DS-GA 1003: Machine Learning

Lecture 2: Optimization and Gradient Descent

Slides adapted from material from David Rosenberg.

Logistics & Announcements

PS 1 due date. Next Tuesday, February 3rd 11:59 PM.

Need help? Full office hours schedule this week; Ed Discussion for questions.

Feedback? Two channels: Anonymous Feedback Form and Lab Attendance Form.

5 minute break roughly halfway.

Math Review Videos. From feedback on Lab Attendance Forms and posted on Course Content page. First set on Bayes Hypothesis derivation and conditional expectations.

Lecture Recordings. Can be found on Brightspace => Zoom.

Outline

Statistical Learning Setup

Formalization of Prediction Problem

1. Observe an input $x \in \mathcal{X}$.
2. Predict an action $a \in \mathcal{A}$.
3. Observe the true outcome $y \in \mathcal{Y}$.
4. Evaluate the actions in relation to the outcome.

$\mathcal{X}$ is the input space (e.g. $\mathbb{R}^d$, pixels, words).

$\mathcal{Y}$ is the output space (e.g. $\{0,1\}$ or $\mathbb{R}$).

$\mathcal{A}$ is the action space (e.g. prediction of y, some decision).

$h : \mathcal{X} \to \mathcal{A}$ is a hypothesis to generate action $h(x)$.

Evaluate h with loss function $\ell : \mathcal{A} \times \mathcal{Y} \to \mathbb{R}$.

$\ell(h(x), y)$ evaluates h on (x, y).

$R(h) = \mathbb{E}_{(x,y)\sim P_{\mathcal{X}\times\mathcal{Y}}}[\ell(h(x), y)]$ is risk of h.

The Main Cast

Summary of the Problem

Examples from input space $\mathcal{X}$ and output space $\mathcal{Y}$; unknown distribution $P_{\mathcal{X}\times\mathcal{Y}}$ over $\mathcal{X} \times \mathcal{Y}$.

Action space $\mathcal{A}$ as the output (often, a *prediction*) of learned hypothesis/predictor.

We evaluate actions with a loss function $\ell : \mathcal{A} \times \mathcal{Y} \to \mathbb{R}$.

Goal: Find a hypothesis $h : \mathcal{X} \to \mathcal{A}$ to minimize the risk $R(h) := \mathbb{E}[\ell(h(x), y)]$.

We can approximate risk with the empirical risk over sample $D_n = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$:

$$\hat{R}_n(h) := \frac{1}{n}\sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)}).$$

The Main Cast

Summary of the Problem

Goal: Find a hypothesis $h : \mathcal{X} \to \mathcal{A}$ to minimize the risk $R(h) := \mathbb{E}[\ell(h(x), y)]$.

We can approximate risk with the empirical risk over sample $D_n = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$.

Choose a hypothesis class $\mathcal{H}$ and find the empirical risk minimizer $\hat{h}_n \in \mathcal{H}$:

$$\hat{h}_n \in \operatorname*{argmin}_{h \in \mathcal{H}} \underbrace{\frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)})}_{\hat{R}_n(h)}$$

Or find $\tilde{h}_n$ that approximates $\hat{h}_n$ well.

The Main Cast

Summary of the Problem

<u>Goal:</u> Find a hypothesis $h : \mathcal{X} \to \mathcal{A}$ to minimize the risk $R(h) := \mathbb{E}[\ell(h(x), y)]$.

Overall quality (excess risk) of our produced $\tilde{h}_n$:

$$R(\tilde{h}_n) - R(h^*) = \underbrace{R(\tilde{h}_n) - R(\hat{h}_n)}_{\text{opt. error}} + \underbrace{R(\hat{h}_n) - R(h^*_{\mathcal{H}})}_{\text{est. error}} + \underbrace{R(h^*_{\mathcal{H}}) - R(h^*)}_{\text{approx. error}}$$

Choose $\mathcal{H}$ that balances approximation error and estimation error.

With more data, estimation error typically decreases, can use bigger $\mathcal{H}$.

Produce $\tilde{h}_n$ via an algorithm that (approximately and efficiently) minimizes empirical error.

Excess Risk

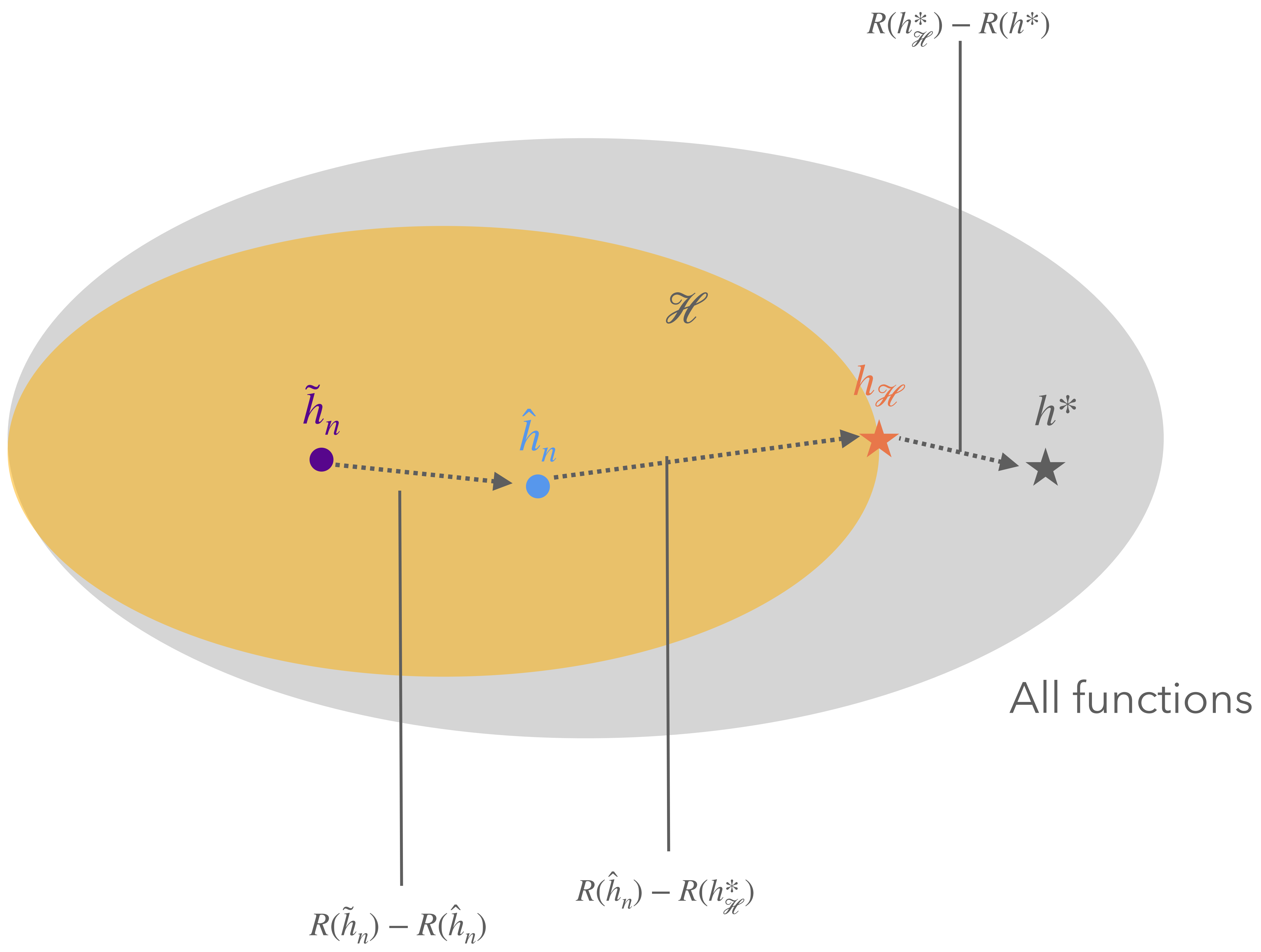
Full Decomposition

We receive $\tilde{h}_n$ from an algorithm.

Excess risk of $\tilde{h}_n$:

$$R(\tilde{h}_n) - R(h^*) =$$

$$\underbrace{R(\tilde{h}_n) - R(\hat{h}_n)}_{\text{opt. error}} + \underbrace{R(\hat{h}_n) - R(h^*_{\mathscr{H}})}_{\text{est. error}} + \underbrace{R(h^*_{\mathscr{H}}) - R(h^*)}_{\text{approx. error}}$$



Supervised Learning

Excess Risk Formalization

1. Collect training dataset, a collection of labeled input-output pairs.

2. Decide on the template of the hypothesis mapping that will map inputs to outputs. — Representation

3. A learning algorithm takes the labeled training data as input and outputs a hypothesis. — Optimization

4. The hypothesis predicts on new, unseen data which we hope it does well on, under a notion of loss. — Generalization

We receive $\tilde{h}_n$ from an algorithm.

Excess risk of $\tilde{h}_n$:

$$R(\tilde{h}_n) - R(h^*) =$$

$$\underbrace{R(\tilde{h}_n) - R(\hat{h}_n)}_{\text{opt. error}} + \underbrace{R(\hat{h}_n) - R(h^*_{\mathscr{H}})}_{\text{est. error}} + \underbrace{R(h^*_{\mathscr{H}}) - R(h^*)}_{\text{approx. error}}$$

Optimization — Generalization — Representation

Supervised Learning

Excess Risk Formalization

1. Collect training dataset, a collection of labeled input-output pairs.

2. Decide on the template of the hypothesis mapping that will map inputs to outputs. — Representation

3. A learning algorithm takes the labeled training data as input and outputs a hypothesis. — Optimization

4. The hypothesis predicts on new, unseen data which we hope it does well on, under a notion of loss. — Generalization

We receive $\tilde{h}_n$ from an algorithm.

Excess risk of $\tilde{h}_n$:

$$R(\tilde{h}_n) - R(h^*) =$$

$$\underbrace{R(\tilde{h}_n) - R(\hat{h}_n)}_{\text{opt. error}} + \underbrace{R(\hat{h}_n) - R(h^*_{\mathscr{H}})}_{\text{est. error}} + \underbrace{R(h^*_{\mathscr{H}}) - R(h^*)}_{\text{approx. error}}$$

Optimization — Generalization — Representation

How do we get a good approximation to the ERM?

Learning as Optimization

Recurring Theme

$$\underbrace{R(\tilde{h}_n) - R(\hat{h}_n)}_{\text{opt. error}} + \underbrace{R(\hat{h}_n) - R(h^*_{\mathscr{H}})}_{\text{est. error}} + \underbrace{R(h^*_{\mathscr{H}}) - R(h^*)}_{\text{approx. error}}$$

<u>Estimation error:</u> As $n \to \infty$, typically $R(\hat{h}_n) - R(h^*_{\mathscr{H}}) \to 0$.

<u>Approximation error:</u> Controlled by choosing a good hypothesis class $\mathscr{H}$.

<u>Optimization error:</u> Can we make this small using an efficient algorithm?

Can we solve the optimization problem: $\min_{h \in \mathscr{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)})$?

Outline

Linear (Least Squares) Regression

Running Example

Input space: $\mathcal{X} = \mathbb{R}^d$

Hypothesis class is parametrized by $w \in \mathbb{R}^d$

Output space: $\mathcal{Y} = \mathbb{R}$ Action space: $\mathcal{A} = \mathcal{Y} = \mathbb{R}$

Loss Function: $\ell(\hat{y}, y) = (\hat{y} - y)^2$

Hypothesis Class: $\mathcal{H} = \{h : \mathbb{R}^d \to \mathbb{R} : h(x) = w^\top x, w \in \mathbb{R}^d\}$

Given dataset $D_n := \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ we want to minimize the empirical risk:

$\hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} (w^\top x^{(i)} - y^{(i)})^2$ or $\hat{R}_n(w) = \frac{1}{n} \|Xw - y\|^2$ with $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$.

Objective in scalar form

Objective in matrix-vector form

Linear Regression

Examples

Predicting stock prices.

Inputs: metrics about company (earnings reports, historical prices, etc.). **Output:** stock price.

Predicting the weather.

Inputs: weather data, meteorological measurements. **Output:** tomorrow's temperature.

Predicting sports performance.

Inputs: historical performance (batting averages, free throw percentages. **Output:** player score.

⋮

Linear Regression

Matrix-vector Form

$$\hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n}(w^\top x^{(i)} - y^{(i)})^2$$

$X = \begin{bmatrix} \leftarrow & x^{(1)} & \rightarrow \\ & \vdots & \\ \leftarrow & x^{(n)} & \rightarrow \end{bmatrix} \in \mathbb{R}^{n \times d}$ is the design matrix and $y = \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} \in \mathbb{R}^n$ is the output vector.

Linear Regression

Matrix-vector Form

$$\hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n}(w^\top x^{(i)} - y^{(i)})^2$$

$$Xw - y = \begin{bmatrix} \leftarrow & x^{(1)} & \rightarrow \\ & \vdots & \\ \leftarrow & x^{(n)} & \rightarrow \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix} - \begin{bmatrix} y^{(1)} \\ \vdots \\ y^{(n)} \end{bmatrix} = \begin{bmatrix} w^\top x^{(1)} - y^{(1)} \\ \vdots \\ w^\top x^{(n)} - y^{(n)} \end{bmatrix}$$

Therefore, $\|Xw - y\|^2 = \sum_{i=1}^{n}(w^\top x^{(i)} - y^{(i)})^2$.

So we can always rewrite $\hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n}(w^\top x^{(i)} - y^{(i)})^2 = \frac{1}{n}\|Xw - y\|^2$.

Linear Regression

A note on intercepts

For each $i \in [n]$, what if we want to predict: $w_1 x_1^{(i)} + \ldots + w_d x_d^{(i)} + w_0$?

<u>Solution:</u> We add a "dummy" 1 to each example:

$$\tilde{x}^{(i)} = \begin{pmatrix} x_1^{(i)} & x_2^{(i)} & \ldots & x_d^{(i)} & 1 \end{pmatrix}.$$

Solve problem with $\mathscr{X} = \mathbb{R}^{d+1}$, $\mathscr{H} = \{h : \mathbb{R}^{d+1} \to \mathbb{R} : h(x) = w^\top x, w \in \mathbb{R}^{d+1}\}$ and modified dataset $D_n := \{(\tilde{x}^{(1)}, y^{(1)}), \ldots, (\tilde{x}^{(n)}, y^{(n)})\}$.

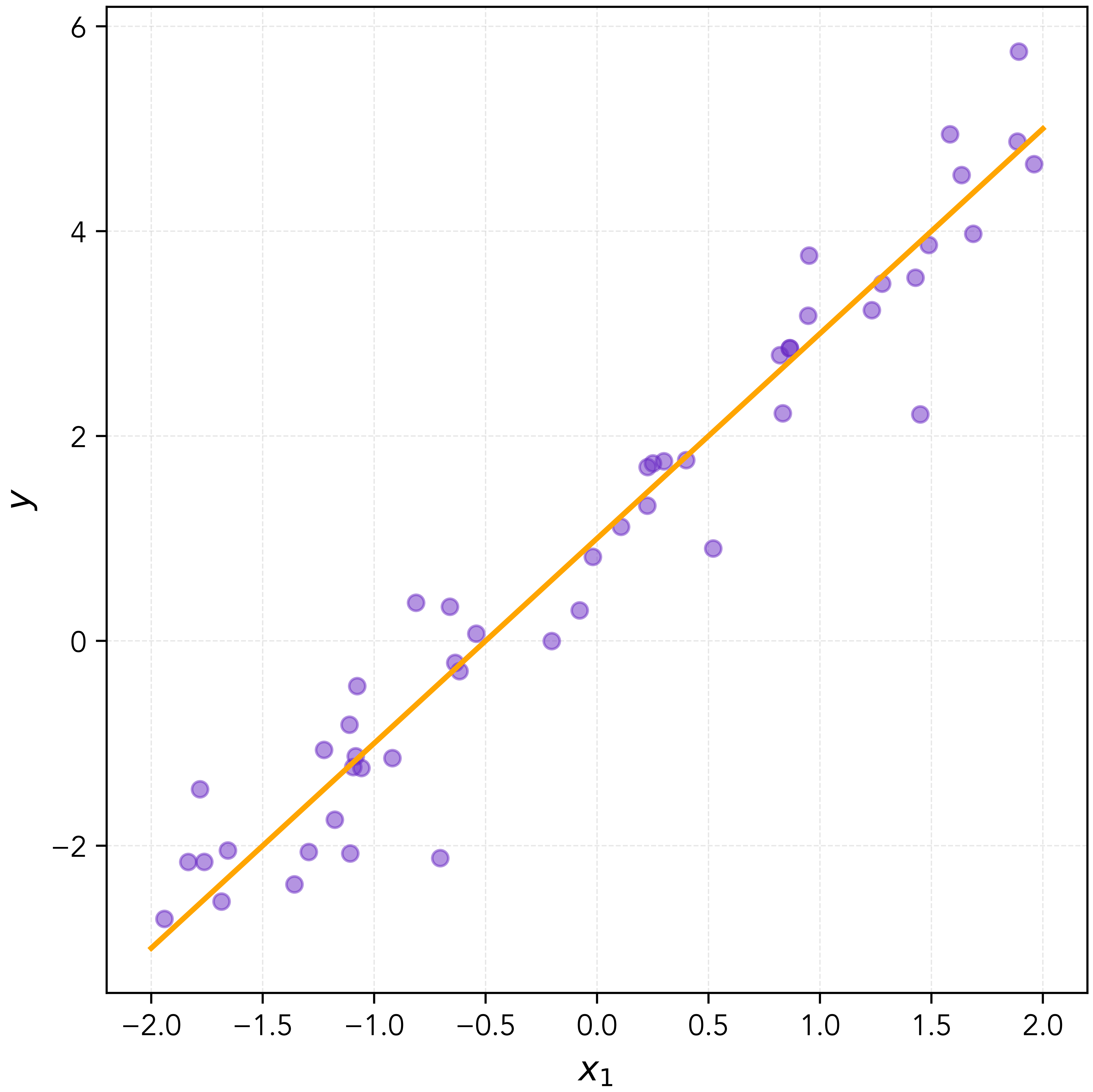
Minimizer will be $\tilde{w} = (w_1 \quad w_2 \quad \cdots \quad w_d \quad w_0) \in \mathbb{R}^{d+1}$, so w_0 is your intercept term.

We can always do this without loss of generality (so focus on the 0 intercept case).

Linear Regression

Example: $d = 1$

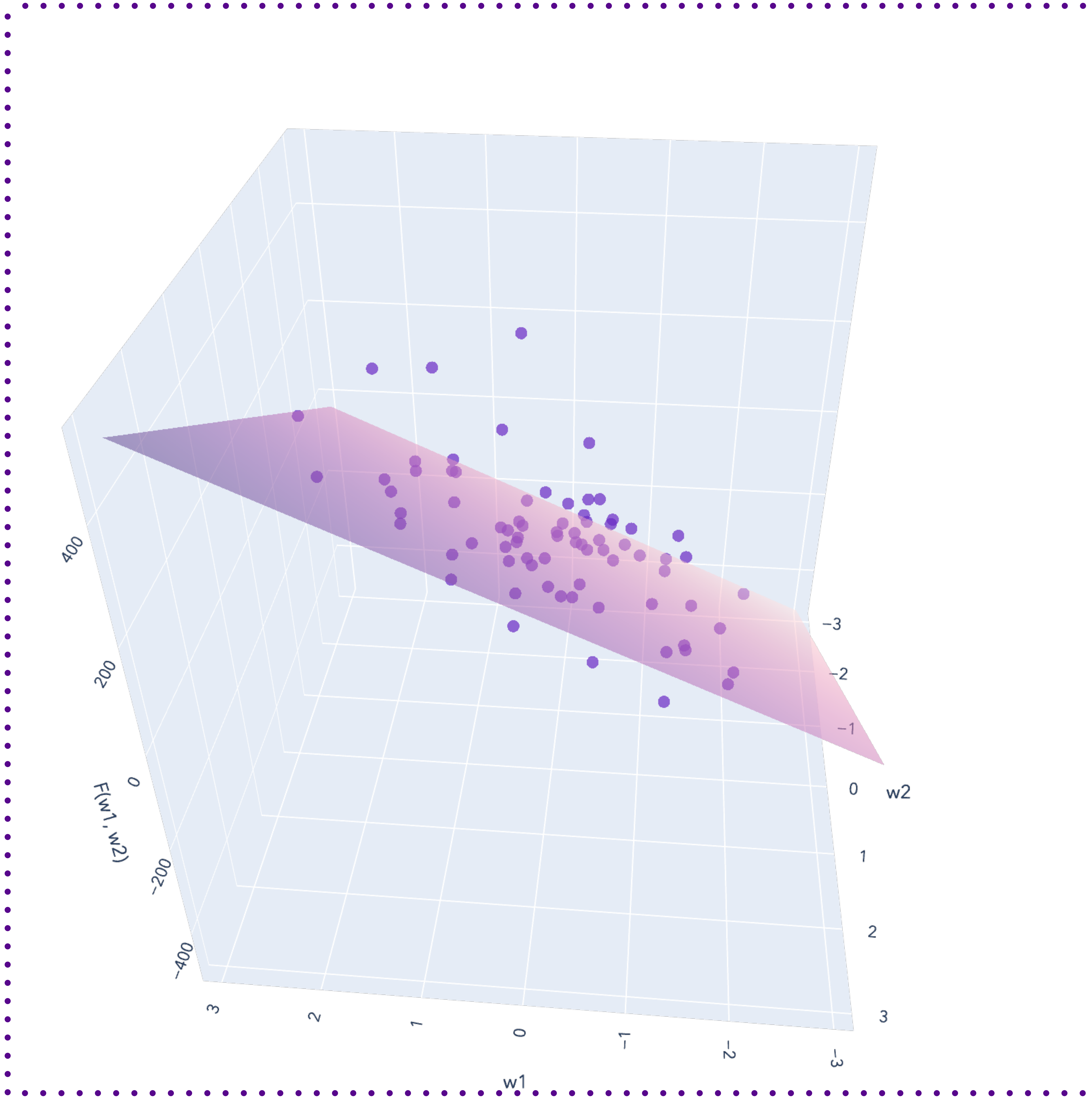
$$X = \begin{bmatrix} \vdots \\ -0.58 \\ 1.36 \\ 1.30 \\ -0.86 \\ \vdots \end{bmatrix} y = \begin{bmatrix} \vdots \\ -0.30 \\ 3.16 \\ 3.29 \\ -1.75 \\ \vdots \end{bmatrix}$$



Linear Regression

Example: $d = 2$

$$X = \begin{bmatrix} \vdots & \vdots \\ 0.51 & -0.53 \\ -0.56 & -1.72 \\ -0.57 & -0.99 \\ 1.54 & 0.36 \\ \vdots & \vdots \end{bmatrix} y = \begin{bmatrix} \vdots \\ -85.35 \\ -121.2 \\ -46.14 \\ 154.72 \\ \vdots \end{bmatrix}$$



Linear Regression

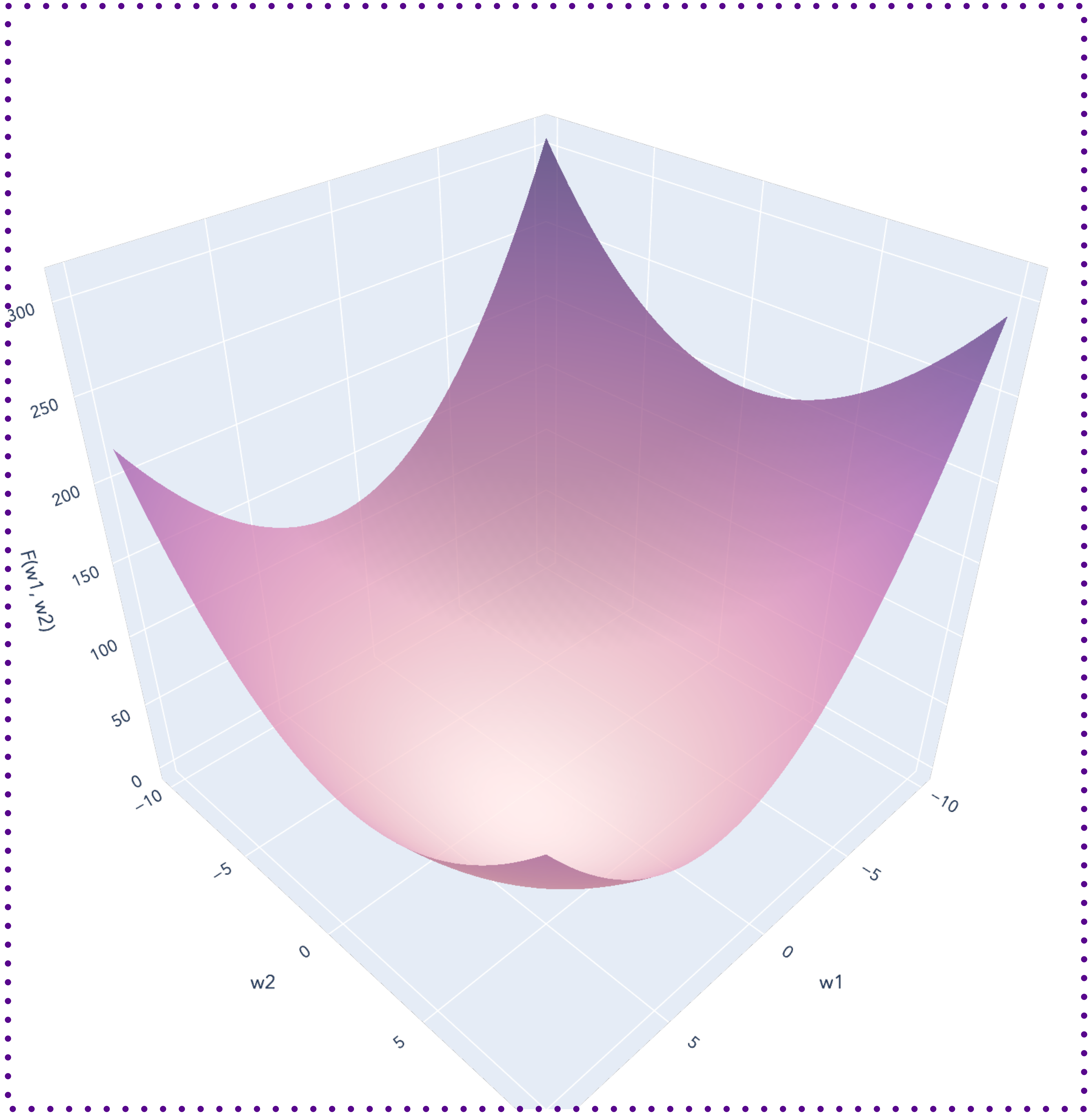
Example: Loss Surface

For a fixed dataset $X \in \mathbb{R}^{n \times 2}$ and $y \in \mathbb{R}^n$, the loss of any $w \in \mathbb{R}^2$ is:

$$\hat{R}_n : \mathbb{R}^2 \to \mathbb{R}$$

$$\hat{R}_n(w) = \|Xw - y\|^2.$$

We can visualize it with a loss surface.



Linear Regression

Example: Loss Surface

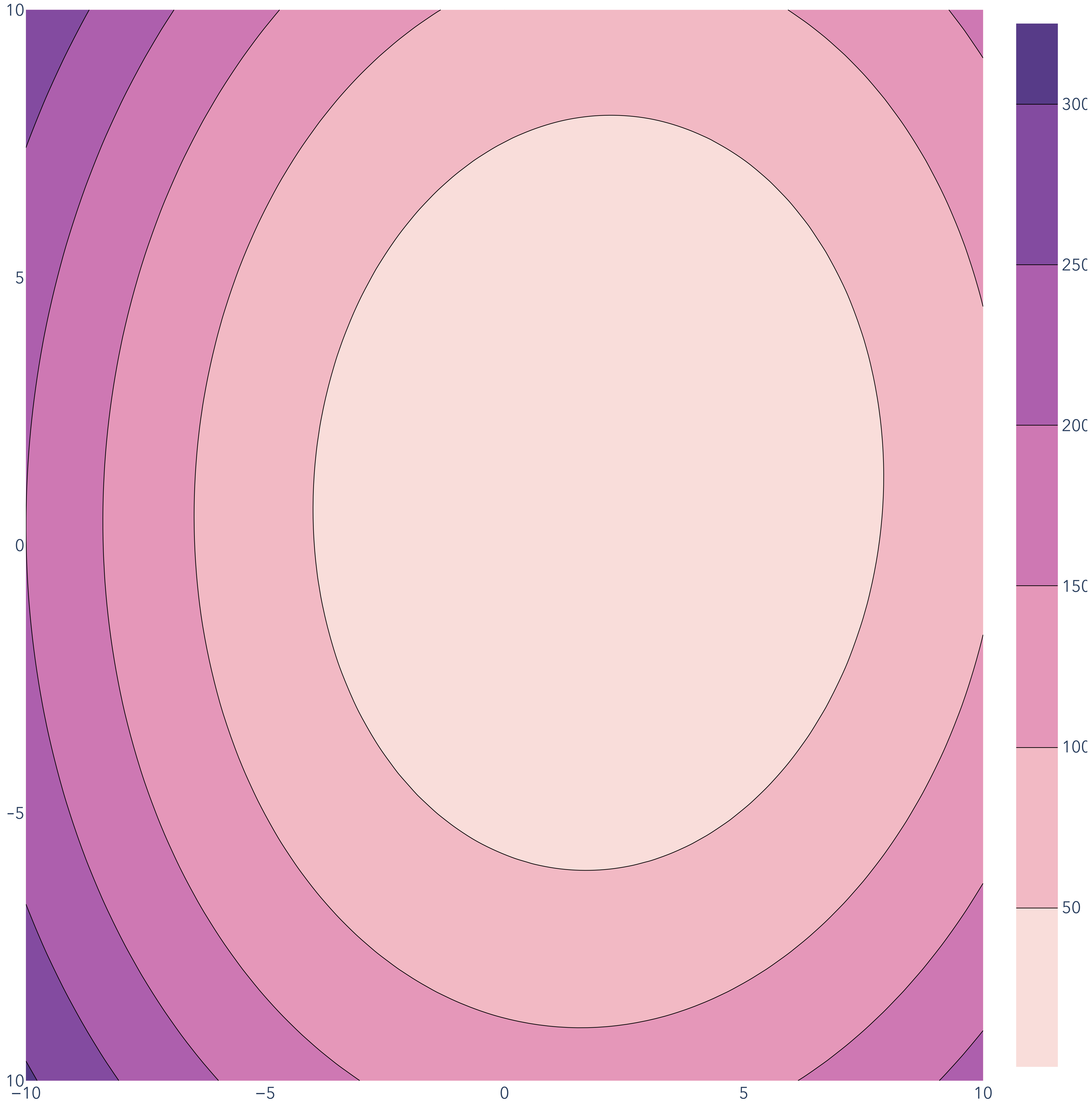
For a fixed dataset $X \in \mathbb{R}^{n \times 2}$ and $y \in \mathbb{R}^n$, the loss of any $w \in \mathbb{R}^2$ is:

$$\hat{R}_n : \mathbb{R}^2 \to \mathbb{R}$$

$$\hat{R}_n(w) = \|Xw - y\|^2.$$

We can visualize it with a loss surface.

