CS074/274 Machine Learning and Statistical Data Analysis

Gradient Descent

Least Square Regression

Sarah Preum

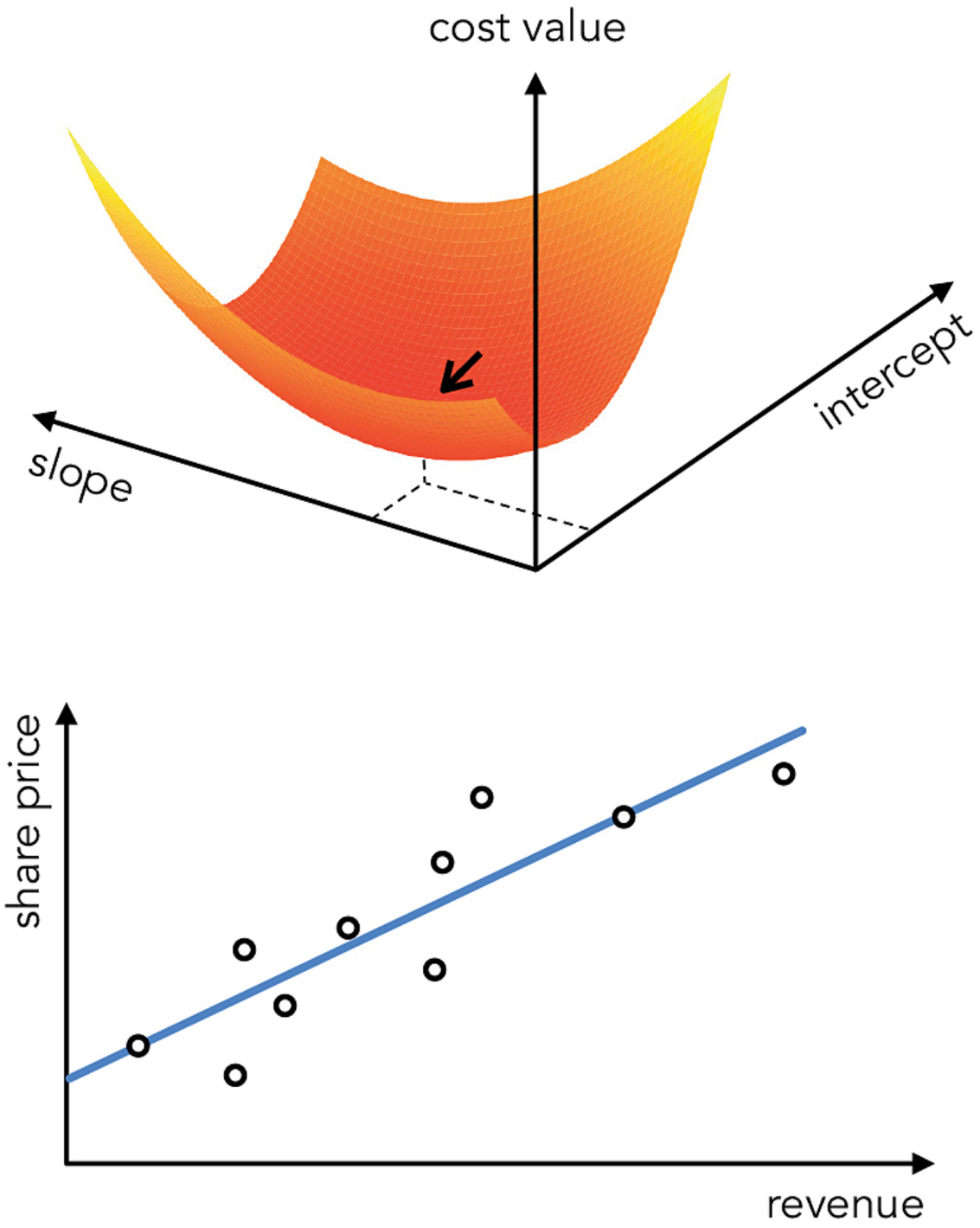
Summary

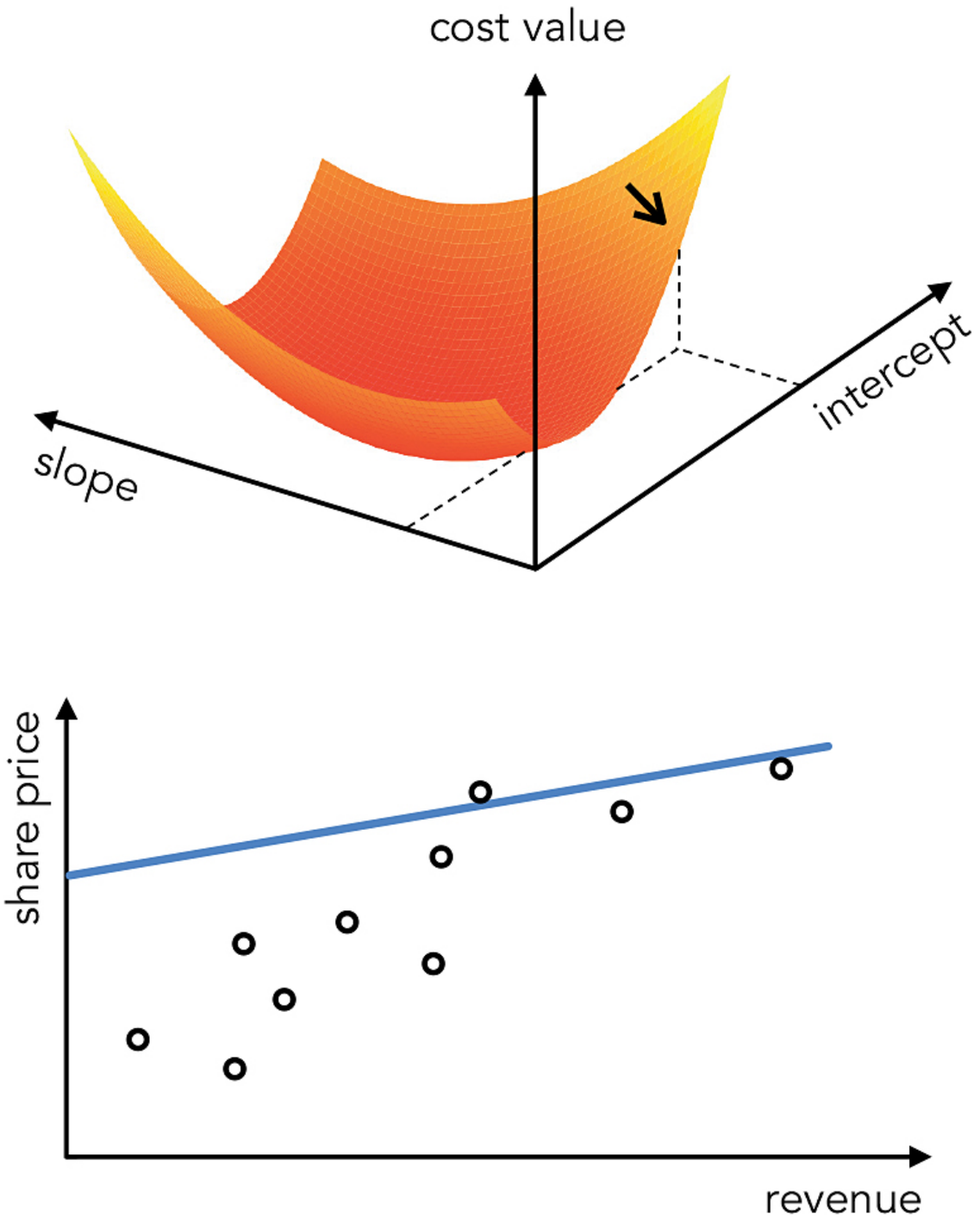
- We have models that are parametrized in our defined feature space.
- We learn the best values for our parameters (also called weights) by minimizing a cost function of the parameters using our training data.
- Minimization is typically done using greedy methods, most commonly, Gradient Descent.

Cost Functions

- The costs functions take in a specific set of model parameters and return a score indicating how well the learning task would be accomplished using those specific parameters.

- As the name suggests, you want to ***minimize*** the cost function. This corresponds to finding the set of parameters that provide the smallest value (minimum) of a cost function.





Minimizing a Cost Function

- The tuning of these parameters require the ***minimization*** *of* ***a cost function*** . It can be formally written as follows.

$$\underset{\mathbf{w}}{\text{minimize}} \quad g(\mathbf{w})$$

- This says formally: look over every possible input w, and find the one that gives the smallest value of g(w).

Optimization

- With general N dimensional input, any N, dimensional point v where every partial derivative of g is zero, that is

$$\frac{\partial}{\partial w_1} g(\mathbf{v}) = 0$$

$$\frac{\partial}{\partial w_2} g(\mathbf{v}) = 0$$

$$\vdots$$

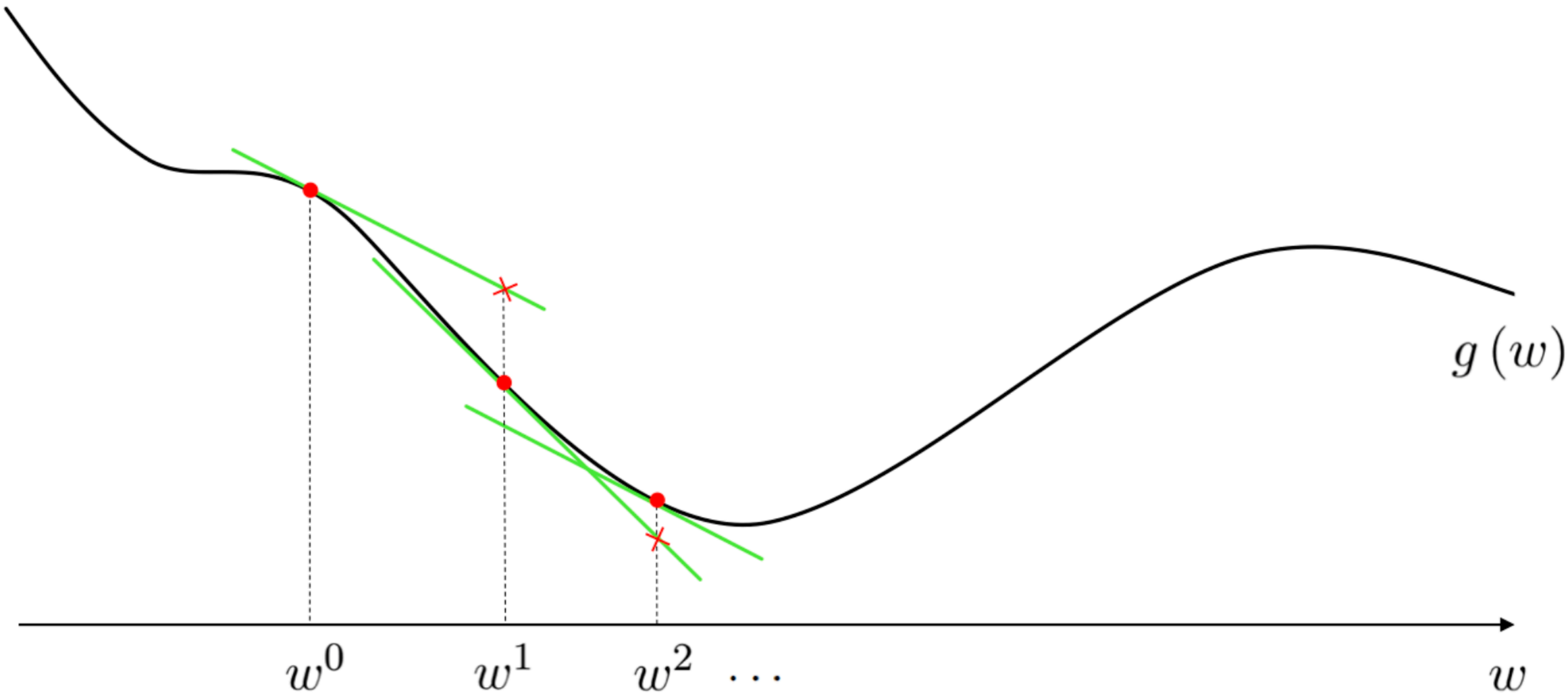
$$\frac{\partial}{\partial w_N} g(\mathbf{v}) = 0$$

is a potential minimum

Optimization

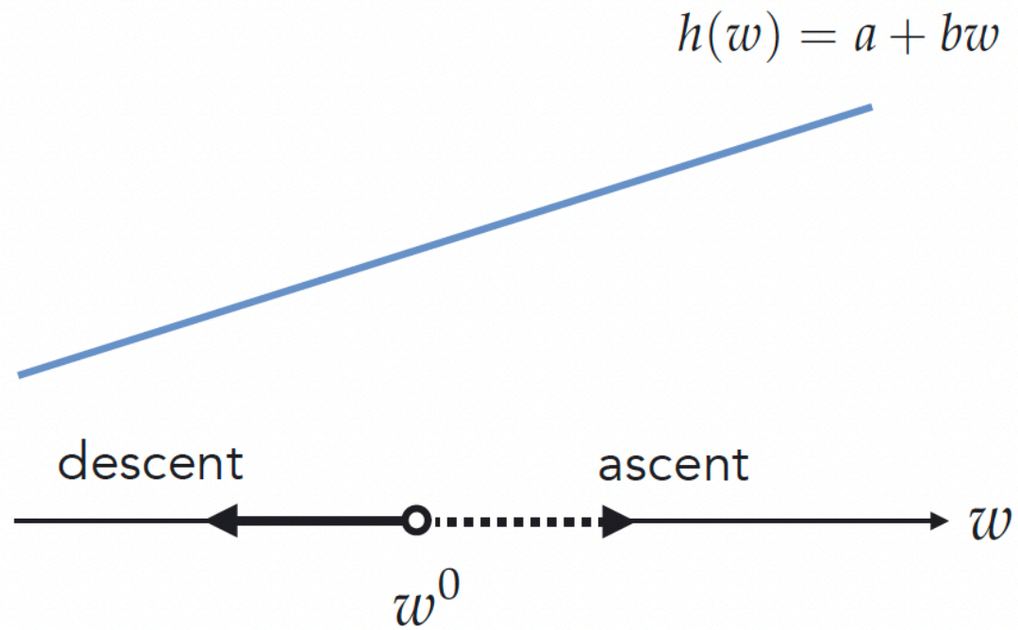
- With few exceptions it is virtually impossible to solve a general function's first order systems of equations 'by hand' (i.e., no closed form solutions).

Gradient Descent



$$\mathbf{w}^k = \mathbf{w}^{k-1} + \alpha \mathbf{d}^k$$

Descent direction



Gradient Descent

$$\mathbf{w}^{k} = \mathbf{w}^{k-1} + \alpha \mathbf{d}^{k}$$

- Setting the descent direction

$$\mathbf{d}^{k} = -\nabla g\left(\mathbf{w}^{k-1}\right)$$

we get

$$\mathbf{w}^{k} = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

Gradient Descent

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

α (learning rate) is an example of a ***hyperparameter***

Gradient Descent

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

α (learning rate) is an example of a ***hyperparameter***

$$w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}$$

these are ***parameters*** of the model (also called **weights**)

Gradient Descent

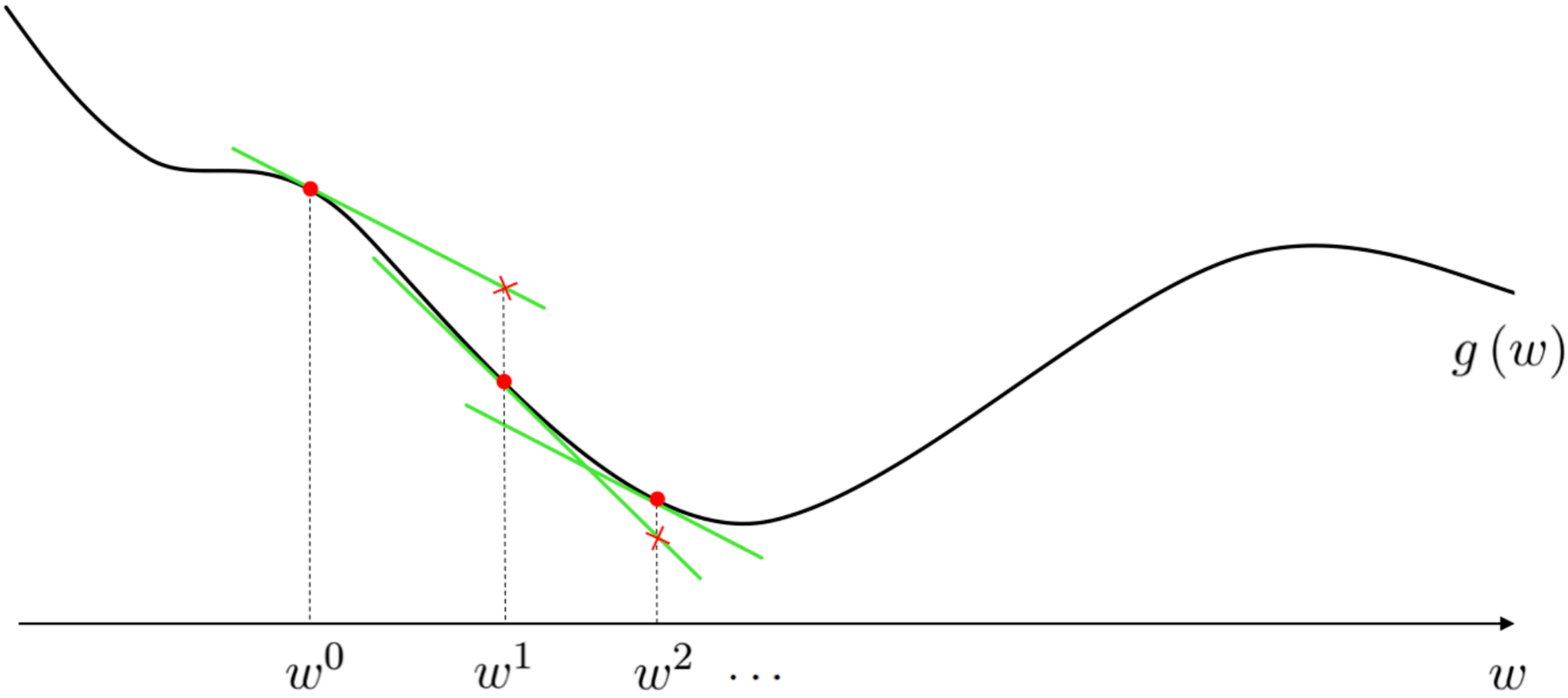
$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

