Northeastern University
Khoury College of
Computer Sciences

Recap

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

Spring 2026

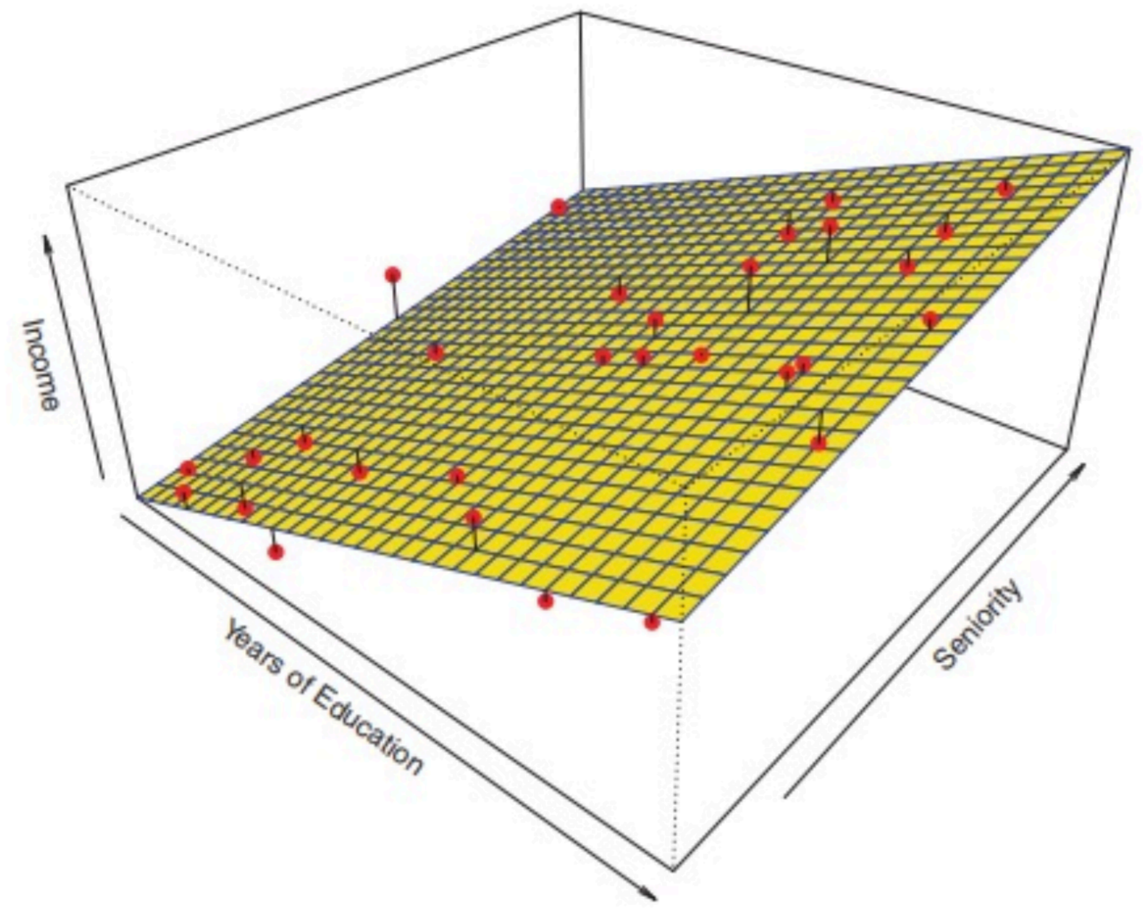
Monday | January 26, 2026

Linear Regression

Linear Regression

Linear Regression

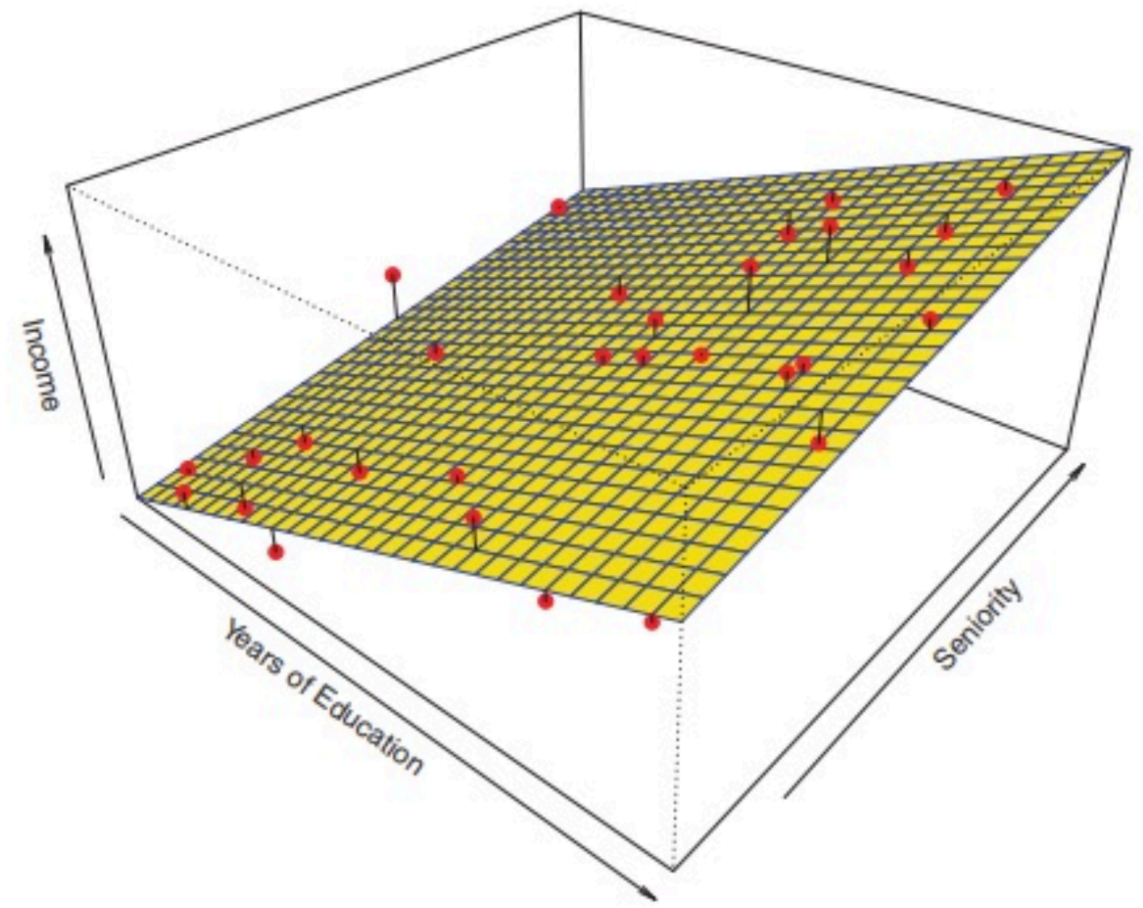
Linear Regression



- Linear Model

$$f_\theta(x) = \theta_0 + \theta_1 x_0 + \theta_2 x_1$$

Linear Regression