The contours of the loss surface are its level sets $L_c := \{w \in \mathbb{R}^2 : \hat{R}_n(w) = c\}$.



Linear (Least Squares) Regression

Closed Form Solution

Given dataset $D_n := \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ we want to minimize the empirical risk:

$$\hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n}(w^\top x^{(i)} - y^{(i)})^2 = \frac{1}{n}\|Xw - y\|^2$$, where $X \in \mathbb{R}^{n\times d}$ is the **design matrix**.

Can we solve this this optimization problem?

$$w \in \arg\min_{w\in\mathbb{R}^d} \|Xw - y\|^2$$

Linear (Least Squares) Regression

Closed Form Solution

$$w \in \arg\min_{w \in \mathbb{R}^d} \|Xw - y\|^2$$

Analogy. How would you solve the one-dimensional optimization problem:

$$w \in \arg\min_{w \in \mathbb{R}^d} (aw - b)^2 \text{ ?}$$

From elementary calculus. Take derivative, set to 0 to find candidate minimizers, and verify the critical points are indeed minimizers by taking second derivatives.

Linear (Least Squares) Regression

Closed Form Solution

Given dataset $D_n := \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ we want to minimize the empirical risk:

$$\hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} (w^\top x^{(i)} - y^{(i)})^2 = \frac{1}{n} \|Xw - y\|^2$$, where $X \in \mathbb{R}^{n \times d}$ is the **design matrix**.

If $X \in \mathbb{R}^{n \times d}$ with $n \geq d$ and $\mathrm{rank}(X) = d$, the **closed form solution** is:

$$\hat{w} = (X^\top X)^{-1} X^\top y.$$

Outline

ERM: Learning as Optimization

Optimizing Linear Regression: Closed Form

Gradient Descent Intuition & Example

Gradient Descent Algorithm & Descent Lemma

Gradient Descent on Convex Functions

Stochastic Gradient Descent

Linear (Least Squares) Regression

Running Example

Input space: $\mathscr{X} = \mathbb{R}^d$

Hypothesis class is parametrized by $w \in \mathbb{R}^d$

Output space: $\mathscr{Y} = \mathbb{R}$ Action space: $\mathscr{A} = \mathscr{Y} = \mathbb{R}$

Loss Function: $\ell(\hat{y}, y) = (\hat{y} - y)^2$

Hypothesis Class: $\mathscr{H} = \{h : \mathbb{R}^d \to \mathbb{R} : h(x) = w^\top x, w \in \mathbb{R}^d\}$

Given dataset $D_n := \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ we want to minimize the empirical risk:

$$\hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n}(w^\top x^{(i)} - y^{(i)})^2 \text{ or } \hat{R}_n(w) = \frac{1}{n}\|Xw - y\|^2 \text{ with } X \in \mathbb{R}^{n\times d}, y \in \mathbb{R}^n.$$

Objective in scalar form

Objective in matrix-vector form

Linear Regression

Running Example

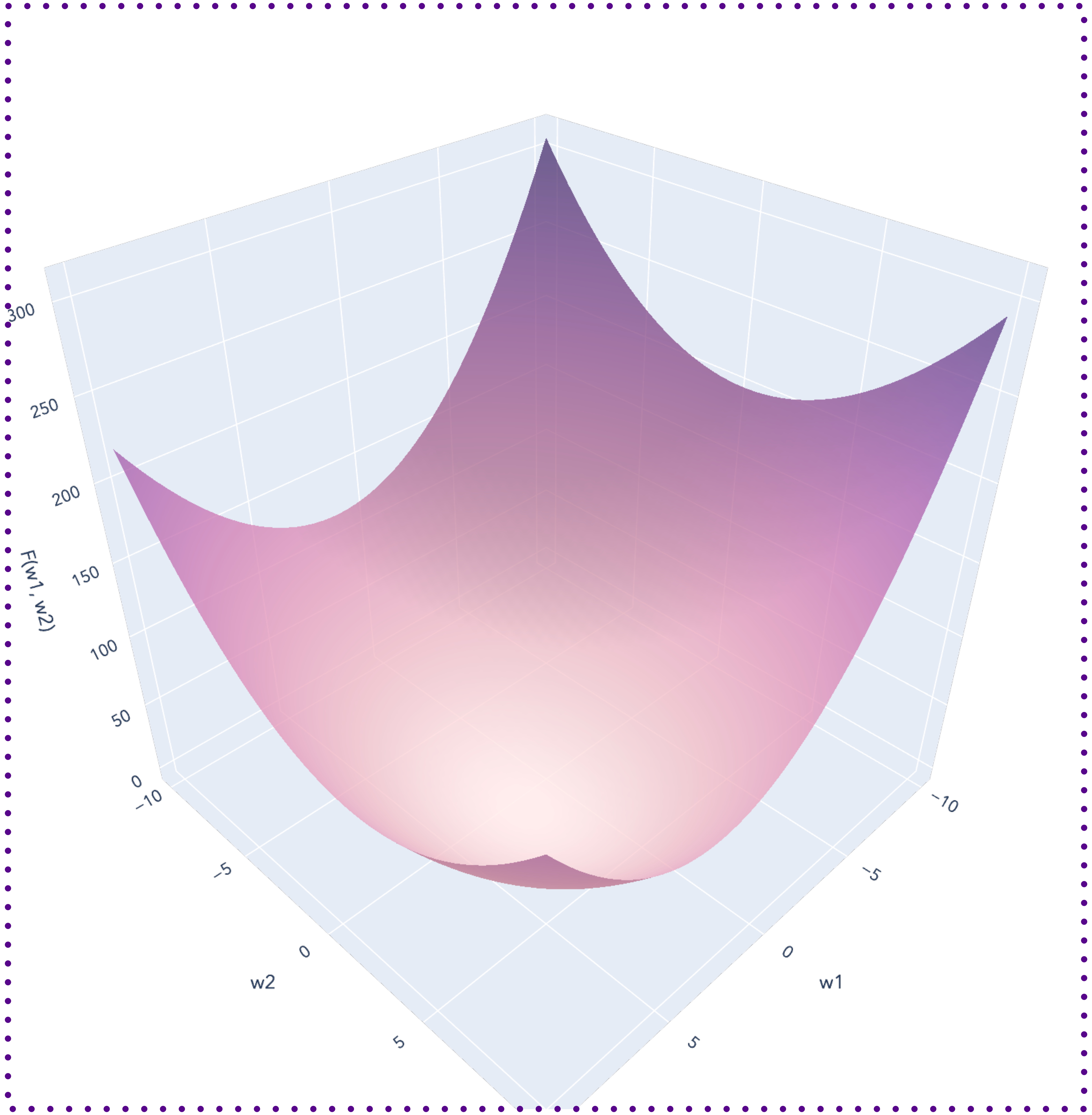
Given $D_n := \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ we want to minimize the empirical risk:

$$\hat{R}_n(w) = \frac{1}{n}\|Xw - y\|^2$$

with $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$.

Closed-form solution: $(X^\top X)^{-1} X^\top y$.

We can also solve *iteratively*.



Linear Regression

Running Example

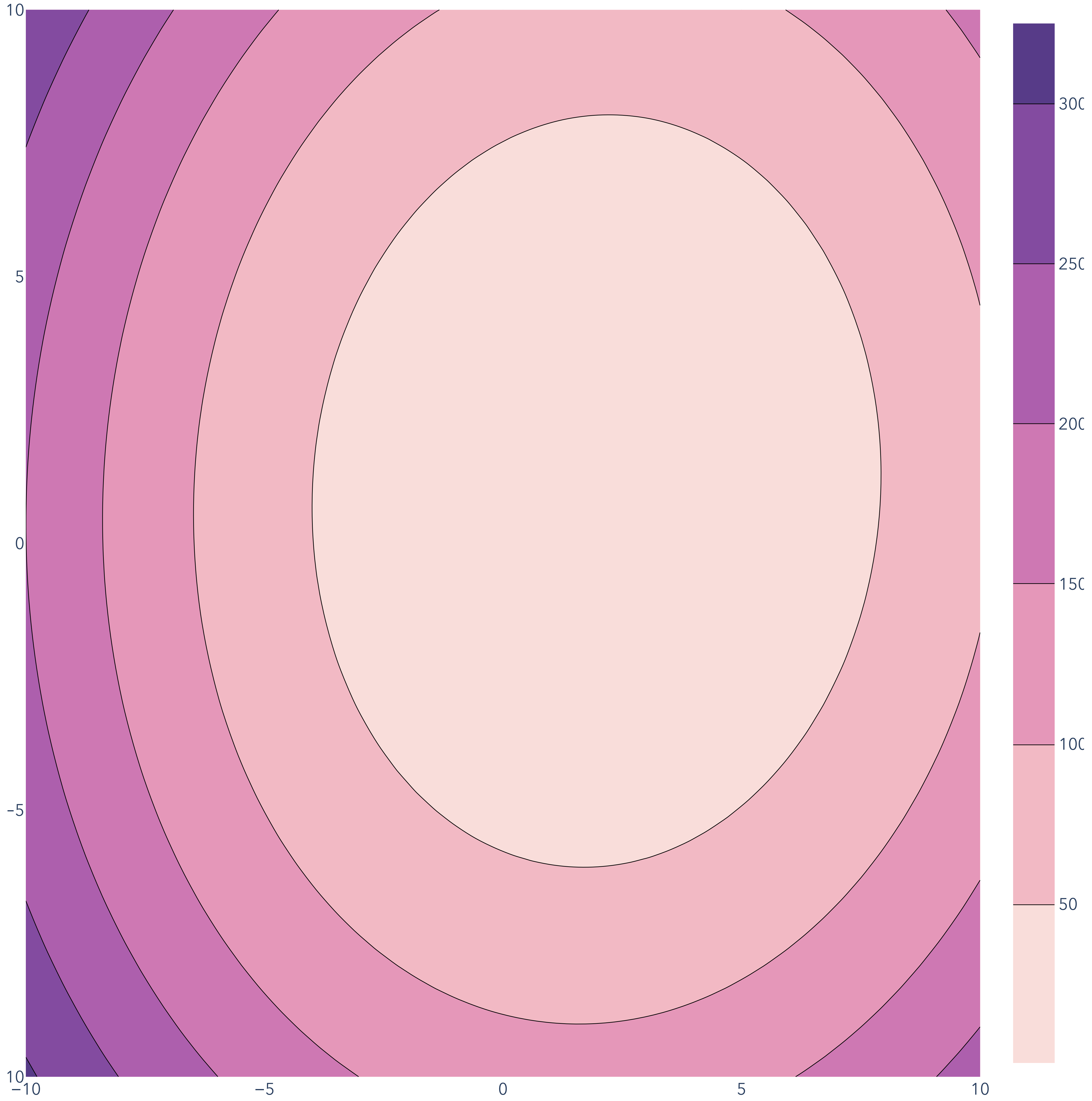
Given $D_n := \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ we want to minimize the empirical risk:

$$\hat{R}_n(w) = \frac{1}{n}\|Xw - y\|^2$$

with $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$.

Closed-form solution: $(X^\top X)^{-1} X^\top y$.

We can also solve *iteratively*.



Unconstrained Optimization

Setting

$$\min_{w \in \mathbb{R}^d} \; F(w)$$

where we assume the <u>objective function</u> $F : \mathbb{R}^d \to \mathbb{R}$ is differentiable.

<u>Goal:</u> Given an objective function F, find the w that makes $F(w)$ as small as possible.

Unconstrained Optimization

Setting

$$\min_{w \in \mathbb{R}^d} \; F(w)$$

where we assume the objective function $F : \mathbb{R}^d \to \mathbb{R}$ is differentiable.

Example: Linear regression ERM objective: $\min_{w \in \mathbb{R}^d} \boxed{\frac{1}{n} \sum_{i=1}^{n} (w^\top x^{(i)} - y^{(i)})^2 = \frac{1}{n} \|Xw - y\|^2}$

Objective function $F(w)$

A candidate algorithm

Moving in steepest descent direction

$$\underset{w \in \mathbb{R}}{\text{minimize}} \quad F(w)$$

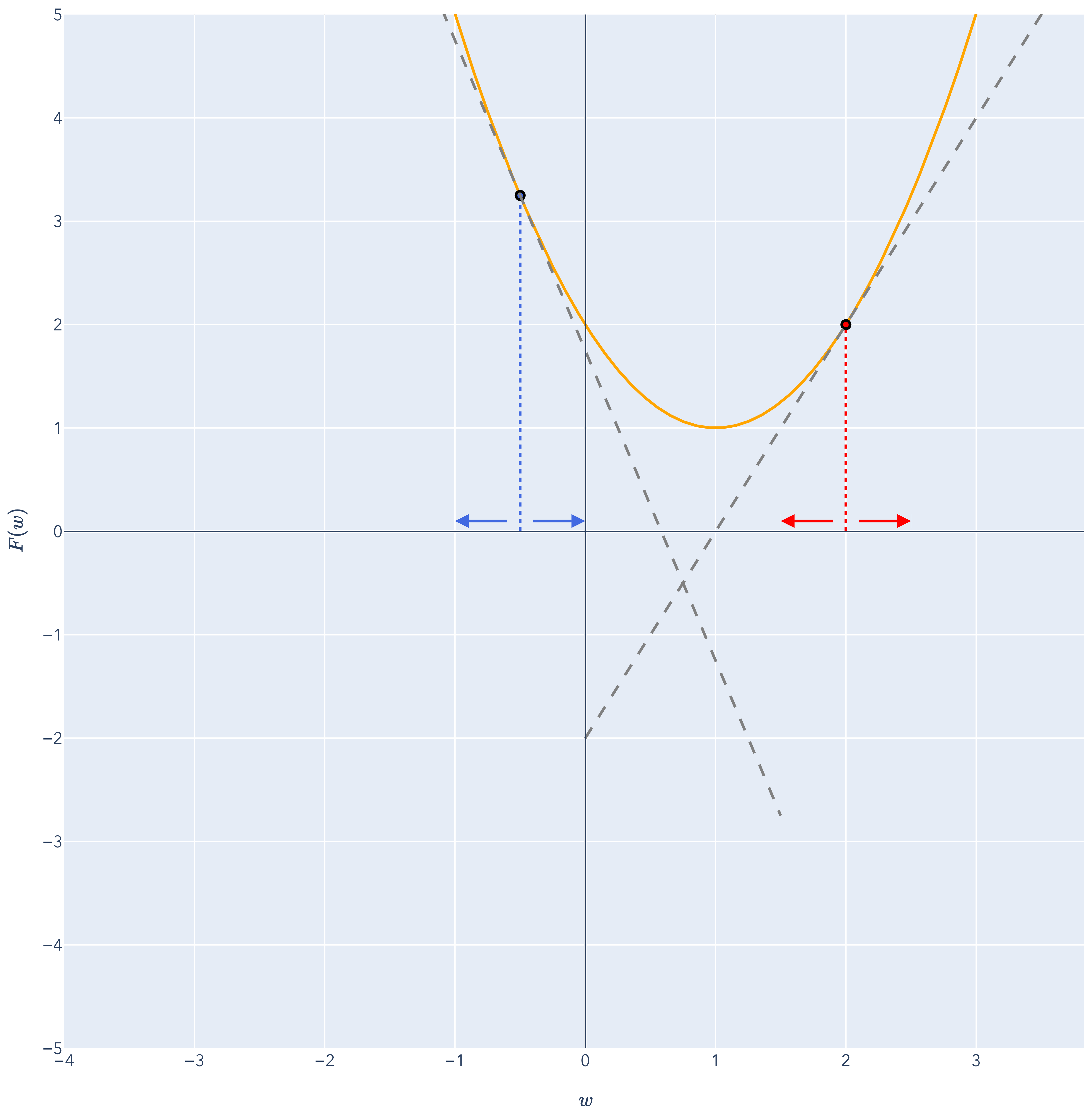
Suppose I drop you off at $w = -0.5$.

Or at $w = 2$.

Which direction to go in to *decrease* F?

If slope is negative, go right.

If slope is positive, go left.



A candidate algorithm

Moving in steepest descent direction

$$\underset{w \in \mathbb{R}}{\text{minimize}} \quad F(w)$$

Suppose I drop you off at $w = -0.5$.

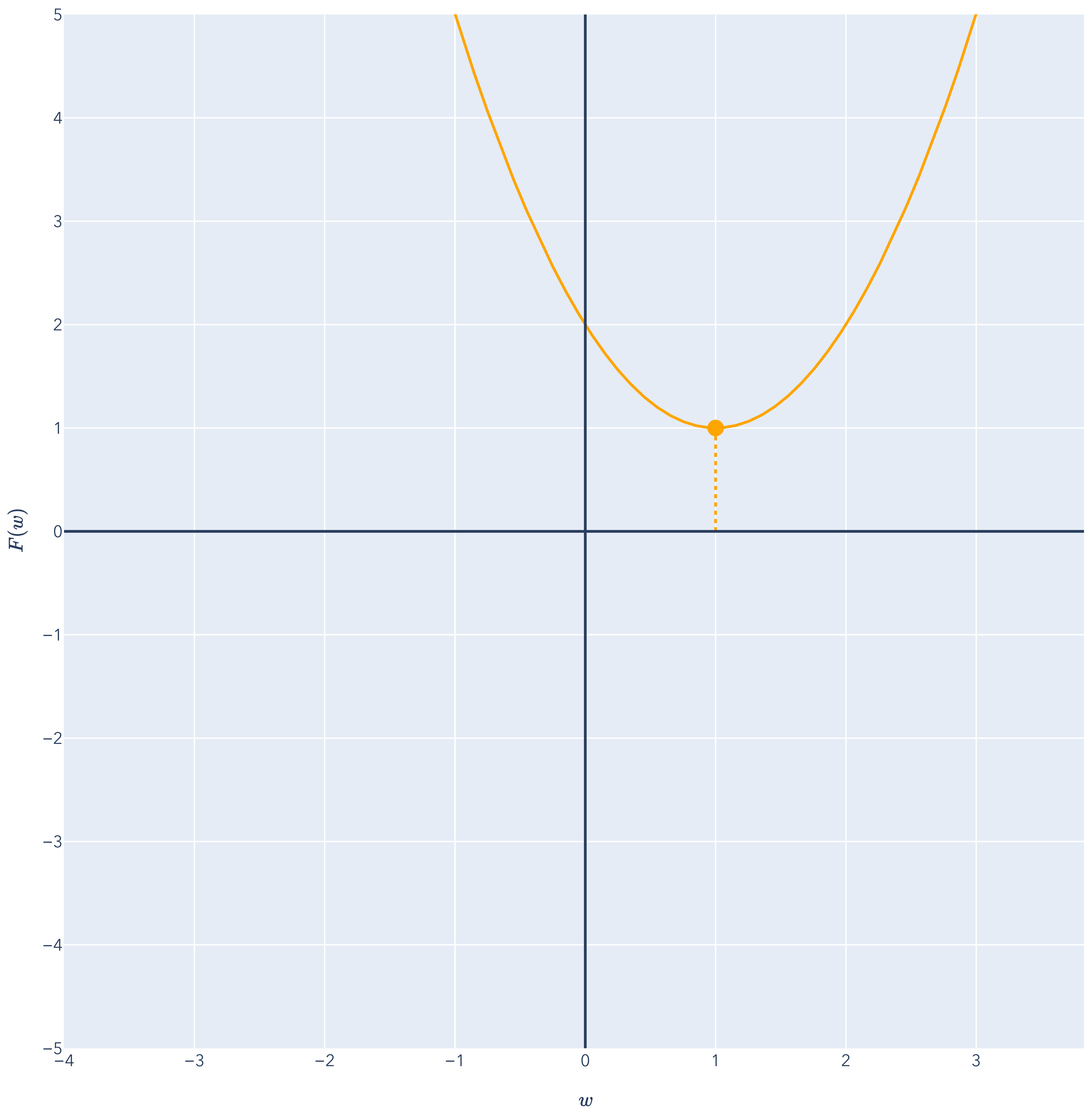
Or at $w = 2$.

Which direction to go in to *decrease* F?

Follow the derivative (slope at a point)!

Repeat over and over to minimize.

Eventually, we might reach a minimum!



A candidate algorithm

Moving in steepest descent direction

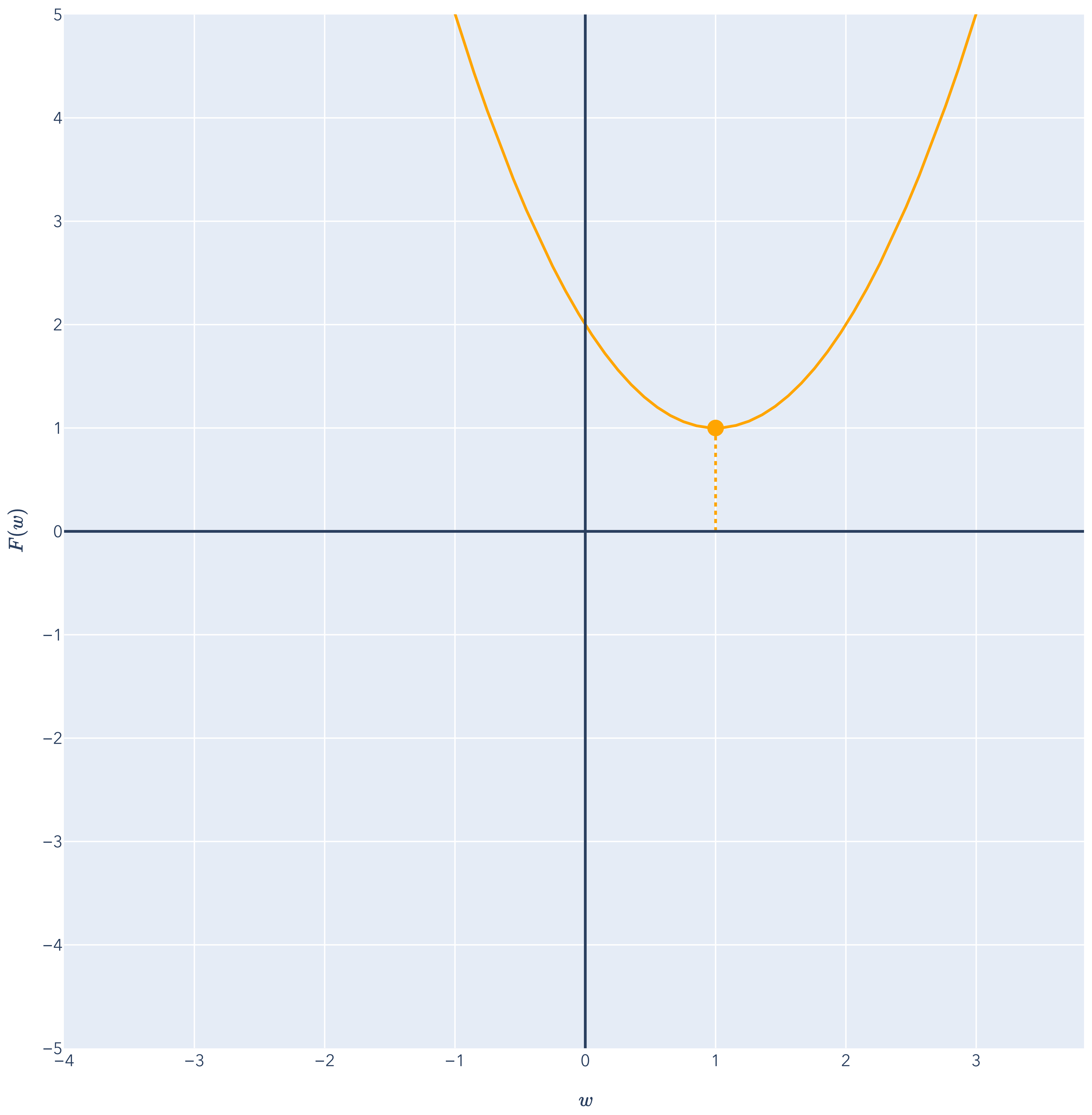
$$\underset{w \in \mathbb{R}}{\text{minimize}} \quad F(w)$$

But we can also just minimize in one shot!

$$F'(w) = 0$$

(first order condition)

Not always possible (e.g. logistic regression, neural networks, etc.), so we need an *iterative* algorithm.



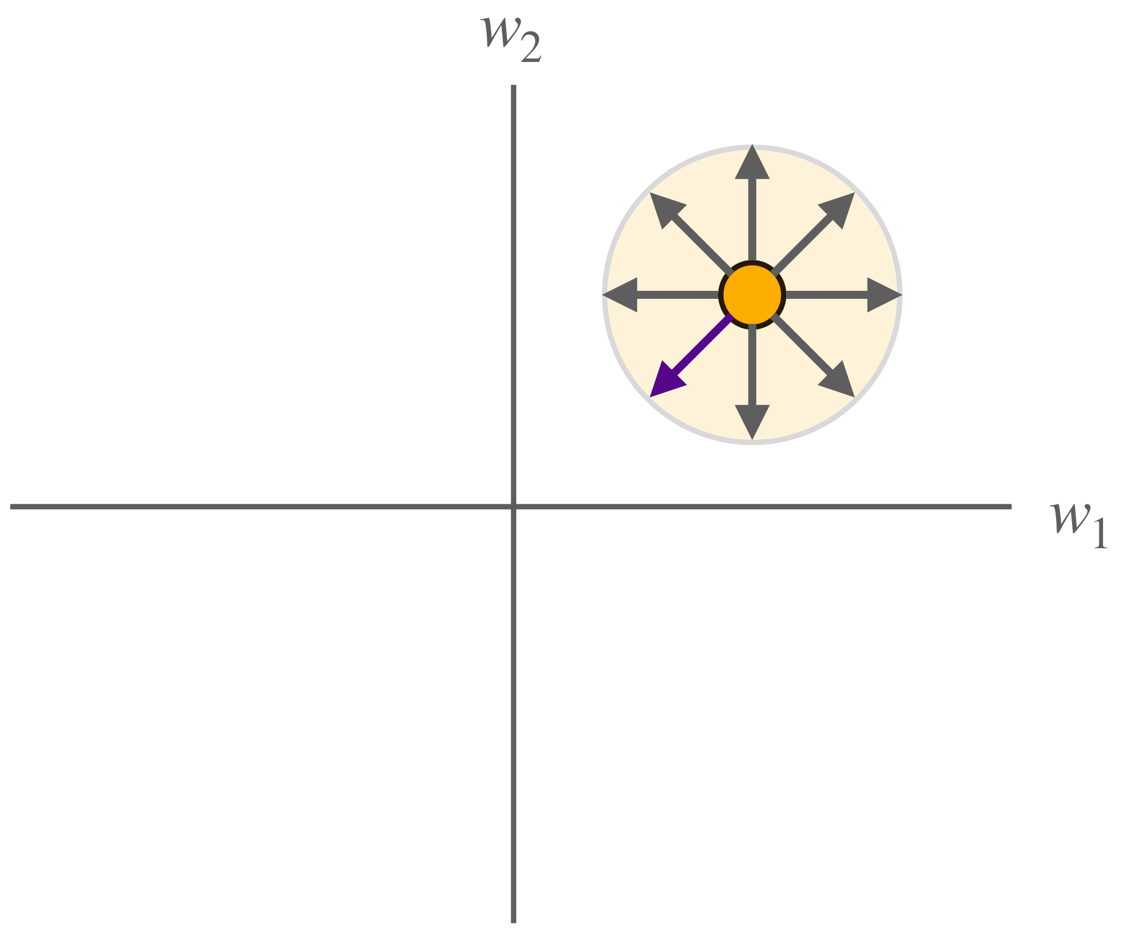
A candidate algorithm

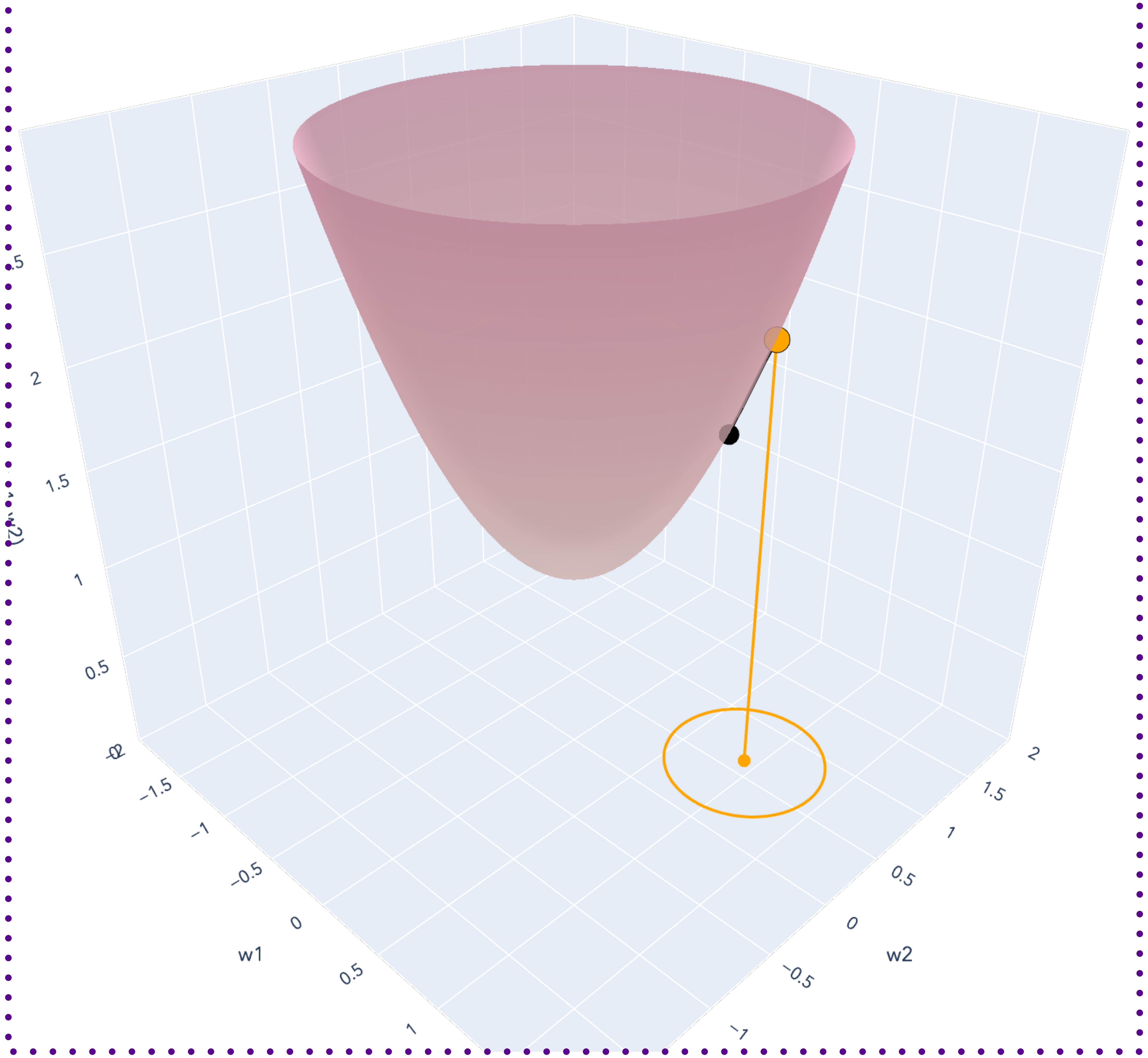
Moving in steepest descent direction

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \quad F(w)$$

Infinitely many directions now in $d \geq 2$…

But still can go in the "steepest decrease" direction!