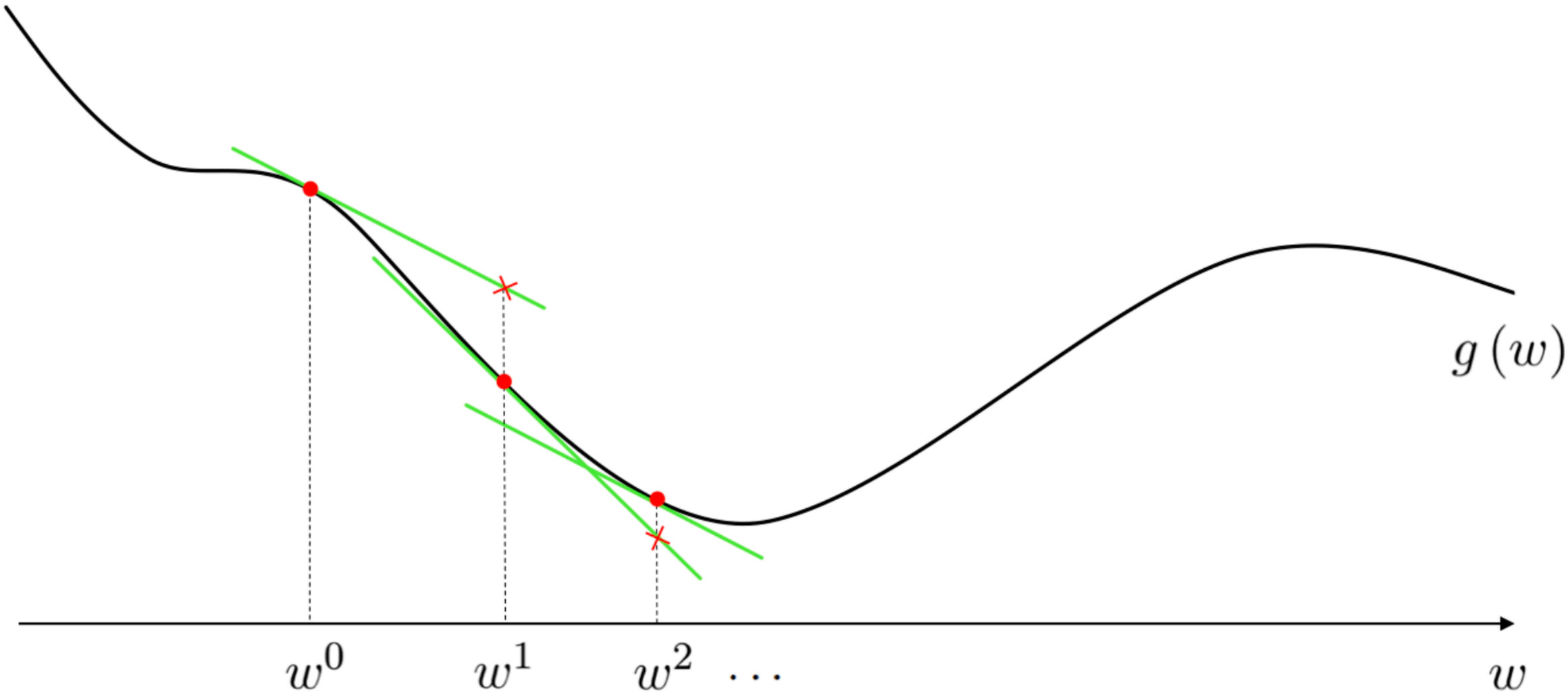
α (learning rate) is an example of a ***hyperparameter***

$$\nabla g(w) = \begin{pmatrix} \frac{\partial g(w)}{\partial w_1} \\ \frac{\partial g(w)}{\partial w_2} \\ \vdots \\ \frac{\partial g(w)}{\partial w_N} \end{pmatrix} \qquad w = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_N \end{pmatrix}$$

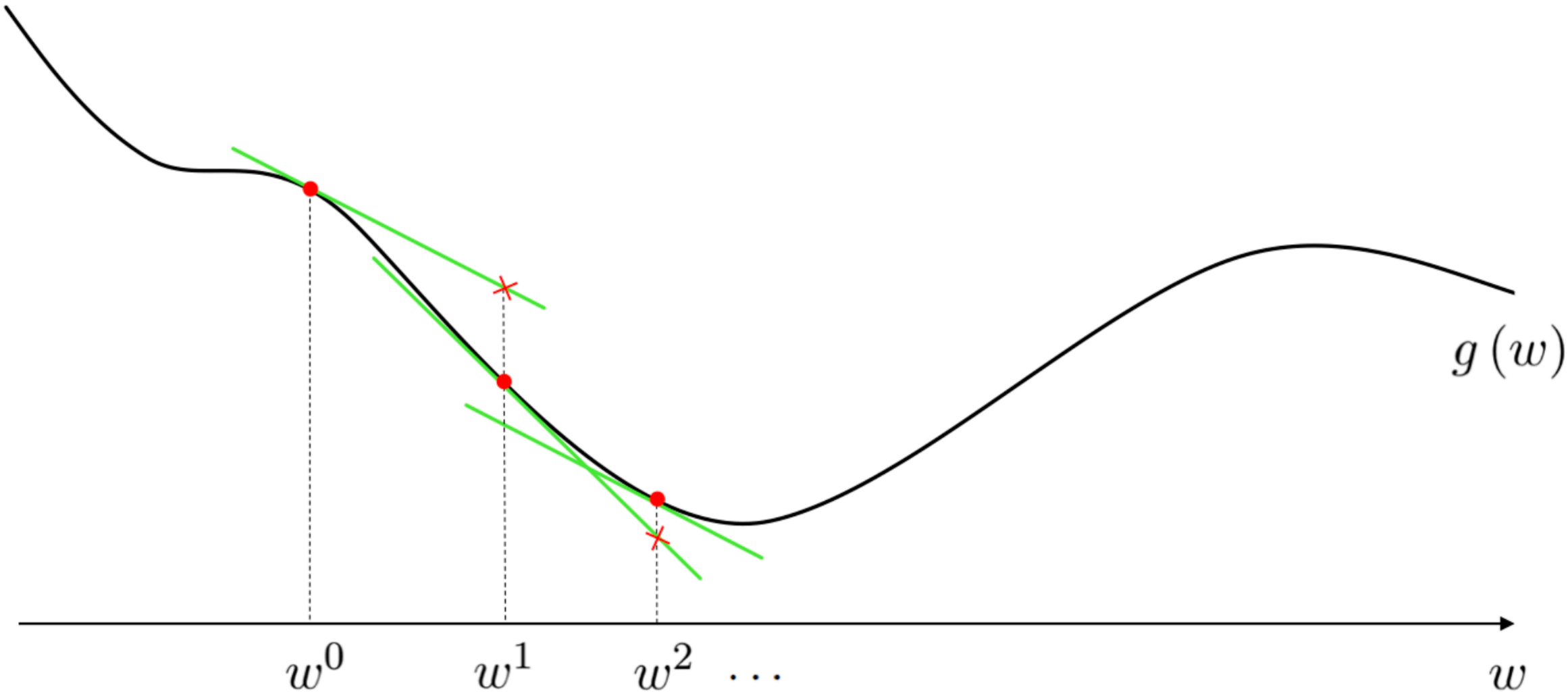
Called the gradient

these are ***parameters*** of the model (also called **weights**)





$$w^1 = w^0 - \alpha \frac{\partial}{\partial w} g\left(w^0\right)$$



$$w^2 = w^1 - \alpha \frac{\partial}{\partial w} g\left(w^1\right)$$

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

Stopping criteria: maximum number of iterations || ?

by hand :

$$g(w) = \frac{1}{50}\left(w^4 + w^2 + 10w\right)$$

$$\underset{\mathbf{w}}{\text{minimize}} \quad g\left(\mathbf{w}\right)$$

$$g(w) = \frac{1}{50}\left(w^4 + w^2 + 10w\right)$$

$$\underset{\mathbf{w}}{\text{minimize}} \quad g\left(\mathbf{w}\right)$$

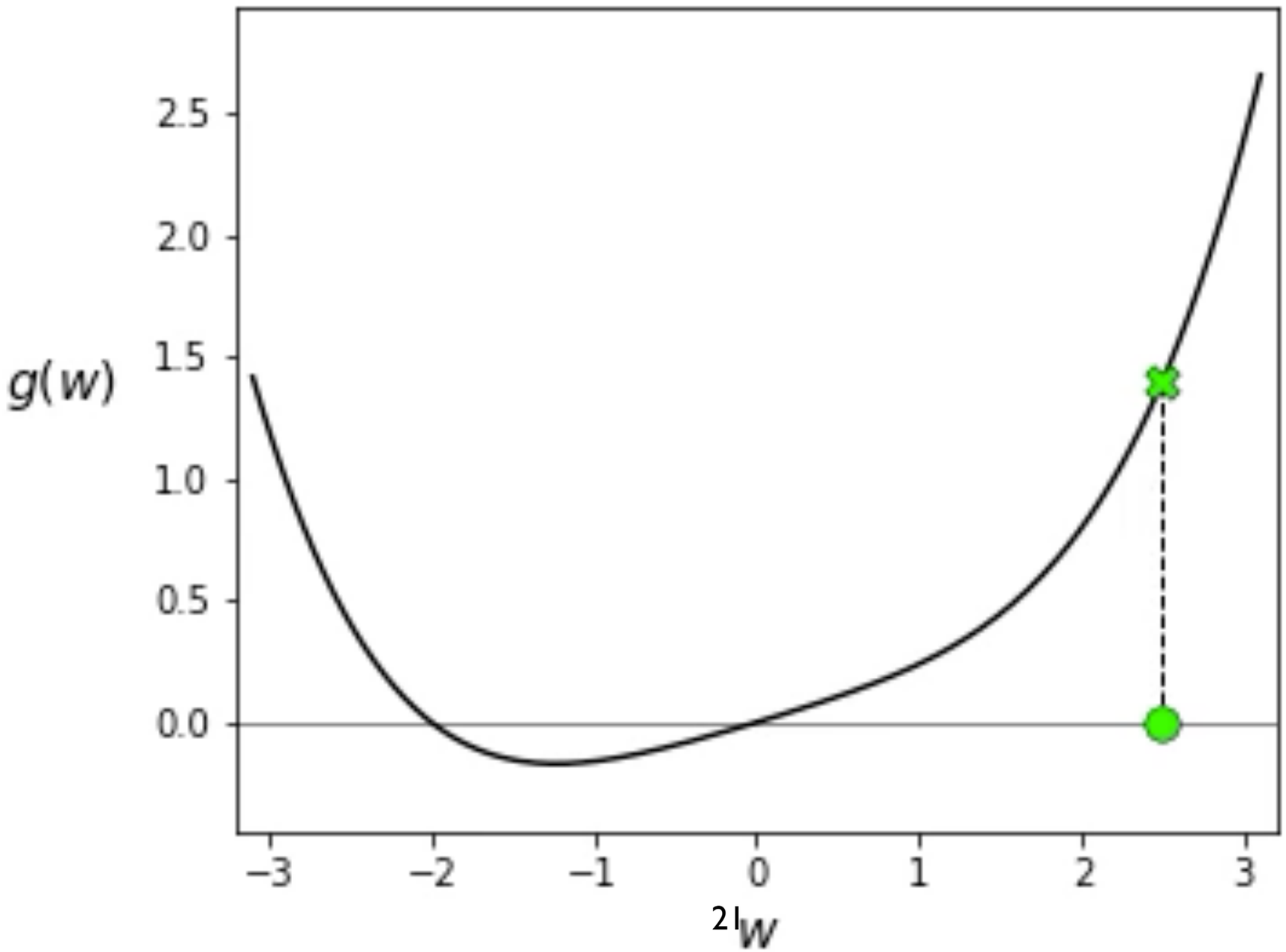
by gradient descent: $K = 5 \quad \alpha = 1 \quad w^0 = 2.5$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

$$g(w) = \frac{1}{50}\left(w^4 + w^2 + 10w\right)$$

$$\underset{\mathbf{w}}{\text{minimize}} \quad g\left(\mathbf{w}\right)$$

by gradient descent: $\alpha = 1 \quad w^0 = 2.5$



The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

How to set w^0 ?

When does it stop?

How to set α ?

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

How to set w^0 ? random initialization

When does it stop?

How to set α ?

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

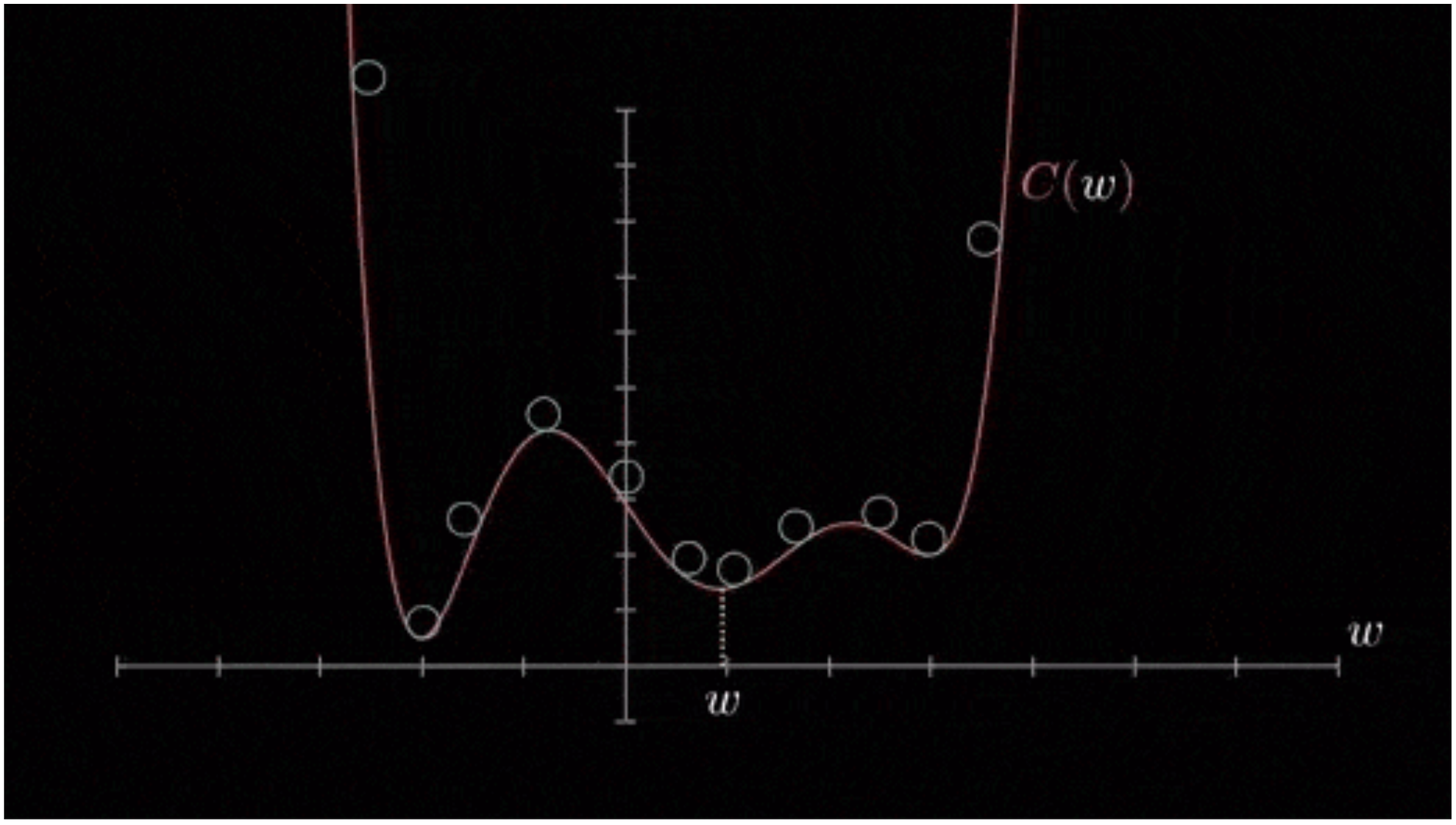
4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

How to set w^0 ? random initialization

When does it stop? when gradient is close to 0, or when max iterations is reached

How to set α ?

Convergence of cost function with different starting points



Source: Gfycat

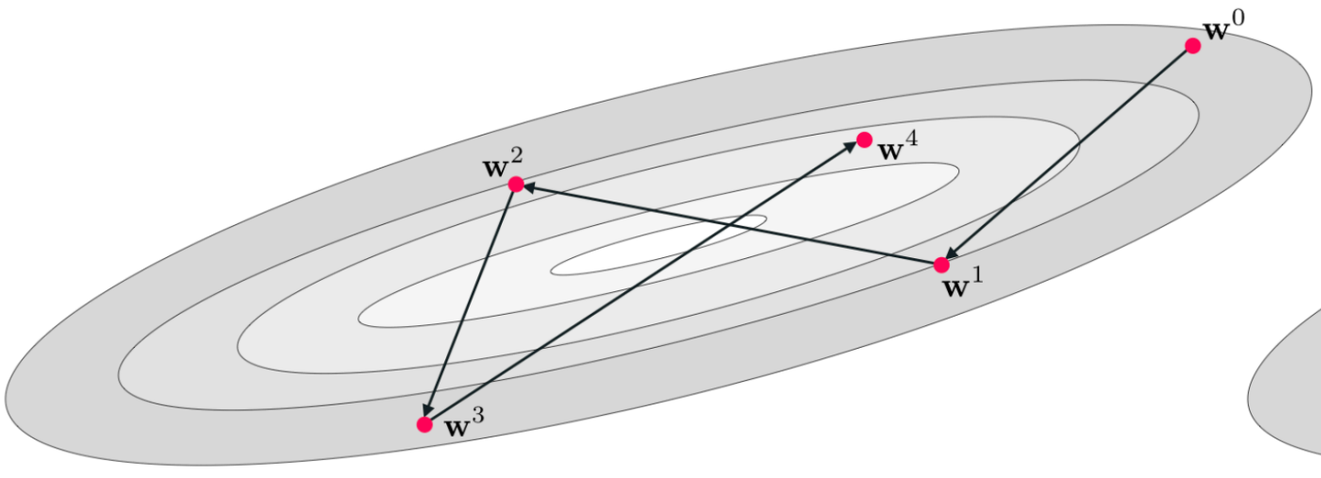
Setting the learning rate, α

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

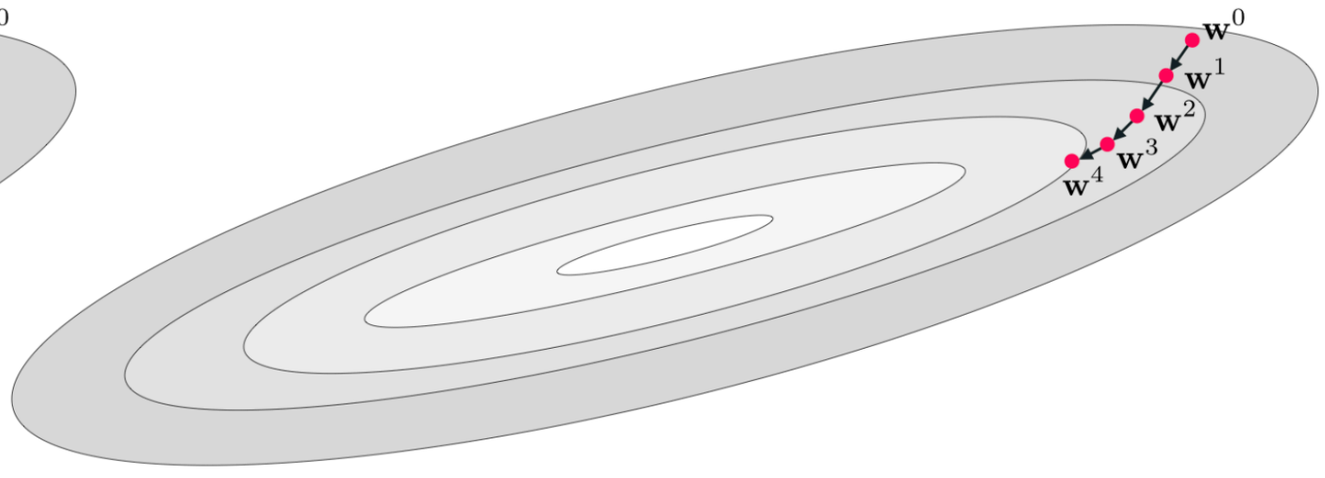
Setting the learning rate, α

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

- If too large, then it can oscillate wildly at each update step never reaching an approximate minimum.
- If they are too small we move so sluggishly slow that far too many steps would be required to reach an approximate minimum.



