Supervised Learning and Generalization

DSCC 251/451: Machine Learning with Limited Data

C.M. Downey

Spring 2026

Supervised Learning Review

Supervised Learning Basics

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
  - The dataset contains **pairs of inputs and outputs**

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
  - The dataset contains **pairs of inputs and outputs**
  - Ex: speed $\rightarrow$ whether you get a speeding ticket

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
  - The dataset contains **pairs of inputs and outputs**
  - Ex: speed $\rightarrow$ whether you get a speeding ticket
    - {(30, False), (33, False), (35, False), (37, True), (39, True)}

Supervised Learning Basics

- Overall idea: learn a **mapping** between **inputs** X and **outputs** Y
  - In math terms, learning a **function** $f(x) = y$
- The function is learned from a **dataset** of examples
  - $D = \{(x_1, y_1), (x_2, y_2), \ldots (x_n, y_n)\}$
  - The dataset contains **pairs of inputs and outputs**
  - Ex: speed $\rightarrow$ whether you get a speeding ticket
    - {(30, False), (33, False), (35, False), (37, True), (39, True)}
- Goal: learn the function that **best matches the dataset**

Learning a Function

Learning a Function

- We want to find a function $f : X \rightarrow Y$ such that...

Learning a Function

- We want to find a function $f : X \to Y$ such that...
  - $\hat{y}_i = f(x_i)$ is **"close"** to the true y_i for all $(x_i, y_i) \in D$

Learning a Function

- We want to find a function $f : X \to Y$ such that...
  - $\hat{y}_i = f(x_i)$ is **"close"** to the true y_i for all $(x_i, y_i) \in D$
    - $\hat{y}$, pronounced "y-hat", is the **predicted value** of y

Learning a Function

- We want to find a function $f : X \rightarrow Y$ such that...
  - $\hat{y}_i = f(x_i)$ is **"close"** to the true y_i for all $(x_i, y_i) \in D$
    - $\hat{y}$, pronounced "y-hat", is the **predicted value** of y
    - $\in$ means "is an element of" or just "in"

Learning a Function

- We want to find a function $f : X \rightarrow Y$ such that...
  - $\hat{y}_i = f(x_i)$ is **"close"** to the true y_i for all $(x_i, y_i) \in D$
    - $\hat{y}$, pronounced "y-hat", is the **predicted value** of y
    - $\in$ means "is an element of" or just "in"
  - The function f also **generalizes** well to **new data** (examples not in D)

Learning a Function

- We want to find a function $f : X \rightarrow Y$ such that...
  - $\hat{y}_i = f(x_i)$ is **"close"** to the true y_i for all $(x_i, y_i) \in D$
    - $\hat{y}$, pronounced "y-hat", is the **predicted value** of y
    - $\in$ means "is an element of" or just "in"
  - The function f also **generalizes** well to **new data** (examples not in D)
- How do we know **what kind of function** to learn?

Learning a Function

- We want to find a function $f: X \to Y$ such that...
  - $\hat{y}_i = f(x_i)$ is **"close"** to the true y_i for all $(x_i, y_i) \in D$
    - $\hat{y}$, pronounced "y-hat", is the **predicted value** of y
    - $\in$ means "is an element of" or just "in"
  - The function f also **generalizes** well to **new data** (examples not in D)
- How do we know **what kind of function** to learn?
  - Infinitely many to choose from

Learning a Function

- We want to find a function $f : X \rightarrow Y$ such that...
  - $\hat{y}_i = f(x_i)$ is **"close"** to the true y_i for all $(x_i, y_i) \in D$
    - $\hat{y}$, pronounced "y-hat", is the **predicted value** of y
    - $\in$ means "is an element of" or just "in"
  - The function f also **generalizes** well to **new data** (examples not in D)
- How do we know **what kind of function** to learn?
  - Infinitely many to choose from
  - Solution: learn the weights of a **parameterized function**

Parameterized Functions

Parameterized Functions

- A learning searches for a function f in a space of **possible functions**
- Parameters define a **family** of functions that share a common form
  - θ: general symbol for parameters/weights (usually represents **several**)
  - $\hat{y} = f(x; \theta)$: the function $f(x)$, **given parameters** θ
- Example: the **family of linear functions** $f(x) = mx + b$
  - $\theta = \{m, b\}$
  - This defines **all possible lines** (with different slopes and intercepts)
- Later: Neural Networks define their own family of functions

Loss Function

Loss Function

- We need a way to **measure how close** our parameterized function is to the "true" input/output mapping
  - In other words, we want to measure the **error** of our model

Loss Function

- We need a way to **measure how close** our parameterized function is to the "true" input/output mapping
    - In other words, we want to measure the **error** of our model
- "Loss Function": a measure of how much the **predicted output** $\hat{y}$ **diverges** from the **true output** y
    - $\ell(\hat{y}, y) = \ell(f(x, \theta), y)$
    - Common example: **squared error** $\ell(\hat{y}, y) = (\hat{y} - y)^2$ ((Q: why squared?))

Loss Function

- We need a way to **measure how close** our parameterized function is to the "true" input/output mapping
  - In other words, we want to measure the **error** of our model
- "Loss Function": a measure of how much the **predicted output** $\hat{y}$ **diverges** from the **true output** y
  - $\ell(\hat{y}, y) = \ell(f(x, \theta), y)$
  - Common example: **squared error** $\ell(\hat{y}, y) = (\hat{y} - y)^2$ ((Q: why squared?))
- We always want to **minimize the loss/error**
  - This is a type of **optimization problem**, which is a huge subfield of math

Loss Minimization

Loss Minimization

- Optimization problem: find the values of the **parameters** θ that **minimize the loss function**

Loss Minimization

- Optimization problem: find the values of the **parameters** θ that **minimize the loss function**
    - We will view loss as a **function of the parameters:** $\ell(\theta) := \ell(f(x, \theta), y)$

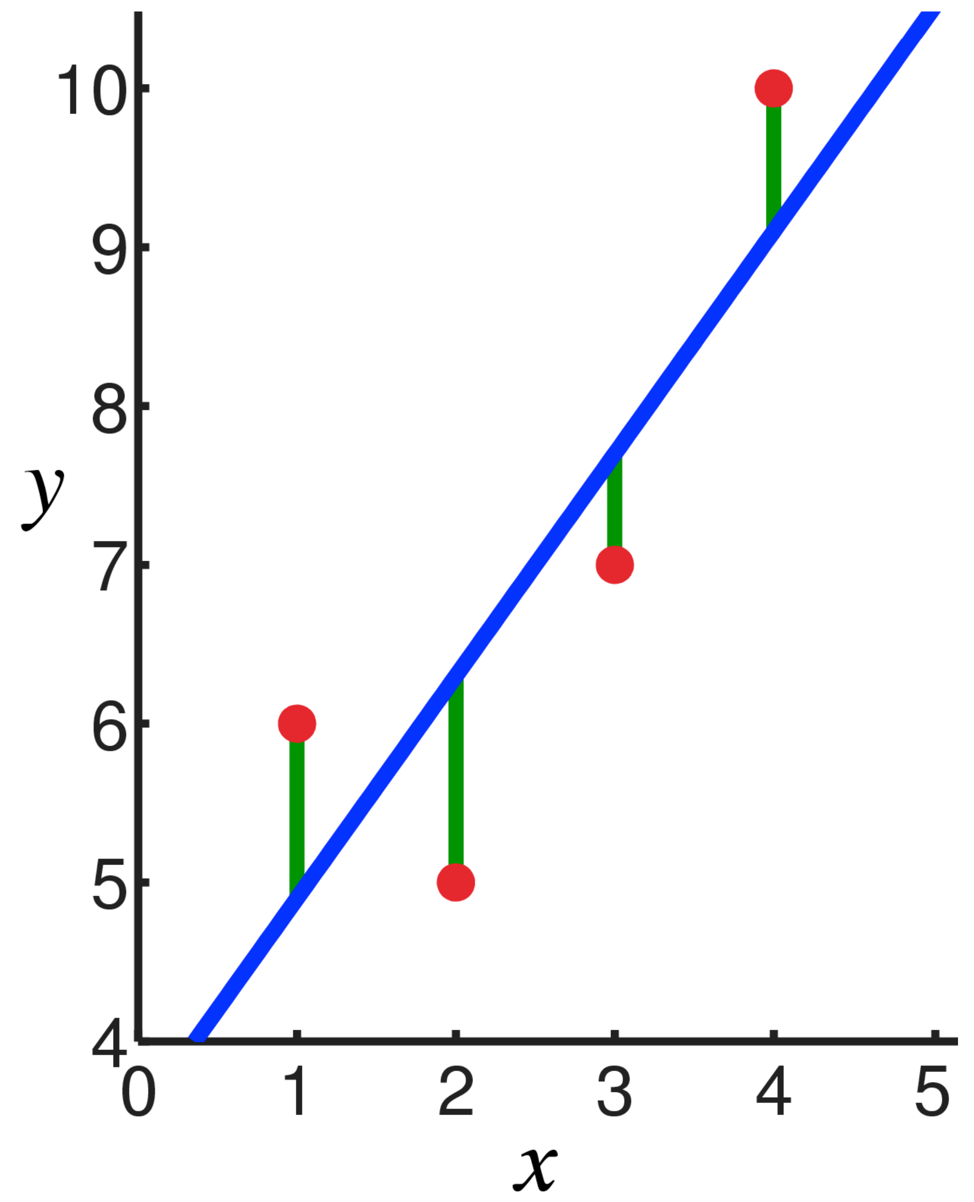
Loss Minimization

- Optimization problem: find the values of the **parameters** θ that **minimize the loss function**
  - We will view loss as a **function of the parameters:** $\ell(\theta) := \ell(f(x, \theta), y)$
  - In math terms, $\theta*$ are the **optimal parameters** $\theta* = \arg\min_{\theta} \ell(\theta)$

Loss Minimization

- Optimization problem: find the values of the **parameters** θ that **minimize the loss function**
  - We will view loss as a **function of the parameters:** $\ell(\theta) := \ell(f(x, \theta), y)$
  - In math terms, θ^* are the **optimal parameters** $\theta^* = \arg\min_{\theta} \ell(\theta)$
- Example: **Linear Regression** ("Least-Squares" method)

$$m^*, b^* = \arg\min_{m,b} \sum_i ((mx_i + b) - y_i)^2$$



Guessing a number

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**
  - (Trivially easy for humans, we **don't actually need** Gradient Descent to solve it)

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**
  - (Trivially easy for humans, we **don't actually need** Gradient Descent to solve it)
- Idea:

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**
  - (Trivially easy for humans, we **don't actually need** Gradient Descent to solve it)
- Idea:
  - You give me any **input number** (we'll call it x)

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**
  - (Trivially easy for humans, we **don't actually need** Gradient Descent to solve it)
- Idea:
  - You give me any **input number** (we'll call it x)
  - I'll **add a secret number** to it (call it θ)

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**
  - (Trivially easy for humans, we **don't actually need** Gradient Descent to solve it)
- Idea:
  - You give me any **input number** (we'll call it x)
  - I'll **add a secret number** to it (call it θ)
  - I'll tell you the **output number** (y)

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**
  - (Trivially easy for humans, we **don't actually need** Gradient Descent to solve it)
- Idea:
  - You give me any **input number** (we'll call it x)
  - I'll **add a secret number** to it (call it θ)
  - I'll tell you the **output number** (y)
  - You have to **deduce the value of the secret number**

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**
  - (Trivially easy for humans, we **don't actually need** Gradient Descent to solve it)
- Idea:
  - You give me any **input number** (we'll call it x)
  - I'll **add a secret number** to it (call it θ)
  - I'll tell you the **output number** (y)
  - You have to **deduce the value of the secret number**
- What is the **equation for the function** that we're applying?

Guessing a number

- We'll illustrate Gradient Descent with a **(very) simple number game**
    - (Trivially easy for humans, we **don't actually need** Gradient Descent to solve it)
- Idea:
    - You give me any **input number** (we'll call it x)
    - I'll **add a secret number** to it (call it θ)
    - I'll tell you the **output number** (y)
    - You have to **deduce the value of the secret number**
- What is the **equation for the function** that we're applying?
    - $\hat{y} = f(x) = x + \theta$

Process

Process

- Here are some **input-output pairs** that define our dataset:
    - $\{(2, 4), (3, 5), (5, 7), (8, 10)\}$

Process

- Here are some **input-output pairs** that define our dataset:
    - $\{(2, 4), (3, 5), (5, 7), (8, 10)\}$
- You can **see that $\theta = 2$,** but how would we **learn this algorithmically?**

Process

- Here are some **input-output pairs** that define our dataset:
  - $\{(2, 4), (3, 5), (5, 7), (8, 10)\}$
- You can **see that $\theta = 2$,** but how would we **learn this algorithmically?**
- First define a **parameterized function** $f(x, \theta)$
  - Model prediction: $\hat{y} = f(x, \theta) = x + \theta$

Process

- Here are some **input-output pairs** that define our dataset:
  - $\{(2, 4), (3, 5), (5, 7), (8, 10)\}$
- You can **see that $\theta = 2$,** but how would we **learn this algorithmically?**
- First define a **parameterized function** $f(x, \theta)$
  - Model prediction: $\hat{y} = f(x, \theta) = x + \theta$
- Then define a **loss/error function**
  - We'll use **Squared Error:** $\ell(\hat{y}, y) = (\hat{y} - y)^2 = (f(x, \theta) - y)^2$

Process

- Here are some **input-output pairs** that define our dataset:
  - $\{(2, 4), (3, 5), (5, 7), (8, 10)\}$
- You can **see that $\theta = 2$,** but how would we **learn this algorithmically?**
- First define a **parameterized function** $f(x, \theta)$
  - Model prediction: $\hat{y} = f(x, \theta) = x + \theta$
- Then define a **loss/error function**
  - We'll use **Squared Error:** $\ell(\hat{y}, y) = (\hat{y} - y)^2 = (f(x, \theta) - y)^2$
- Lastly **learn the optimal value of θ** (i.e. the value that **minimizes the loss**)

Loss function

Loss function

- We cast loss as a **function of the parameter(s)** θ

Loss function

- We cast loss as a **function of the parameter(s)** θ
  - $\ell(f(x, \theta), y) = (f(x, \theta) - y)^2 = (x + \theta - y)^2$

Loss function

- We cast loss as a **function of the parameter(s)** θ
  - $\ell(f(x, \theta), y) = (f(x, \theta) - y)^2 = (x + \theta - y)^2$
  - x, y are treated as **constants provided by the data**

Loss function

- We cast loss as a **function of the parameter(s)** θ
  - $\ell(f(x, \theta), y) = (f(x, \theta) - y)^2 = (x + \theta - y)^2$
  - x, y are treated as **constants provided by the data**
- Plug in the datapoint $(x = 2, y = 4)$

Loss function

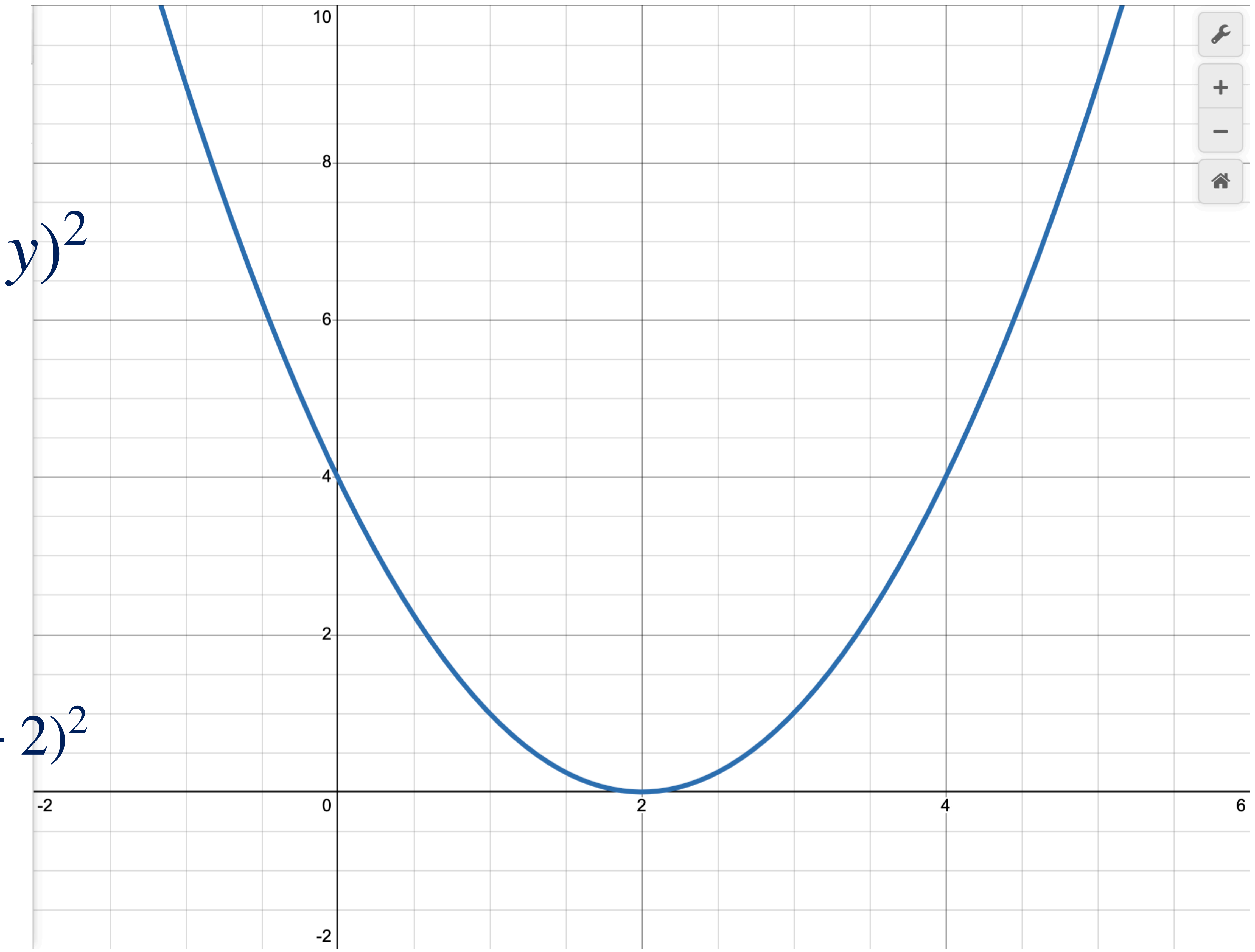
- We cast loss as a **function of the parameter(s)** θ
  - $\ell(f(x,\theta),y) = (f(x,\theta) - y)^2 = (x + \theta - y)^2$
  - x, y are treated as **constants provided by the data**
- Plug in the datapoint $(x = 2, y = 4)$
  - $\ell(f(x,\theta),y) = (x + \theta - y)^2 = (2 + \theta - 4)^2 = (\theta - 2)^2$

Loss function

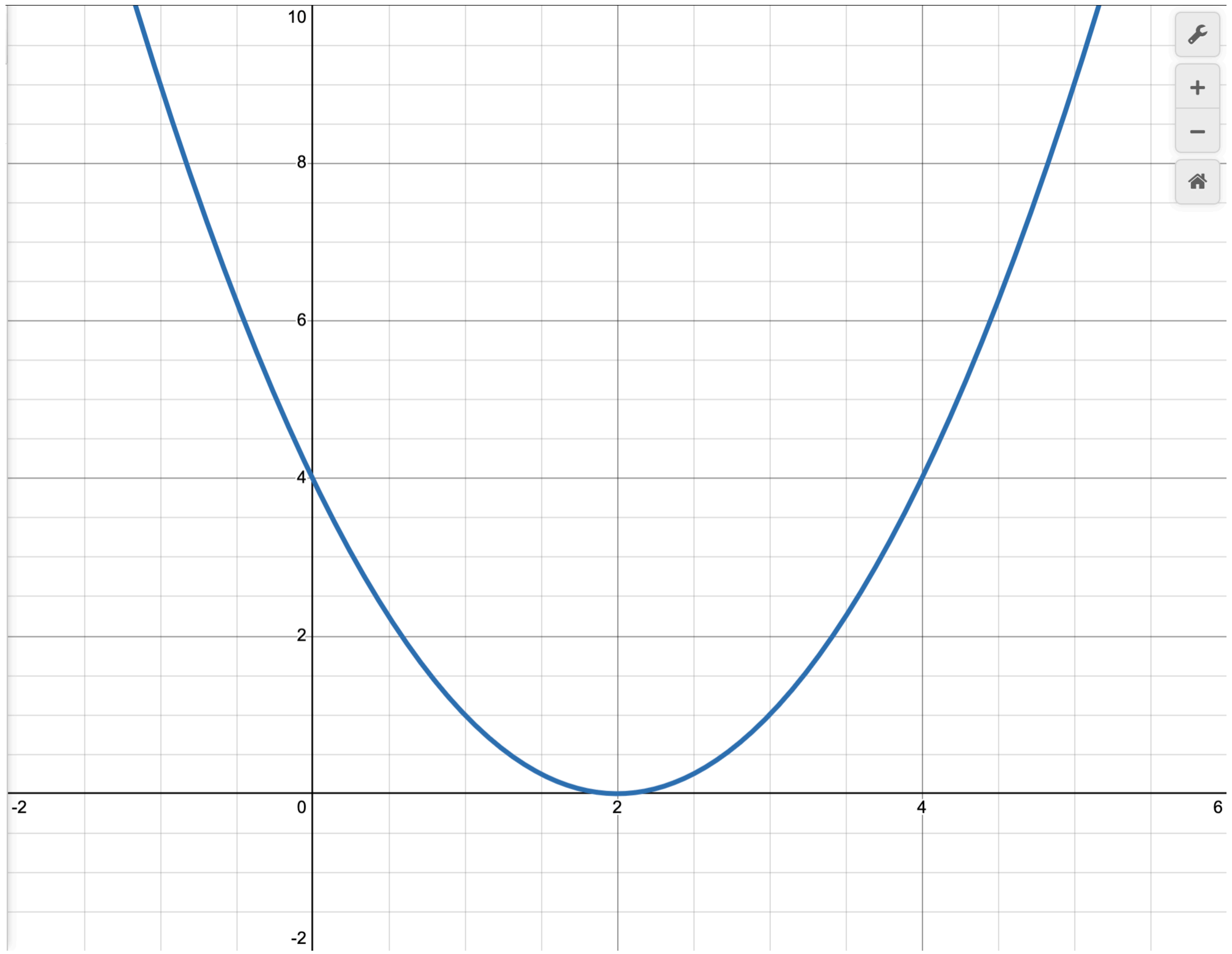
- We cast loss as a **function of the parameter(s)** θ
  - $\ell(f(x,\theta),y) = (f(x,\theta) - y)^2 = (x + \theta - y)^2$
  - x, y are treated as **constants provided by the data**
- Plug in the datapoint $(x = 2,\, y = 4)$
  - $\ell(f(x,\theta),y) = (x + \theta - y)^2 = (2 + \theta - 4)^2 = (\theta - 2)^2$
  - We can **plot this loss curve!**

Loss function

- We cast loss as a **function of the parameter(s)** θ
  - $\ell(f(x,\theta), y) = (f(x,\theta) - y)^2 = (x + \theta - y)^2$
  - x, y are treated as **constants provided by the data**
- Plug in the datapoint $(x = 2,\ y = 4)$
  - $\ell(f(x,\theta), y) = (x + \theta - y)^2 = (2 + \theta - 4)^2 = (\theta - 2)^2$
  - We can **plot this loss curve!**

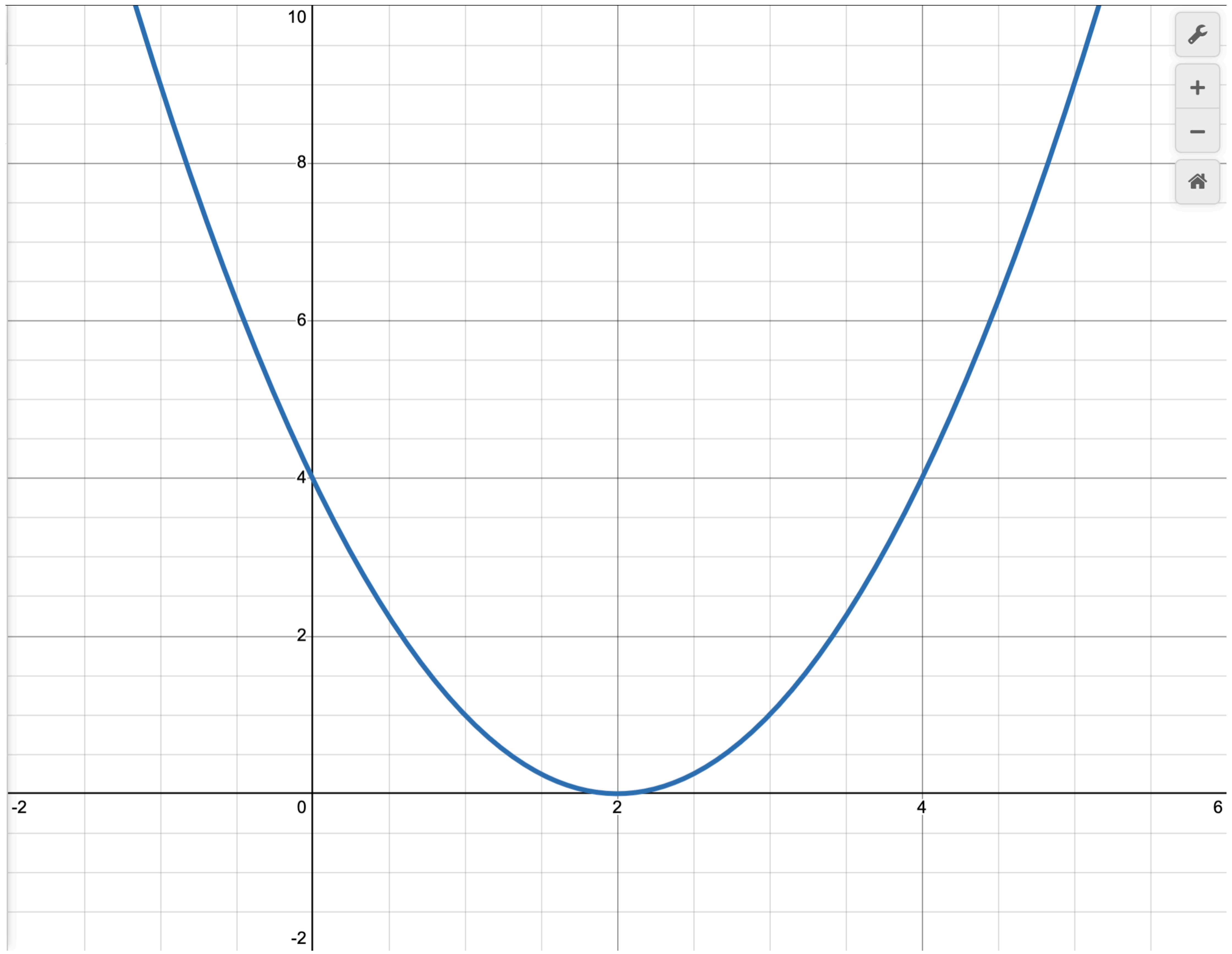


Loss function



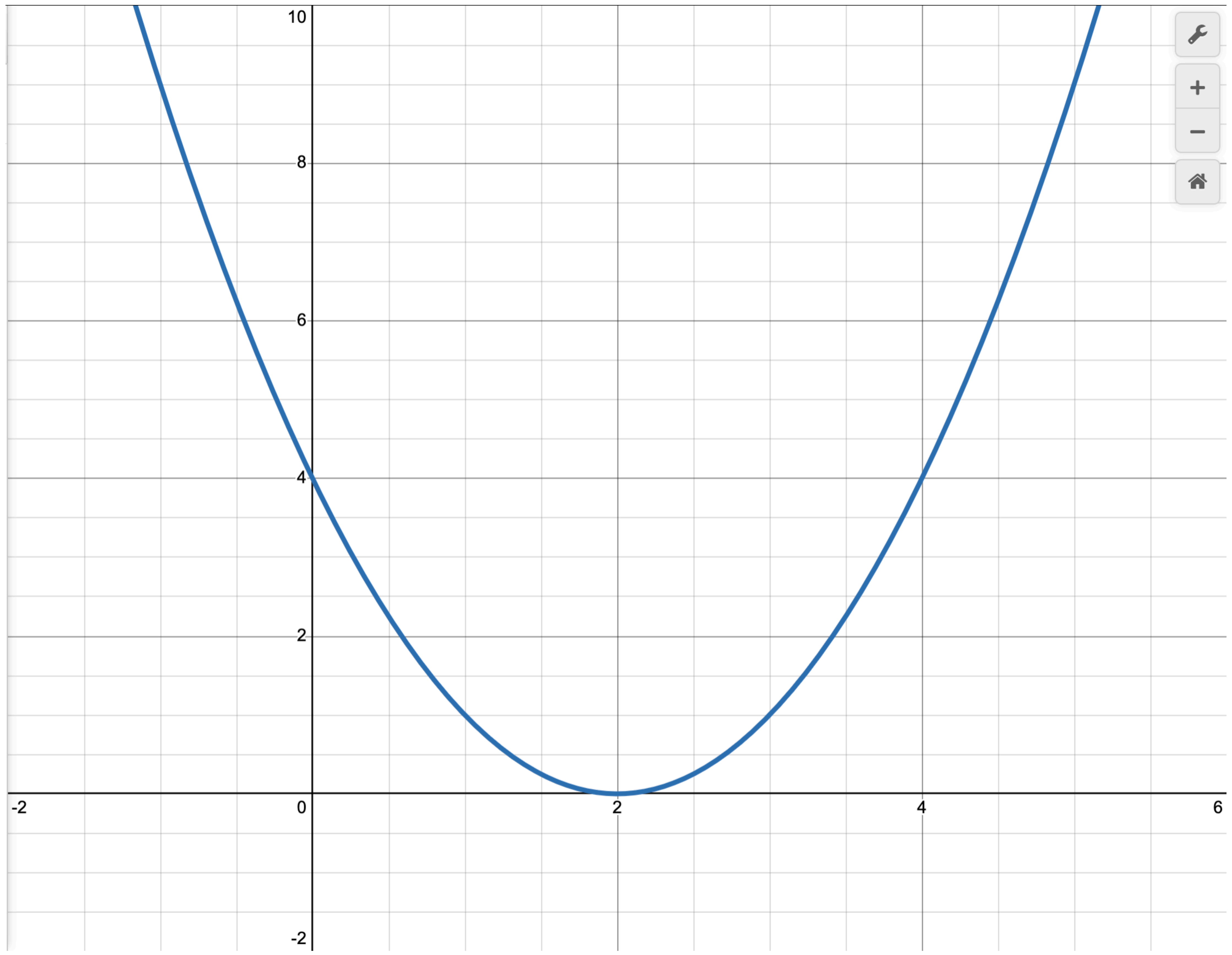
Loss function

- This curve shows the **properties we expect**
  - Loss is **minimized** where $\theta = 2$
  - Loss **grows large** the farther θ is from the true value



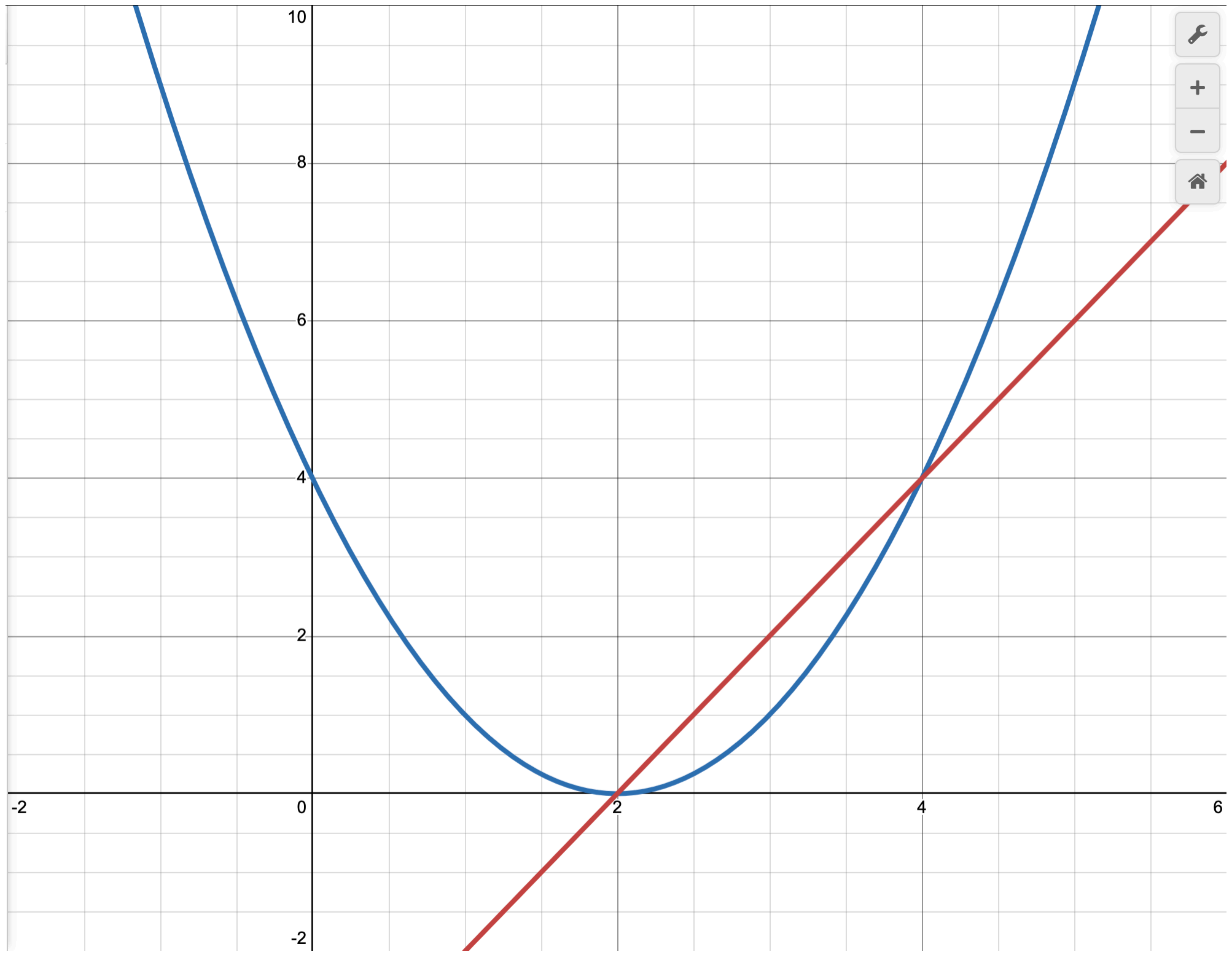
Loss function

- This curve shows the **properties we expect**
  - Loss is **minimized** where $\theta = 2$
  - Loss **grows large** the farther θ is from the true value
- Gradient Descent idea: minimize loss by **following the slope (i.e. "gradient") of the loss function**
  - $\frac{d}{d\theta}(\theta - 2)^2 = 2\theta - 4$

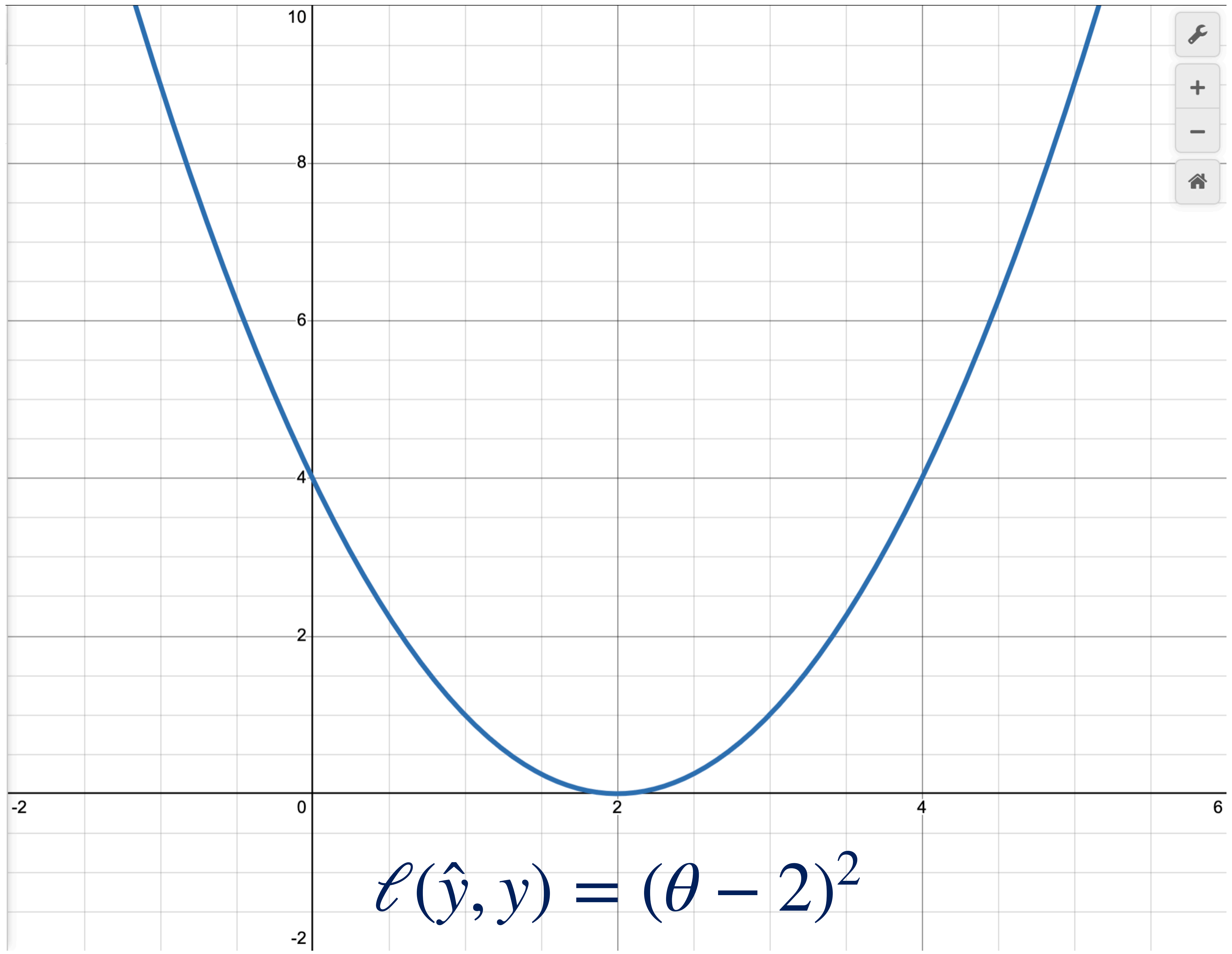


Loss function

- This curve shows the **properties we expect**
  - Loss is **minimized** where $\theta = 2$
  - Loss **grows large** the farther θ is from the true value
- Gradient Descent idea: minimize loss by **following the slope (i.e. "gradient") of the loss function**
  - $\frac{d}{d\theta}(\theta - 2)^2 = 2\theta - 4$



Loss function

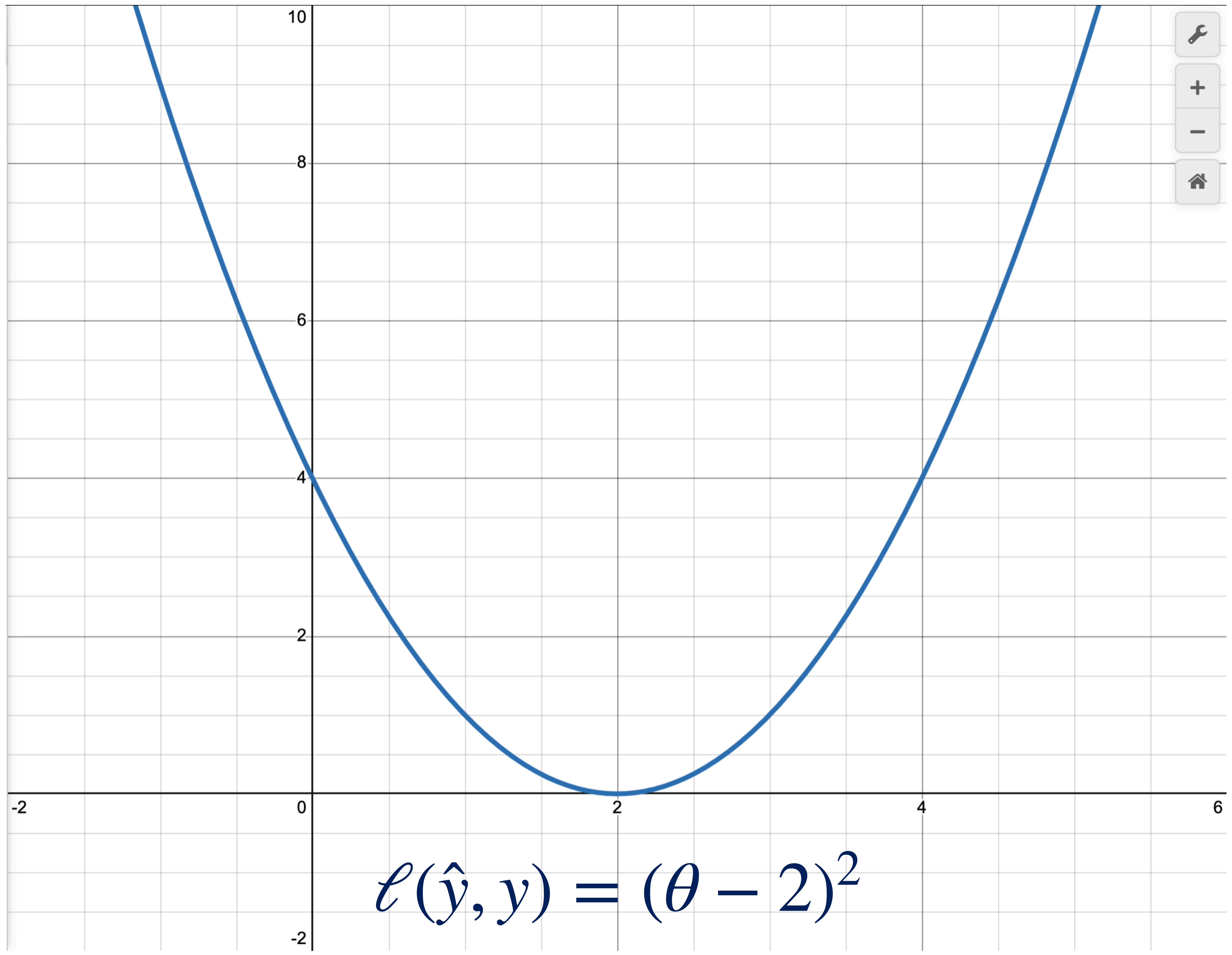


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Loss function

- **NOTE**: for this example, **every datapoint** gives us the **exact same loss curve**
  - $(\theta - 2)^2 = (2 + \theta - 4)^2$
  - $= (3 + \theta - 5)^2$
  - $= (5 + \theta - 7)^2$
  - ...etc.