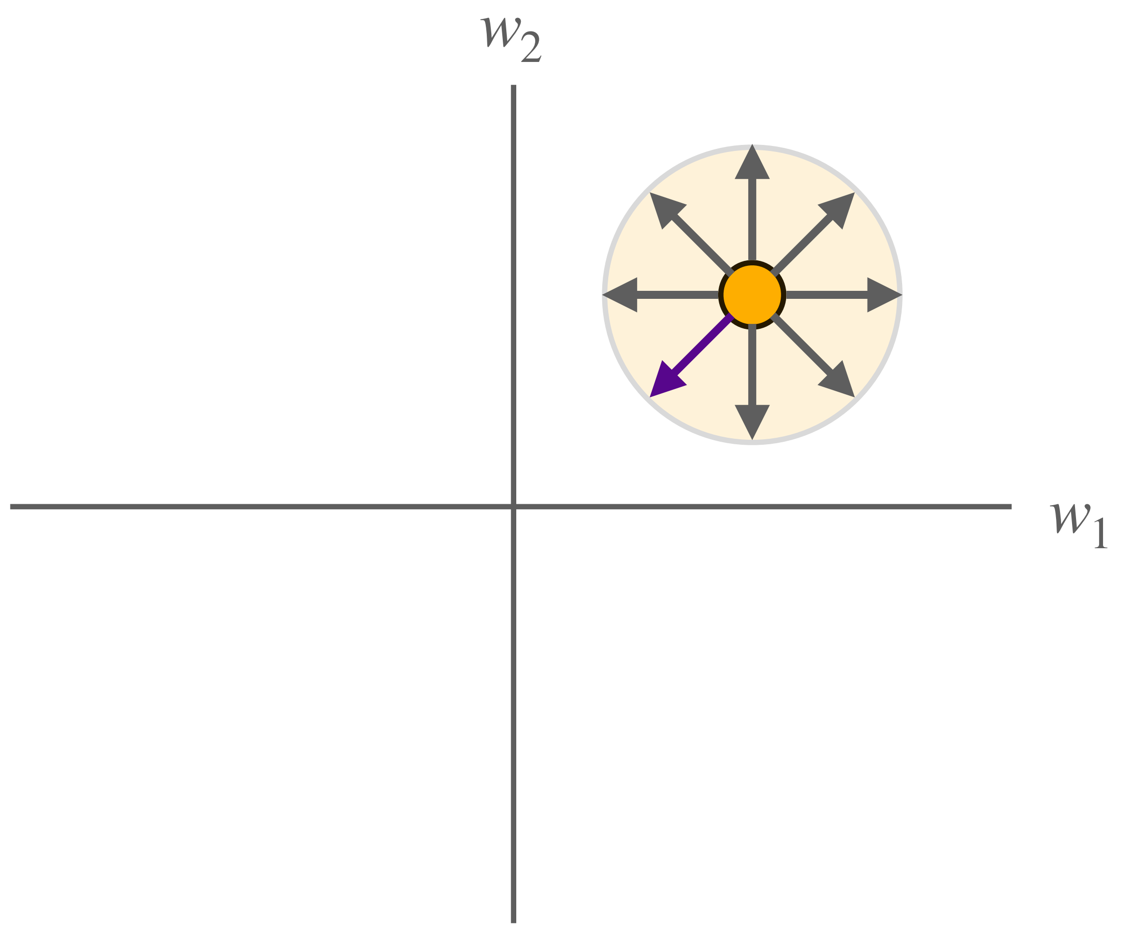
A candidate algorithm

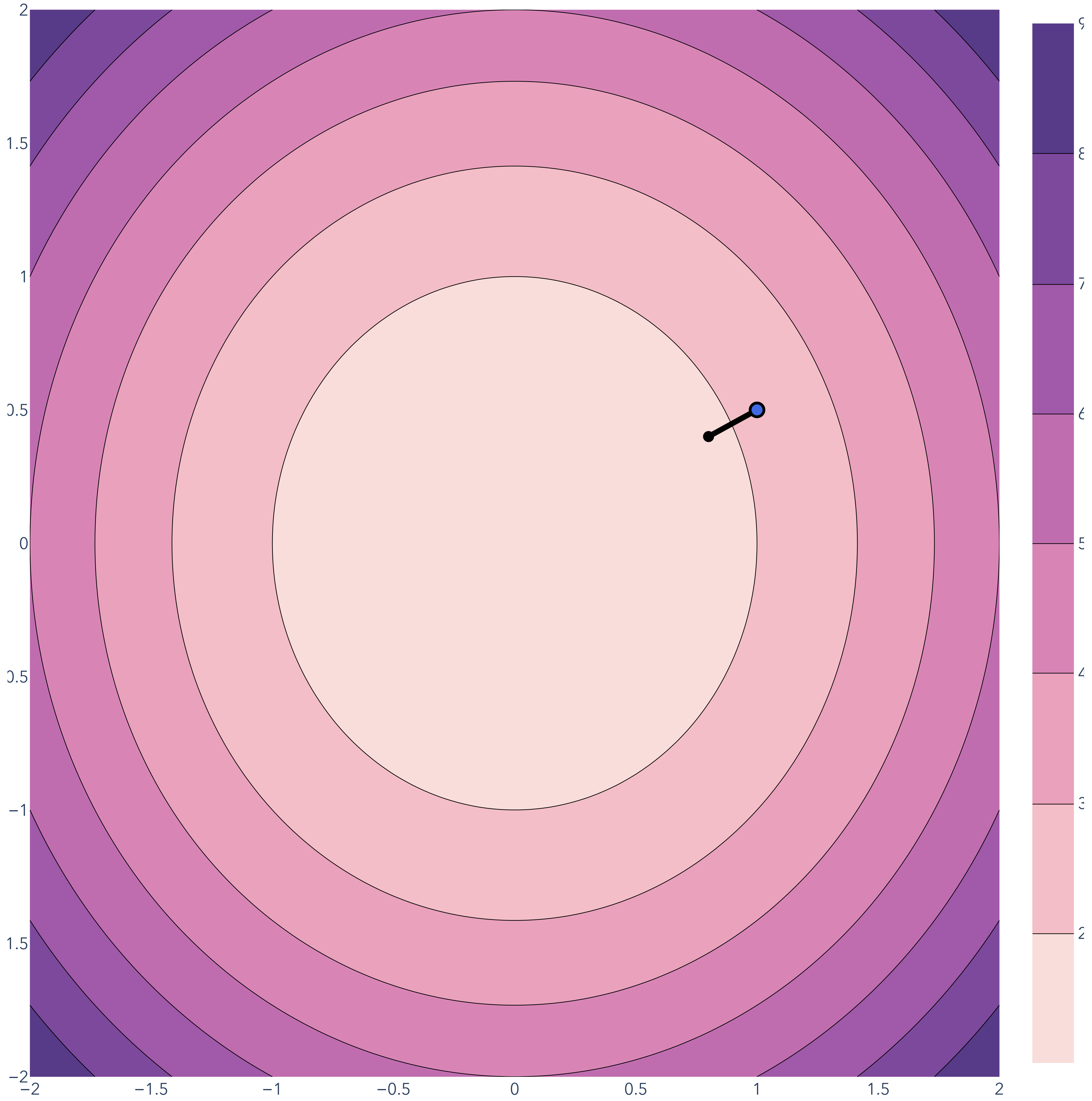
Moving in steepest descent direction

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \quad F(w)$$

Infinitely many directions now in $d \geq 2$…

But still can go in the "steepest decrease" direction!





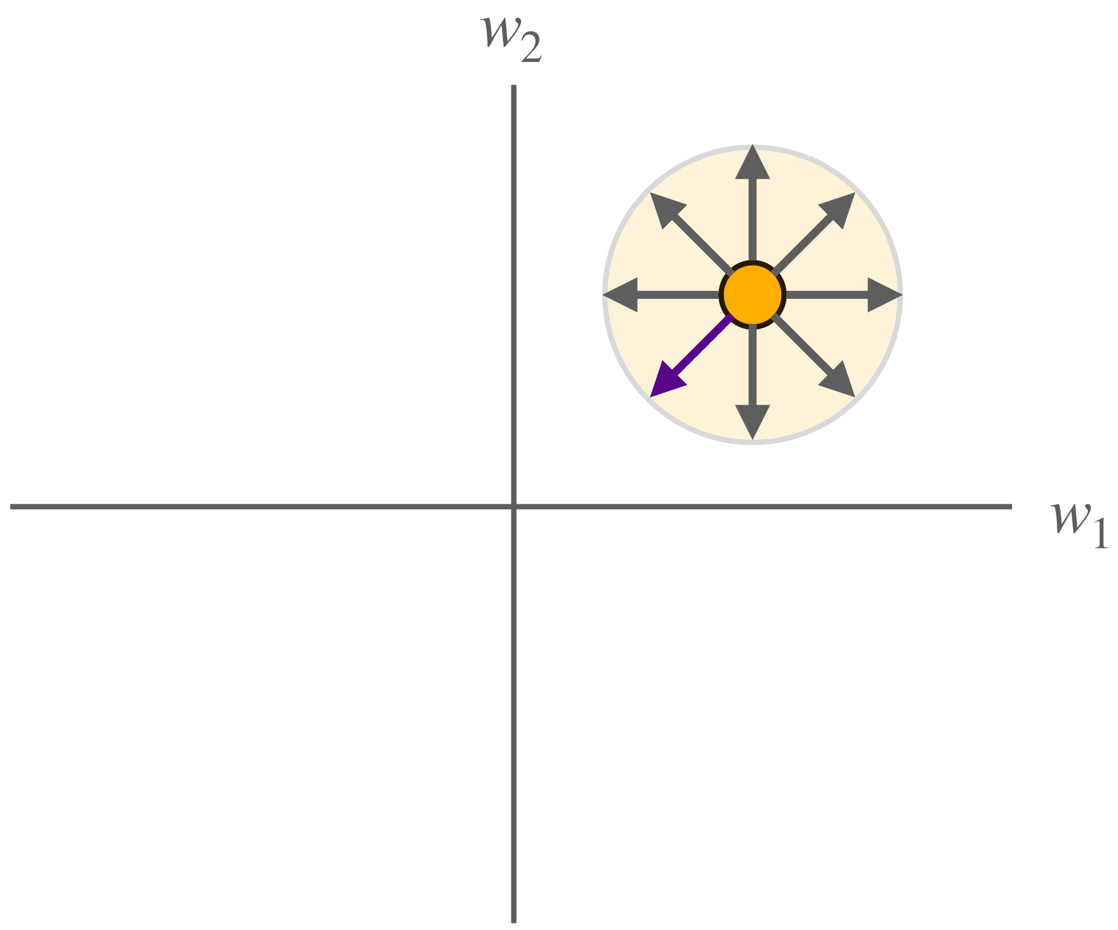
A candidate algorithm

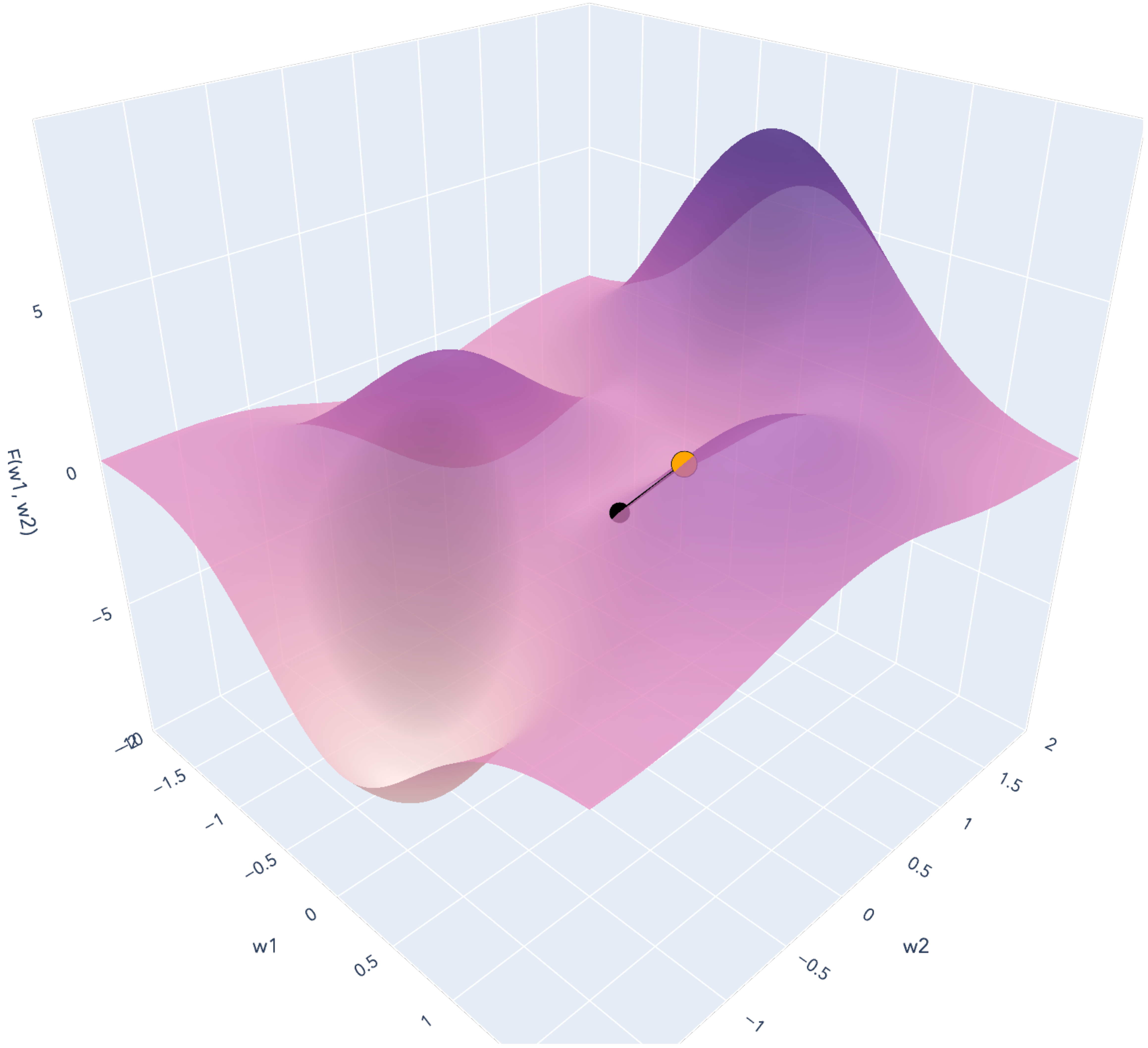
Moving in steepest descent direction

$$\underset{\mathbf{w} \in \mathbb{R}^d}{\text{minimize}} \quad F(w)$$

$$F(w_1, w_2)$$

This "greedy" strategy works for arbitrarily complex functions.





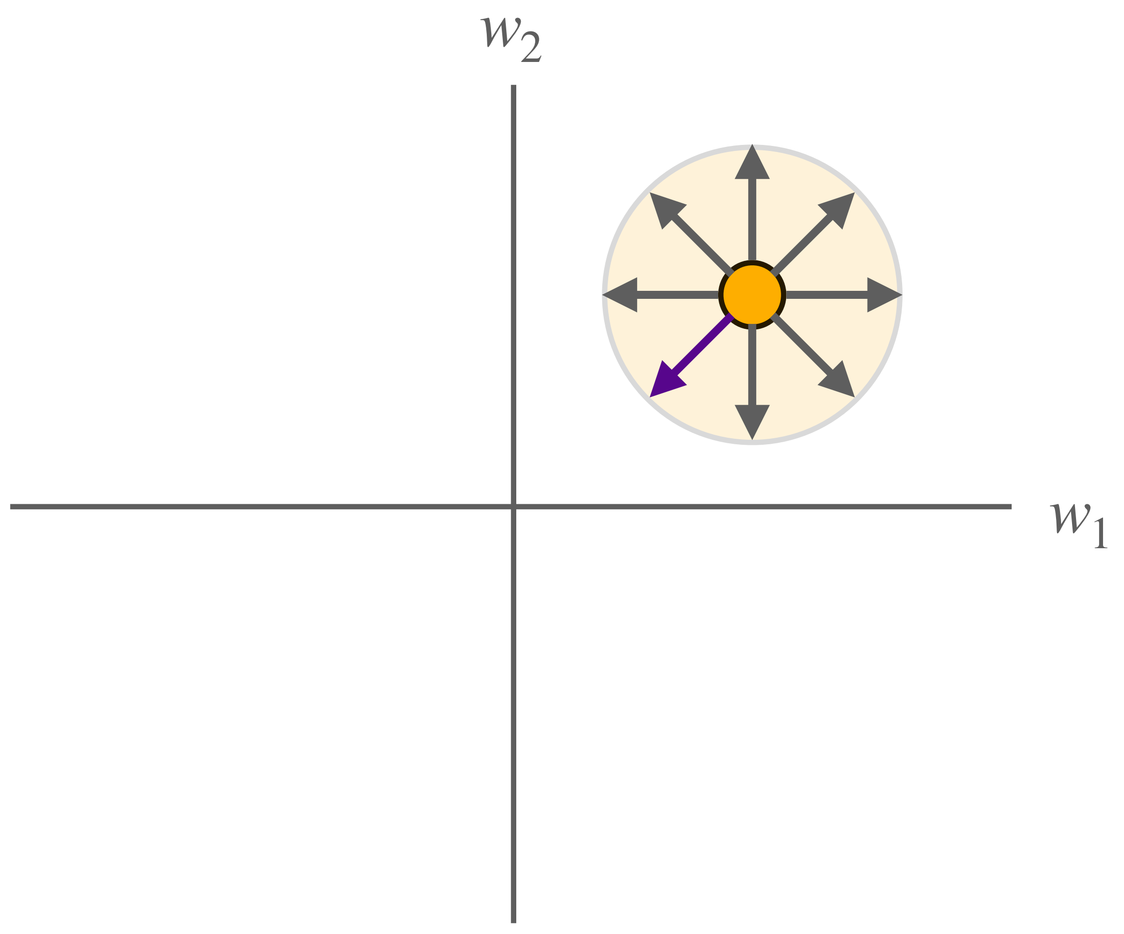
A candidate algorithm

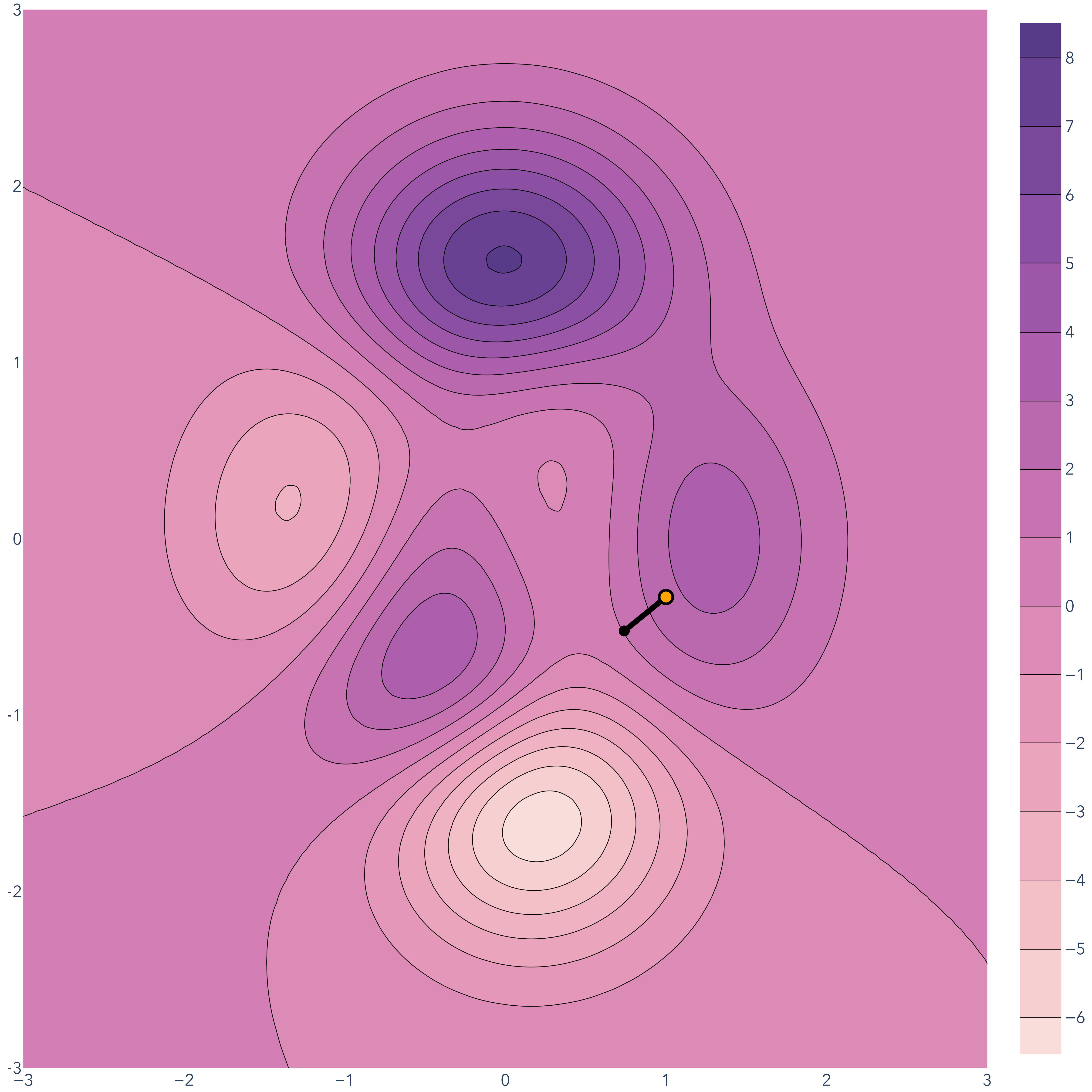
Moving in steepest descent direction

$$\underset{\mathbf{w}\in\mathbb{R}^d}{\text{minimize}} \quad F(w)$$

$$F(w_1, w_2)$$

This "greedy" strategy works for arbitrarily complex functions.





A candidate algorithm

Moving in steepest descent direction

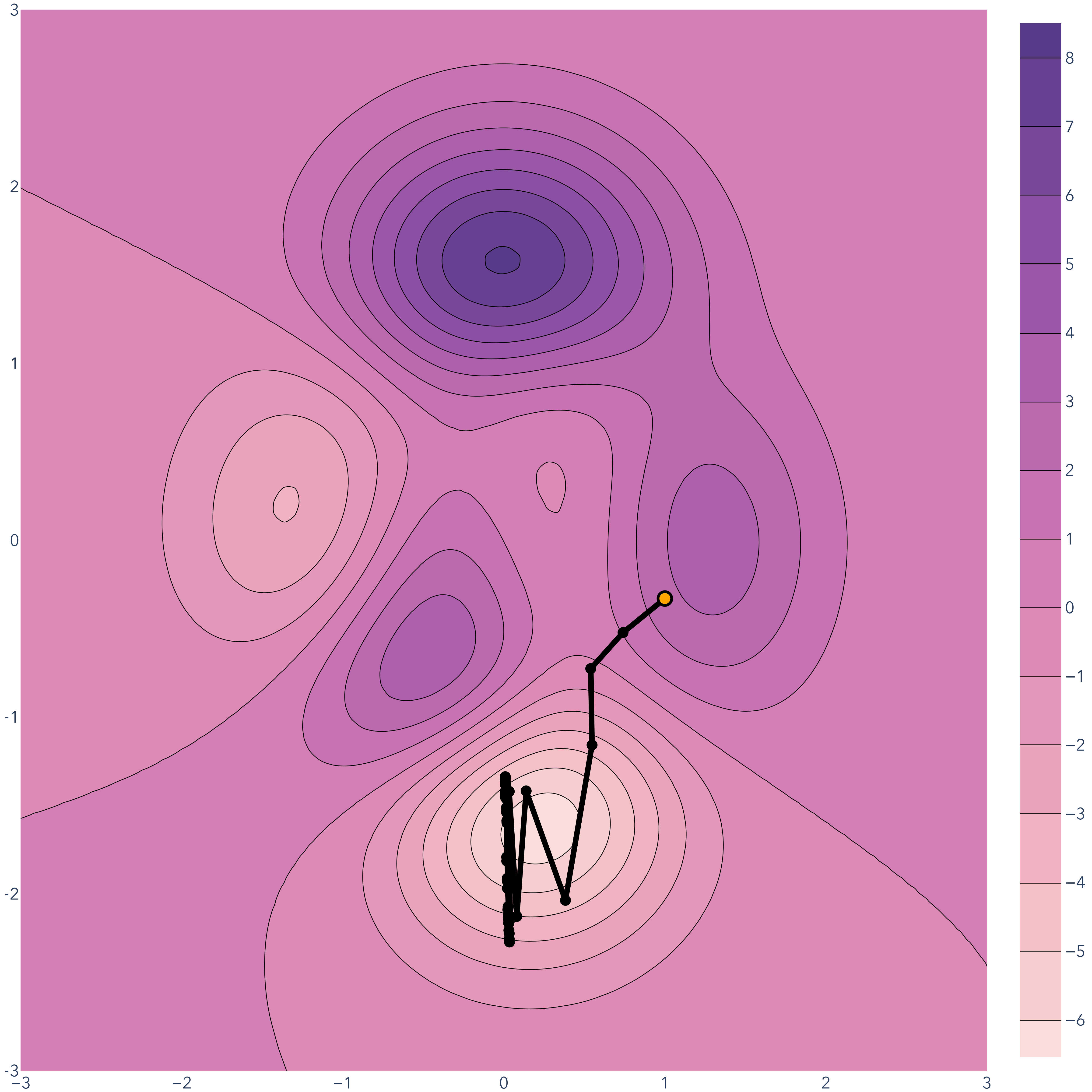
Start at some arbitrary point $w^{(0)} \in \mathbb{R}^d$.

Step in the direction of steepest decrease for $F(w)$…

Take another step in the direction of steepest decrease for $F(w)$…

$\vdots$

Repeat until satisfied.



A candidate algorithm

Moving in steepest descent direction

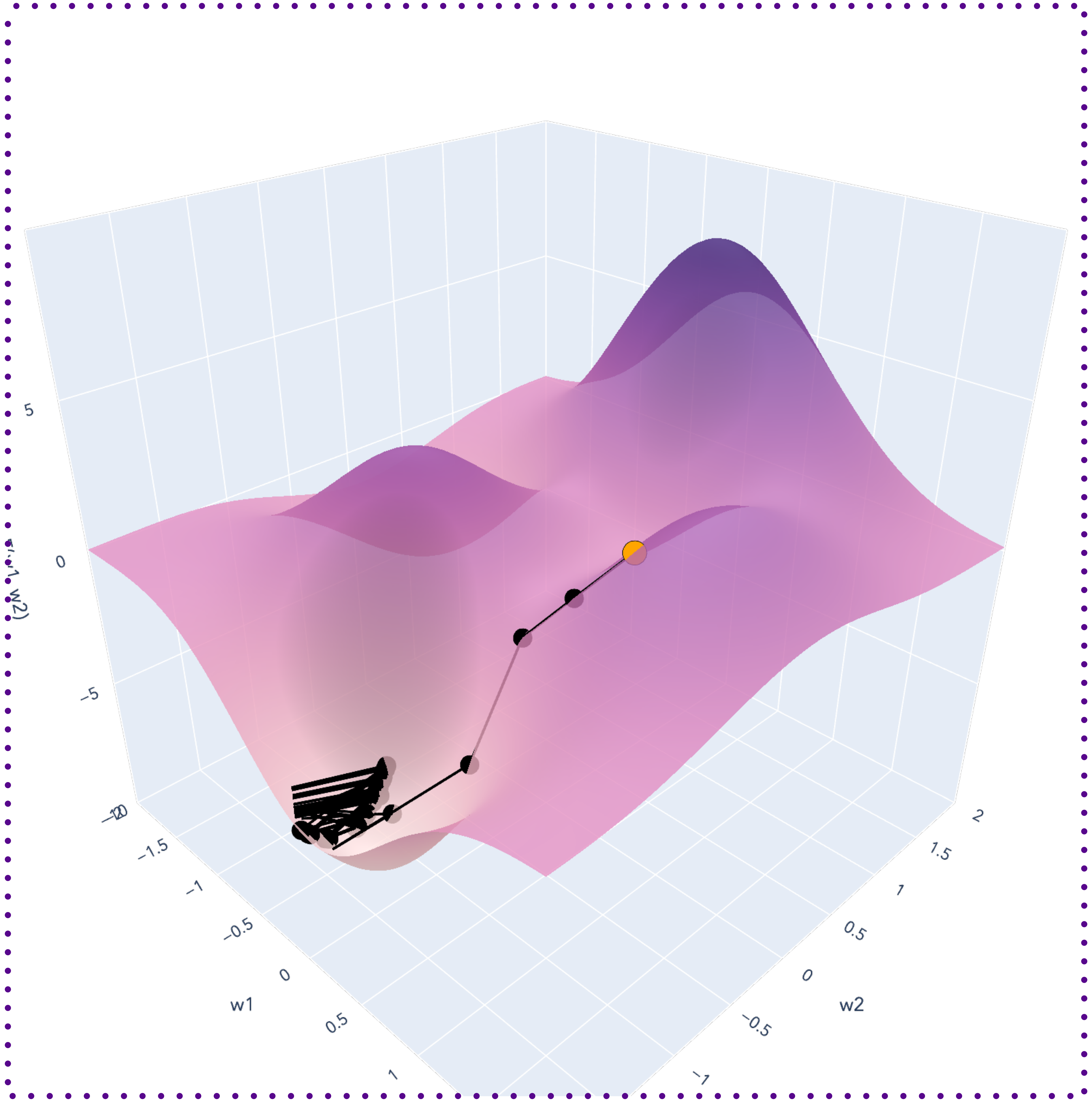
Start at some arbitrary point $w^{(0)} \in \mathbb{R}^d$.

