Gradient Descent

Gradient Descent

- Gradient descent is one of the most popular algorithms to perform optimization and is the most common way to optimize neural networks. It is an iterative optimization algorithm used to find the minimum value for a function.
- A **derivative** is a term that comes from calculus and is calculated as the slope of the graph at a particular point. The slope is described by drawing a tangent line to the graph at the point.
- Method to find local optima of differentiable a function f
  - Intuition: gradient tells us direction of greatest increase, negative gradient gives us direction of greatest decrease
    - Take steps in directions that reduce the function value
  - Definition of derivative guarantees that if we take a small enough step in the direction of the negative gradient, the function will decrease in value
    - How small is small enough?

Gradient Descent

•

Gradient Descent Algorithm:

- Pick an initial point x_0
- Iterate until convergence

$$x_{t+1} = x_t - \gamma_t \nabla f(x_t)$$

where γ_t is the t^{th} step size (sometimes called learning rate)

Gradient Descent

•

Gradient Descent Algorithm:

- Pick an initial point x_0
- Iterate until convergence

$$x_{t+1} = x_t - \gamma_t \nabla f(x_t)$$

where γ_t is the t^{th} step size (sometimes called learning rate)

When do we stop?

Gradient Descent

•

Gradient Descent Algorithm:

- Pick an initial point x_0
- Iterate until convergence

$$x_{t+1} = x_t - \gamma_t \nabla f(x_t)$$

where γ_t is the t^{th} step size (sometimes called learning rate)

Possible Stopping Criteria: iterate until
$\|\nabla f(x_t)\| \leq \epsilon$ for some $\epsilon > 0$

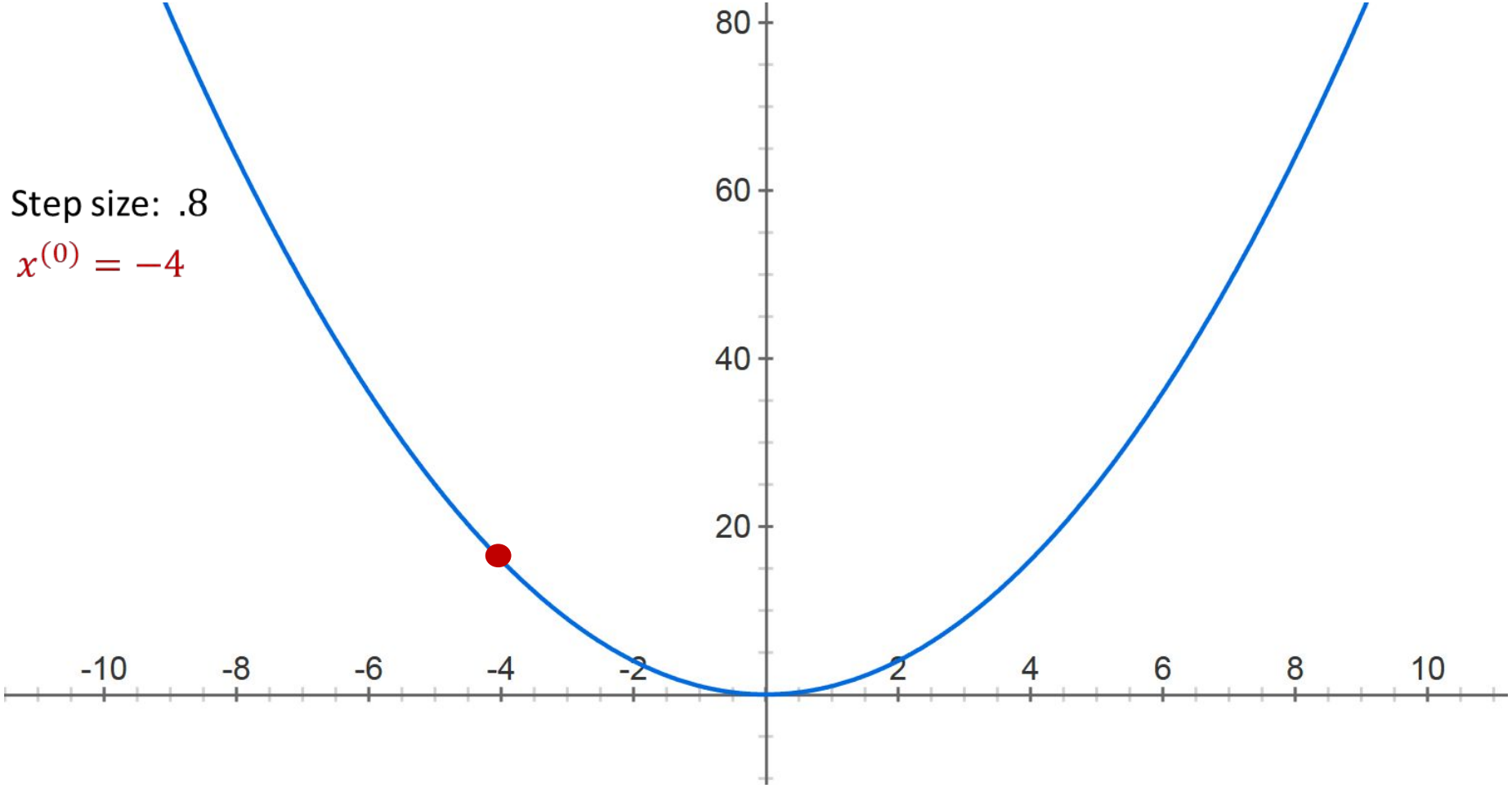
How small should ϵ be?