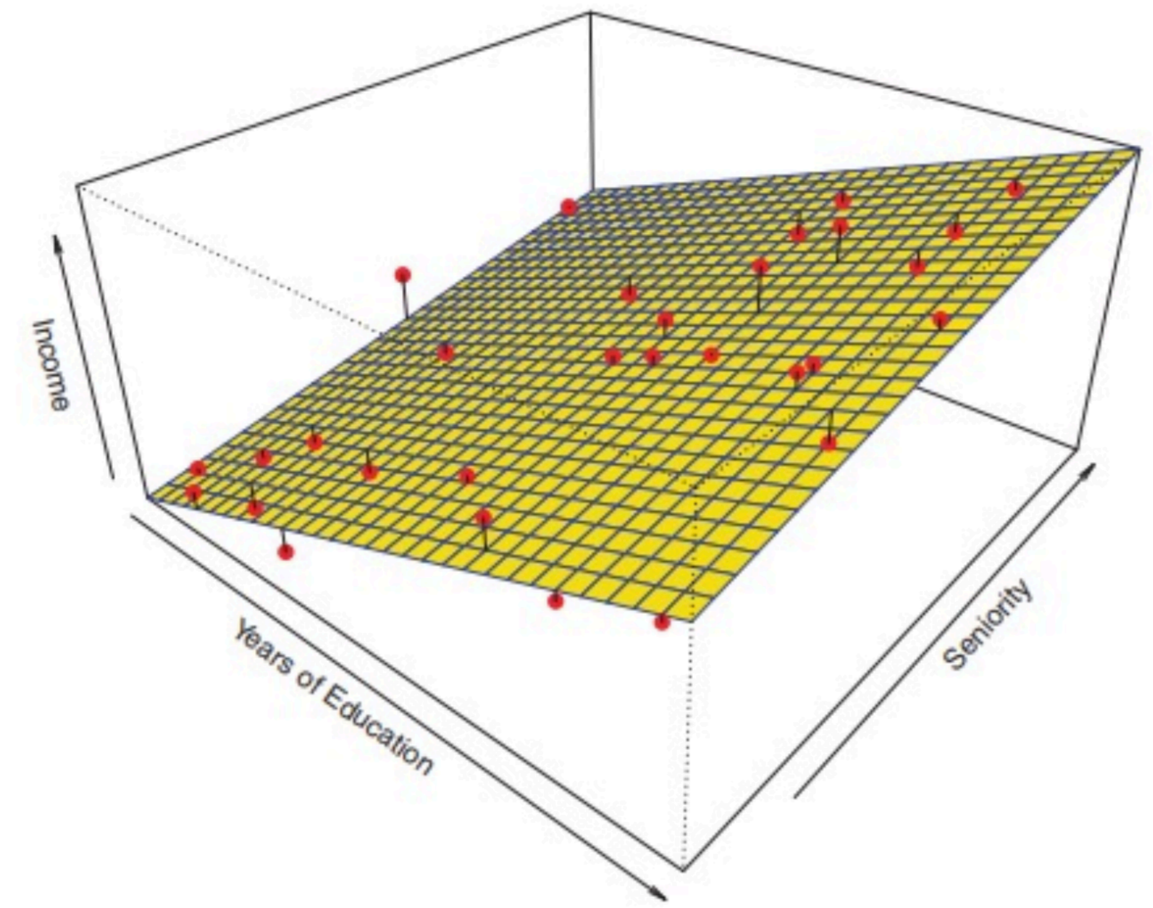


- Linear Model

$$f_{\theta}(x) = \theta_0 + \theta_1 x_0 + \theta_2 x_1$$

Learnable parameters

Linear Regression



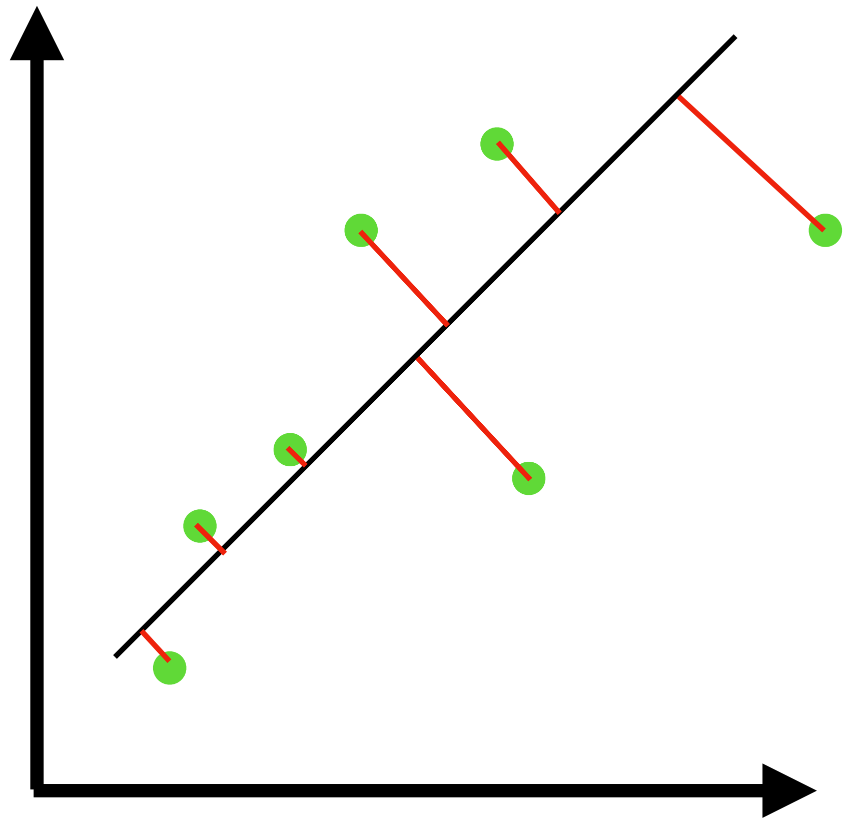
- Linear Model

$$f_\theta(x) = \theta_0 + \theta_1 x_0 + \theta_2 x_1$$

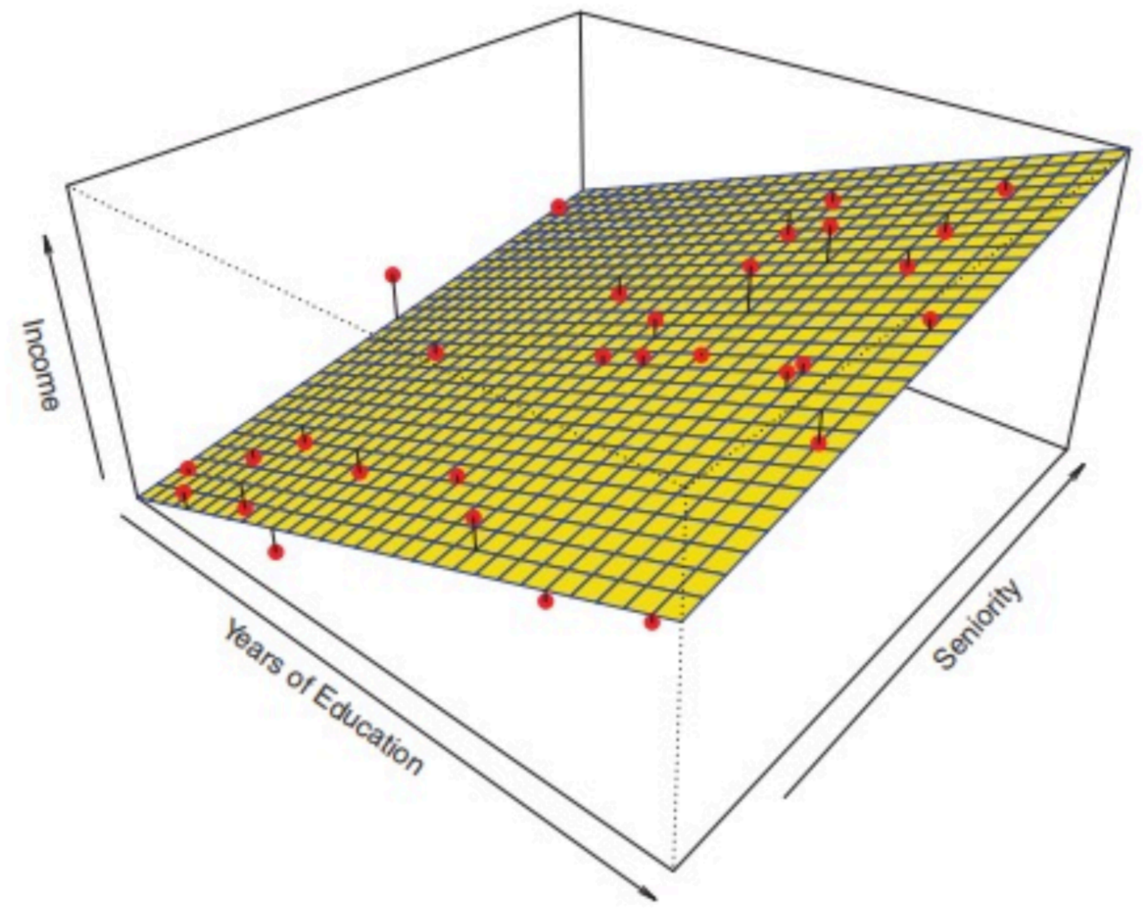
- Loss Functions (also called Cost Functions)

The red lines are called residuals