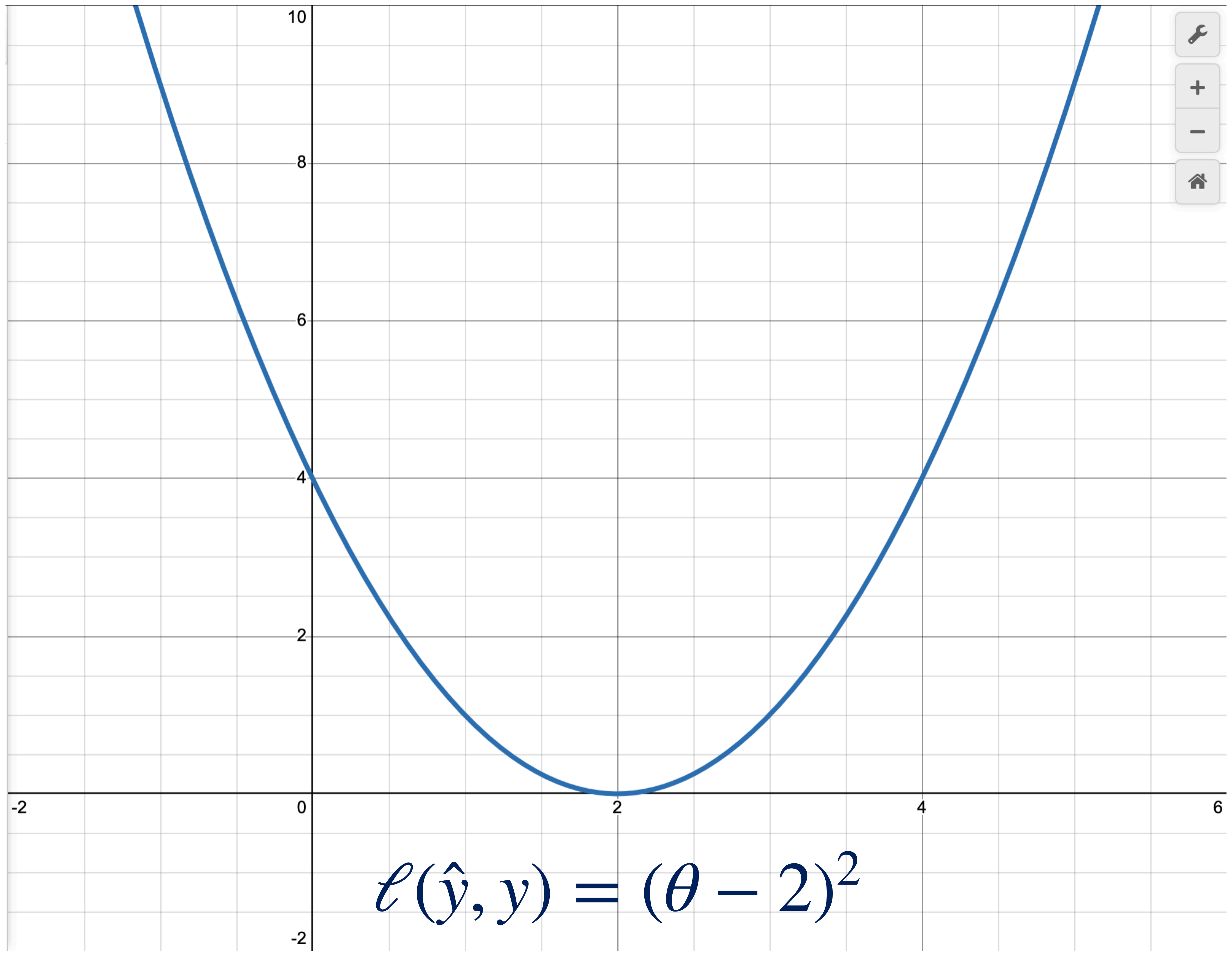


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Loss function

- **NOTE**: for this example, **every datapoint** gives us the **exact same loss curve**
  - $(\theta - 2)^2 = (2 + \theta - 4)^2$
  - $= (3 + \theta - 5)^2$
  - $= (5 + \theta - 7)^2$
  - ...etc.
- This is **NOT always the case**
  - Will show an example later on

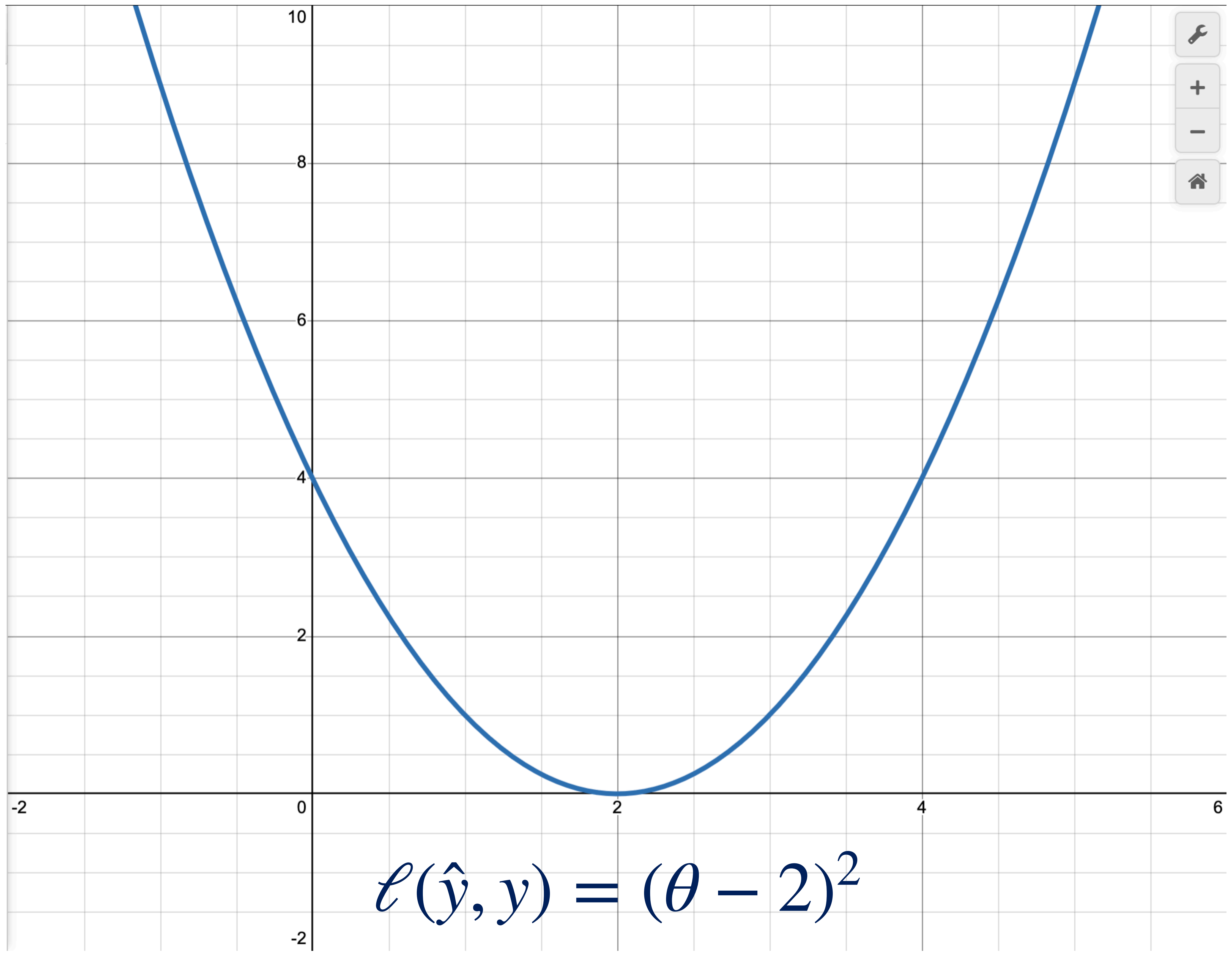


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

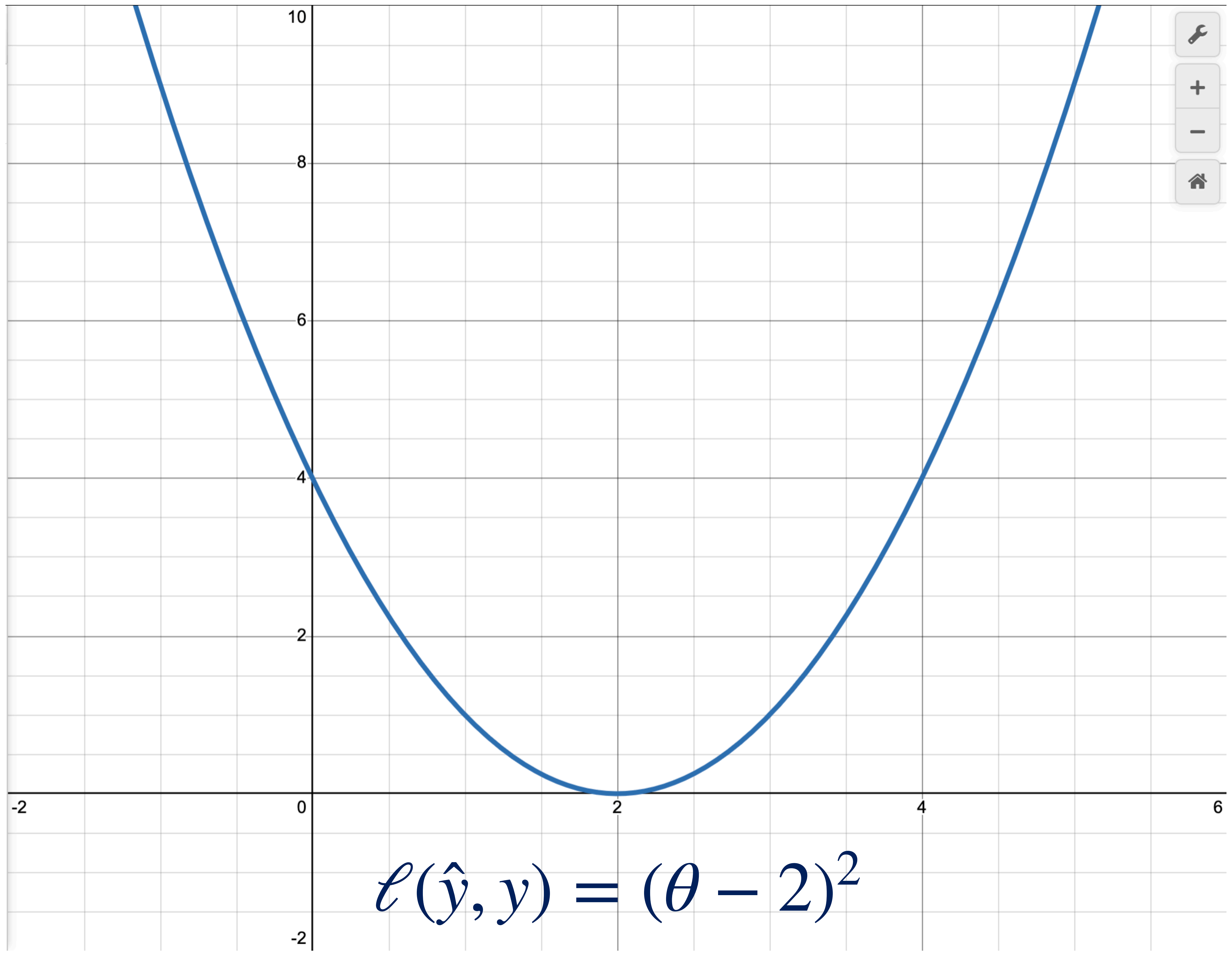
Loss function

- **NOTE**: for this example, **every datapoint** gives us the **exact same loss curve**
    - $(\theta - 2)^2 = (2 + \theta - 4)^2$
    - $= (3 + \theta - 5)^2$
    - $= (5 + \theta - 7)^2$
    - ...etc.
- This is **NOT always the case**
    - Will show an example later on
- For this example **ONLY**, solving for one datapoint solves the whole problem



$$\ell(\hat{y}, y) = (\theta - 2)^2$$
$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

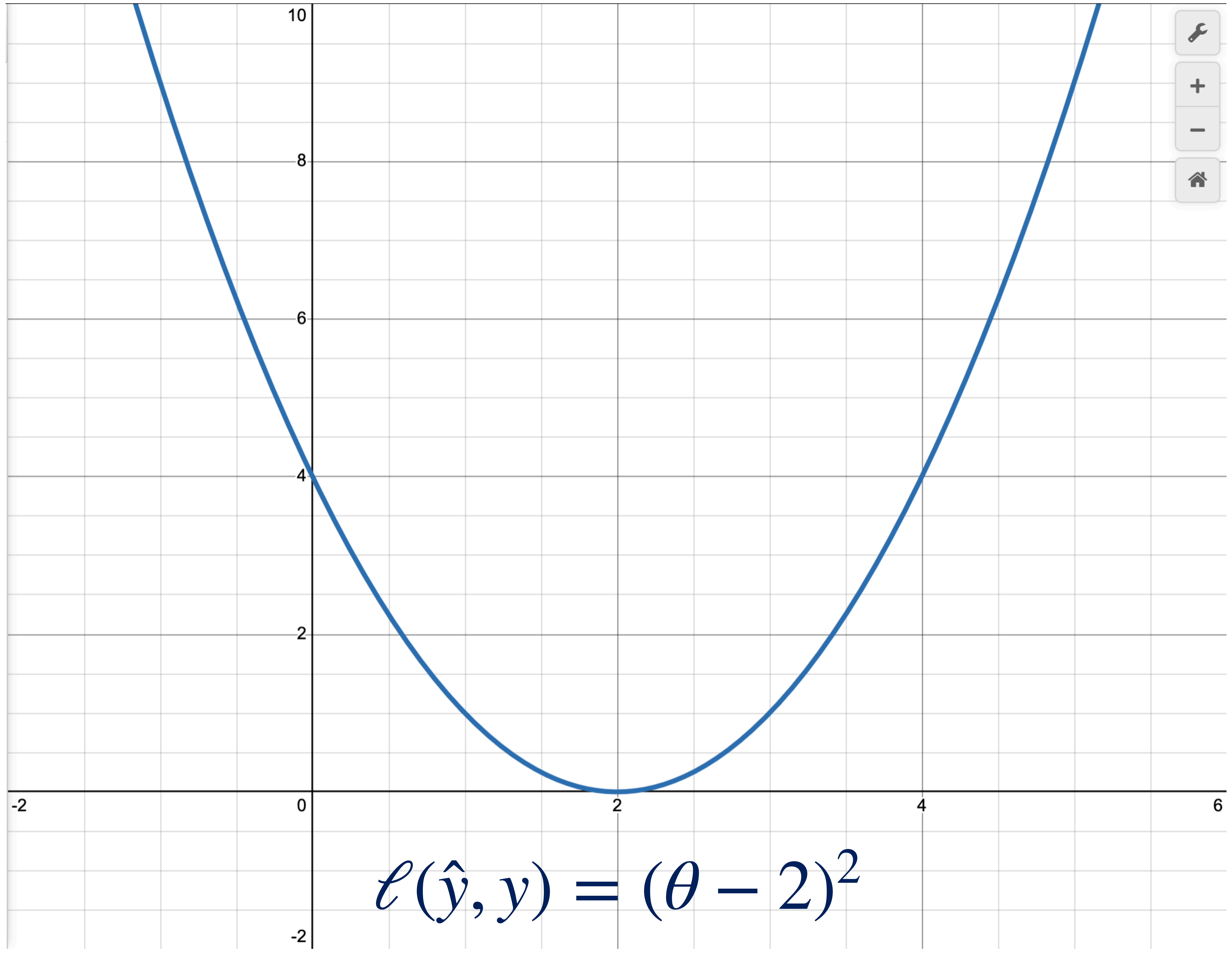


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- Gradient Descent is an **iterative algorithm**
  - I.e. you **repeatedly adjust** θ until the loss is minimized

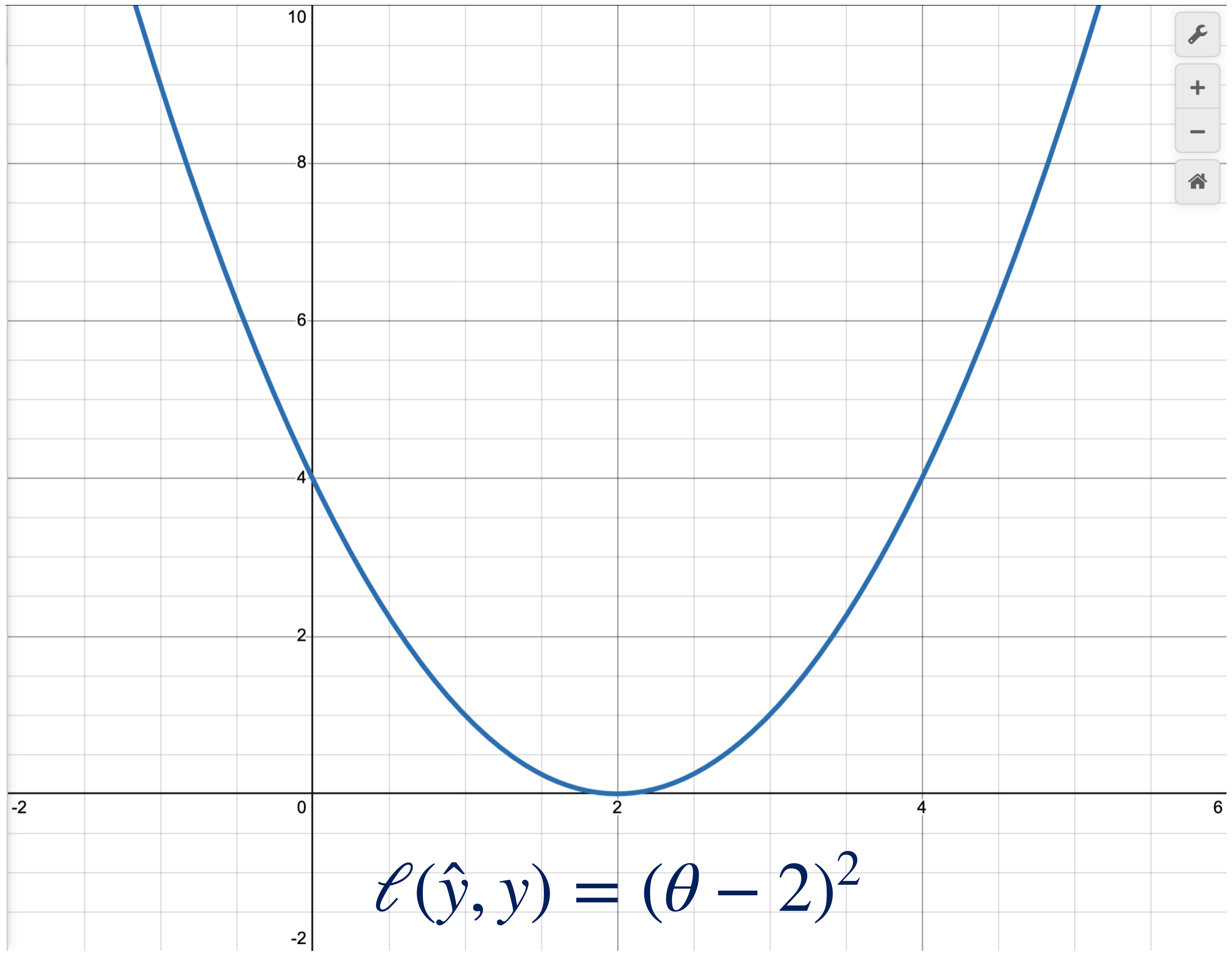


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- Gradient Descent is an **iterative algorithm**
  - I.e. you **repeatedly adjust** θ until the loss is minimized
- The **initial value of** θ is a design choice
  - Sometimes **randomly initialized**
  - Sometimes **set to zero**
  - We'll start with $\theta = 5$

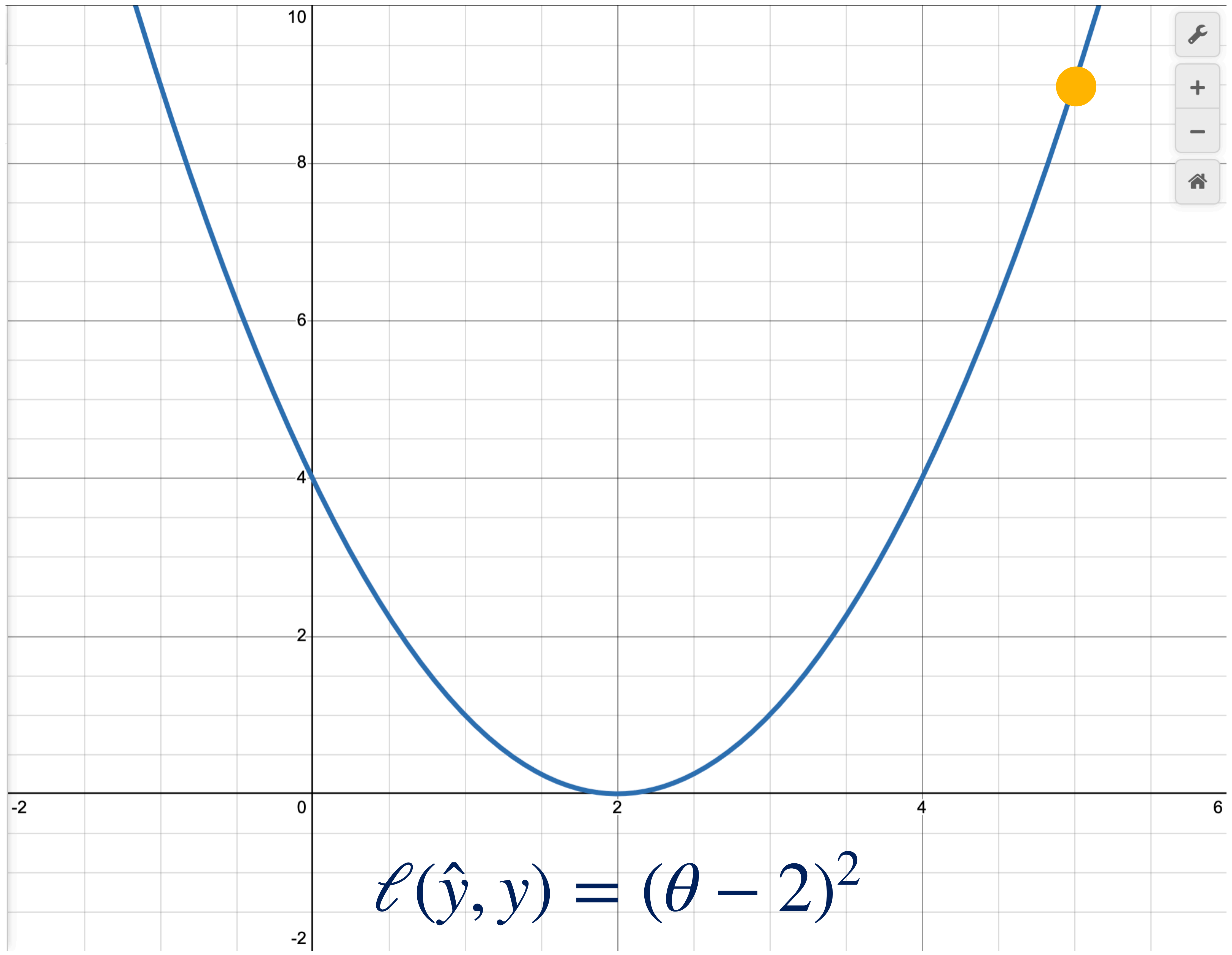


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

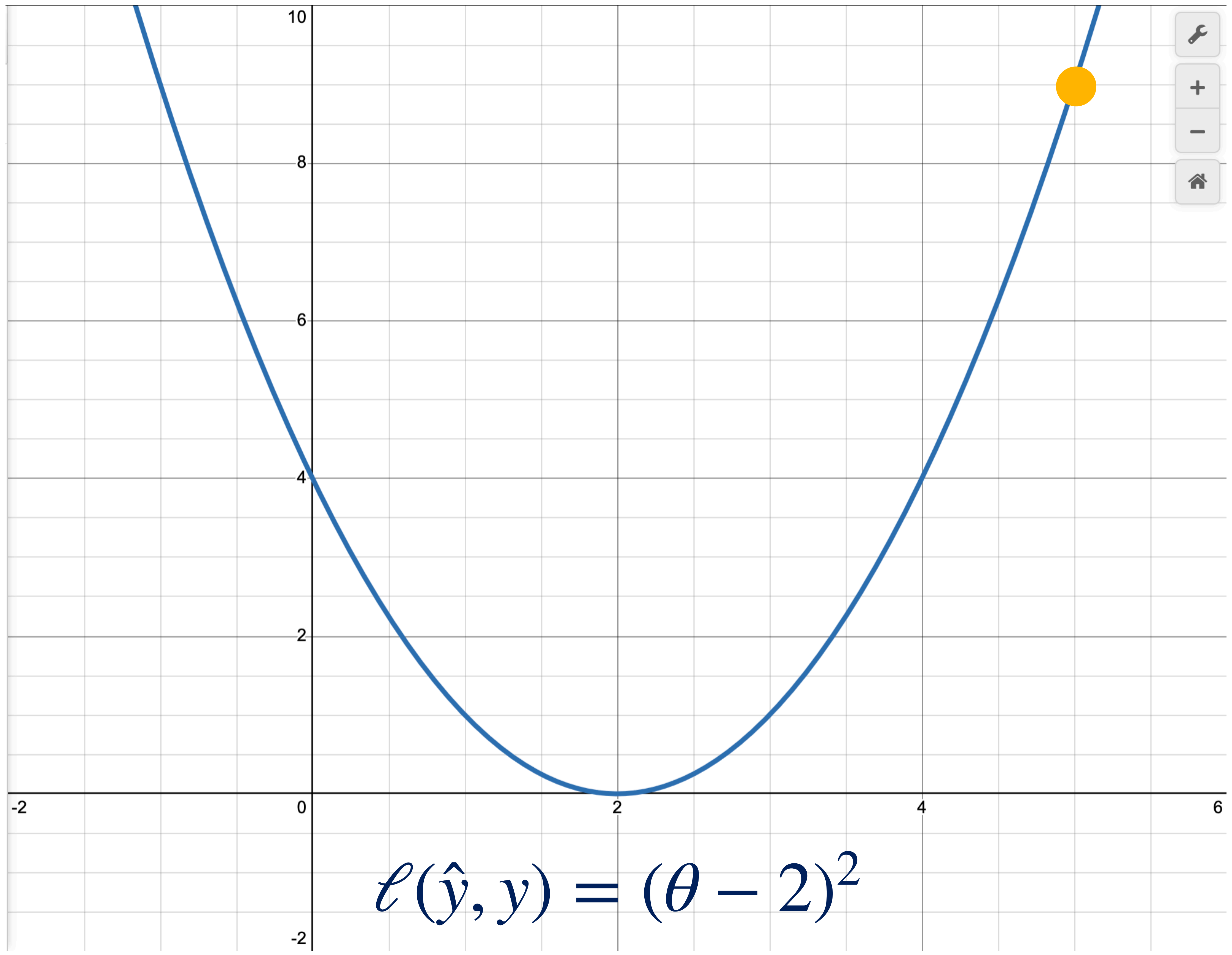
Gradient Descent (first try)

- Gradient Descent is an **iterative algorithm**
  - I.e. you **repeatedly adjust** θ until the loss is minimized
- The **initial value of** θ is a design choice
  - Sometimes **randomly initialized**
  - Sometimes **set to zero**
  - We'll start with $\theta = 5$



$$\ell(\hat{y}, y) = (\theta - 2)^2$$
$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

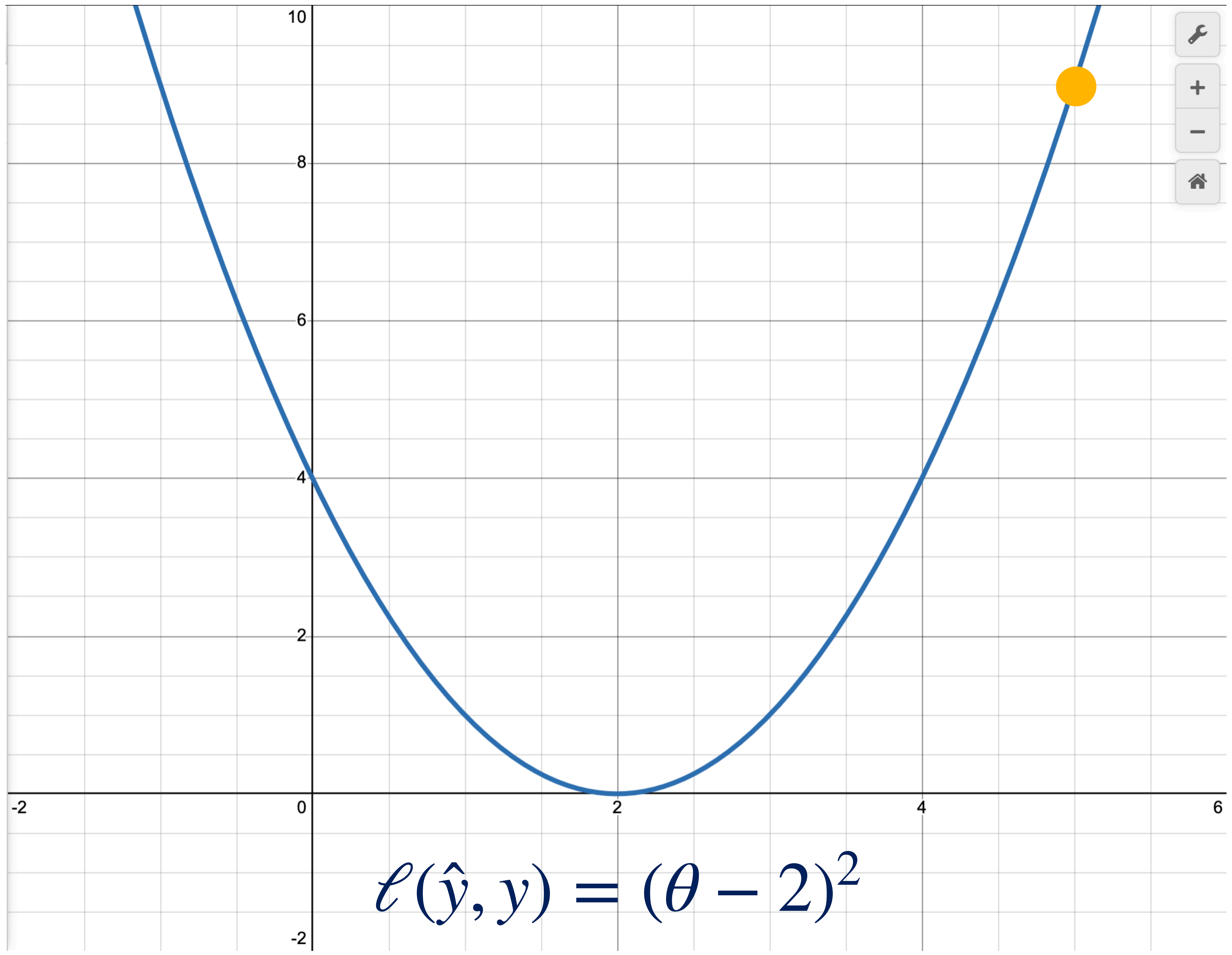


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- At each step of the algorithm:

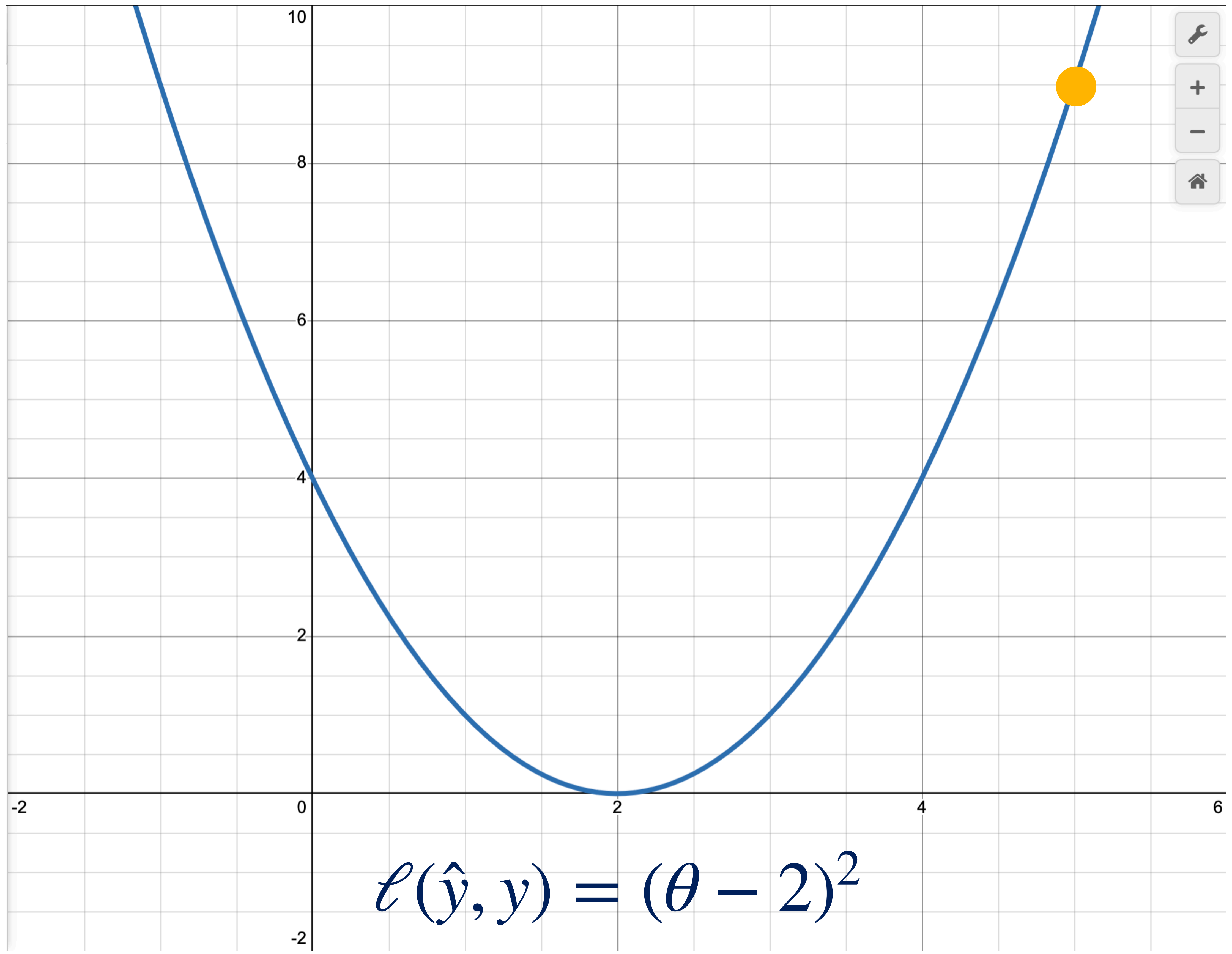


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- At each step of the algorithm:
  - Calculate the **slope at the current point**

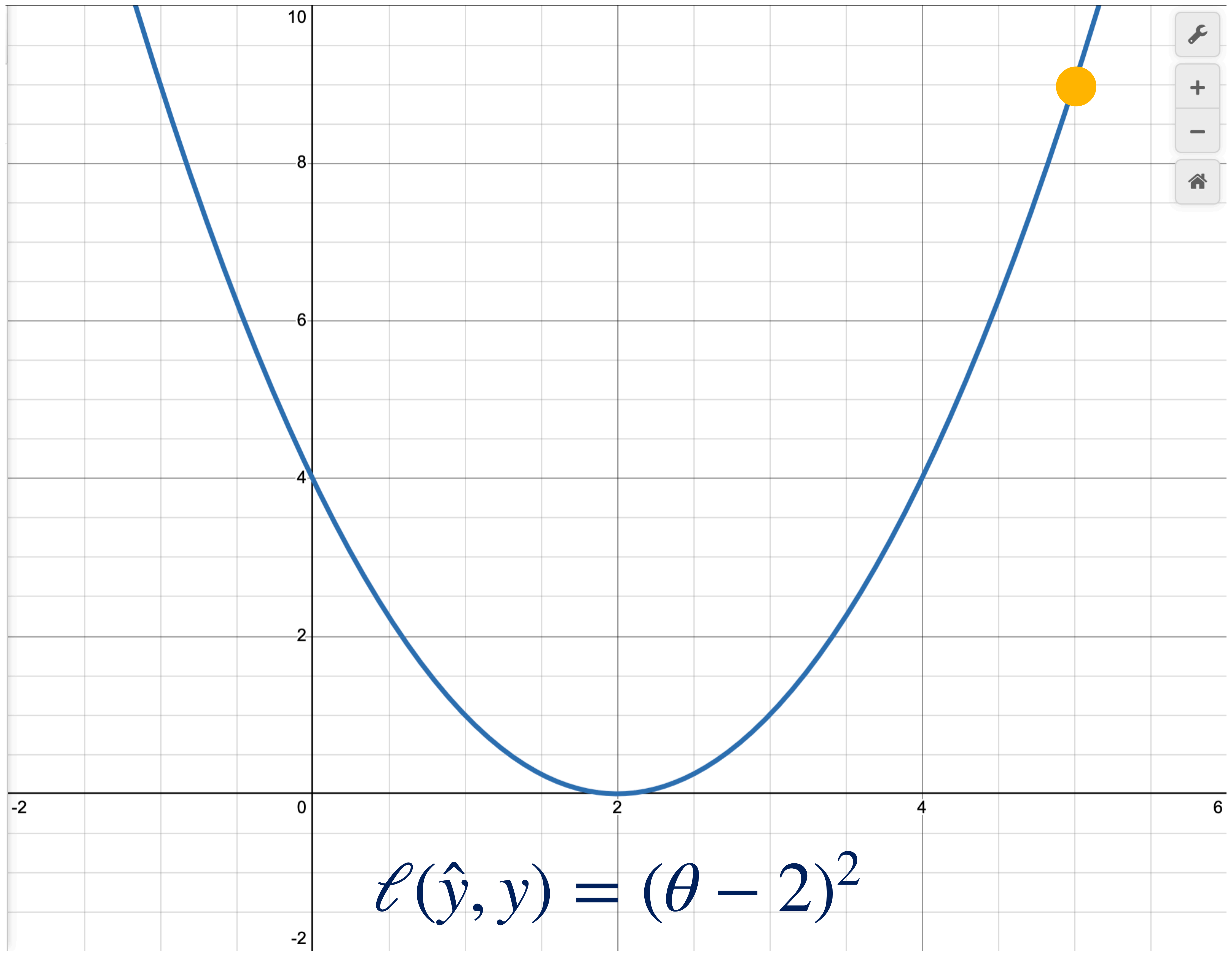


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- At each step of the algorithm:
  - Calculate the **slope at the current point**
  - Adjust θ by the **negative of the slope** (because we want to **minimize** the function)

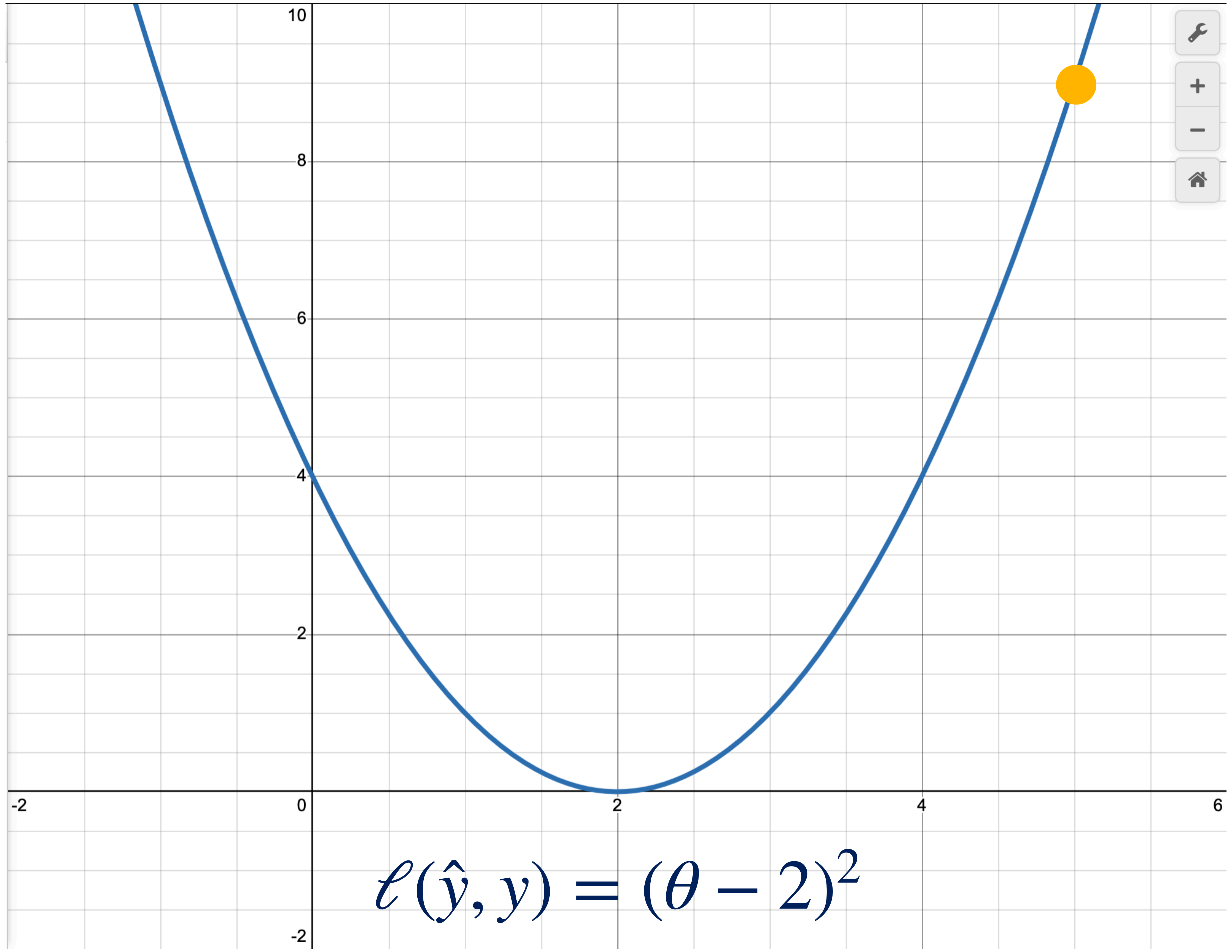


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- At each step of the algorithm:
  - Calculate the **slope at the current point**
  - Adjust θ by the **negative of the slope** (because we want to **minimize** the function)
- First step:

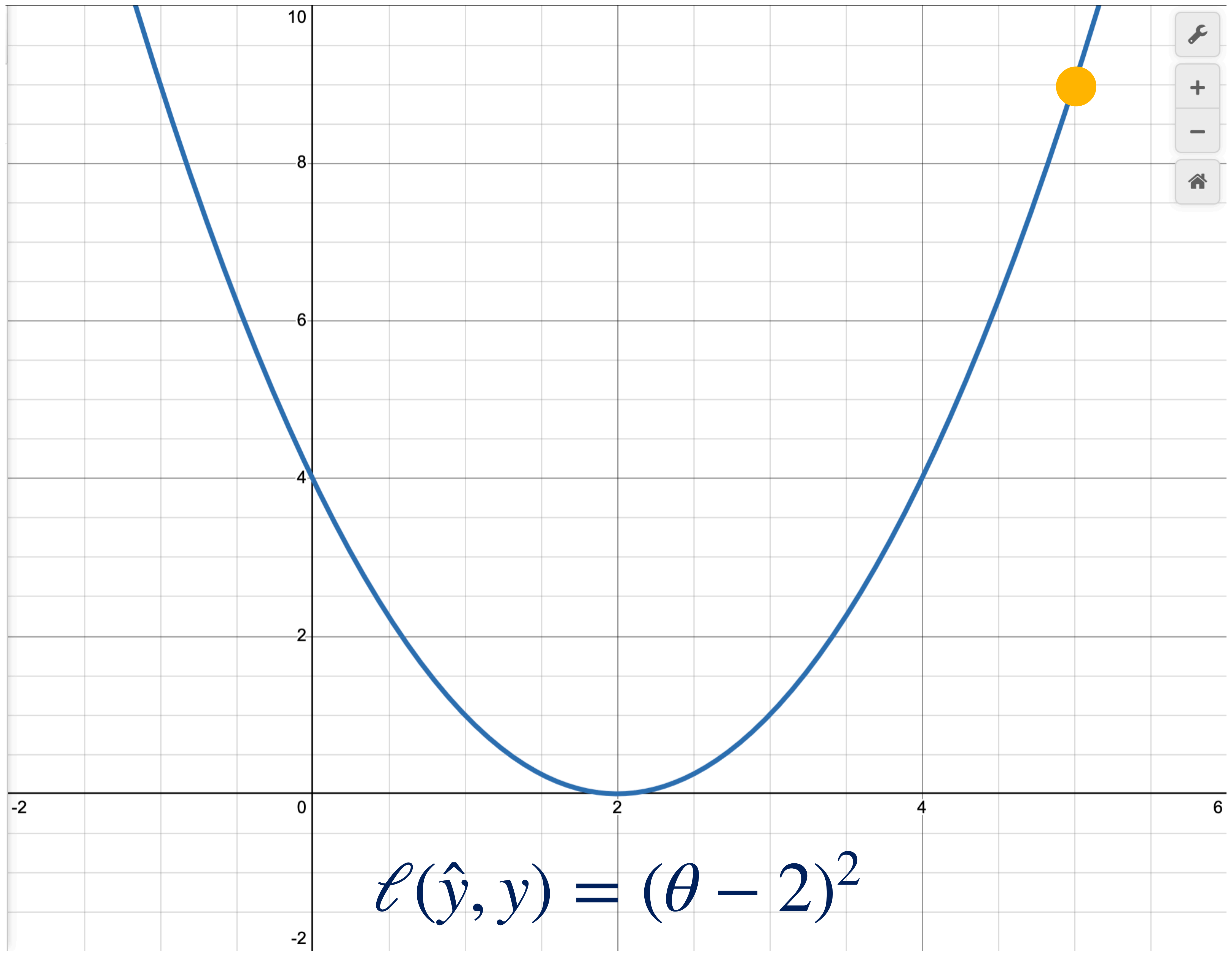


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- At each step of the algorithm:
  - Calculate the **slope at the current point**
  - Adjust θ by the **negative of the slope** (because we want to **minimize** the function)
- First step:
  - Plug in θ to derivative:

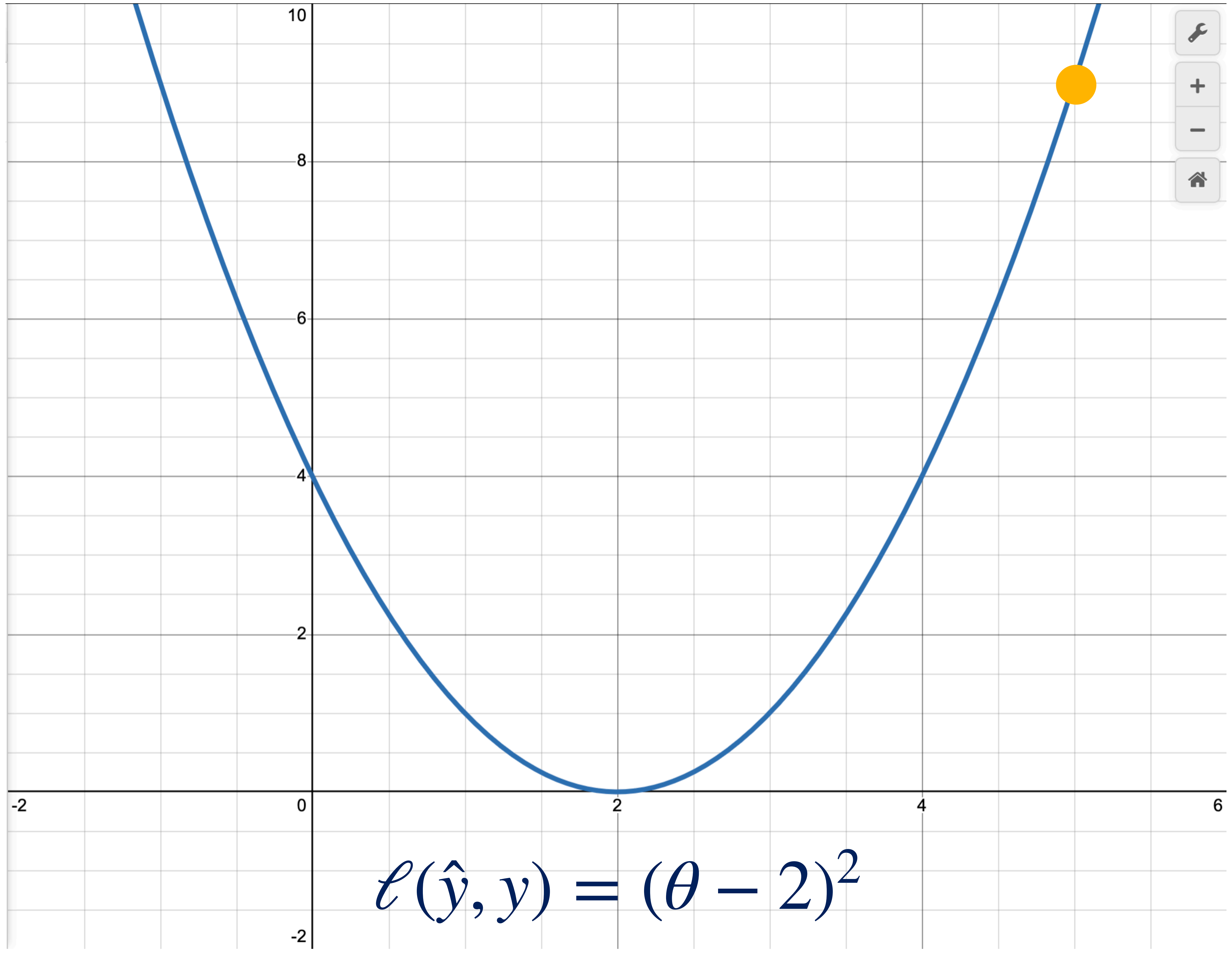


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- At each step of the algorithm:
  - Calculate the **slope at the current point**
  - Adjust θ by the **negative of the slope** (because we want to **minimize** the function)
- First step:
  - Plug in θ to derivative:
    - $2 \cdot 5 - 4 = 6$

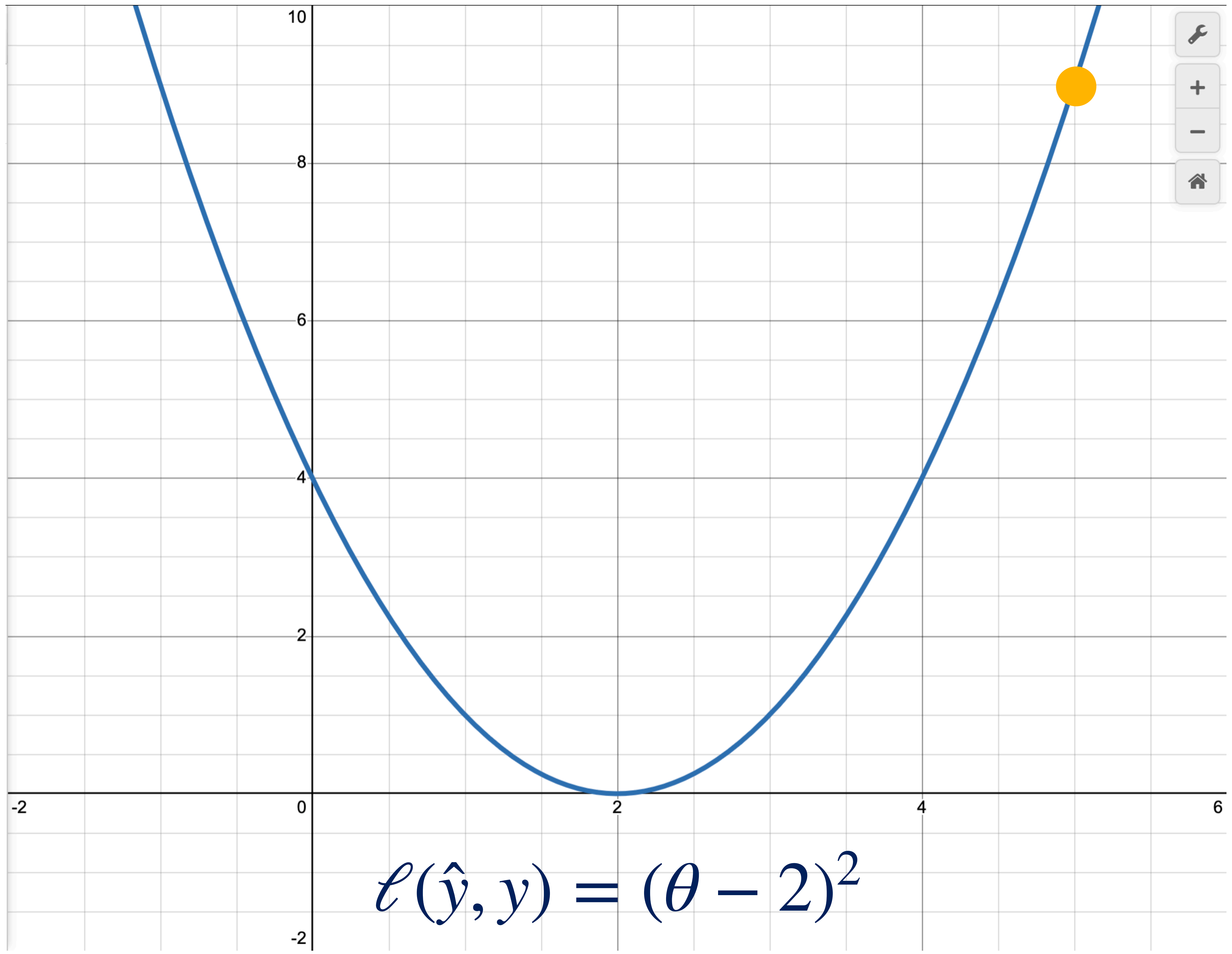


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- At each step of the algorithm:
  - Calculate the **slope at the current point**
  - Adjust θ by the **negative of the slope** (because we want to **minimize** the function)
- First step:
  - Plug in θ to derivative:
    - $2 \cdot 5 - 4 = 6$
  - Update θ:

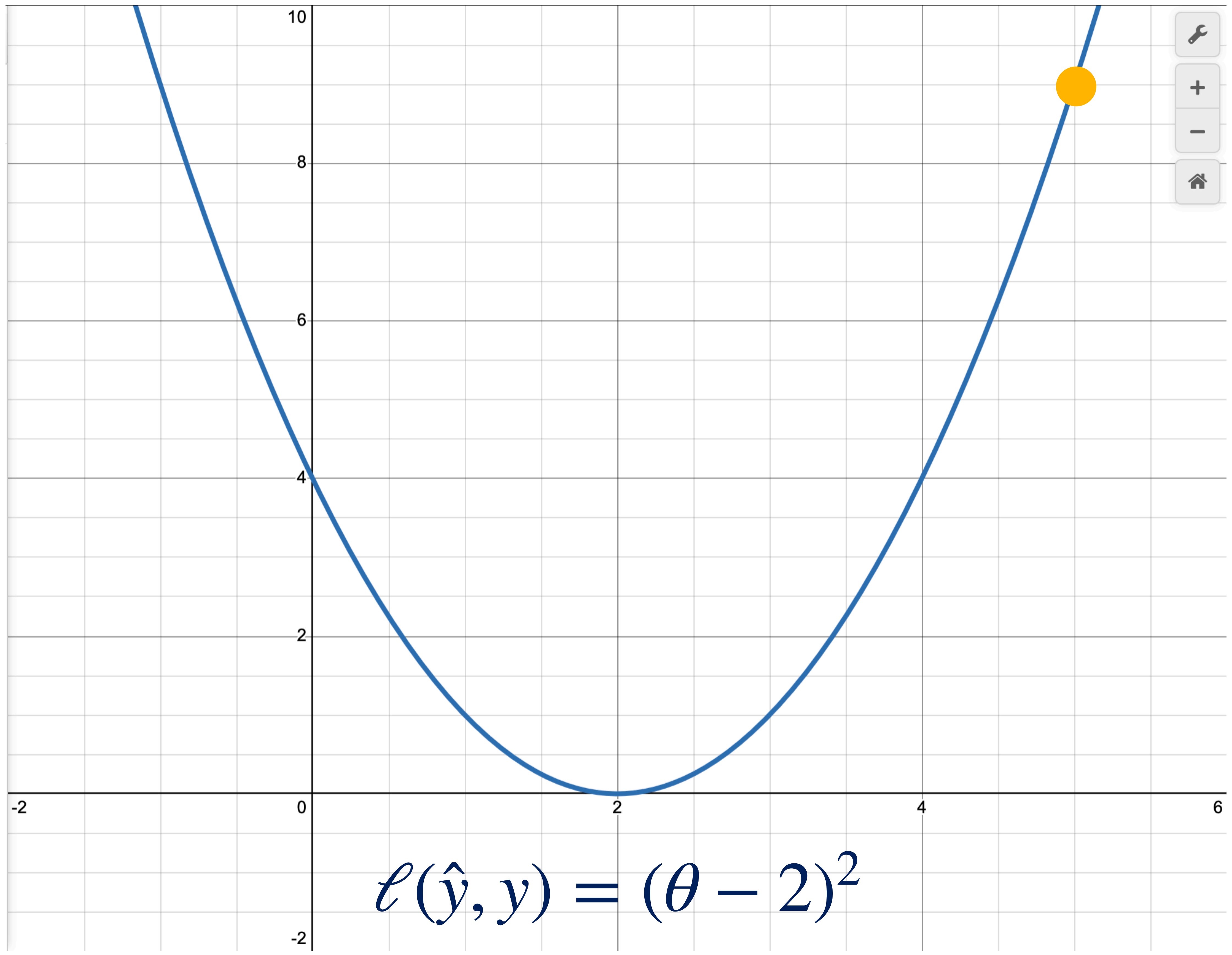


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- At each step of the algorithm:
  - Calculate the **slope at the current point**
  - Adjust θ by the **negative of the slope** (because we want to **minimize** the function)
- First step:
  - Plug in θ to derivative:
    - $2 \cdot 5 - 4 = 6$
  - Update θ:
    - $\theta_1 = \theta_0 - 6 = -1$

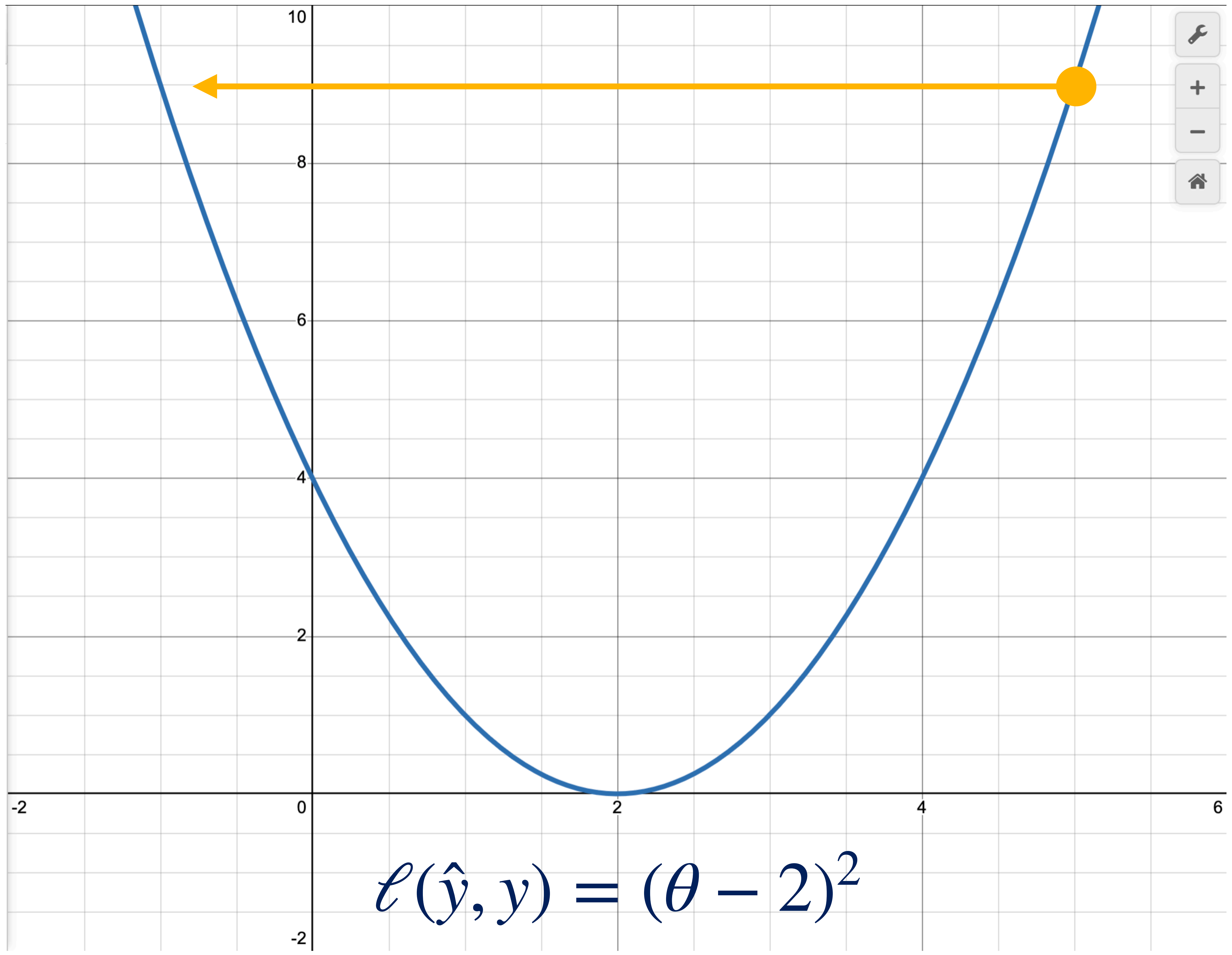


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

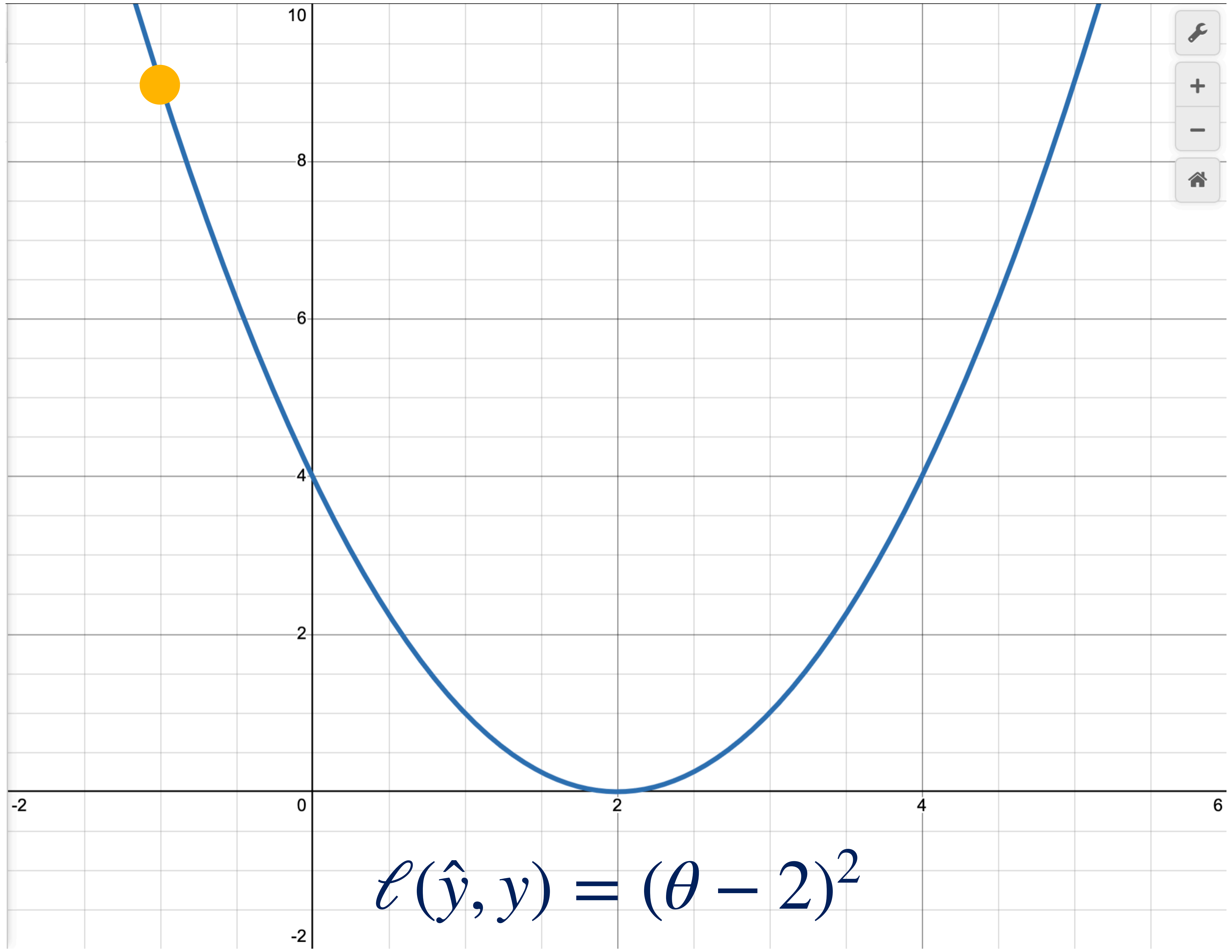
- At each step of the algorithm:
  - Calculate the **slope at the current point**
  - Adjust θ by the **negative of the slope** (because we want to **minimize** the function)
- First step:
  - Plug in θ to derivative:
    - $2 \cdot 5 - 4 = 6$
  - Update θ:
    - $\theta_1 = \theta_0 - 6 = -1$



$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

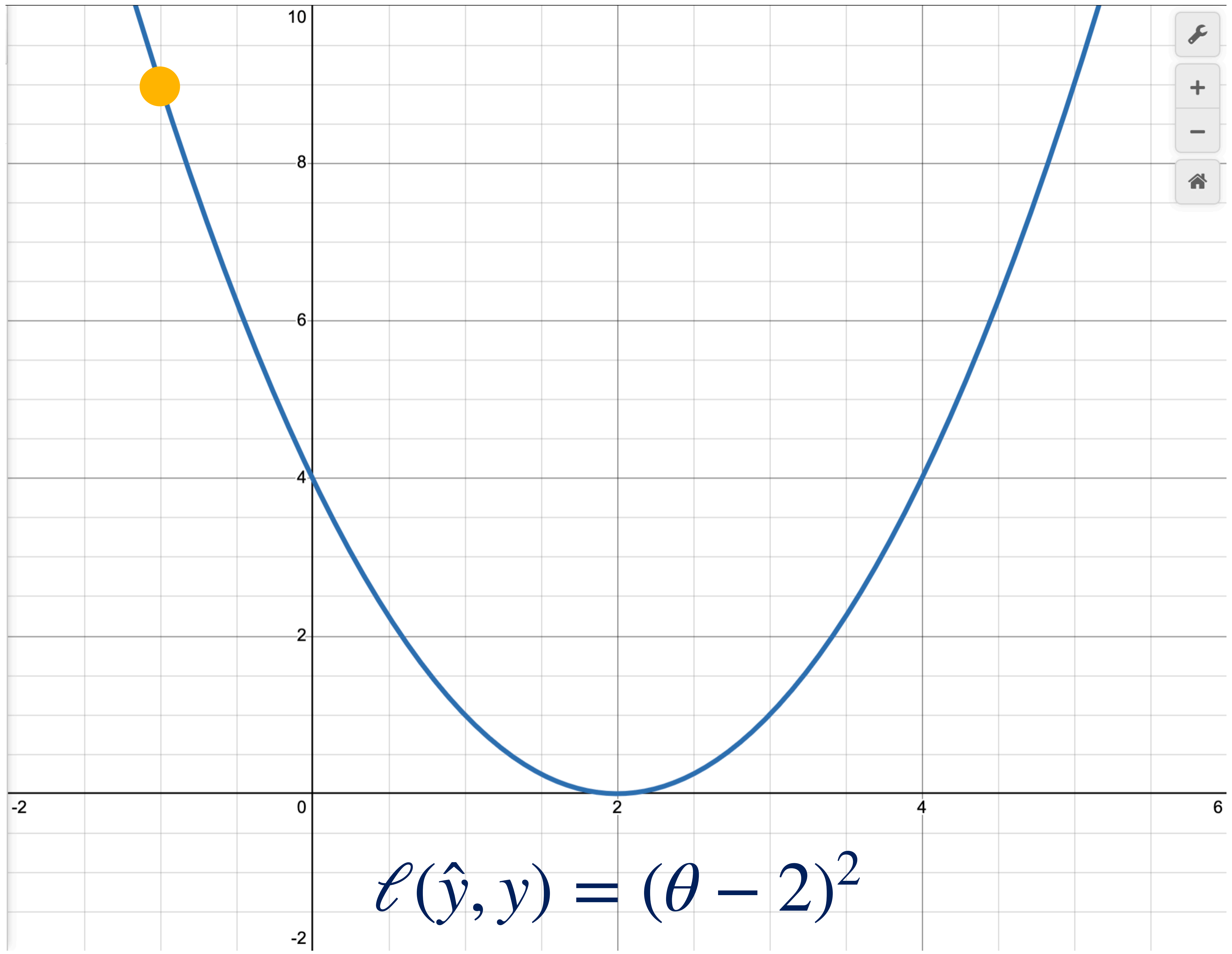


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- Second step:

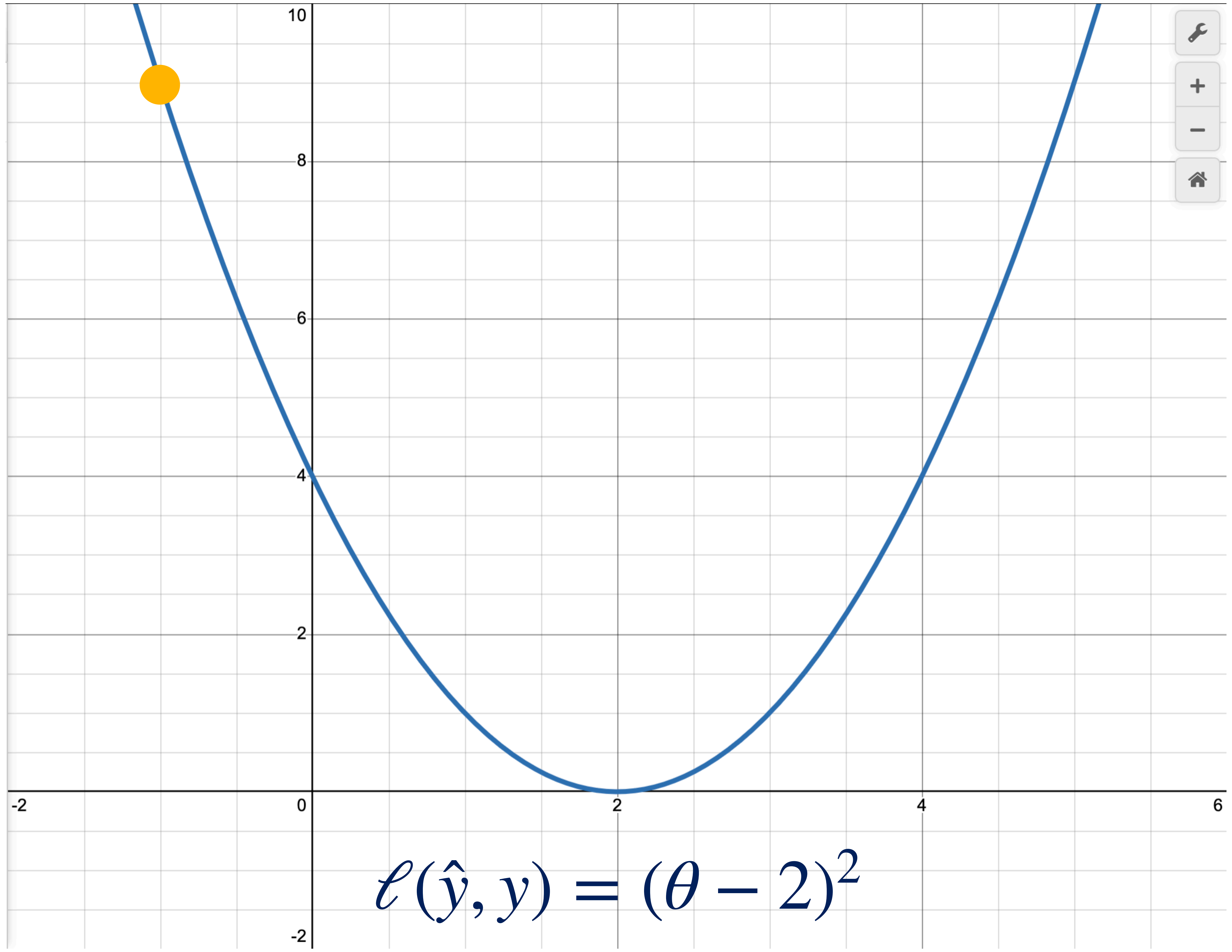


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- Second step:
  - Plug in new θ_1:

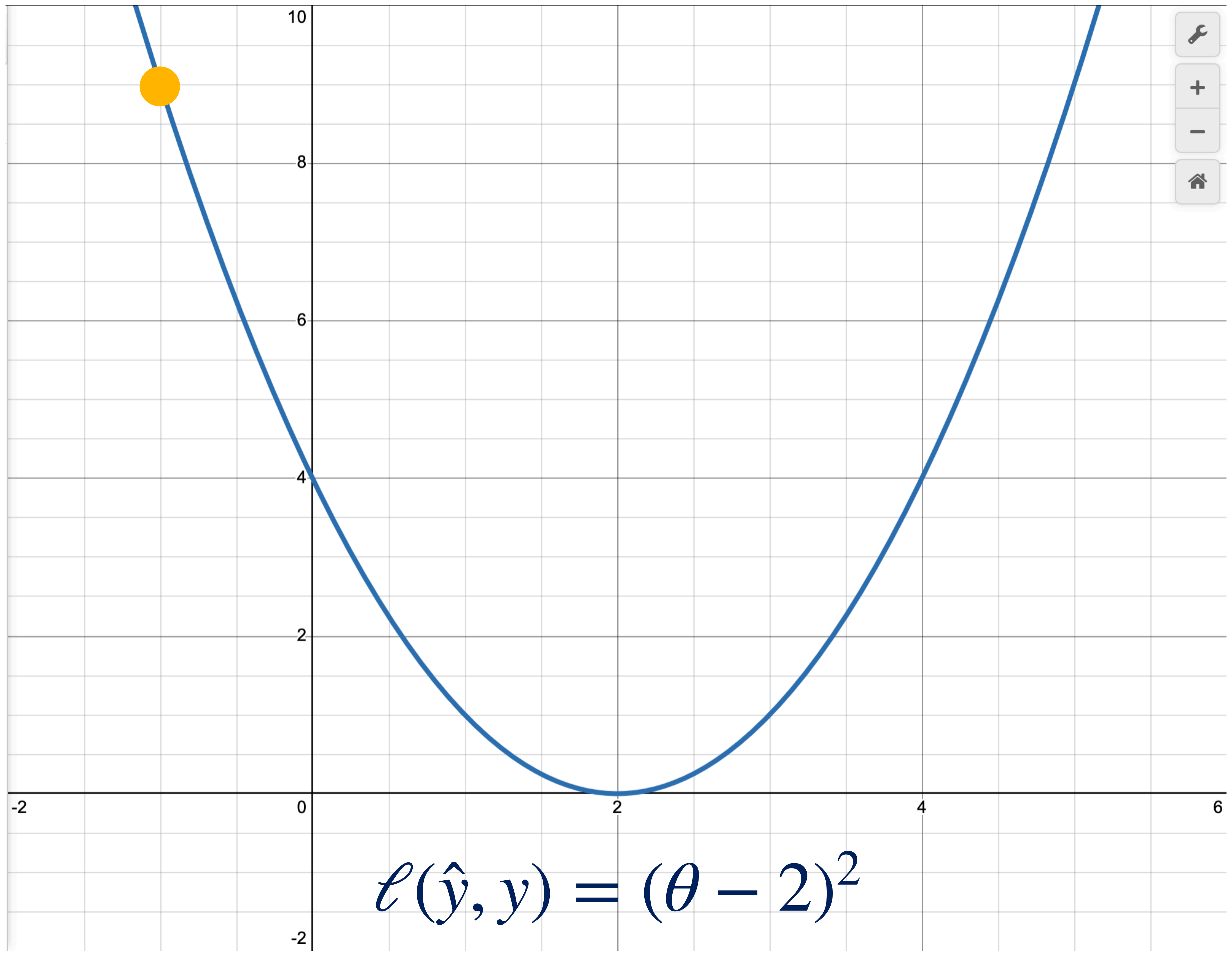


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- Second step:
  - Plug in new θ_1:
    - $2 \cdot (-1) - 4 = -6$

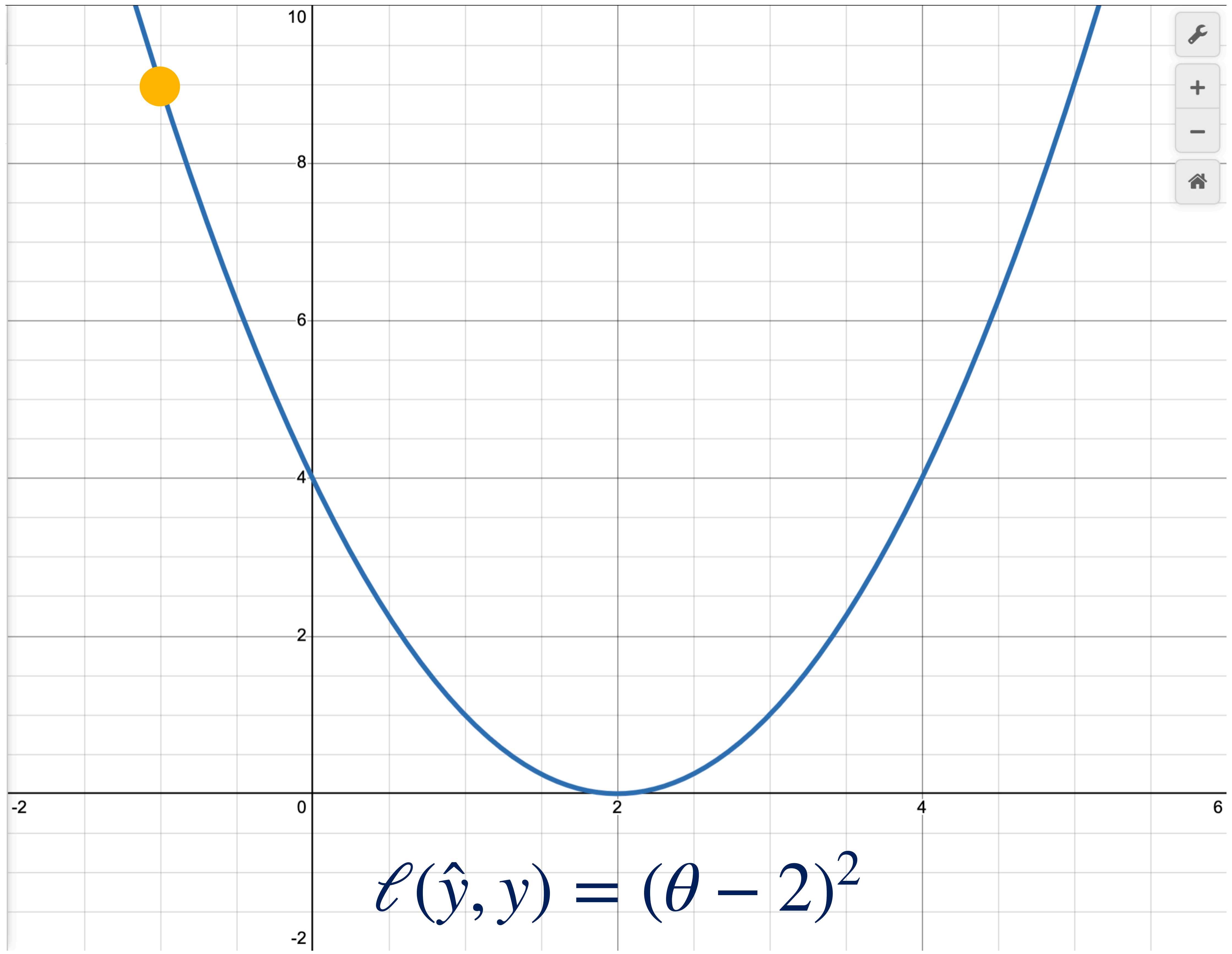


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- Second step:
  - Plug in new θ_1:
    - $2 \cdot (-1) - 4 = -6$
  - Update θ:

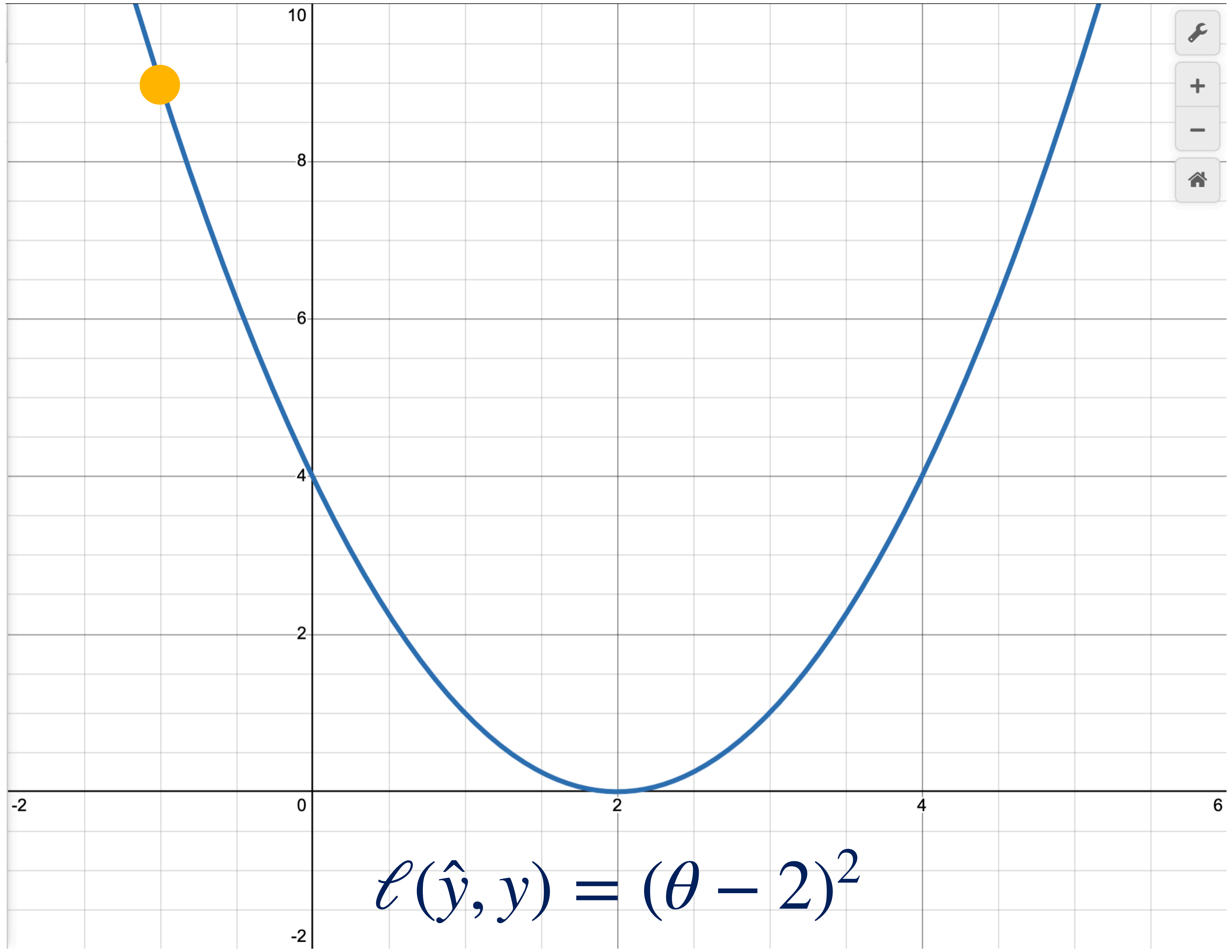


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

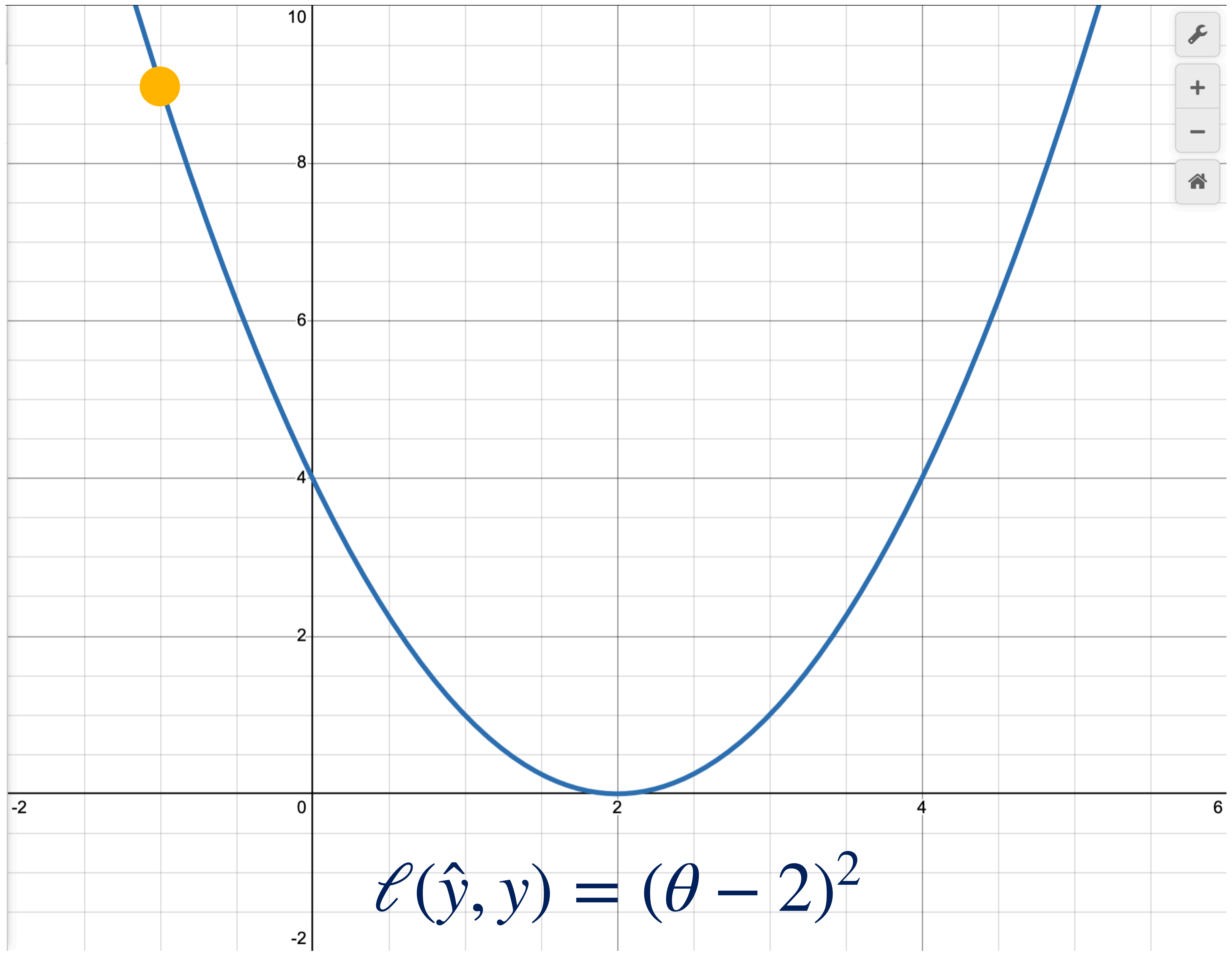
- Second step:
  - Plug in new θ_1:
    - $2 \cdot (-1) - 4 = -6$
  - Update θ:
    - $\theta_2 = \theta_1 - (-6) = 5$



$$\ell(\hat{y}, y) = (\theta - 2)^2$$
$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

- Second step:
  - Plug in new θ_1:
    - $2 \cdot (-1) - 4 = -6$
  - Update θ:
    - $\theta_2 = \theta_1 - (-6) = 5$
- Whoops! We're **back where we started!**

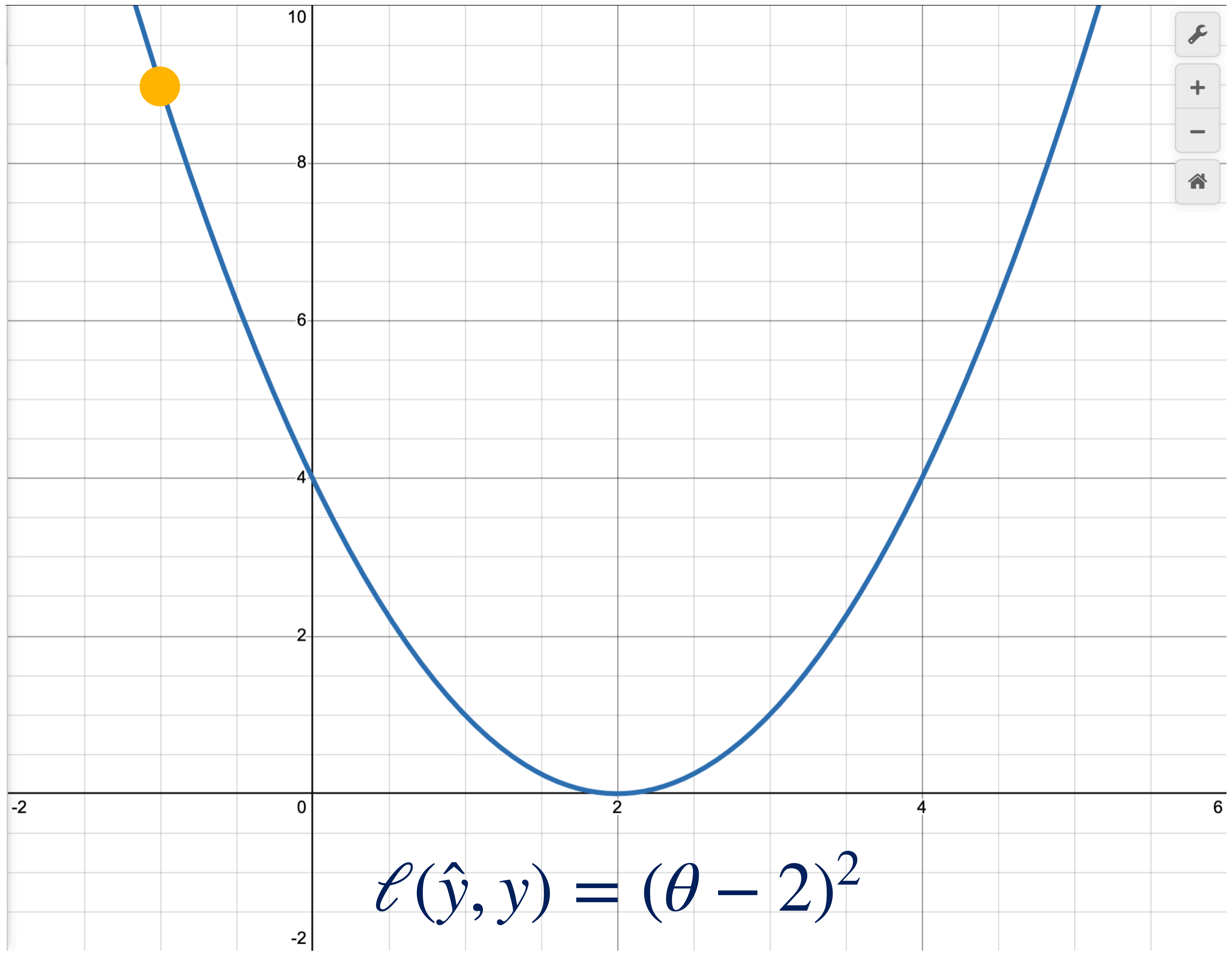


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

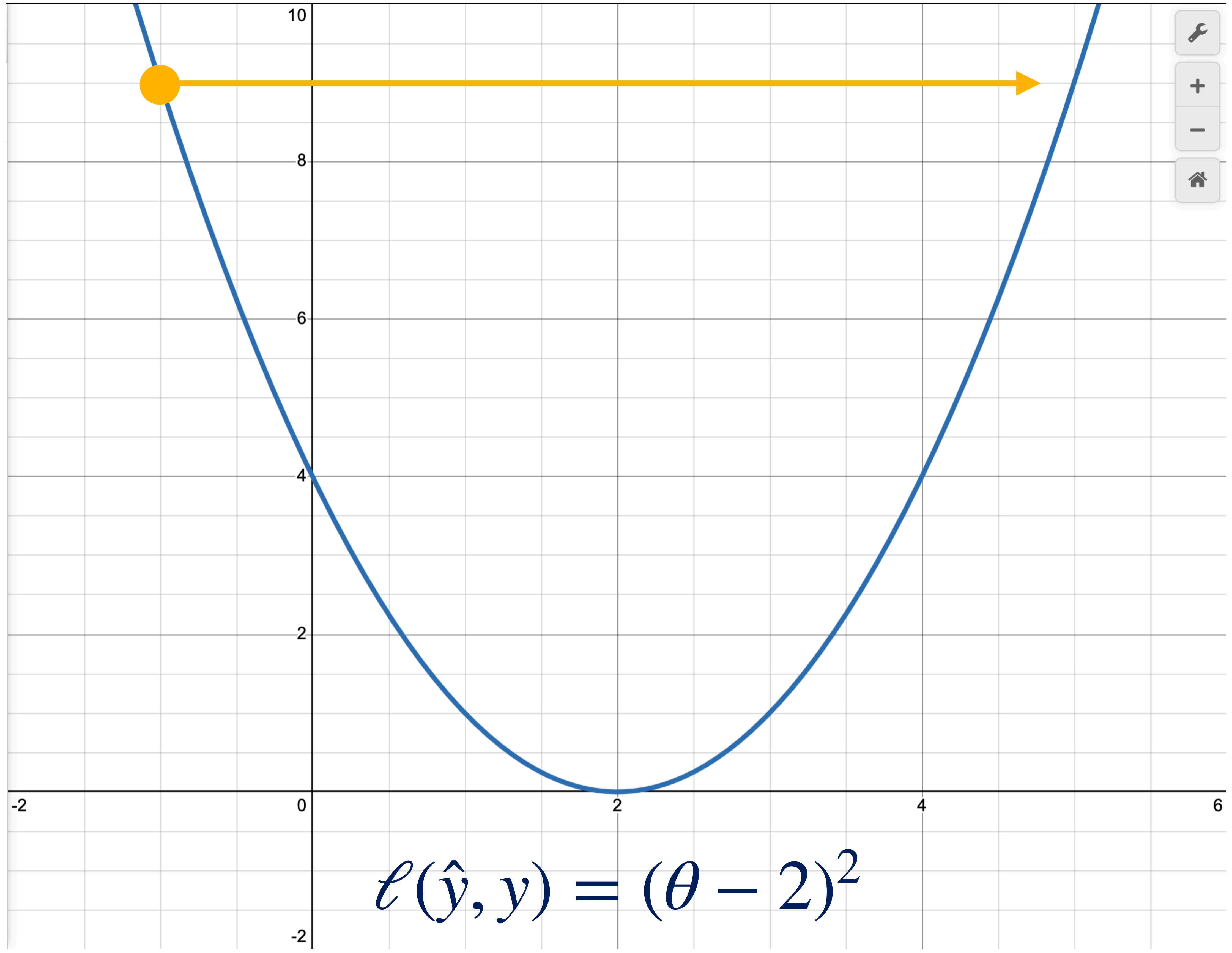
- Second step:
  - Plug in new θ_1:
    - $2 \cdot (-1) - 4 = -6$
  - Update θ:
    - $\theta_2 = \theta_1 - (-6) = 5$
- Whoops! We're **back where we started!**
  - This process would **"bounce back and forth"** forever!



$$\ell(\hat{y}, y) = (\theta - 2)^2$$
$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (first try)

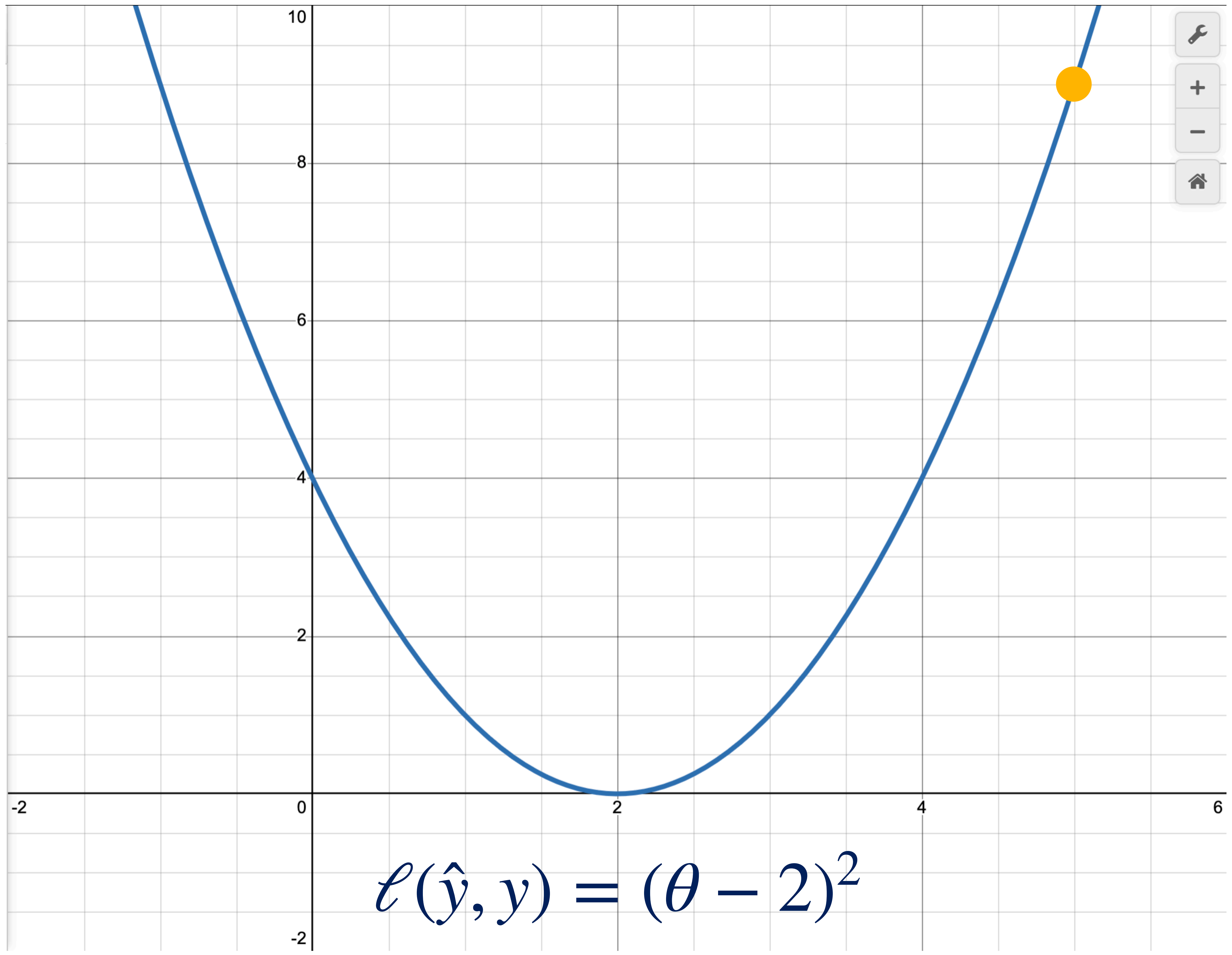
- Second step:
  - Plug in new θ_1:
    - $2 \cdot (-1) - 4 = -6$
  - Update θ:
    - $\theta_2 = \theta_1 - (-6) = 5$
- Whoops! We're **back where we started!**
  - This process would **"bounce back and forth"** forever!



$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

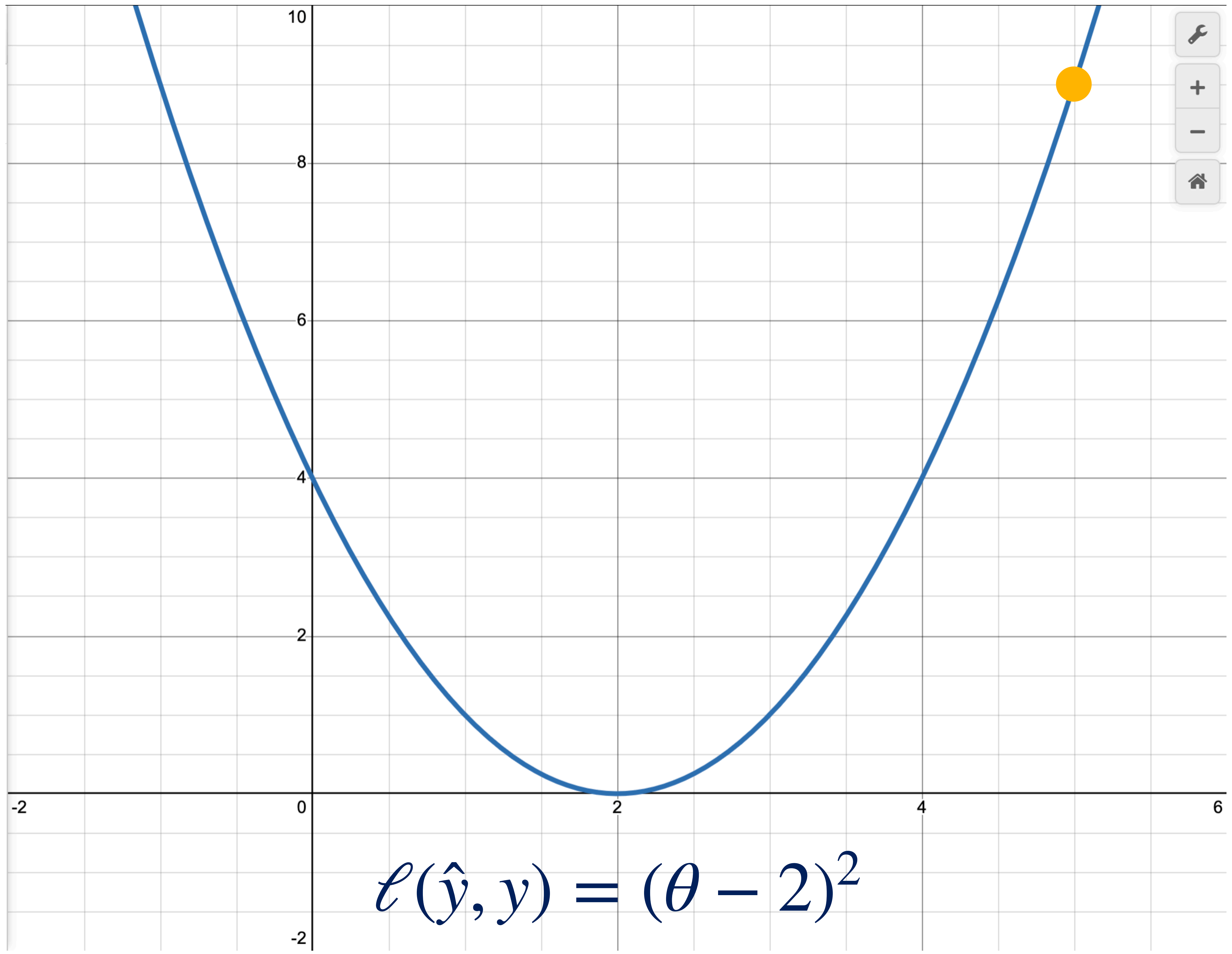


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- What if we adjusted θ by a **fraction** of the slope? (Say, 0.75)

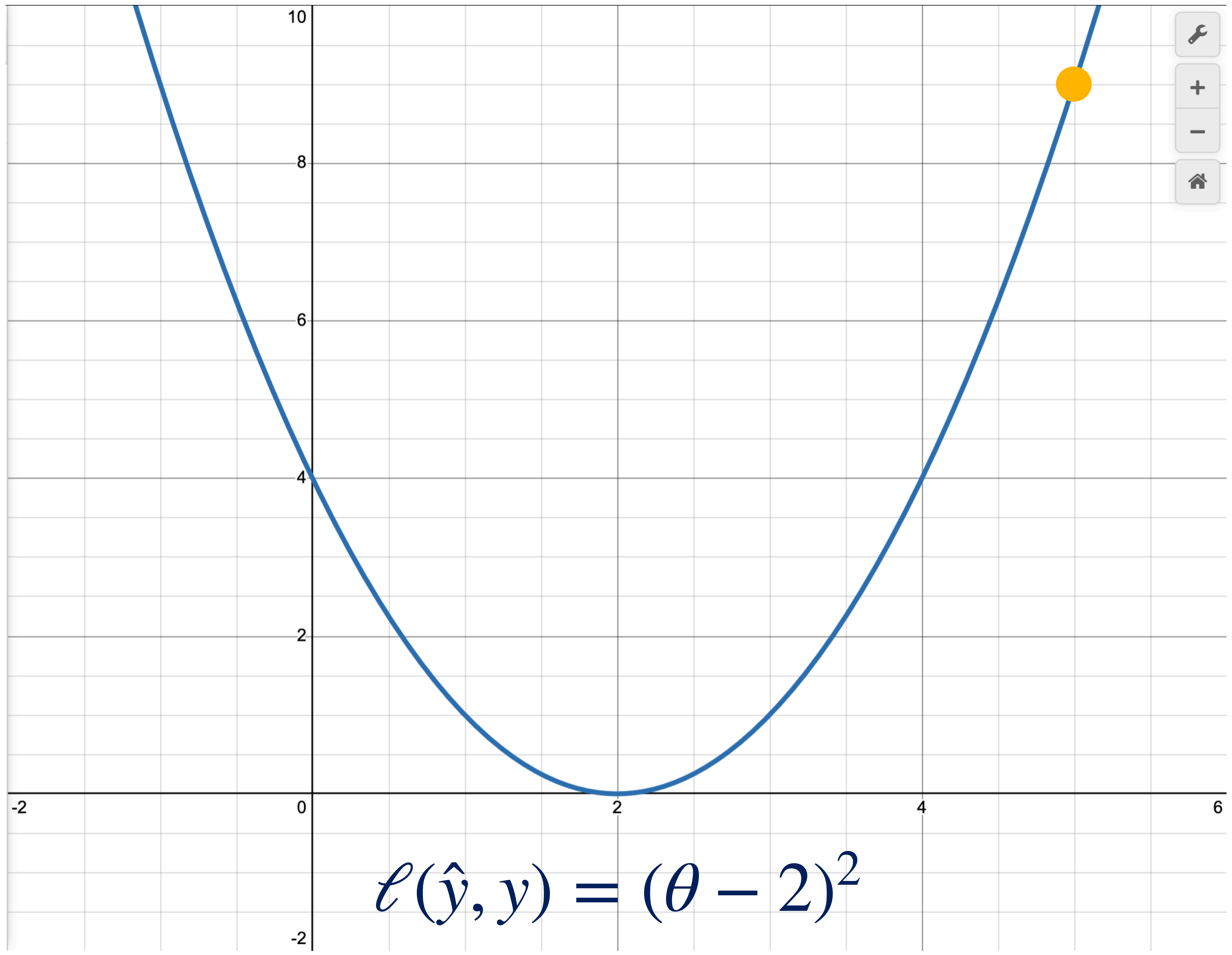


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- What if we adjusted θ by a **fraction** of the slope? (Say, 0.75)
- Let's start again:

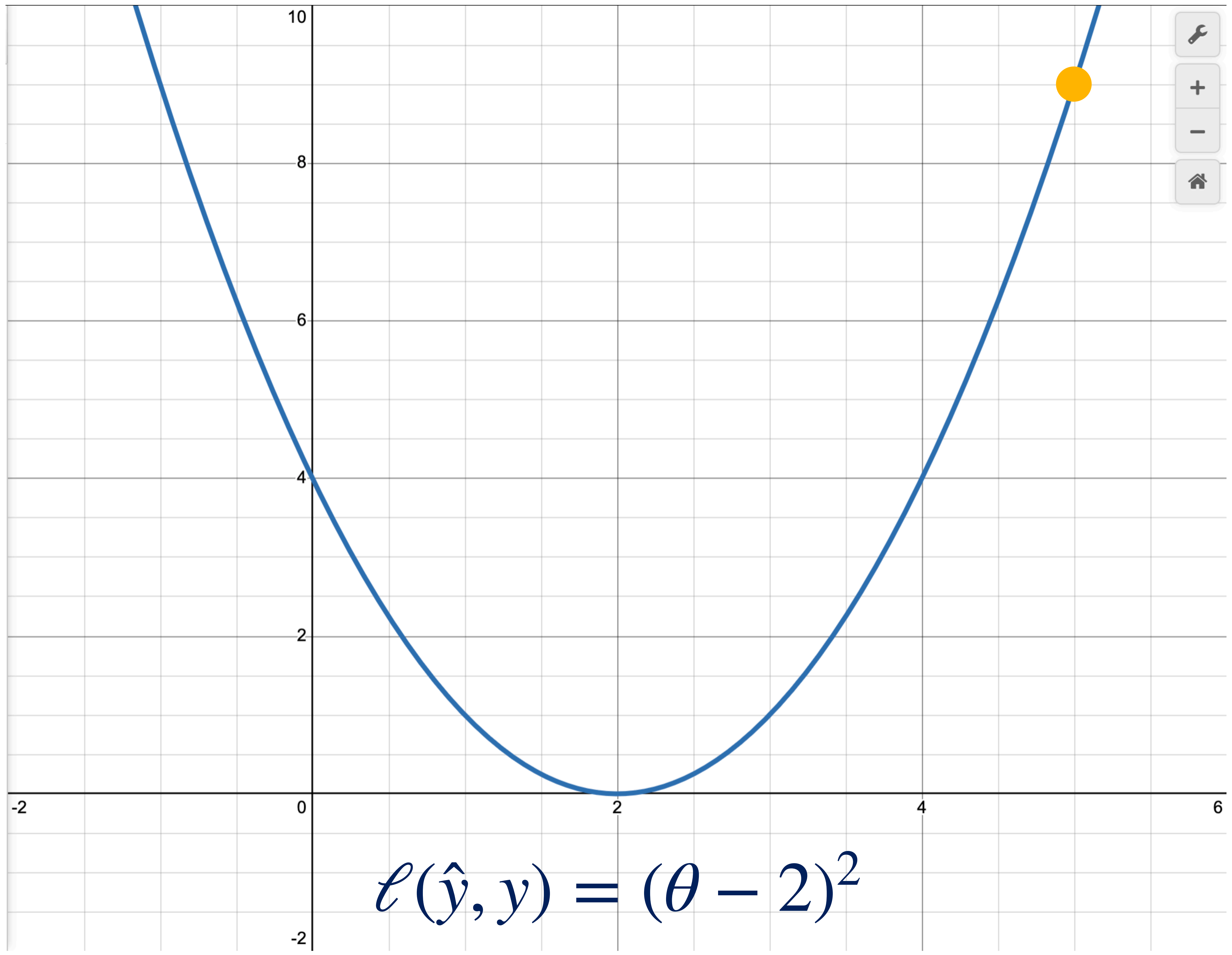


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- What if we adjusted θ by a **fraction** of the slope? (Say, 0.75)
- Let's start again:
    - Plug in θ_0 to derivative:

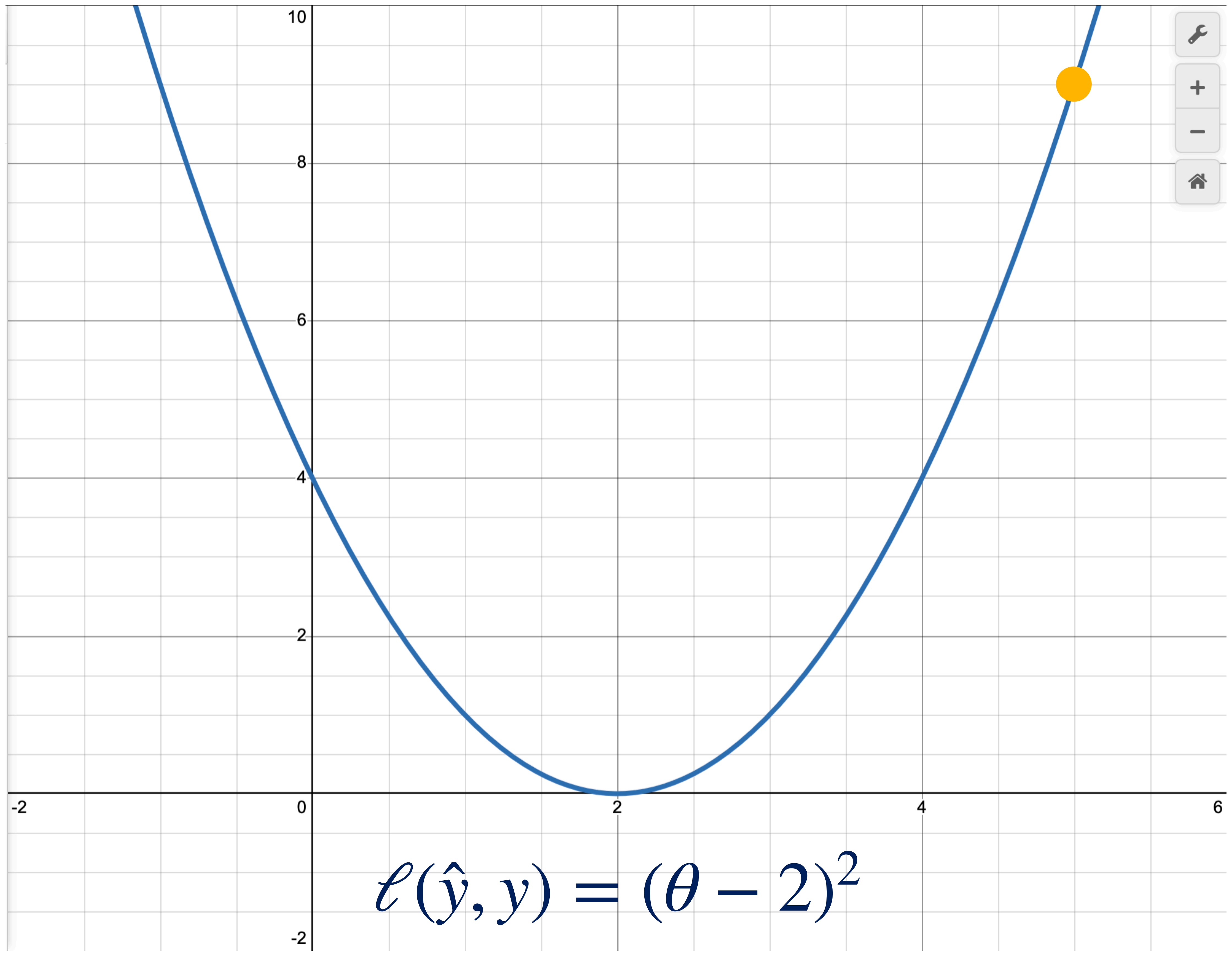


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- What if we adjusted θ by a **fraction** of the slope? (Say, 0.75)
- Let's start again:
  - Plug in θ_0 to derivative:
    - $2 \cdot 5 - 4 = 6$

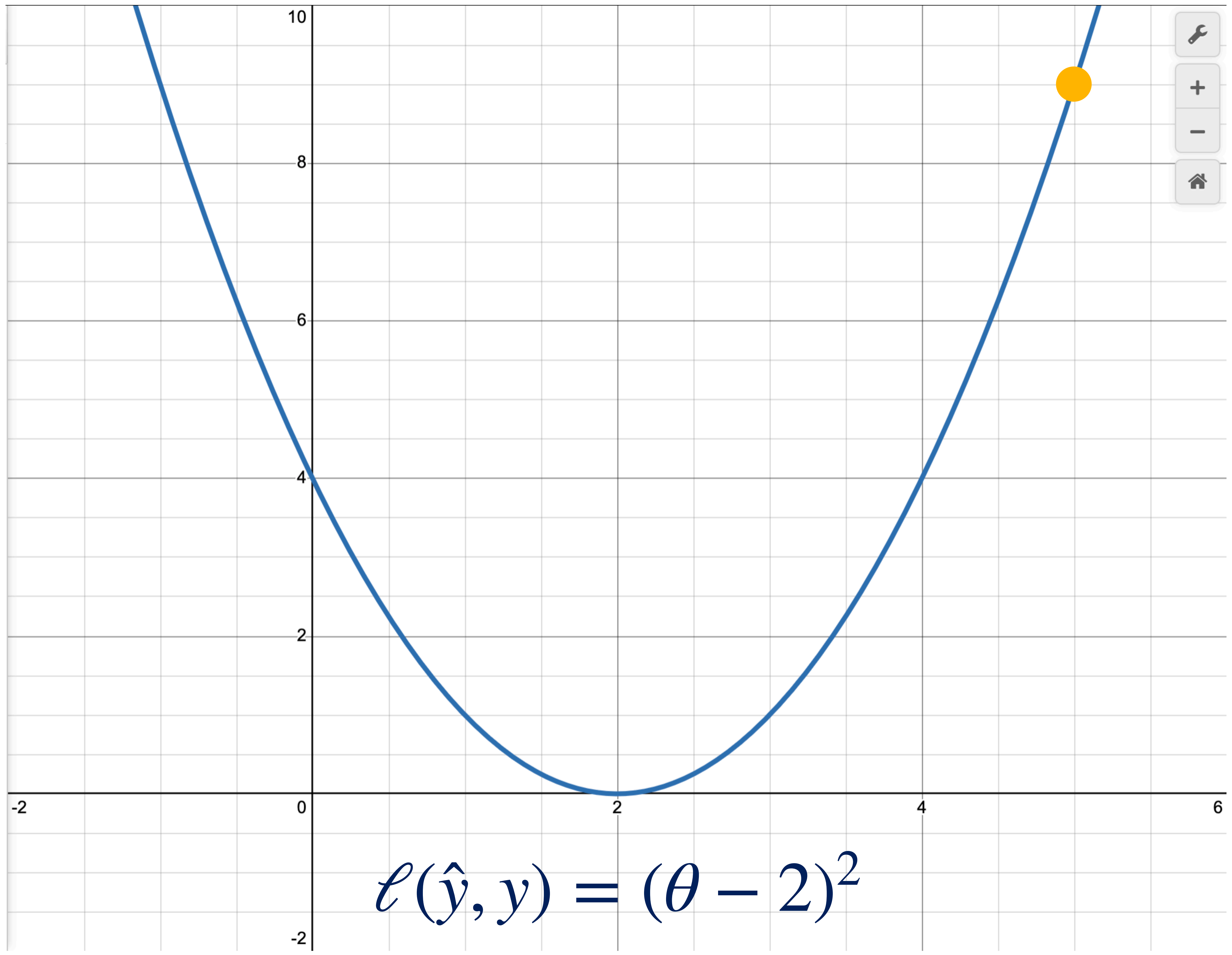


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- What if we adjusted θ by a **fraction** of the slope? (Say, 0.75)
- Let's start again:
  - Plug in θ_0 to derivative:
    - $2 \cdot 5 - 4 = 6$
  - Update θ:

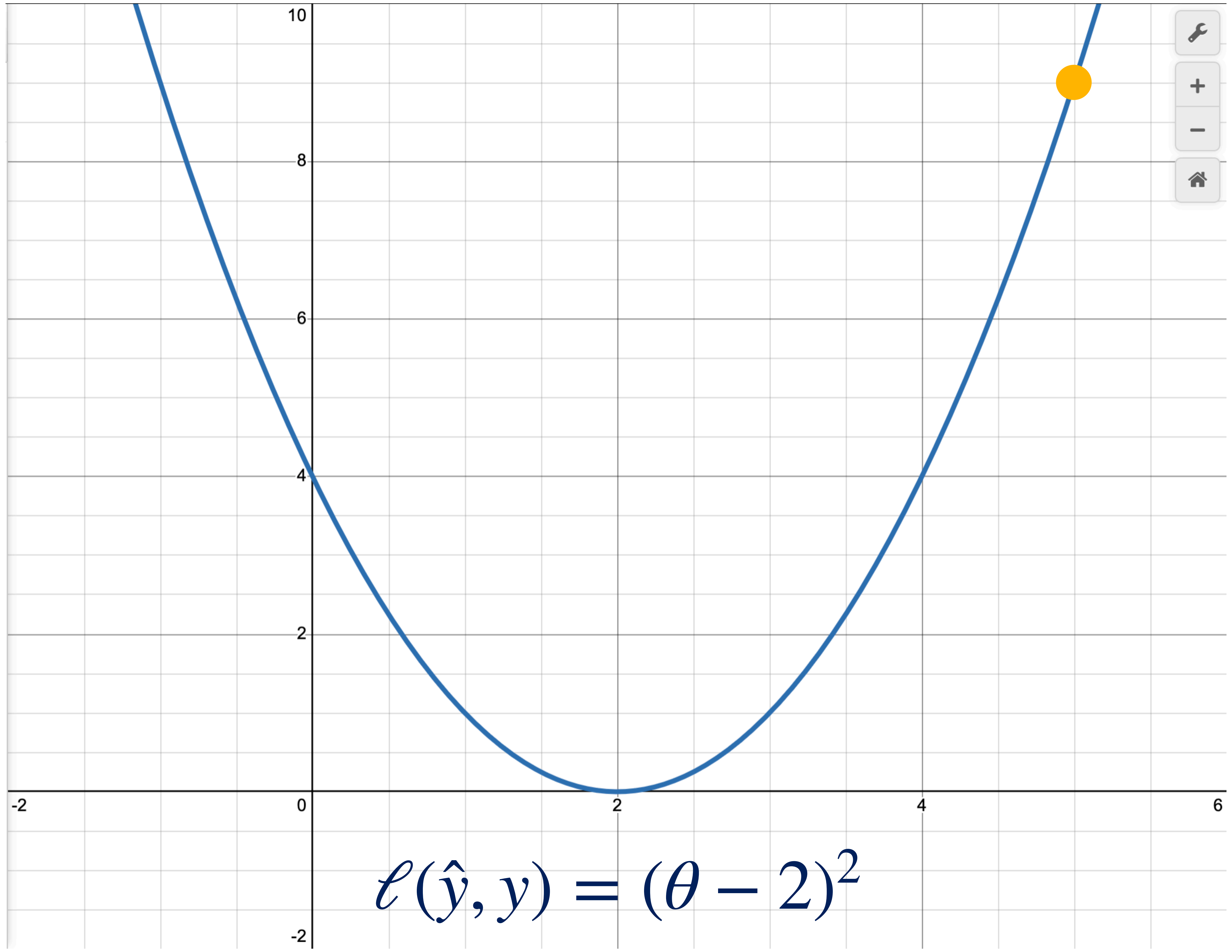


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

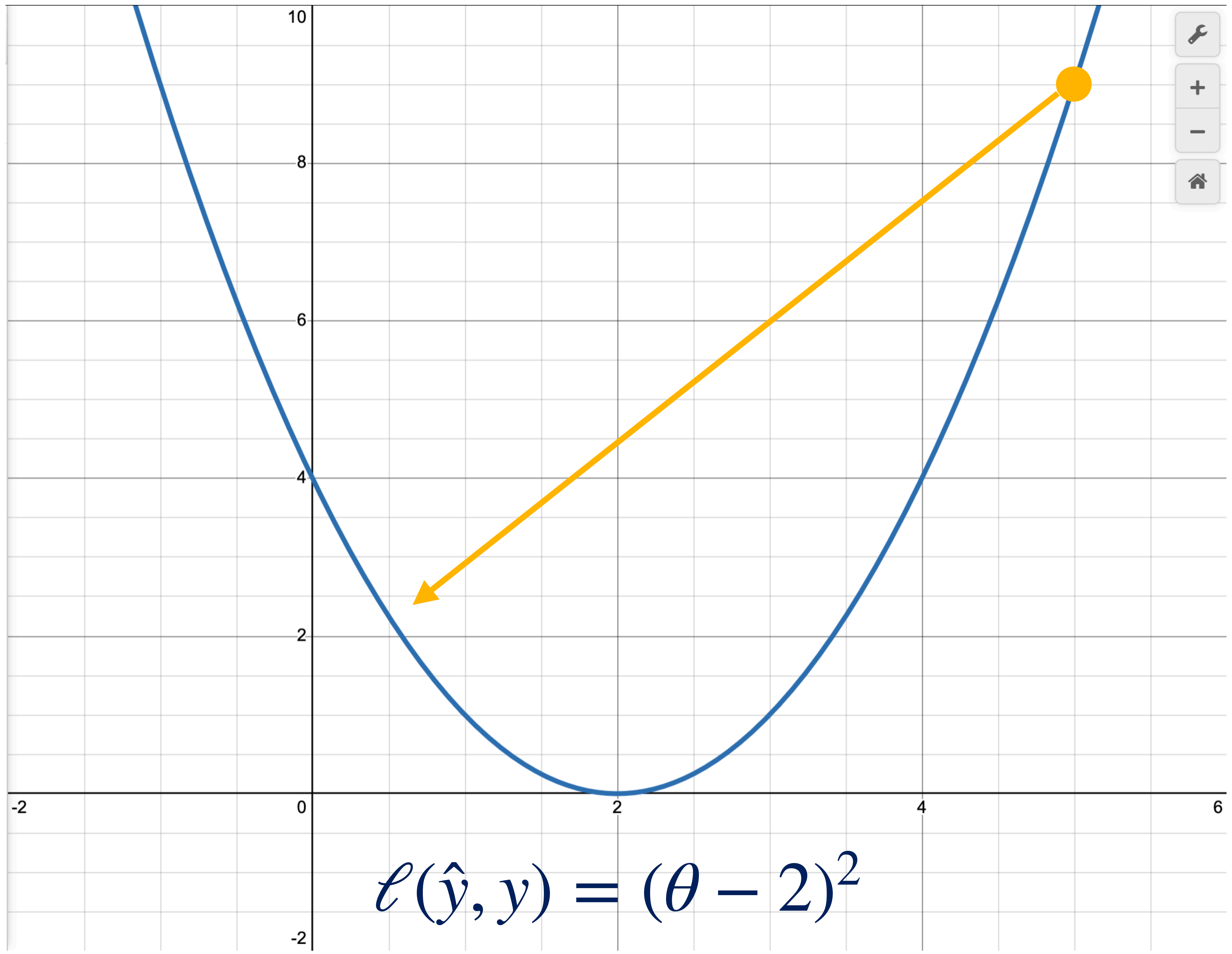
- What if we adjusted θ by a **fraction** of the slope? (Say, 0.75)
- Let's start again:
  - Plug in θ_0 to derivative:
    - $2 \cdot 5 - 4 = 6$
  - Update θ:
    - $\theta_1 = \theta_0 - 0.75 \cdot 6 = 0.5$



$$\ell(\hat{y}, y) = (\theta - 2)^2$$
$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