Gradient Descent

$f(x) = x^2$

Step size: .8

$x^{(0)} = -4$



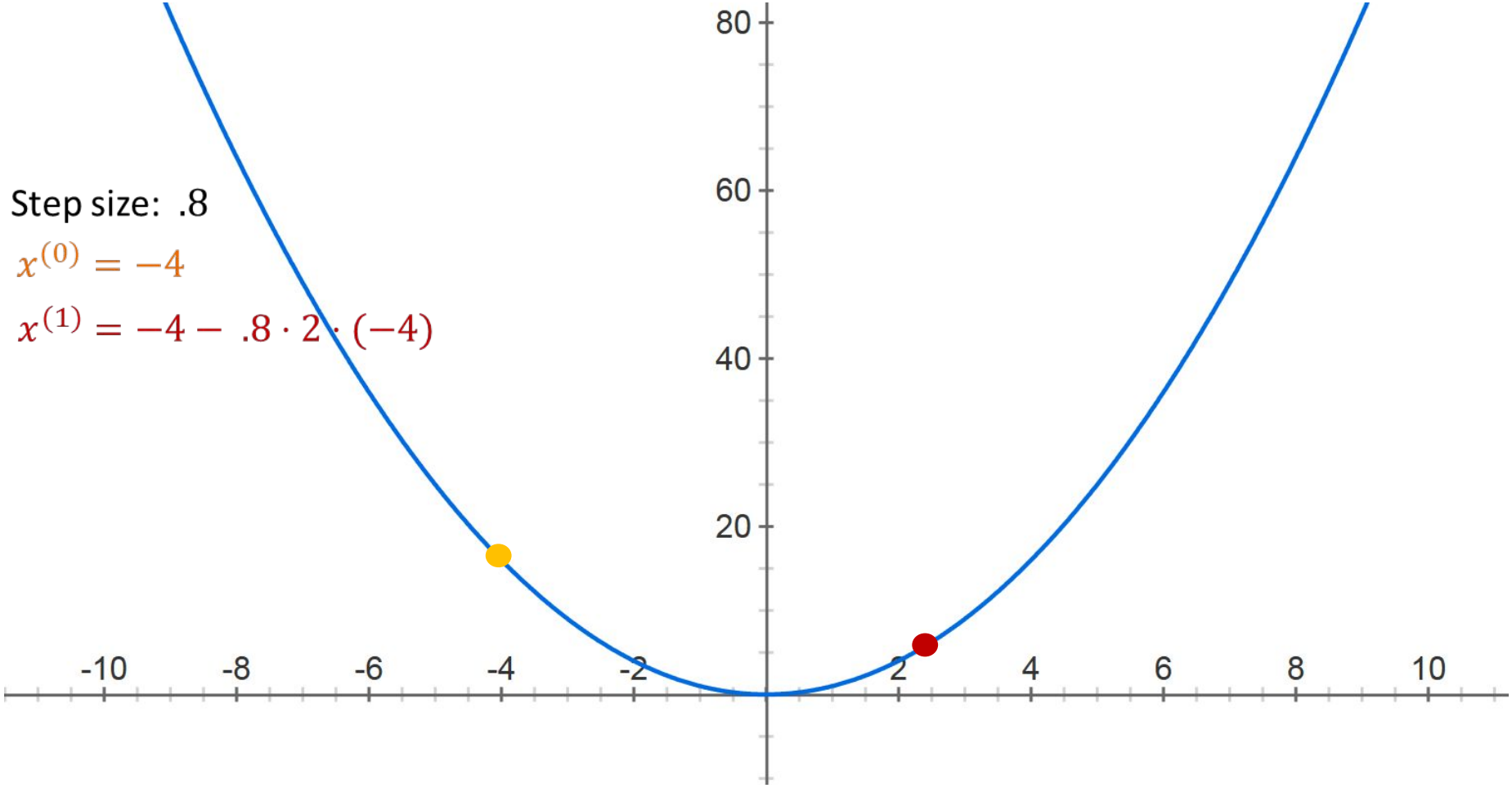
Gradient Descent

$f(x) = x^2$

Step size: .8

$x^{(0)} = -4$

$x^{(1)} = -4 - .8 \cdot 2 \cdot (-4)$



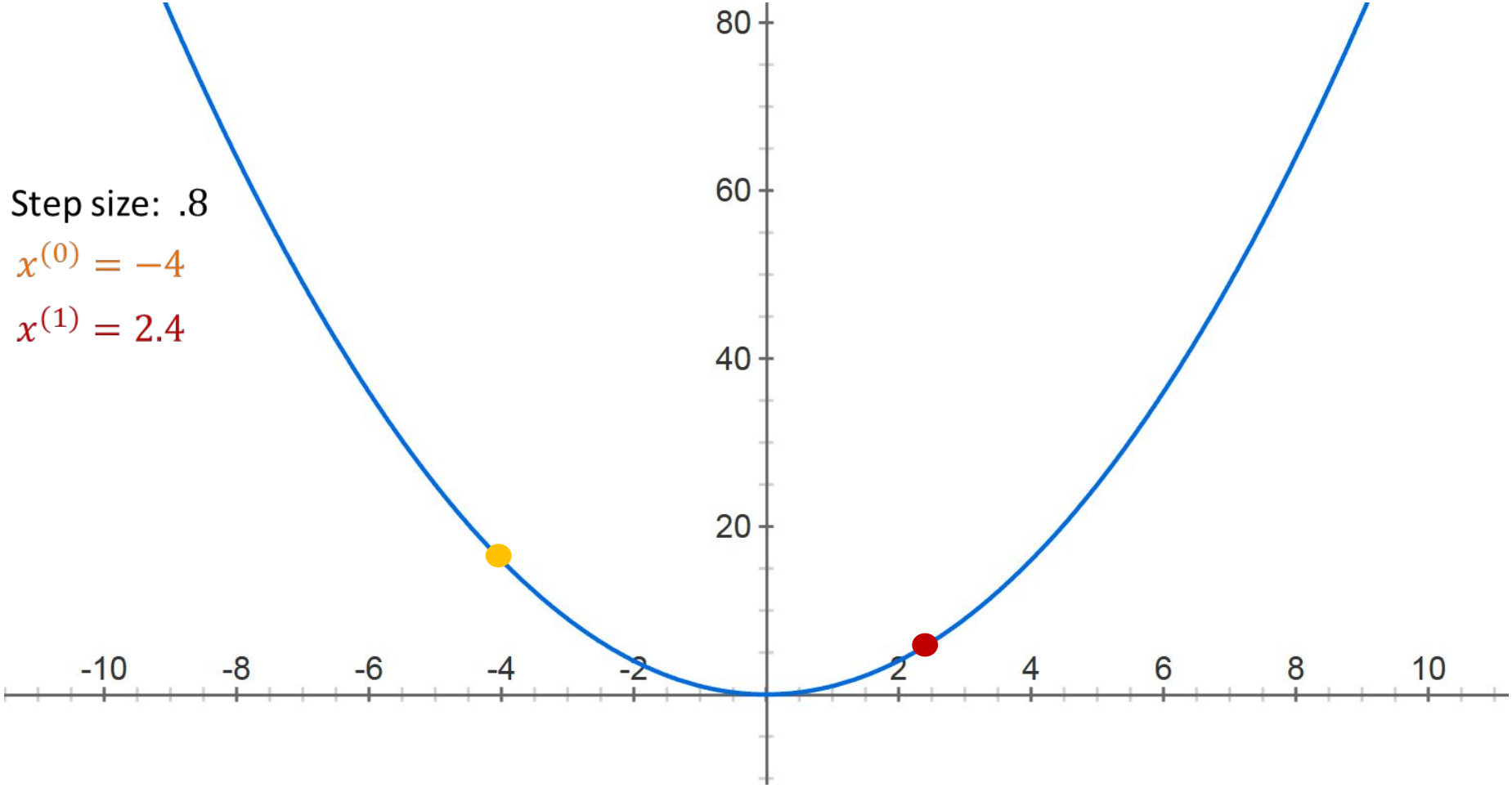
Gradient Descent

$f(x) = x^2$