$$L(\theta) = \frac{1}{m} \sum_{i=1}^{m} [f_\theta(x_i) - y_i]^2$$ - Mean Squared Error



Linear Regression



- Linear Model

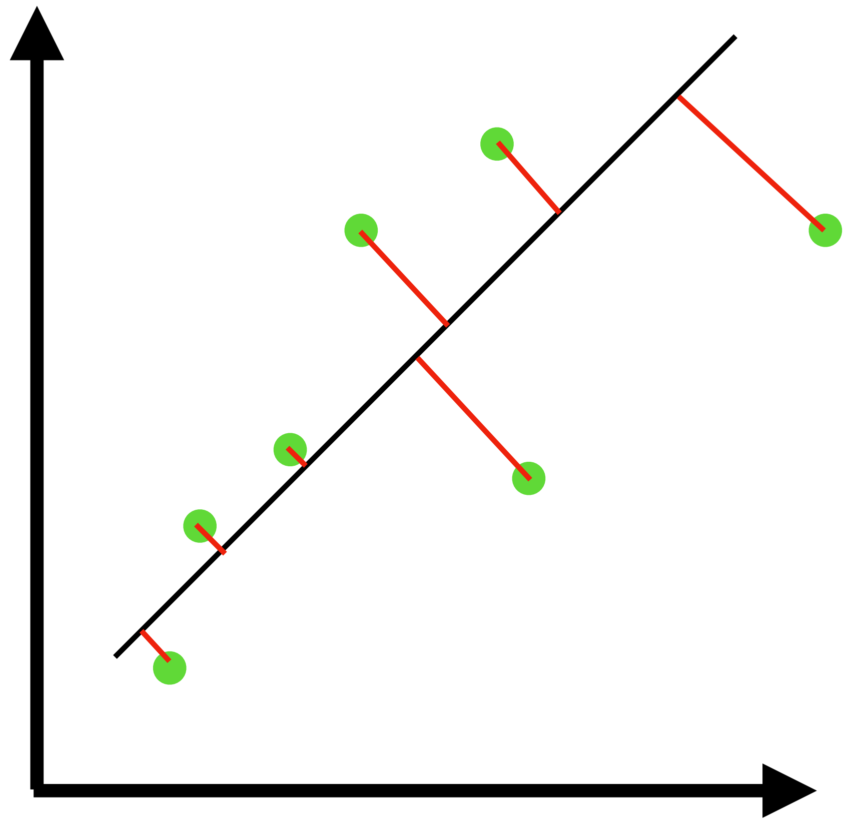
$$f_\theta(x) = \theta_0 + \theta_1 x_0 + \theta_2 x_1$$

- Loss Functions (also called Cost Functions)

The red lines are called residuals

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}[f_\theta(x_i) - y_i]^2$$ - Mean Squared Error