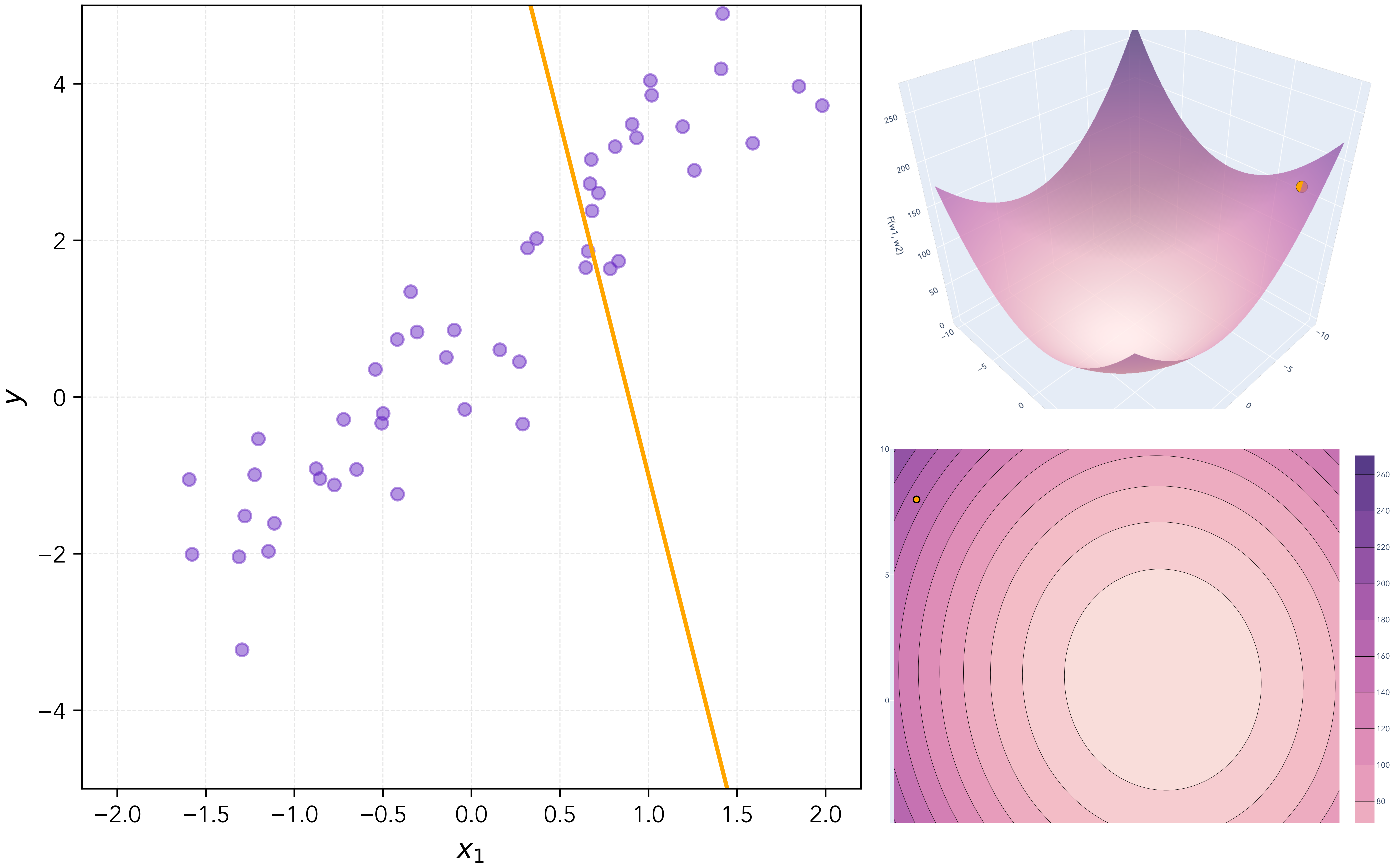
Step in the direction of steepest decrease for $F(w)$…

Take another step in the direction of steepest decrease for $F(w)$…

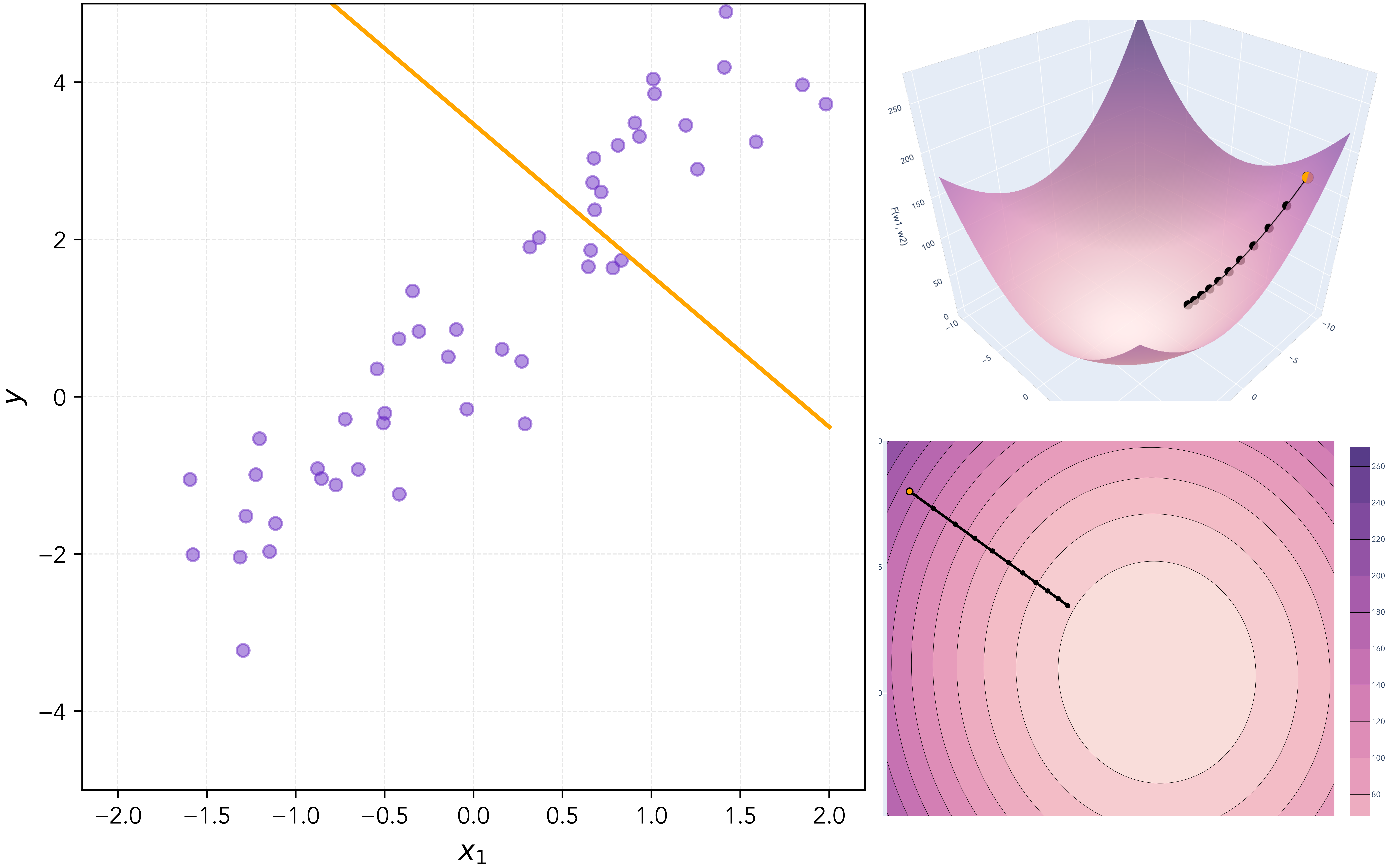
$\vdots$

Repeat until satisfied.



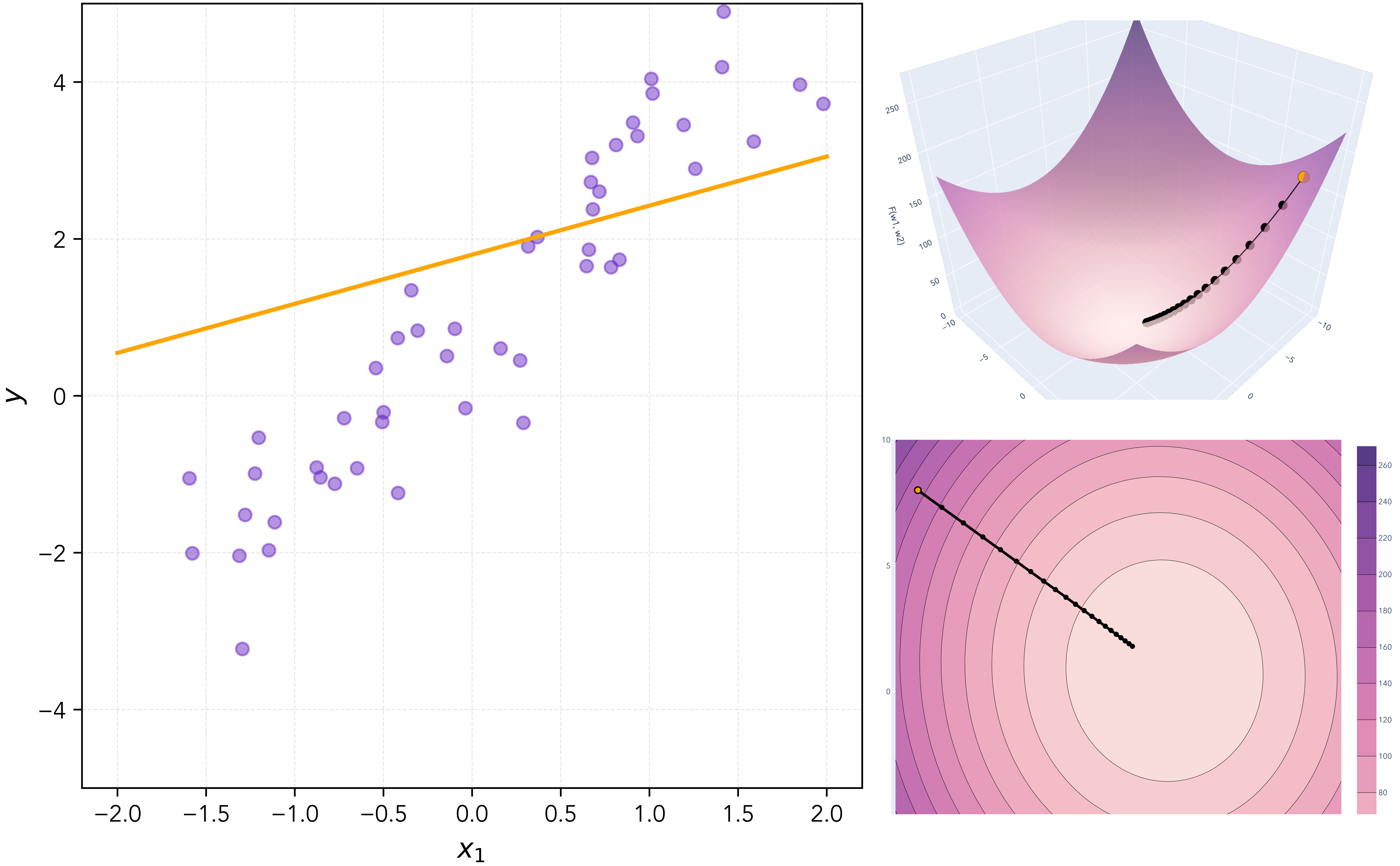


$T = 0$

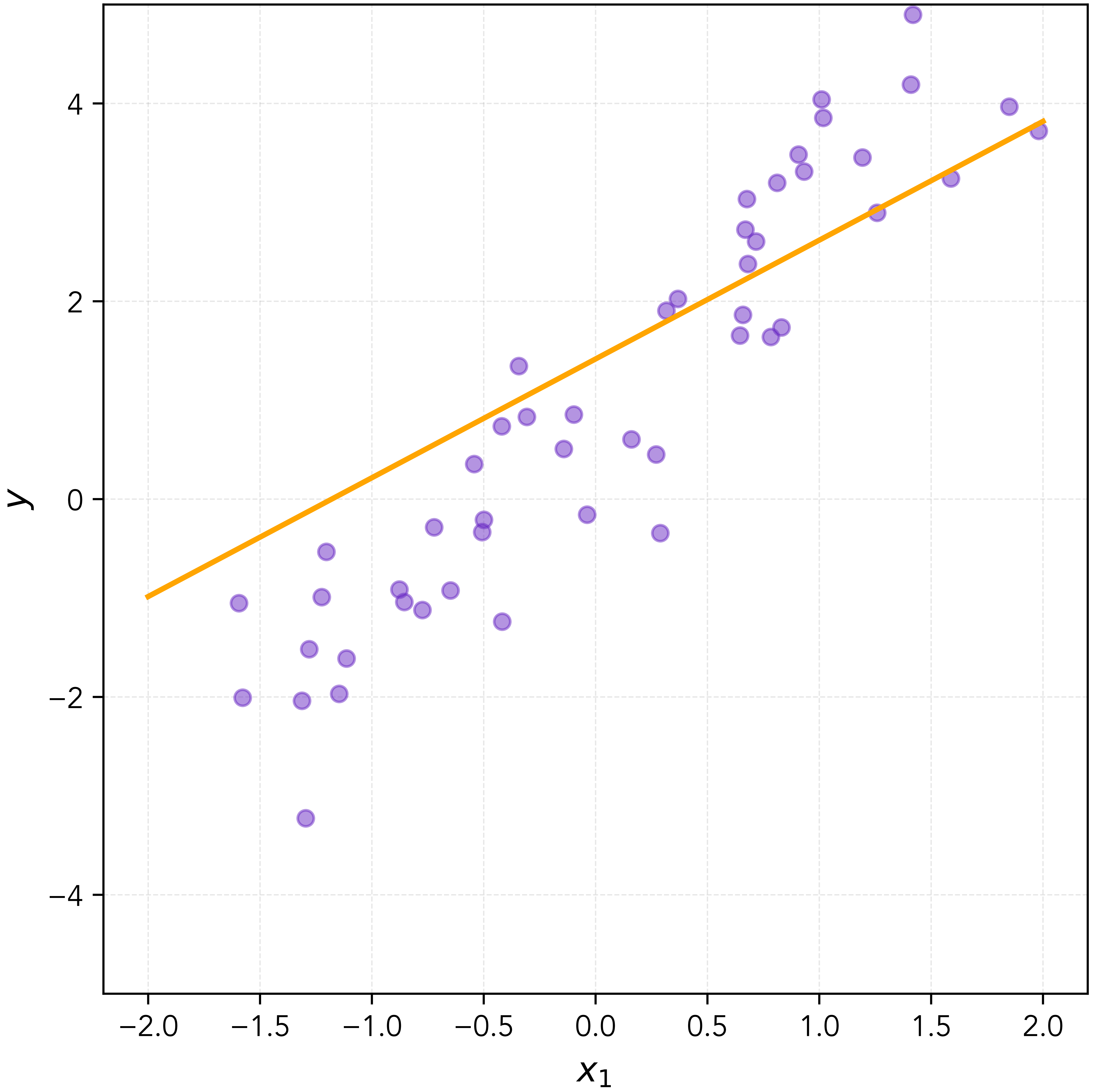


y
4
2
0
−2
−4
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_1
F(w1, w2)
250
200
150
100
50
0
−10
−5
0
260
240
220
200
180
160
140
120
100
80

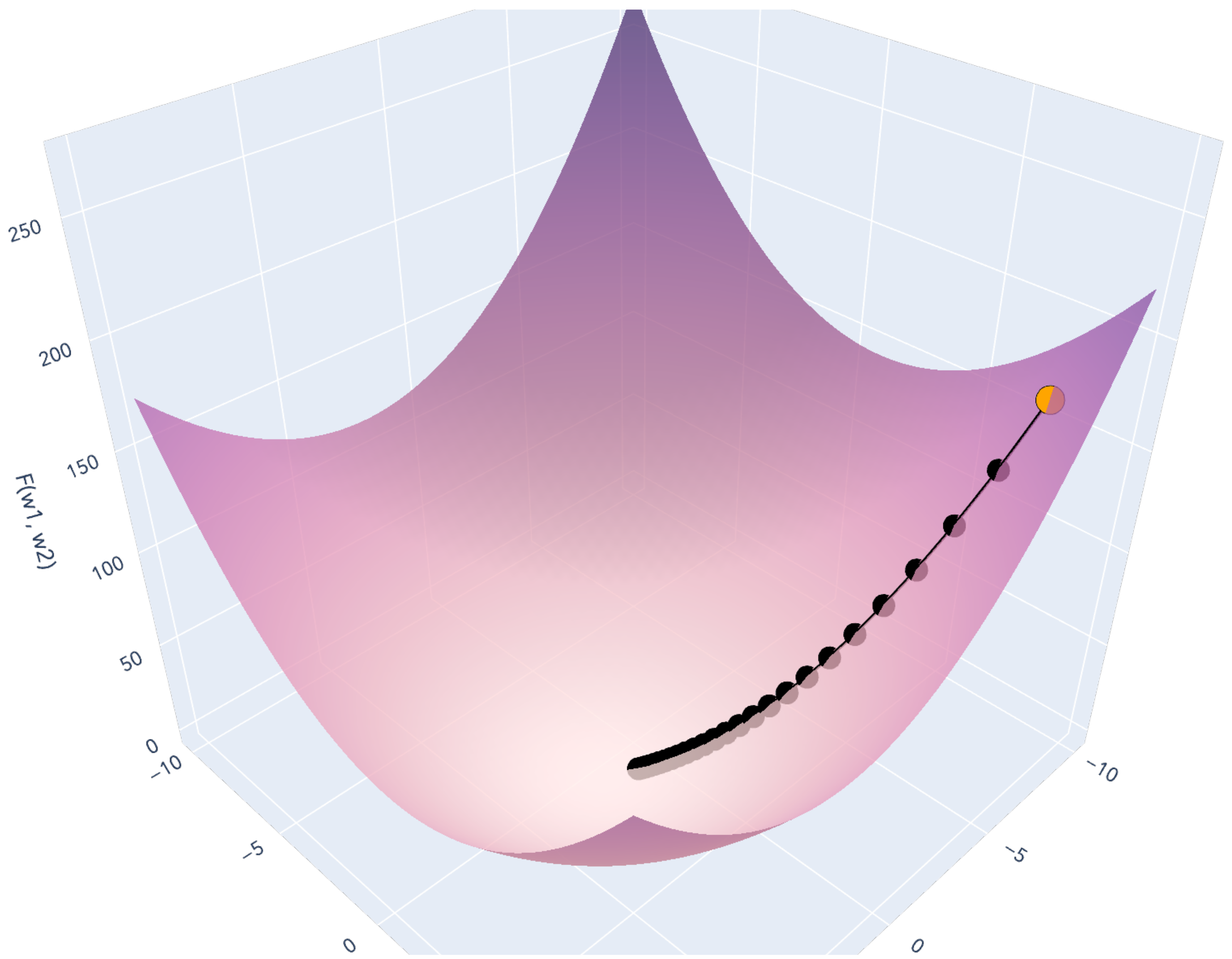
$T = 10$


4
2
0
−2
−4
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x1
F(w1, w2)
250
200
150
100
50
0
−10
−5
0
−10
−5
0
10
5
0
260
240
220
200
180
160
140
120
100
80

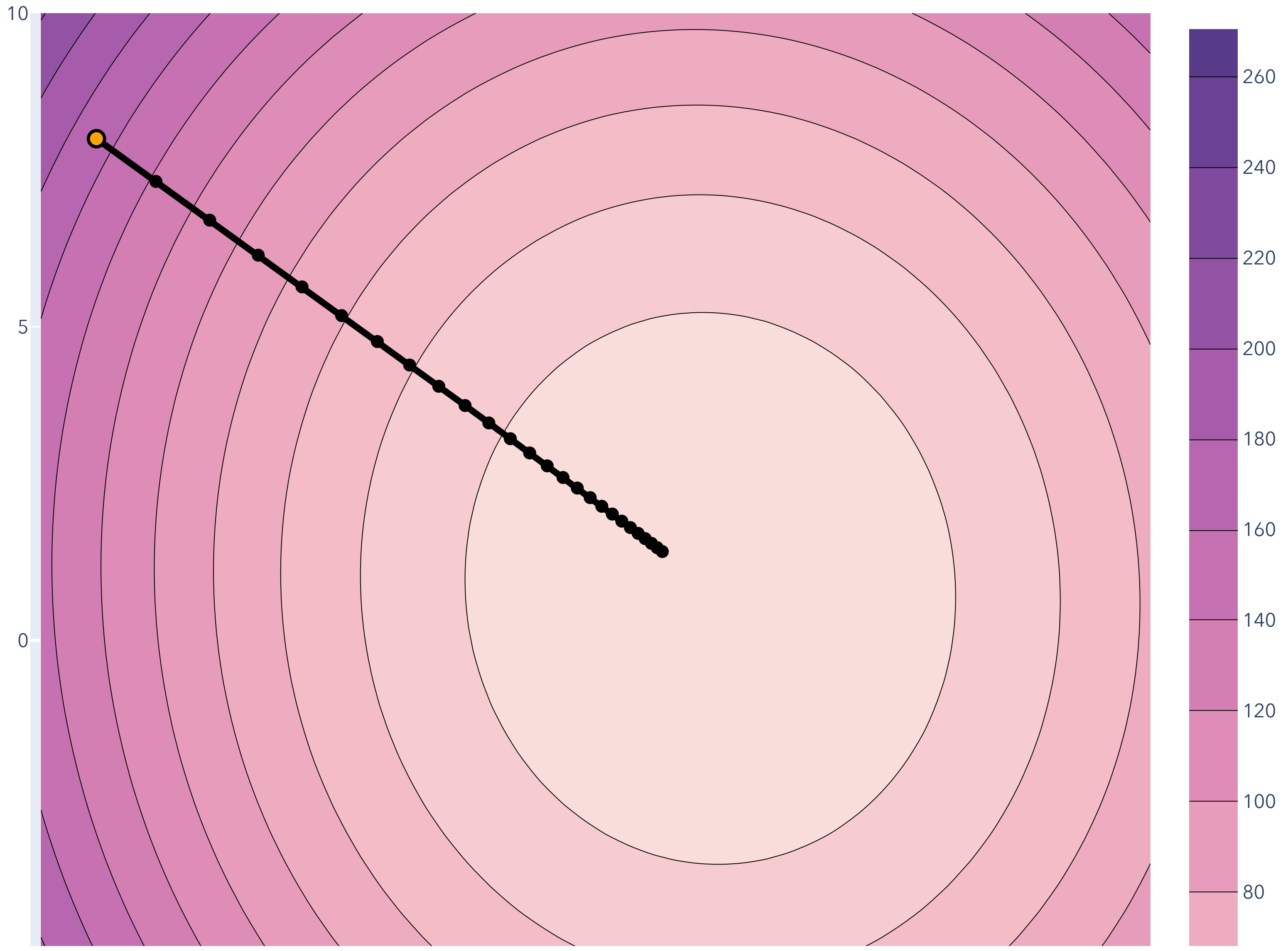
$T = 20$



4
2
0
−2
−4
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_1

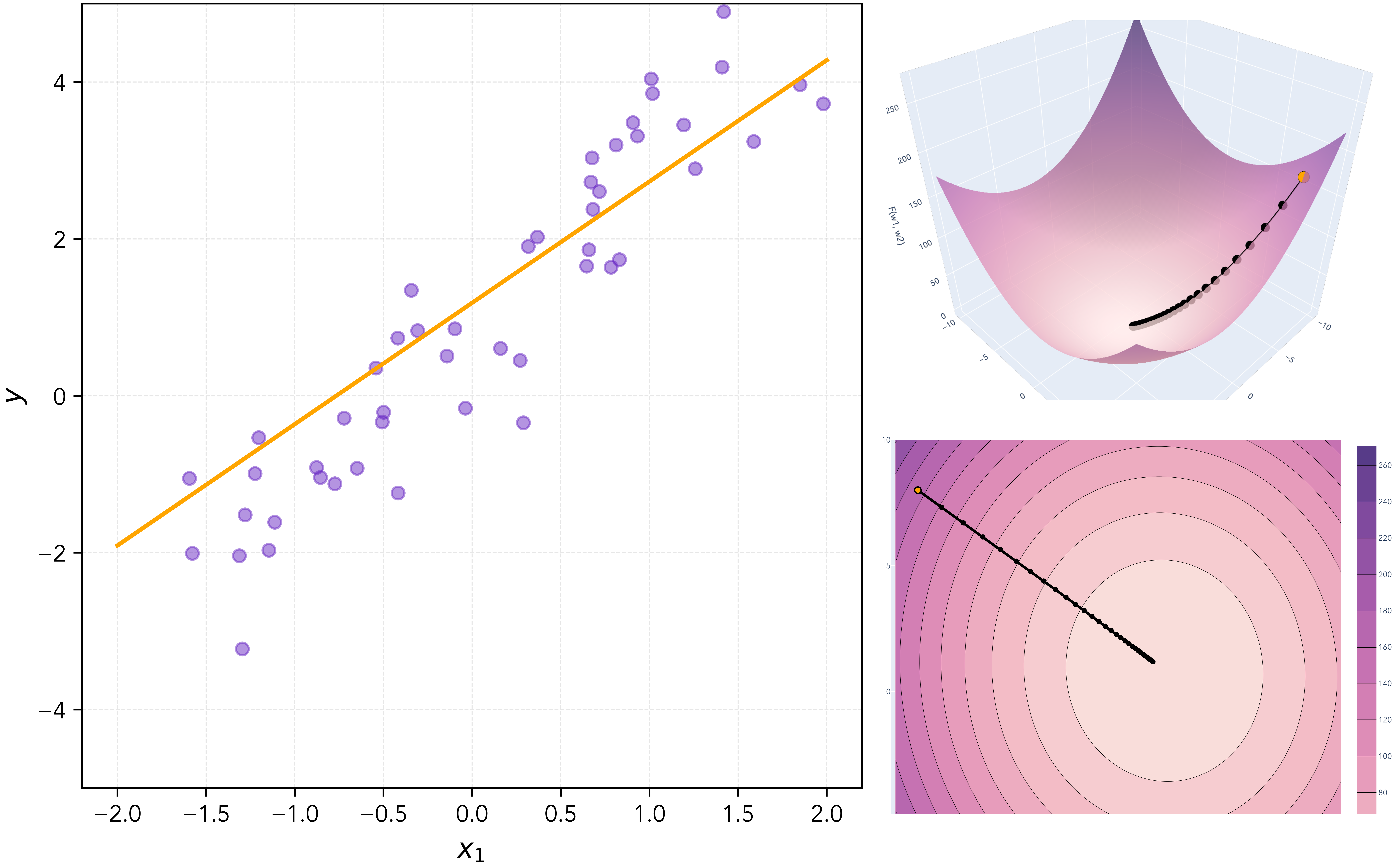


250
200
150
100
50
0
F(w1, w2)
−10
−5
0
−10
−5
0



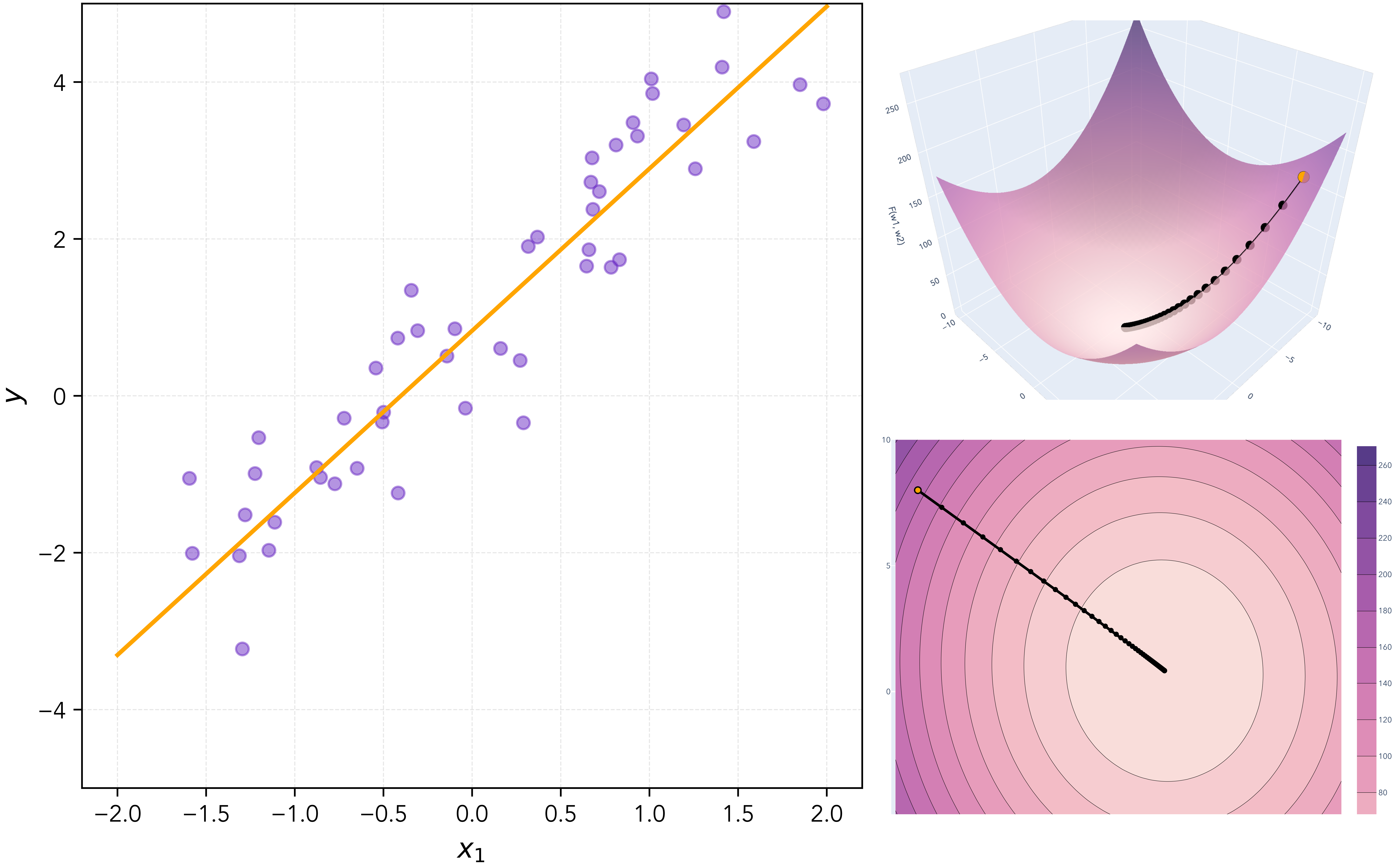
10
5
0
260
240
220
200
180
160
140
120
100
80

