Worksheets

Problems are sorted by topic in the worksheets below. Each worksheet also has a link to the discussion review slides for that week from Winter 2025.

- Arrays and Probability (slides)
- DataFrames and Querying (slides)
- Grouping, Merging, and Pivoting (slides)
- Visualization, Imputation, and Web Scraping (slides)
- Regular Expressions and Text Features (slides)
- Loss Functions and Simple Linear Regression (slides)
- Multiple Linear Regression (slides)
- Feature Engineering and Pipelines (slides)
- Cross-Validation and Regularization (slides)
- Gradient Descent and Classification (slides)
- Logistic Regression (slides)
- Clustering (slides)

lecture 20+21 (today)

lecture 22+23 (Thursday)

next Tuesday (Lecture 24)

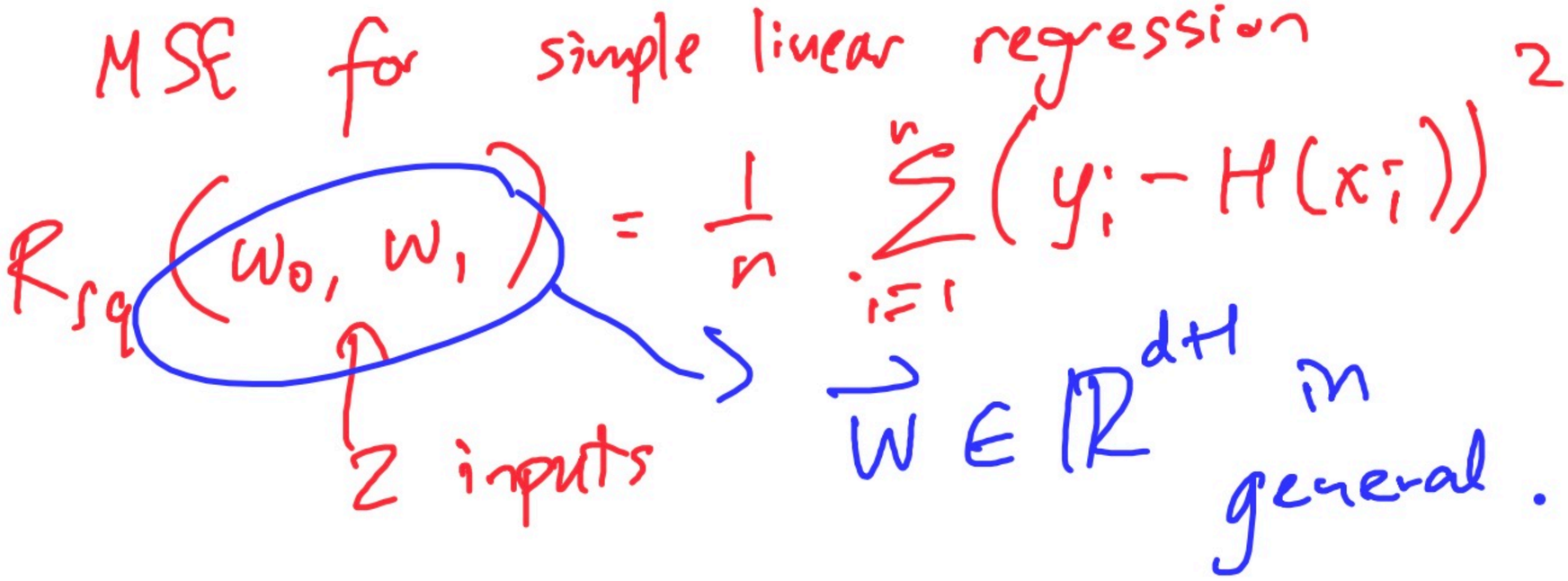
In addition, we have a few "review" worksheets which contain topics from multiple weeks, designed to be completed in the few days before an exam. You can treat them like mock exams, with the caveat that these worksheets contain topics from several different old exams, and were never tested as real exams, and so they may be too long or short and be missing topics.

- Midterm Review
- Final Review, Worksheet 1
- Final Review, Worksheet 2

Minimizing arbitrary functions

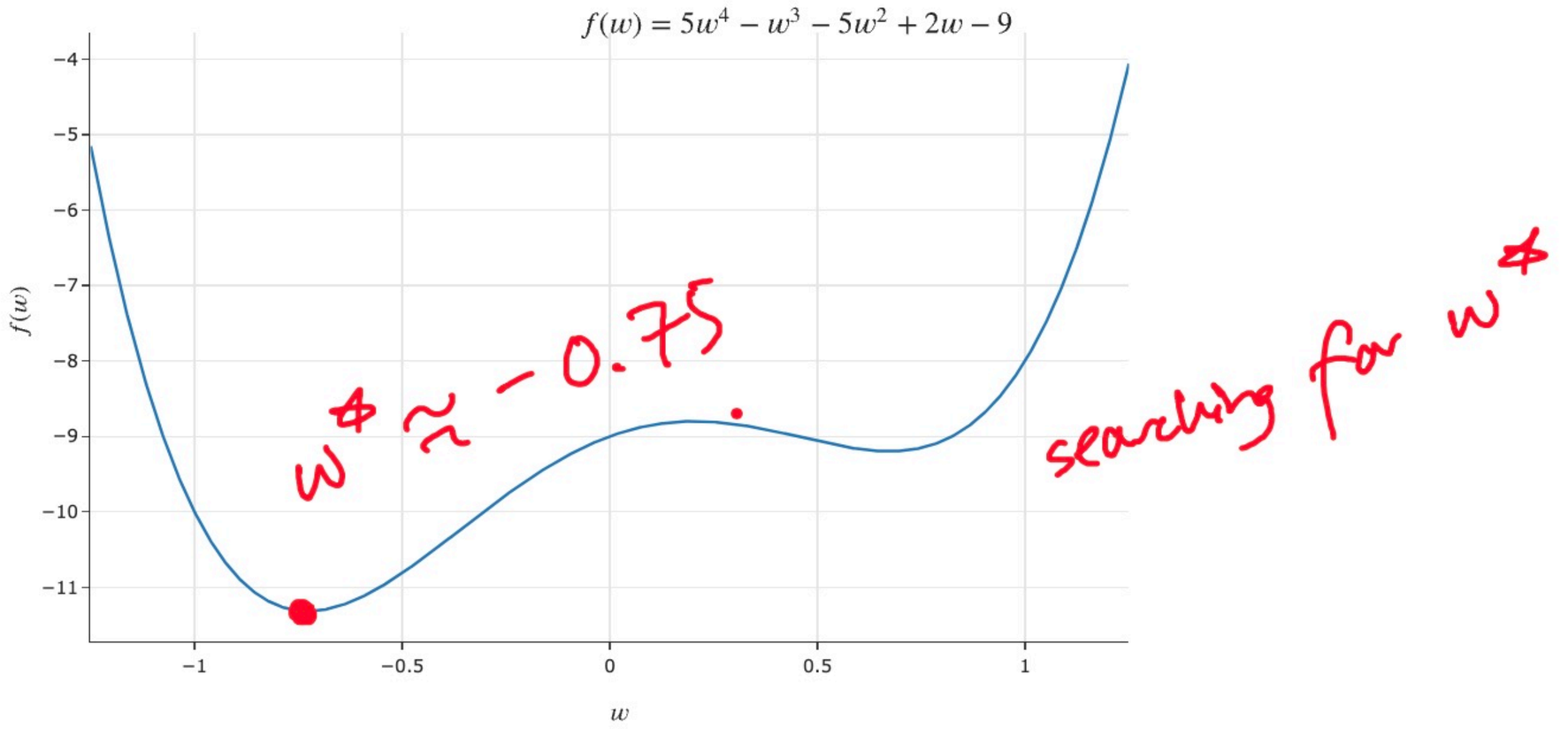
- Assume $f(w)$ is some **differentiable** function.

  For now, we'll assume f takes in a scalar, w, as input and returns a scalar as its output.