$$L(\theta) = \sum_{i=1}^{m}[f_\theta(x_i) - y_i]^2$$ - Residual Sum of Squares



Linear Regression

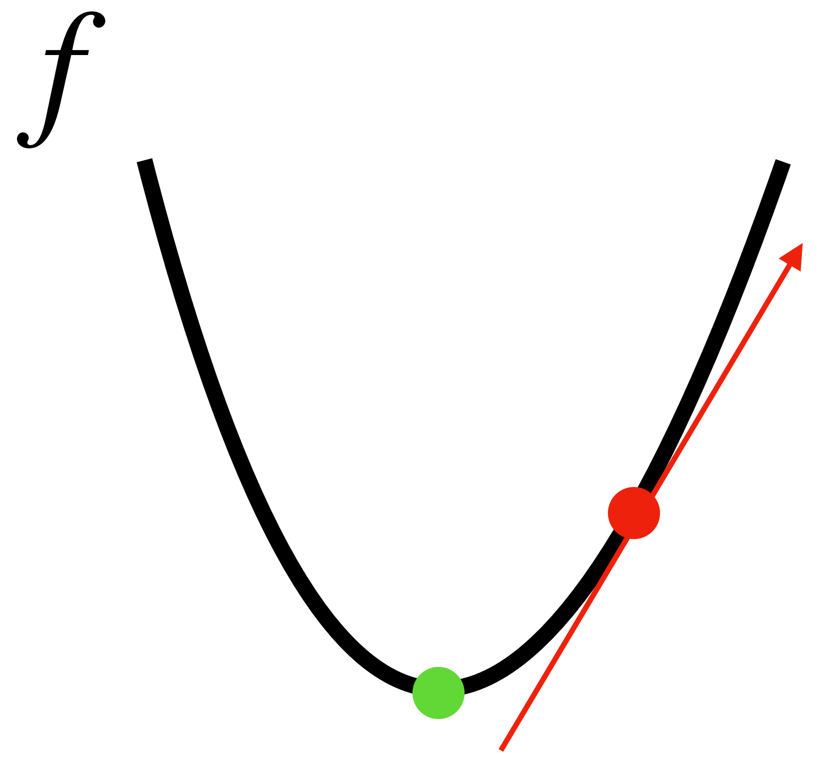
- Linear Model

$$f_\theta(x) = \theta_0 + \theta_1 x$$

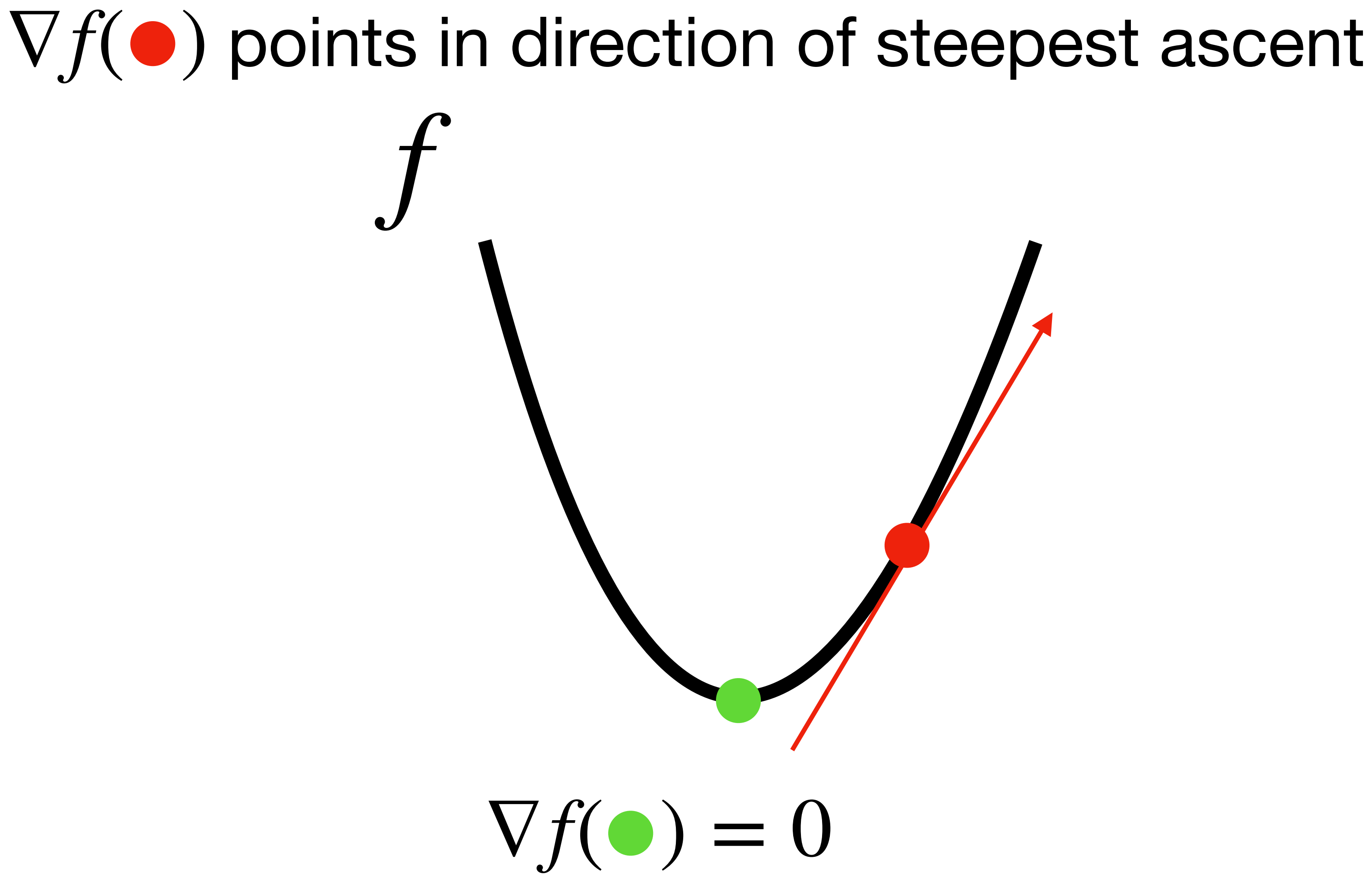
- How do we find the solution to this? How do we find the optimal θ?
  - We optimize θ to minimize the loss function

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}[f_\theta(x_i) - y_i]^2$$

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}[\theta_0 + \theta_1 \cdot x - y_i]^2$$



Linear Regression



Linear Regression

- How do we find the solution to this? How do we find the optimal θ?
    - We optimize θ to minimize the loss function