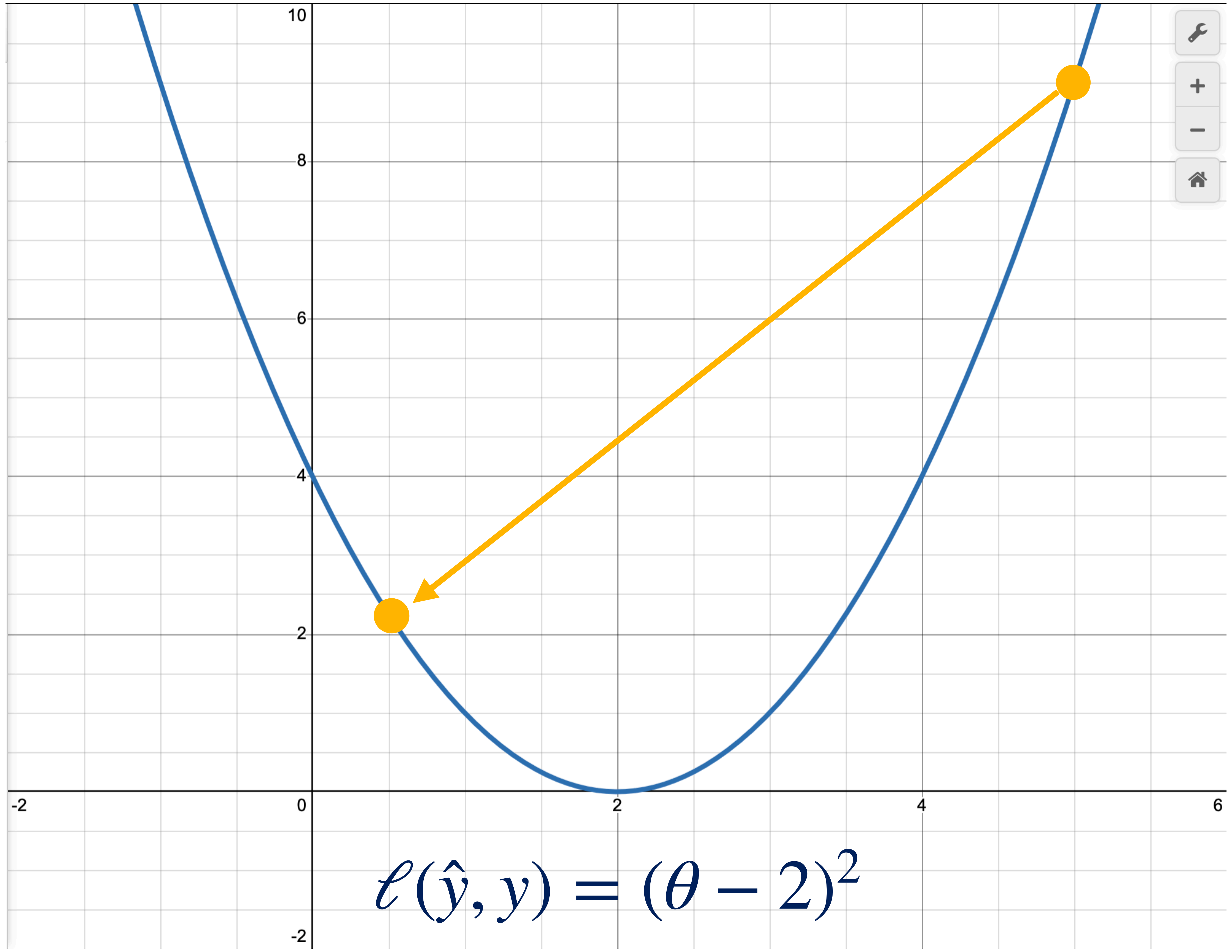
- What if we adjusted θ by a **fraction** of the slope? (Say, 0.75)
- Let's start again:
  - Plug in θ_0 to derivative:
    - $2 \cdot 5 - 4 = 6$
  - Update θ:
    - $\theta_1 = \theta_0 - 0.75 \cdot 6 = 0.5$



$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

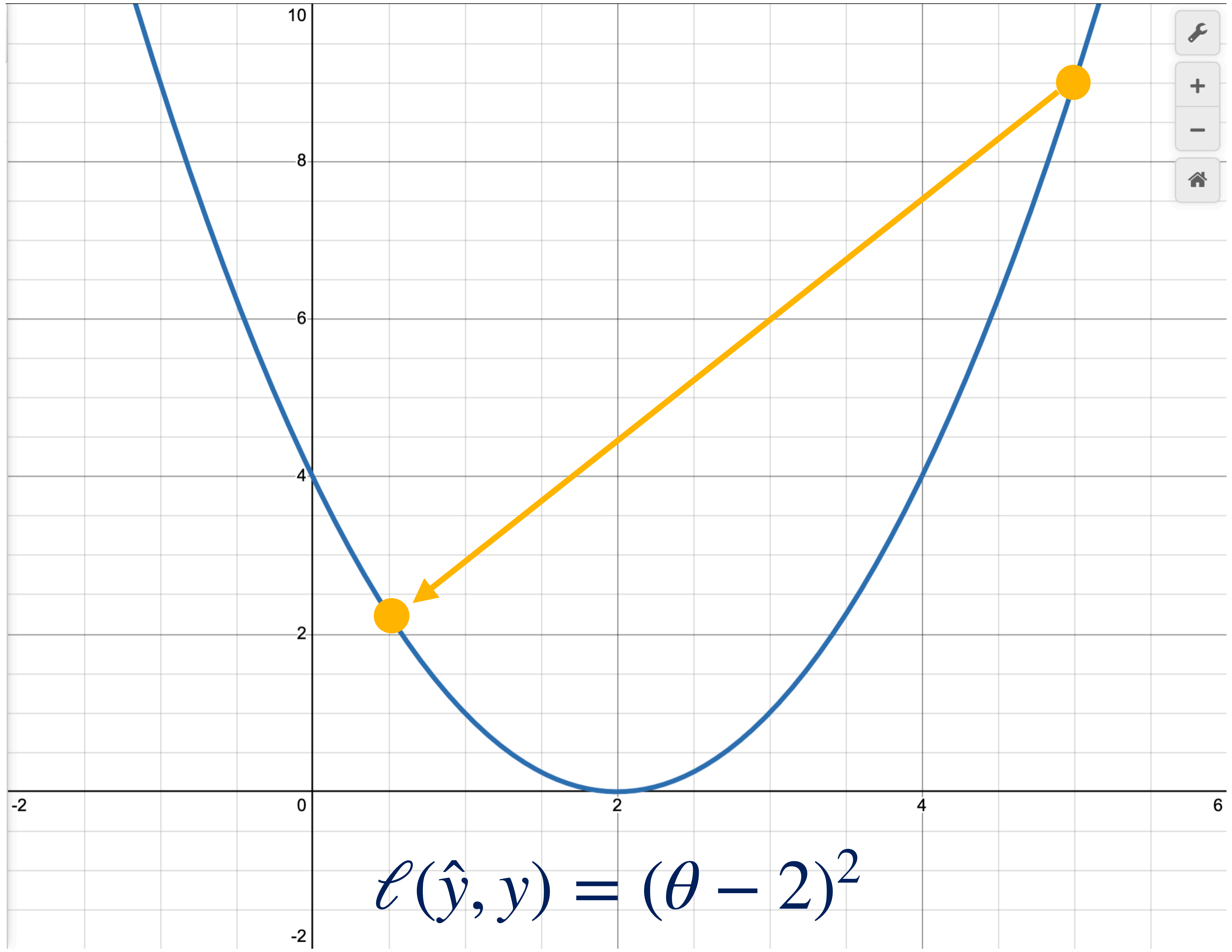


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- Second step:

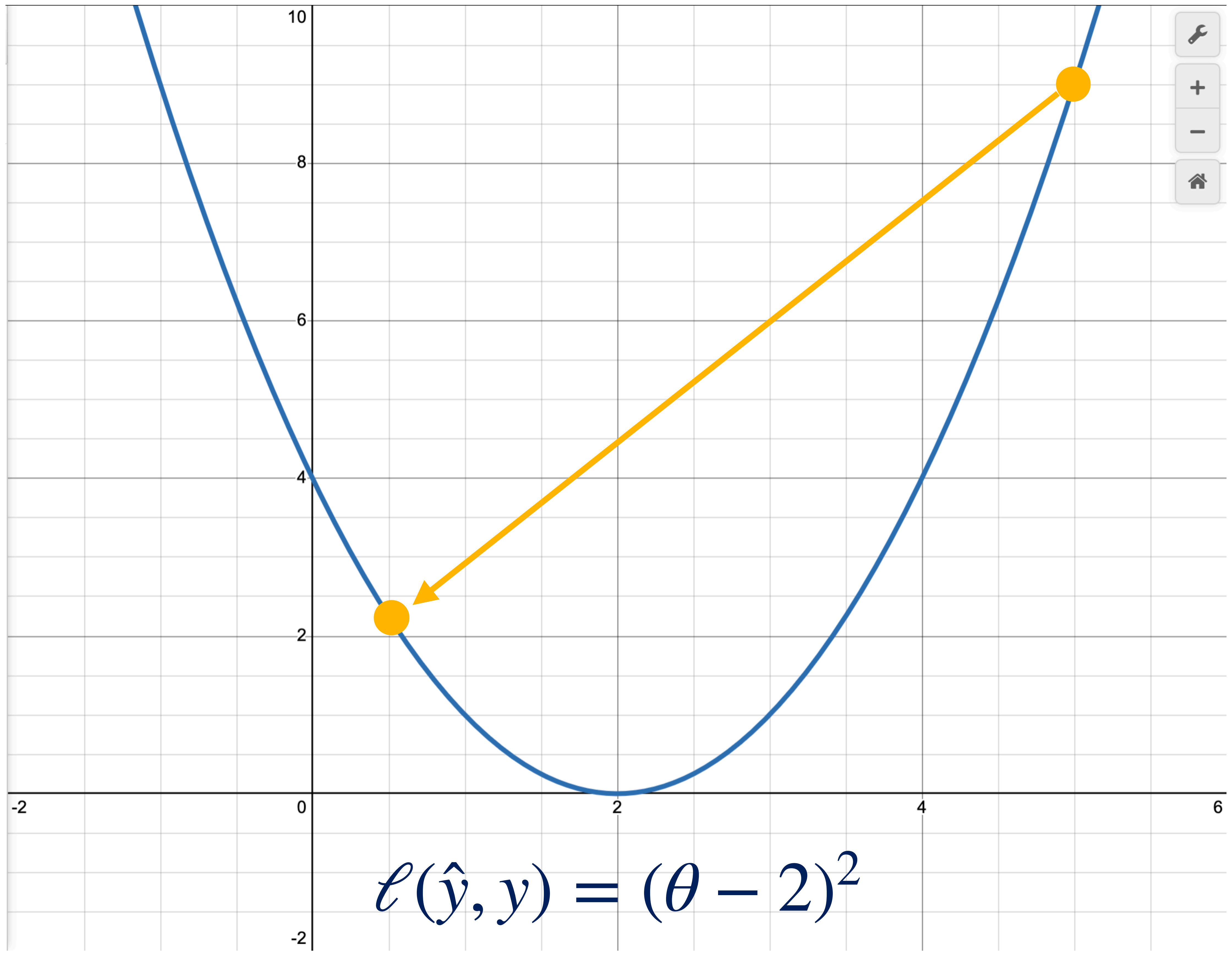


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- Second step:
  - Plug in θ_1 to derivative:

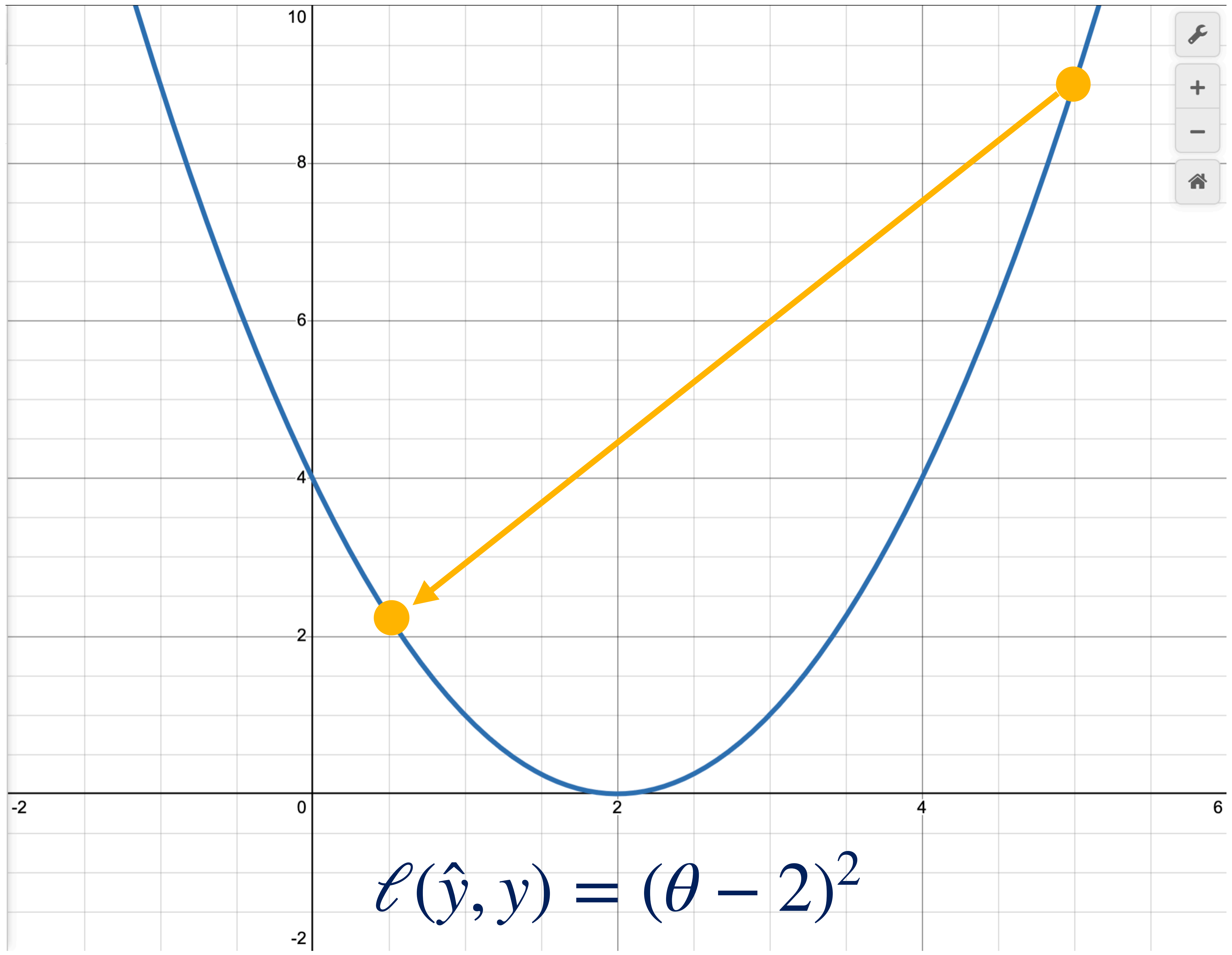


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- Second step:
  - Plug in θ_1 to derivative:
    - $2 \cdot 0.5 - 4 = -3$

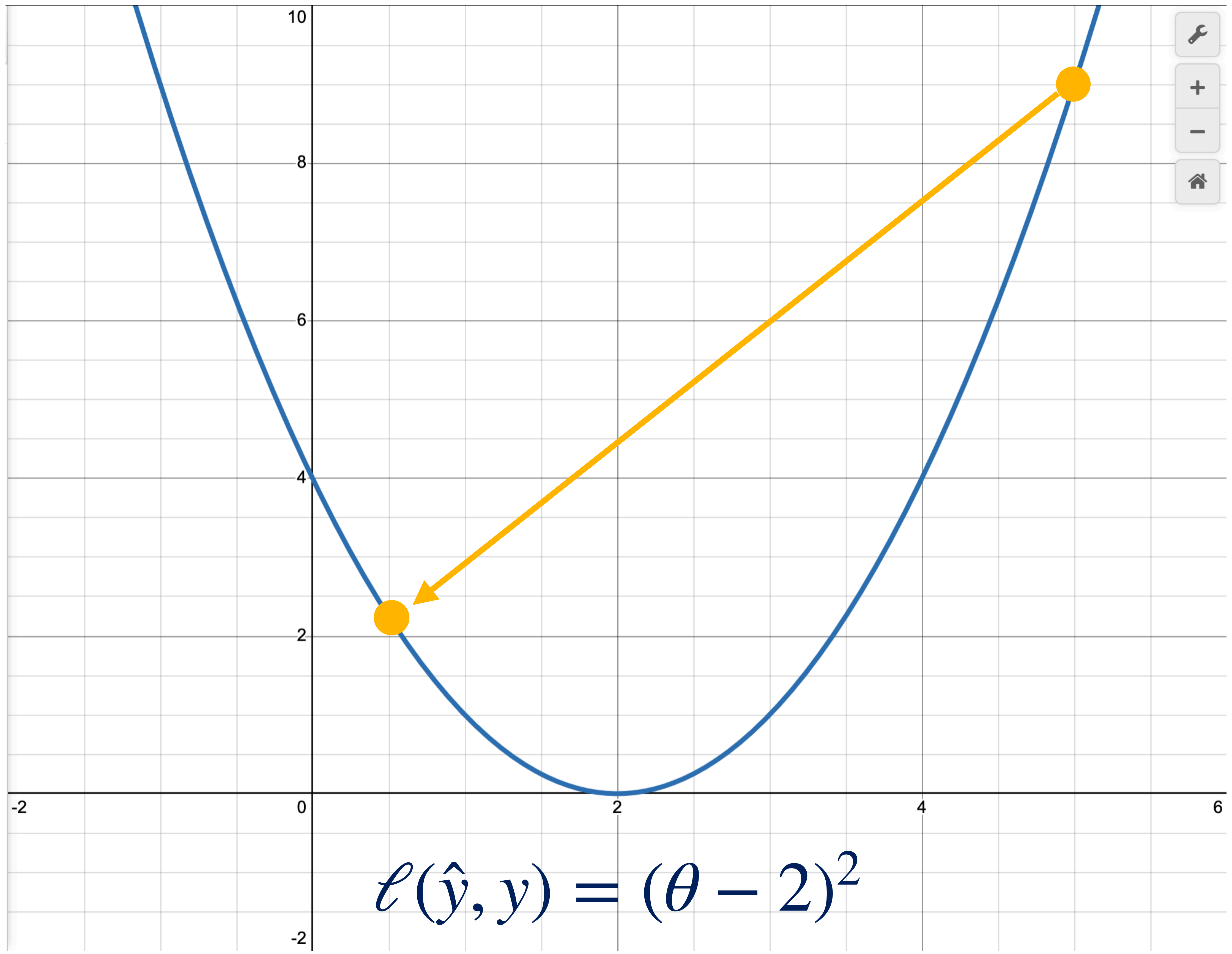


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- Second step:
  - Plug in θ_1 to derivative:
    - $2 \cdot 0.5 - 4 = -3$
  - Update θ:

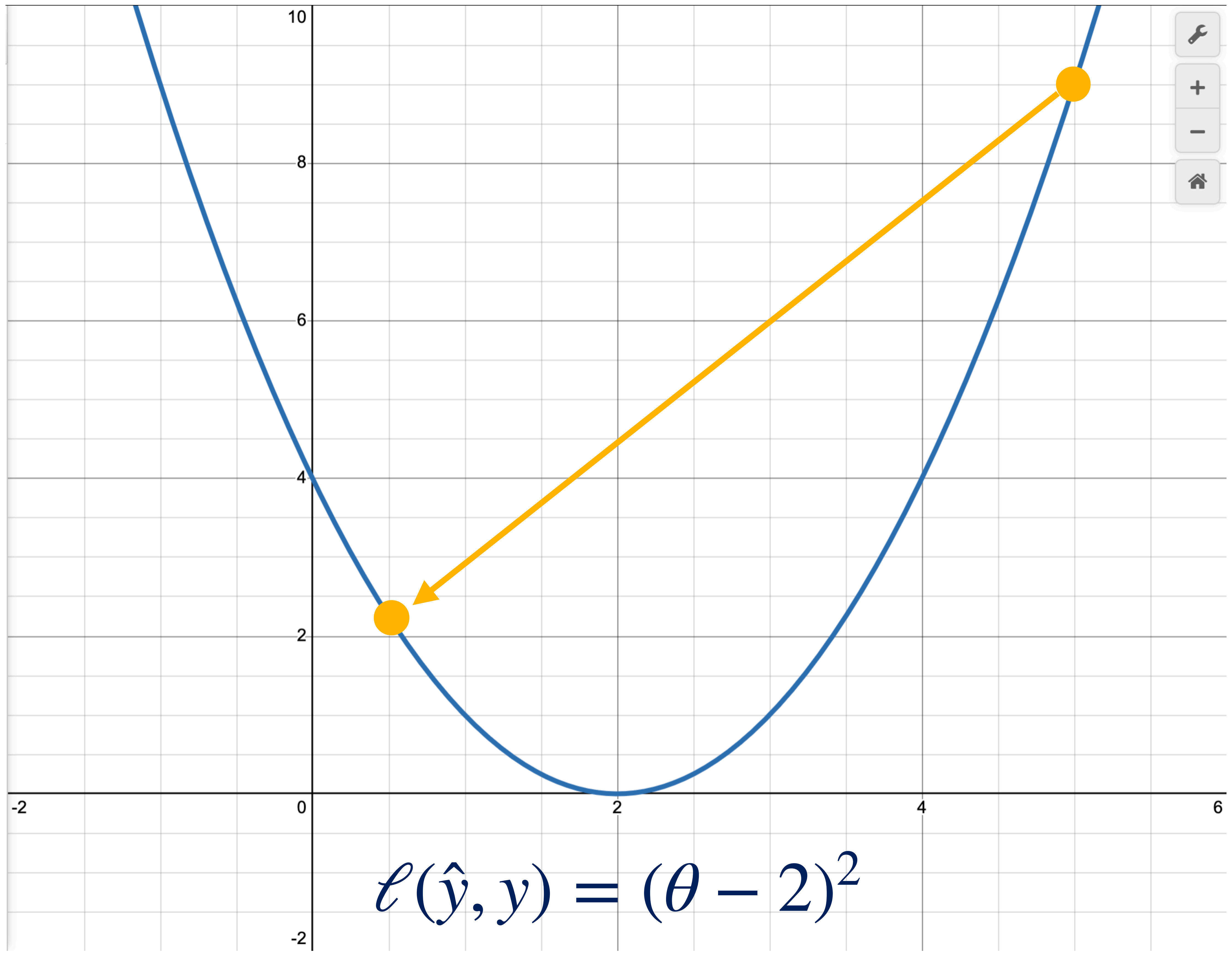


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

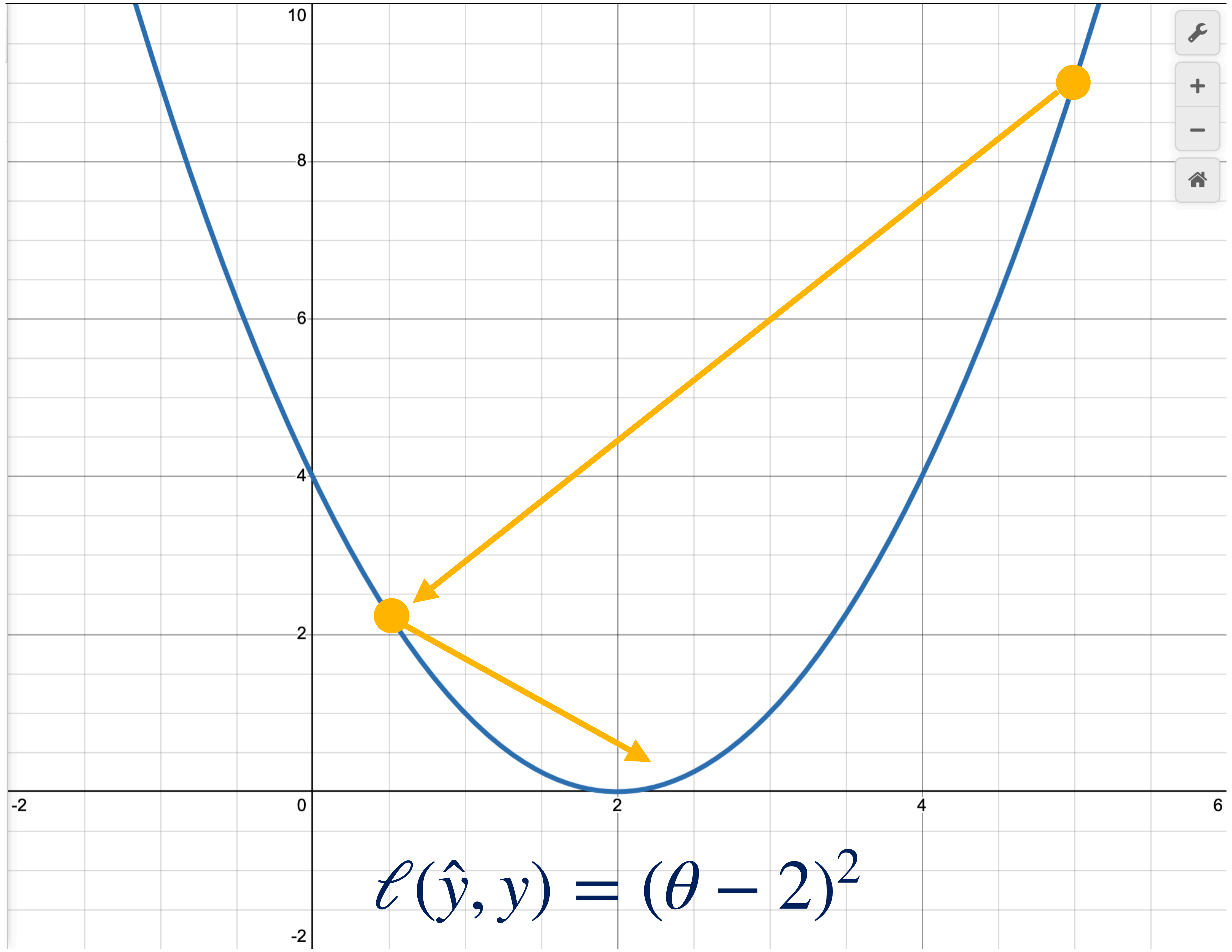
- Second step:
  - Plug in θ_1 to derivative:
    - $2 \cdot 0.5 - 4 = -3$
  - Update θ:
    - $\theta_2 = \theta_1 - 0.75 \cdot (-3) = 2.25$



$$\ell(\hat{y}, y) = (\theta - 2)^2$$
$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- Second step:
  - Plug in θ_1 to derivative:
    - $2 \cdot 0.5 - 4 = -3$
  - Update θ:
    - $\theta_2 = \theta_1 - 0.75 \cdot (-3) = 2.25$

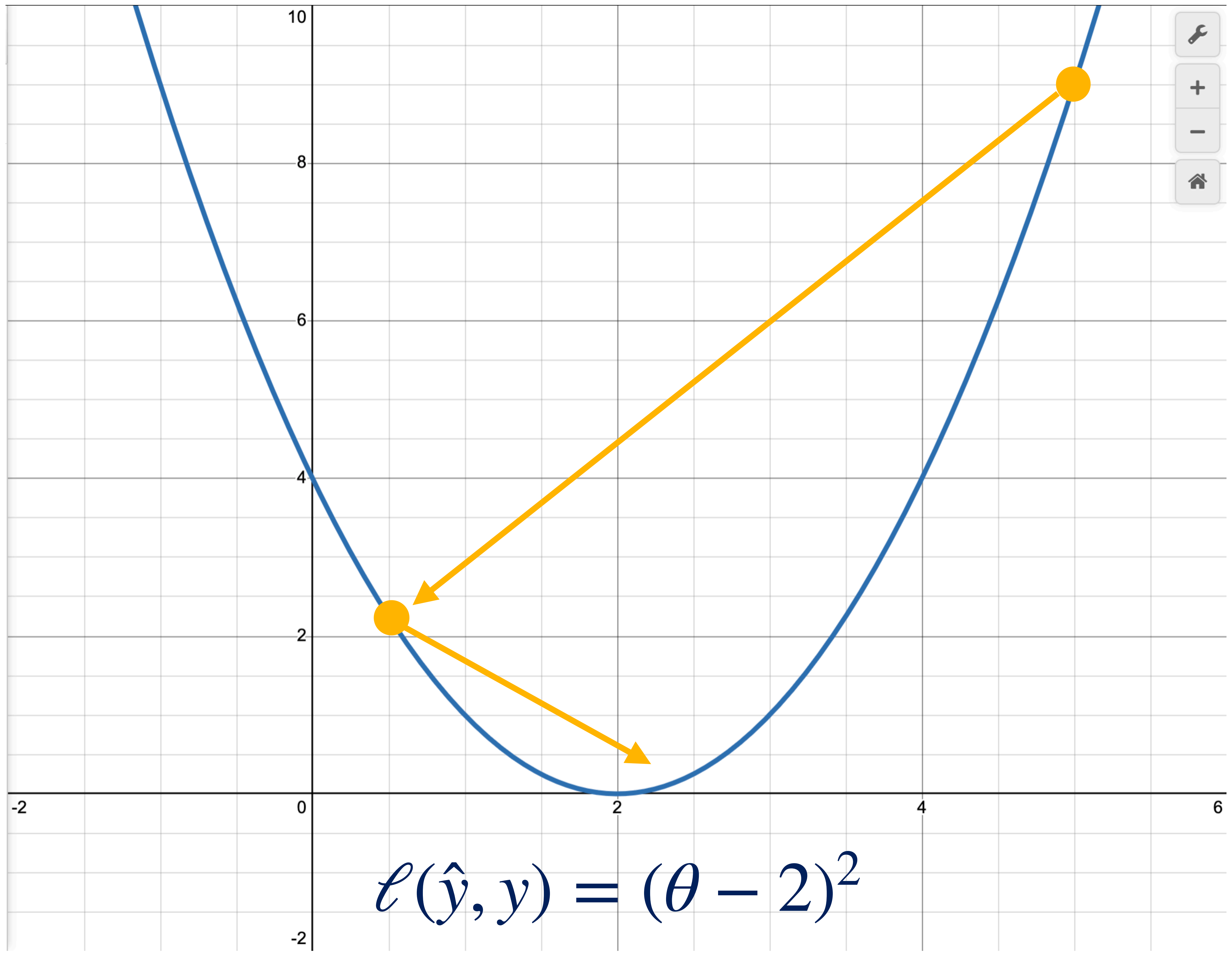


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

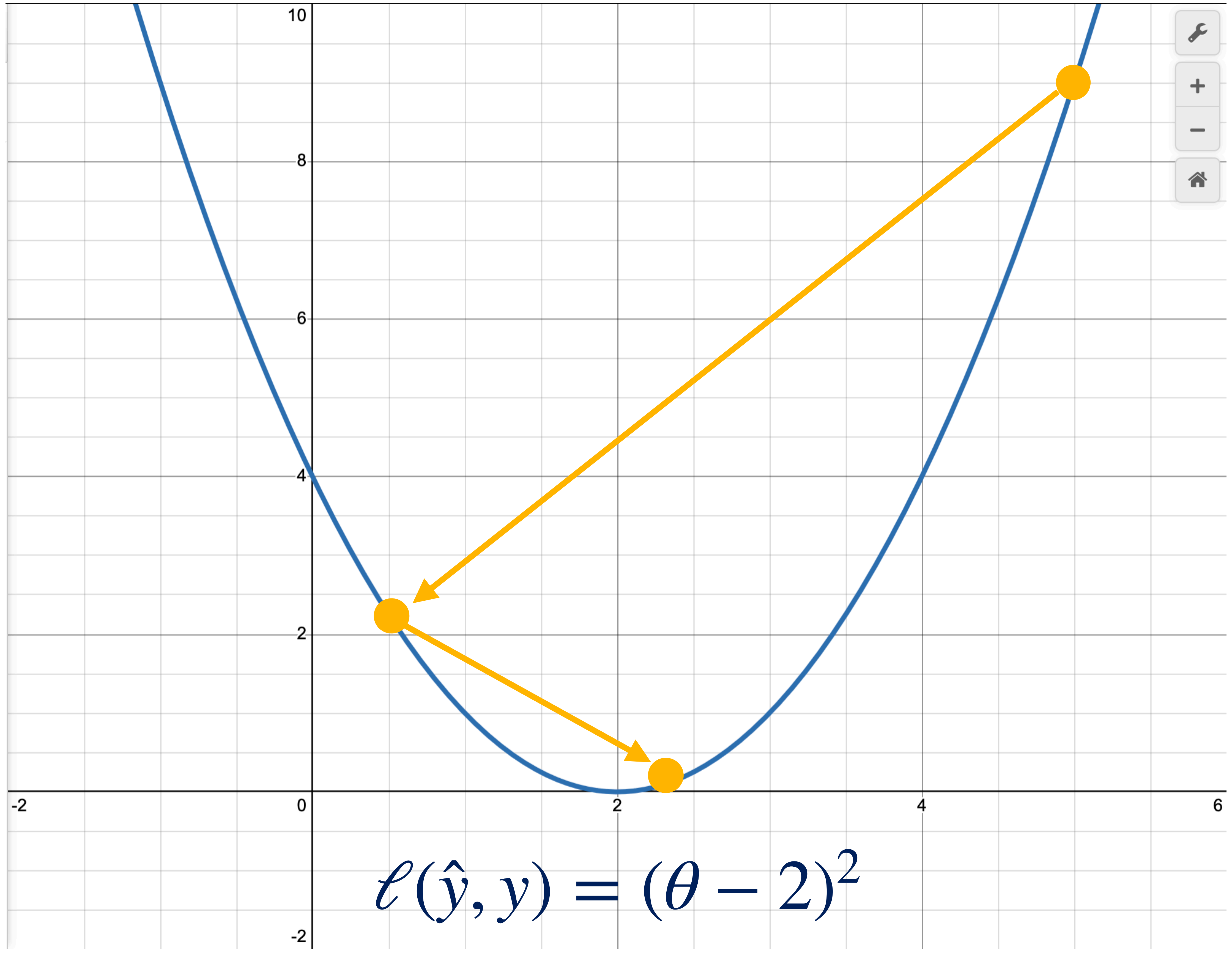
- Second step:
  - Plug in θ_1 to derivative:
    - $2 \cdot 0.5 - 4 = -3$
  - Update θ:
    - $\theta_2 = \theta_1 - 0.75 \cdot (-3) = 2.25$
- (This is looking better)



$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

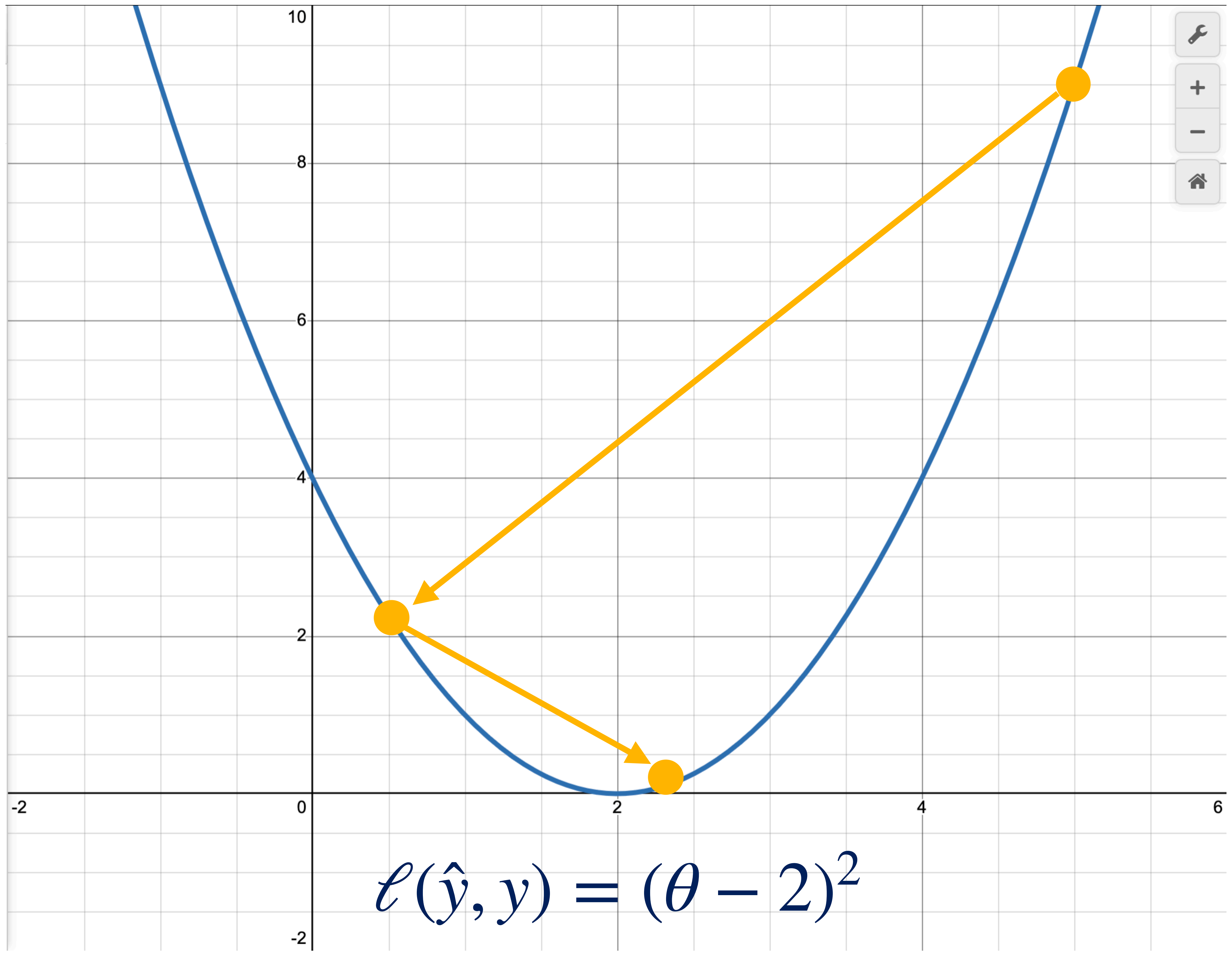


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- Third step:
  - Plug in θ_2 to derivative:
    - $2 \cdot 2.25 - 4 = 0.5$
  - Update θ:
    - $\theta_3 = \theta_2 - 0.75 \cdot 0.5 = 1.875$

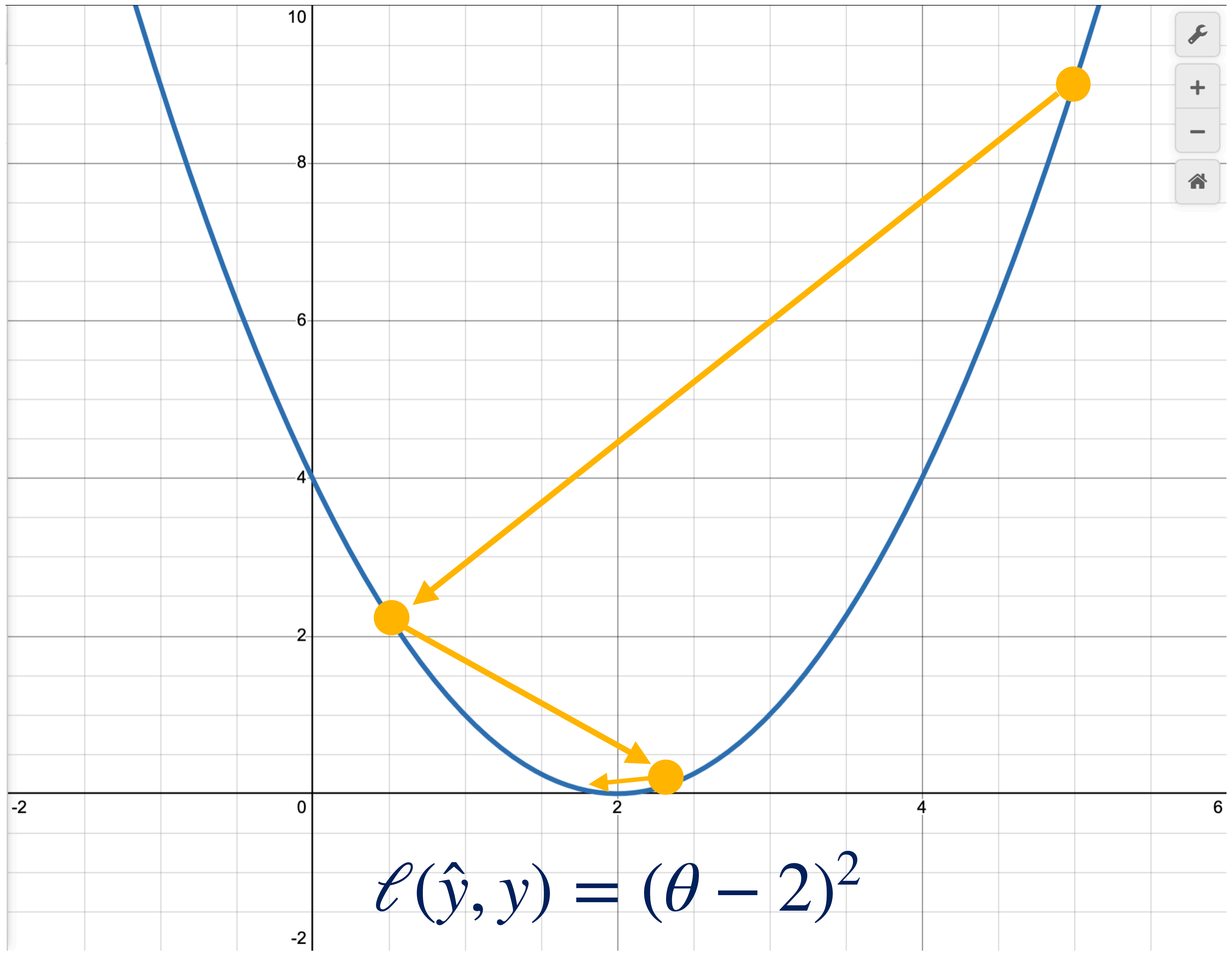


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

- Third step:
  - Plug in θ_2 to derivative:
    - $2 \cdot 2.25 - 4 = 0.5$
  - Update θ:
    - $\theta_3 = \theta_2 - 0.75 \cdot 0.5 = 1.875$