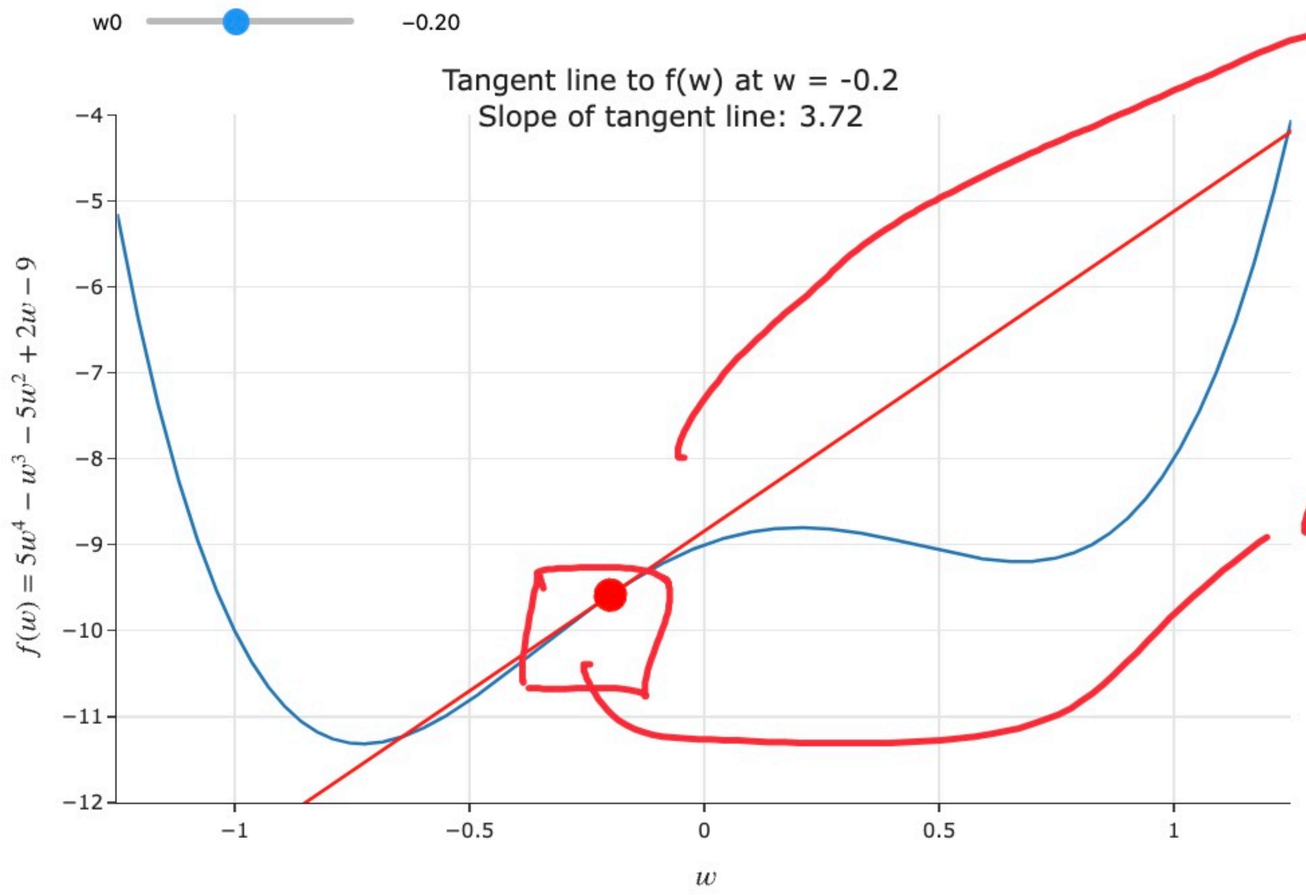
$$f(w) = 5w^4 - w^3 - 5w^2 + 2w - 9$$

In [2]:
```
1 util.draw_f()
```



- What does $\frac{d}{dw} f(w)$ mean?

In [3]:
```
from ipywidgets import interact
interact(util.show_tangent, w0=(-1.5, 1.5));
```

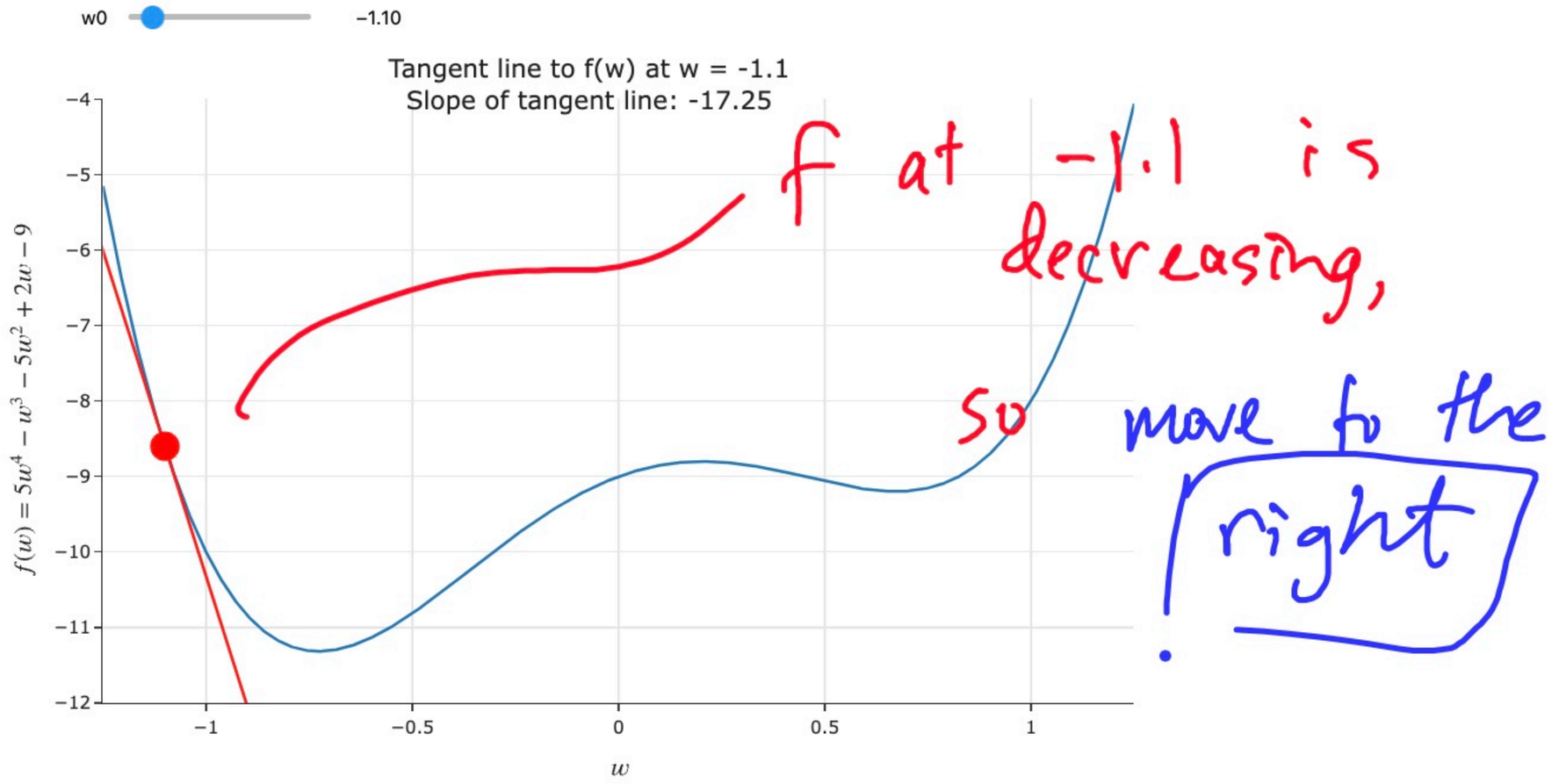


at $w = -0.2$,
f is increasing
so minimum is
to the left

derivative
= slope of
tangent line

- What does $\frac{d}{dw} f(w)$ mean?

In [3]:
```
from ipywidgets import interact
interact(util.show_tangent, w0=(-1.5, 1.5));
```



Gradient descent

- To minimize a **differentiable** function f:

1. Pick a positive number, α. This number is called the **learning rate**, or **step size**.
   Think of α as a hyperparameter of the minimization process.
2. Pick an **initial guess**, $w^{(0)}$.
3. Then, repeatedly update your guess using the **update rule**:

$$w^{(t+1)} = w^{(t)} - \alpha \frac{df}{dw}(w^{(t)})$$

next guess

current guess

derivative at current guess

move opposite the direction of derivative

(if $\frac{d}{dw}$ pos, move left

if $\frac{d}{dw}$ neg, move right)

α: multiplier of derivative steps

size of an

to make sure aren't too big/small

Gradient descent

- To minimize a **differentiable** function f:

1. Pick a positive number, α. This number is called the **learning rate**, or **step size**.
   Think of α as a hyperparameter of the minimization process.
2. Pick an **initial guess**, $w^{(0)}$.
3. Then, repeatedly update your guess using the **update rule**:

$$w^{(t+1)} = w^{(t)} - \alpha \frac{df}{dw}(w^{(t)})$$

next guess

current guess

derivative at current guess

move opposite the direction of derivative

(if $\frac{d}{dw}$ pos, move left

if $\frac{d}{dw}$ neg, move right)

α: multiplier of derivative steps

size of an

to make sure aren't too big/small

```
w = 0
alpha = 0.01

# while the derivative is not close to 0...
while np.abs(util.df(w)) > 0.0001:
    print(f'current w: {w}, current f: {util.f(w)}, current df: {util.df(w)}')
    w = w - alpha * util.df(w)
```