$T = 25$



y
4
2
0
−2
−4
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_1
F(w1, w2)
250
200
150
100
50
0
−10
−5
0
−10
−5
0
10
5
0
260
240
220
200
180
160
140
120
100
80

$T = 30$



4
2
0
−2
−4
y
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
2.0
x_1
F(w1, w2)
250
200
150
100
50
0
−10
−5
0
−10
−5
0
10
5
0
260
240
220
200
180
160
140
120
100
80

$T = 100$

A candidate algorithm

Moving in steepest descent direction

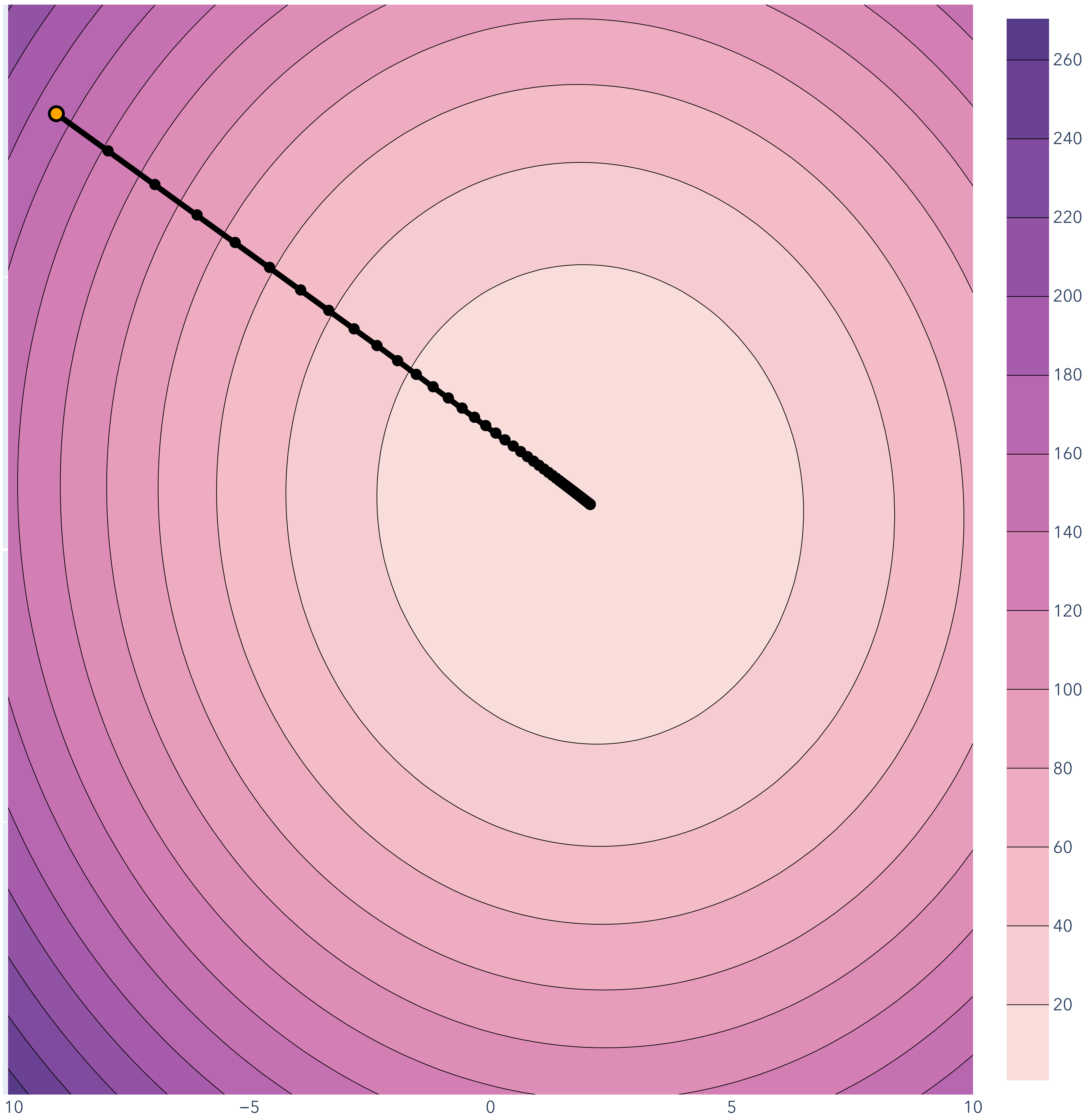
Start at some arbitrary point $w^{(0)} \in \mathbb{R}^d$.

Step in the **direction of steepest decrease** for $F(w)$…

Take another step in the **direction of steepest decrease** for $F(w)$…

$\vdots$

Repeat until satisfied.



Outline

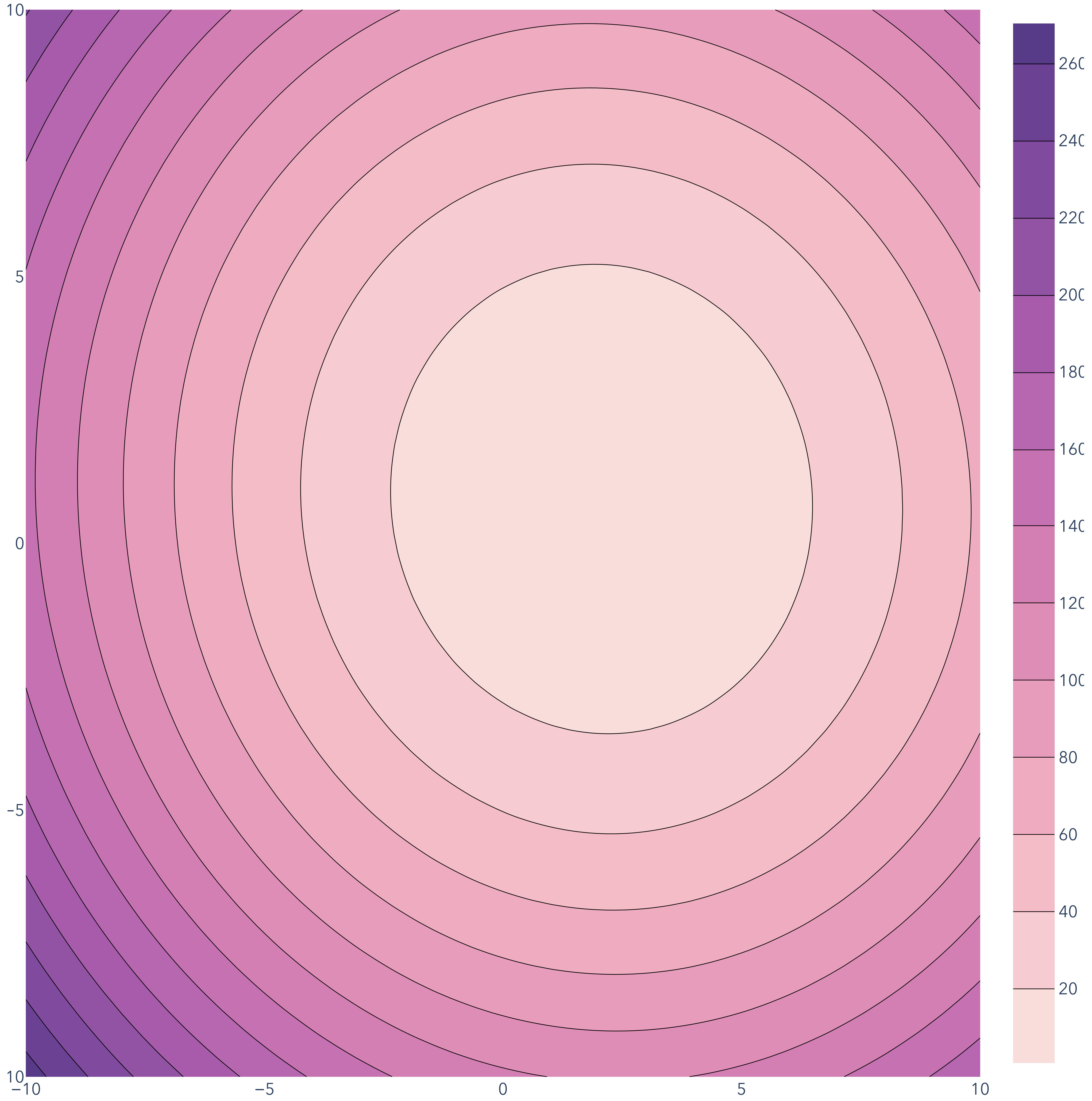
Gradient

Review

The gradient of F at $u \in \mathbb{R}^d$ is a vector $\nabla F(u) \in \mathbb{R}^d$:

$$\nabla F(w_0) := \left(\frac{\partial F}{\partial w_1}(u), \ldots, \frac{\partial F}{\partial w_d}(u) \right)$$

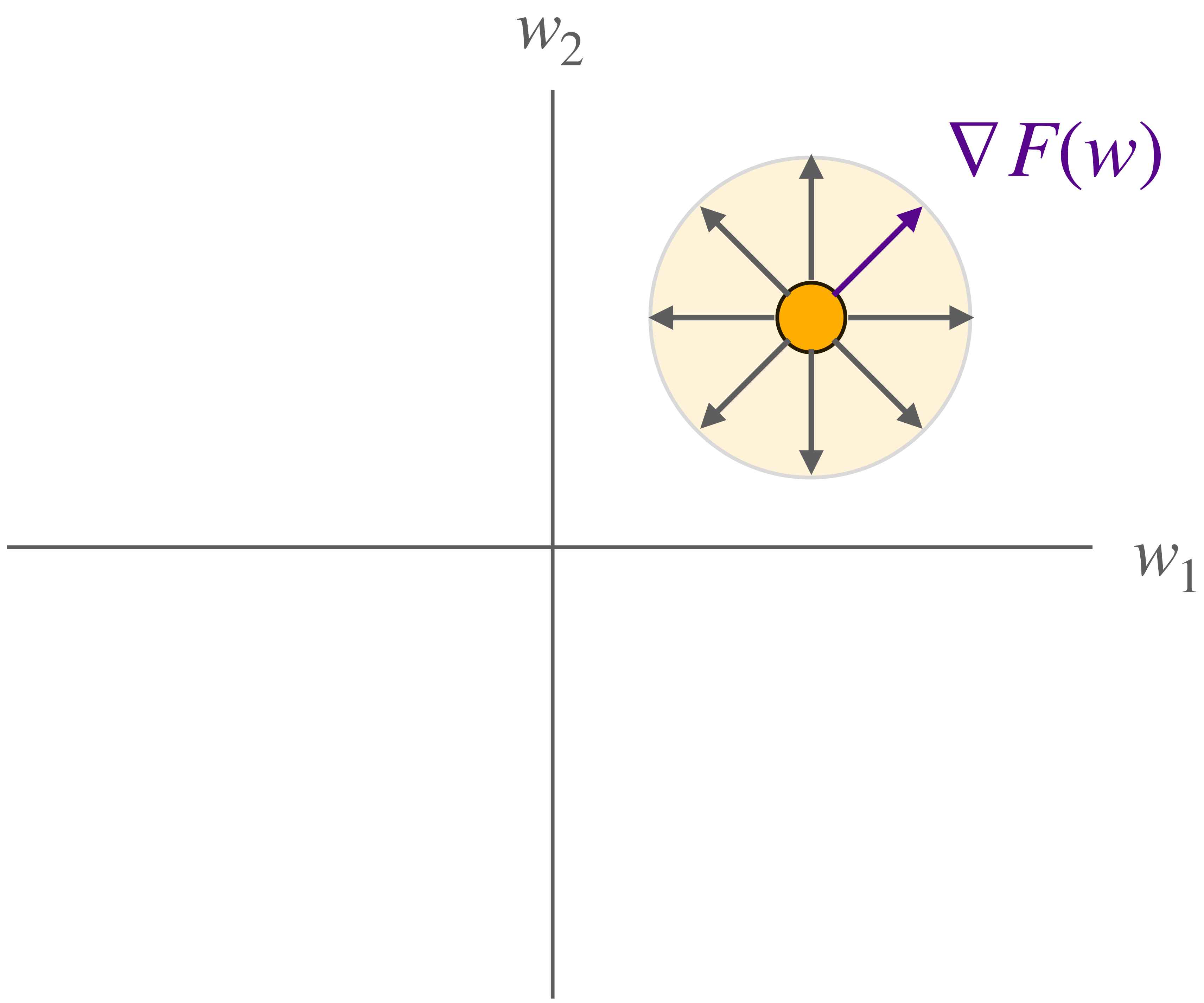
It is the direction F increases the fastest at a fixed point u.



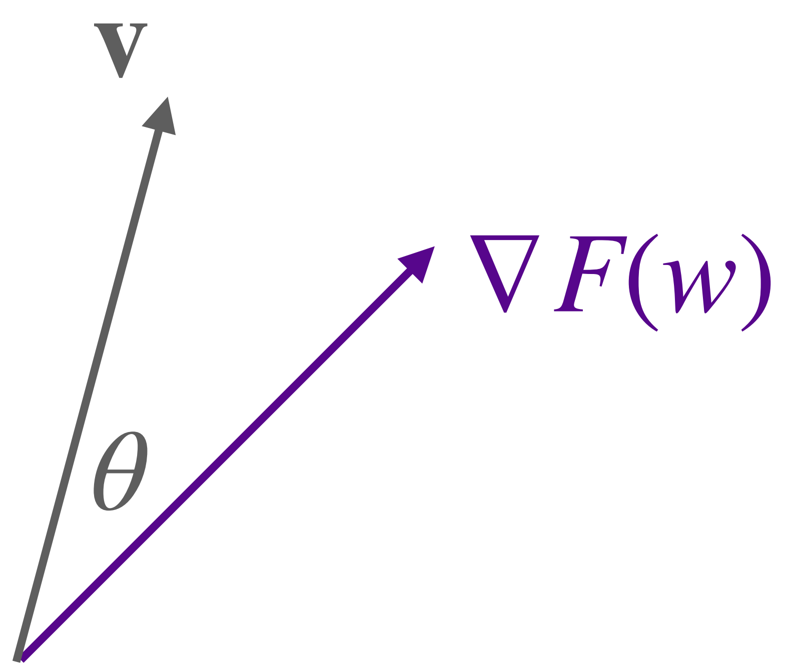
Gradient

The direction of steepest ascent (Why?)

Steepest increase direction?



Recall: directional derivative is the rate of change of F in direction $\mathbf{v} \in \mathbb{R}^d$



$$v^\top \nabla F(w) = \|v\| \|\nabla F(w)\| \cos\theta$$

$$\|v\| = 1$$

Maximized when $\theta = 0$,
i.e. when v is exactly in $\nabla F(w)$ direction!

Gradient Descent

Algorithm

Initialize at a randomly chosen $w^{(0)} \in \mathbb{R}^d$.

For iteration $t = 1,2,\ldots$ (until "stopping condition" is satisfied):

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

Return final $w^{(t)}$, with objective value $F(w^{(t)})$.

Gradient Descent

Stopping Condition

For iteration $t = 1,2,\ldots$ (until "stopping condition" is satisfied):

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

Typically:

Until $\|\nabla f(w^{(t)})\| \leq \epsilon$ (recall: $\nabla f(w) = 0$ at a minimum).

In practice, with validation data, can implement early stopping:

Evaluate performance on validation as you go, stop when no longer improving.

Gradient Descent

Step Size

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

The step size/learning rate of gradient descent is a positive number $\eta > 0$.

A fixed step size will work as long as it is *small enough*.

η too large: optimization might diverge.

η too small: optimization might take a long time.

In practice, can make sense to try several fixed step sizes or decaying step sizes η_t.

What properties of F relate to how large/small a step to take?

Differential Calculus

Review: Derivative

If $F : \mathbb{R}^d \to \mathbb{R}$ is *differentiable*, then for any $u \in \mathbb{R}^d$,

Linear approximation of F at point u.

$$\lim_{w \to u} \frac{F(w) - (F(u) + \langle \nabla F(u), w - u \rangle)}{\|w - u\|} = 0$$

At any point $u \in \mathbb{R}^d$, $F(w) \approx F(u) + \langle \nabla F(u), w - u \rangle$ for all w close to u.

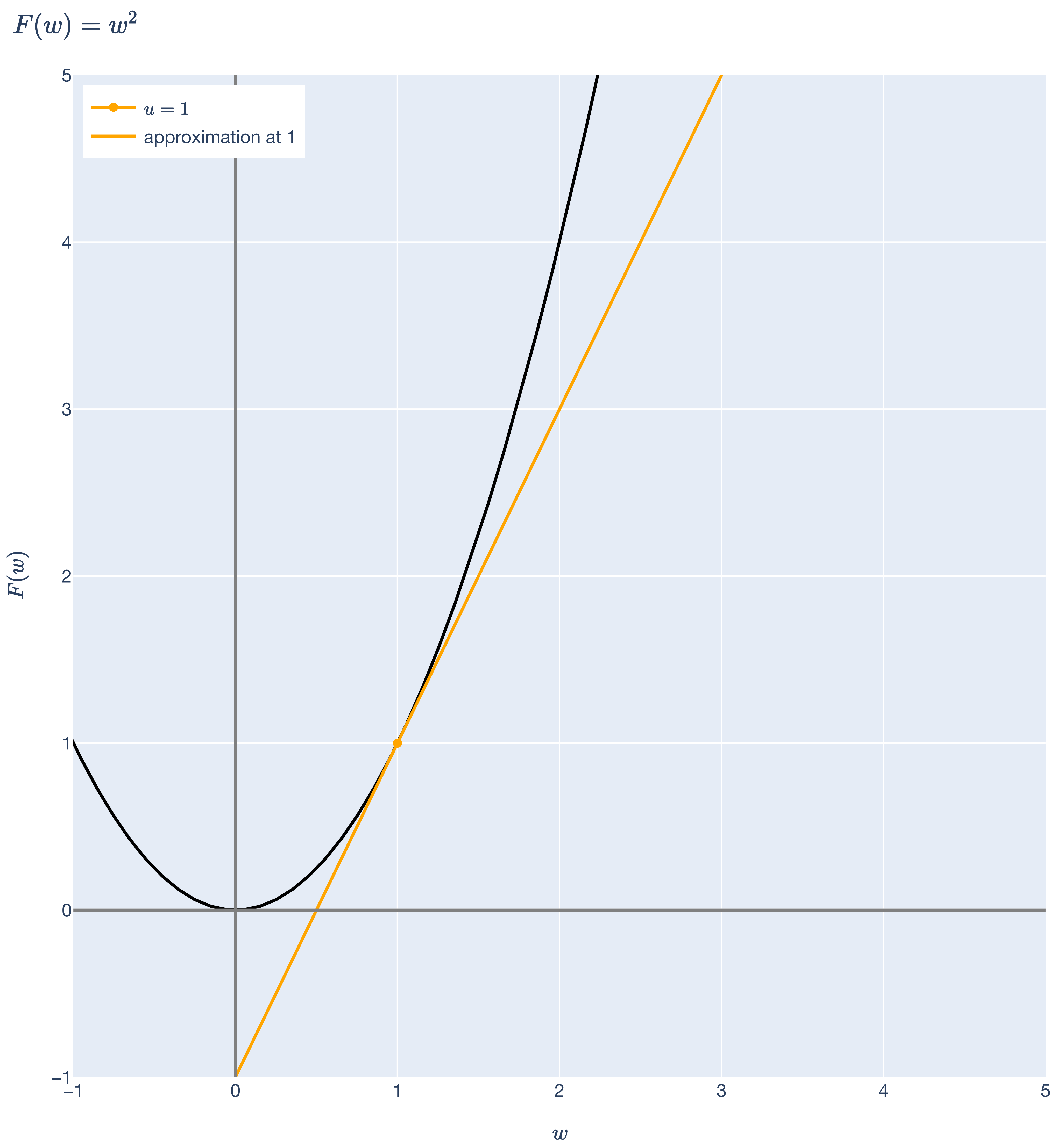
Differential Calculus

Review: Derivative

If $F : \mathbb{R}^d \to \mathbb{R}$ is *differentiable*, then for any $u \in \mathbb{R}^d$,

$$\lim_{w \to u} \frac{F(w) - (F(u) + \langle \nabla F(u), w - u \rangle)}{\|w - u\|} = 0$$

At any point $u \in \mathbb{R}^d$,
$F(w) \approx F(u) + \langle \nabla F(u), w - u \rangle$ *for all* w
close to u.



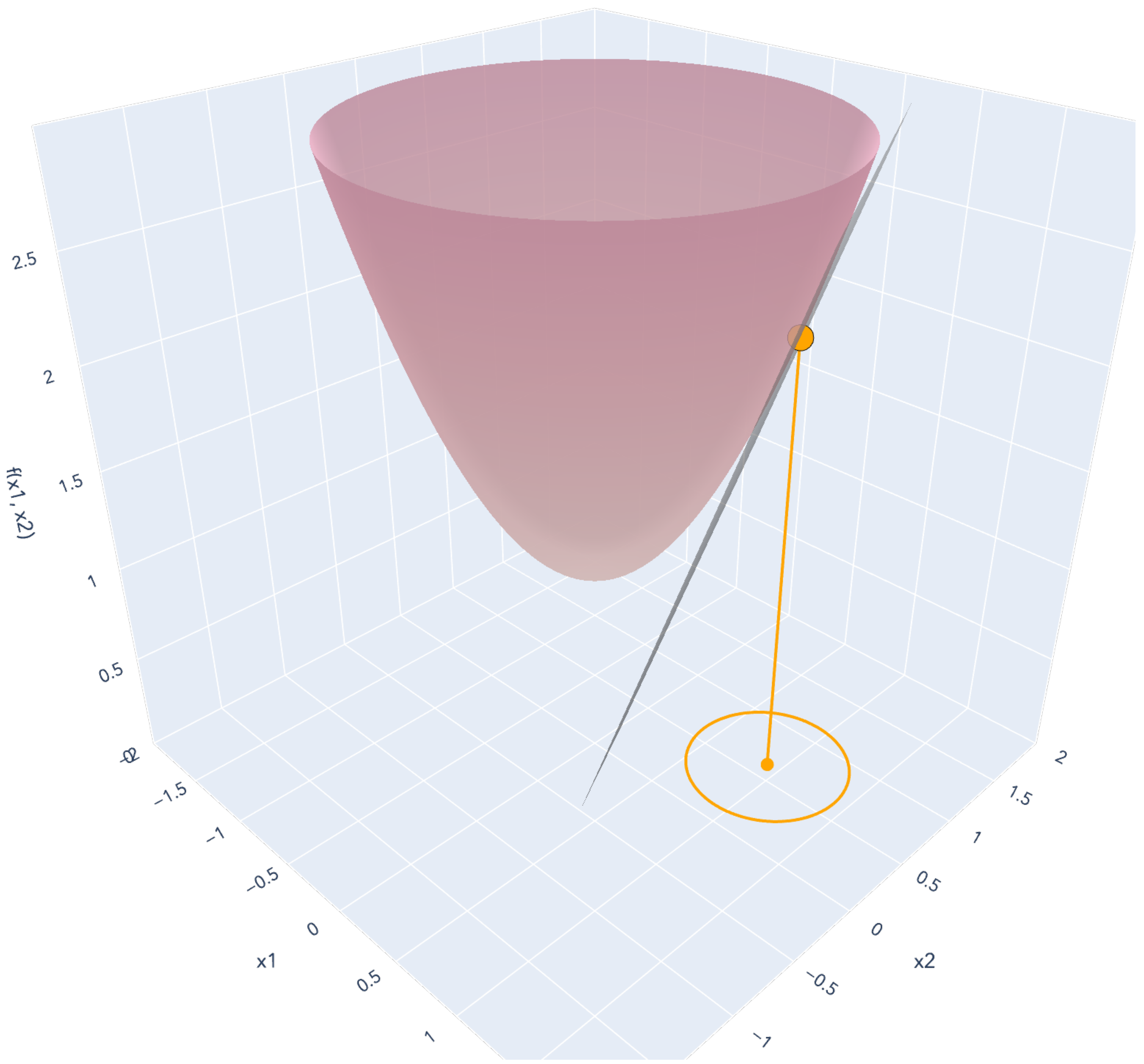
Differential Calculus

Review: Derivative

If $F : \mathbb{R}^d \to \mathbb{R}$ is *differentiable*, then for any $u \in \mathbb{R}^d$,

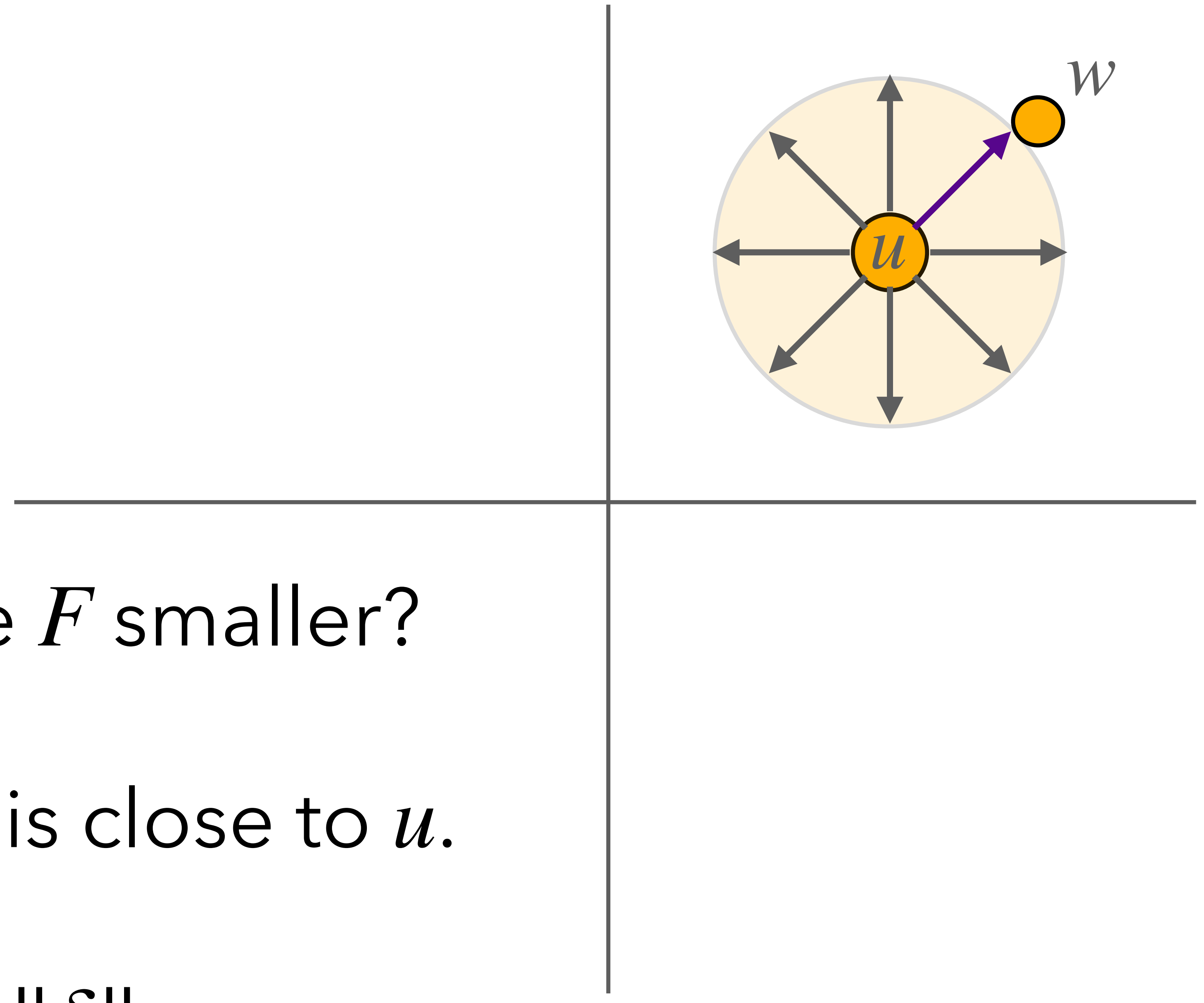
$$\lim_{w \to u} \frac{F(w) - (F(u) + \langle \nabla F(u), w - u \rangle)}{\|w - u\|} = 0$$

At any point $u \in \mathbb{R}^d$,
$F(w) \approx F(u) + \langle \nabla F(u), w - u \rangle$ for all w close to u.