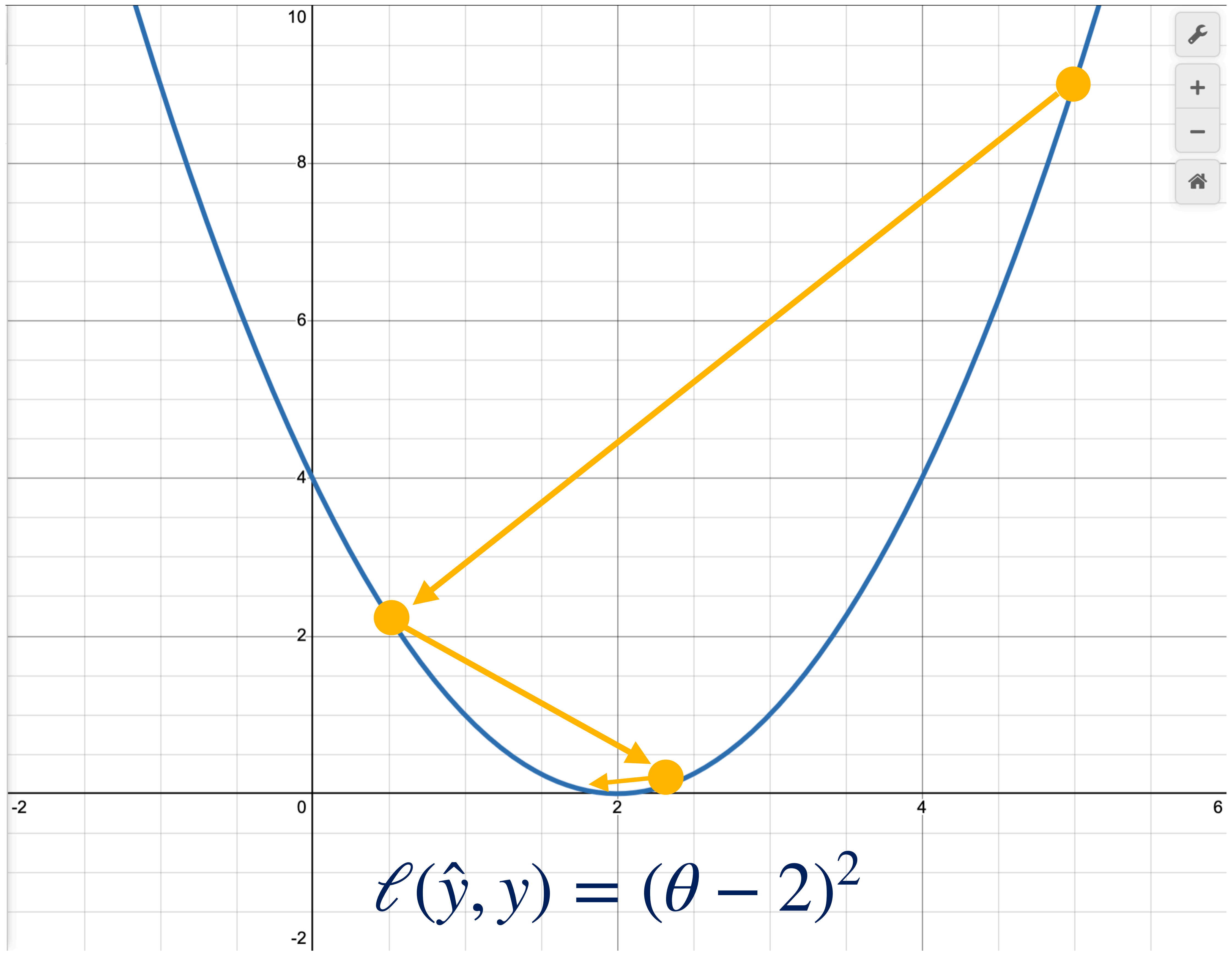


$\ell(\hat{y}, y) = (\theta - 2)^2$

$\ell'(\hat{y}, y) = 2\theta - 4$

Gradient Descent (second try)

- Third step:
  - Plug in θ_2 to derivative:
    - $2 \cdot 2.25 - 4 = 0.5$
  - Update θ:
    - $\theta_3 = \theta_2 - 0.75 \cdot 0.5 = 1.875$
- **Loss gets lower with every step!**

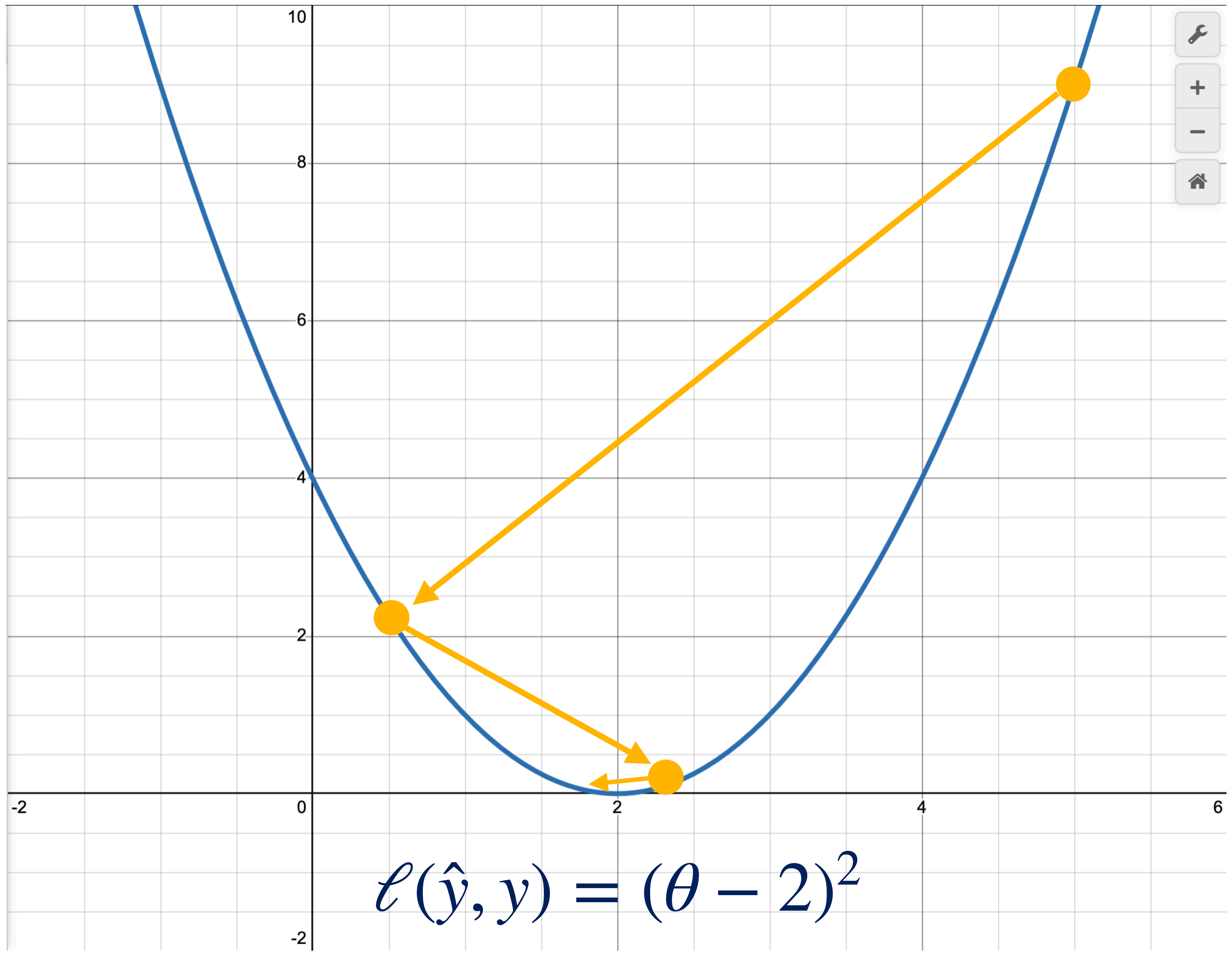


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Gradient Descent (second try)

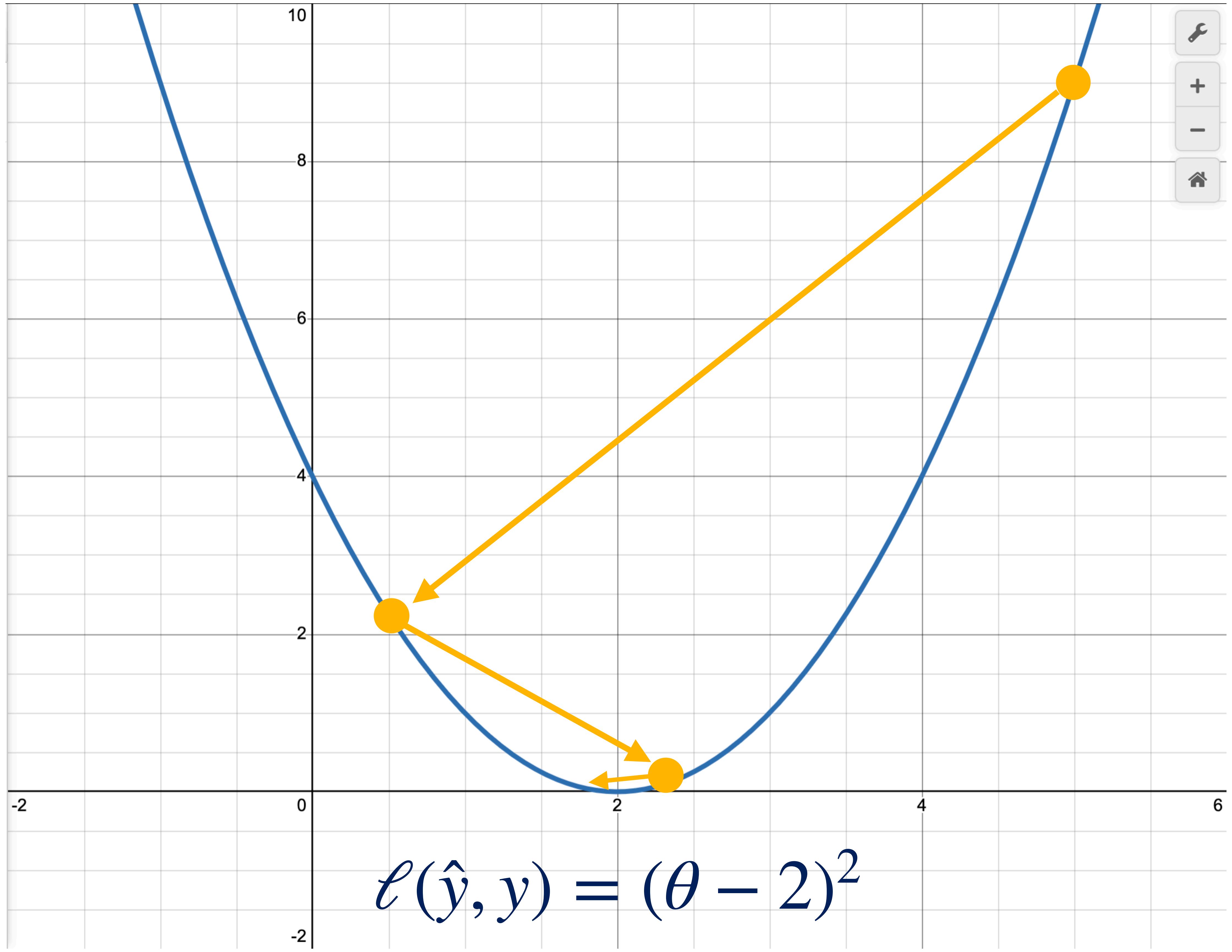
- Third step:
  - Plug in θ_2 to derivative:
    - $2 \cdot 2.25 - 4 = 0.5$
  - Update θ:
    - $\theta_3 = \theta_2 - 0.75 \cdot 0.5 = 1.875$
- **Loss gets lower with every step!**
- θ gets **arbitrarily close to the optimal value** with more steps



$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Learning Rate

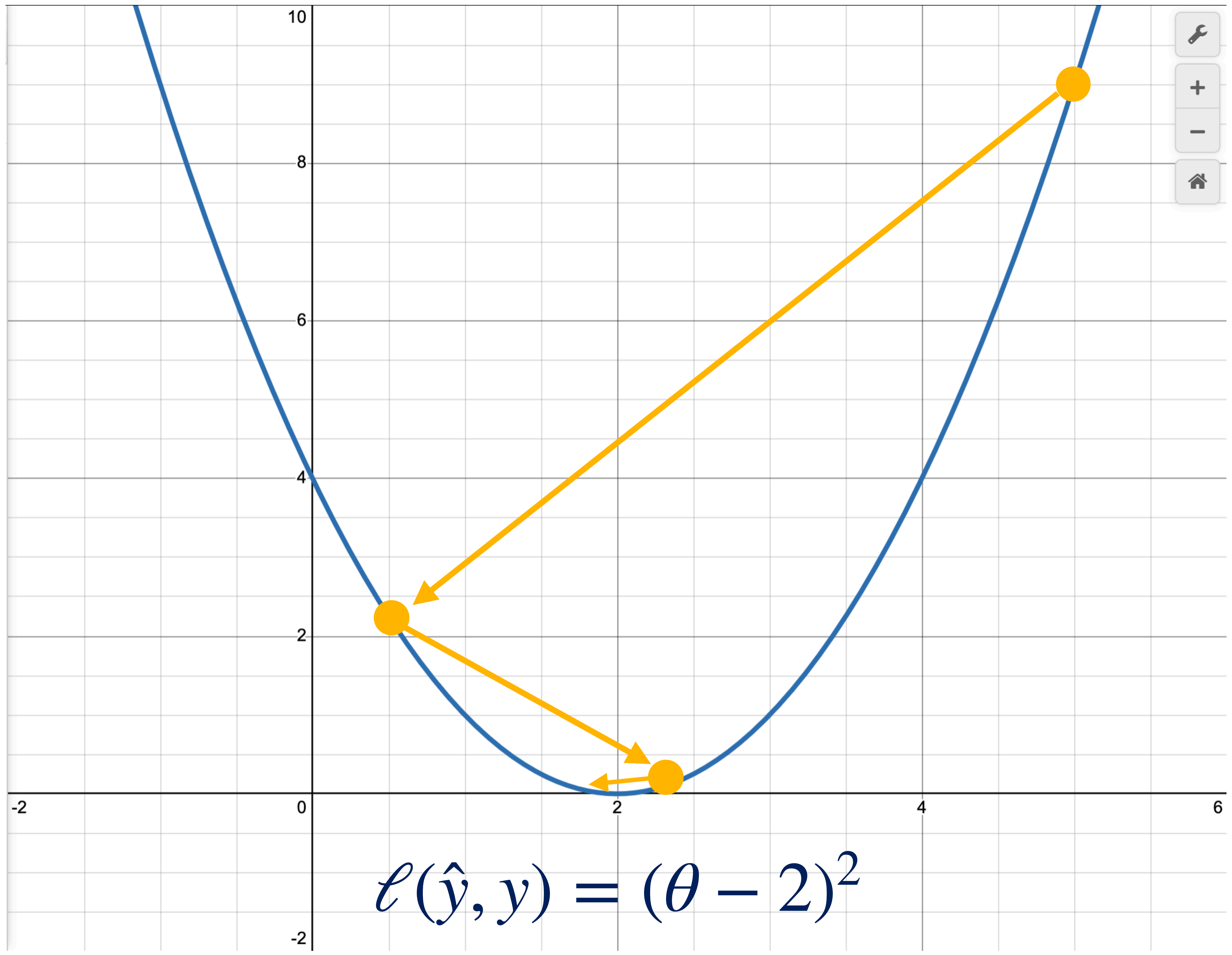


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Learning Rate

- The fraction by which we multiplied our adjustment is called the **Learning Rate**

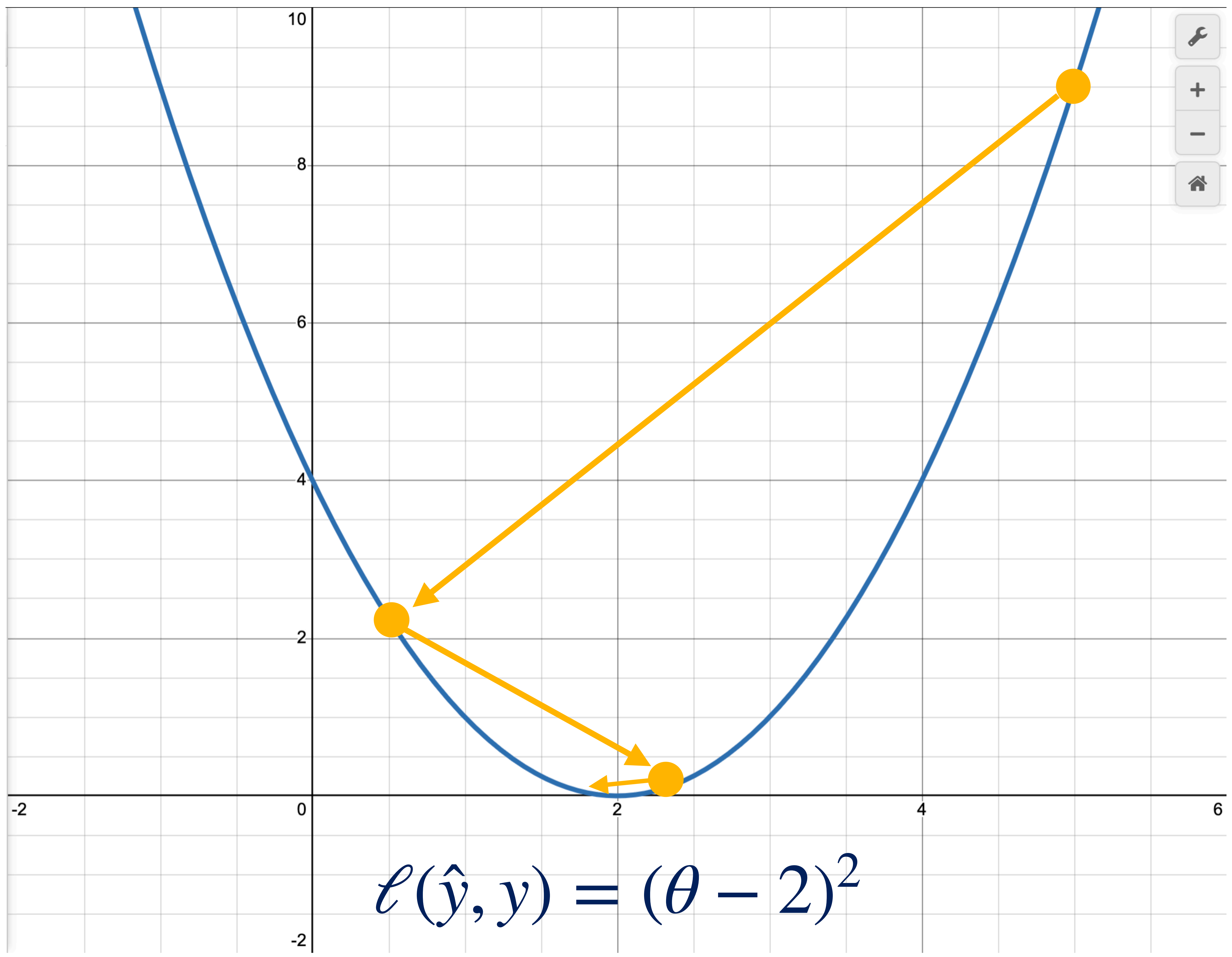


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Learning Rate

- The fraction by which we multiplied our adjustment is called the **Learning Rate**
- In practice, LR is chosen by **trial and error** (called "tuning")

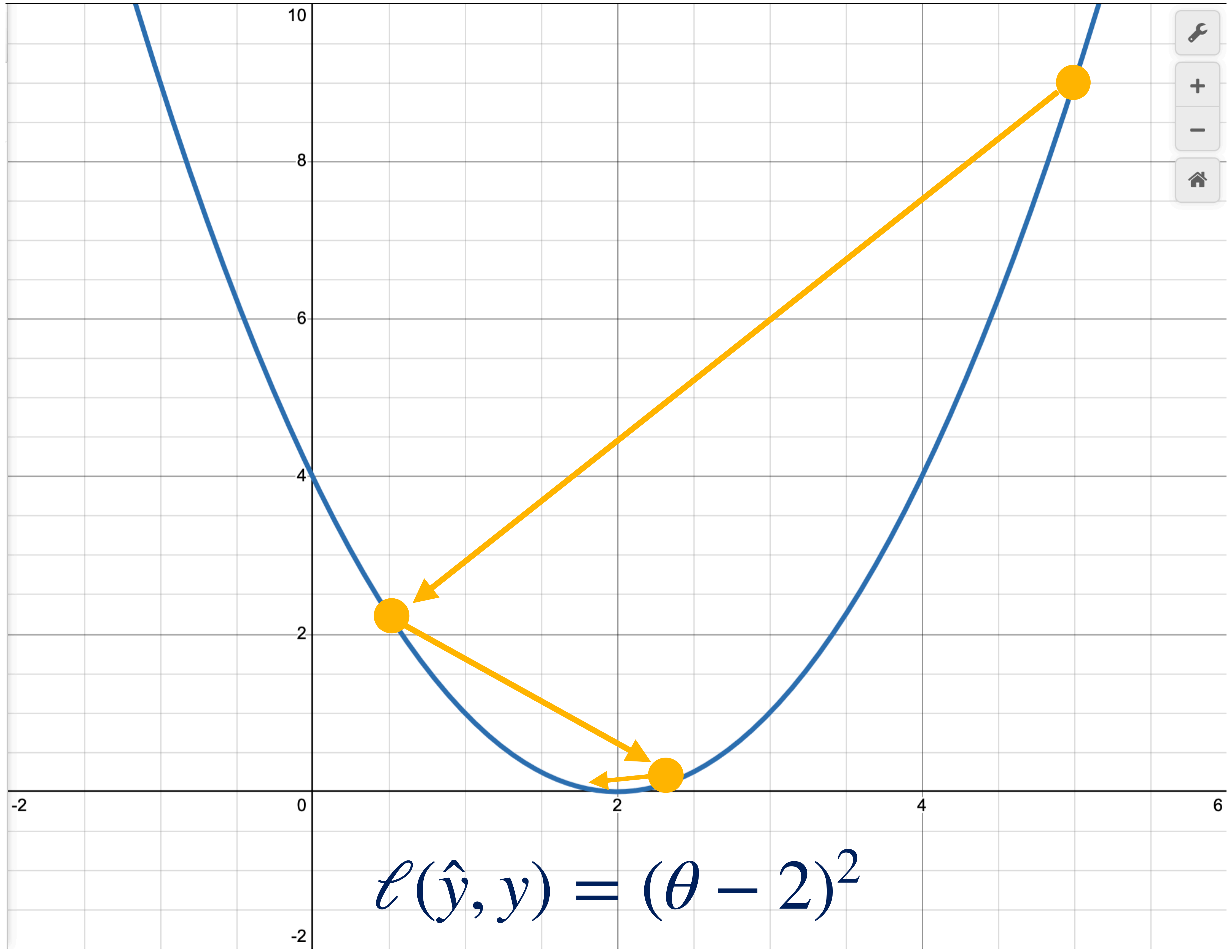


$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Learning Rate

- The fraction by which we multiplied our adjustment is called the **Learning Rate**
- In practice, LR is chosen by **trial and error** (called "tuning")
- Risks of different values
  - Too high → **"bouncing around"** and missing an optimum
  - Too low → taking **many steps** to reach the optimum



$$\ell(\hat{y}, y) = (\theta - 2)^2$$

$$\ell'(\hat{y}, y) = 2\theta - 4$$

Supervised Paradigms

Supervised Paradigms

- **Classification**: model outputs **discrete categories**
  - E.g. **binary** classification: spam/not-spam, human/bot, true/false
  - **Multi-class:** dog/cat/other, choosing word from finite vocabulary
  - Almost always outputs a **probability distribution**
    - Often achieved with the **sigmoid** or **softmax function**

Supervised Paradigms

- **Classification**: model outputs **discrete categories**
  - E.g. **binary** classification: spam/not-spam, human/bot, true/false
  - **Multi-class:** dog/cat/other, choosing word from finite vocabulary
  - Almost always outputs a **probability distribution**
    - Often achieved with the **sigmoid** or **softmax function**
- **Regression**: model outputs a **continuous number**
  - E.g. predicting the price of a house, sales numbers

Model Families (non-exhaustive)

Model Families (non-exhaustive)

- **Linear**: output decided from **linear combination of input features**
  - Examples: Linear/Logistic Regression, Naive Bayes, Linear SVM
  - Feature weights tend to be **fairly interpretable**
  - **Strong baseline** to try out with limited data!

Model Families (non-exhaustive)

- **Linear**: output decided from **linear combination of input features**
  - Examples: Linear/Logistic Regression, Naive Bayes, Linear SVM
  - Feature weights tend to be **fairly interpretable**
  - **Strong baseline** to try out with limited data!
- **Tree-based**: **branching decision processes** based on input features
  - Examples: Decision Tree, Random Forest
  - Single trees tend to **overfit** to small data (Random Forest helps)

Model Families (non-exhaustive)

- **Linear**: output decided from **linear combination of input features**
  - Examples: Linear/Logistic Regression, Naive Bayes, Linear SVM
  - Feature weights tend to be **fairly interpretable**
  - **Strong baseline** to try out with limited data!
- **Tree-based**: **branching decision processes** based on input features
  - Examples: Decision Tree, Random Forest
  - Single trees tend to **overfit** to small data (Random Forest helps)
- **Neural Networks**: hierarchical **non-linear transformations** of input features
  - Examples: Feedforward, CNNs, RNNs, Transformers, etc.
  - **Quickly overfit** to limited data (without regularization + other tricks)

Model Generalization

ML Dataset Splits

ML Dataset Splits

- Reminder: we usually **split our dataset** into multiple subsets

ML Dataset Splits

- Reminder: we usually **split our dataset** into multiple subsets
- **Training**: the data that the model actually uses to learn / optimize for

ML Dataset Splits

- Reminder: we usually **split our dataset** into multiple subsets
- **Training**: the data that the model actually uses to learn / optimize for
- **Test**: data that is **held-out** until the model is fully trained
  - Tests generalization to **completely unseen data**
  - Important not to touch until the end!

ML Dataset Splits

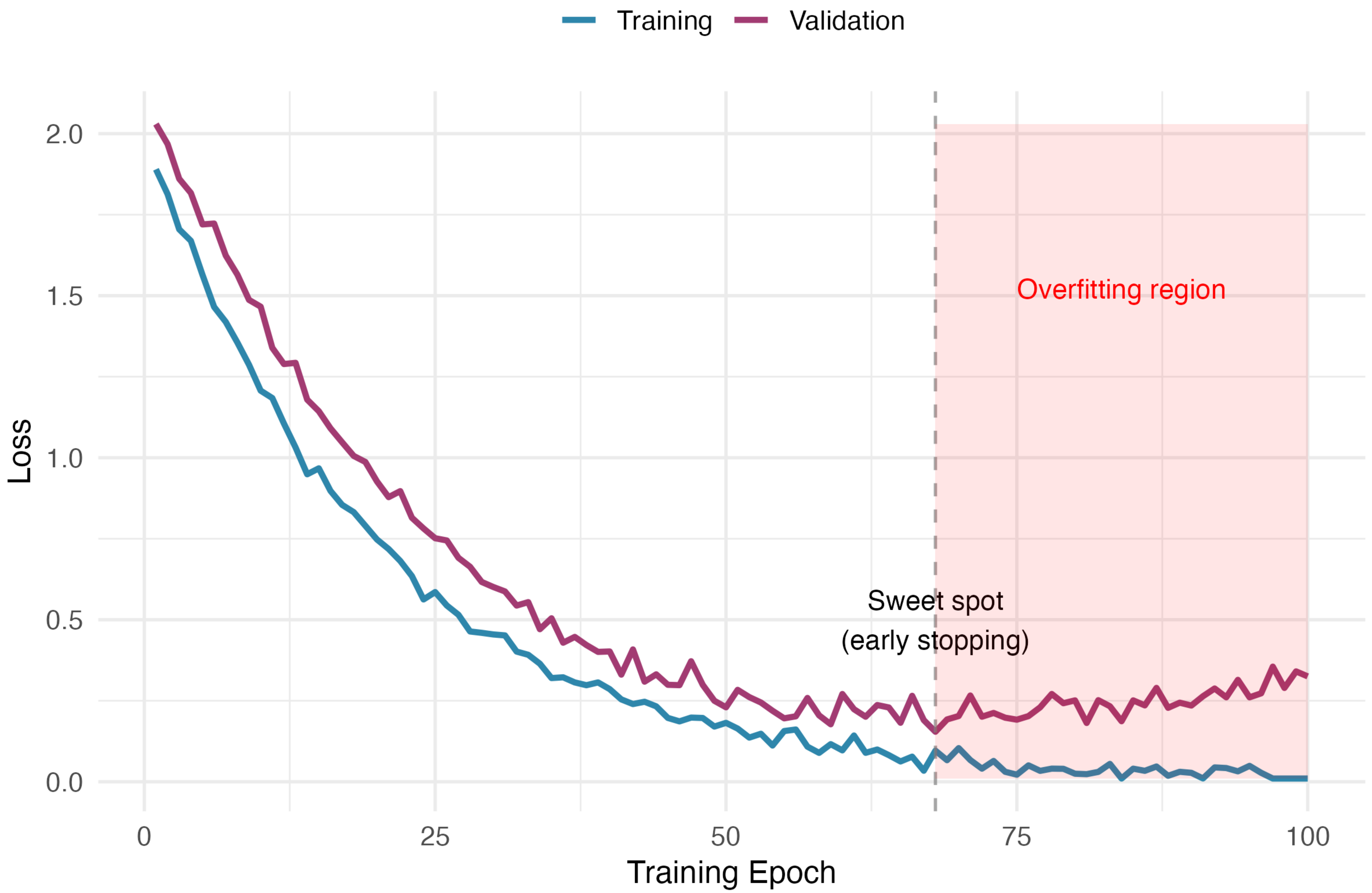
- Reminder: we usually **split our dataset** into multiple subsets
- **Training**: the data that the model actually uses to learn / optimize for
- **Test**: data that is **held-out** until the model is fully trained
  - Tests generalization to **completely unseen data**
  - Important not to touch until the end!
- **"Development"/"Validation"**: used **during training** to tune generalization
  - Also **held-out** (not actually seen by model during training)
  - Used to help **tune hyperparameters** and **detect overfitting**

ML Dataset Splits

- Reminder: we usually **split our dataset** into multiple subsets
- **Training**: the data that the model actually uses to learn / optimize for
- **Test**: data that is **held-out** until the model is fully trained
  - Tests generalization to **completely unseen data**
  - Important not to touch until the end!
- **"Development"/"Validation"**: used **during training** to tune generalization
  - Also **held-out** (not actually seen by model during training)
  - Used to help **tune hyperparameters** and **detect overfitting**
- What is overfitting?

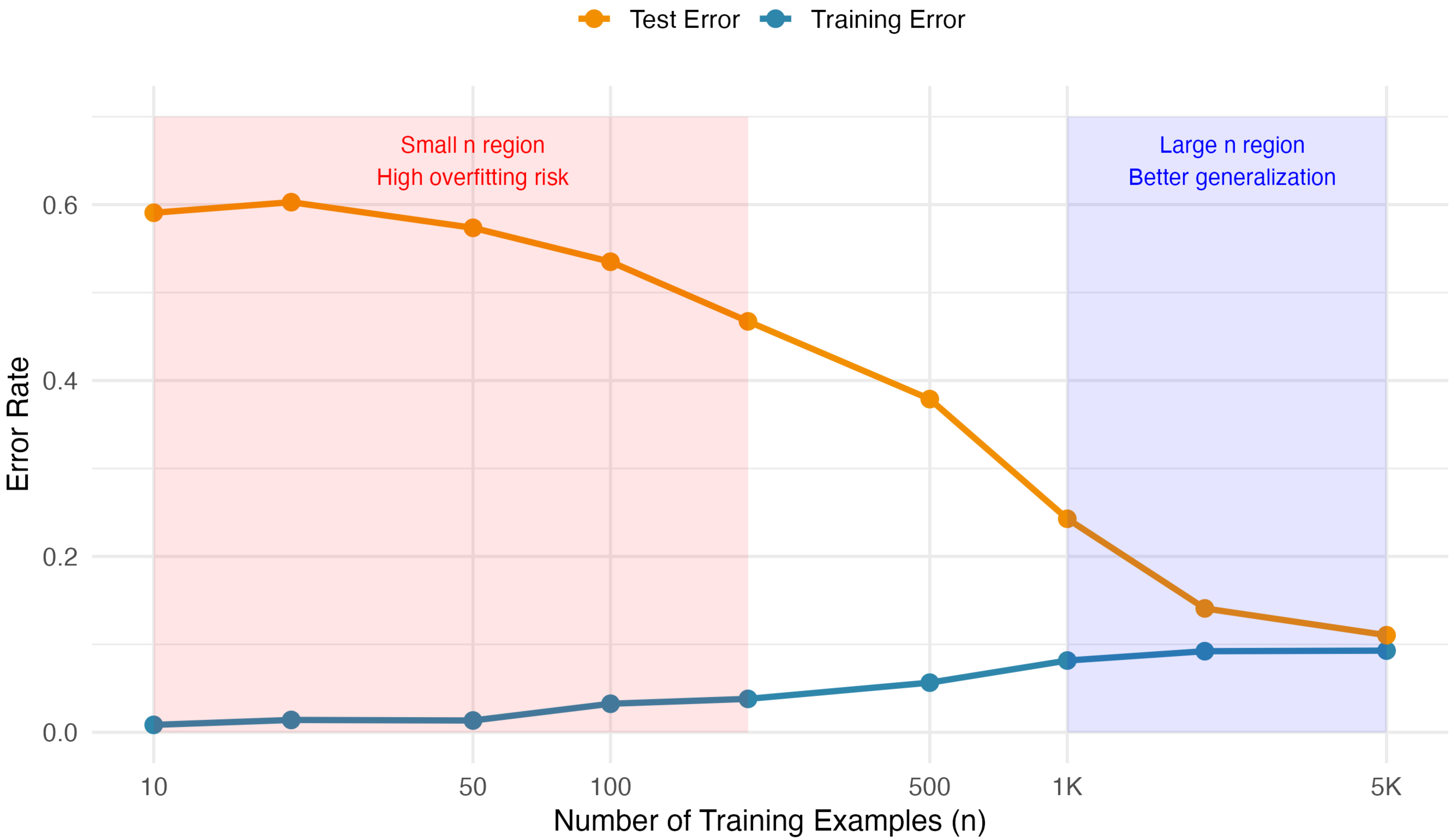
What Overfitting Looks Like

Training loss keeps improving, but validation loss increases

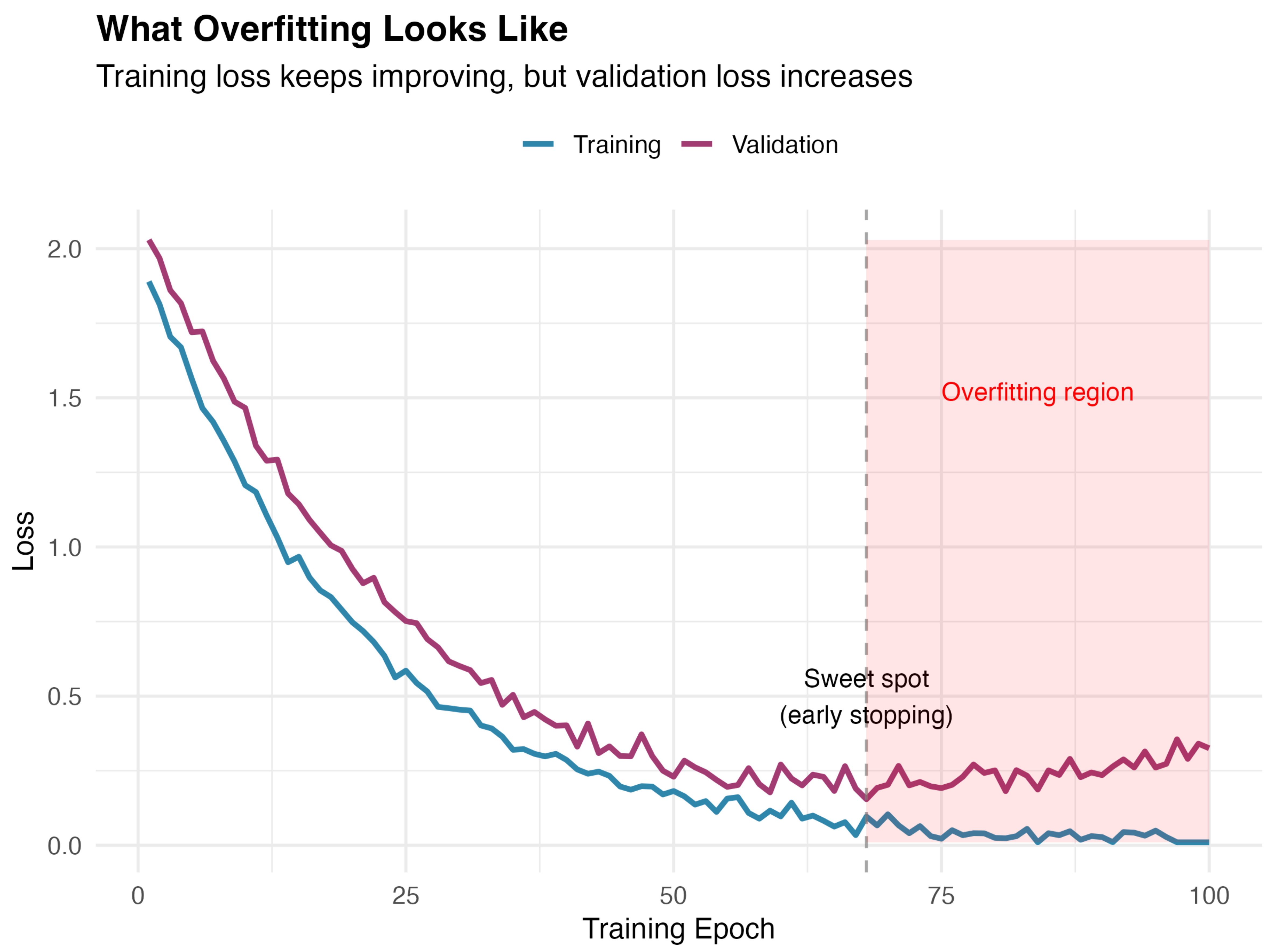


The Generalization Gap Depends on Data Size

This is the fundamental picture of the course

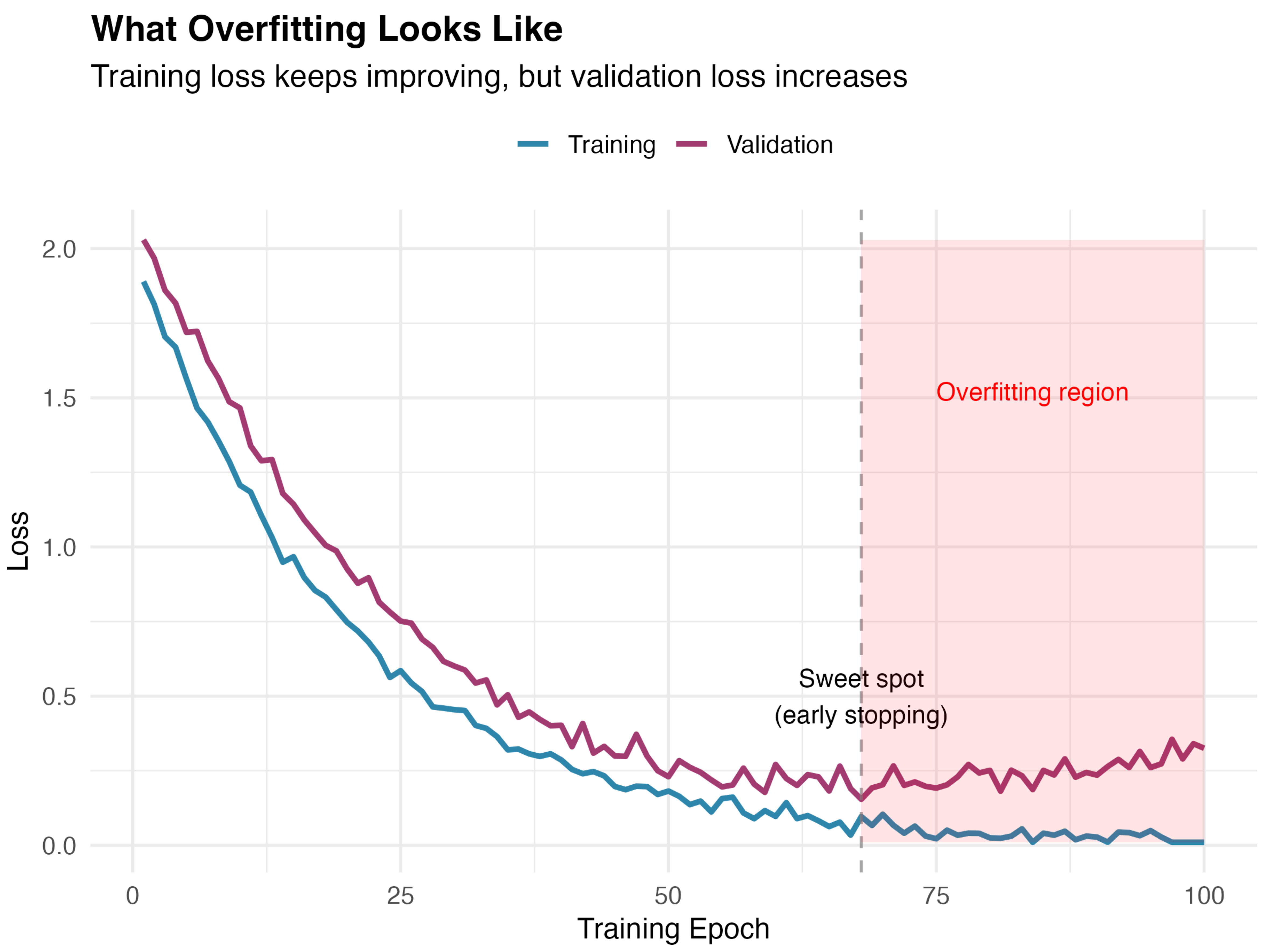


Why does overfitting happen?