0 − 0.01×2 = −0.02

```
current w: 0, current f: -9, current df: 2
current w: -0.02, current f: -9.0419912, current df: 2.19864
current w: -0.041986400000000001, current f: -9.092697534534848, current df: 2.4130951056030714
current w: -0.06611735105603073, current f: -9.153707640152518, current df: 2.64227835278975 18
current w: -0.09254013458392825, current f: -9.226739485158518, current df: 2.883860640779675
current w: -0.121378740991724 99, current f: -9.313547950444573, current df: 3.1338240423160784
current w: -0.1527169814148857 7, current f: -9.415764917165259, current df: 3.385967620974774
current w: -0.1865766576246335 2, current f: -9.534653693114066, current df: 3.6314361909895023
current w: -0.2228910195345285 6, current f: -9.670770041544909, current df: 3.8584026460669874
current w: -0.2614750459951984 4, current f: -9.82354751342766, current df: 4.0521060683072125
current w: -0.3019961066782706, current f: -9.990869209447597, current df: 4.195505266352445
current w: -0.3439511593417950 4, current f: -10.168746781865512, current df: 4.270799440635486
current w: -0.3866591537481499, current f: -10.351278044747946, current df: 4.2619237692687495
current w: -0.4292783914408374, current f: -10.531053598024561, current df: 4.157796200656783
current w: -0.4708563534474052 3, current f: -10.700082595297433, current df: 3.955615616354196
current w: -0.5104125096109472, current f: -10.851100752632558, current df: 3.663099472791226
current w: -0.5470435043388595, current f: -10.97889105993112, current df: 3.298537720969345
current w: -0.5800288815485529, current f: -11.081146620386715, current df: 2.8881653309894535
current w: -0.6089105348584475, current f: -11.158554739933203, current df: 2.4614492130172607
current w: -0.6335250269886201, current f: -11.214127069534642, current df: 2.0458328258724965
```

```
current w: -0.7192943783906268, current f: -11.315267348949988, current df: 0.19775672775758135
current w: -0.7212719456682026, current f: -11.315267348949988, current df: 0.14742726770020997
current w: -0.7227462183452047, current f: -11.31545689038677, current df: 0.1096713033253635?
current w: -0.7238429313784583, current f: -11.315561705332776, current df: 0.08145389035954498
current w: -0.7246574702820537, current f: -11.315619492162984, current df: 0.06042428318983450?
current w: -0.7252617131138952, current f: -11.315651279613537, current df: 0.04478421644448982
current w: -0.7257095552178397, current f: -11.315668736007677, current df: 0.03317046929149114?
current w: -0.726041259971311?, current f: -11.315678310434798, current df: 0.02455644357527298
current w: -0.7262868244070646, current f: -11.315683556939849, current df: 0.018172790401585814
current w: -0.7264685523110804, current f: -11.315686429904257, current df: 0.013445004670057159
current w: -0.726603002357781, current f: -11.31568800233065, current df: 0.009945206910921378
current w: -0.7267024544268902, current f: -11.315688862625834, current df: 0.007355339012650397
current w: -0.7267760078170167, current f: -11.315689333174019, current df: 0.005439315124045052
current w: -0.7268304009682571, current f: -11.315689590492822, current df: 0.004022080185654531
current w: -0.7268706217701136, current f: -11.315689731185978, current df: 0.002973934011605728
current w: -0.7269003611102297, current f: -11.31568980810356, current df: 0.002198835686455069?
current w: -0.7269223494670942, current f: -11.315689850151257, current df: 0.001625698715013612?
current w: -0.7269386064542444, current f: -11.3156898731356, current df: 0.0012019236357527774
current w: -0.7269506256906019, current f: -11.315689885698859, current df: 0.0008885992570517587
current w: -0.7269595116831724, current f: -11.315689892565713, current df: 0.0006569454229836?8
current w: -0.7269660811374022, current f: -11.315689896318922, current df: 0.0004856779694417668
current w: -0.7269709379170967, current f: -11.31568989837027, current df: 0.0003590578185006521
current w: -0.7269745284952817, current f: -11.315689899491439, current df: 0.0002654471448240159
current w: -0.7269771829667299, current f: -11.315689900104207, current df: 0.0001962411221292370?
current w: -0.7269791453779512, current f: -11.31568990043911, current df: 0.0001450777175318052
current w: -0.7269805961551264, current f: -11.31568990062215, current df: 0.0001072532535202341?
```

We all close to one another.

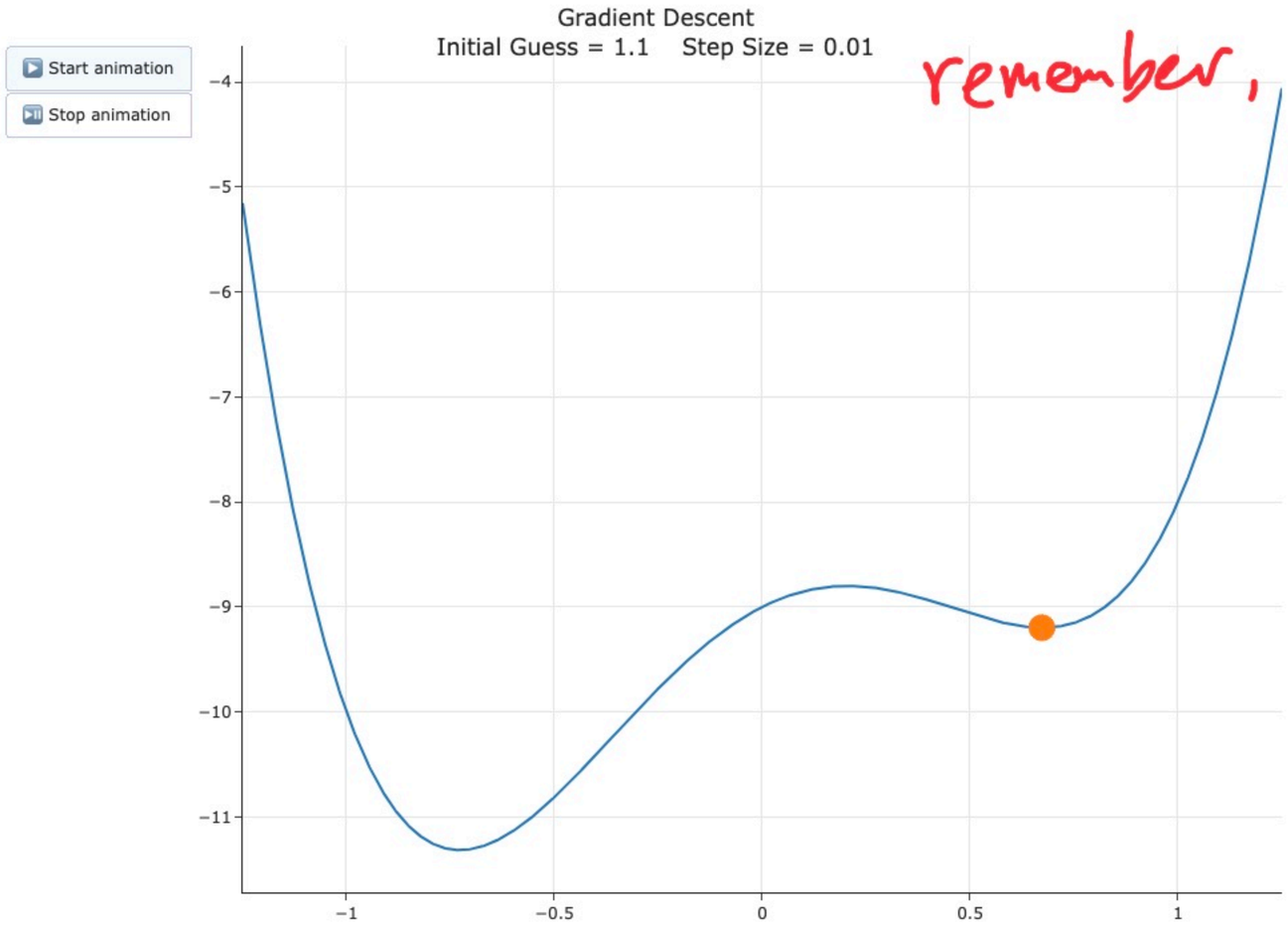
df close to 0

n [ ]: 1

Visualizing $w^{(0)} = 1.1, \alpha = 0.01$

What if we start with a different initial guess?

In [10]:
```
1 util.minimizing_animation(w0=1.1, alpha=0.01)
```



remember, in higher dimensions, we can't always visualize f!

Lingering questions

- When is gradient descent *guaranteed* to converge to a global minimum? What kinds of functions work well with gradient descent?
- How do we choose a step size?
- How do we use gradient descent to minimize functions of multiple variables, e.g.:

$$R_{\text{ridge}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2 + \lambda \sum_{j=1}^{d} w_j^2$$

This is a function of $d + 1$ variables: $w_0, w_1, \ldots, w_d$.

mean squared error

regularization penalty.

Lingering questions

- When is gradient descent *guaranteed* to converge to a global minimum? What kinds of functions work well with gradient descent?
- How do we choose a step size?
- How do we use gradient descent to minimize functions of multiple variables, e.g.:

$$R_{\text{ridge}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2 + \lambda \sum_{j=1}^{d} w_j^2$$

L_2 regularization

This is a function of $d + 1$ variables: $w_0, w_1, \ldots, w_d$.

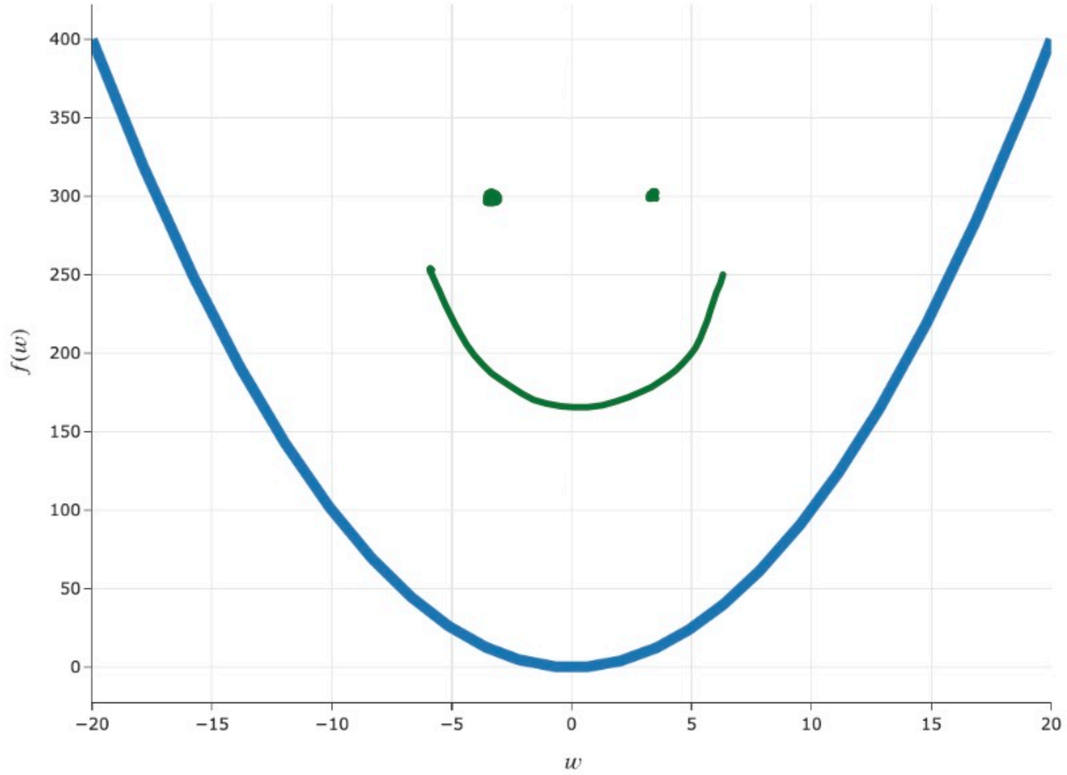
- **Question**: Why **can't** we use gradient descent to find $\vec{w}^*_{\text{LASSO}}$?