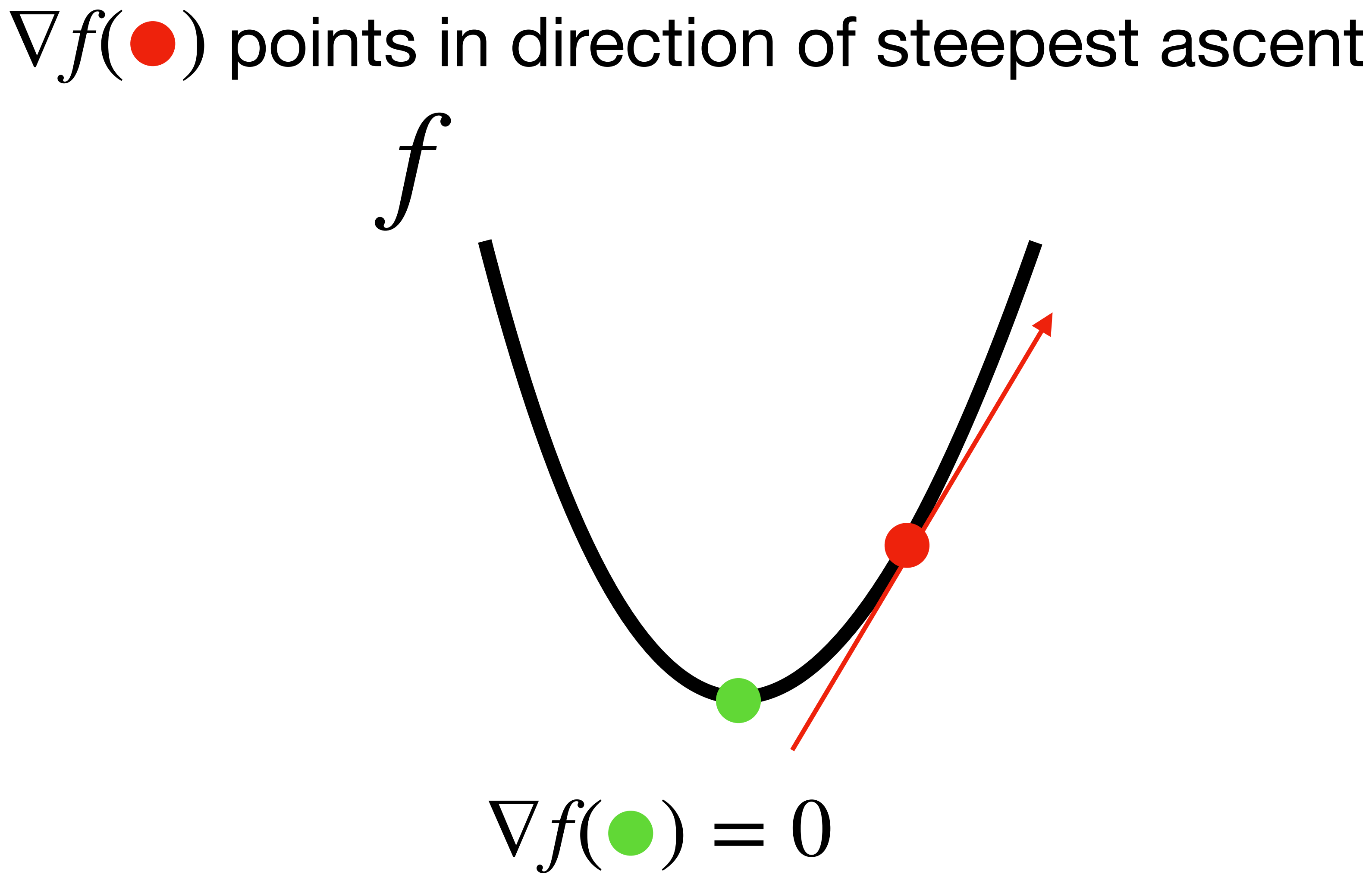
$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}[f_\theta(x_i) - y_i]^2$$

$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}[\theta_0 + \theta_1 \cdot x_i - y_i]^2$$

Find the point where $\nabla L(\theta) = 0$

$$\frac{\partial L(\theta)}{\partial \theta_0} = \frac{2}{m}\sum_{i=1}^{m}(\theta_0 + \theta_1 x_i - y_i) = 0$$

$$\frac{\partial L(\theta)}{\partial \theta_1} = \frac{2}{m}\sum_{i=1}^{m}x_i \cdot (\theta_0 + \theta_1 x_i - y_i) = 0$$



Linear Regression

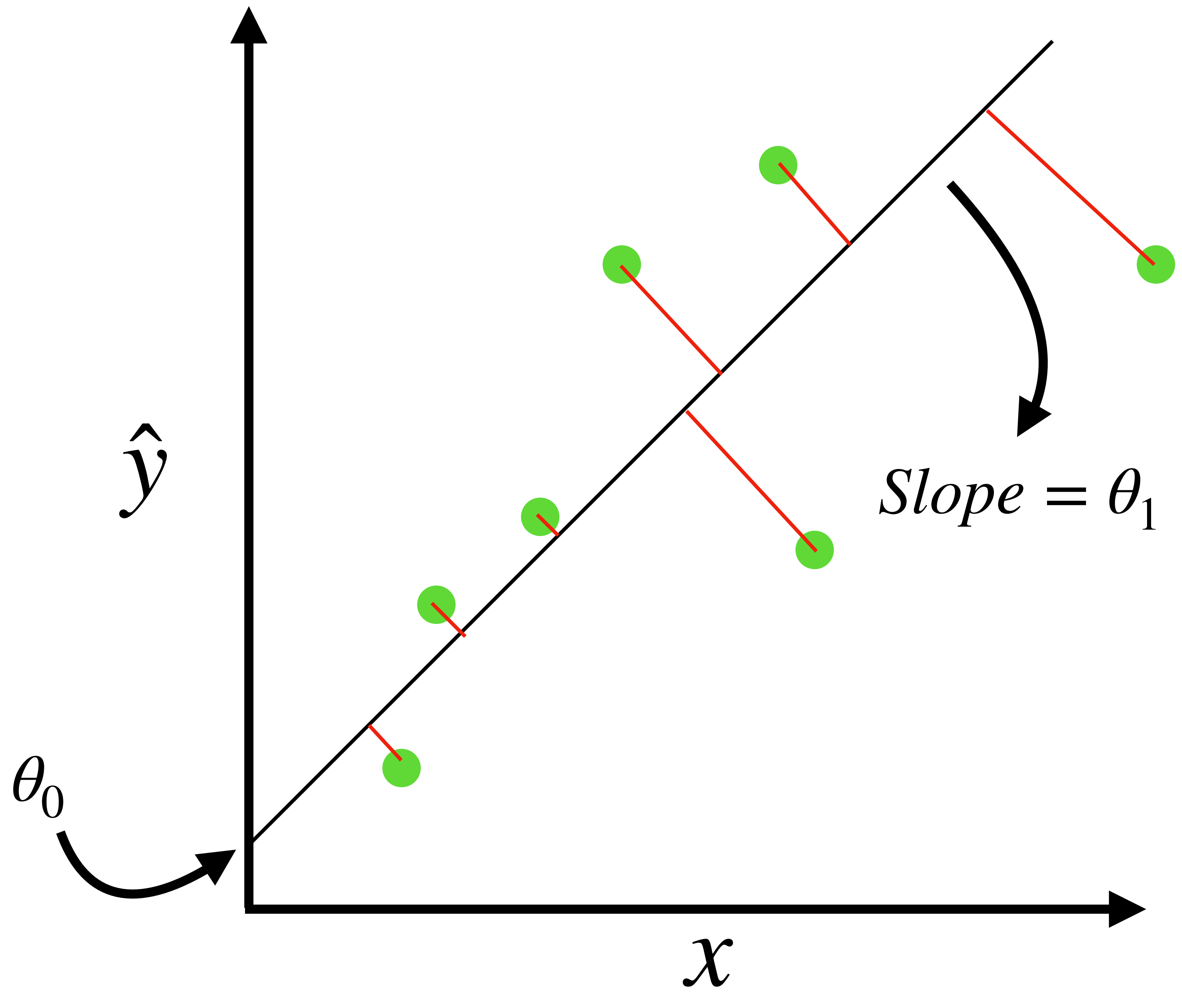
$$\theta_0 = \bar{y} - \theta_1 \bar{x}$$

$$\theta_1 = \frac{Cov(x, y)}{Var(x)}$$

The slope $\theta_1 = \dfrac{Cov(x, y)}{Var(x)}$ makes sense:

- If x and y covary strongly (move together), the slope is steeper
- If x has high variance (spread out), the slope is gentler
- The **sign** of covariance determines if the line goes up or down

$$f_\theta(x) = \theta_0 + \theta_1 x$$



Linear Regression

Solutions in Matrix Form

Linear Regression

Solutions in Matrix Form

- Let's look at the matrix formulation of the same problem

$$L(\theta) = \frac{1}{m}\sum_i (y_i - \hat{y}_i)^2$$

But in matrix form, $f_\theta(x) = \hat{Y} = X\theta$, where $X \in \mathbb{R}^{m \times d}$ has m rows of data and d columns of features and $\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix} \in \mathbb{R}^{d \times 1}$

$$L(\theta) = \frac{1}{m}\sum (Y - X\theta)^2$$

(think back to system of equations for why this is true)

Quick Recap

Systems of Linear Equations - Linear Regression Example

- Consider the equation $y = w_0 x_0 + w_1 x_1$

y Price	x_0 # Rooms	x_1 Sq. Ft.
2000	1	450
2100	1	510
2400	2	980
3000	3	1500

$$(1) \cdot w_0 + (450) \cdot w_1 = 2000$$

$$(1) \cdot w_0 + (510) \cdot w_1 = 2100$$

$$(2) \cdot w_0 + (980) \cdot w_1 = 2400$$

$$(3) \cdot w_0 + (1500) \cdot w_1 = 3000$$

$$\longrightarrow$$

$X \in \mathbb{R}^{4\times2}$ $\quad W \in \mathbb{R}^{2\times1}$ $\quad y \in \mathbb{R}^{4\times1}$

$$\begin{bmatrix} 1 & 450 \\ 1 & 510 \\ 2 & 980 \\ 3 & 1500 \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \end{bmatrix} = \begin{bmatrix} 2000 \\ 2100 \\ 2400 \\ 3000 \end{bmatrix}$$

Linear Regression
Solution

We want to find the minimum so set gradient to zero

$$\nabla L(\theta) = -2X^TY + 2X^TX\theta = 0$$

$$2X^TX\theta = 2X^TY$$

$$X^TX\theta = X^TY$$

If X^TX is invertible, then

$$\theta = (X^TX)^{-1}X^TY$$

Practical Issues in Linear Regression

Multicollinearity

- When two features are highly correlated or are linearly dependent on each other

Practical Issues in Linear Regression

Multicollinearity

- When two features are highly correlated or are linearly dependent on each other
- Why it's a problem: $\theta = (X^T X)^{-1} X^T Y$
    - $X^T X$ becomes nearly singular (ill-conditioned)
    - Small changes in data cause huge changes in coefficients
    - Coefficients become unreliable and hard to interpret
    - Standard errors blow up

Practical Issues in Linear Regression

Multicollinearity

- When two features are highly correlated or are linearly dependent on each other
- Why it's a problem: $\theta = (X^T X)^{-1} X^T Y$

Simple Detection:
If correlation between features ≥ 0.8

  - $X^T X$ becomes nearly singular (ill-conditioned)
  - Small changes in data cause huge changes in coefficients
  - Coefficients become unreliable and hard to interpret
  - Standard errors blow up

Practical Issues in Linear Regression

Quick Aside

$$\theta = \boxed{(X^T X)^{-1}} X^T Y$$

When else is this not going to be invertible?

$$X \in \mathbb{R}^{m \times n}$$

m: Number of training examples

n: Number of parameters in the model

Practical Issues in Linear Regression

Quick Aside

$$X \in \mathbb{R}^{m \times n}$$

$$\theta = (X^T X)^{-1} X^T Y$$

m: Number of training examples
n: Number of parameters in the model
$rank(X) = min(m, n)$

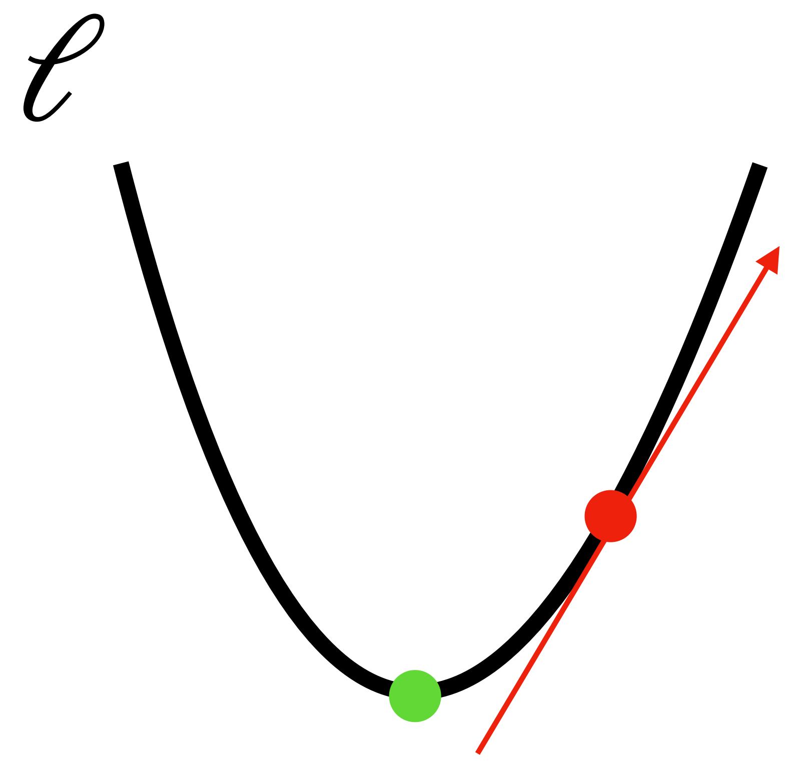
When else is this not going to be invertible?