Too large



Too short

Setting the learning rate parameter, α

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

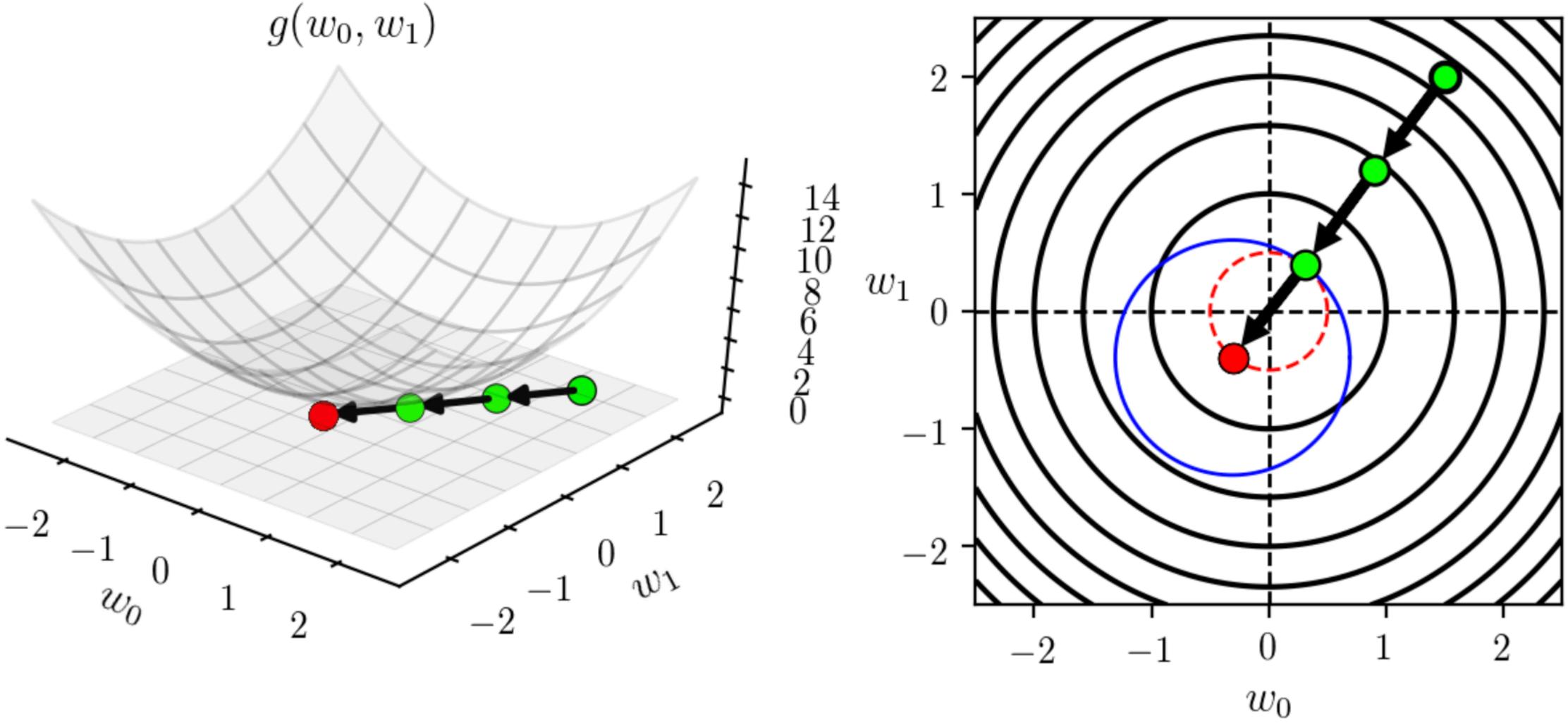
- A common choice is to set α to some fixed small constant value for each of the K steps.
- This choice, to take one value for α and use if to each and every step of the algorithm - is called a ***fixed learning rate*** rule.

Setting the learning rate, α

- We can also change α from step-to-step using a rule: often referred to as an ***adjustable*** learning rate rule.

- The most common adjustable is the ***diminishing*** learning rate rule.

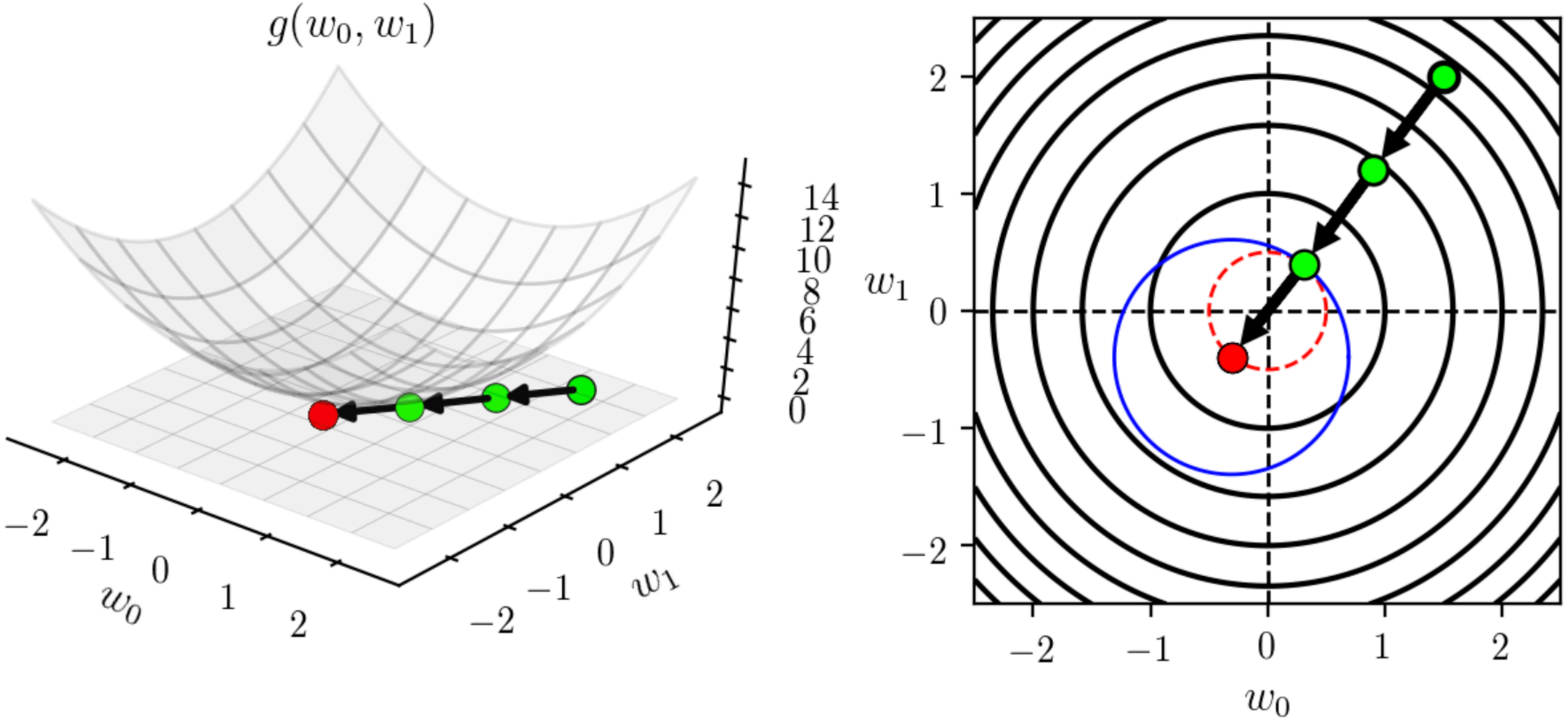
Fixed α



$$\alpha = 1$$

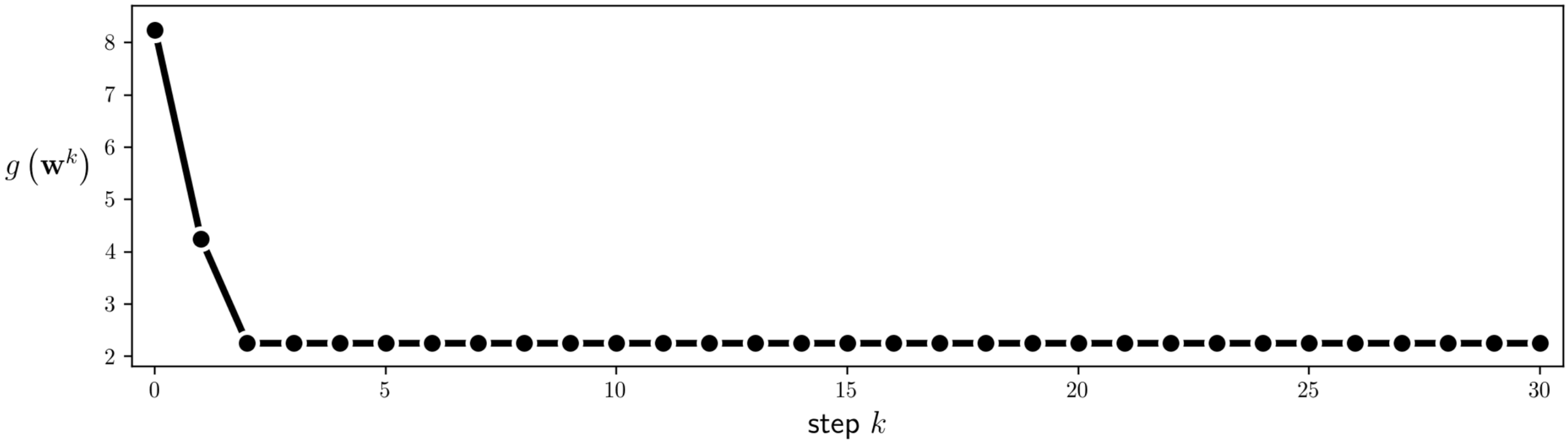
$$K = 3$$

Fixed α



fails to converge to the global minimum

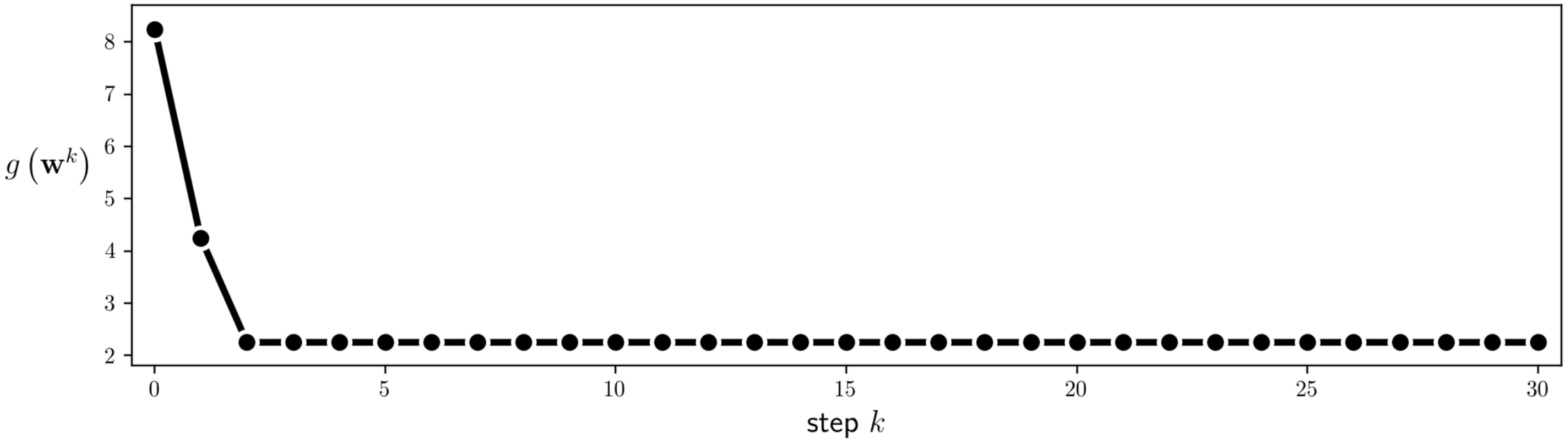
Fixed α



Cost function vs iteration (i.e., steps of gradient descent)

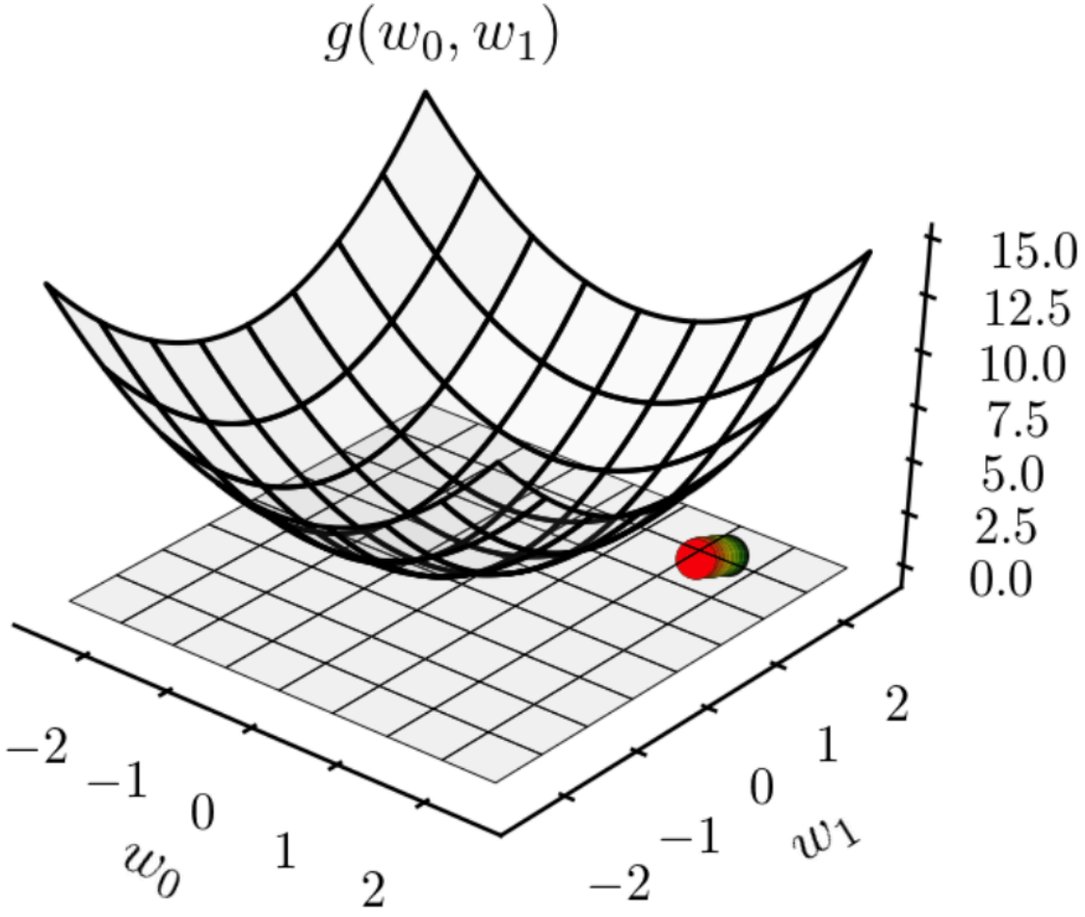
Fixed learning rate, fails to converge to global minimum. What can we do?

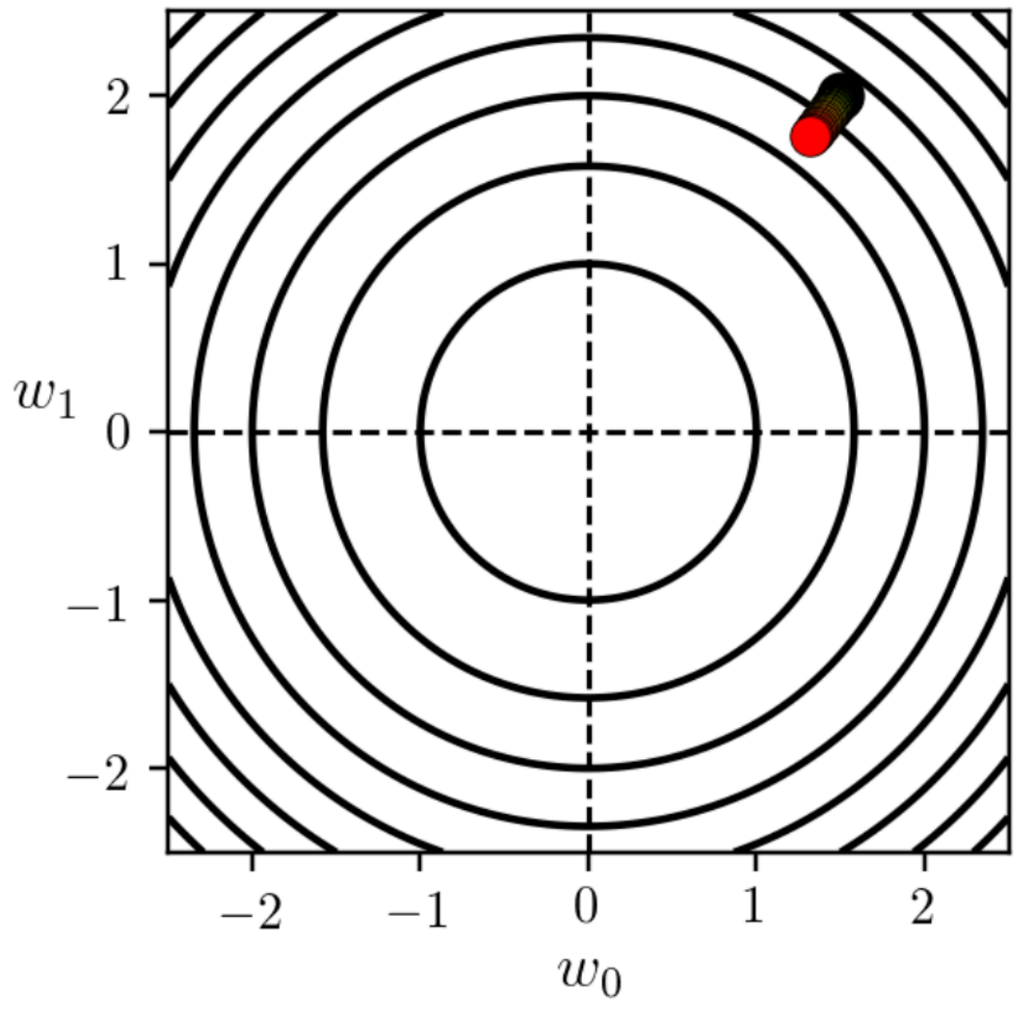
Fixed α



Fixed learning rate, fails to converge to global minimum.
What can we do? Lower the learning rate.