L_1 regularization

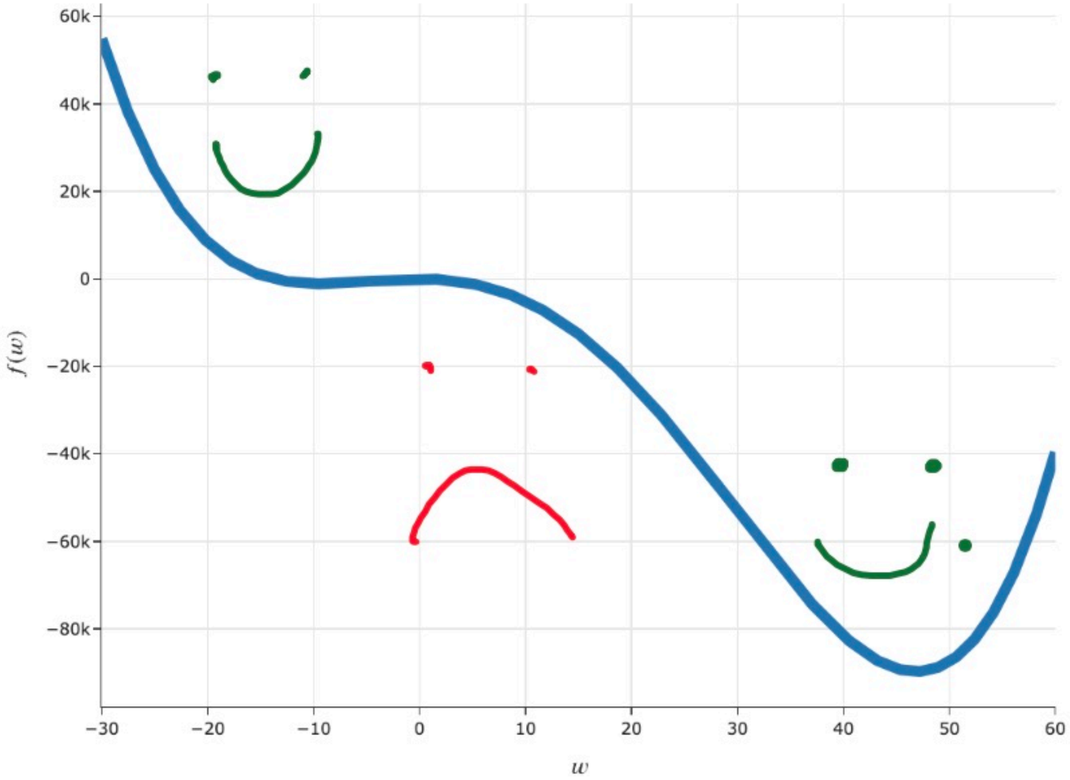
$$R_{\text{LASSO}}(\vec{w}) = \frac{1}{n}\|\vec{y} - X\vec{w}\|^2 + \lambda \sum_{j=1}^{d} |w_d|$$

$|w|$ not differentiable

What makes a function convex?

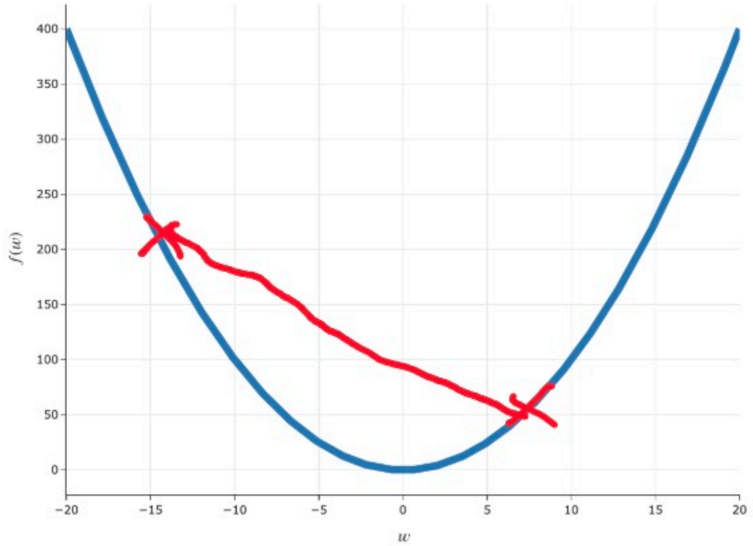


A **convex** function ✅.

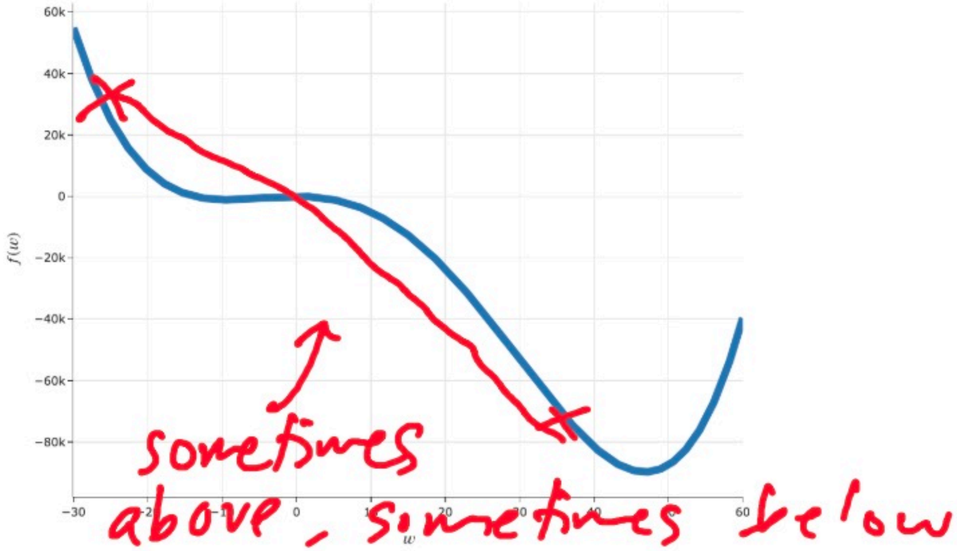


A **non-convex** function ❌.

Intuitive definition of convexity



A **convex** function ✅.

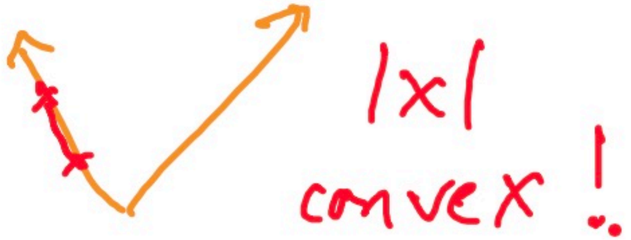


A **non-convex** function ❌.

- A function f is **convex** if, for **every** a, b in the domain of f, the line segment between:

$$(a, f(a)) \text{ and } (b, f(b))$$

does not go below the plot of f.



Second derivative test for convexity

- If $f(w)$ is a function of a single variable and is **twice** differentiable, then $f(w)$ is convex **if and only if**:

$$\frac{d^2 f}{dw^2}(w) \geq 0, \quad \forall\, w$$

- Example: $f(w) = w^4$ is convex.

$f'(w) = 4w^3$ $\quad$ $f''(w) = 12w^2 \geq 0$

e.g. $f(w) = |w|$

Second derivative test for convexity

- If $f(w)$ is a function of a single variable and is **twice** differentiable, then $f(w)$ is convex **if and only if**:

$$\frac{d^2 f}{dw^2}(w) \geq 0, \quad \forall\, w$$

- Example: $f(w) = w^4$ is convex.

