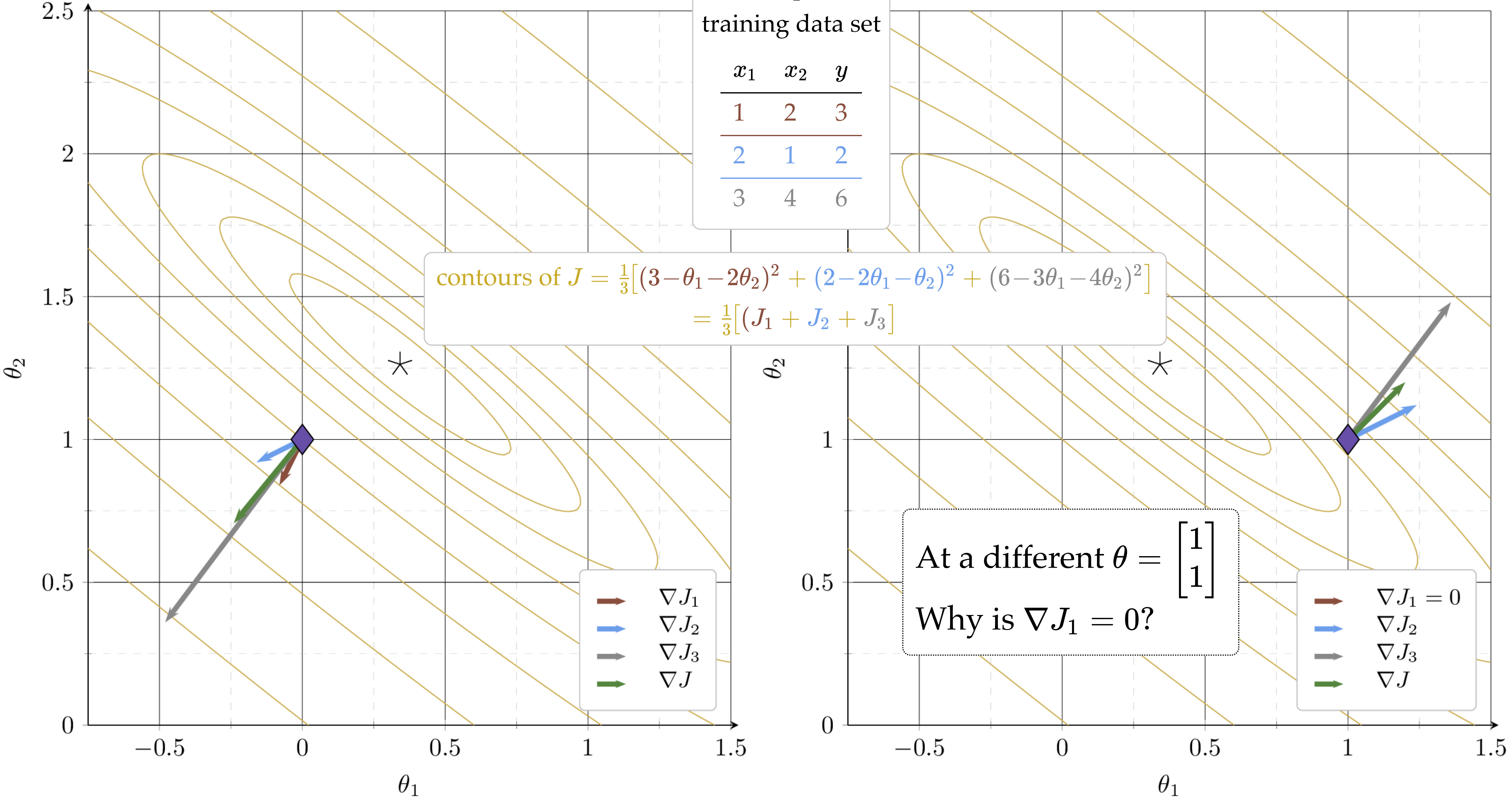
MSE could get better, by leveraging the gradient



Besides $J = \frac{1}{3}[(J_1 + J_2 + J_3]$ we also have $\nabla_\theta J = \frac{1}{3}\left[\nabla_\theta J_1 + \nabla_\theta J_2 + \nabla_\theta J_3\right]$







Example 3:
training data set

x_1	x_2	y
1	2	3
2	1	2
3	4	6

contours of $J = \frac{1}{3}\left[(3-\theta_1-2\theta_2)^2 + (2-2\theta_1-\theta_2)^2 + (6-3\theta_1-4\theta_2)^2\right]$
$= \frac{1}{3}\left[(J_1 + J_2 + J_3\right]$

At a different $\theta = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$

Why is $\nabla J_1 = 0$?