Fixed α

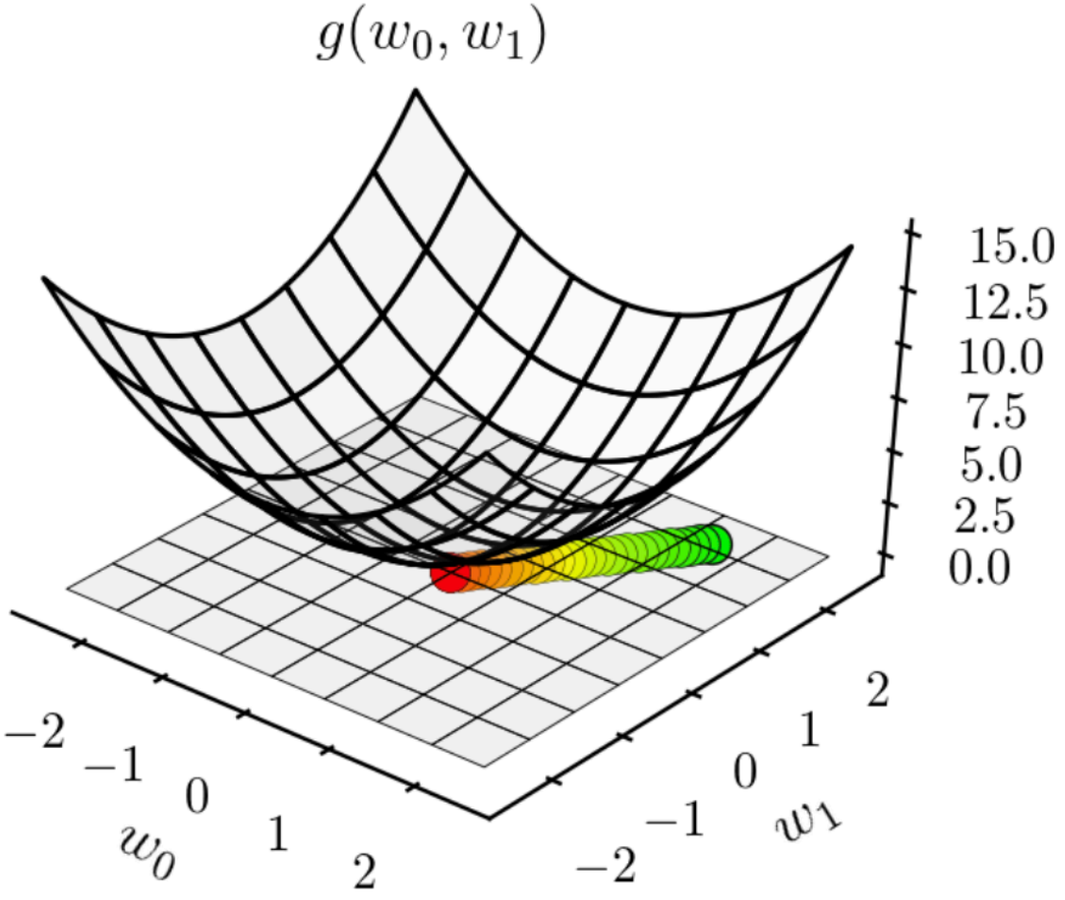


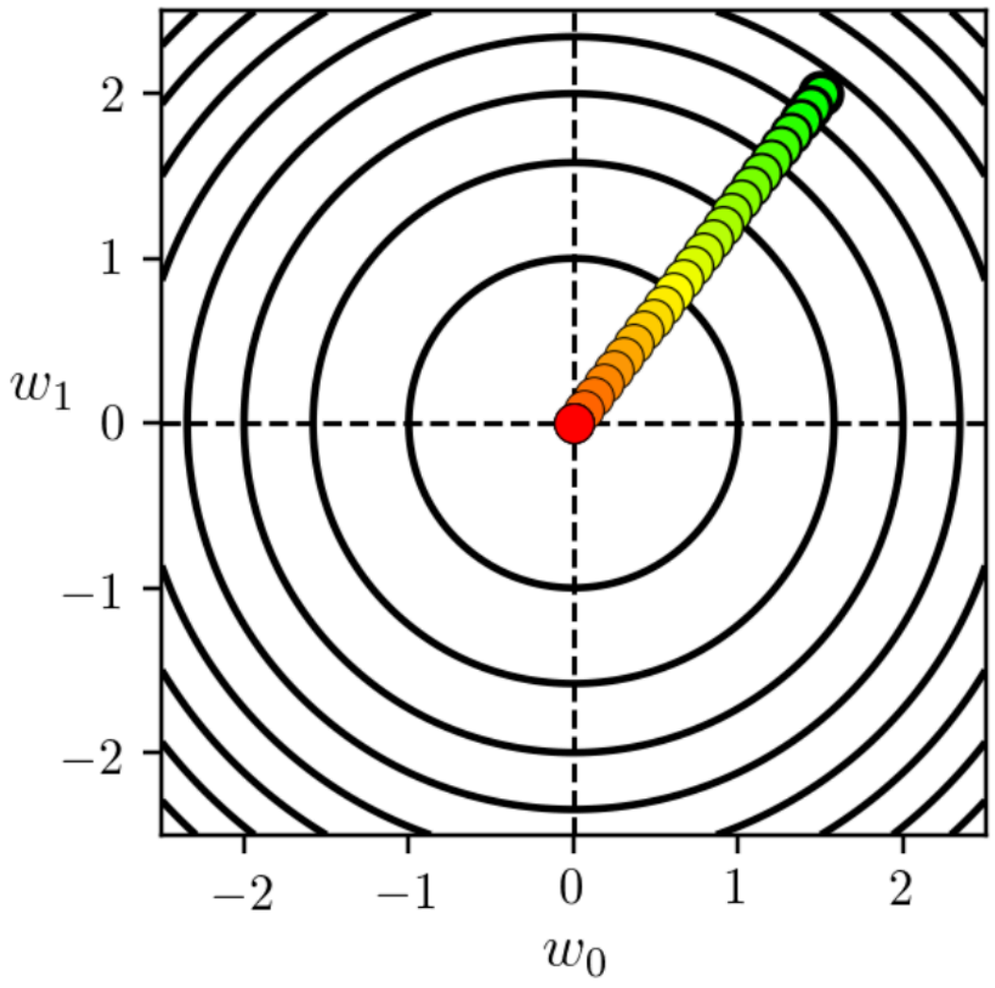


$$\alpha = 0.01$$

$$K = 30$$

Fixed α





$$\alpha = 0.1$$

$$K = 30$$

- The combination of learning rate and maximum number of iterations are best chosen together.
- The trade-off here is simple:
  - A small learning rate combined with a large number of steps can guarantee convergence to towards a local minimum, but can be very computationally expensive.
  - Conversely a large learning rate and small number of maximum iterations can be cheaper but less effective.

Diminishing α

- Diminish the size of the rate at each step.
- Common way of setting it is:

Diminishing α

- Diminish the size of the rate at each step.
- Common way of setting it is:

$$\alpha = \frac{1}{k}$$

at step k of the process

Diminishing α

$$\alpha = \frac{1}{k}$$ at step k of the process

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{1}{k} \nabla g\left(\mathbf{w}^{k-1}\right)$$

Diminishing α

$\alpha = \frac{1}{k}$ at step k of the process

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{1}{k} \nabla g\left(\mathbf{w}^{k-1}\right)$$

What happens to α as k increases?

Diminishing α

$\alpha = \frac{1}{k}$ at step k of the process

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \frac{1}{k} \nabla g\left(\mathbf{w}^{k-1}\right)$$

What happens to α as k increases?

$$\alpha = \frac{1}{k} \longrightarrow 0$$

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

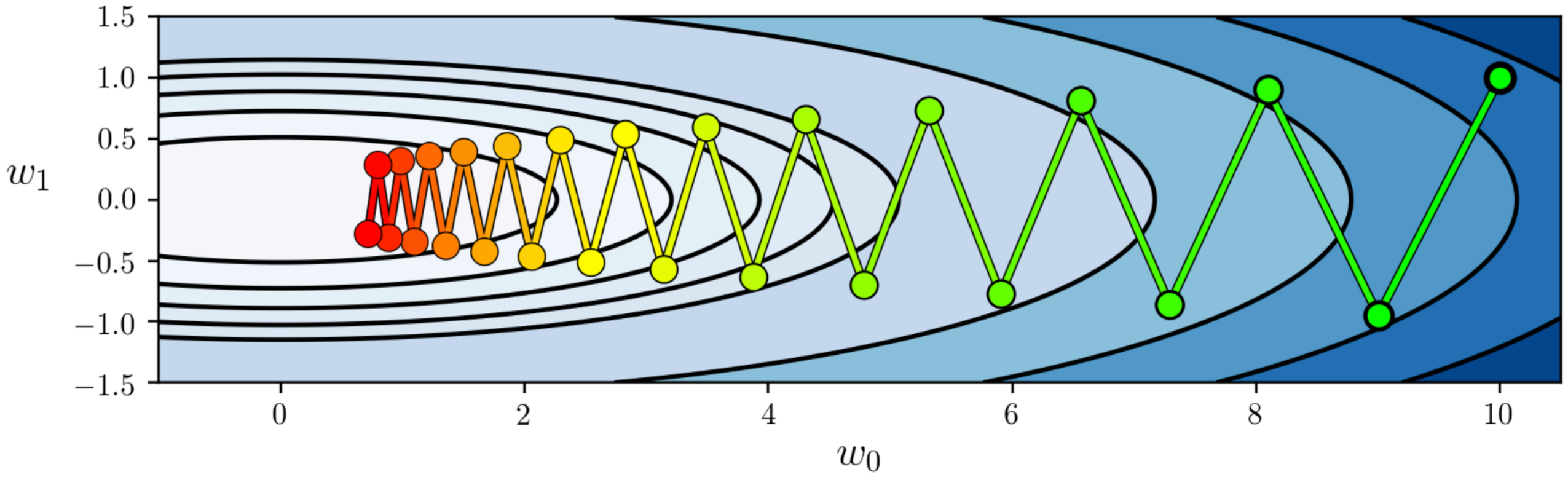
Stopping criteria: maximum number of iterations || step size smaller than a threshold

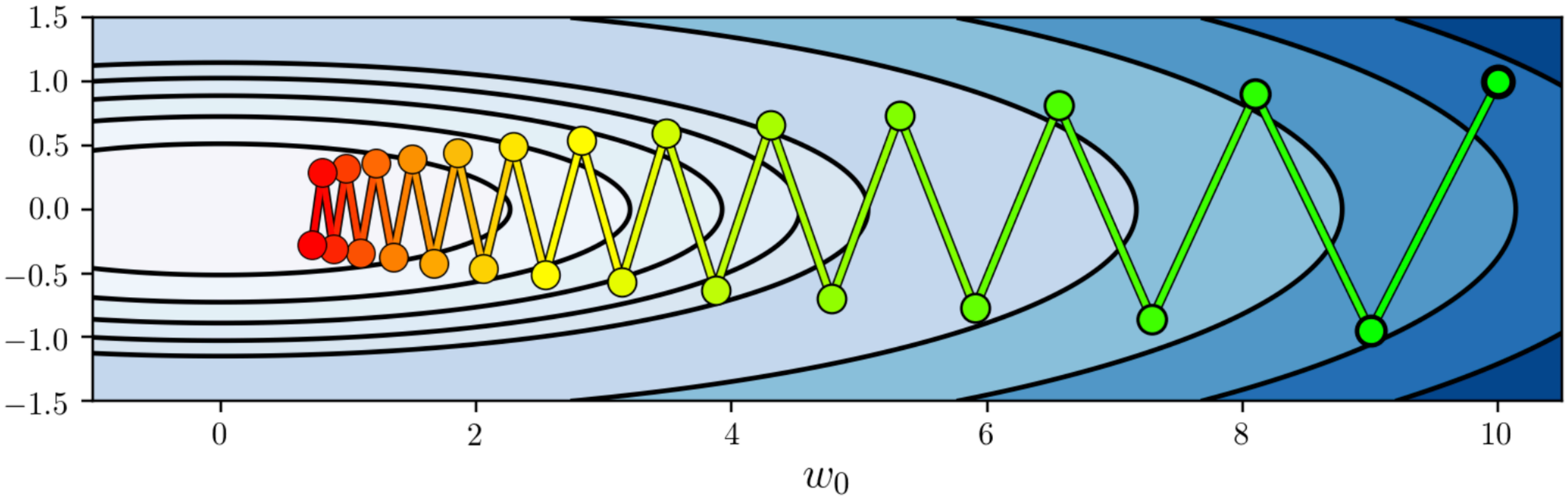
Tuning the learning rate parameter is very much dependent on your task, cost function, and computational power. There is no one-size-fits-all. You need to build up an intuition through practice.

Gradient Descent behaviors to watch out for!

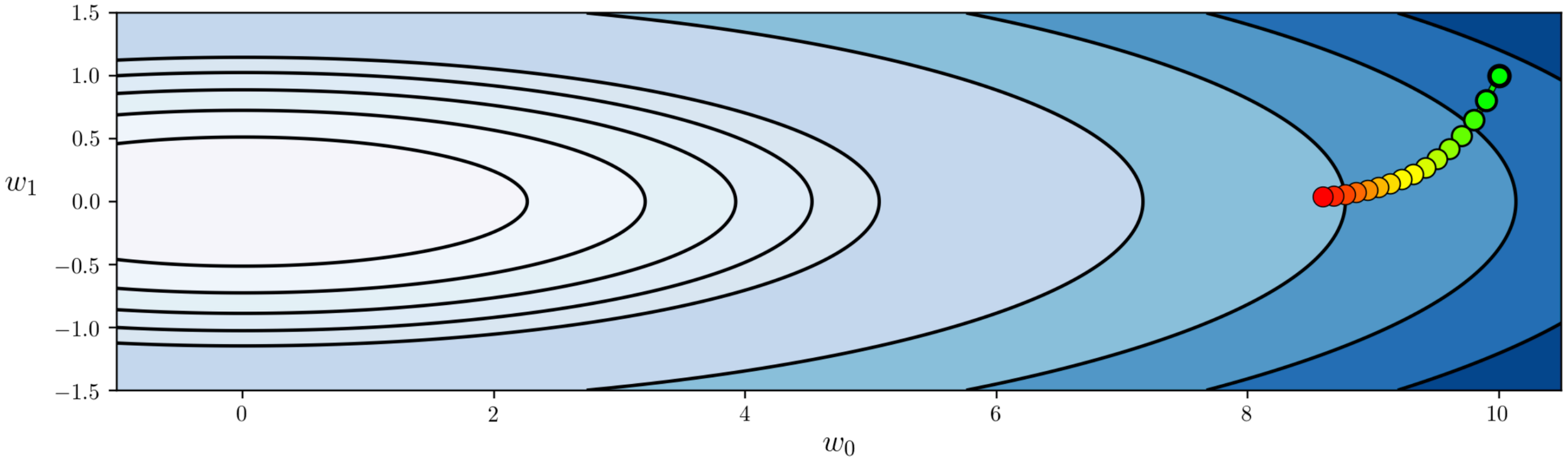
Oscillating Behavior of Gradient Descent

- The negative gradient direction can oscillate rapidly or zig-zag during a run of gradient descent.



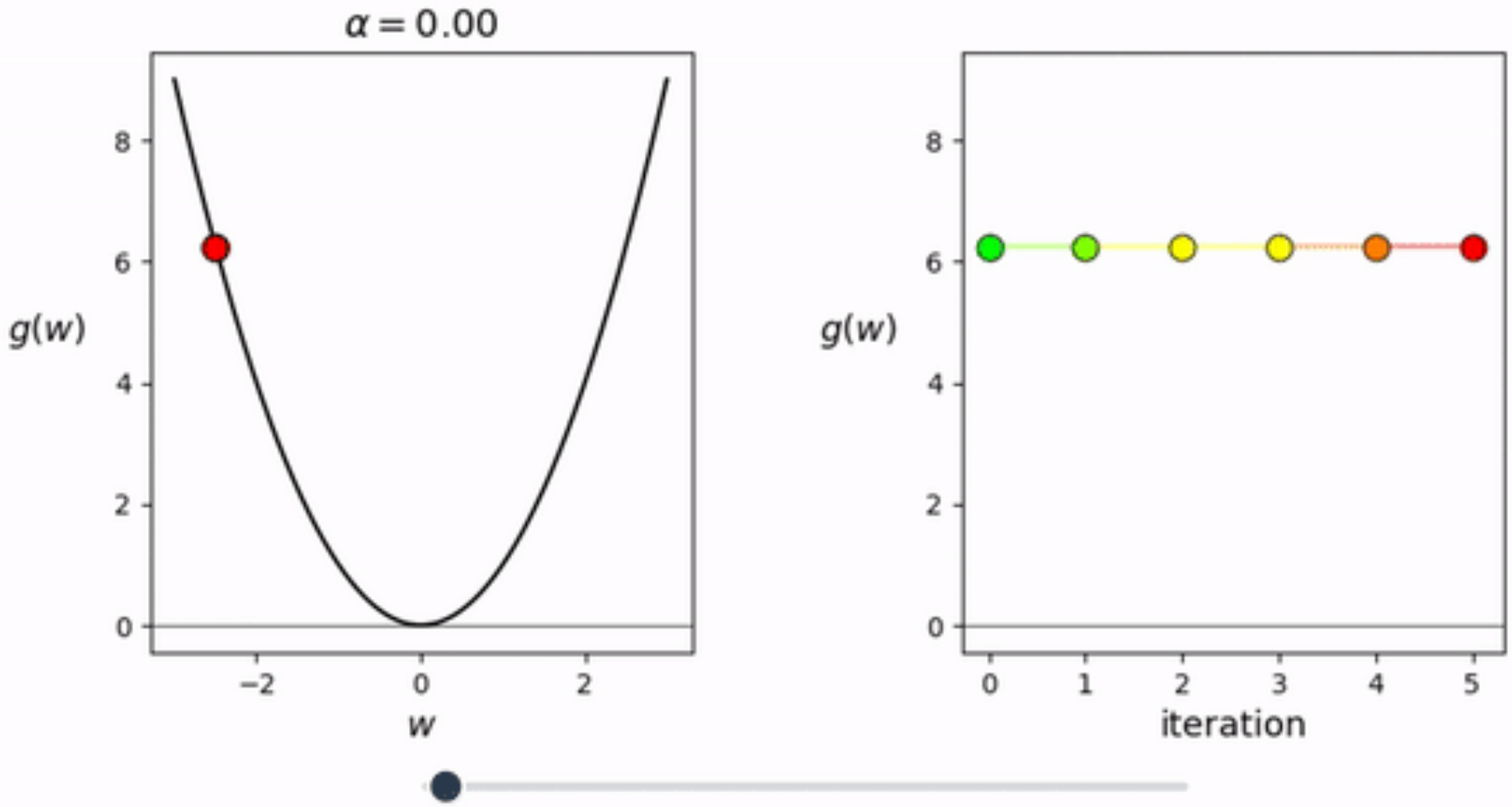


What can we do about this?



reduce learning rate

- Trade-off:
  - A small learning rate combined with a large number of steps can guarantee convergence to towards a local minimum, but can be very computationally expensive.
  - Conversely a large learning rate and small number of maximum iterations can be cheaper but less effective.