Step size: .8

$x^{(0)} = -4$

$x^{(1)} = 2.4$



Gradient Descent

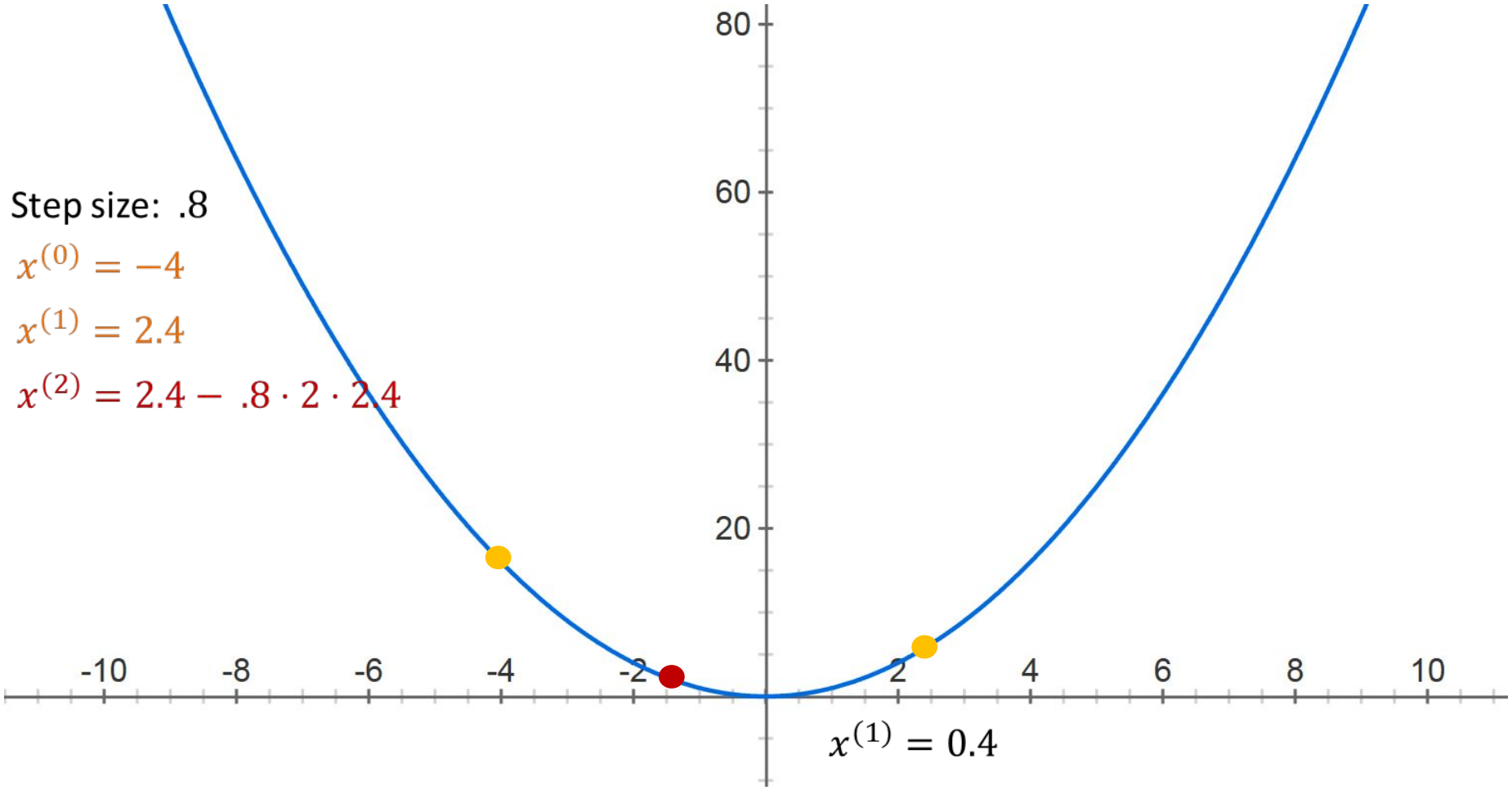
$f(x) = x^2$

Step size: .8

$x^{(0)} = -4$

$x^{(1)} = 2.4$

$x^{(2)} = 2.4 - .8 \cdot 2 \cdot 2.4$



Gradient Descent

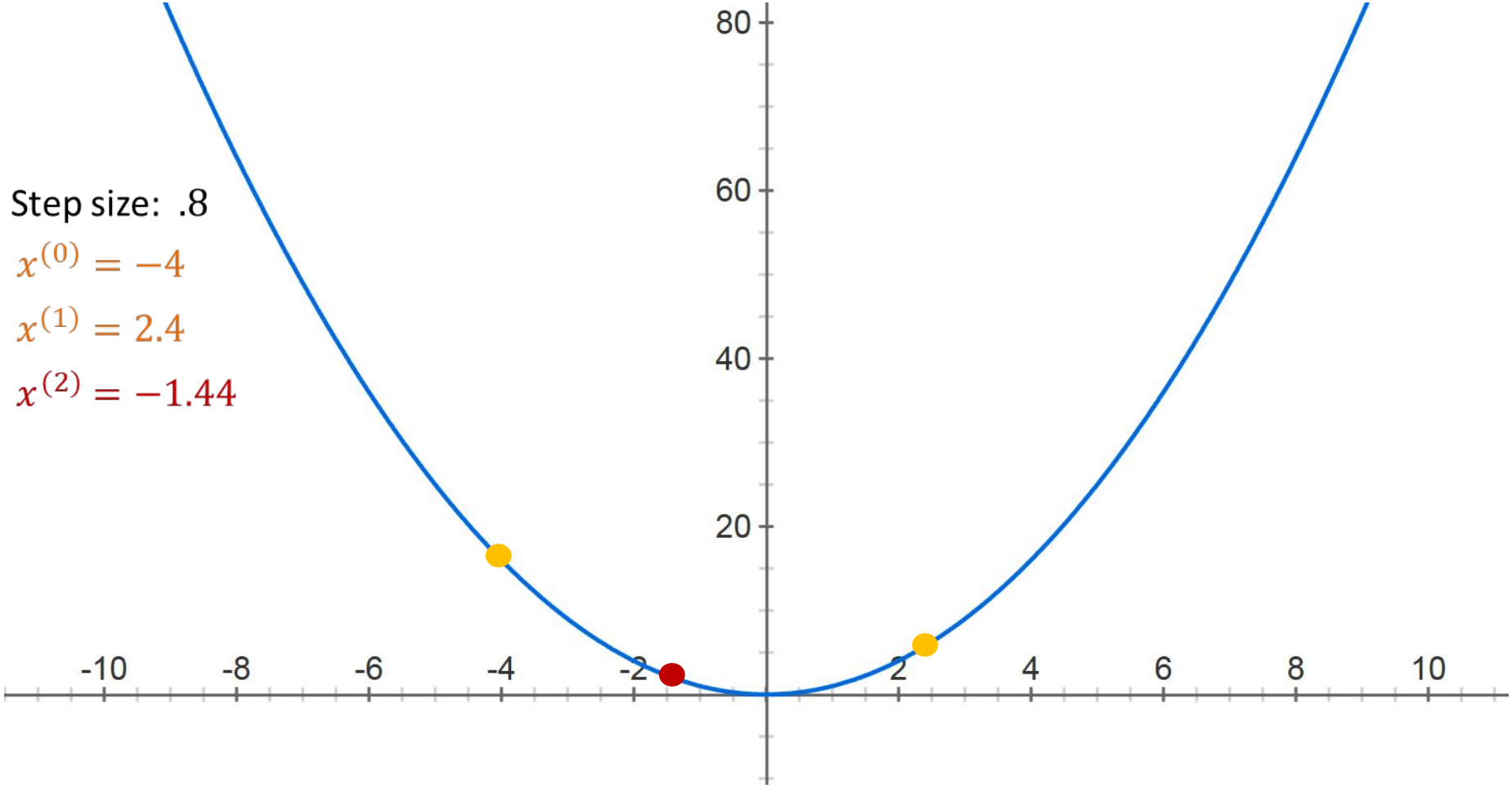