Gradient Descent

Rough Derivation



Given $u \in \mathbb{R}^d$ with objective $F(u)$, how do we change u to make F smaller?

$$F(w) \approx F(u) + \nabla F(u)^\top (w - u), \text{ as long as } w \text{ is close to } u.$$

For any direction $\delta \in \mathbb{R}^d$ with small $\|\delta\|$:

$$F(u+\delta) \approx F(u) + \nabla F(u)^\top (u + \delta - u) \Longrightarrow F(u+\delta) \approx F(u) + \nabla F(u)^\top \delta$$

So, if $\delta = -\eta \nabla F(u)$, we should have:

$$F(u - \eta \nabla F(u)) \approx F(u) - \eta \|\nabla F(u)\|^2 \text{ as long as } \eta \text{ is small.}$$

Lipschitz & Smoothness

Definition

Lipschitzness: *"function doesn't change too much"*

A function $F : \mathbb{R}^d \to \mathbb{R}$ is a Lipschitz continuous with constant $L > 0$ if

$$\|F(x) - F(y)\| \leq L\|x - y\|.$$

A function $F : \mathbb{R}^d \to \mathbb{R}$ is a L-smooth if ∇F is Lipschitz continuous:

$$\|\nabla F(x) - \nabla F(y)\| \leq \|x - y\| \text{ for all } x, y.$$

If twice-differentiable, F is L-smooth if the eigenvalues of its Hessian are at most L.

$$\lambda_{\max}(\nabla^2 F(x)) \leq L.$$

Gradient Descent Guarantees

Theorem 1: Descent Lemma

Theorem (Descent Lemma).

If F is "smooth enough," then there is a choice of $\eta > 0$ such that, for any $w \in \mathbb{R}^d$,

$$F(w - \eta \nabla F(w)) \leq F(w) - \frac{\eta}{2} \|\nabla F(w)\|^2.$$

"Smooth enough" : F is an L-smooth function.

Taylor's Theorem: makes the $\approx$ rigorous!

Gradient Descent Guarantees

Theorem 1: Descent Lemma

Theorem (Descent Lemma).

If F is continuously twice-differentiable and L-smooth for any $w \in \mathbb{R}^d$,

$$F(w - \eta \nabla F(w)) \leq F(w) - \frac{\eta}{2} \|\nabla F(w)\|^2$$

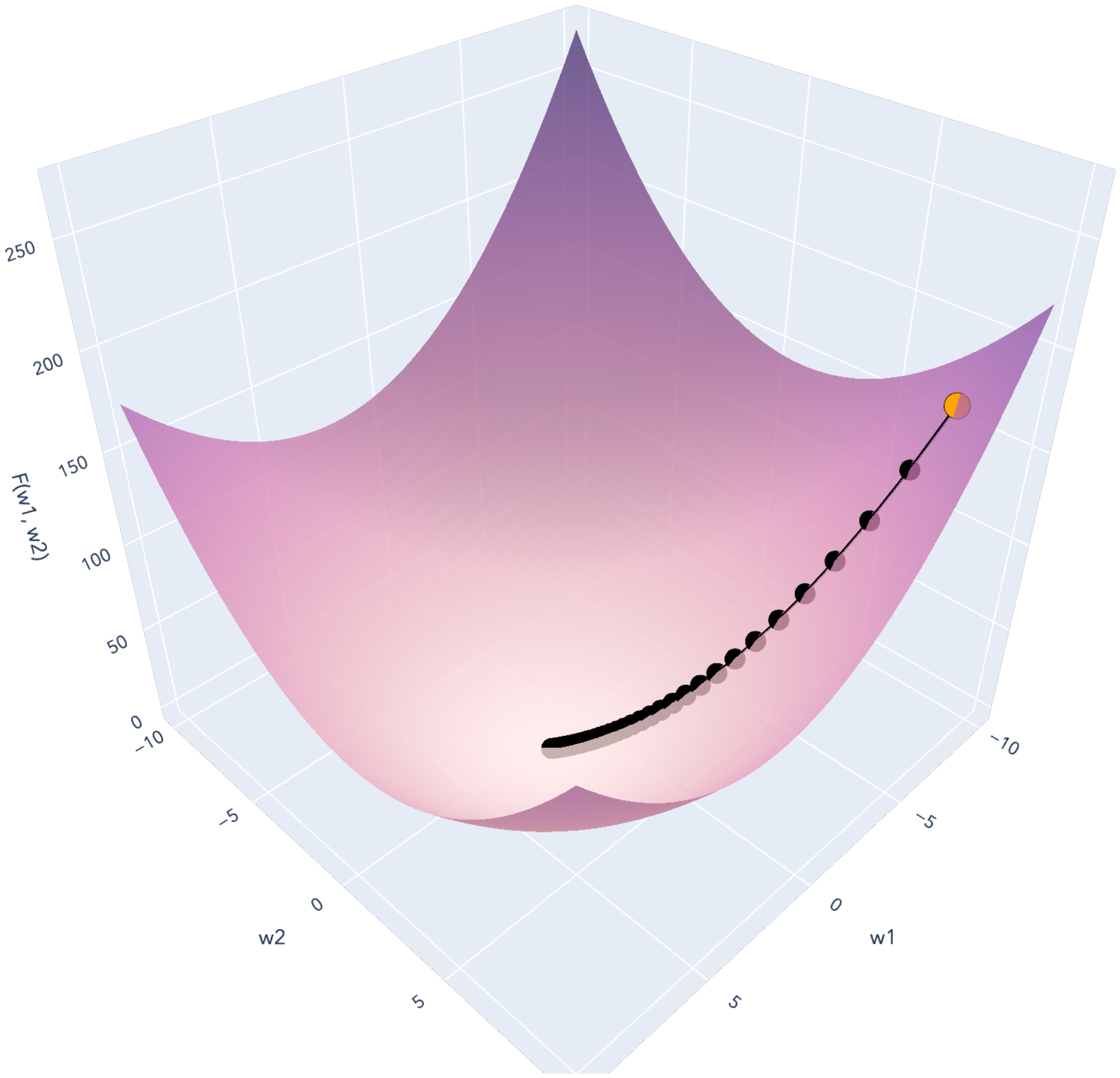
when $\eta \leq 1/L$.

Gradient Descent

Example: Linear Regression

$$\min_{w \in \mathbb{R}^d} \frac{1}{n} \|Xw - y\|^2$$

where $X \in \mathbb{R}^{n \times d}$, $y \in \mathbb{R}^n$



Gradient descent:

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \cdot \frac{2}{n} X^\top (Xw - y)$$

Gradient Descent

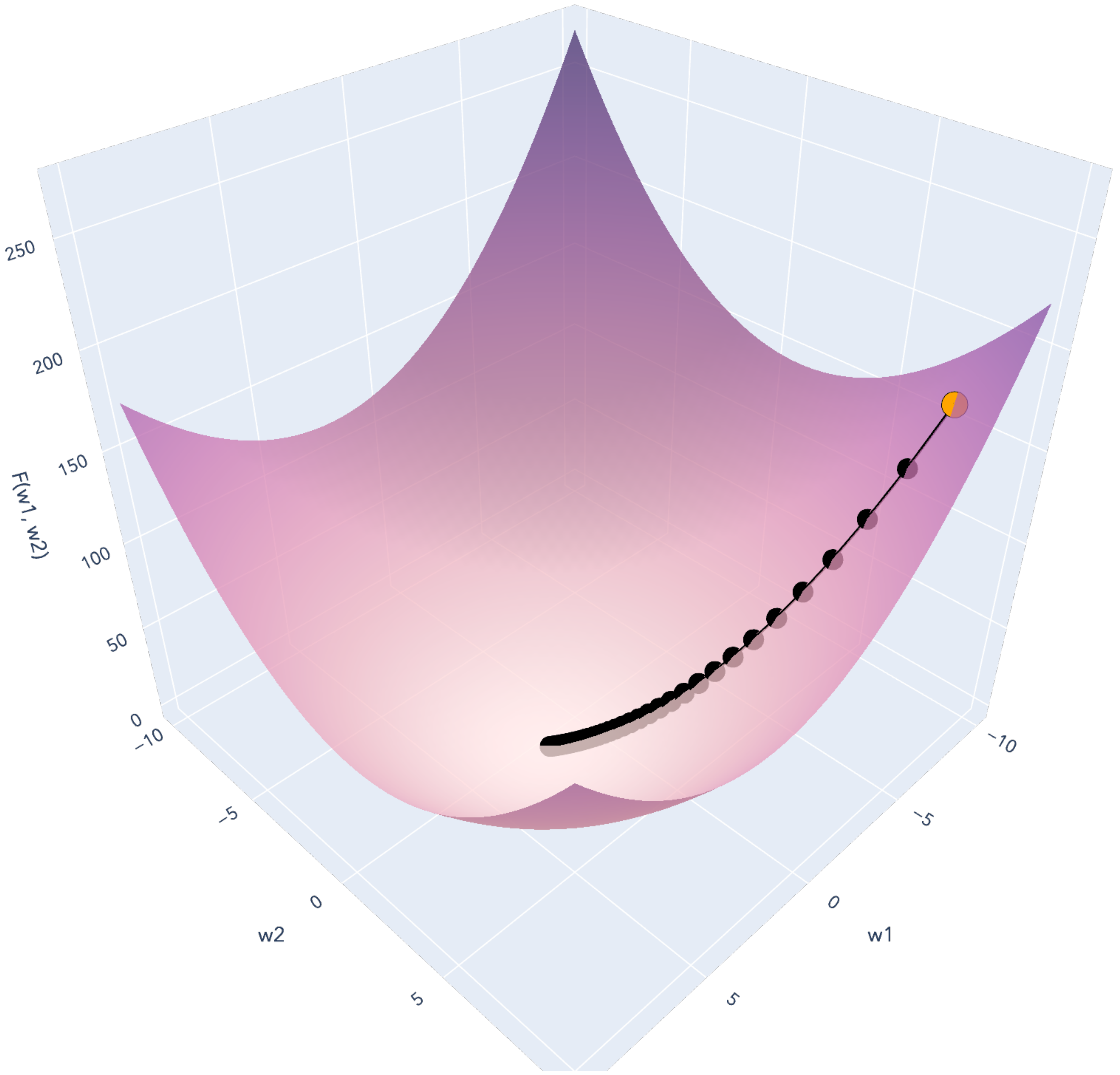
Example: Linear Regression

Initialize at a randomly chosen $w^{(0)} \in \mathbb{R}^d$.

For iteration $t = 1,2,\ldots,T$:

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \cdot \frac{2}{n} X^\top (Xw - y)$$

Return final $w^{(T)}$.



Gradient Descent

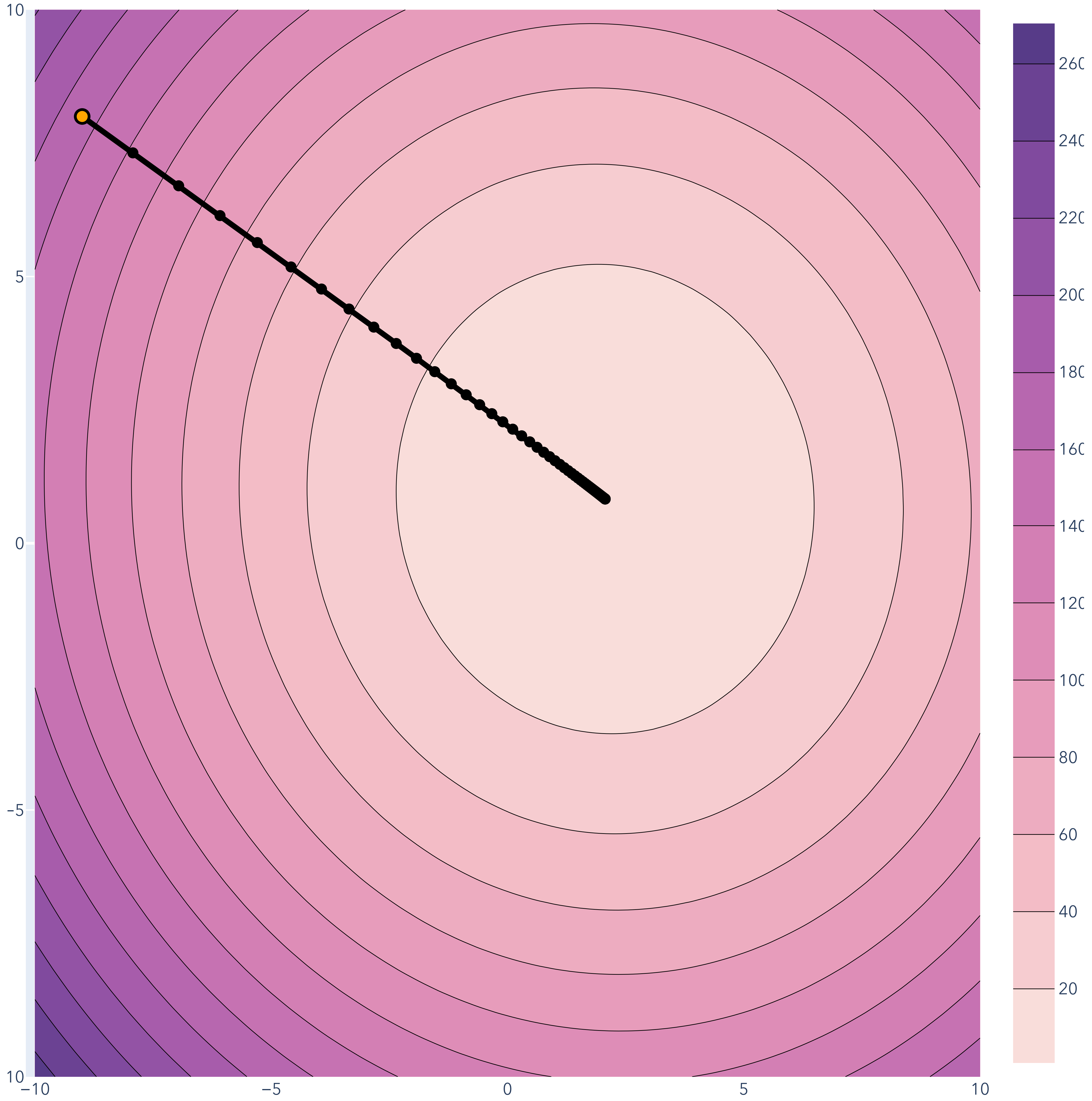
Example: Linear Regression

Initialize at a randomly chosen $w^{(0)} \in \mathbb{R}^d$.

For iteration $t = 1,2,\ldots,T$:

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \cdot \frac{2}{n} X^\top (Xw - y)$$

Return final $w^{(T)}$.



Descent Lemma

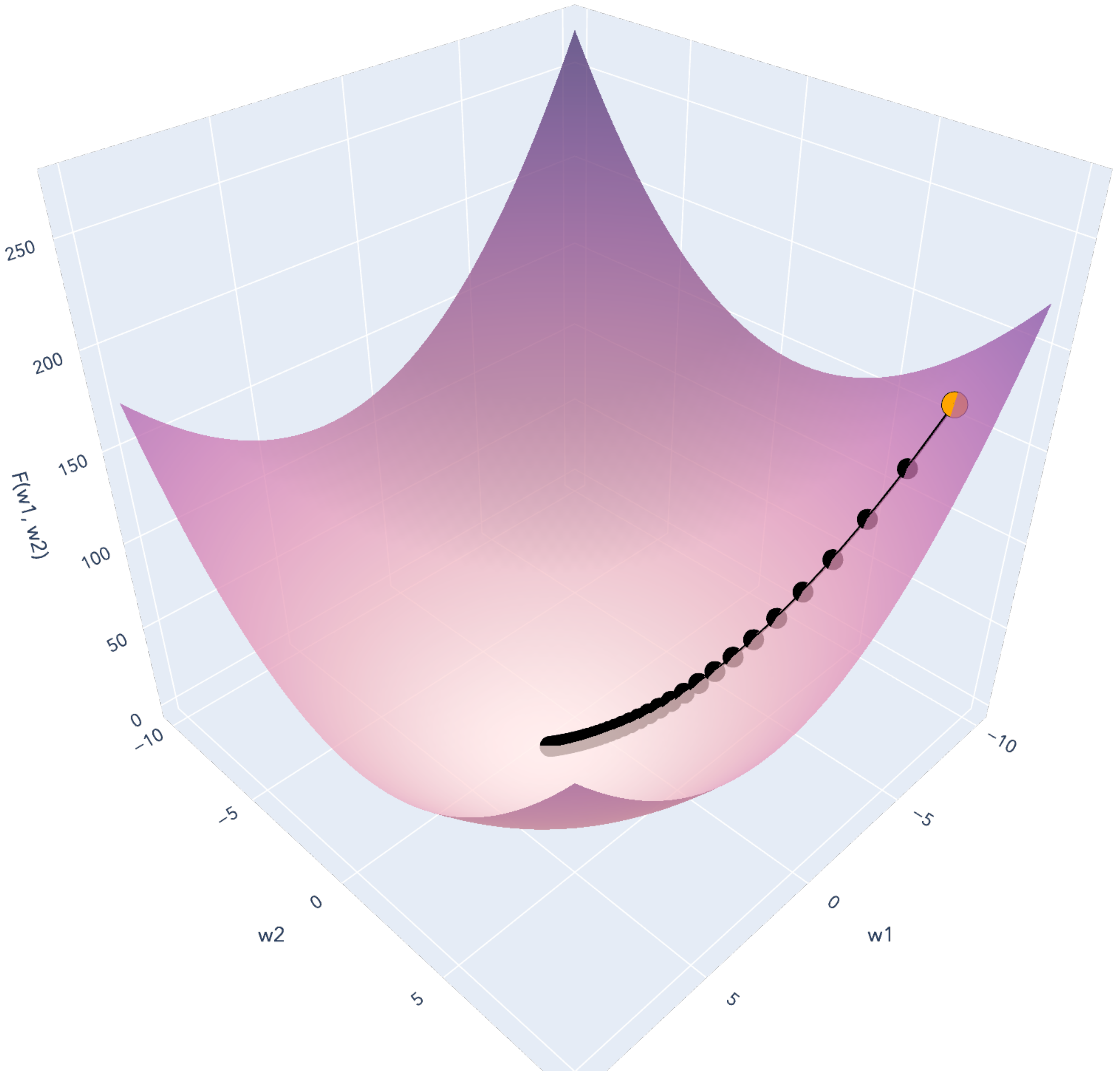
Guarantee (Informal)

If η is small enough, then the gradient descent update rule

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

has the property:

$$F(w^{(t)}) \lessapprox F(w^{(t-1)}) - \eta \|\nabla F(w^{(t-1)})\|^2.$$



Descent Lemma

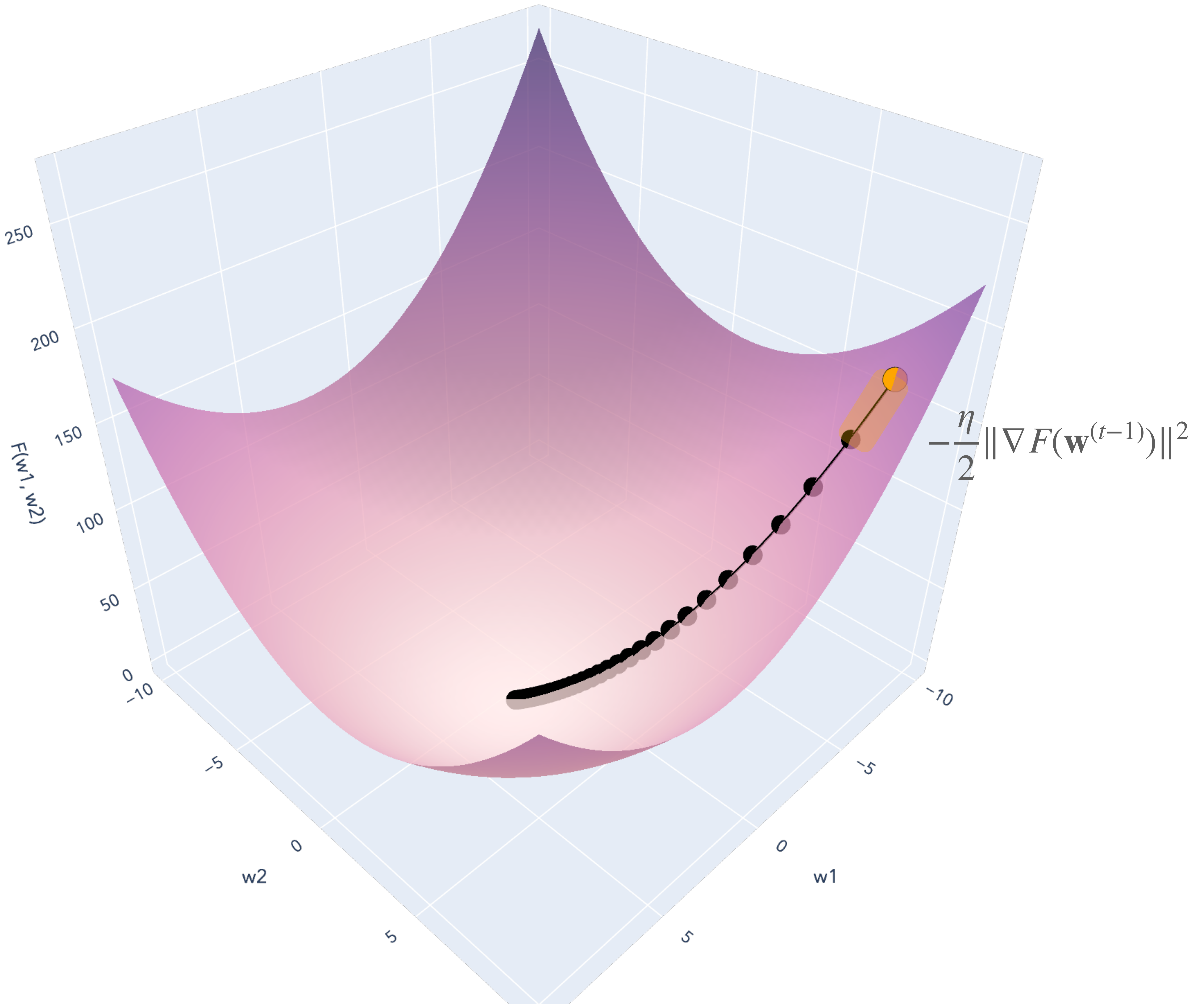
Guarantee (Informal)

If η is small enough, then the gradient descent update rule

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

has the property:

$$F(w^{(t)}) \lessapprox F(w^{(t-1)}) - \eta \|\nabla F(w^{(t-1)})\|^2.$$



Descent Lemma

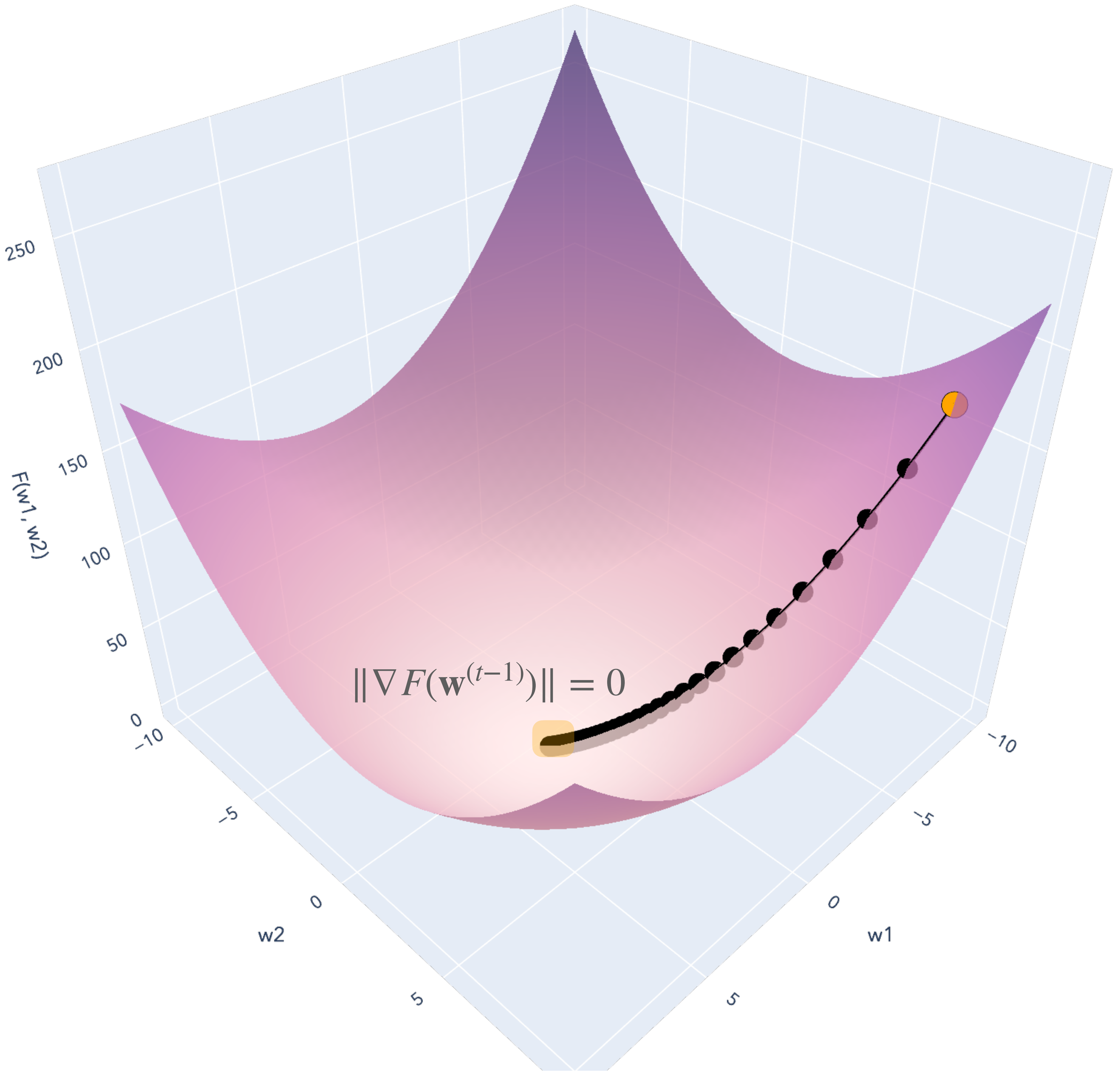
Guarantee (Informal)

If η is small enough, then the gradient descent update rule

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

has the property:

$$F(w^{(t)}) \lessapprox F(w^{(t-1)}) - \eta \|\nabla F(w^{(t-1)})\|^2.$$



Descent Lemma

Guarantee (Informal)

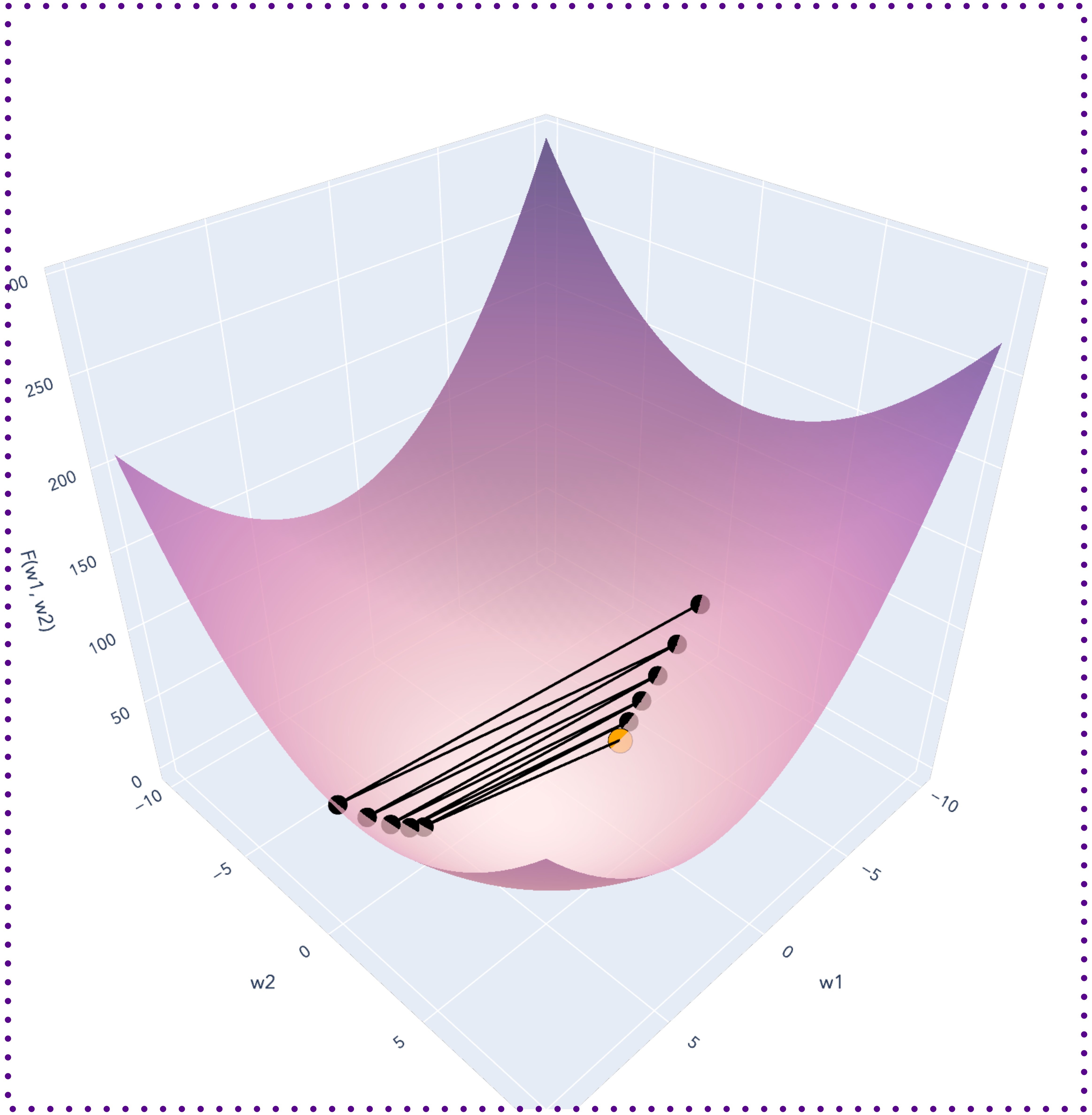
If η is small enough, then the gradient descent update rule

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

has the property:

$$F(w^{(t)}) \lessapprox F(w^{(t-1)}) - \eta \|\nabla F(w^{(t-1)})\|^2.$$

When η is too large, all bets are off – gradient descent may **diverge**!