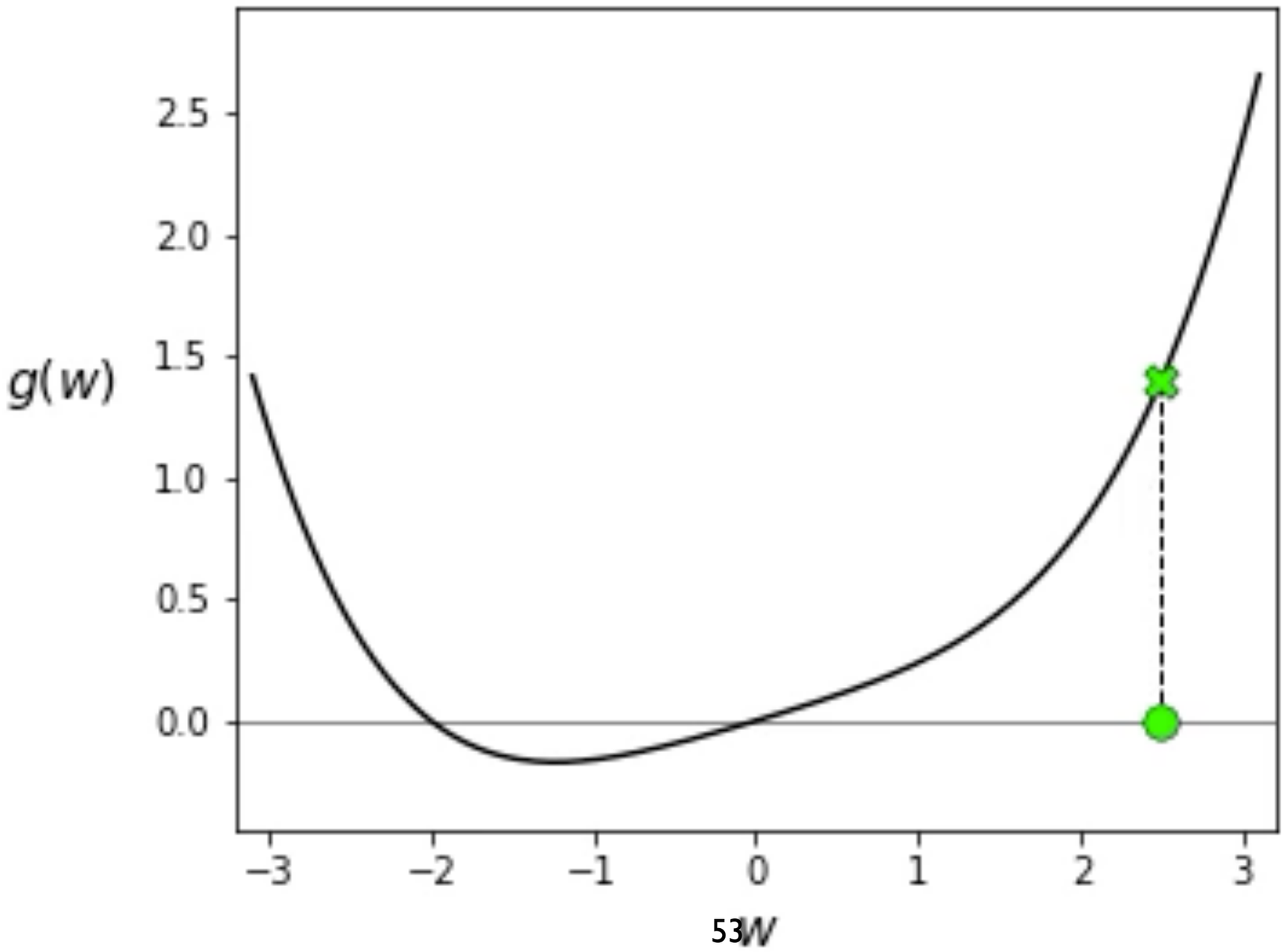


The Slow-crawling Behavior of Gradient Descent (Vanishing Gradients)

- Gradient magnitudes near optima start to vanish (get smaller).
- As a consequence GD progresses very slowly, or 'crawls', near these points.

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

The Slow-crawling Behavior of Gradient Descent (Vanishing Gradients)



Take away: Tuning the learning rate is very important! It is task, and model specific!

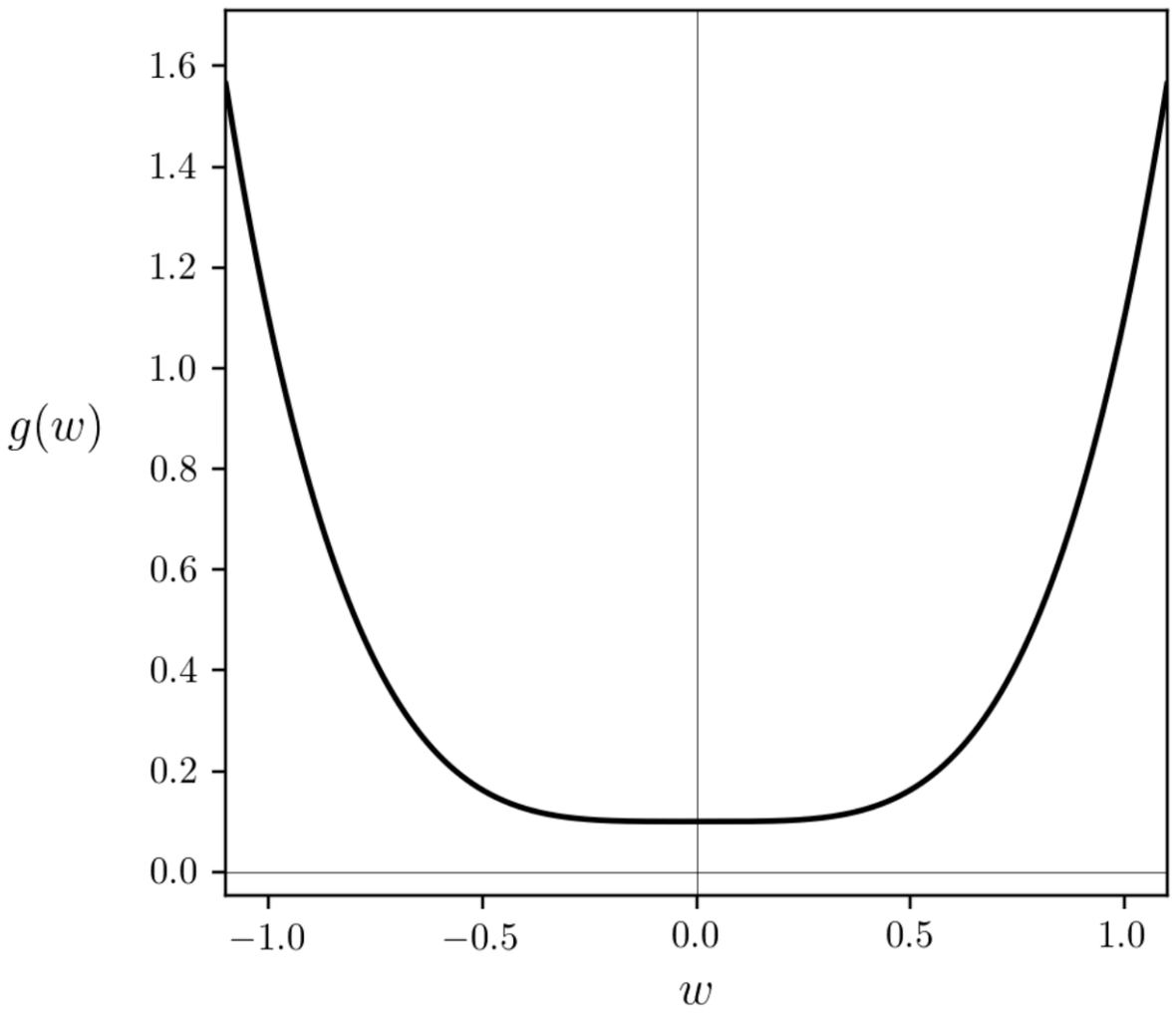
When are you guaranteed to find the global minimum of a function?

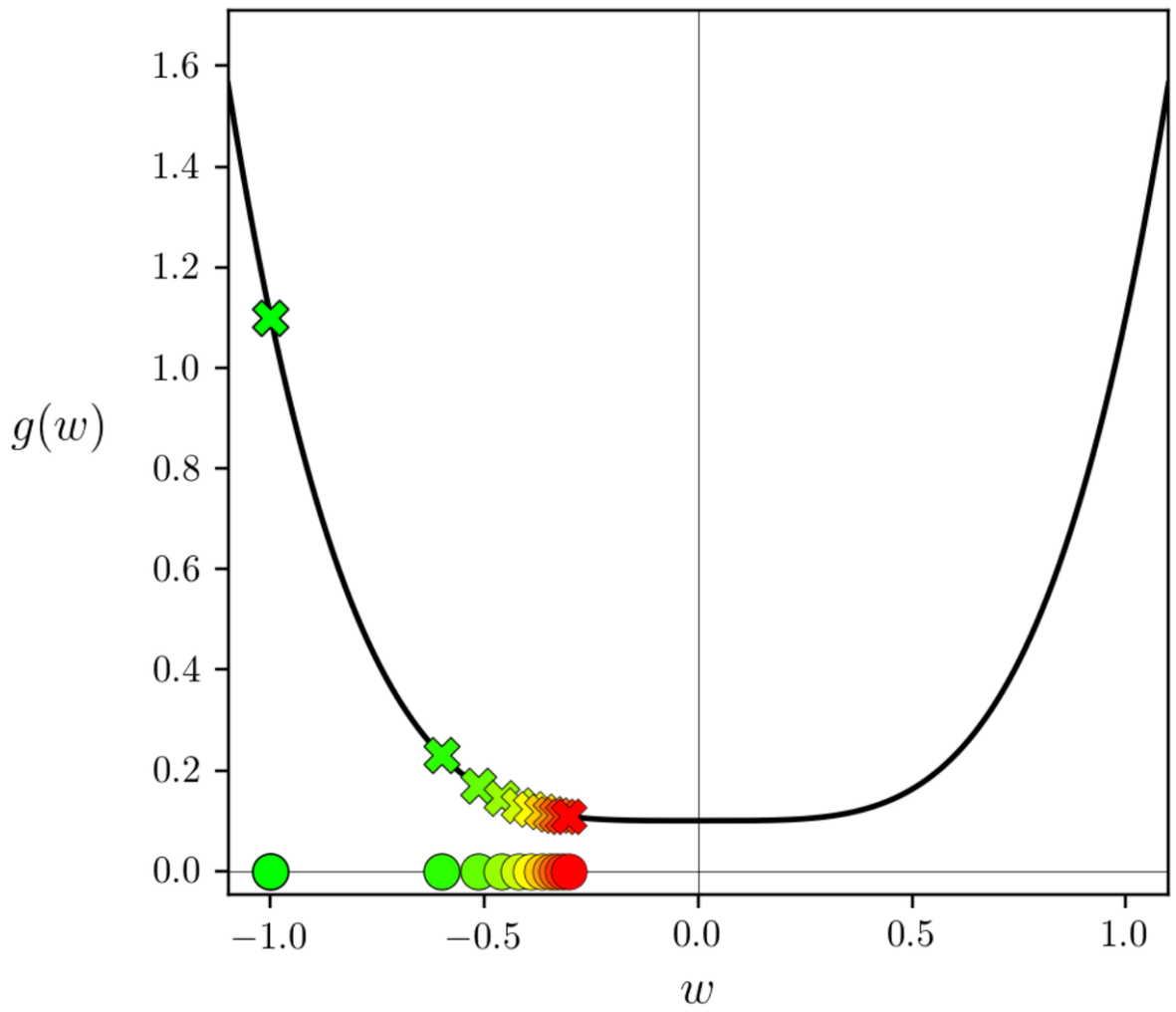
When your function is convex

Convex Functions

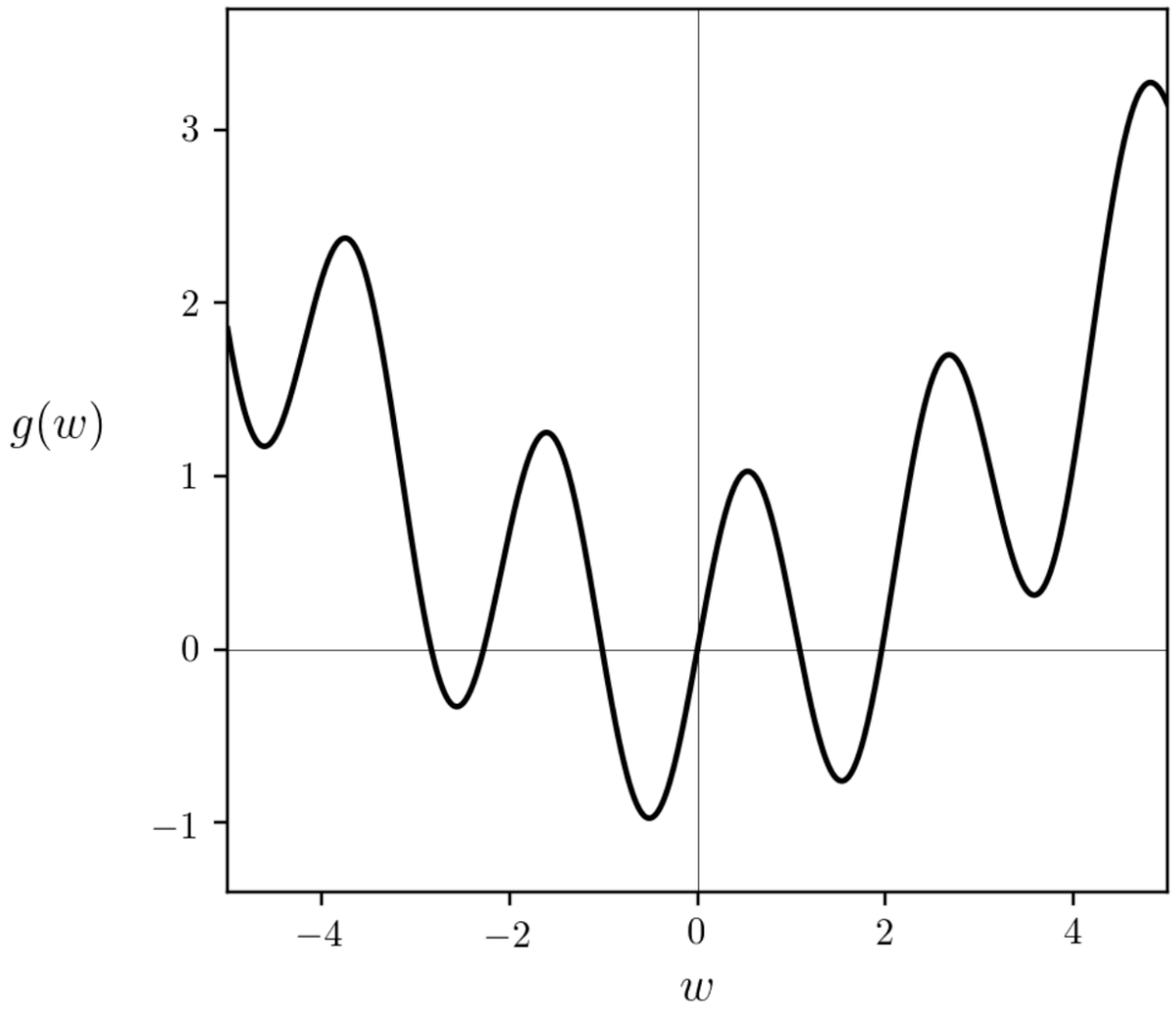
- A function is convex if the line segment between any two points on the graph of the function lies above or on the graph.