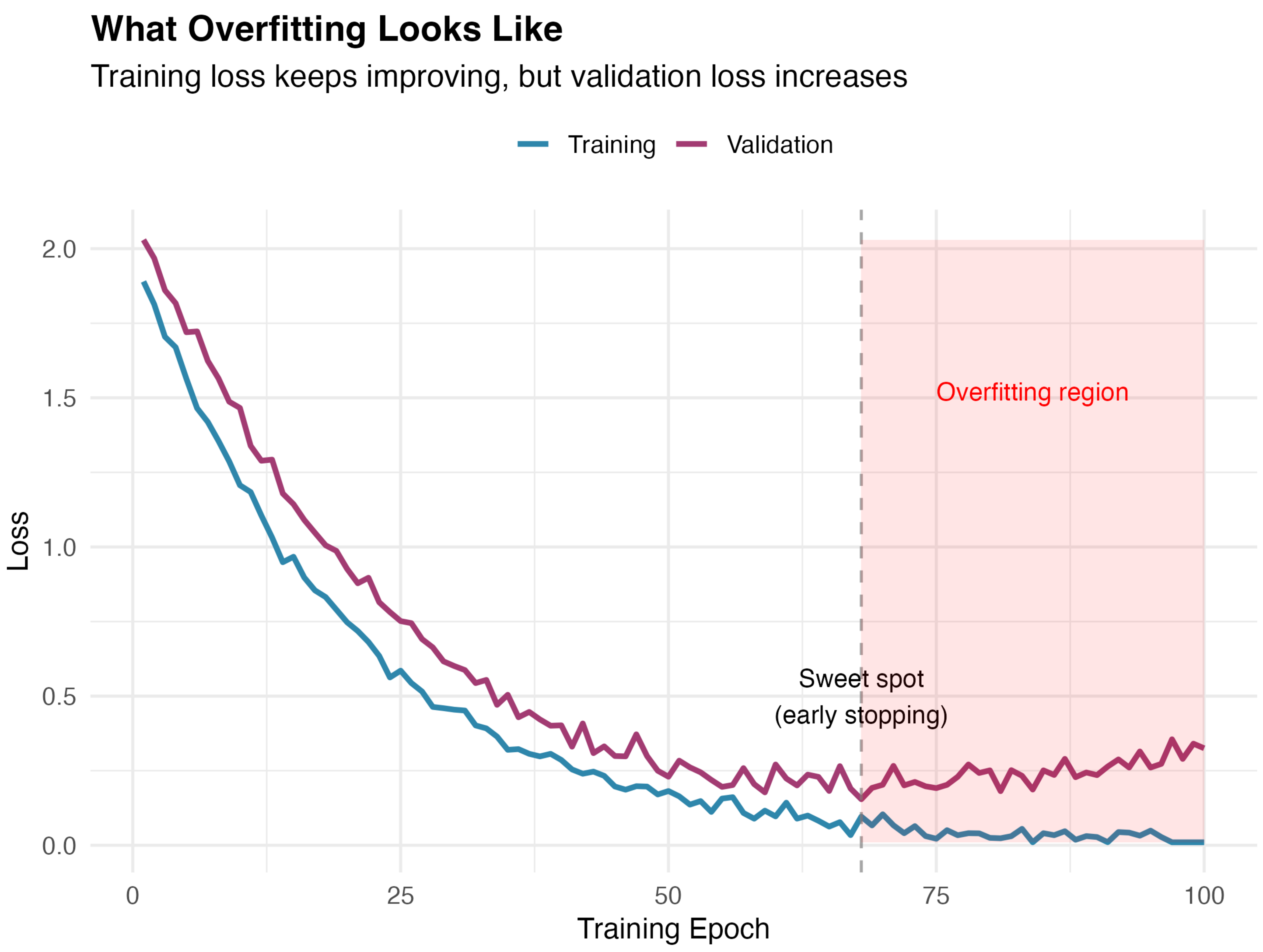
Why does overfitting happen?

- Model overfitting depends on **two key factors**:
  - **Data size**: how much data is there to learn from?
  - **Model complexity**: what capacity does the model have to **fit complex patterns?**

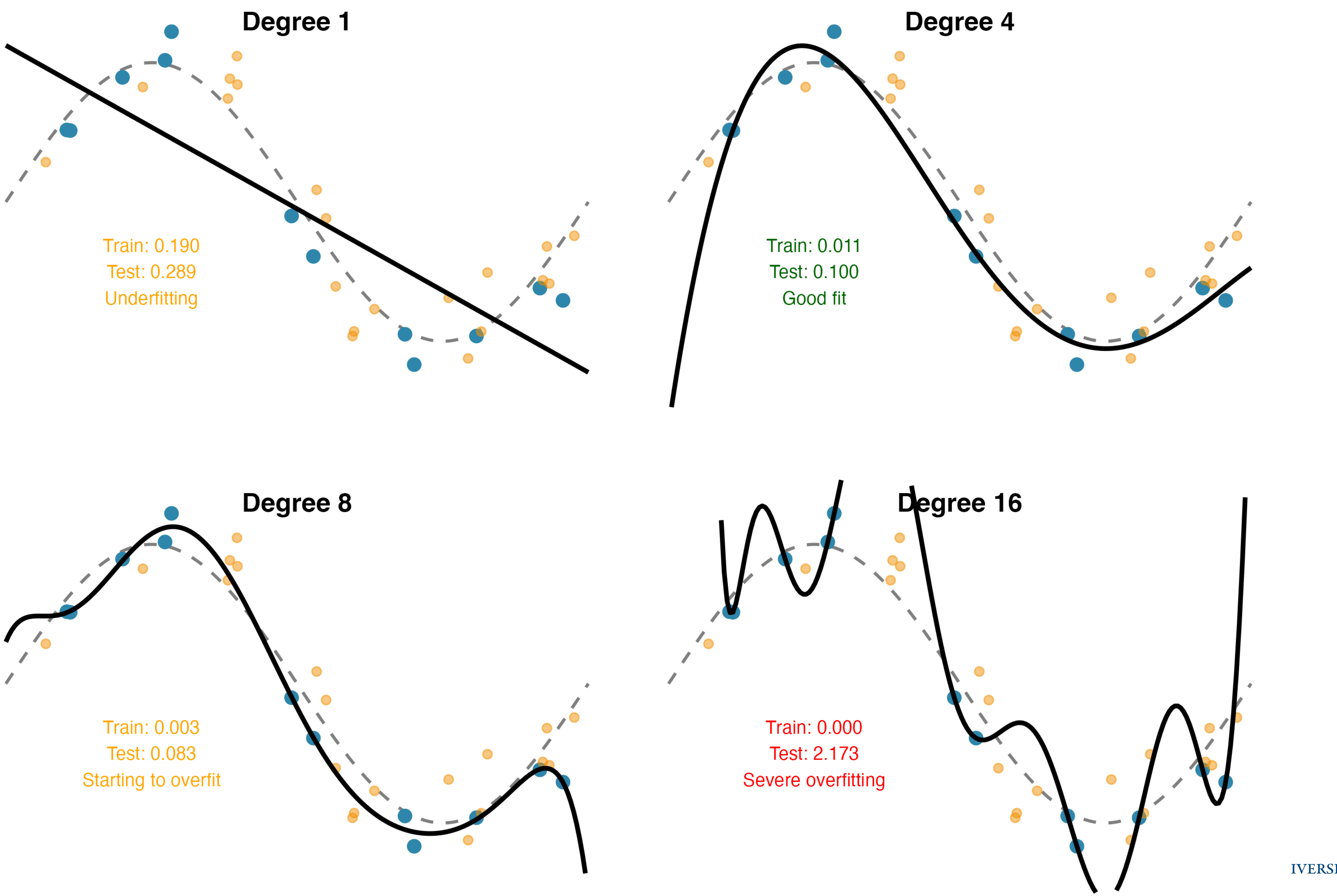


Why does overfitting happen?

- Model overfitting depends on **two key factors**:
  - **Data size**: how much data is there to learn from?
  - **Model complexity**: what capacity does the model have to **fit complex patterns?**
- The latter is classically demonstrated with **polynomial regression** (next slide)



Model Complexity vs. Overfitting
Blue = training data, Orange = test data, Dashed = true function



Bias and Variance

Bias and Variance

- **Bias:** the error stemming from **model assumptions**
  - Example: fitting a **linear regression** to **non-linear data**
  - Intuition: a model's **lack of flexibility**, leading to **under-fitting** the data

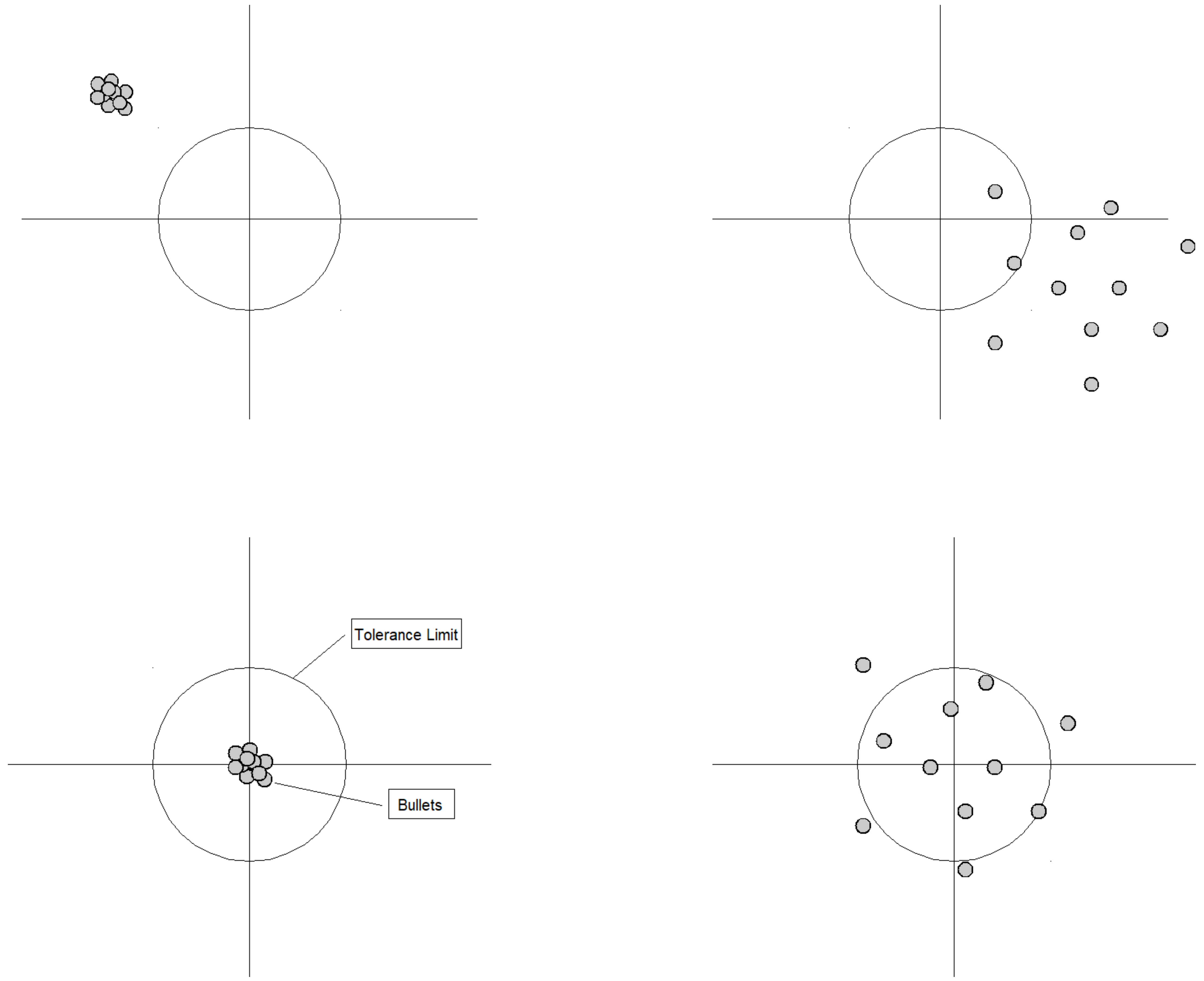
Bias and Variance

- **Bias:** the error stemming from **model assumptions**
  - Example: fitting a **linear regression** to **non-linear data**
  - Intuition: a model's **lack of flexibility**, leading to **under-fitting** the data
- **Variance:** the error from **sensitivity to noise** in the **training data**
  - Example: fitting a **large neural network** to **limited data**
  - Intuition: if we **randomly re-sampled** the training data, how much would the model's outputs change?

Bias and Variance

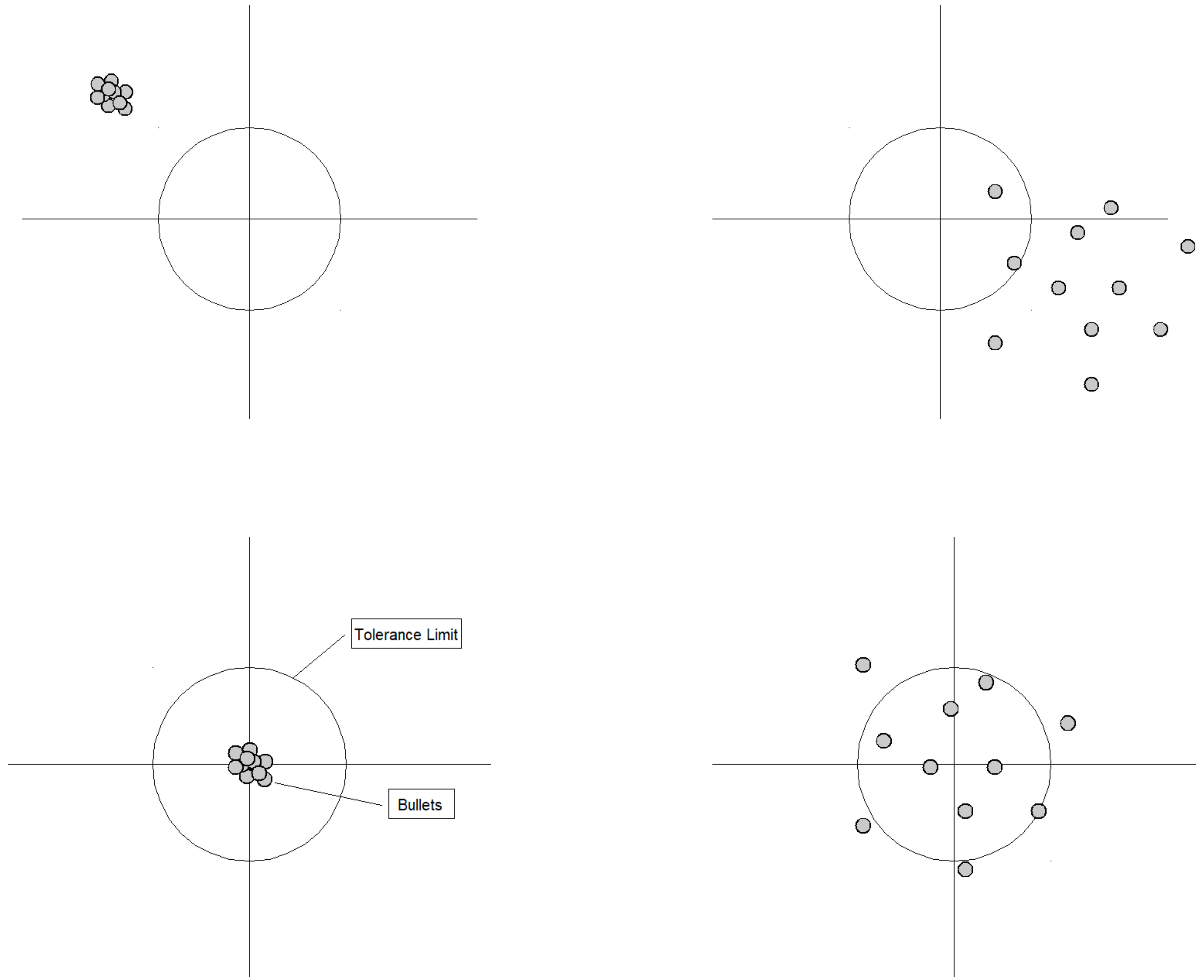
- **Bias:** the error stemming from **model assumptions**
  - Example: fitting a **linear regression** to **non-linear data**
  - Intuition: a model's **lack of flexibility**, leading to **under-fitting** the data
- **Variance:** the error from **sensitivity to noise** in the **training data**
  - Example: fitting a **large neural network** to **limited data**
  - Intuition: if we **randomly re-sampled** the training data, how much would the model's outputs change?
- Bias and Variance inherently form a **tradeoff** - we can't minimize both at once

Bias and Variance



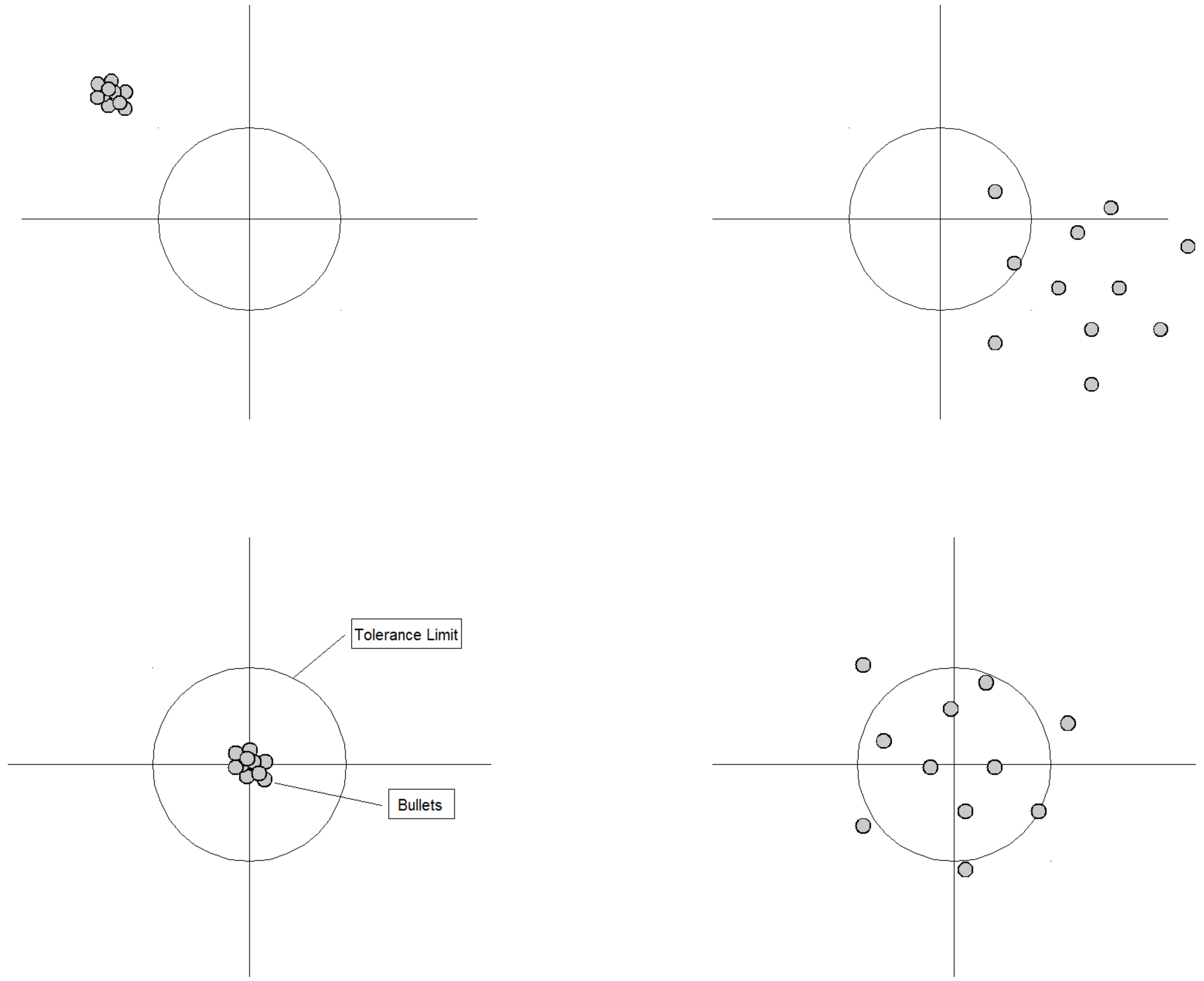
Bias and Variance

high bias
low variance



Bias and Variance

high bias
low variance

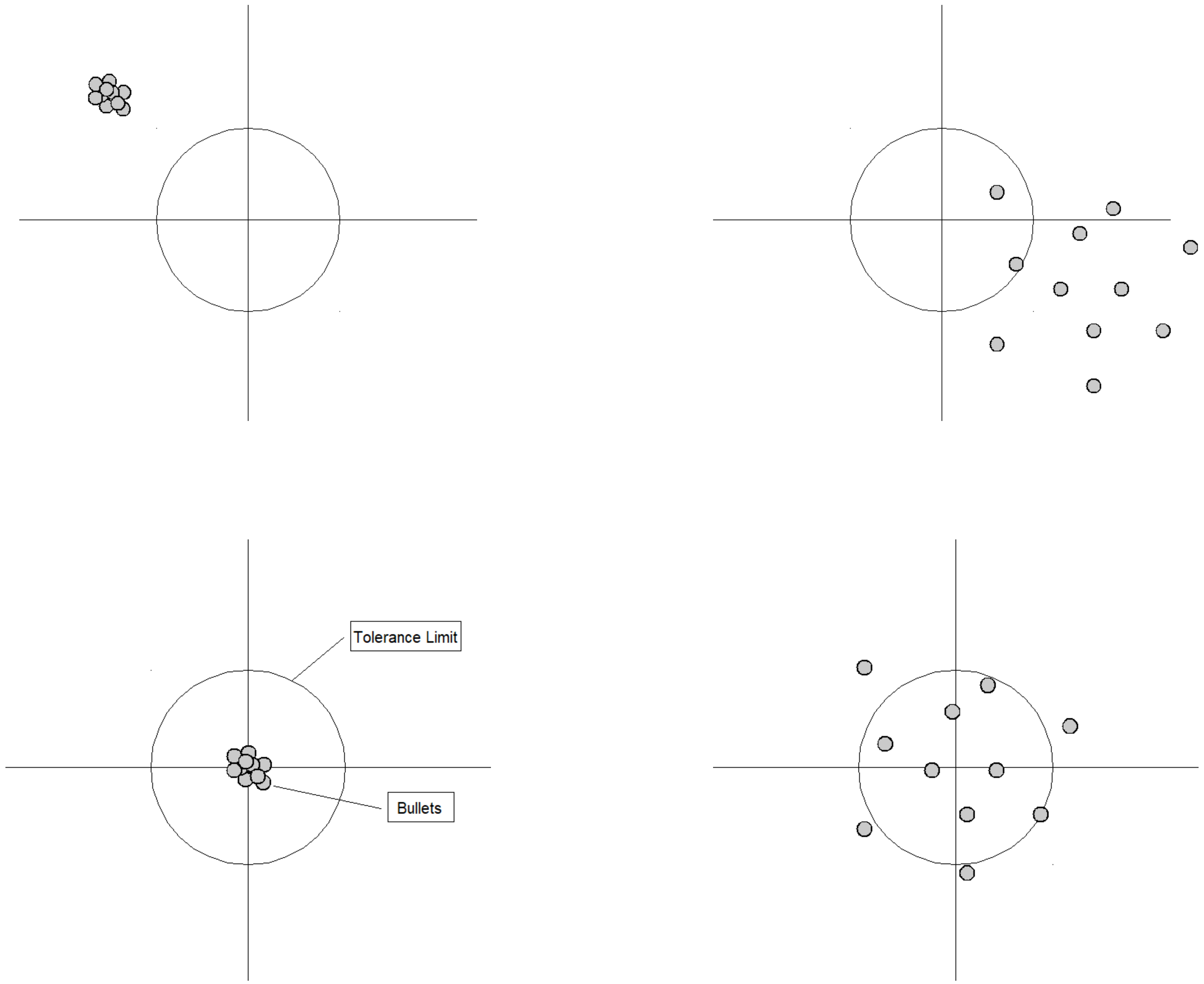


high bias
high variance

Bias and Variance

high bias
low variance

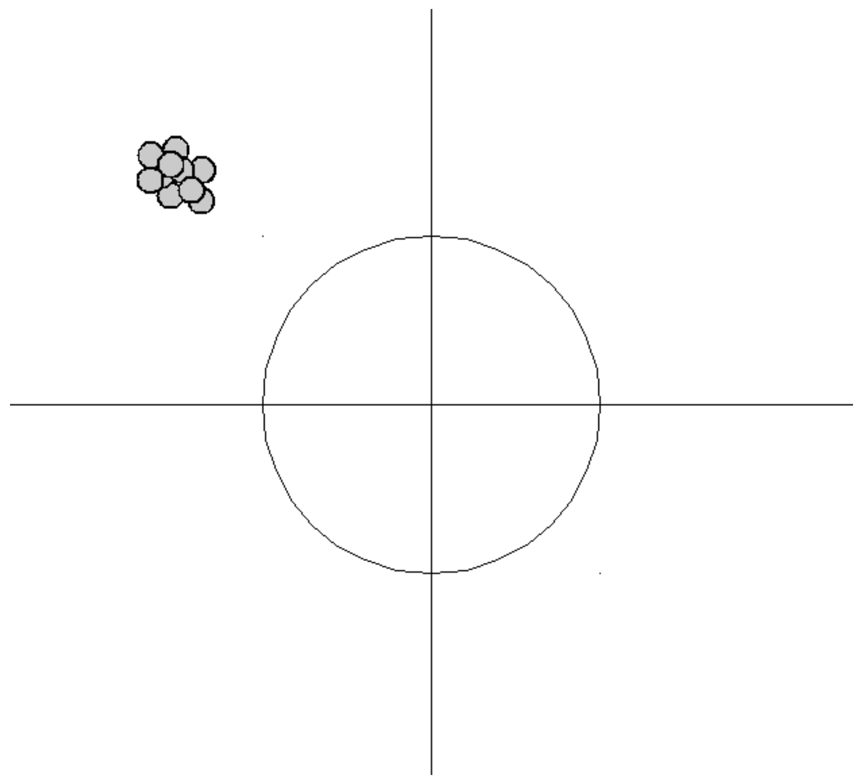
high bias
high variance

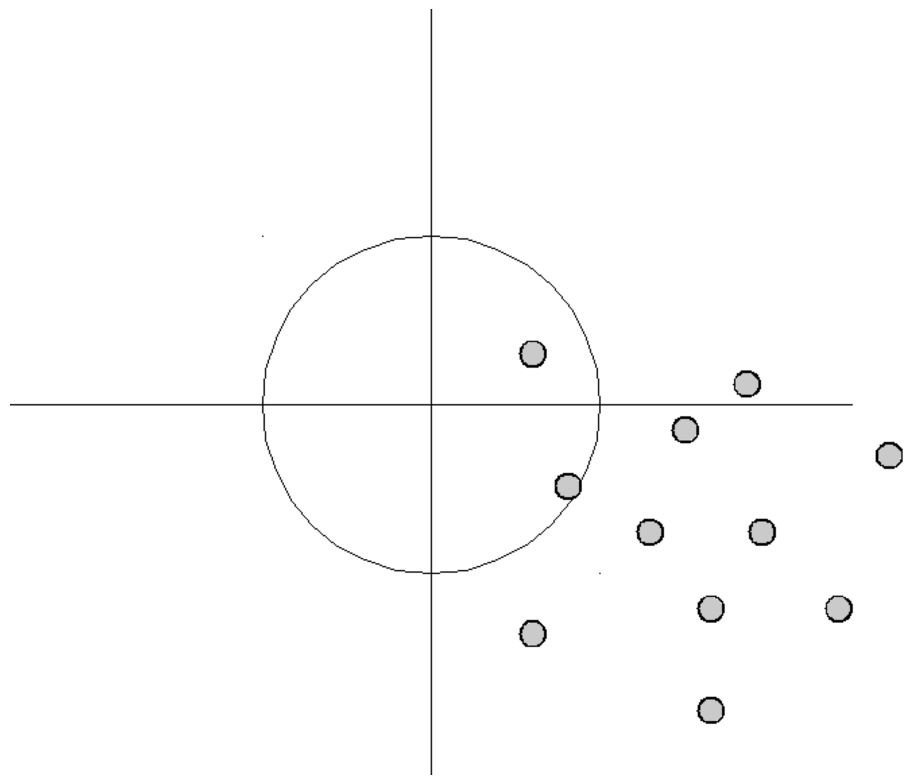


low bias
low variance

Bias and Variance

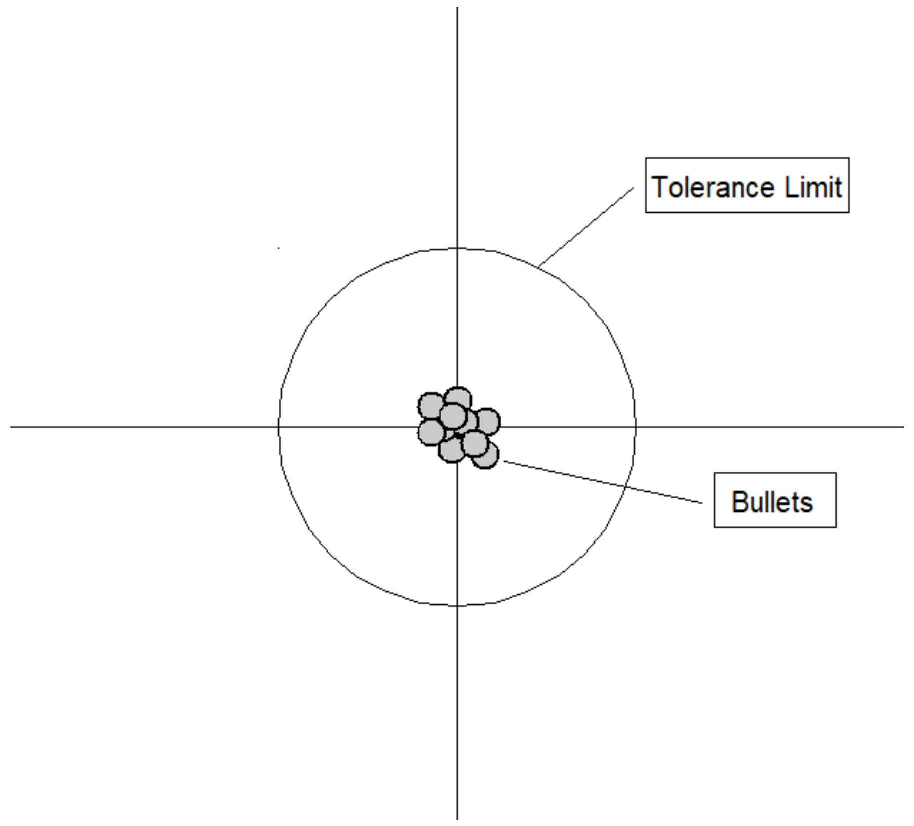
high bias
low variance

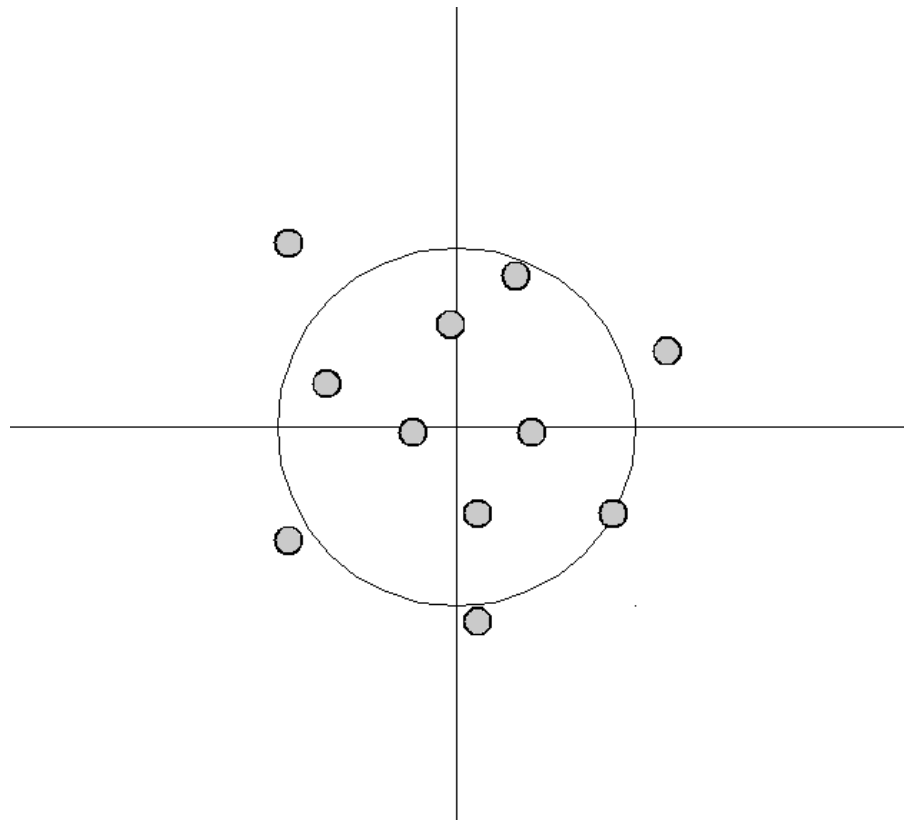




high bias
high variance

low bias
low variance

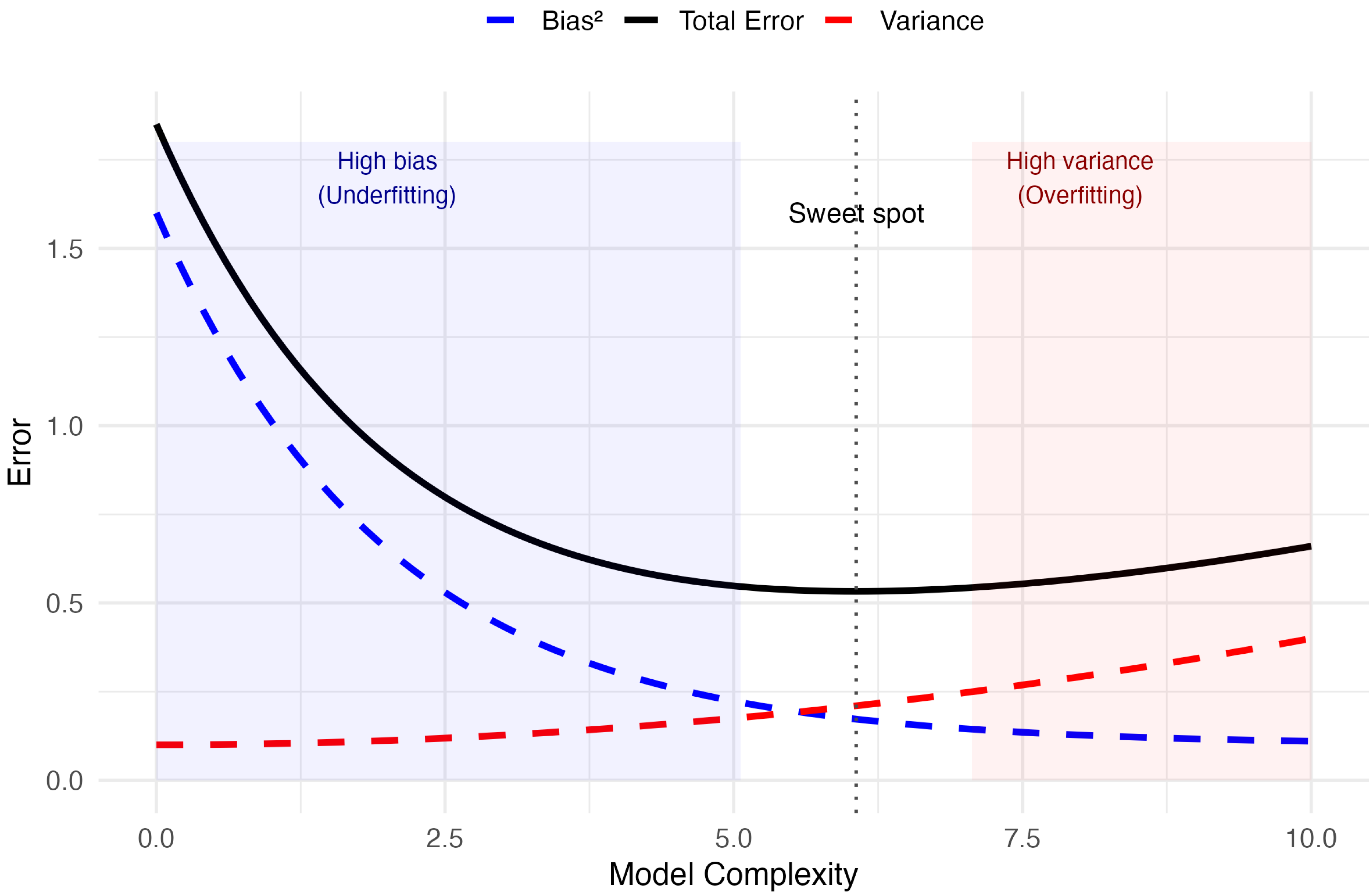




low bias
high variance

The Bias-Variance Tradeoff

Total error is minimized at intermediate complexity



Mitigating Overfitting

Regularization

Regularization

- Many techniques mitigate overfitting by **preferring simpler solutions**

Regularization

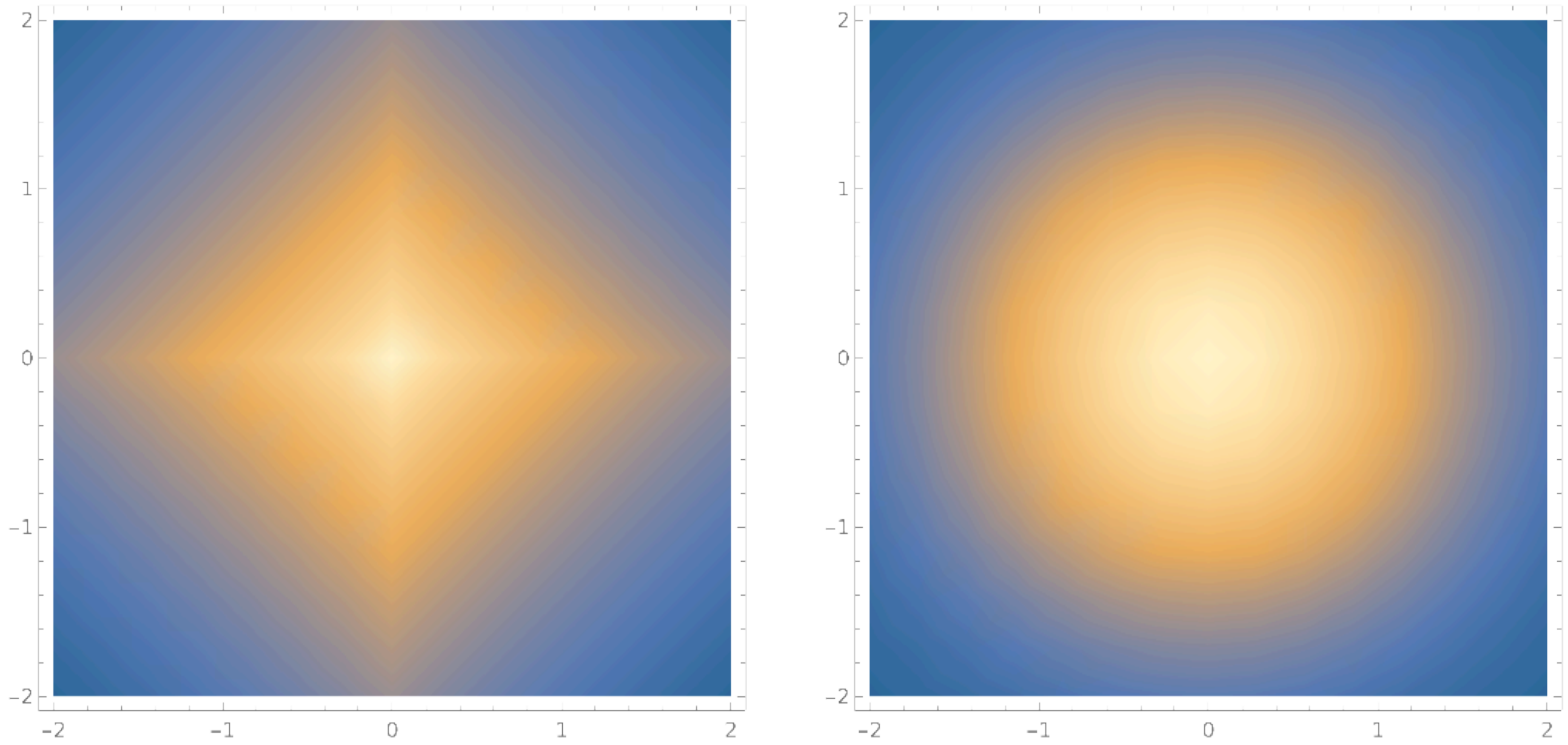
- Many techniques mitigate overfitting by **preferring simpler solutions**
- **L2 Regularization**: penalizes large weights
  - Based on the "L2 Norm" (**Euclidian Distance**) of the weight vector
  - Strength controlled by **hyperparameter** λ: loss += $\lambda \Sigma \theta_i^2$