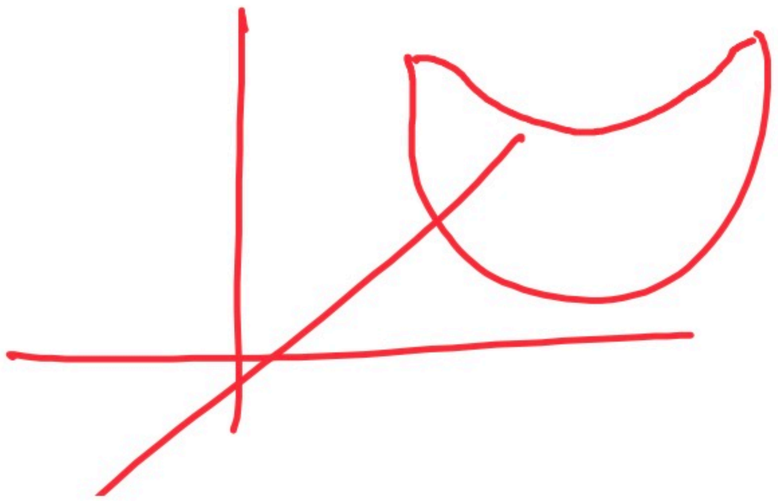
$f'(w) = 4w^3$ $\quad f''(w) = 12w^2 \geq 0$

Minimizing functions of multiple variables

- We will typically use gradient descent to minimize empirical risk functions, $R(\vec{w})$, to find optimal model parameters.

  A model with $d + 1$ parameters $w_0, w_1, \ldots, w_d$ will have a $(d + 2)$-dimensional loss surface.

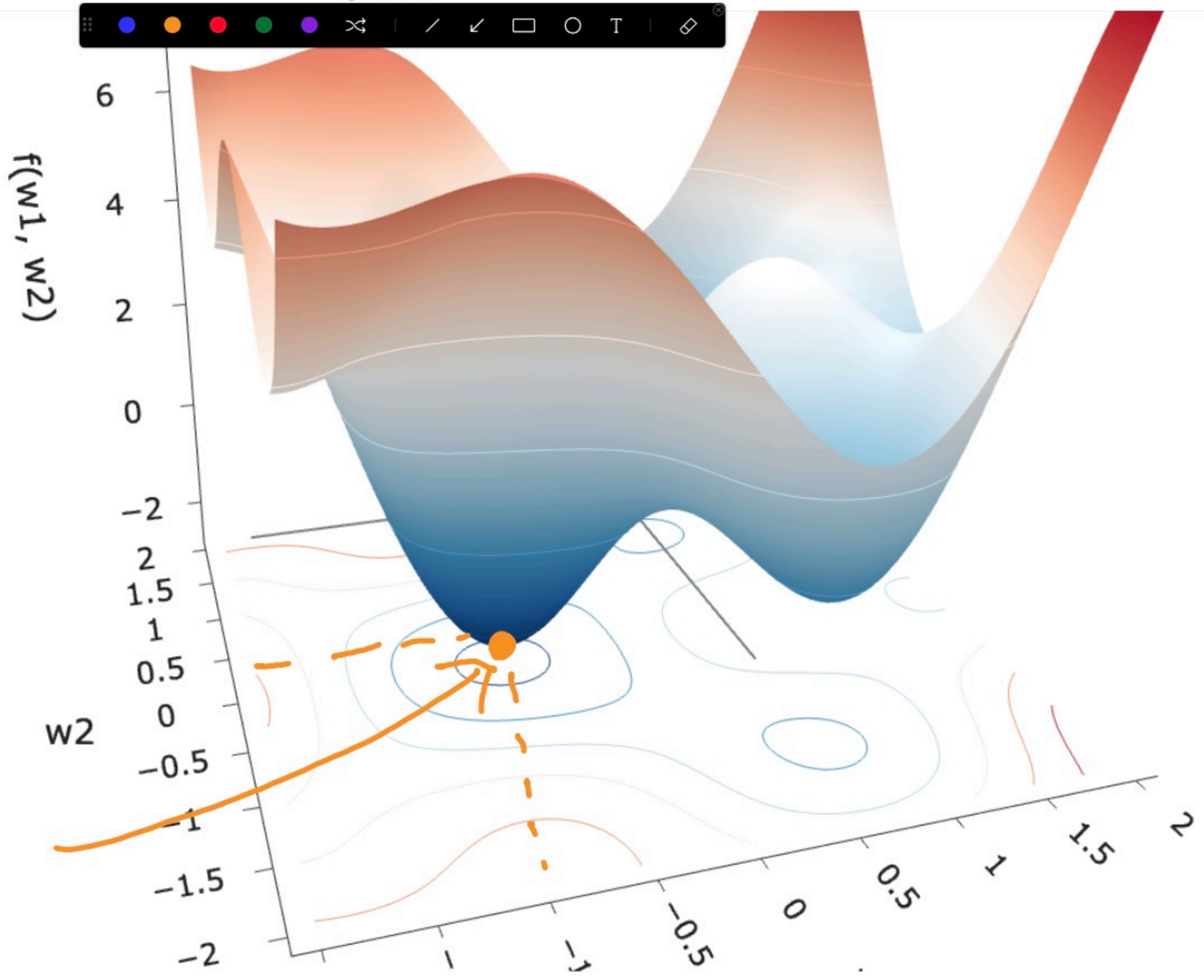
e.g. SLR



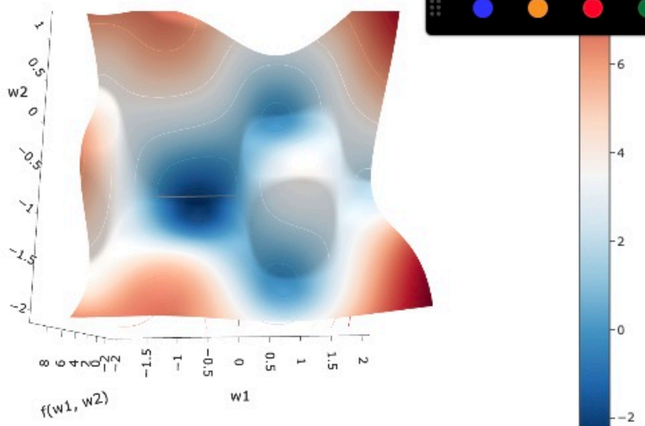
$d = 1$

2 parameters, w_0, w_1

3 dimensional loss surface.

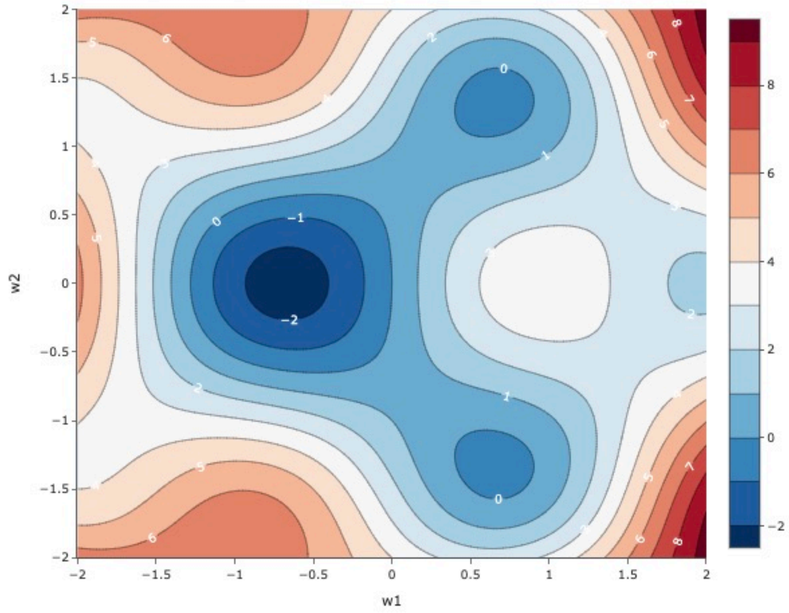


we want to find the coords of bottom



In [17]:
```
1 util.make_3D_contour()
```

$$f(w_1, w_2) = 3\sin(2w_1)\cos(2w_2) + w_1^2 + w_2^2$$



contour plot
= birds eye view
of 3D version

Minimizing functions of multiple variables

- Consider the function:

$$f(w_1, w_2) = 3\sin(2w_1)\cos(2w_2) + w_1^2 + w_2^2$$

- It has two **partial derivatives**: $\frac{\partial f}{\partial w_1}$ and $\frac{\partial f}{\partial w_2}$.

  See the annotated slides for what they are and how we find them.

$$\frac{\partial f}{\partial w_1} = 3\cos(2w_2)\cos(2w_1)\cdot 2 + 2w_1 = 6\cos(2w_1)\cos(2w_2) + 2w_1$$

$$\frac{\partial f}{\partial w_2} = -6\sin(2w_1)\sin(2w_2) + 2w_2$$

The gradient vector

"nabla"

- If $f(\vec{w})$ is a function of multiple variables, then its **gradient**, $\nabla f(\vec{w})$, is a vector containing its partial derivatives.

- Example:

$$f(\vec{w}) = f(w_1, w_2) = 3\sin(2w_1)\cos(2w_2) + w_1^2 + w_2^2$$

$$\nabla f(\vec{w}) = \begin{bmatrix} 6\cos(2w_1)\cos(2w_2) + 2w_1 \\ -6\sin(2w_1)\sin(2w_2) + 2w_2 \end{bmatrix}$$

The gradient vector

"nabla"

- If $f(\vec{w})$ is a function of multiple variables, then its **gradient**, $\nabla f(\vec{w})$, is a vector containing its partial derivatives.