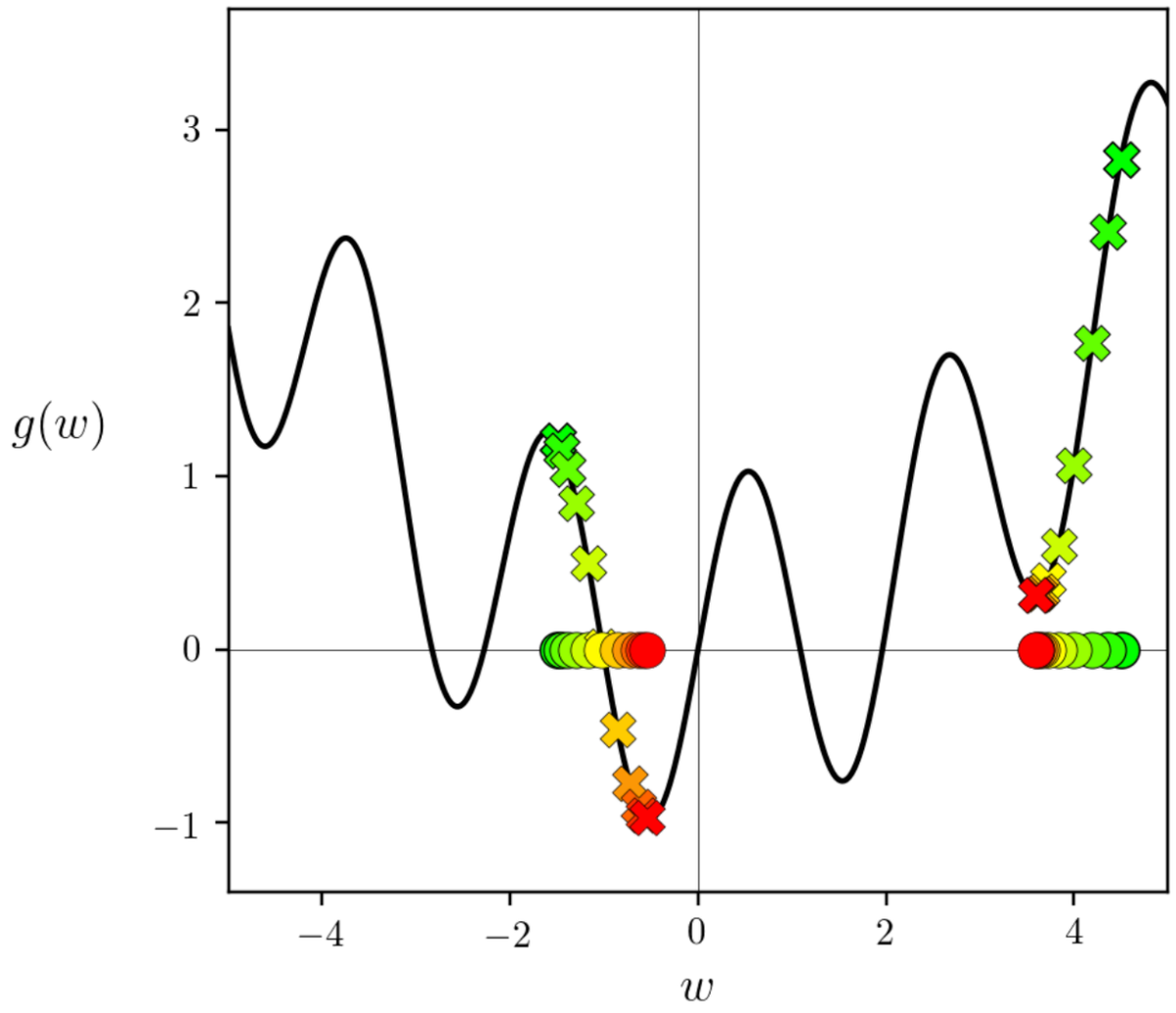
Convex Example





Non-convex Example





Gradient Descent in Python

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

Gradient Descent in Python

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

The autograd library can take in $g(w)$

and return $\nabla g(w)$

```
from autograd import grad

def gradient_descent(g, alpha, max_its, w):
    ....
```

```
from autograd import grad

def gradient_descent(g, alpha, max_its, w):
    ....
```

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

Recap

- We have models that are parametrized in our defined feature space.
- We learn the best values for our parameters (also called weights) by minimizing a cost function of the parameters using our training data.
- Minimization is typically done using greedy methods, most commonly, Gradient Descent.
- Gradient Descent is a learning algorithm that has its own hyper-parameters, most importantly the learning rate, which needs to be tuned to allow for optimal learning. (Fast Convergences)

Least Squares Linear Regression

- Data for regression problems comes in the form of a set of P input/output observation pairs:
- More compactly this is (a set of input-output tuples):

$$(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_P, y_P)$$

$$\{(\mathbf{x}_p, y_p)\}_{p=1}^{P}$$

p as an index

‘p’ as in data point

- Each input $\mathbf{x}_p$ may be a vector of length N

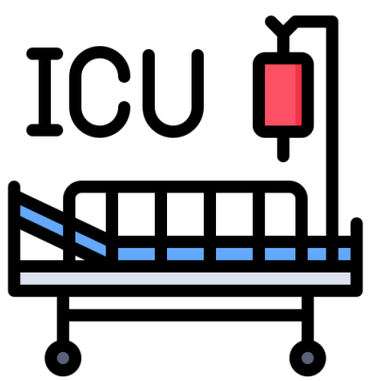
$$\mathbf{x}_p = \begin{bmatrix} x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix}$$

Recall these are your ***features***

- Each input $\mathbf{x}_p$ may be a vector of length N

$$\mathbf{x}_p = \begin{bmatrix} x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix}$$

Recall these are your ***features***



ICU mortality risk prediction:
age, pre-existing condition, organ failure, ventilator use, sepsis

- Each input $\mathbf{x}_p$ may be a vector of length N

$$\mathbf{x}_p = \begin{bmatrix} x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix}$$

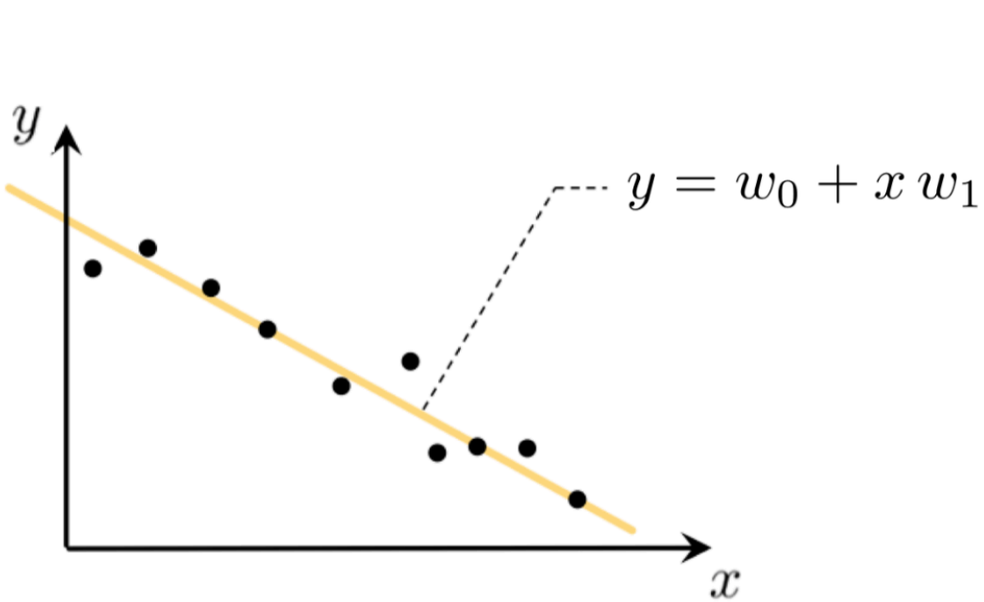
Recall these are your ***features***

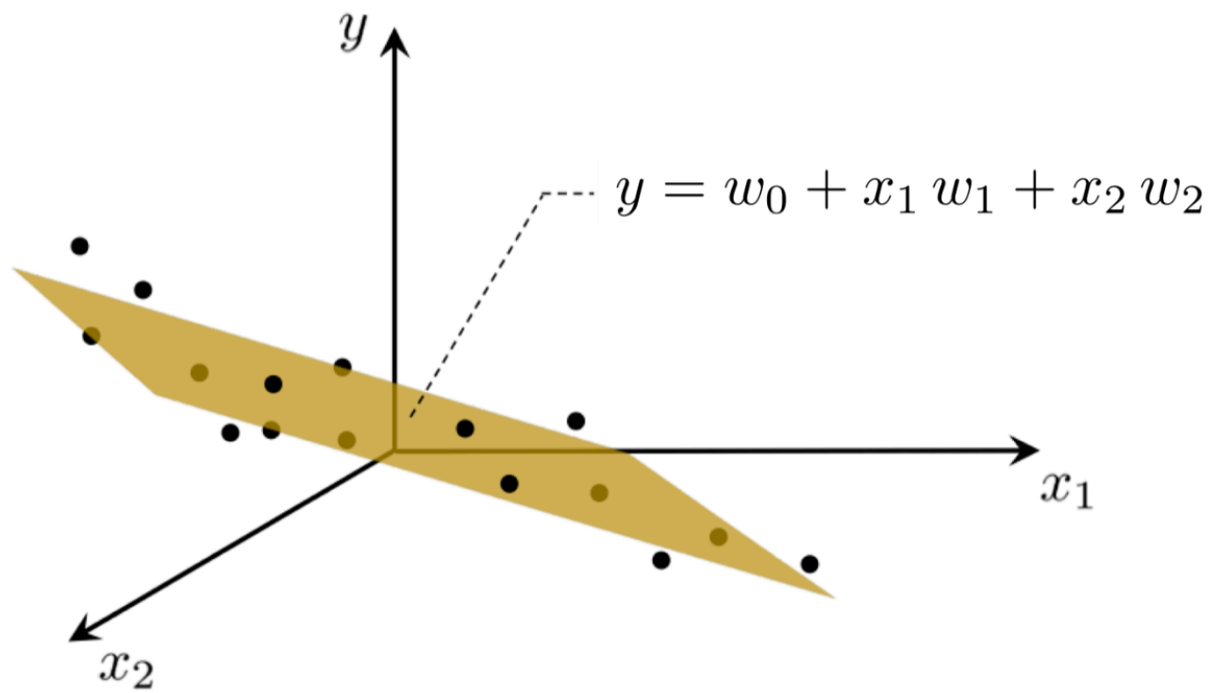
Course project score prediction (n=75): time spent by team, number of missing classes, office hour attendance, grad_vs_undergrad, …

- Each input $\mathbf{x}_p$ may be a vector of length N

$$\mathbf{x}_p = \begin{bmatrix} x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix}$$

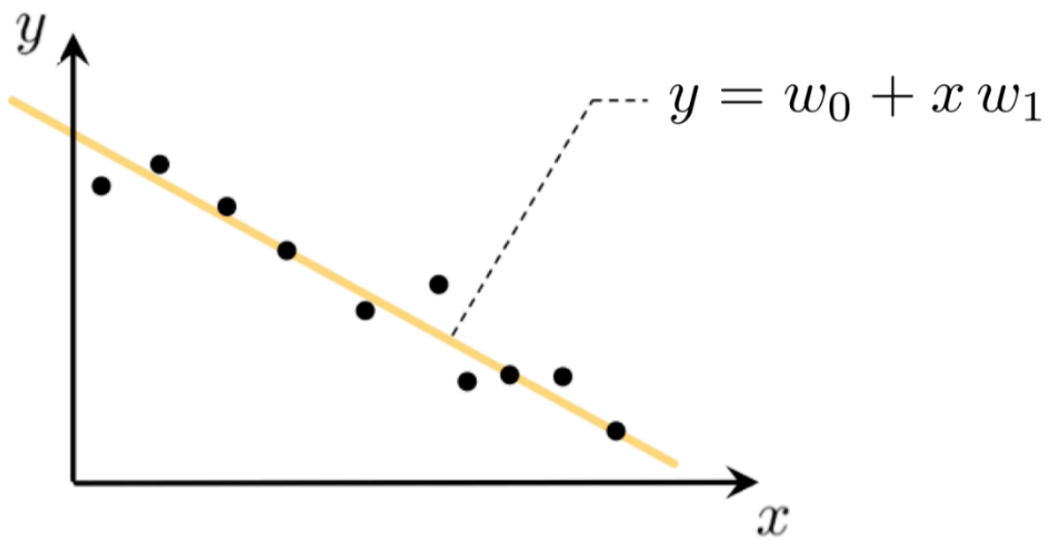
- The linear regression: fitting a hyperplane to our data points in N+1 dimensional space.





When N=1 (1 feature)

$$w_0 + x_p w_1 \approx y_p, \qquad p = 1, \ldots, P.$$



- When dealing with N dimensional input (corresponding to N features) we have a bias (also referred to as the intercept) and N associated weights/parameters to tune properly.

$$w_0 + x_{1,p} w_1 + x_{2,p} w_2 + \cdots + x_{N,p} w_N \approx y_p, \quad p = 1, \ldots, P.$$

- When dealing with N dimensional input (corresponding to N features) we have a bias (also referred to as the intercept) and N associated weights/parameters to tune properly.

$$w_0 + x_{1,p}w_1 + x_{2,p}w_2 + \cdots + x_{N,p}w_N \approx y_p, \quad p = 1, \ldots, P.$$

$x_{1,p}, x_{2,p}, \ldots, x_{N,p}$ these are your features

$w_1, w_2, \ldots, w_N$ these are the weights of your features

w_0 this is the bias

- For any N we can write the above more compactly - in particular using the notation $\mathring{\mathbf{x}}$ to denote an input $\mathbf{x}$ with a 1 placed on top of it as

$$\mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_N \end{bmatrix} \qquad \mathring{\mathbf{x}} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}$$

$$\mathring{\mathbf{x}}_p = \begin{bmatrix} 1 \\ x_{1,p} \\ x_{2,p} \\ \vdots \\ x_{N,p} \end{bmatrix}, \quad p = 1, \ldots, P$$

$$w_0 + x_{1,p}w_1 + x_{2,p}w_2 + \cdots + x_{N,p}w_N \approx y_p, \quad p = 1, \ldots, P.$$

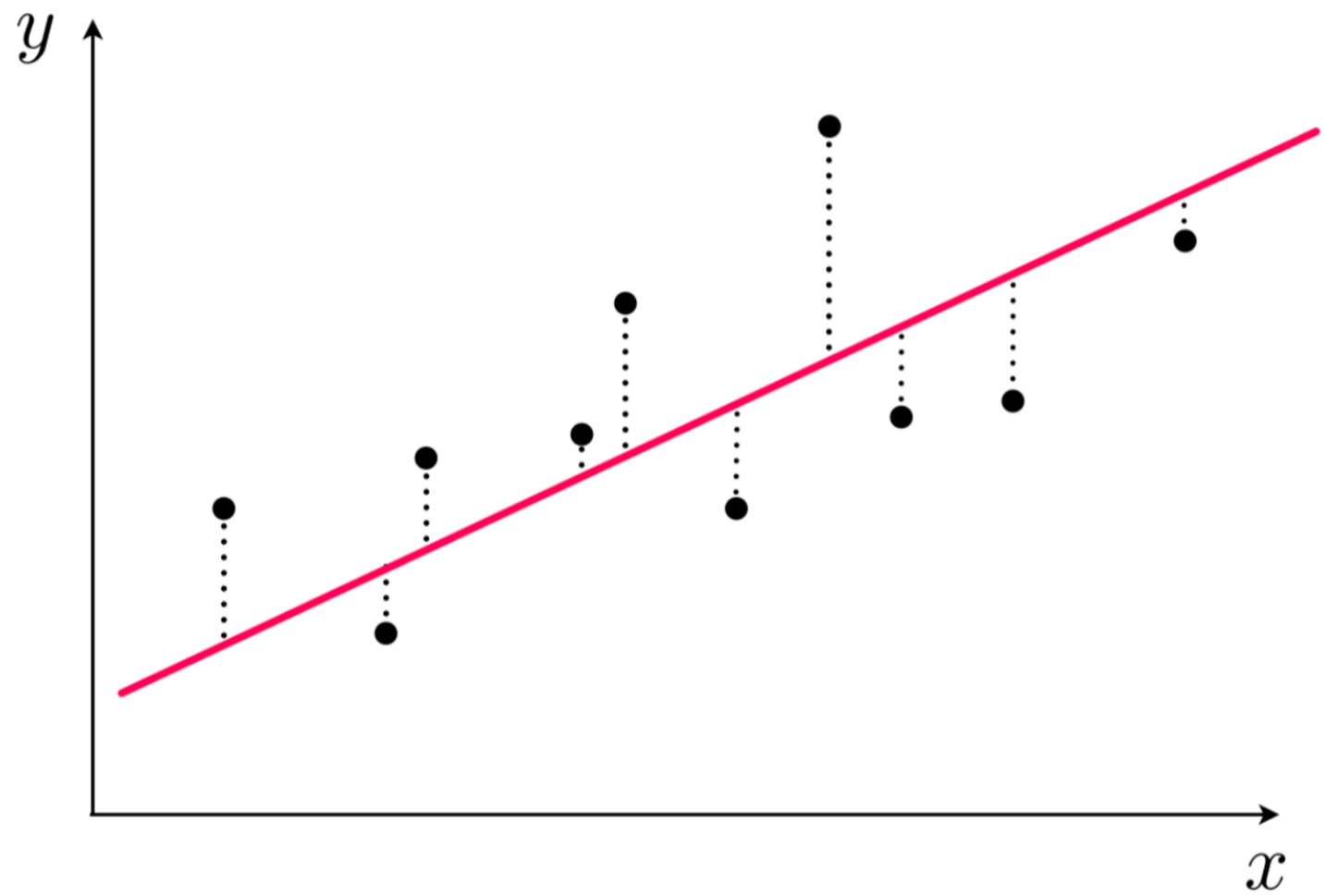
becomes

$$\mathring{\mathbf{x}}_p^T \mathbf{w} \approx y_p \quad p = 1, \ldots, P.$$

The Least Squares Cost Function

- Your first "real" cost function.
- For a given set of parameters $\mathbf{w}$ this cost function computes the total squared error between the associated hyperplane and the data.

The Least Squares Cost Function



Naturally then the best fitting hyperplane is the one whose parameters minimize this error

- Where does this "Least Squares" cost come from?

- Remember, we want to find a weight vector $\mathbf{w}$ so that each of P approximate equalities below holds as tightly as possible:

$$\mathring{\mathbf{x}}_p^T \mathbf{w} \approx y_p \quad p = 1, \ldots, P.$$

- Another way of stating the above is to say that the error between

$$\mathring{\mathbf{x}}_p^T \mathbf{w} \quad \& \; y_p$$

is small

- One natural way to measure error between two quantities like this measure its *square* (so that both negative and positive errors are treated equally) as:

$$g_p\left(\mathbf{w}\right) = \left(\overset{\circ}{\mathbf{x}}_p^T \mathbf{w} - y_p\right)^2$$

- One natural way to measure error between two quantities like this measure its *square* (so that both negative and positive errors are treated equally) as:

$$g_p(\mathbf{w}) = \left(\overset{\circ}{\mathbf{x}}_p^T \mathbf{w} - y_p\right)^2$$

Predicted value

Actual value

- One natural way to measure error between two quantities like this measure its *square* (so that both negative and positive errors are treating equally) as:

$$g_p(\mathbf{w}) = \left(\mathring{\mathbf{x}}_p^T \mathbf{w} - y_p\right)^2$$

Predicted value

Actual value

This is an example of a ***point-wise cost*** that measures the error of a model (here a linear one) on each point

- Since we want all P such values to be small we can take their average - forming a ***Least Squares cost function*** for linear regression:

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P} g_p(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\mathring{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

Note that the Least Squares cost function is not just a function of the weights w but of the data as well.

$$g\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P} g_p\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P}\left(\mathring{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

For notational simplicity we express the function in mathematical shorthand as $g(\mathbf{w})$ (also because we are given the data and are learning the weights)

$$g\left(\mathbf{w}; \left\{\mathring{\mathbf{x}}_p,\ y_p\right\}_{p=1}^{P}\right)$$

We will make this sort of notational simplification for virtually all future machine learning cost functions we study.