$f(x) = x^2$

Step size: .8

$x^{(0)} = -4$

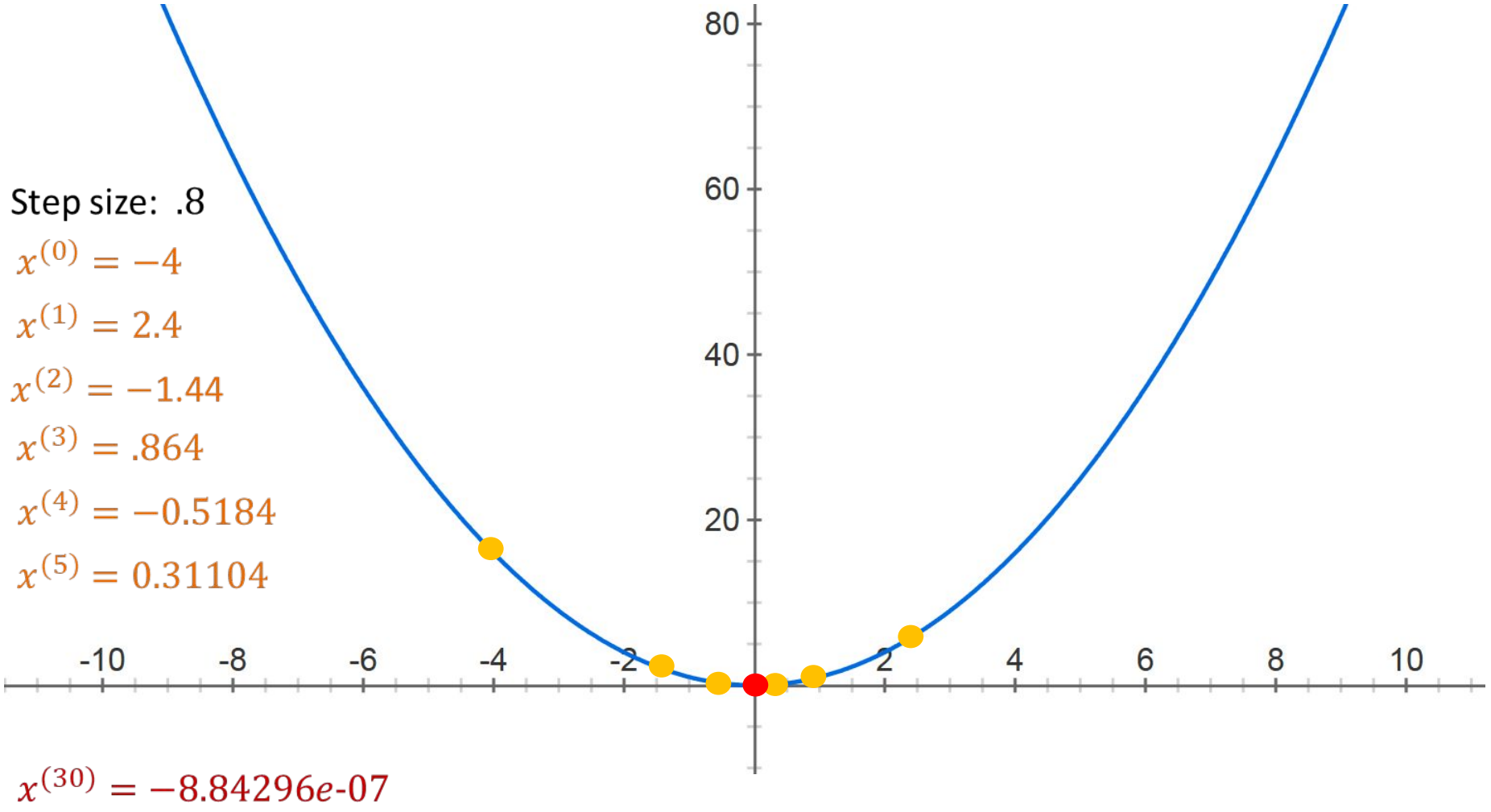
$x^{(1)} = 2.4$

$x^{(2)} = -1.44$



Gradient Descent

$f(x) = x^2$



Step size: .8

$x^{(0)} = -4$

$x^{(1)} = 2.4$

$x^{(2)} = -1.44$

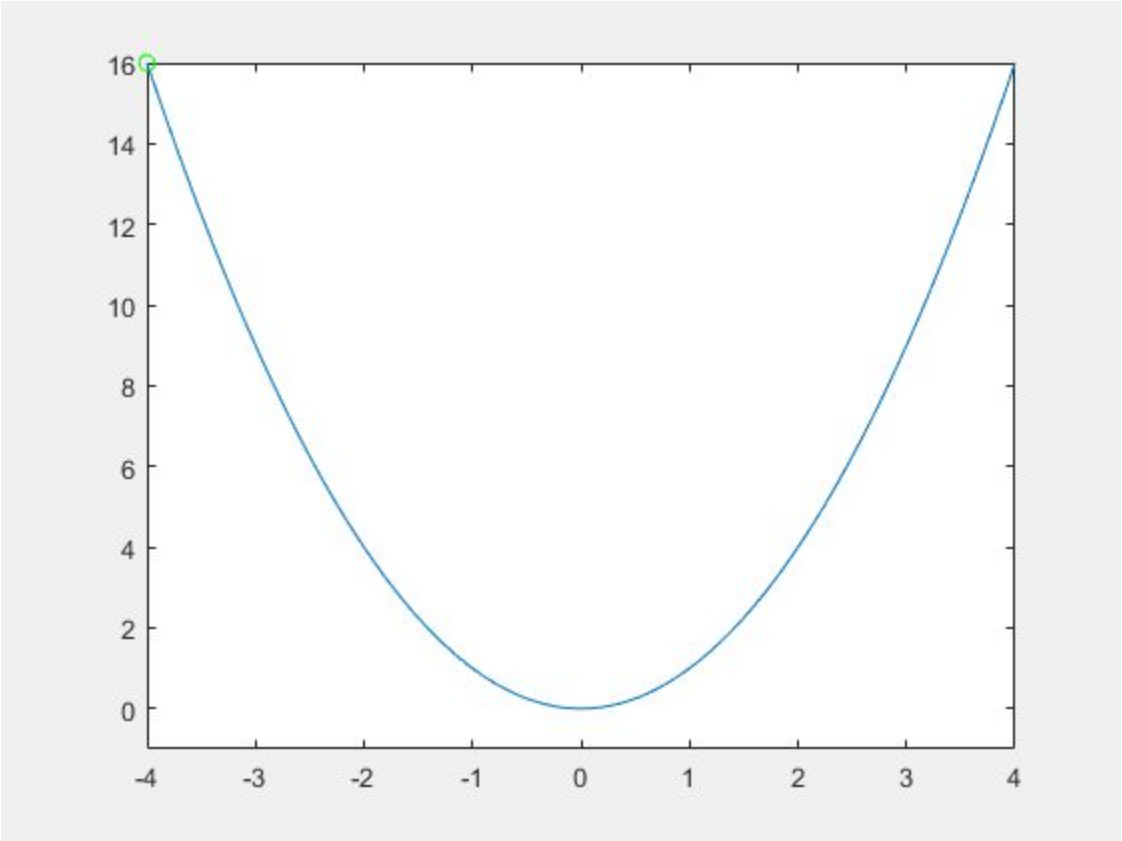
$x^{(3)} = .864$