- Example:

$$f(\vec{w}) = f(w_1, w_2) = 3\sin(2w_1)\cos(2w_2) + w_1^2 + w_2^2$$

$$\nabla f(\vec{w}) = \begin{bmatrix} 6\cos(2w_1)\cos(2w_2) + 2w_1 \\ -6\sin(2w_1)\sin(2w_2) + 2w_2 \end{bmatrix}$$

e.g. $\vec{w} = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$, $\nabla f(\vec{w}) = \begin{bmatrix} 6 \cdot \cos(2 \cdot 0)\cos(2 \cdot 1) + 0 \\ -6\sin(2 \cdot 0)\sin(2 \cdot 1) + 2 \end{bmatrix} = \begin{bmatrix} 0.5 \\ -3 \end{bmatrix}$

The gradient vector

- If $f(\vec{w})$ is a function of multiple variables, then its **gradient**, $\nabla f(\vec{w})$, is a vector containing its partial derivatives.

- Example:

$$f(\vec{w}) = f(w_1, w_2) = 3\sin(2w_1)\cos(2w_2) + w_1^2 + w_2^2$$

$$\nabla f(\vec{w}) = \begin{bmatrix} 6\cos(2w_1)\cos(2w_2) + 2w_1 \\ 6\sin(2w_1)\sin(2w_2) + 2w_2 \end{bmatrix}$$

$\nabla f(\vec{w}) = \begin{bmatrix} 2w_0 \\ 2w_1 \\ \vdots \end{bmatrix} = 2\vec{w}$

- Example:

$\vec{w} = \begin{bmatrix} w_0 \\ w_1 \\ \vdots \\ w_d \end{bmatrix}$

$\frac{\partial f}{\partial w_1} = 2w_1$

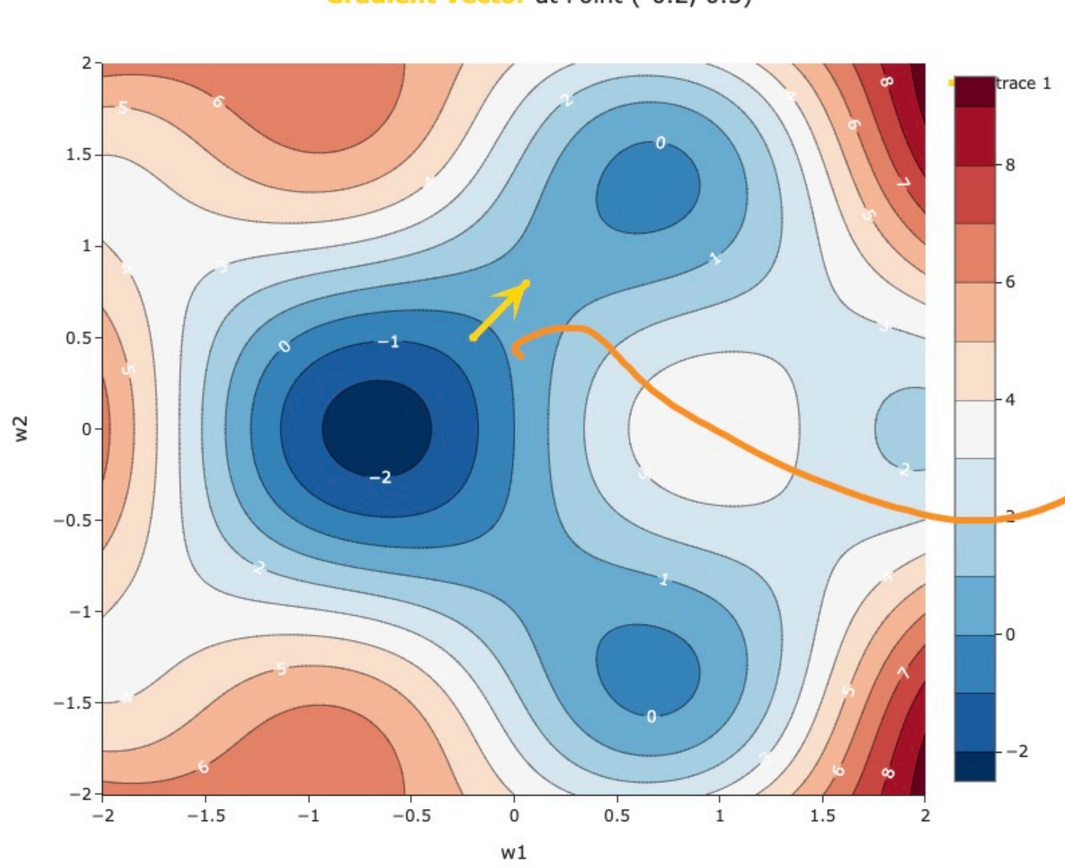
$\frac{\partial f}{\partial w_7} = 2w_7$

$f(\vec{w}) = \vec{w}^T\vec{w} = w_0^2 + w_1^2 + \cdots + w_d^2$

$\nabla f(\vec{w}) = 2\vec{w}$

$$\nabla f(\vec{w}) = \begin{bmatrix} \\ -6\sin(2w_1)\sin(2w_2) + 2w_2 \end{bmatrix}$$

In [18]:
```
1 util.make_3D_contour(with_gradient=True, w1_start=-0.2, w2_start=0.5)
```



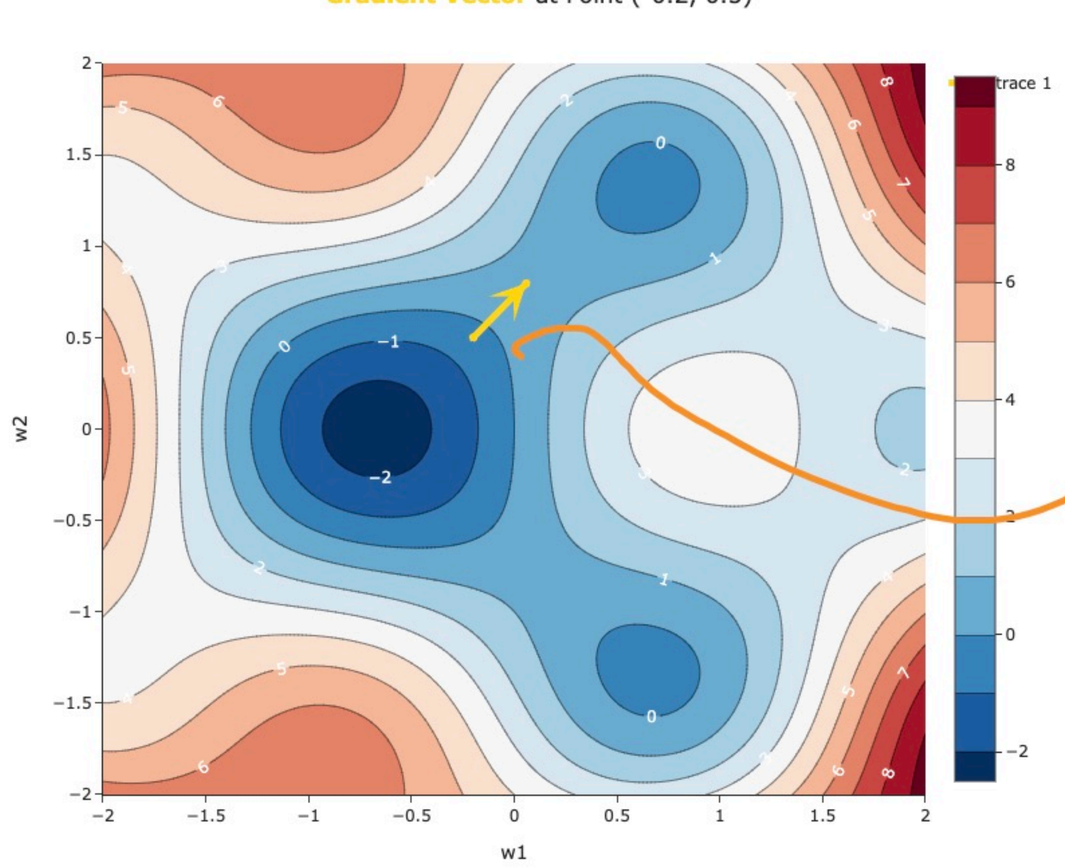
$$\vec{w} = \begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix}$$

$$\nabla f(\vec{w}) \simeq \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

describes the direction the function is ~~increasing~~ decreasing: the QUICKEST

$$\nabla f(\vec{w}) = \begin{bmatrix} \\ -6\sin(2w_1)\sin(2w_2) + 2w_2 \end{bmatrix}$$