We will refer to all cost functions as: $g(\mathbf{w})$

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P} g_p(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\mathring{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

we want to tune our parameters/weights $\mathbf{w}$

to minimize the Least Squares cost:

$$\underset{\mathbf{w}}{\text{minimize}}\ \frac{1}{P}\sum_{p=1}^{P}\left(\mathring{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

$$\underset{\mathbf{w}}{\text{minimize}} \ \frac{1}{P} \sum_{p=1}^{P} \left(\overset{\circ}{\mathbf{x}}_p^T \mathbf{w} \ - y_p \right)^2$$

you should know by now how to solve this by now!

- When implementing a cost function like Least squares it is helpful to think modularly:
  - We have our ***model*** - a linear combination of input.
  - And the ***cost*** (squared error) itself.

- Our model is a function of what?

- Our model is a function of what?
  - Our input features (the data).
  - The weights.

$$\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) = \mathring{\mathbf{x}}_p^T \mathbf{w}$$

$$\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) = \mathring{\mathbf{x}}_p^T \mathbf{w}$$

We are trying to find the ideal settings of weights such that:

$$\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) \approx y_p$$

The Least Squares Cost Function

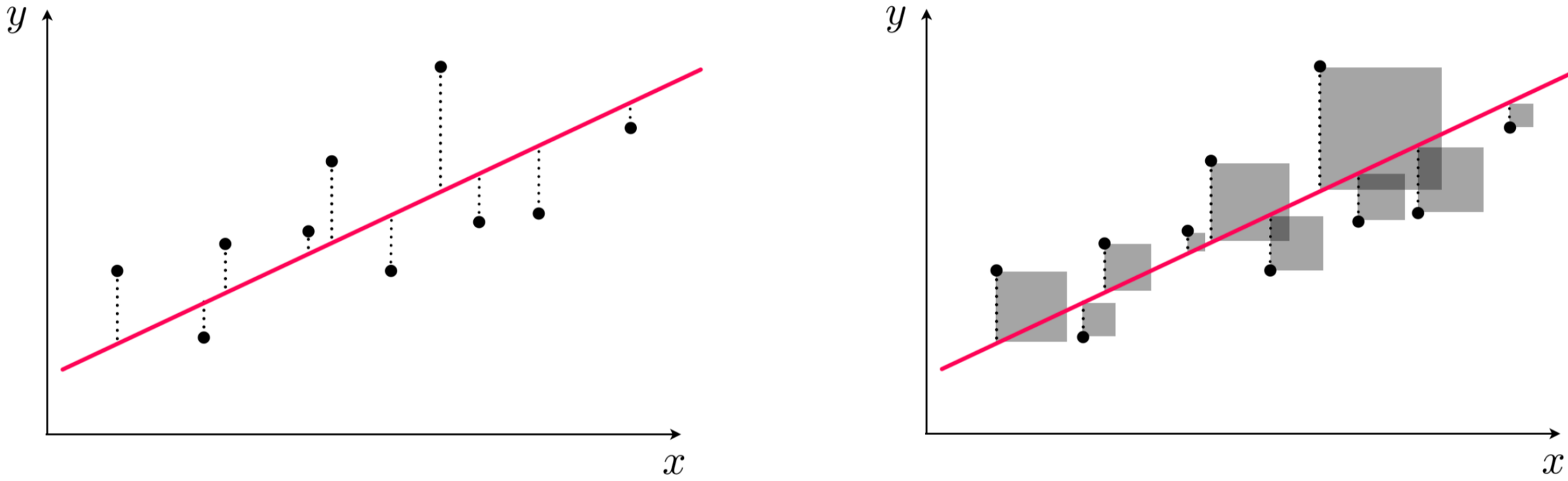
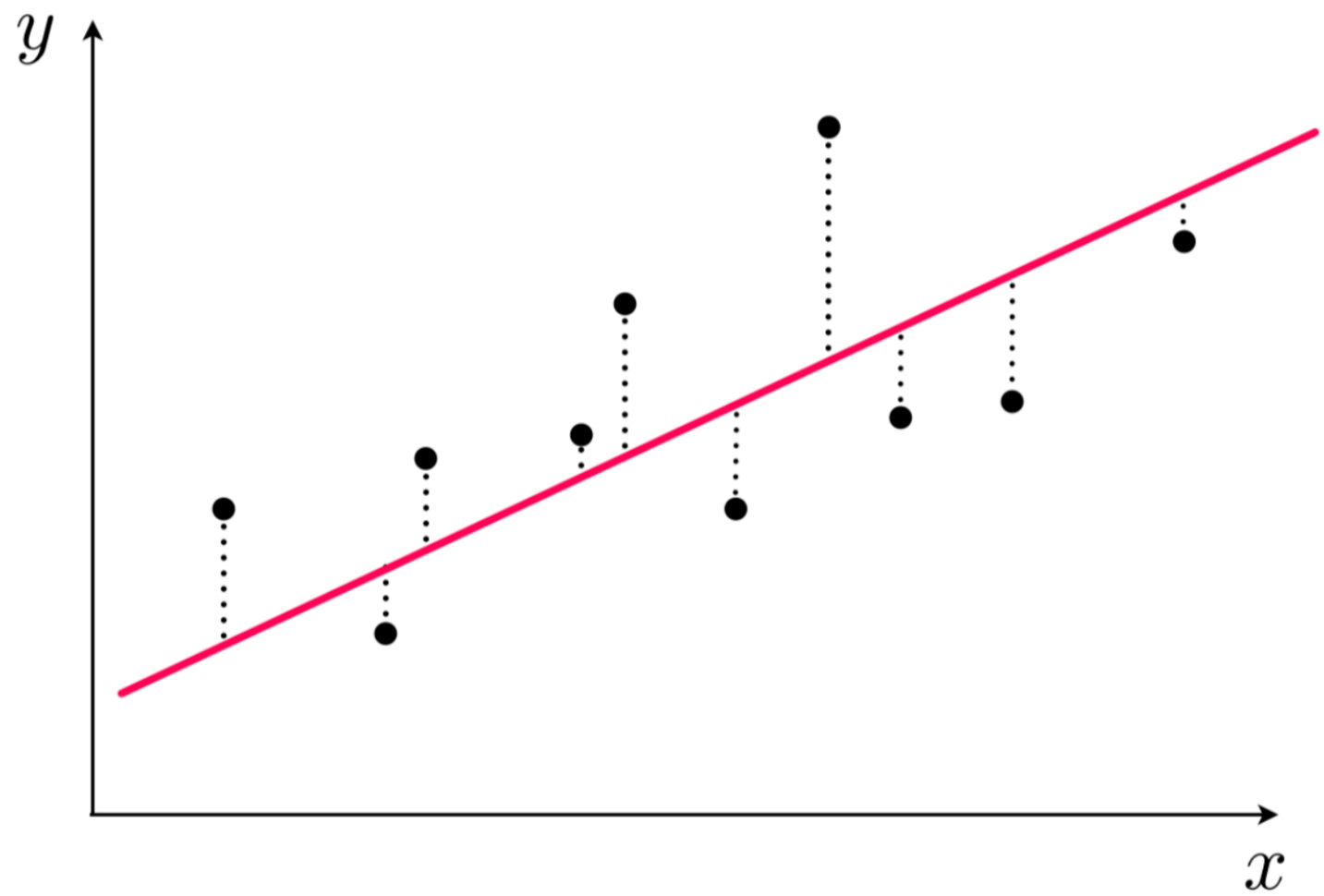


Figure: (left panel) A simulated two dimensional dataset along with a line fit to the data using the Least Squares framework, which aims at recovering the line that minimizes the total squared length of the dashed error bars.
(right panel) The Least Squares error can be thought of as the total area of the gray squares, having dashed error bars as sides.

The Least Squares Cost Function



Naturally then the best fitting hyperplane is the one whose parameters minimize this error

- Where does this "Least Squares" cost come from?
- Remember, we want to find a weight vector $\mathbf{w}$ so that each of P approximate equalities below holds as tightly as possible:

$$\overset{\circ}{\mathbf{x}}_p^T \mathbf{w} \approx y_p \quad p = 1, \ldots, P.$$

- Another way of stating the above is to say that the error between

$$\overset{\circ}{\mathbf{x}}_p^T \mathbf{w} \quad \& \; y_p$$

is small

Least Squares Linear Regression

Model: $\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) = \mathring{\mathbf{x}}_p^T \mathbf{w}$

Cost Function: $g\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P}\left(\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) - y_p\right)^2$

Input Data: $\mathbf{x}_p = x_{1,p}, x_{2,p}, \ldots, x_{N,p}$ (N features, P data point)

Output Data: y_p

Implementing Least Squares Linear Regression

How do we implement the least squares cost function in Python?

You can loop over all the data points.

Or you can use numpy matrices.

Implementing Least Squares Linear Regression

How do we implement the least squares cost function in Python?

```
a = w[0] + np.dot(x_p.T,w[1:])
```

```
# compute linear combination of input point
 def model(x_p,w):
 # compute linear combination and return
 a = w[0] + np.dot(x_p.T,w[1:])
 return a.T
```

Minimizing the Cost Function

Is the Least Squares cost function convex?

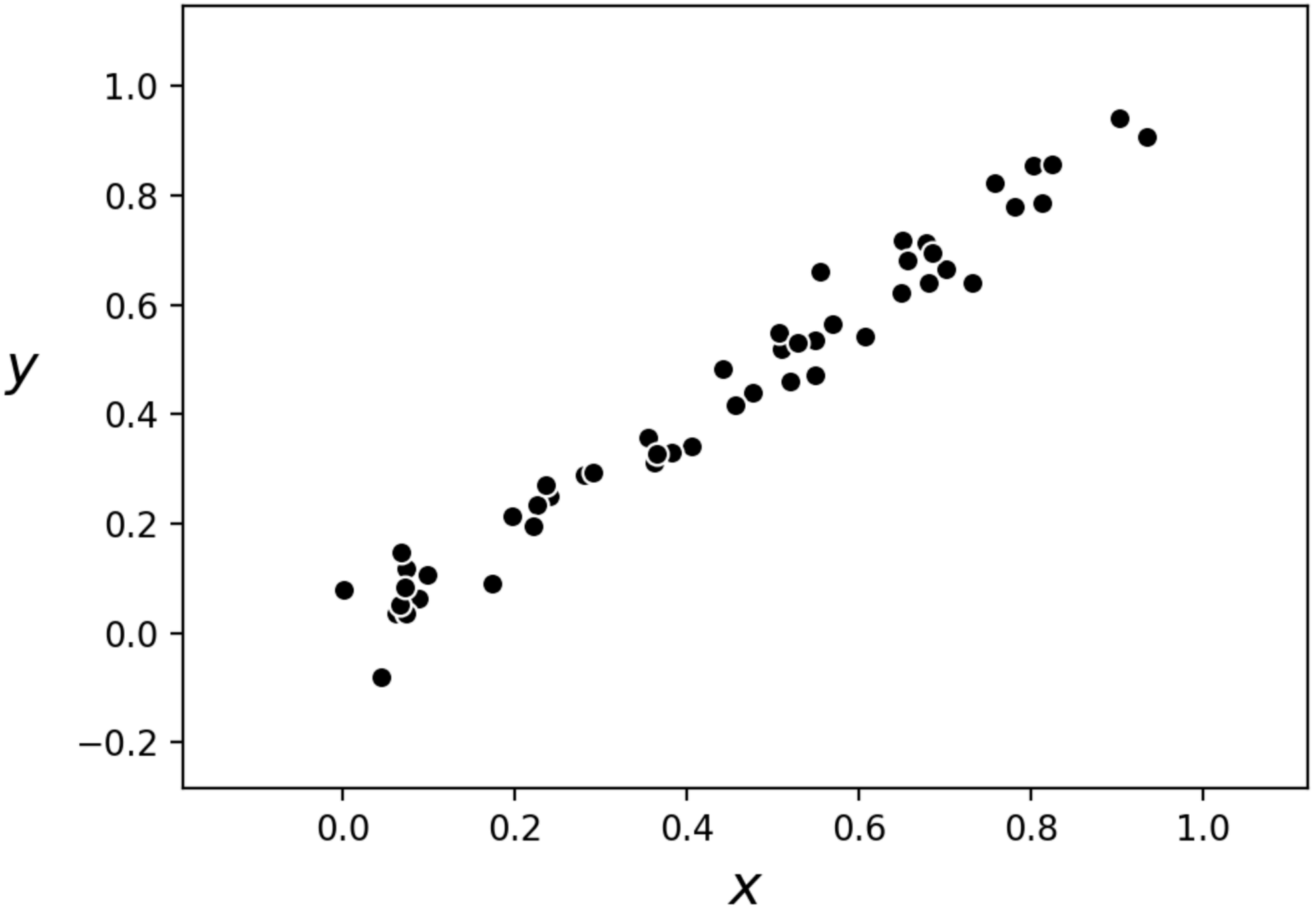
$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) - y_p\right)^2$$

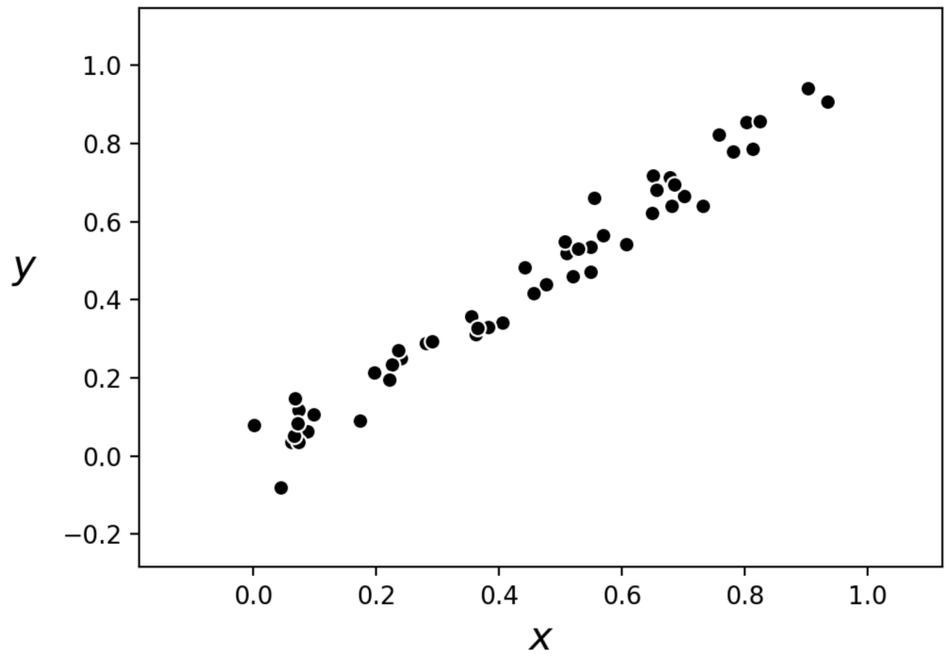
Minimizing the Cost Function

Is the Least Squares cost function convex?

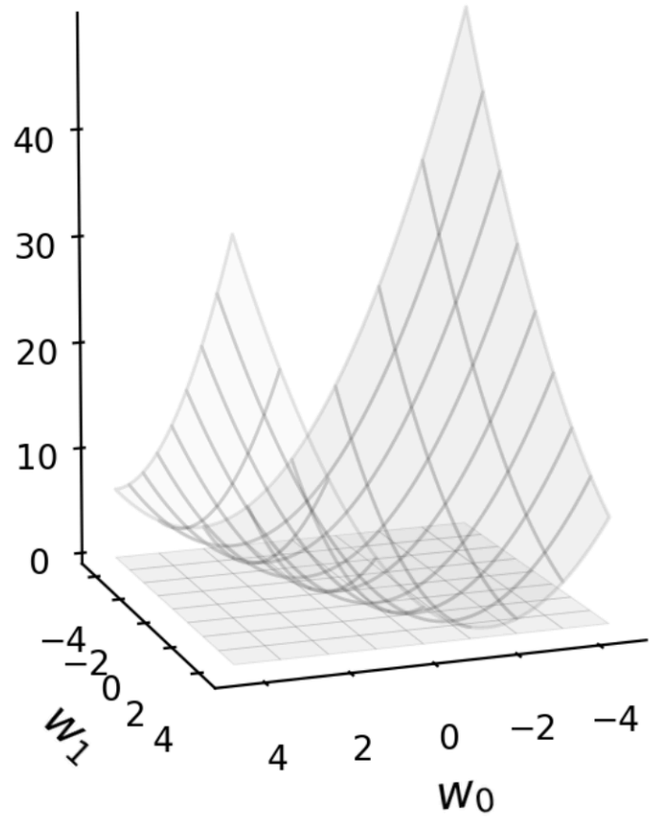
$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) - y_p\right)^2$$

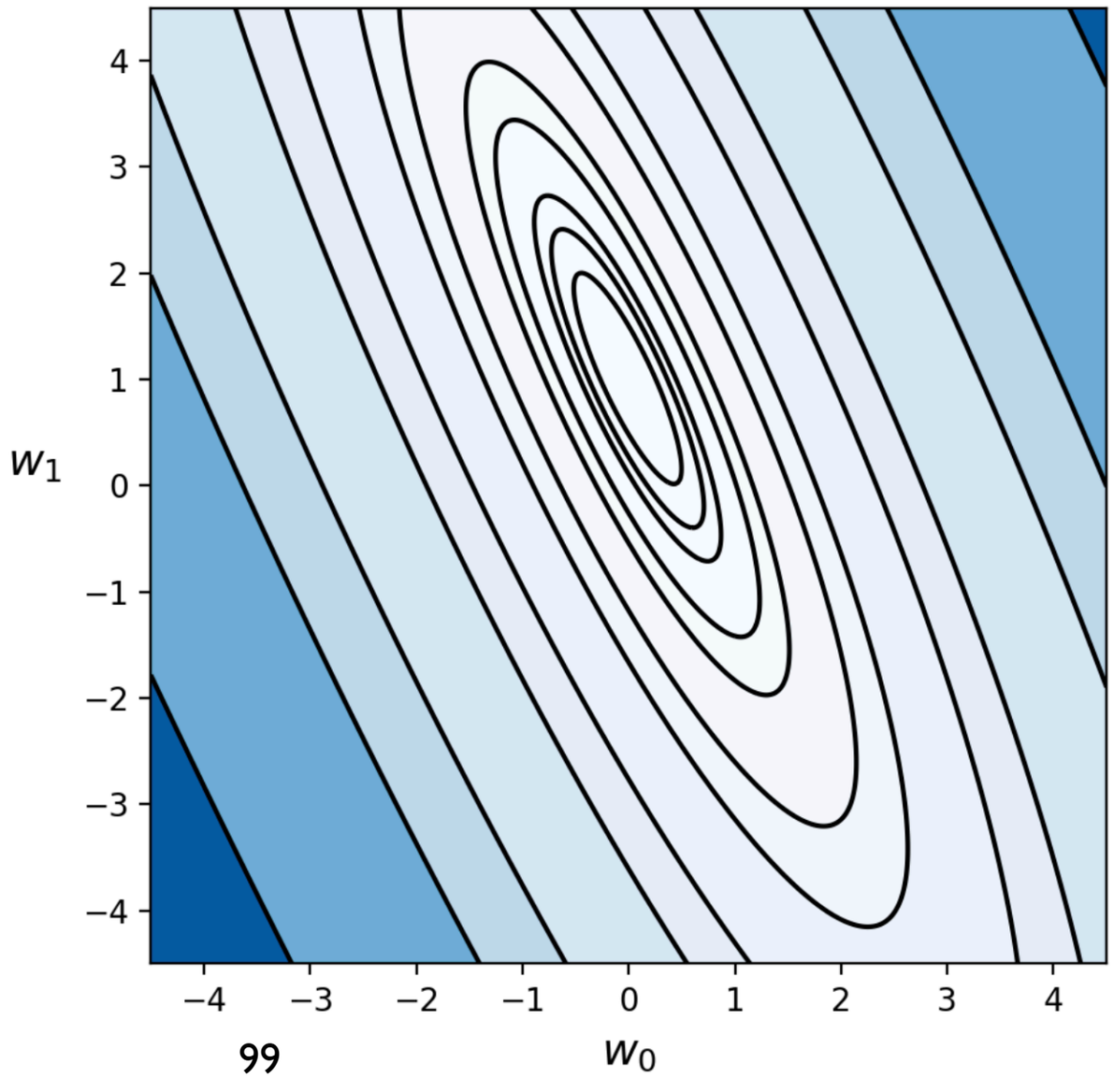
It is ALWAYS a convex quadratic function.