If $m < n$, then $rank(X) \leq m$, so need **more** data points than number of parameters to get a unique set of parameters

Gradient Descent: Optimizing Loss Functions

- For any loss function $\ell(\theta)$
  - To find minimum, set $\nabla \ell = 0$ and solve for θ

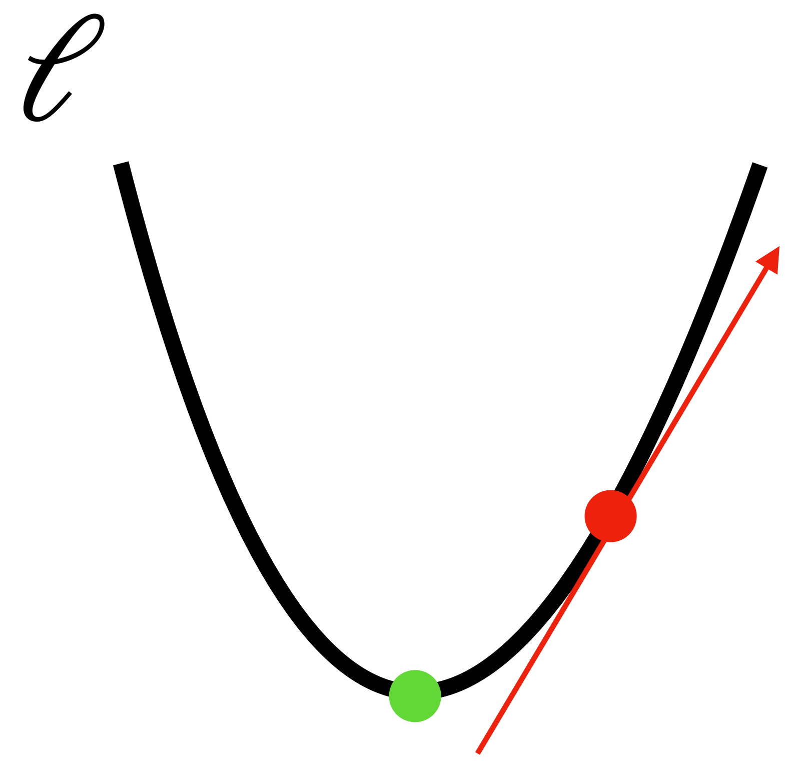
$\nabla \ell(\bullet)$ points in direction of steepest ascent



$\nabla \ell(\bullet) = 0$

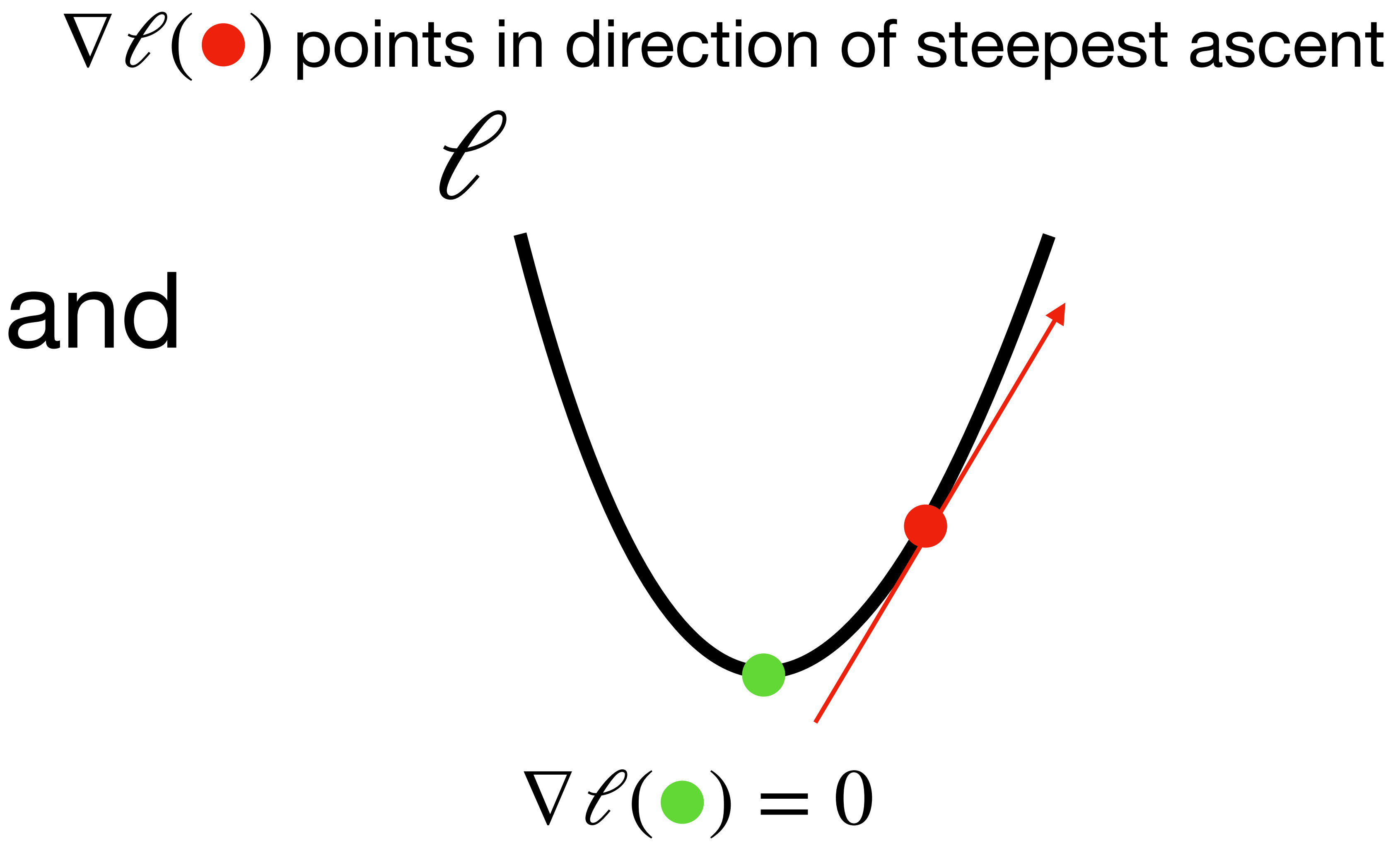
Optimizing Loss Functions

- For any loss function $\ell(\theta)$
  - To find minimum, set $\nabla \ell = 0$ and solve for θ
  - This is called the **closed form solution**
  - But it's not always possible to find closed form solutions, especially when there are a large number of parameters
  - Inverting a matrix is a costly operation - most common methods have complexity $O(n^3)$