$x^{(4)} = -0.5184$

$x^{(5)} = 0.31104$

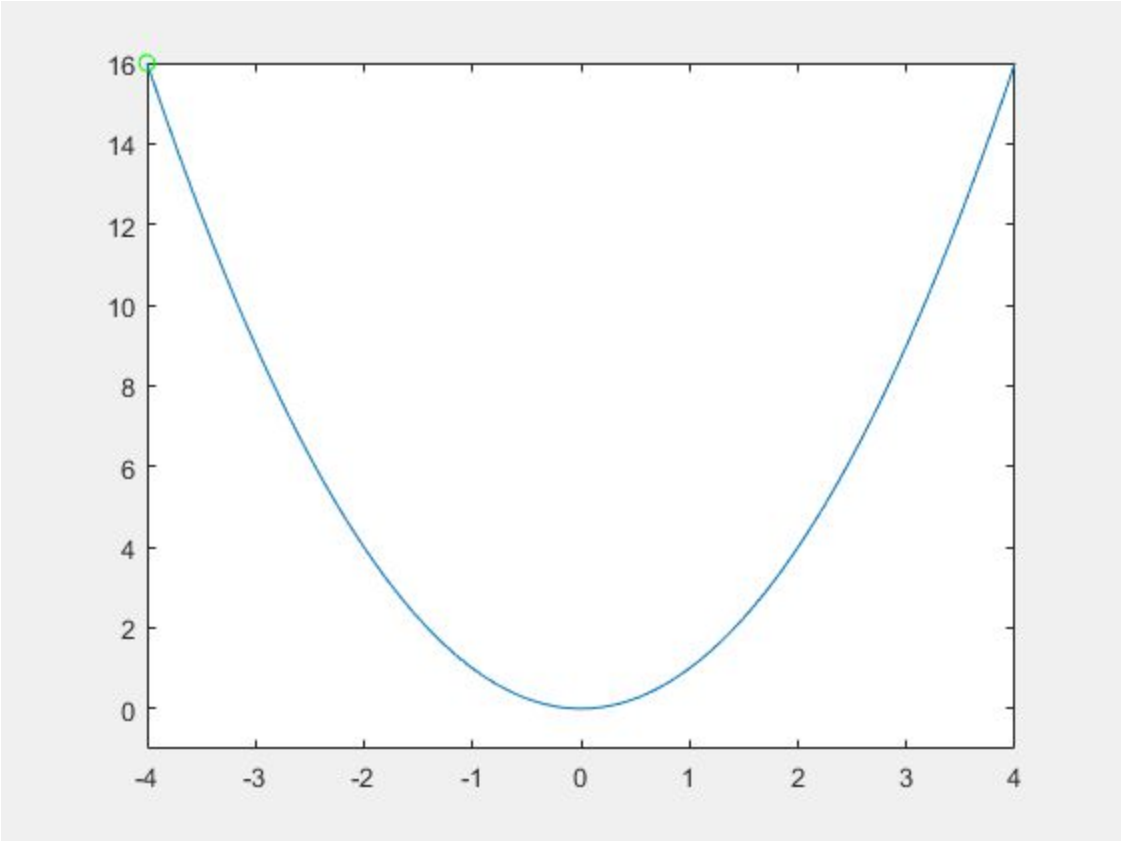
$x^{(30)} = -8.84296e\text{-}07$

Gradient Descent



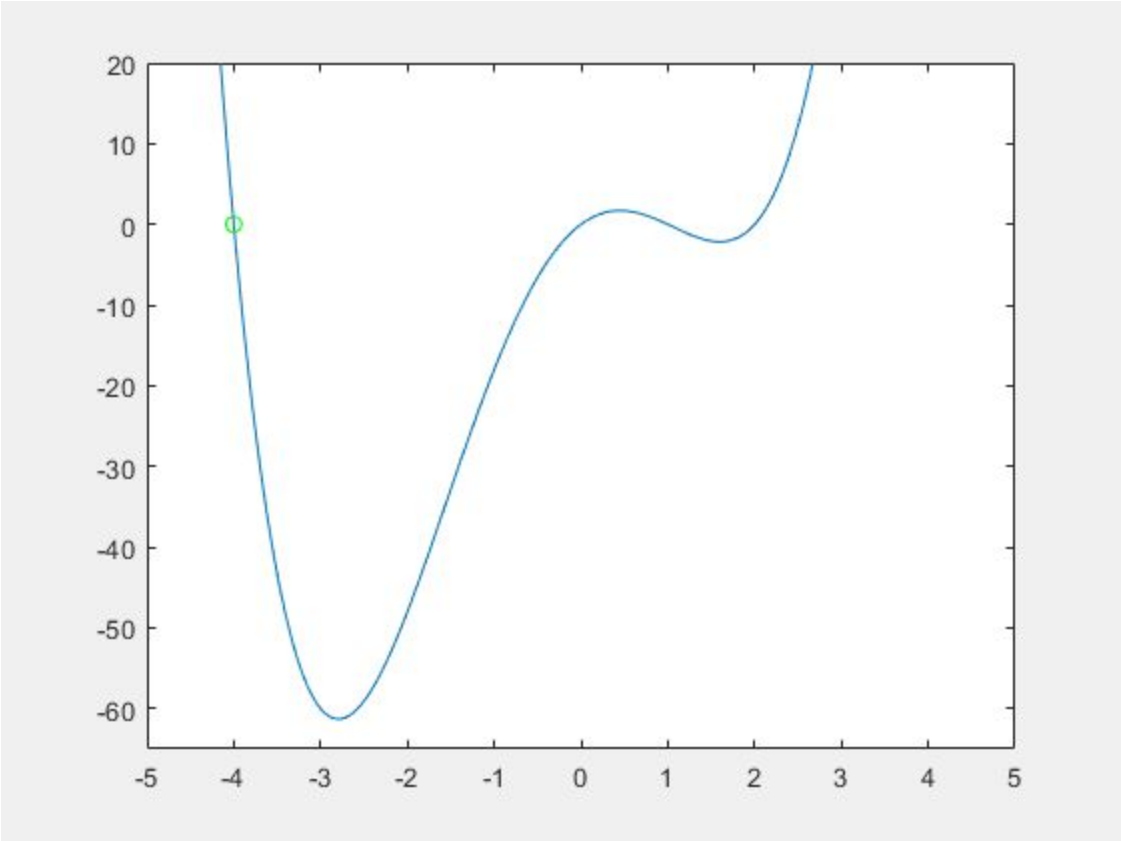
Step size: .9

Gradient Descent



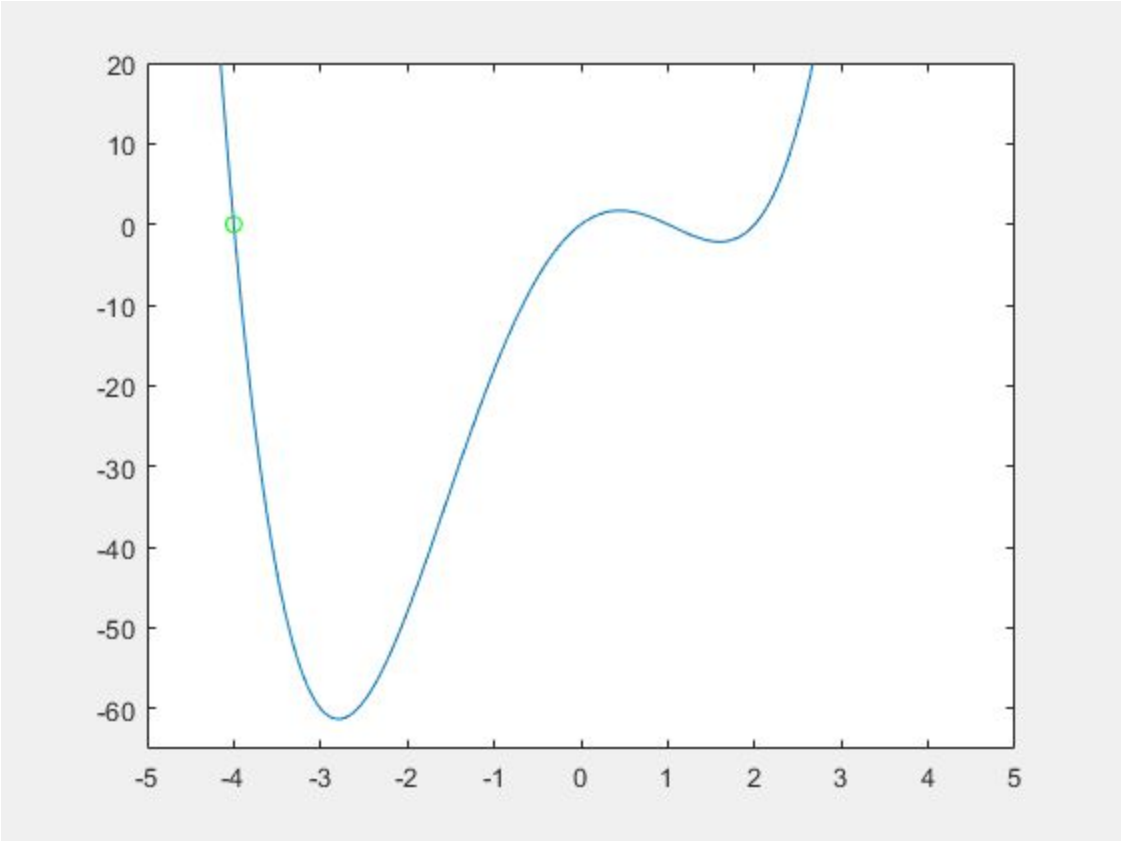
Step size: .2

Gradient Descent



Step size matters!

Gradient Descent



Step size matters!

Derivatives

Machine learning uses derivatives in optimization problems. Optimization algorithms like gradient descent use derivates to decide whether to increase or decrease the weights to increase or decrease any objective function. If we are able to compute the derivative of a function, we know in which direction to proceed to minimize it. Primarily we shall be dealing with two concepts from calculus : Power Rule- Power rule calculates the derivative of a variable raised to a power.