1.0
0.8
0.6
0.4
0.2
0.0
−0.2
y
0.0
0.2
0.4
0.6
0.8
1.0
x

$g(w_0, w_1)$


40
30
20
10
0
−4
−2
0
2
4
w_1
4
2
0
−2
−4
w_0


4
3
2
1
0
−1
−2
−3
−4
w_1
−4
−3
−2
−1
0
1
2
3
4
w_0

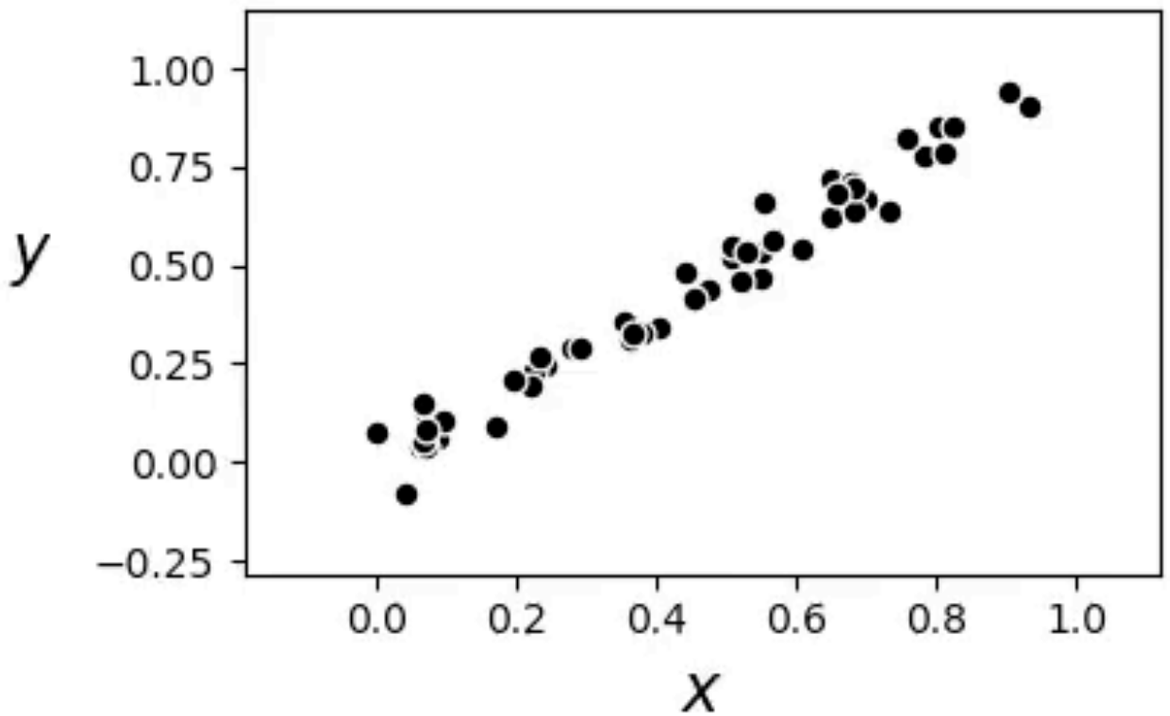
Minimizing the Cost Function

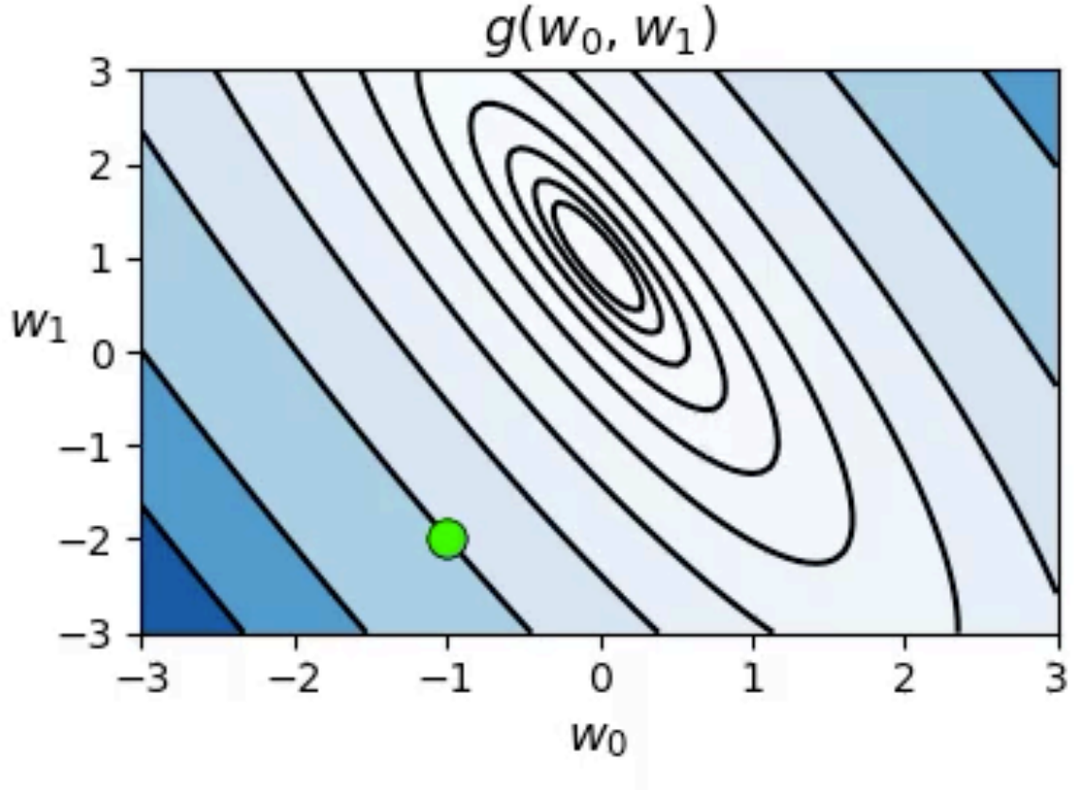
$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) - y_p\right)^2$$

Since it is convex, we know gradient descent can converge to global minimum

α=0.5

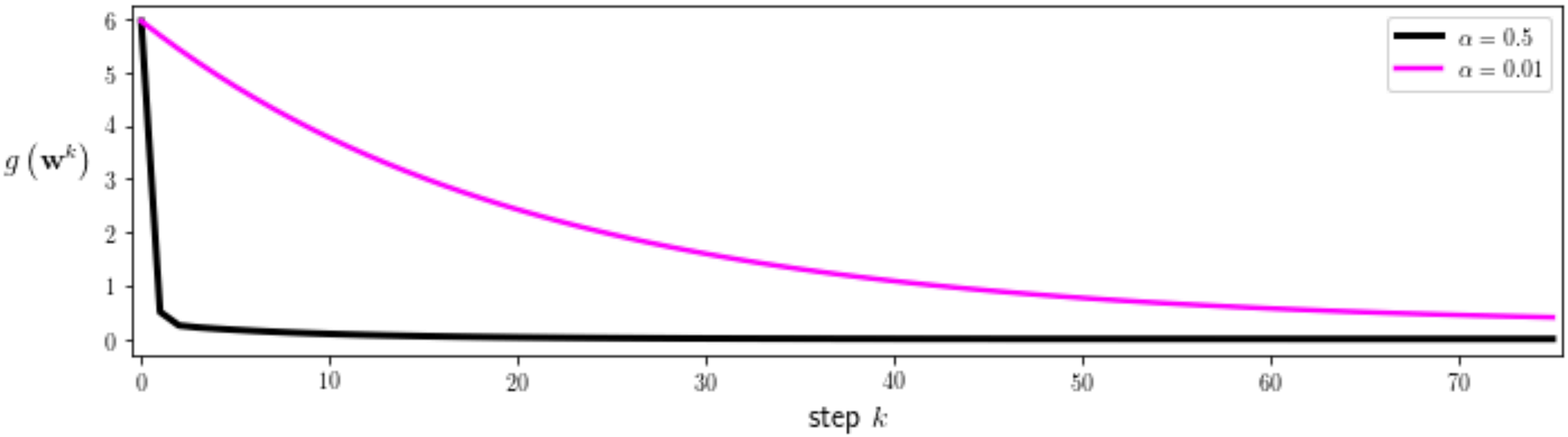
K=75



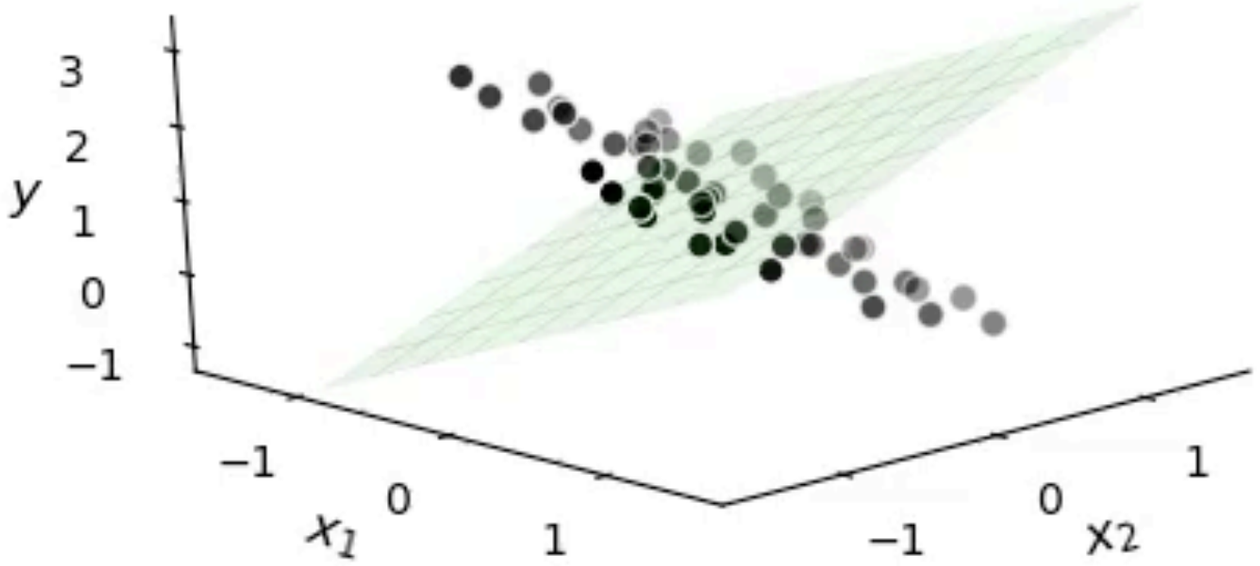


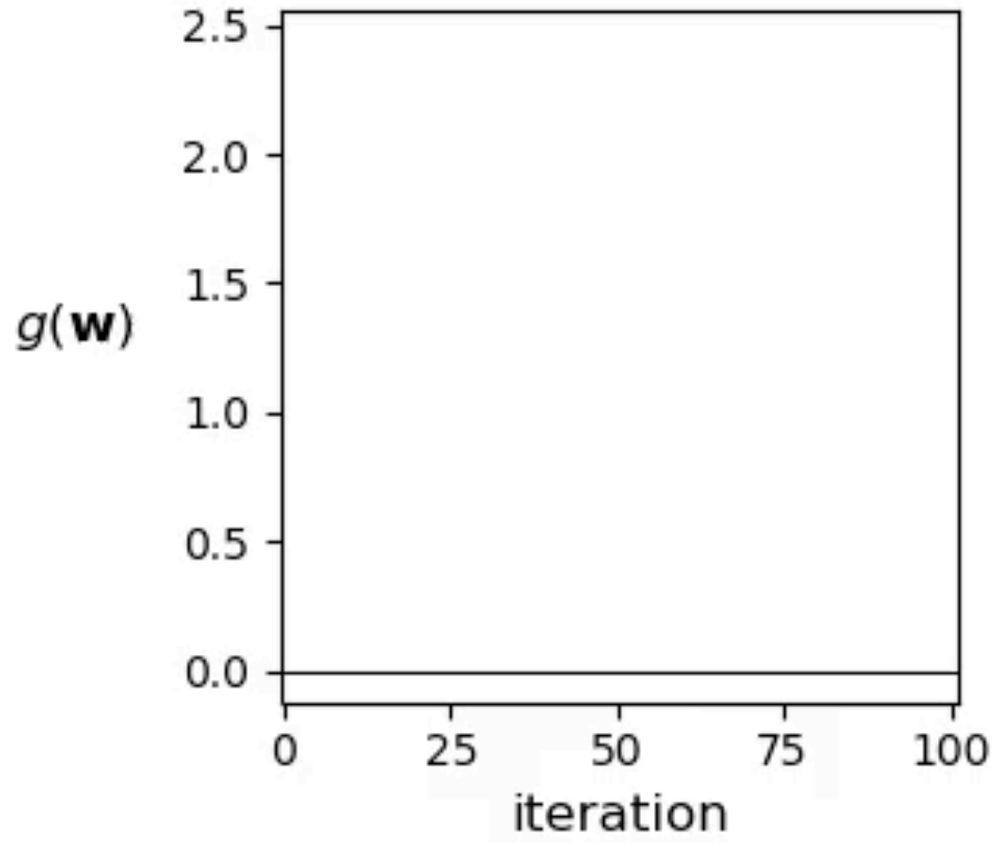
Whenever we use gradient descent we must properly tune the learning rate parameter α

How did we do this?


$\alpha = 0.5$
$\alpha = 0.01$
$g\left(\mathbf{w}^k\right)$
step k
0
1
2
3
4
5
6
0
10
20
30
40
50
60
70

Least Squares Regression in 3D





Design Choices

Goal: learn f s.t. $y = f(x)$

1. **How do we parameterize the function?**
   → choosing the model
2. What learning objective do we use?
   → choosing the cost/loss function
3. How do we optimize the objective?
   → defining the learning algorithm

Linear Regression

- Fitting of a representative line (or hyperplane in higher dimensions) to a set of input/output data points.

$$w_0 + x_{1,p}w_1 + x_{2,p}w_2 + \cdots + x_{N,p}w_N \approx y_p, \quad p = 1, \ldots, P.$$

$$\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) = \mathring{\mathbf{x}}_p^T \mathbf{w}$$

Design Choices

Goal: learn f s.t. $y = f(x)$

1. How do we parameterize the function?
   → choosing the model

2. What learning objective do we use?
   → choosing the cost/loss function

3. How do we optimize the objective?
   → defining the learning algorithm

- We wanted the error between

$$\overset{\circ}{\mathbf{x}}_p^T \mathbf{w} \quad \& \; y_p$$

to be small

- And we wanted the function to be smooth and differentiable.
  - Smooth functions have a uniquely defined first derivative at every point.

- We wanted the error between

$$\overset{\circ}{\mathbf{x}}_p^T \mathbf{w} \quad \& \; y_p$$

to be small

- And we wanted the function to be smooth and differentiable.

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P} g_p(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\overset{\circ}{\mathbf{x}}_p^T \mathbf{w} - y_p\right)^2$$

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P} g_p(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\overset{\circ}{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

we want to ***learn*** our parameters/weights $\mathbf{w}$

to minimize the Least Squares cost:

$$\underset{\mathbf{w}}{\text{minimize}}\ \frac{1}{P}\sum_{p=1}^{P}\left(\overset{\circ}{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

Design Choices

Goal: learn f s.t. $y = f(x)$

1. How do we parameterize the function?
   → choosing the model

2. What learning objective do we use?
   → choosing the cost/loss function

3. **How do we optimize the objective?**
   → defining the learning algorithm

Gradient Descent

$$\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$$

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1 \ldots K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha \nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P} g_p(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\mathring{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

You are "looping" through the data

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$

Least Squares Linear Regression

Model: $$\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) \approx y_p$$

Cost Function: $$g\left(\mathbf{w}\right) = \frac{1}{P}\sum_{p=1}^{P}\left(\text{model}\left(\mathbf{x}_p, \mathbf{w}\right) - y_p\right)^2$$

Learning algorithm: Gradient Descent

Input Data: $\mathbf{x}_p = x_{1,p}, x_{2,p}, \ldots, x_{N,p}$ (N features, P data point)

Output Data: y_p

$$g(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P} g_p(\mathbf{w}) = \frac{1}{P}\sum_{p=1}^{P}\left(\mathring{\mathbf{x}}_p^T\mathbf{w} - y_p\right)^2$$

You are "looping" through the data

The gradient descent algorithm

1: **input:** function g, steplength α, maximum number of steps K, and initial point $\mathbf{w}^0$

2: `for` $k = 1...K$

3: $\mathbf{w}^k = \mathbf{w}^{k-1} - \alpha\nabla g\left(\mathbf{w}^{k-1}\right)$

4: **output:** history of weights $\left\{\mathbf{w}^k\right\}_{k=0}^{K}$ and corresponding function evaluations $\left\{g\left(\mathbf{w}^k\right)\right\}_{k=0}^{K}$