Regularization

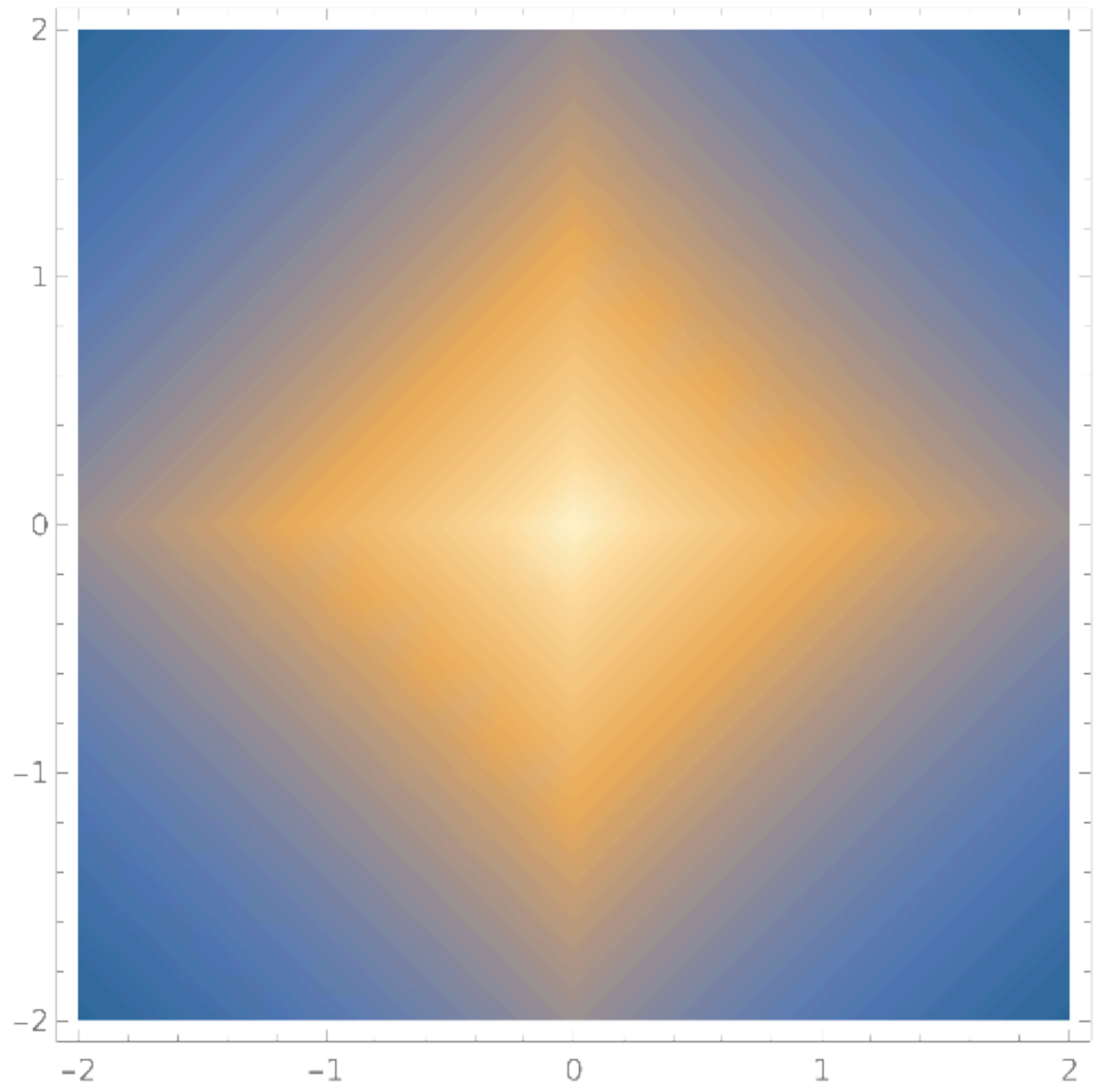
- Many techniques mitigate overfitting by **preferring simpler solutions**
- **L2 Regularization**: penalizes large weights
  - Based on the "L2 Norm" (**Euclidian Distance**) of the weight vector
  - Strength controlled by **hyperparameter** λ**:** loss += $\lambda\Sigma\theta_i^2$
- **L1 Regularization**: penalizes large weights (in a different way)
  - Based on "L1 Norm" aka **"Manhattan Distance"** of the weight vector
  - Tends to **drive some weights to zero** (creating a sparse model)
  - loss += $\lambda\Sigma|\theta_i|$

L1 and L2 Visualized

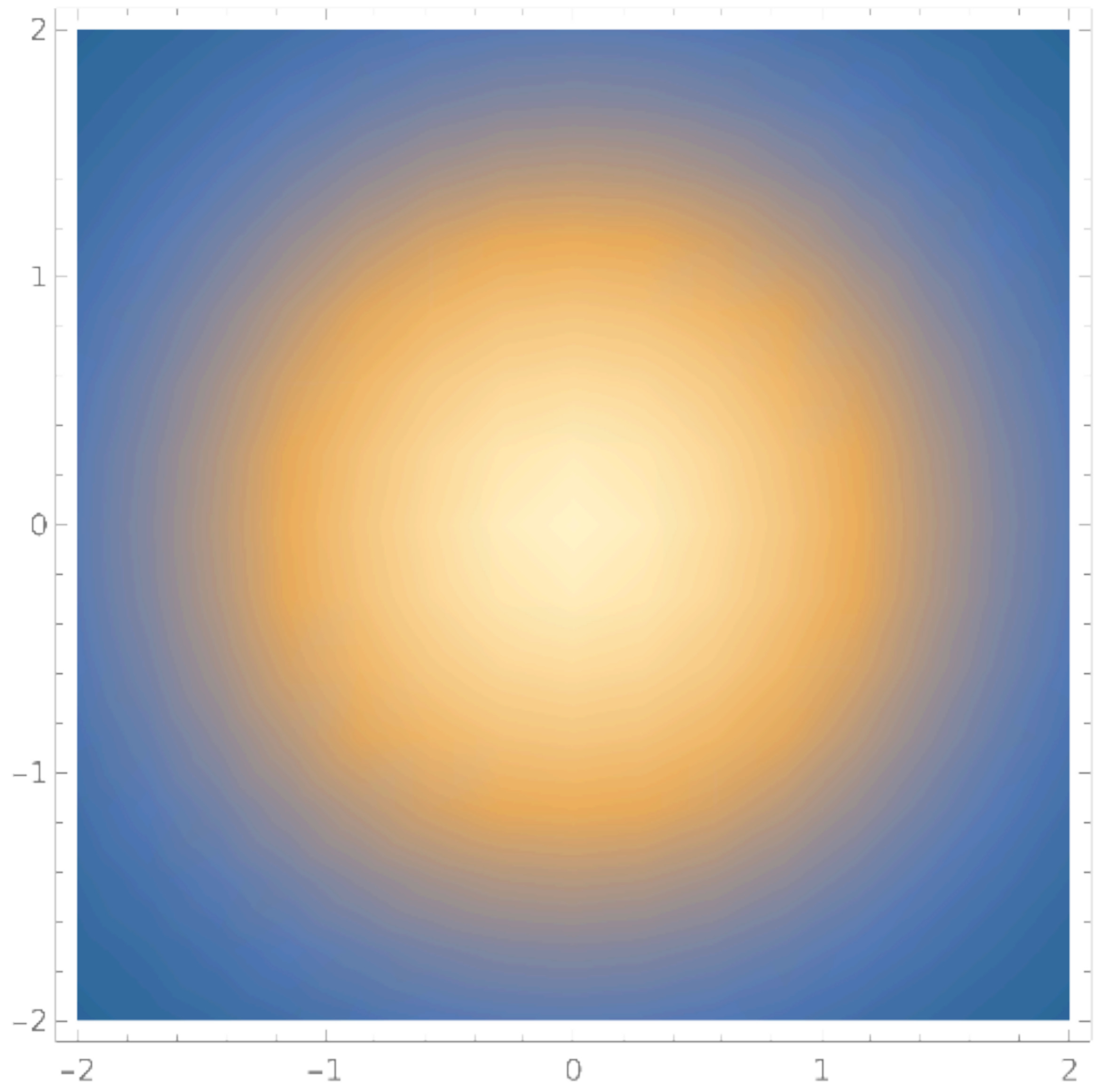
L1 and L2 Visualized



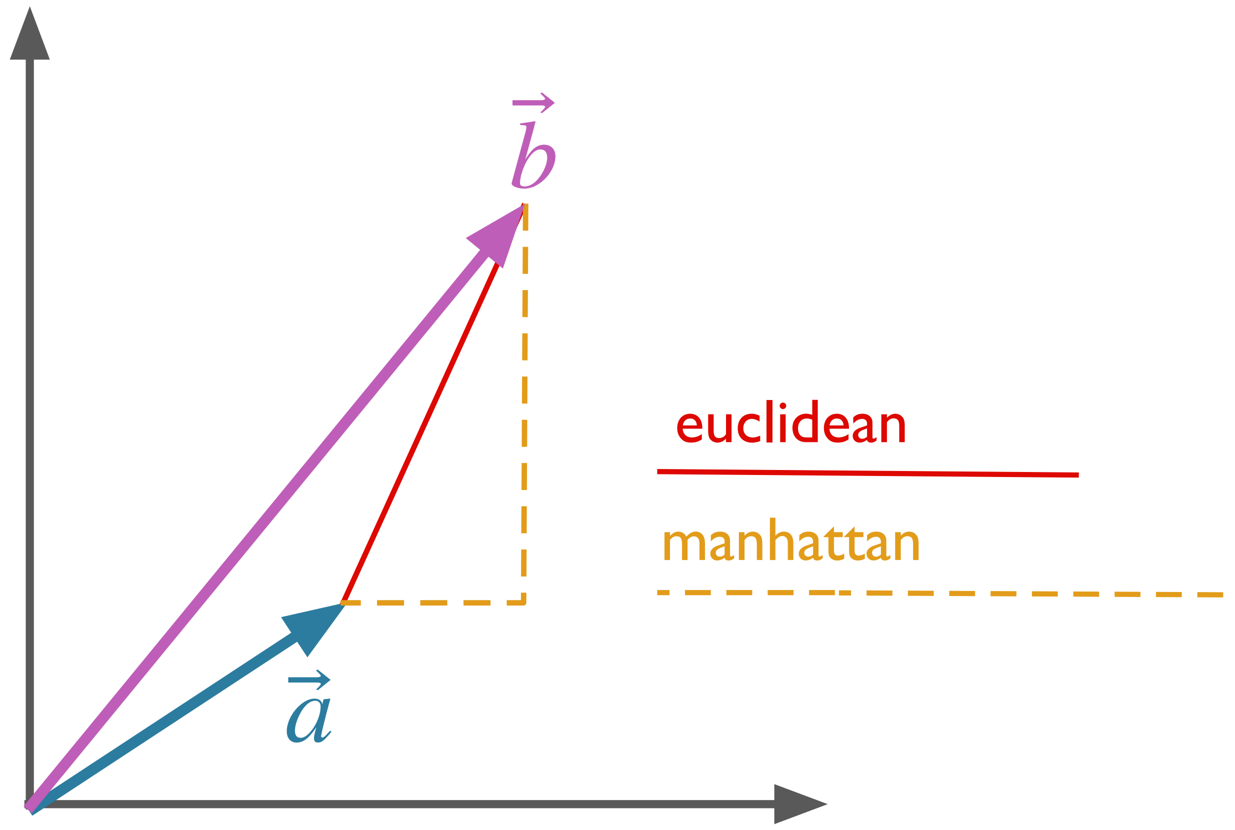
L1 and L2 Visualized



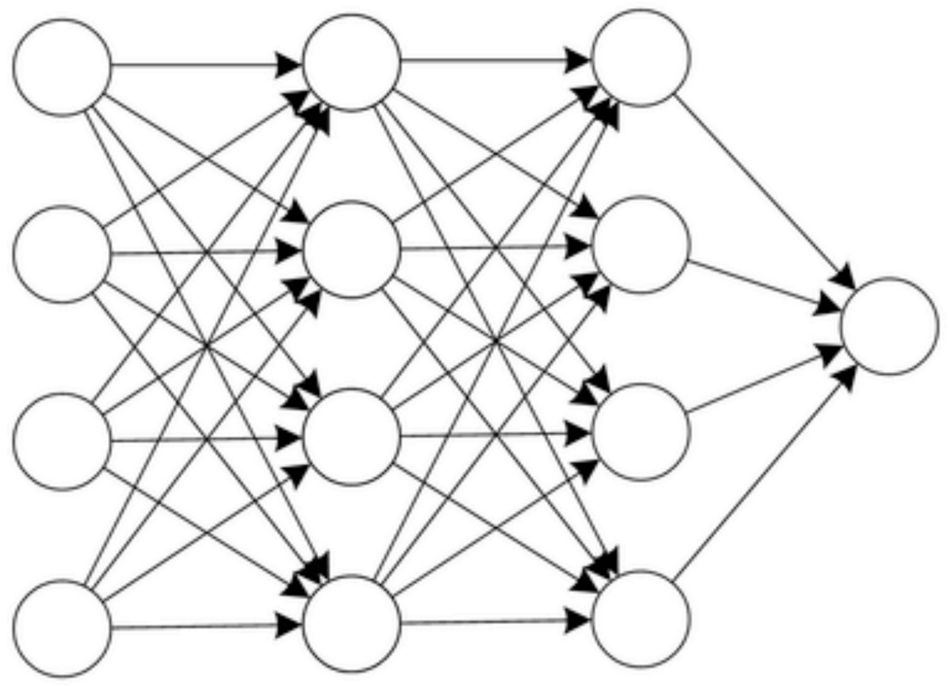
L1



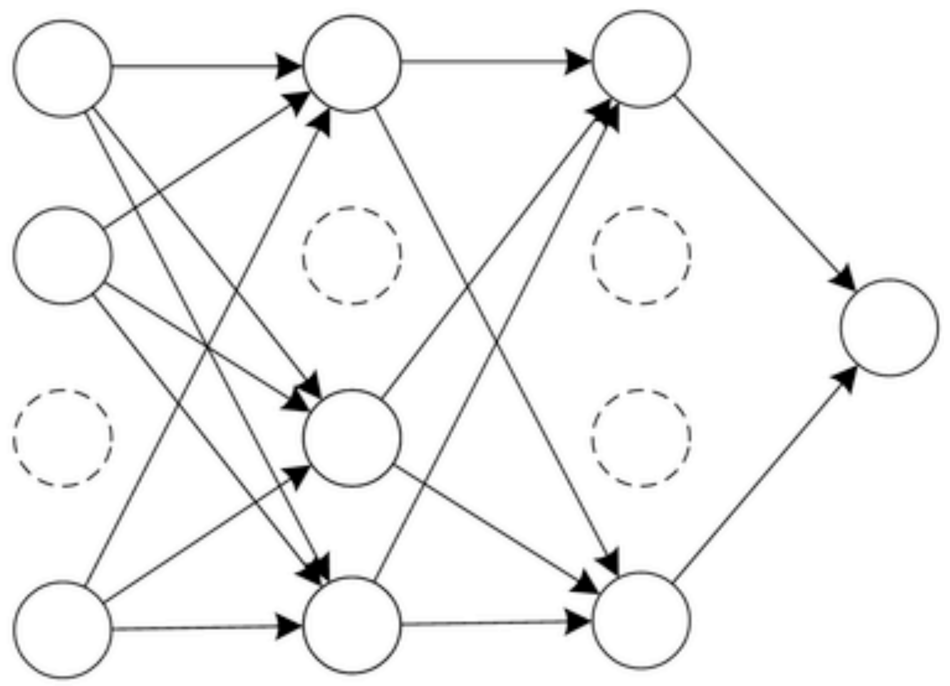
L2



Dropout



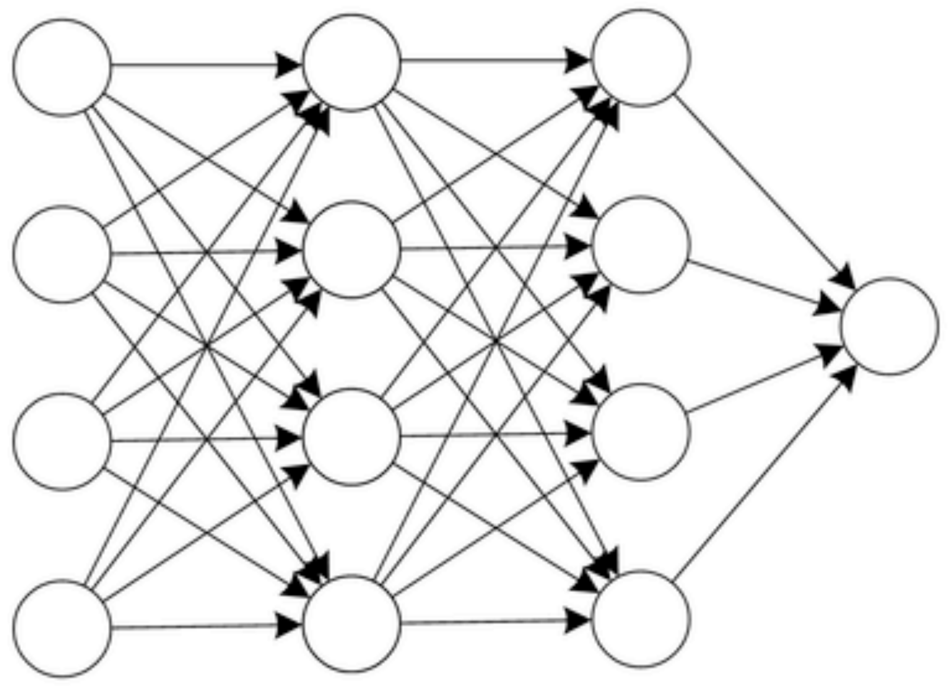
(a) Standard Neural Network



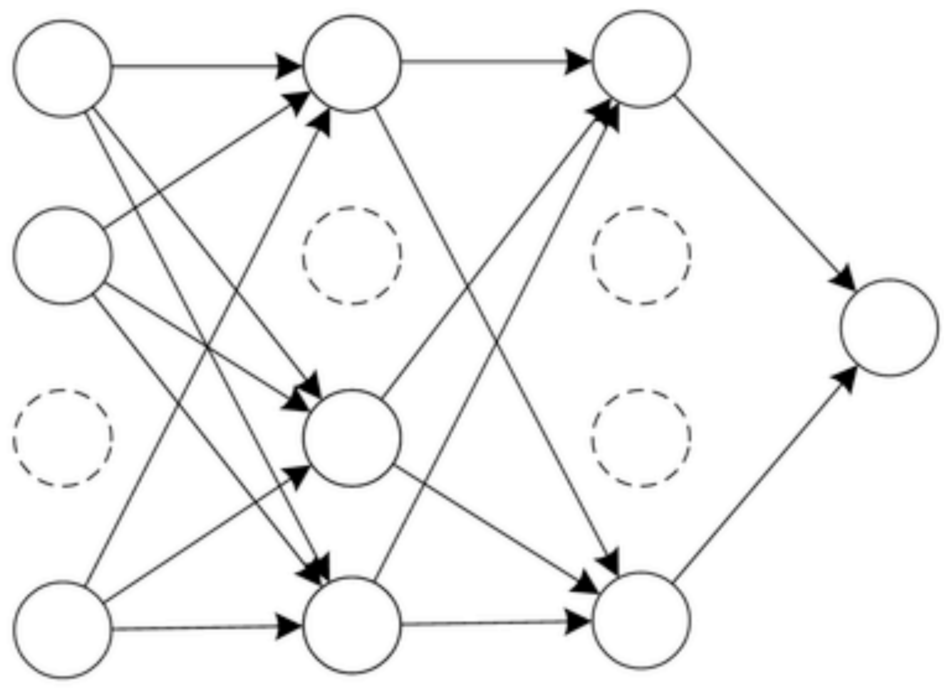
(b) Network after Dropout

Dropout

- Mostly used in **neural networks** (or other models with many parameters)



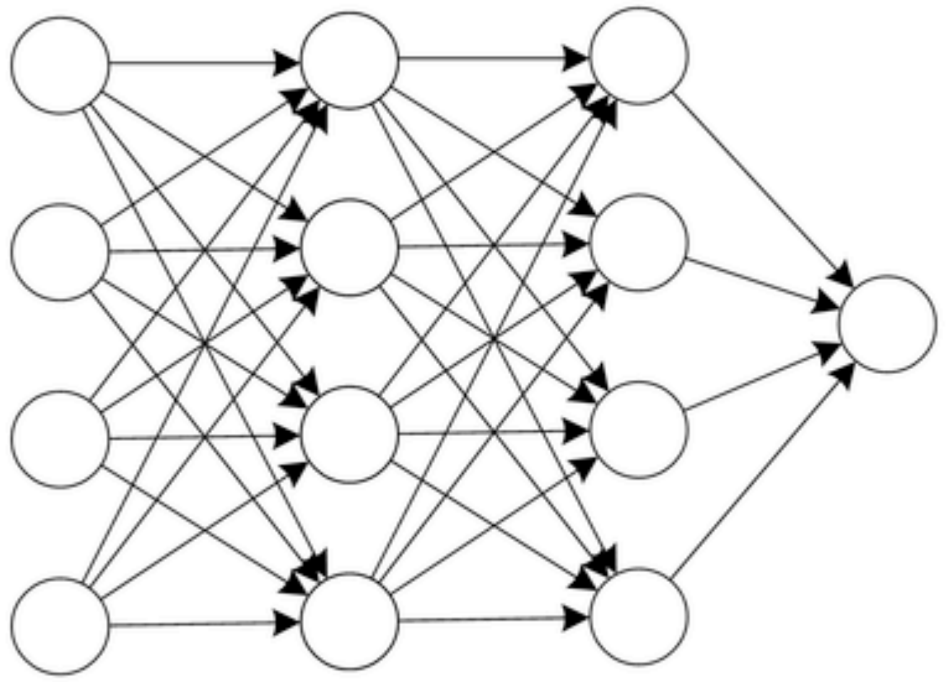
(a) Standard Neural Network



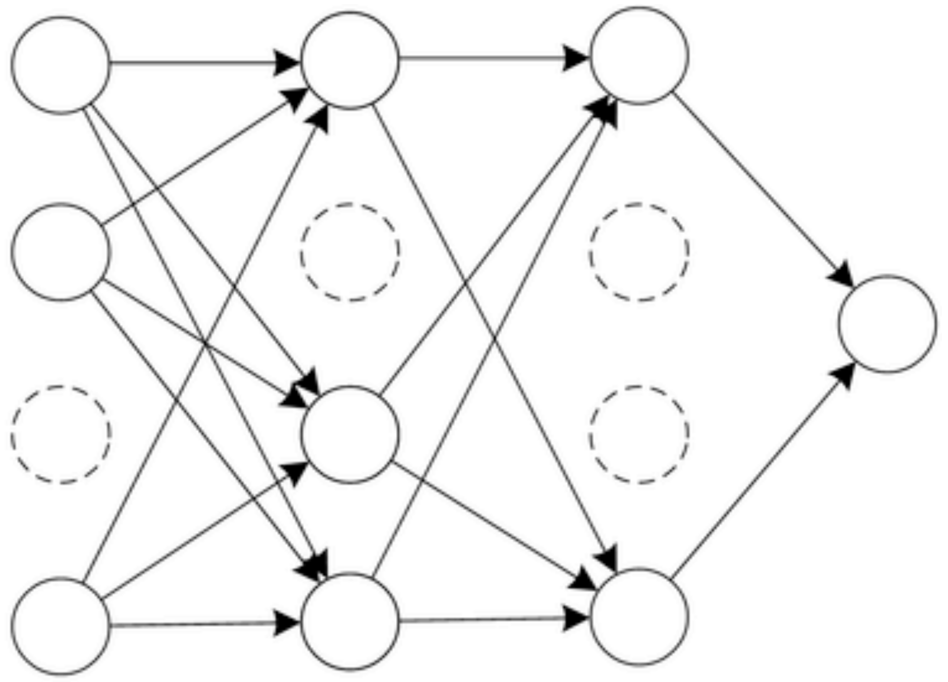
(b) Network after Dropout

Dropout

- Mostly used in **neural networks** (or other models with many parameters)
- During **training** (not evaluation), **randomly** set some **layer outputs to zero**
  - Parameterized by the **proportion** of outputs to drop (e.g. 10%, 20%)



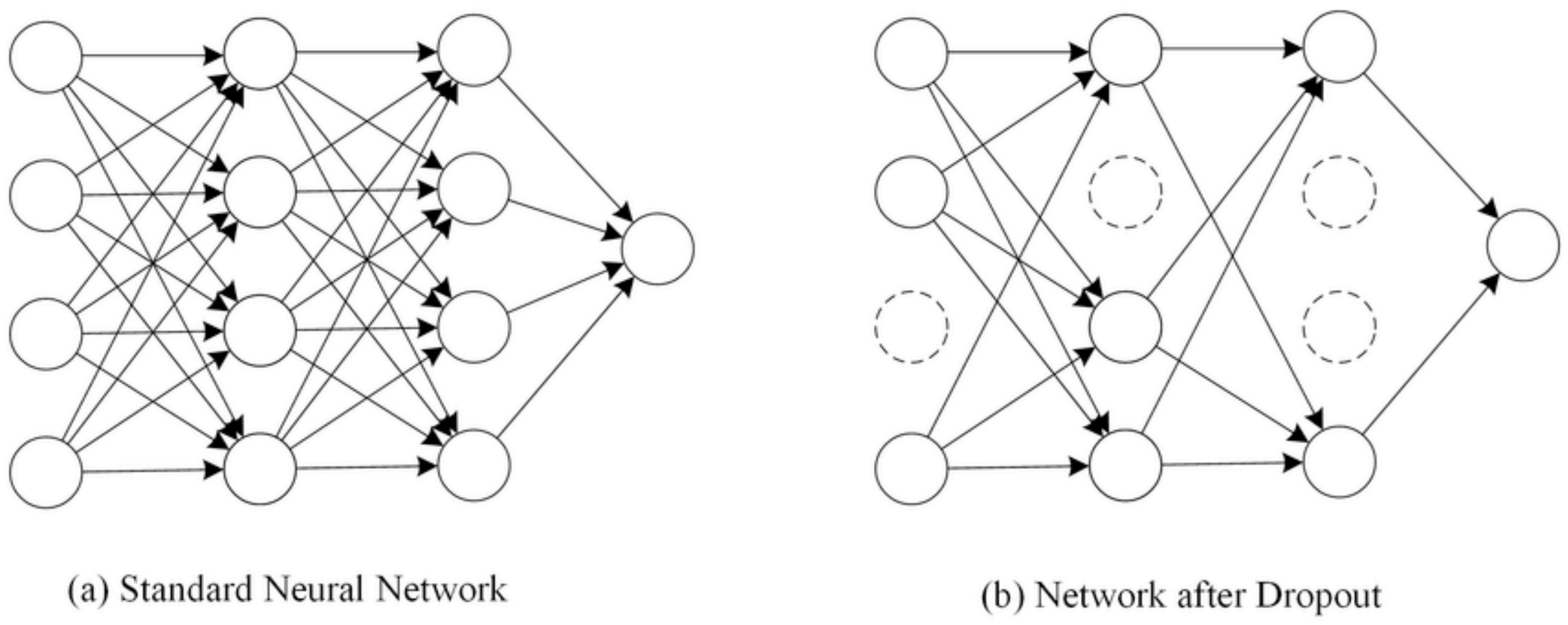
(a) Standard Neural Network



(b) Network after Dropout

Dropout

- Mostly used in **neural networks** (or other models with many parameters)
- During **training** (not evaluation), **randomly** set some **layer outputs to zero**
  - Parameterized by the **proportion** of outputs to drop (e.g. 10%, 20%)
- Forces the network to use **redundant representations**
  - Put another way, avoids **memorizing examples** with **single parameters**



Other Techniques

Other Techniques

- **Early stopping**: stop training when the validation loss **stops decreasing**
  - Simple idea, **almost always used**
  - Often will define a **patience**: i.e. "if my val. loss doesn't decrease for X steps..."

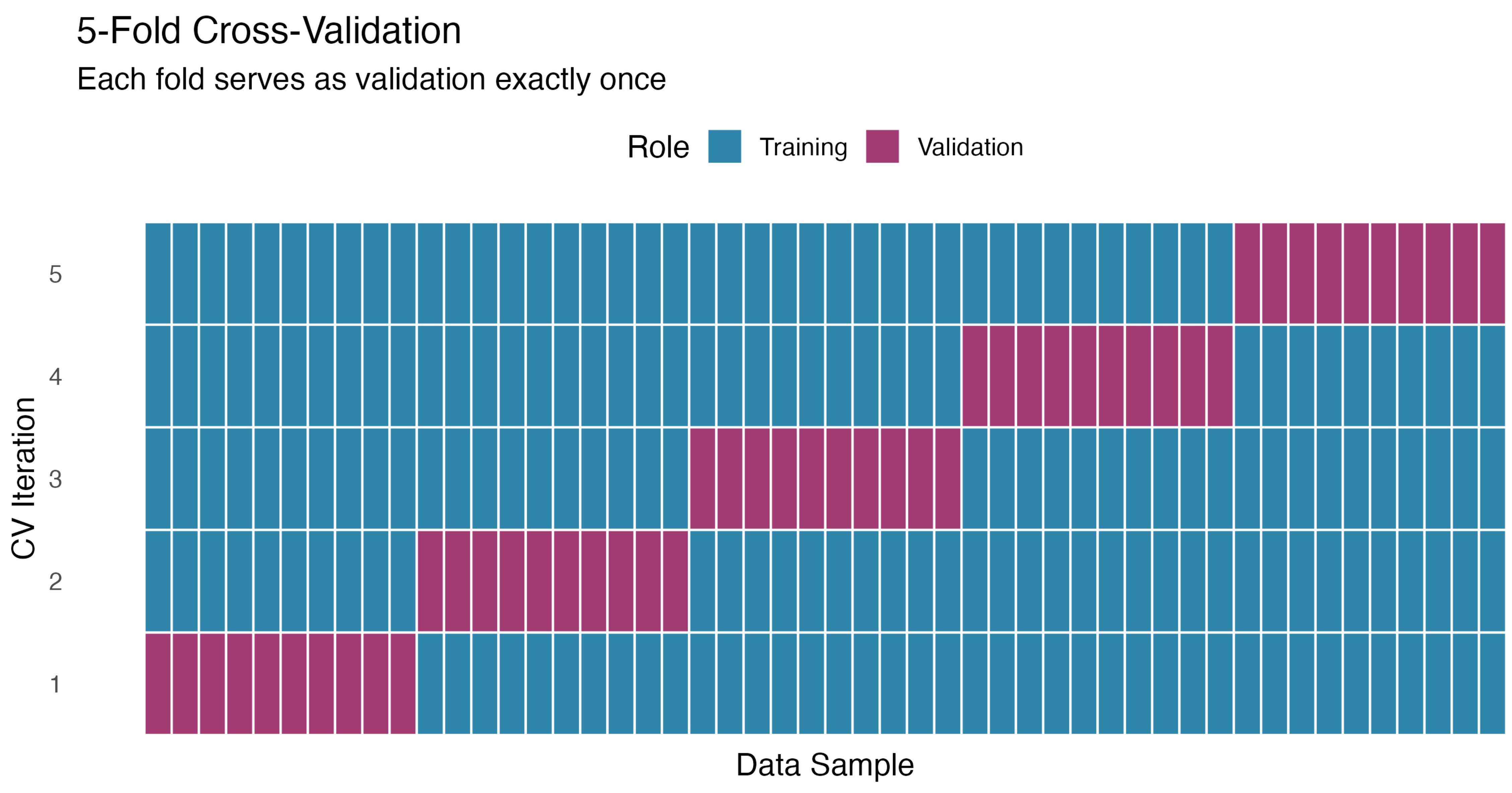
Other Techniques

- **Early stopping**: stop training when the validation loss **stops decreasing**
  - Simple idea, **almost always used**
  - Often will define a **patience**: i.e. "if my val. loss doesn't decrease for X steps..."
- **Model Ensembles**: average the results of multiple models
  - Reduces variance; also kind of the plot of *The Minority Report*

Other Techniques

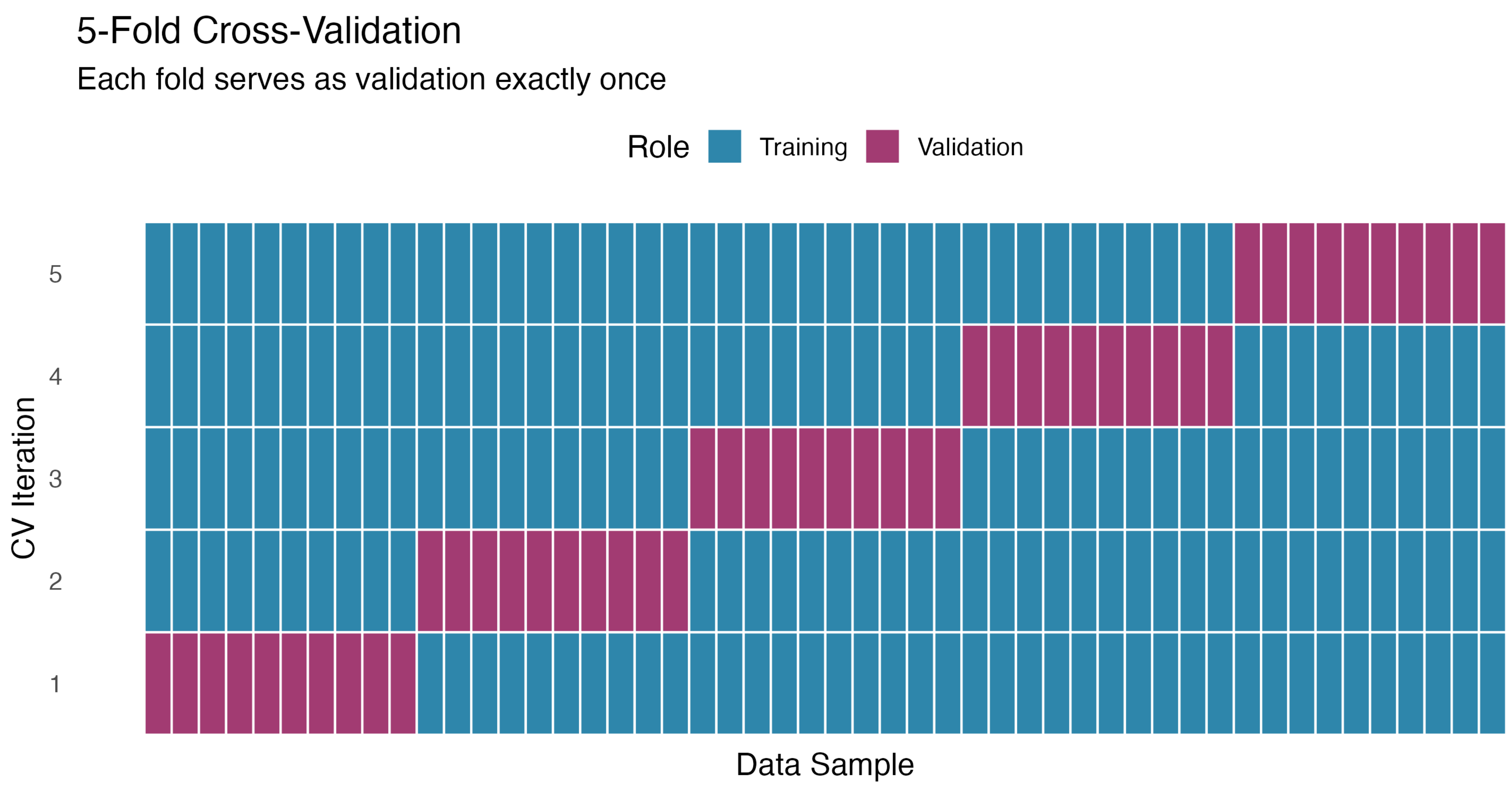
- **Early stopping**: stop training when the validation loss **stops decreasing**
  - Simple idea, **almost always used**
  - Often will define a **patience**: i.e. "if my val. loss doesn't decrease for X steps..."
- **Model Ensembles**: average the results of multiple models
  - Reduces variance; also kind of the plot of *The Minority Report*
- **Data Augmentation**: more later; idea is to **artificially expand** the training set

Final point: Cross-Validation



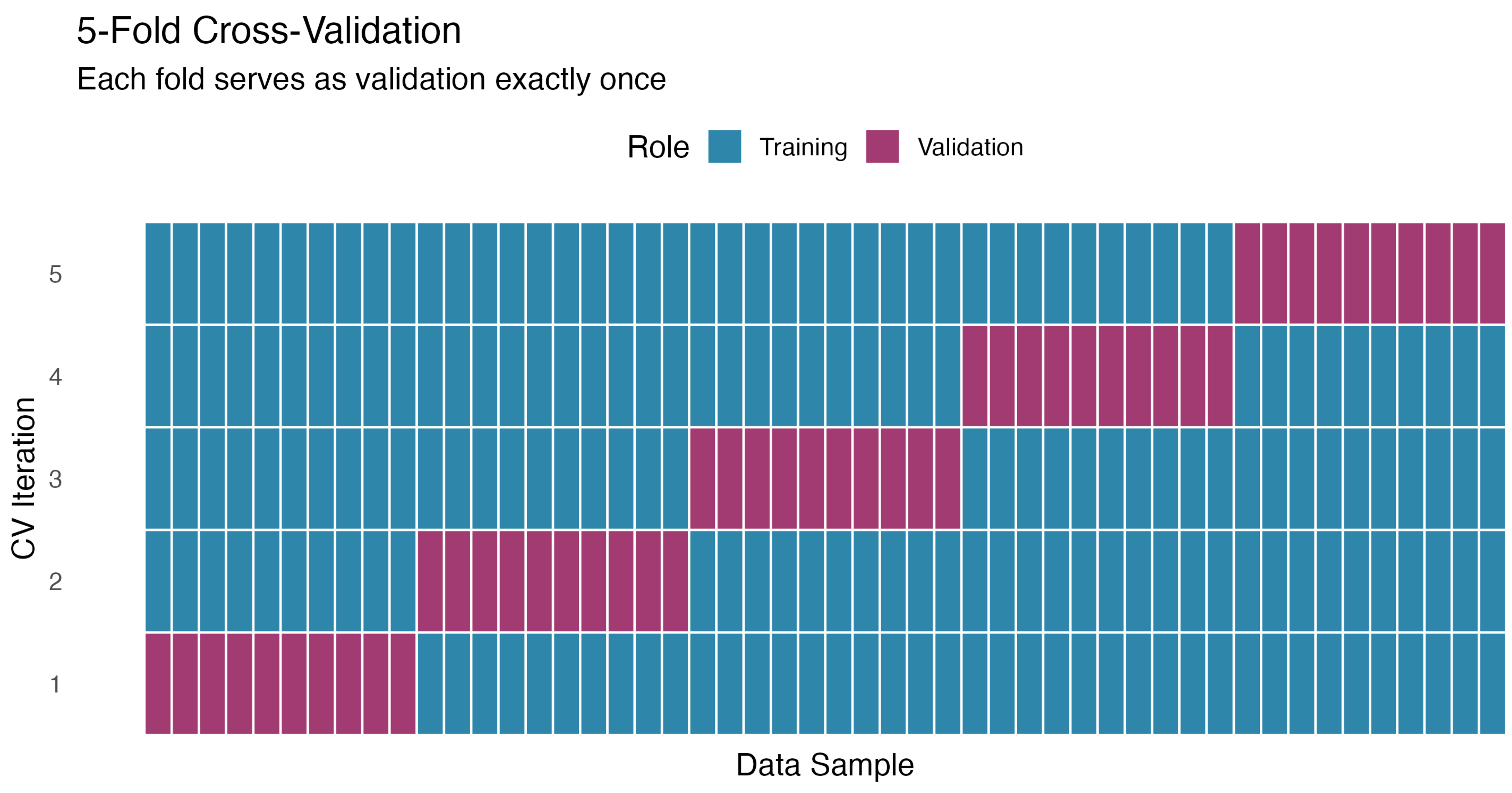
Final point: Cross-Validation

- If your val/test set is very small, won't it be a **noisy estimate?**



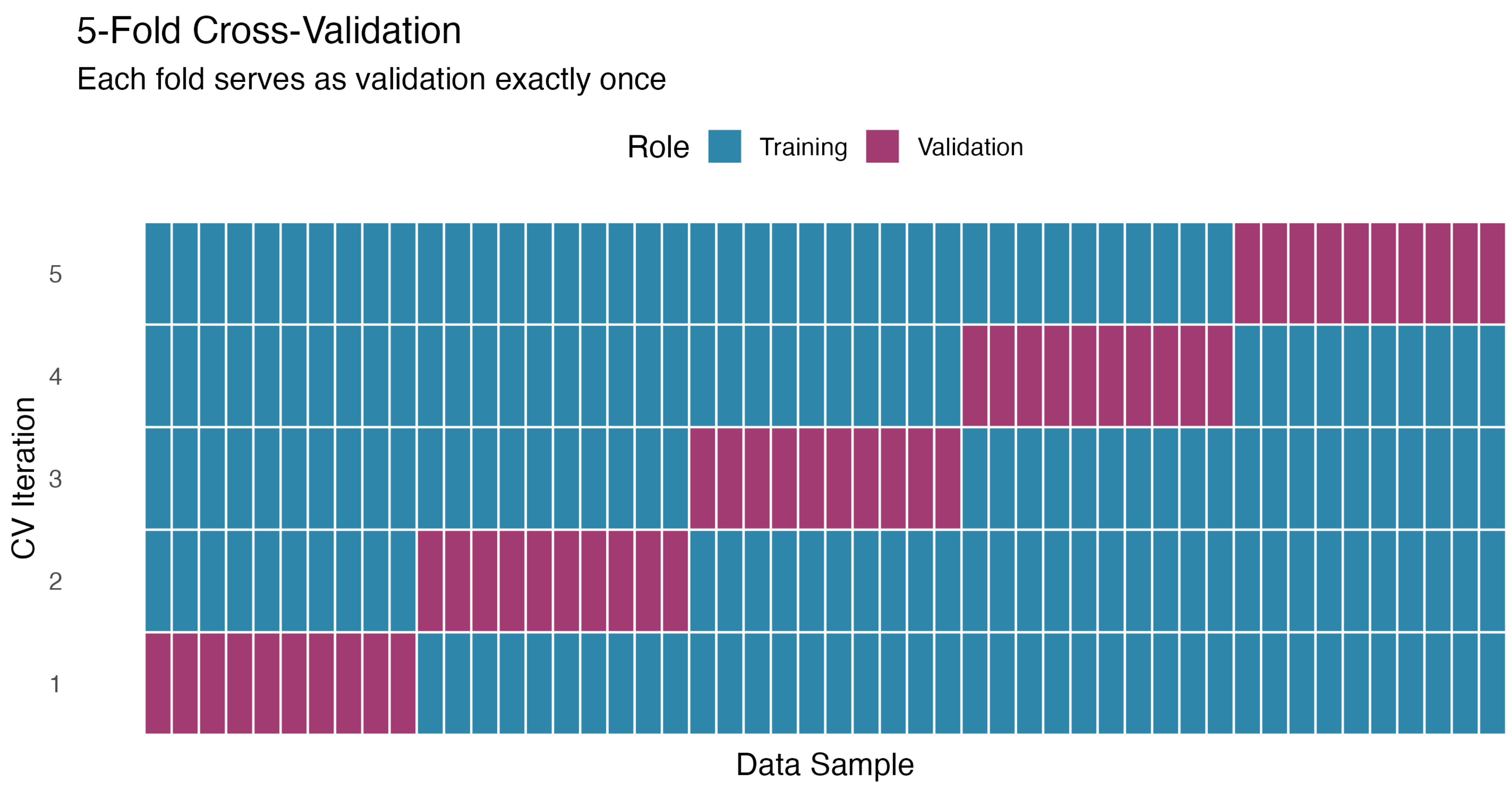
Final point: Cross-Validation

- If your val/test set is very small, won't it be a **noisy estimate?**
- Solution: split into K "folds", use **each** as test in turn
  - I.e. train on **all the other folds**, validate on the held-out fold
  - Do this **K times**, then **average the result**



Final point: Cross-Validation

- If your val/test set is very small, won't it be a **noisy estimate?**
- Solution: split into K "folds", use **each** as test in turn
  - I.e. train on **all the other folds**, validate on the **held-out fold**
  - Do this **K times**, then **average the result**
- Gives a **more reliable estimate** of generalization



5-Fold Cross-Validation

Each fold serves as validation exactly once