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:

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

👆

👋 (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

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

- 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 info from the i^{th} data point's loss

Costly in practice!

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

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($\theta_{\text{init}}, \eta, \{J_i\}_{i=1}^{n}, \epsilon$)

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 $\theta^{(t)}$

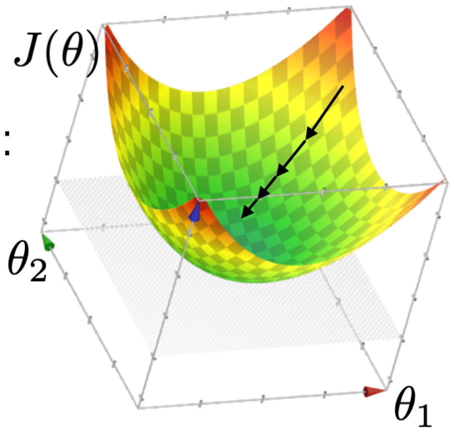
$$\nabla_\theta J(\theta) = \frac{1}{n} \sum_{i=1}^{n} \nabla_\theta J_i(\theta) \approx \nabla_\theta J_i(\theta)$$

Compared with GD, SGD:

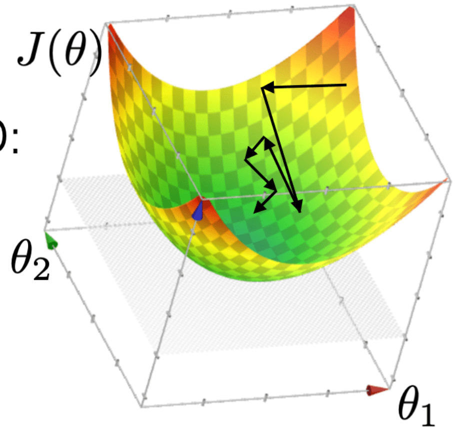
$$\nabla_\theta J(\theta) = \frac{1}{n}\sum_{i=1}^{n} \nabla_\theta J_i(\theta) \approx \nabla_\theta J_i(\theta)$$

is more efficient

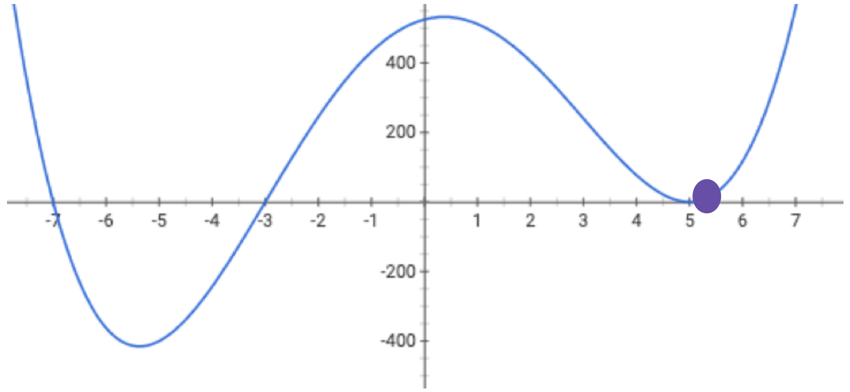
- GD:



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

[1] $\sum_{t=1}^{\infty} \eta(t) = \infty$ and $\sum_{t=1}^{\infty} \eta(t)^2 < \infty$, e.g., $\eta(t) = 1/t$

Algorithm 3 Mini-batch Gradient Descent($\theta_{\text{init}}, \eta, b, \{J_i\}_{i=1}^n, \epsilon$)

batch size

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** $\theta^{(t)}$

SGD	mini-batch GD	GD
$\nabla_\theta J_i(\theta)$	$\frac{1}{b}\sum_{i=1}^{b} \nabla_\theta J_i(\theta)$	$\frac{1}{n}\sum_{i=1}^{n} \nabla_\theta J_i(\theta) = \nabla_\theta J(\theta)$

🥰more accurate gradient, stronger theoretical guarantee

🥺more costly per parameter update step

Summary

- Most ML problems require optimization; closed-form solutions don't always exist or scale.
- Gradient descent iteratively updates θ in the direction of steepest descent of J.
- With a convex J and small enough η, GD converges to a global minimum.
- SGD approximates the full gradient with a single data point — faster but noisier.
- Mini-batch GD interpolates between GD and SGD.