Descent Lemma

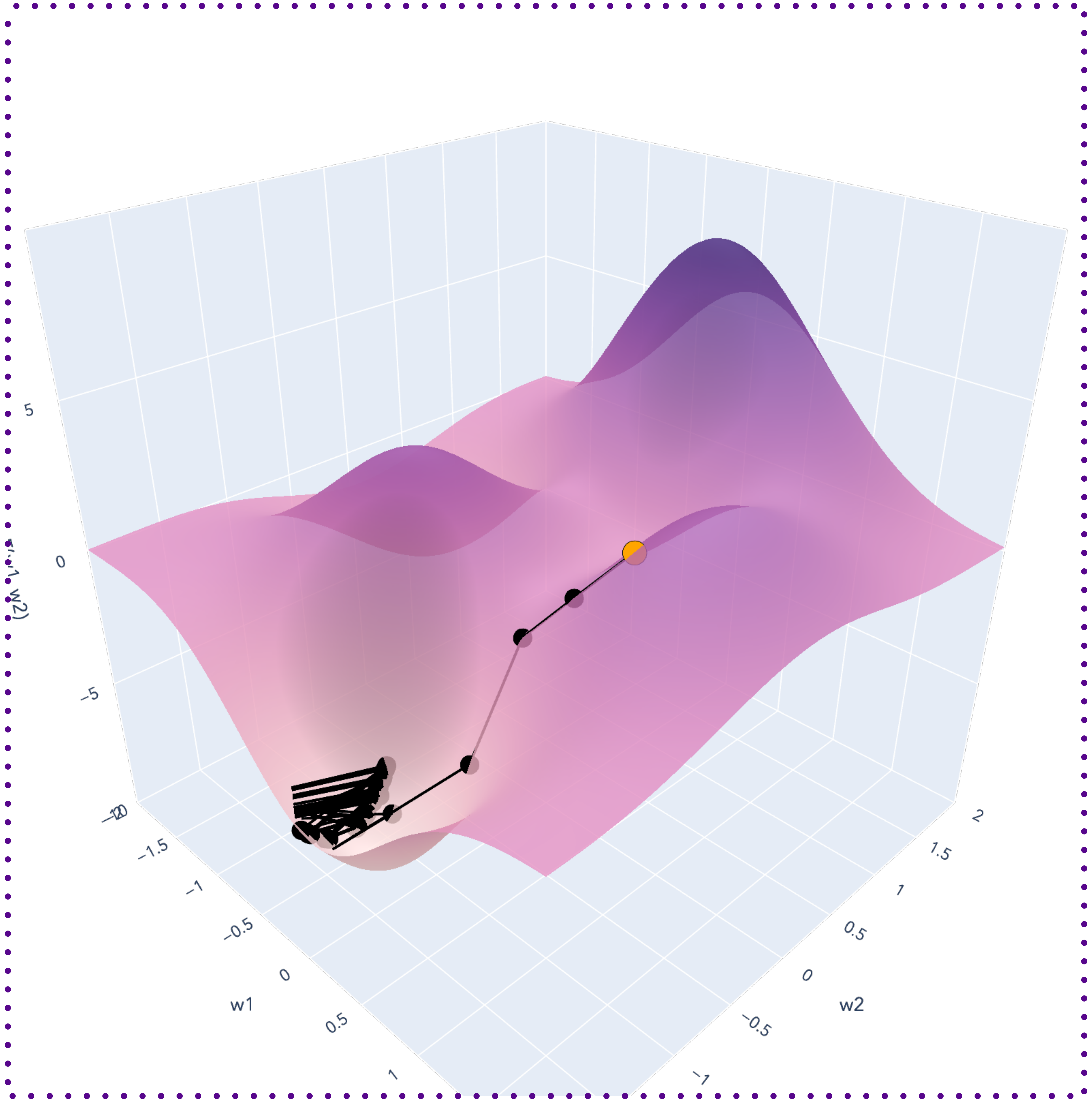
Guarantee (Informal)

If η is small enough, then the gradient descent update rule

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

has the property:

$$F(w^{(t)}) \lessapprox F(w^{(t-1)}) - \eta \|\nabla F(w^{(t-1)})\|^2.$$



Descent Lemma

Guarantee (Informal)

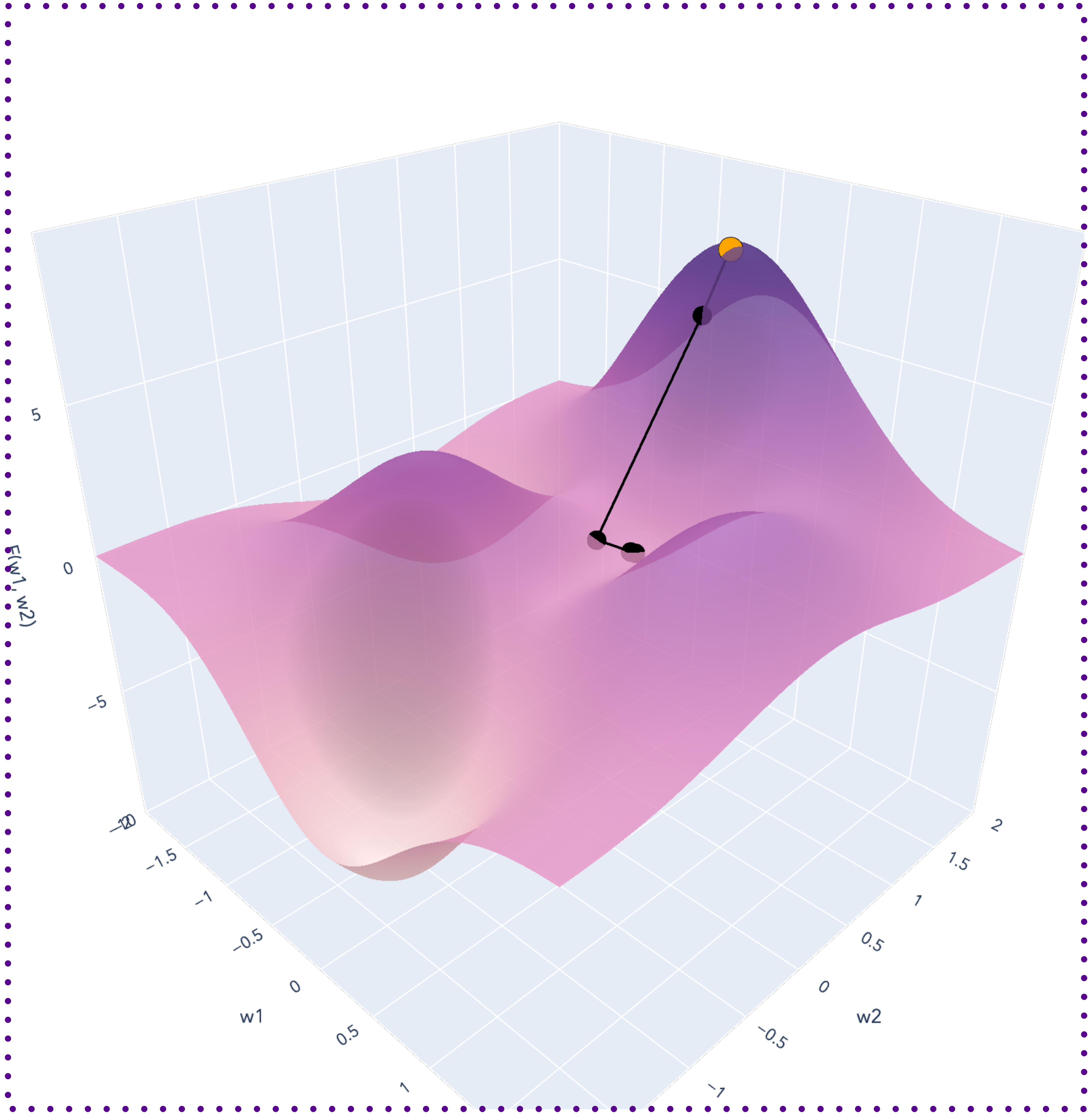
If η is small enough, then the gradient descent update rule

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

has the property:

$$F(w^{(t)}) \lessapprox F(w^{(t-1)}) - \eta \|\nabla F(w^{(t-1)})\|^2.$$

But this can mean getting stuck in a local minimum!



Descent Lemma

Guarantee (Informal)

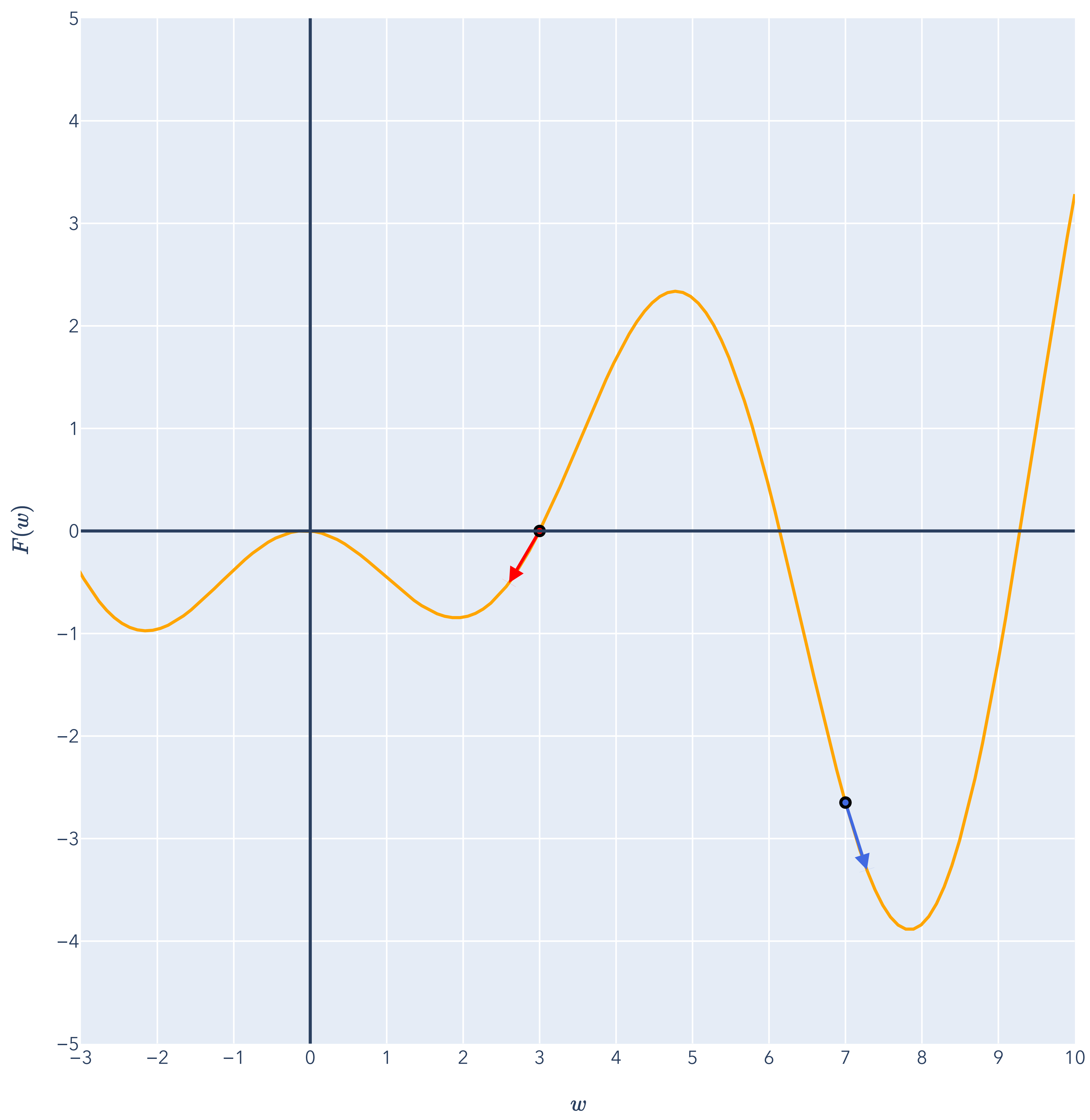
If η is small enough, then the gradient descent update rule

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla F(w^{(t-1)})$$

has the property:

$$F(w^{(t)}) \lessapprox F(w^{(t-1)}) - \eta \|\nabla F(w^{(t-1)})\|^2.$$

But this can mean getting stuck in a local minimum!



Outline

ERM: Learning as Optimization

Optimizing Linear Regression: Closed Form

Gradient Descent Intuition & Example

Gradient Descent Algorithm & Descent Lemma

Gradient Descent on Convex Functions

Stochastic Gradient Descent

Stationary Points

What can happen at $\nabla F(x) = 0$?

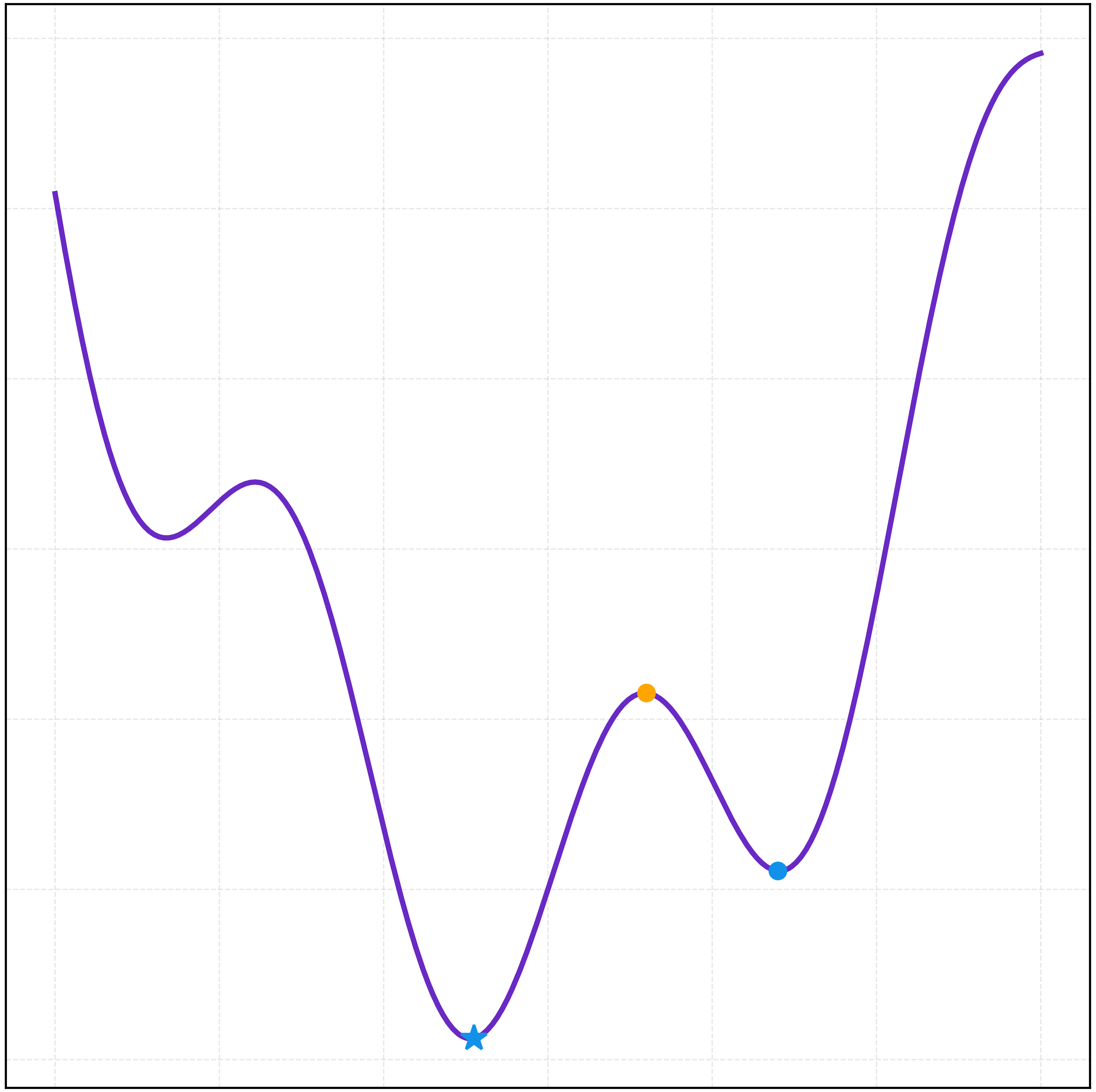
When $\nabla F(x) = 0$, you can have stationary points that are:

Local minima.

Local maxima.

Global maximum.

Global minimum.



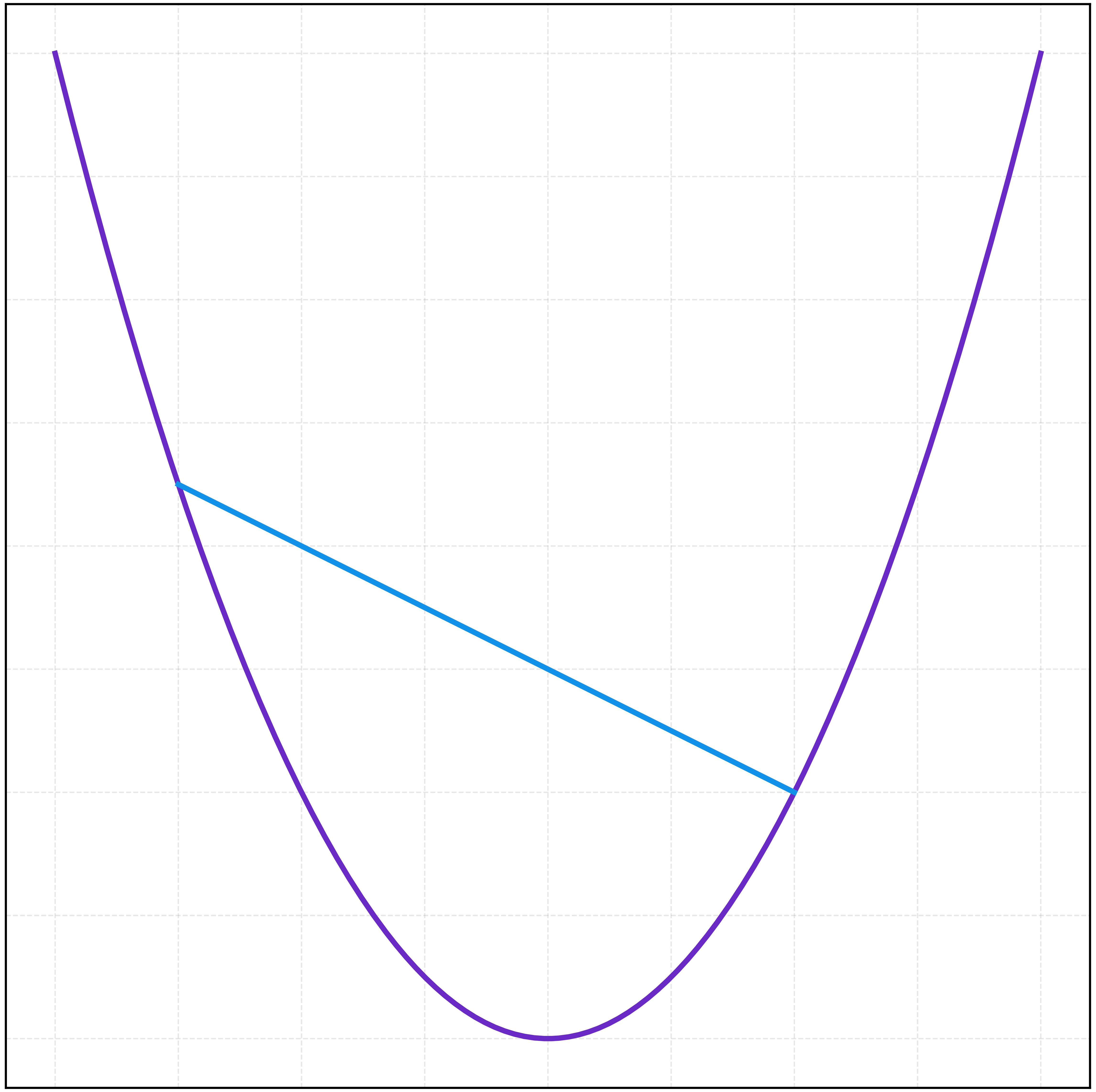
Convex Function

Intuition

A convex function is a function that is "bowl-shaped."

All line segments through any two points lie above the function.

If differentiable, all linear approximations lie below the function.



This function is convex.

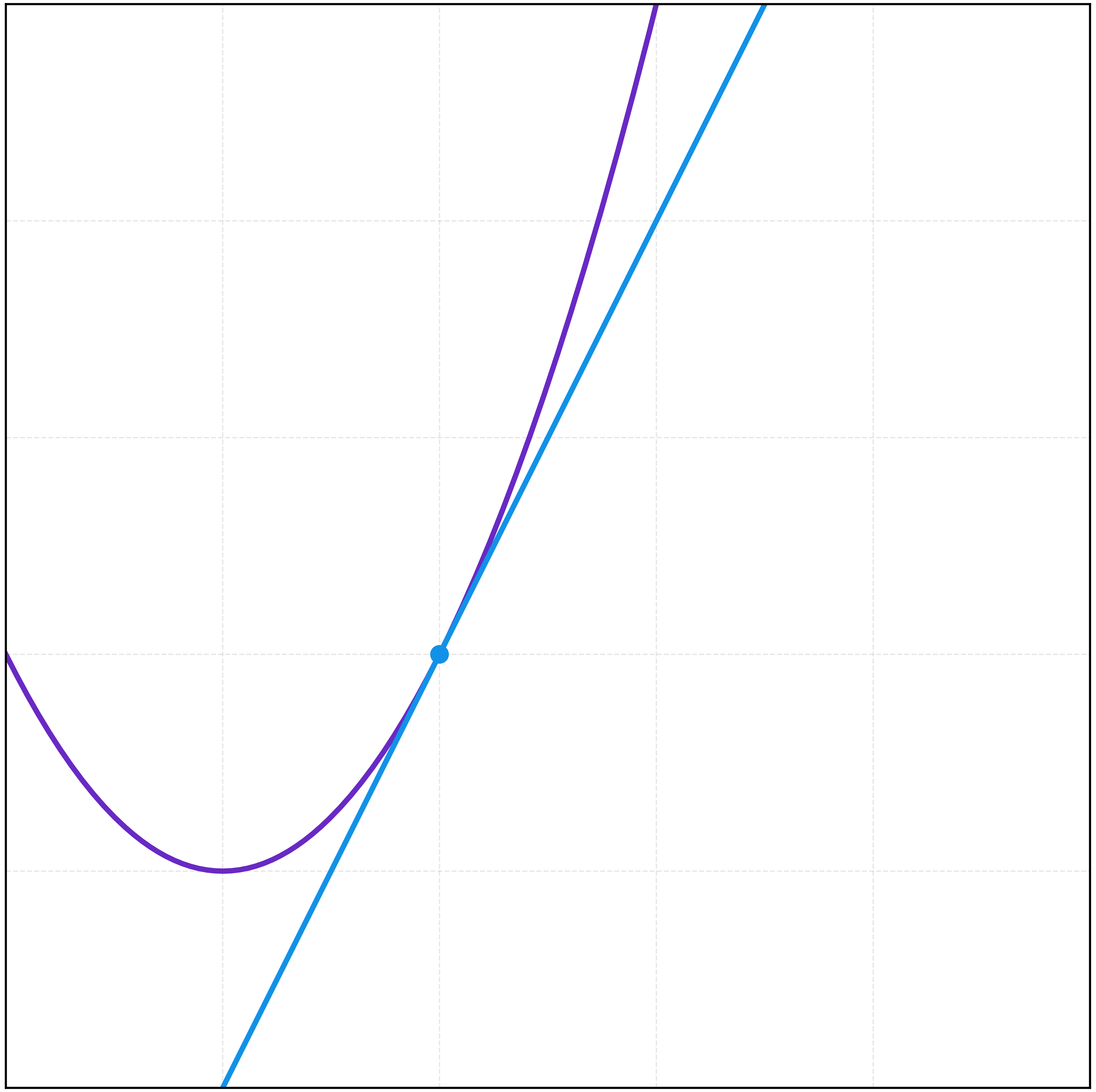
Convex Function

Intuition

A convex function is a function that is "bowl-shaped."

All line segments through any two points lie above the function.

If differentiable, all linear approximations lie below the function.



This function is convex.

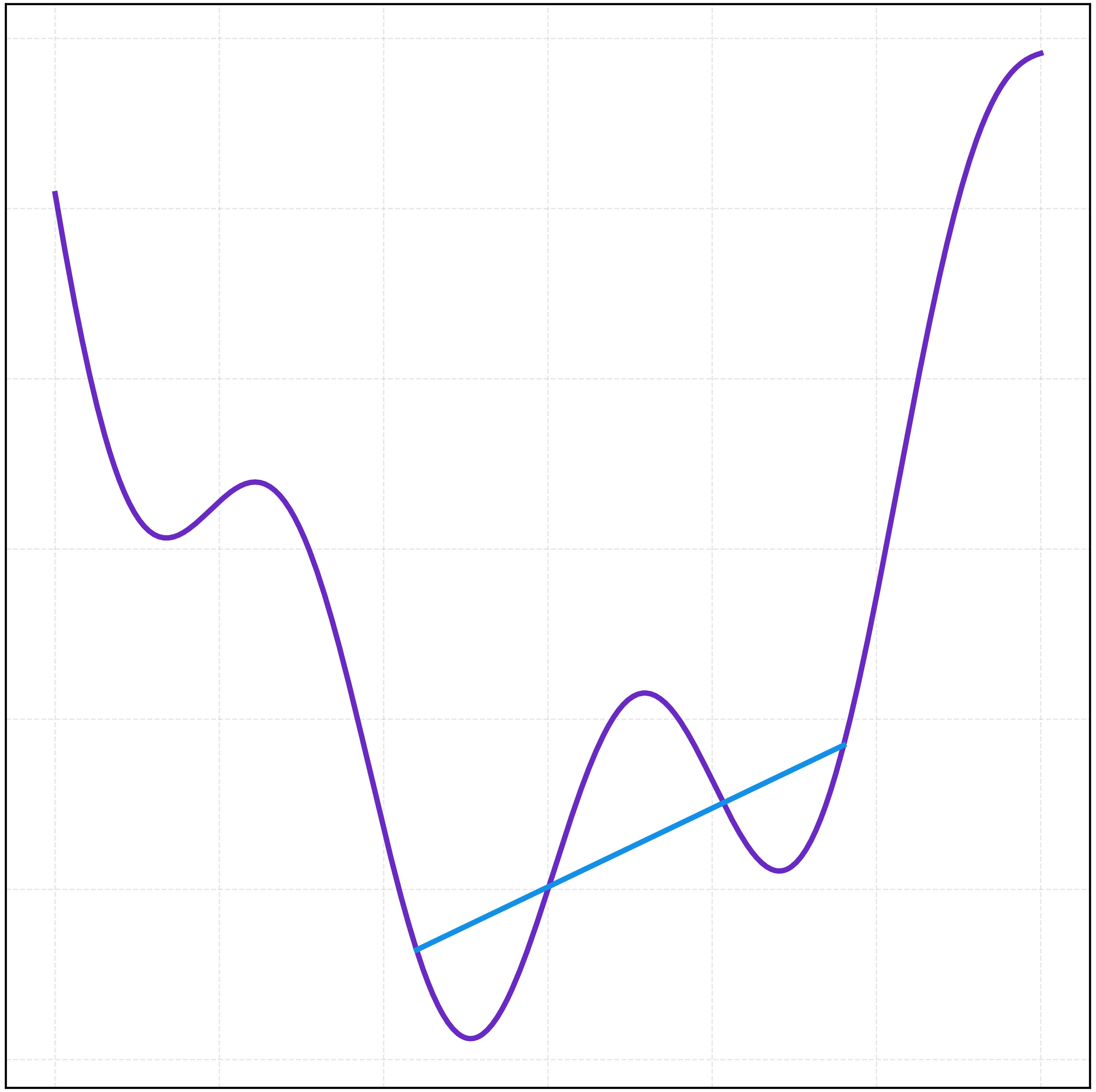
Convex Function

Intuition

A convex function is a function that is "bowl-shaped."

All line segments through any two points lie above the function.

If differentiable, all linear approximations lie below the function.



This function is NOT convex.

Gradient Descent Guarantee

Convex, Smooth Functions

Theorem (GD on Convex, Smooth Functions).