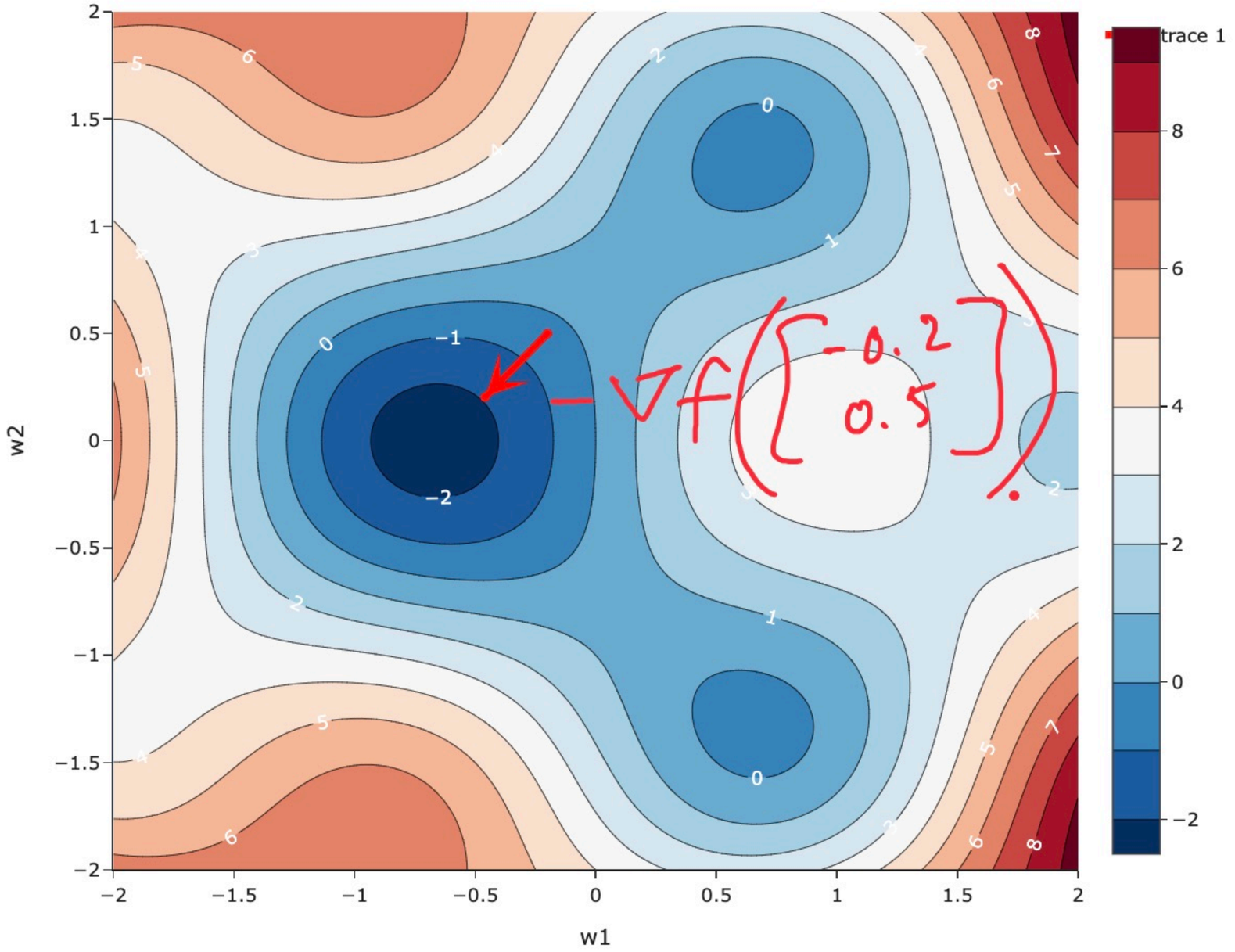
In [18]:
```
1 util.make_3D_contour(with_gradient=True, w1_start=-0.2, w2_start=0.5)
```

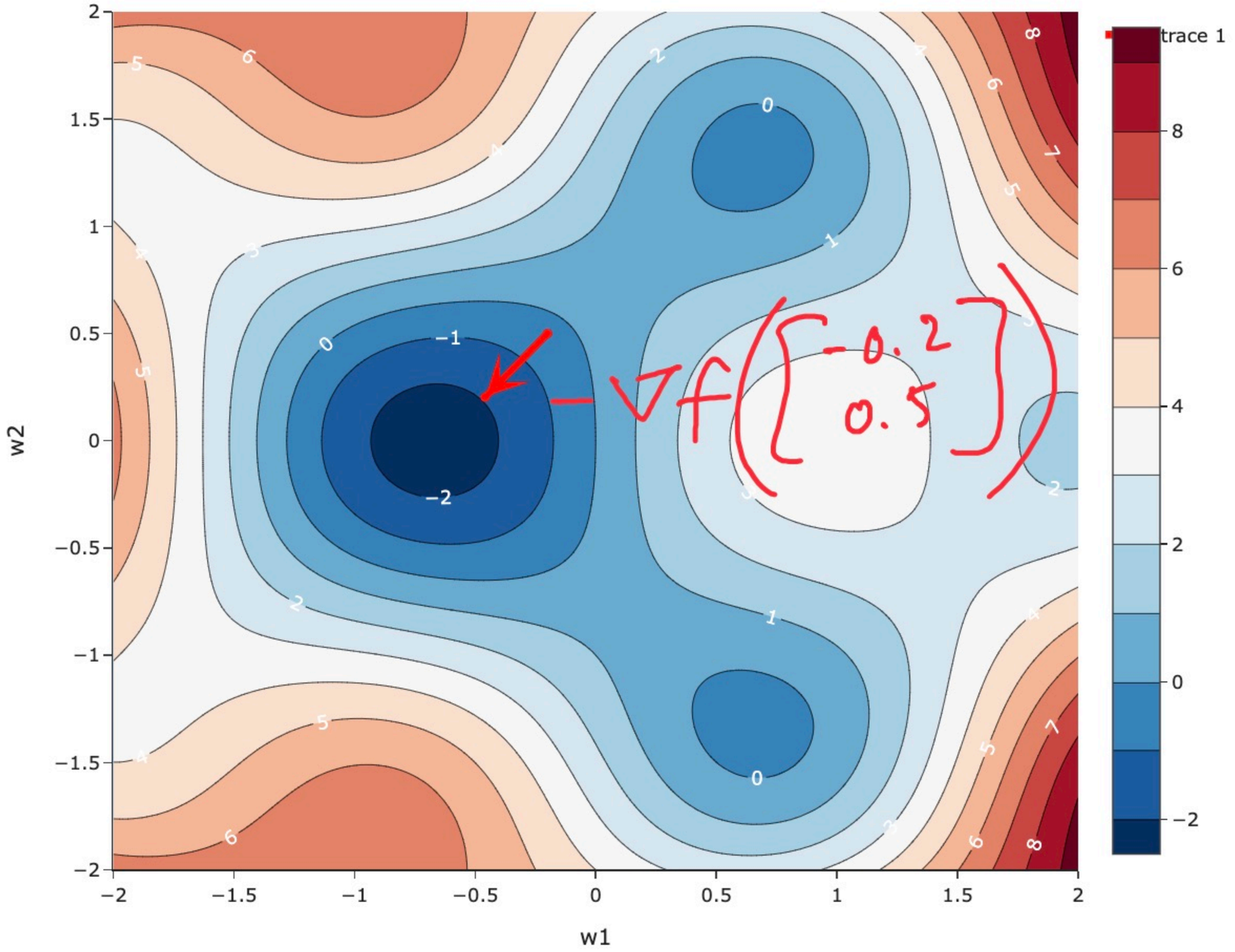


$$\vec{w} = \begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix}$$

$$\nabla f(\vec{w}) \simeq \begin{bmatrix} 0.1 \\ 0.1 \end{bmatrix}$$

describes the direction the function is increasing the QUICKEST (steepest way up the hill).


w2
w1
trace 1
8
6
4
2
0
−2
2
1.5
1
0.5
0
−0.5
−1
−1.5
−2
−1.5
−1
−0.5
0.5
1
1.5
−1
−2
0
$-\nabla f\left(\begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix}\right)$


w2
w1
trace 1
$-\nabla f\left(\begin{bmatrix} -0.2 \\ 0.5 \end{bmatrix}\right)$

Gradient descent for functions of multiple variables

- Example:

$$f(\vec{w}) = f(w_1, w_2) = 3\sin(2w_1)\cos(2w_2) + w_1^2 + w_2^2$$

$$\nabla f(\vec{w}) = \begin{bmatrix} 6\cos(2w_1)\cos(2w_2) + 2w_1 \\ -6\sin(2w_1)\sin(2w_2) + 2w_2 \end{bmatrix}$$

- The global minimizer* of f is a vector, $\vec{w}^* = \begin{bmatrix} w_1^* \\ w_2^* \end{bmatrix}$.

*If one exists.

- We start with an initial guess, $\vec{w}^{(0)}$, and step size α, and update our guesses using:

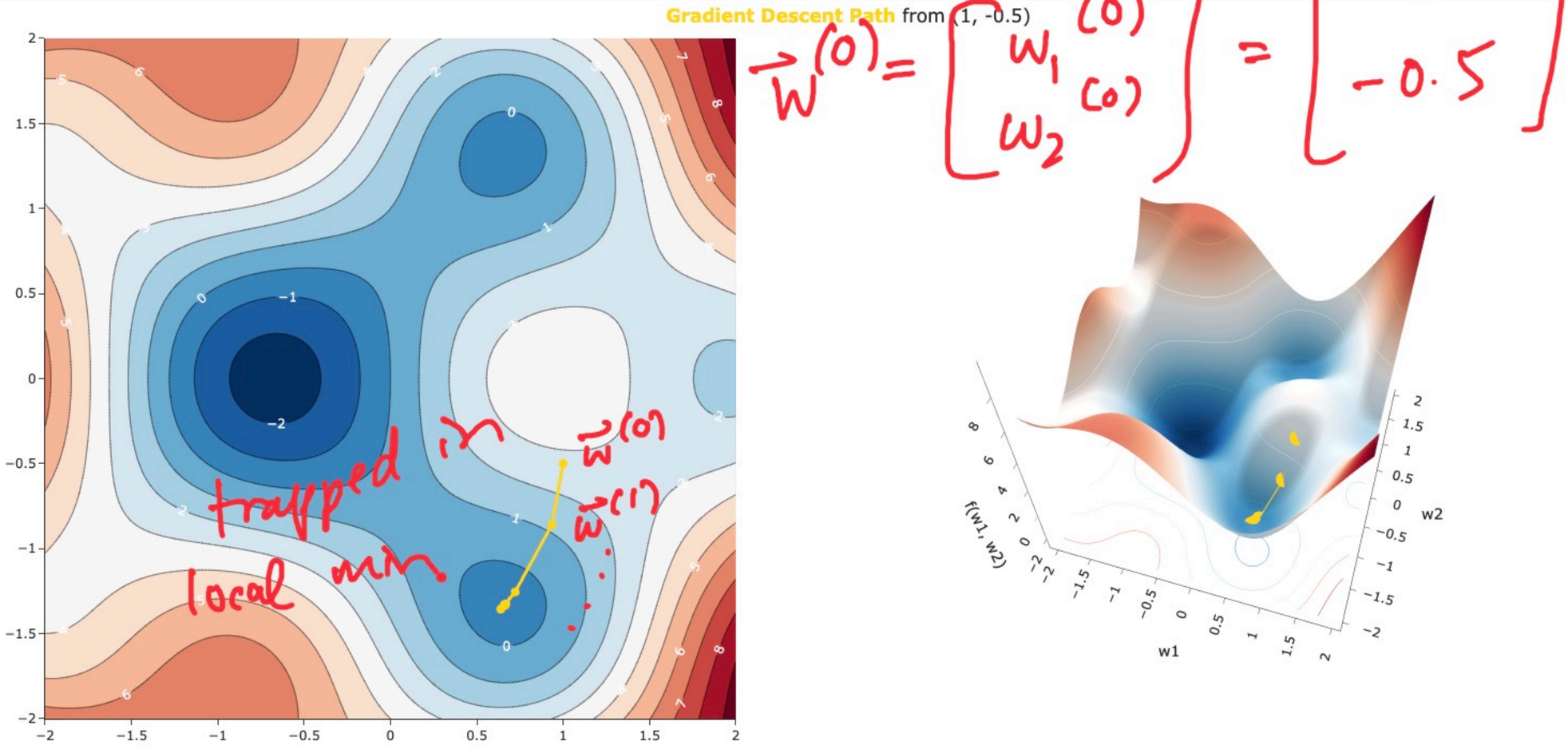
$$\vec{w}^{(t+1)} = \vec{w}^{(t)} - \alpha \nabla f(\vec{w}^{(t)})$$

the gradient descent update rule.

- Let's visualize the execution of gradient descent on our trigonometric example.

  Change `w1_start`, `w2_start`, and `step_size` below and see what happens!

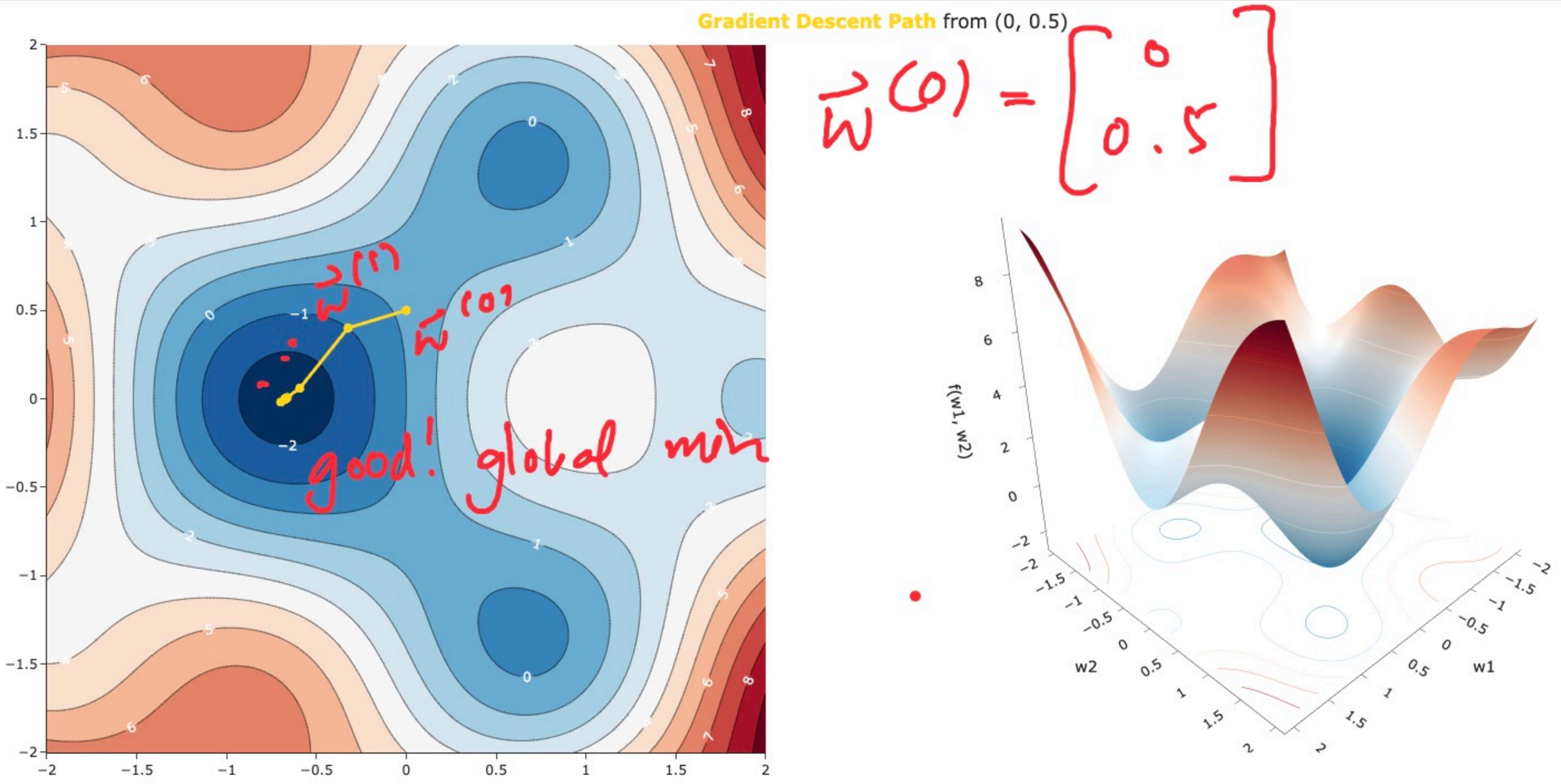
In [20]:
```
1 util.display_paths(w1_start=1, w2_start=-0.5, iterations=10, step_size=0.1)
```



- Let's visualize the execution of gradient descent on our trigonometric example.

  Change `w1_start`, `w2_start`, and `step_size` below and see what happens!

In [21]:
```
1 util.display_paths(w1_start=0, w2_start=0.5, iterations=10, step_size=0.1)
```



First, find $\frac{\partial f}{\partial w_1}$, $\frac{\partial f}{\partial w_2}$

$$\frac{\partial f}{\partial w_1} = 2(w_1 - 2) + 2 = 2w_1 - 2$$

$$\frac{\partial f}{\partial w_2} = -2(w_2 - 3) = 6 - 2w_2$$

$$\nabla f(\vec{w}) = \begin{bmatrix} 2w_1 - 2 \\ 6 - 2w_2 \end{bmatrix}$$

Activity

Consider the following function.

$$f(w_1, w_2) = (w_1 - 2)^2 + 2w_1 - (w_2 - 3)^2$$

Given an initial guess of $\vec{w}^{(0)} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ and a step size of $\alpha = \frac{1}{3}$, perform ~~two~~ one iterations of gradient descent.

What is $\vec{w}^{(1)}$?

$$\vec{w}^{(1)} = \vec{w}^{(0)} - \alpha \nabla f(\vec{w}^{(0)}) = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \frac{1}{3} \begin{bmatrix} 2 \cdot 0 - 2 \\ 6 - 2 \cdot 0 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} - \frac{1}{3} \begin{bmatrix} -2 \\ 6 \end{bmatrix} = \begin{bmatrix} 2/3 \\ -2 \end{bmatrix}$$

empirical risk:

$$R_{\text{sq}}(w_0, w_1) = R_{\text{sq}}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

- This is a function of multiple variables, and is differentiable, so it has a gradient!

$$\nabla R(\vec{w}) = \begin{bmatrix} -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i)) \\ -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))x_i \end{bmatrix}$$

$\frac{\partial R}{\partial w_0}$

$\frac{\partial R}{\partial w_1}$

empirical risk:

$$R_{\text{sq}}(w_0, w_1) = R_{\text{sq}}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

- This is a function of multiple variables, and is differentiable, so it has a gradient!

$$\nabla R(\vec{w}) = \begin{bmatrix} -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i)) \\ -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))x_i \end{bmatrix}$$

- **Key idea**: To find $\vec{w}^* = \begin{bmatrix} w_0^* \\ w_1^* \end{bmatrix}$, we *could* use gradient descent!
- Why would we, when closed-form solutions exist? — can be more efficient!

empirical risk:

$$R_{\text{sq}}(w_0, w_1) = R_{\text{sq}}(\vec{w}) = \frac{1}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))^2$$

mean squared error

- This is a function of multiple variables, and is differentiable, so it has a gradient!

In HW 10, you'll do something similar

$$\nabla R(\vec{w}) = \begin{bmatrix} -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i)) \\ -\frac{2}{n}\sum_{i=1}^{n}(y_i - (w_0 + w_1 x_i))x_i \end{bmatrix}$$

involves $n, x_1, \ldots, x_n, y_1, \ldots, y_n$, but $\vec{w}$ are the only unknown.

- **Key idea**: To find $\vec{w}^* = \begin{bmatrix} w_0^* \\ w_1^* \end{bmatrix}$, we *could* use gradient descent!
- Why would we, when closed-form solutions exist?

can be more efficient!