If we have a function like

$$f(x) = x^n$$

, then

$$\frac{\partial f(x)}{\partial x} = nx^{n-1}$$

Example : Find the derivative of the function f(x) w.r.t. x where

$$f(x) = 3x^5$$

$$\frac{\partial f(x)}{\partial x} = 15x^4$$

- Chain Rule
- The chain rule is used for calculating the derivative of composite functions. The chain rule can also be expressed in Leibniz's notation as follows:

$$\frac{\partial z}{\partial x} = \frac{\partial z}{\partial y} \cdot \frac{\partial y}{\partial x}$$

if

$$y = x^2$$

and

$$x = z^2$$

then, the derivative of **y** w.r.t **z** can be calculated with **chain rule** as follows:

$$\frac{\partial y}{\partial z} = \frac{\partial y}{\partial x} \cdot \frac{\partial x}{\partial z}$$

$$\frac{\partial y}{\partial x} = 2x$$

$$\frac{\partial x}{\partial z} = 2z$$

Hence,

$$\frac{\partial y}{\partial z} = 2x.2z$$

- Partial derivatives, which says that if there is a function of two variables, then to find the partial derivative of that function w.r.t to one variable, treat
- the other variable as constant. Example

Partial Derivatives

Say for instance, we have a function:

$$f(x, y) = x^4 + y^7$$

partial derivative of the function w.r.t 'x' will be :

$$\frac{\partial f}{\partial x} = 4x^3 + 0$$

treating 'y' as a constant

And partial derivative of the function w.r.t 'y' will be :

$$\frac{\partial f}{\partial y} = 0 + 7y^6$$

treating 'x' as a constant

MADALINE

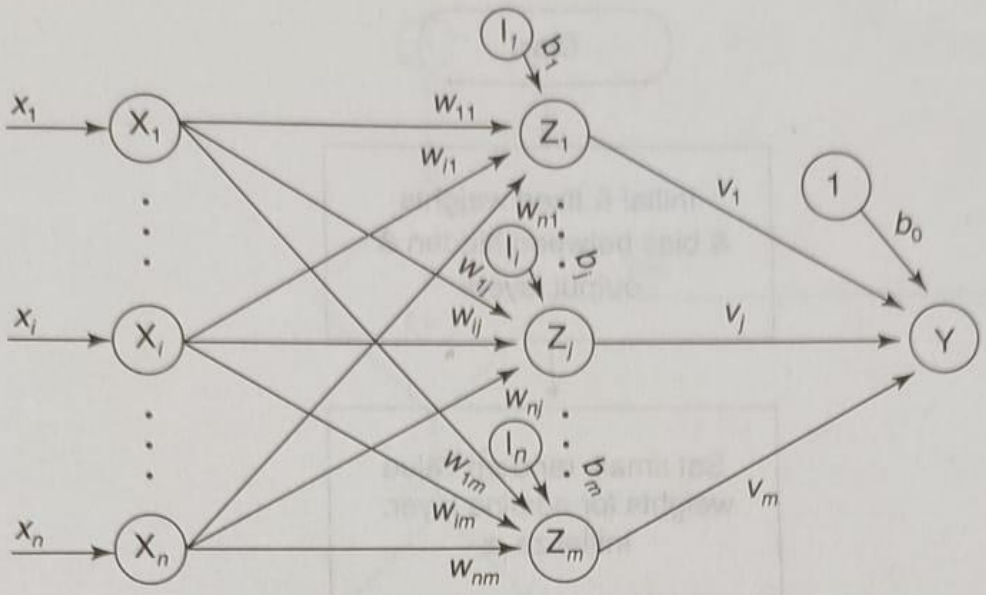


Figure 3-7 Architecture of Madaline layer.

3.4.4 Training Algorithm

In this training algorithm, only the weights between the hidden layer and the input layer are adjusted, and the weights for the output units are fixed. The weights $v_1, v_2, \ldots, v_m$ and the bias b_0 that enter into output unit Y are determined so that the response of unit Y is 1. Thus, the weights entering Y unit may be taken as

$$v_1 = v_2 = \cdots = v_m = \frac{1}{2}$$

and the bias can be taken as

$$b_0 = \frac{1}{2}$$

The activation for the Adaline (hidden) and Madaline (output) units is given by

$$f(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ -1 & \text{if } x < 0 \end{cases}$$

Step 0: Initialize the weights. The weights entering the output unit are set as above. Set initial small random values for Adaline weights. Also set initial learning rate α.

Step 1: When stopping condition is false, perform Steps 2–3.

Step 2: For each bipolar training pair $s{:}t$, perform Steps 3–7.

Step 3: Activate input layer units. For $i = 1$ to n,

$$x_i = s_i$$

Step 4: Calculate net input to each hidden Adaline unit:

$$z_{inj} = b_j + \sum_{i=1}^{n} x_i w_{ij}, \quad j = 1 \text{ to } m$$

Step 5: Calculate output of each hidden unit:

$$z_j = f(z_{inj})$$

Step 6: Find the output of the net:

$$y_{in} = b_0 + \sum_{j=1}^{m} z_j v_j$$

$$y = f(y_{in})$$

Step 7: Calculate the error and update the weights.

1. If $t = y$, no weight updation is required.
2. If $t \neq y$ and $t = +1$, update weights on z_j, where net input is closest to 0 (zero):

$$b_j(\text{new}) = b_j(\text{old}) + \alpha\,(1 - z_{inj})$$
$$w_{ij}(\text{new}) = w_{ij}(\text{old}) + \alpha\,(1 - z_{inj})x_i$$

3. If $t \neq y$ and $t = -1$, update weights on units z_k whose net input is positive:

$$w_{ik}(\text{new}) = w_{ik}(\text{old}) + \alpha\,(-1 - z_{ink})\,x_i$$
$$b_k(\text{new}) = b_k(\text{old}) + \alpha\,(-1 - z_{ink})$$

Step 8: Test for the stopping condition. (If there is no weight change or weight reaches a satisfactory level, or if a specified maximum number of iterations of weight updation have been performed then stop, or else continue).

Madalines can be formed with the weights on the output unit set to perform some logic functions. If there are only two hidden units present, or if there are more than two hidden units, then the "majority vote rule" function may be used.