If $F : \mathbb{R}^d \to \mathbb{R}$ is convex, differentiable, and L-smooth, then gradient descent with $\eta \leq 1/L$ converges:

$$F(w^{(T)}) - F(w^*) \leq \frac{\|w^{(0)} - w^*\|^2}{2\eta T} \text{ after } T \text{ steps.}$$

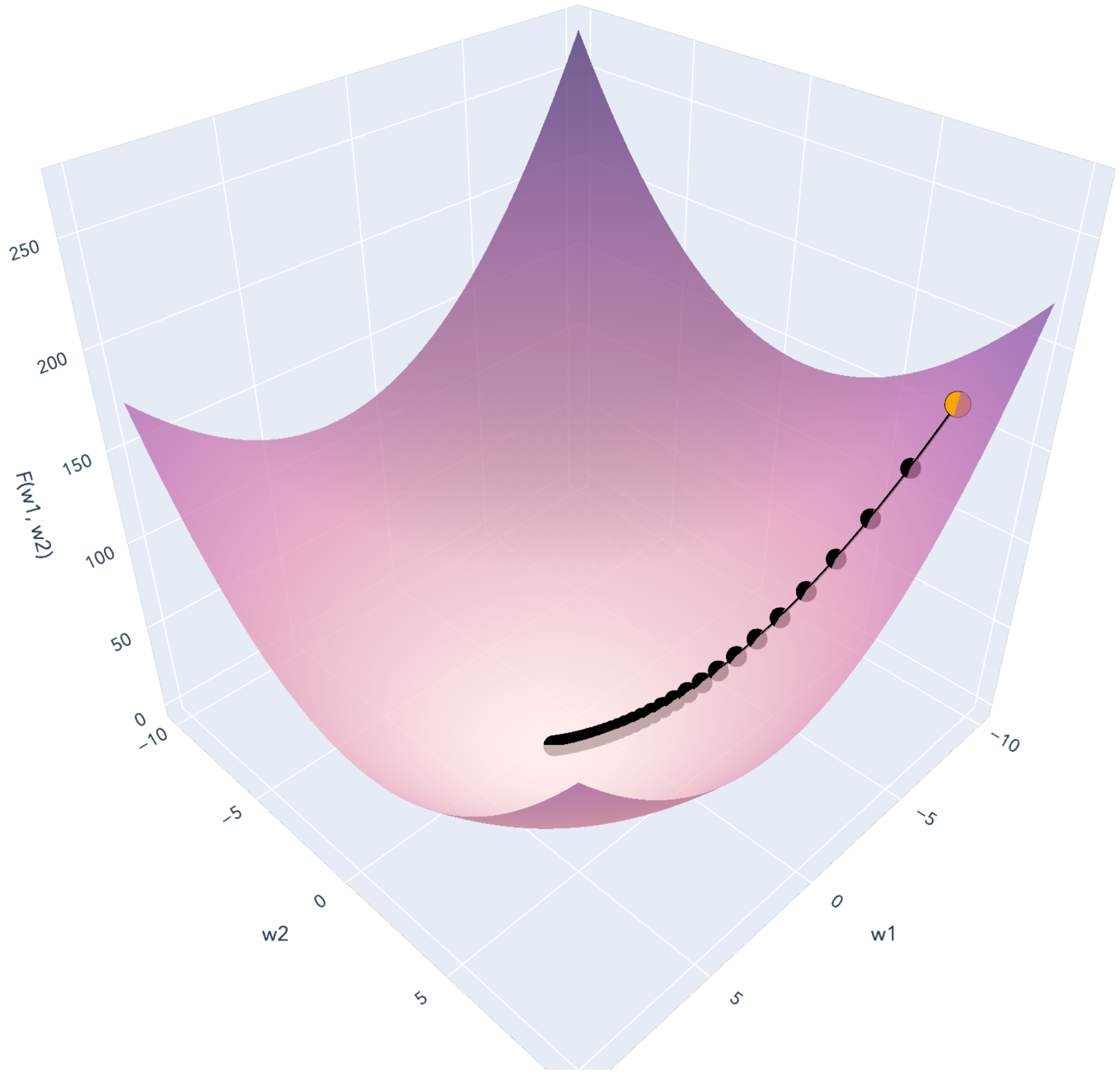
Gradient Descent

Example: Least Squares Regression

Theorem. If $F : \mathbb{R}^d \to \mathbb{R}$ is convex, differentiable, and L-smooth, then gradient descent with $\eta \leq 1/L$ converges:

$$F(w^{(T)}) - F(w^*) \leq \frac{\|w^{(0)} - w^*\|^2}{2\eta T} \text{ after } T \text{ steps.}$$

The "classical ML" part of this course will mainly be concerned with *convex* objectives, where we have nice guarantees about optimization.



Notes on Convergence

Step Size

...gradient descent with $\eta \leq 1/L$ converges.

Fixed step size works as long as it is small enough.

No guarantees (may diverge) if step sizes are too big.

Intuition from theorem: allowable step sizes are sensitive to the “change in the derivative.”

$F : \mathbb{R}^d \rightarrow \mathbb{R}$ is a L-smooth if ∇F is Lipschitz continuous:

$\|\nabla F(x) - \nabla F(y)\| \leq \|x - y\|$ for all x, y.

Notes on Computation

Scalability Issues

Recall our main problem of empirical risk minimization (ERM):

$$\min_{h \in \mathscr{H}} \frac{1}{n} \sum_{i=1}^{n} \ell(h(x^{(i)}), y^{(i)}).$$

Hypothesis Class: $\mathscr{H} = \{h_w : \mathscr{X} \to \mathscr{Y} : w \in \mathbb{R}^d\}$ (e.g. linear functions)

Given dataset $D_n := \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$ we want to minimize the empirical risk:

$$\hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \ell(h_w(x^{(i)}), y^{(i)})$$

Notes on Computation

Scalability Issues

$$\hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \ell(h_w(x^{(i)}), y^{(i)})$$

If $\ell(h_w(x^{(i)}), y^{(i)})$ is differentiable as function of w, we can do gradient descent on $\hat{R}_n(w)$.

$$\text{Example: } \hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} (w^\top x^{(i)} - y^{(i)})^2.$$

Need to iterate over all n training points each step!

At every step, we need to compute the gradient at the current w:

$$\nabla \hat{R}_n(w) = \frac{1}{n}\sum_{i=1}^{n} \nabla_w \ell(h_w(x^{(i)}), y^{(i)})$$

Outline

ERM: Learning as Optimization

Optimizing Linear Regression: Closed Form

Gradient Descent Intuition & Example

Gradient Descent Algorithm & Descent Lemma

Gradient Descent on Convex Functions

Stochastic Gradient Descent

Stochastic Gradient Descent

Intuition

Issue: At every step, we need to compute the gradient at the current w:

$$\nabla \hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \ell(h_w(x^{(i)}), y^{(i)})$$

Claim: If we choose $j \in [n]$ uniformly at random, then:

$$\mathbb{E}_{j \sim [n]}[\nabla_w \ell(h_w(x^{(j)}), y^{(j)})] = \nabla \hat{R}_n(w).$$

The estimate $\nabla_w \ell(h_w(x^{(j)}), y^{(j)})$ is a stochastic gradient.

Stochastic GD

Algorithm

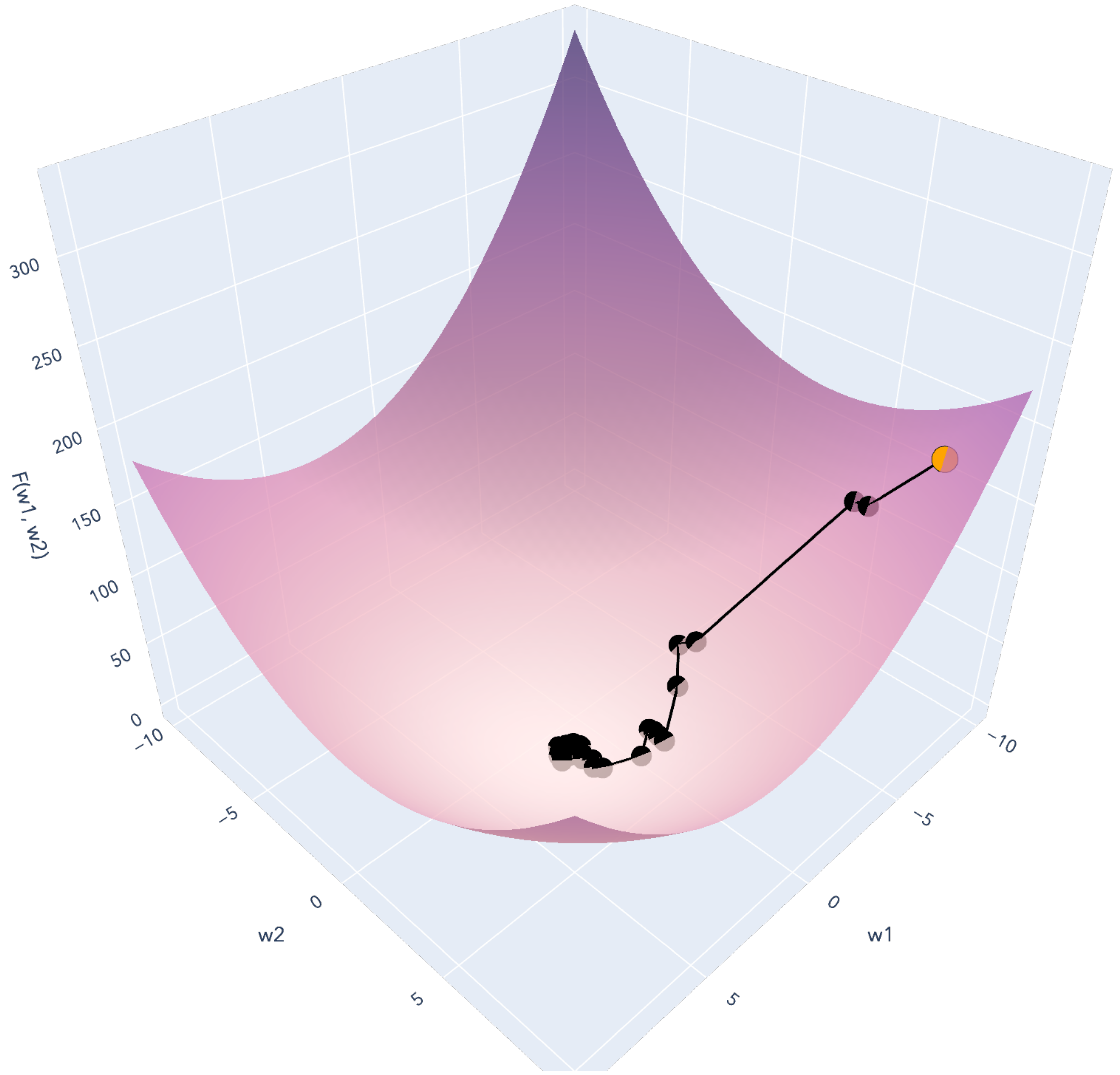
Initialize at a randomly chosen $w^{(0)} \in \mathbb{R}^d$.

For iteration $t = 1,2,\ldots$ (until "stopping condition" is satisfied):

Choose $j \in [n]$ uniformly at random.

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla_w \ell(h_w(x^{(j)}), y^{(j)})$$

Move in direction of steepest descent **in expectation**.



Stochastic GD

Algorithm

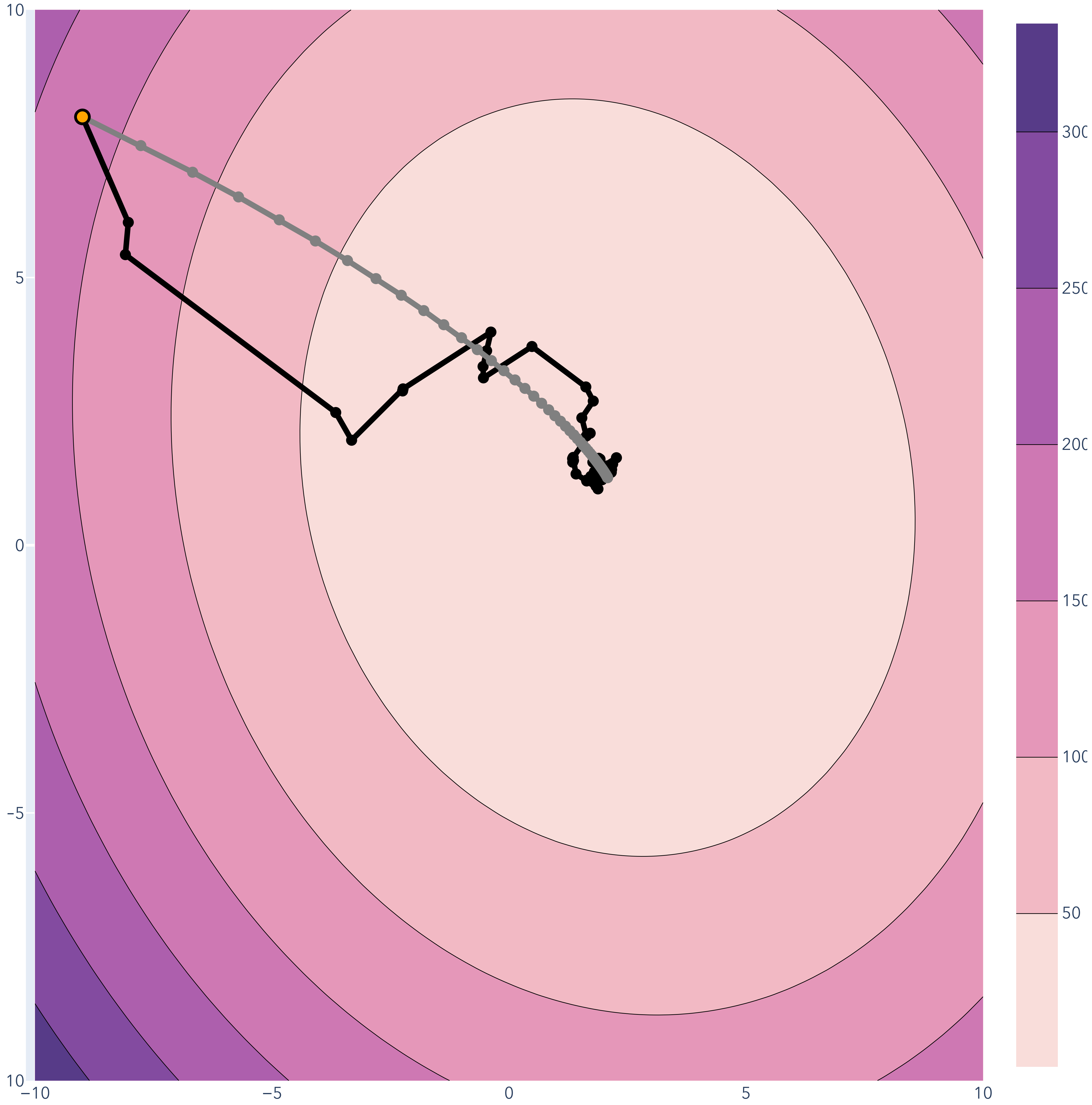
Initialize at a randomly chosen $w^{(0)} \in \mathbb{R}^d$.

For iteration $t = 1,2,\ldots$ (until "stopping condition" is satisfied):

Choose $j \in [n]$ uniformly at random.

$$w^{(t)} \leftarrow w^{(t-1)} - \eta \nabla_w \ell(h_w(x^{(j)}), y^{(j)})$$

Computation: $O(d)$ instead of $O(nd)$ per step because only needs to touch single point.

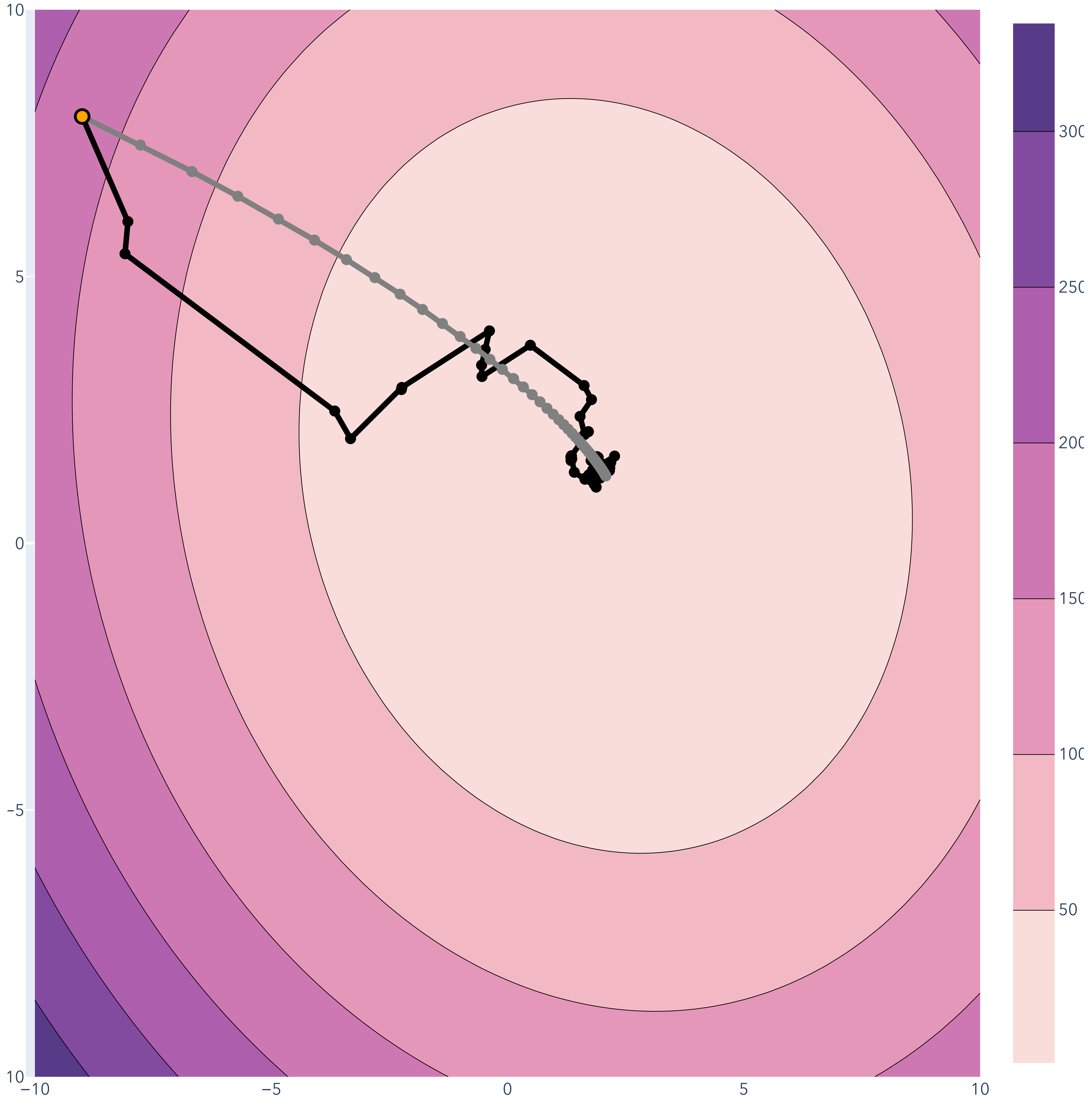


Stochastic GD

Main Difference

Stochastic gradient descent: More iterations to reach minimum, but lower cost per iteration.

Gradient descent: Fewer iterations to reach minimum, but higher cost per iteration.



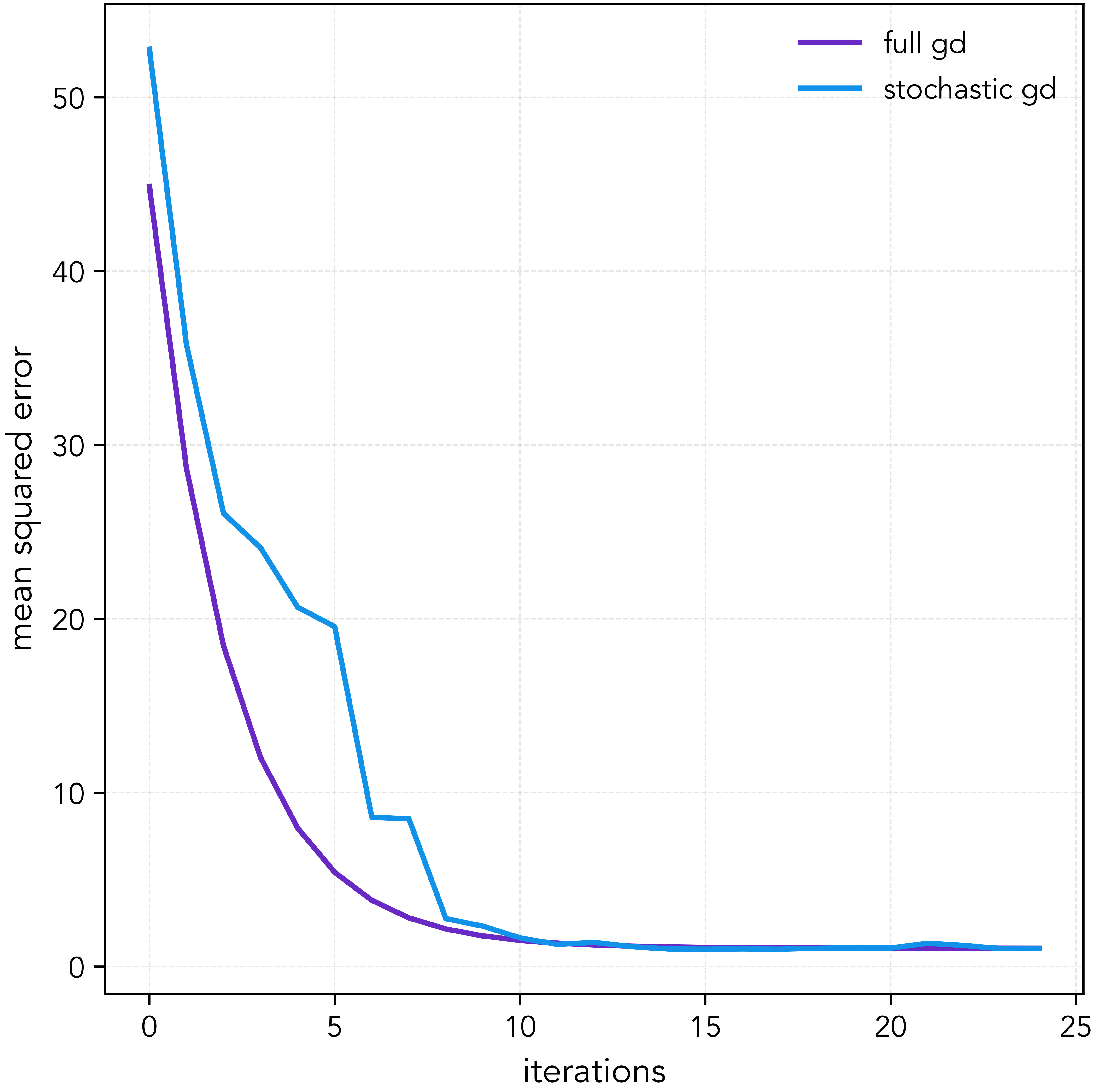
Stochastic GD

Main Difference

Common ("rule of thumb") behavior:

Stochastic gradient descent: More iterations to reach minimum, but lower cost per iteration.

Gradient descent: Fewer iterations to reach minimum, but higher cost per iteration.



Minibatch Gradient Descent

SGD in Practice

Full gradient: $\nabla \hat{R}_n(w) = \frac{1}{n} \sum_{i=1}^{n} \nabla_w \ell(h_w(x^{(i)}), y^{(i)})$

This is an average over the full batch of data $D_n = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)})\}$.

Take a random subsample of size N called a minibatch: ($N = 1$ is stochastic gradient descent)

$$(x^{(m_1)}, y^{(m_1)}), \ldots, (x^{(m_N)}, y^{(m_N)})$$

Minibatch gradient: $\nabla \hat{R}_N(w) = \frac{1}{N} \sum_{i=1}^{N} \nabla_w \ell(h_w(x^{(m_i)}), y^{(m_i)})$

Minibatch Gradient

Properties

$$\nabla \hat{R}_N(w) = \frac{1}{N} \sum_{i=1}^{N} \nabla_w \ell(h_w(x^{(m_i)}), y^{(m_i)})$$

1. $\hat{R}_N(w)$ is an unbiased estimator.

$$\mathbb{E}_{(m_1,\ldots,m_N)}[\nabla \hat{R}_N(w)] = \nabla \hat{R}_n(w)$$

2. $\hat{R}_N(w)$ is a better estimate with a bigger minibatch.

$$\mathrm{Var}\left[\nabla \hat{R}_N(w)\right] = \mathrm{Var}\left[\frac{1}{N} \sum_{i=1}^{N} \nabla_w \ell(h_w(x^{(m_i)}), y^{(m_i)})\right] = \frac{1}{N^2}\mathrm{Var}\left[\sum_{i} \nabla_w \ell^{(m_i)}\right] = \frac{1}{N}\mathrm{Var}\left[\nabla_w \ell^{(m_1)}\right]$$

Minibatch GD

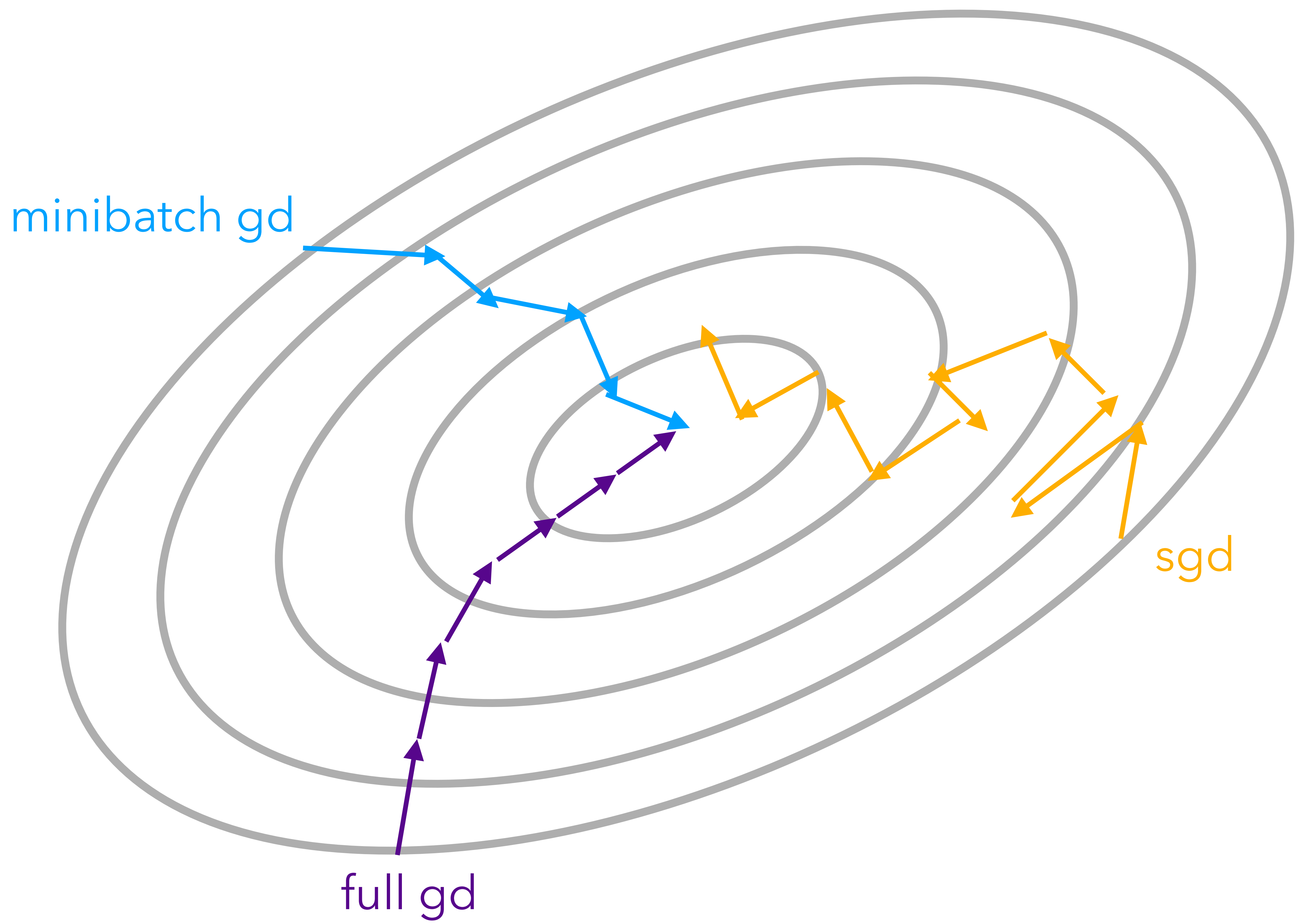
Comparison

Minibatch GD tends to need fewer iterations to converge than SGD.

Price per iteration is $O(Nd)$, where N is the minibatch size.

Typically, $N \ll n$ delivers good performance.

On modern hardware, small minibatch sizes $N \approx 32$ come "for free" because of parallelization.



(toy cartoon of gradient descent paths)

Notes on Convergence

Stochastic Gradient Descent

For convergence guarantees, can use **diminishing step sizes,** e.g. $\eta_t = 1/t$.

Theoretically, GD is much faster than SGD in terms of convergence rate:

But *much* more computationally costly to compute a single step.

Most advantage of GD over SGD comes into play once we're close to minimum.

In many ML problems, we don't care about optimizing close to minimum.

In practice, SGD with fixed step size can work well.

Typical approach: step size reduced by constant factor when validation performance stalls.

Supervised Learning

Excess Risk Formalization

1. Collect training dataset, a collection of labeled input-output pairs.

2. Decide on the template of the hypothesis mapping that will map inputs to outputs. — Representation

3. A learning algorithm takes the labeled training data as input and outputs a hypothesis. — Optimization

4. The hypothesis predicts on new, unseen data which we hope it does well on, under a notion of loss. — Generalization

We receive $\tilde{h}_n$ from an algorithm.

Excess risk of $\tilde{h}_n$:

$$R(\tilde{h}_n) - R(h^*) =$$

$$\underbrace{R(\tilde{h}_n) - R(\hat{h}_n)}_{\text{opt. error}} + \underbrace{R(\hat{h}_n) - R(h^*_{\mathcal{H}})}_{\text{est. error}} + \underbrace{R(h^*_{\mathcal{H}}) - R(h^*)}_{\text{approx. error}}$$

Optimization — Generalization — Representation

How do we get a good approximation to the ERM?

Outline

ERM: Learning as Optimization

Optimizing Linear Regression: Closed Form

Gradient Descent Intuition & Example

Gradient Descent Algorithm & Descent Lemma

Gradient Descent on Convex Functions

Stochastic Gradient Descent