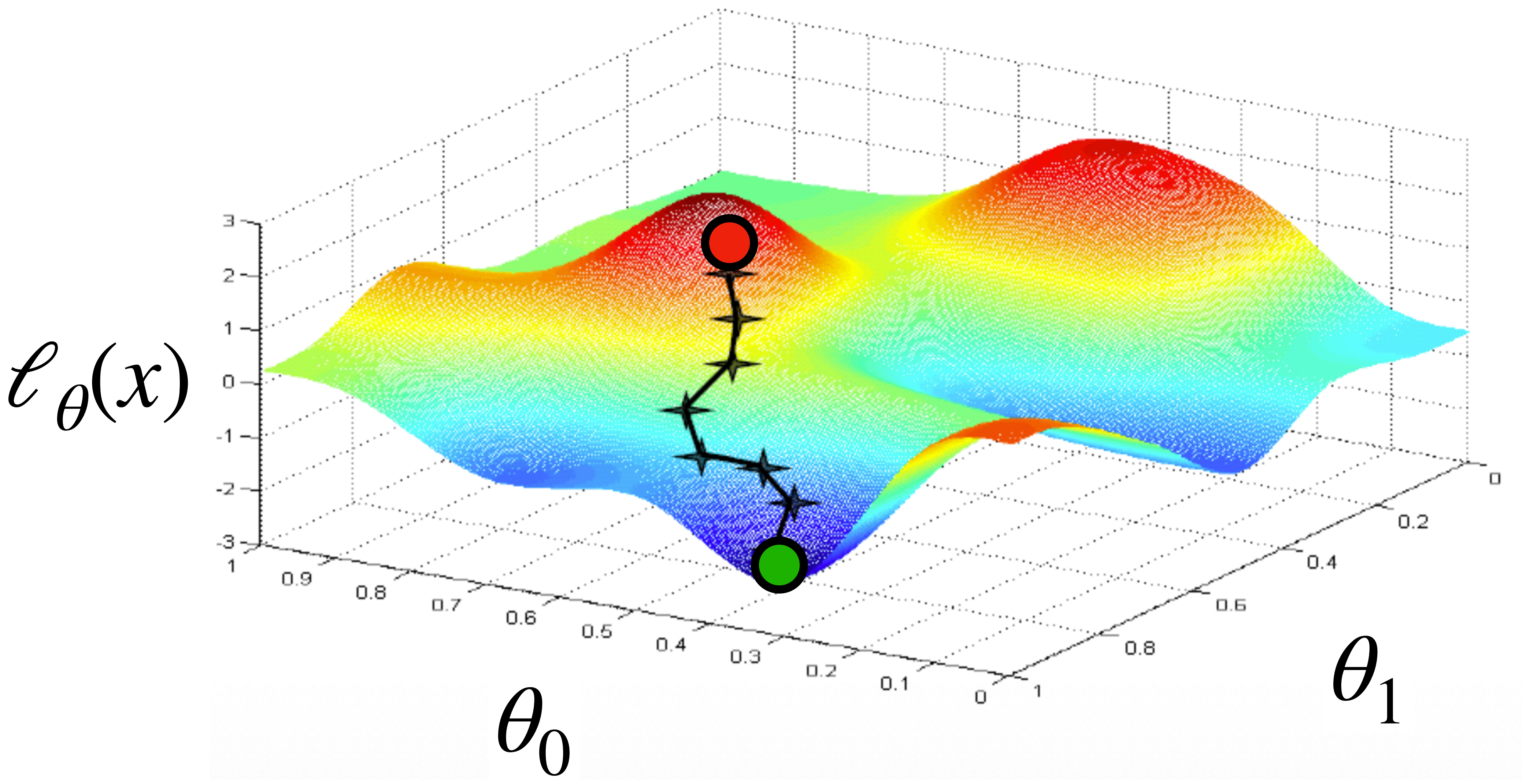
Optimizing Loss Functions

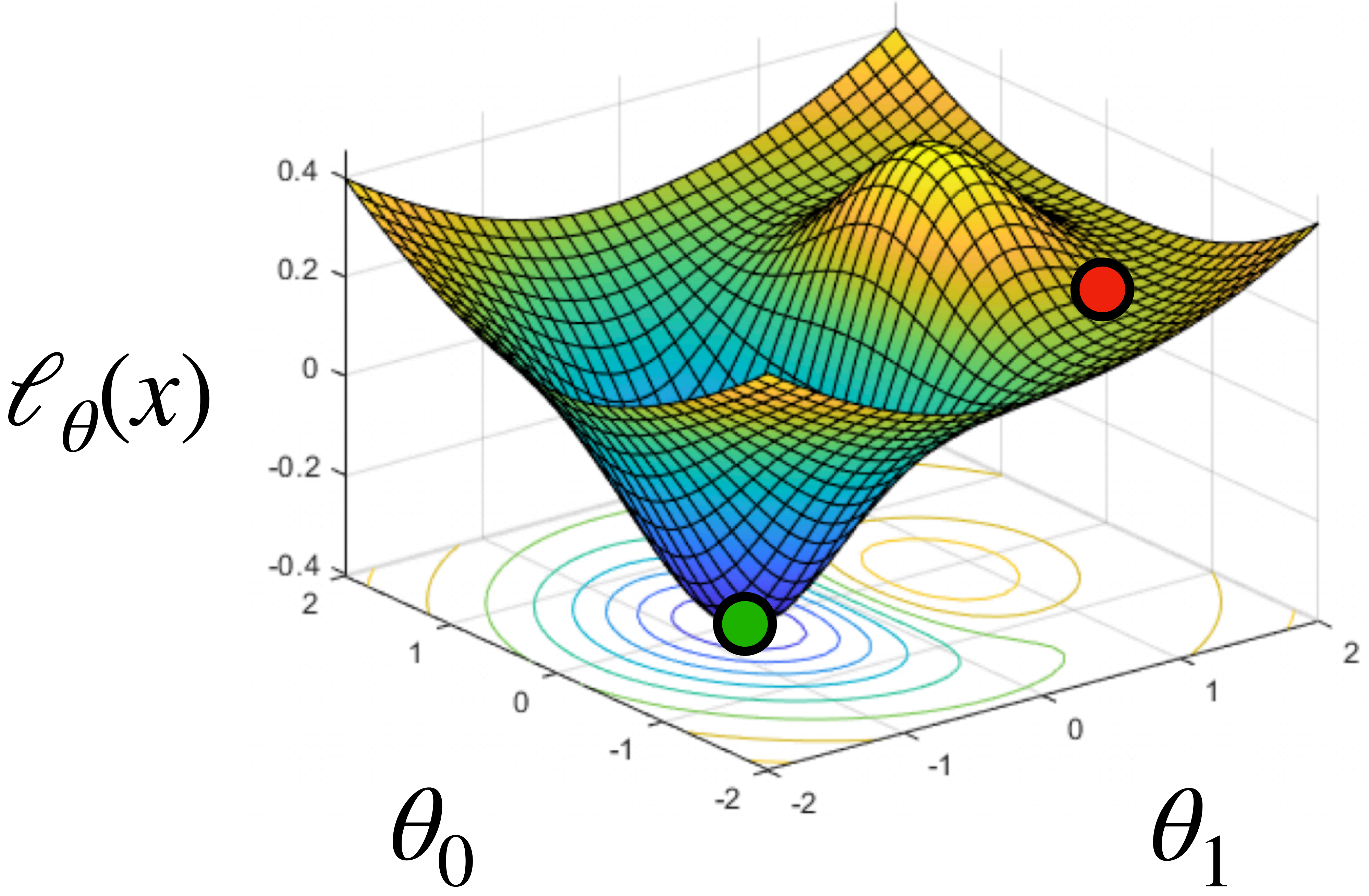
- This is where Gradient Descent comes in
  - Practical and efficient - has $O(mTn)$ where m is number of training points, T is number of epochs and n is number of features
  - Generally applicable to different loss functions
  - Convergence guarantees for certain types of loss functions (e.g., convex functions)



Optimizing Loss Functions

- What does the loss landscape look like with multiple learnable parameters?



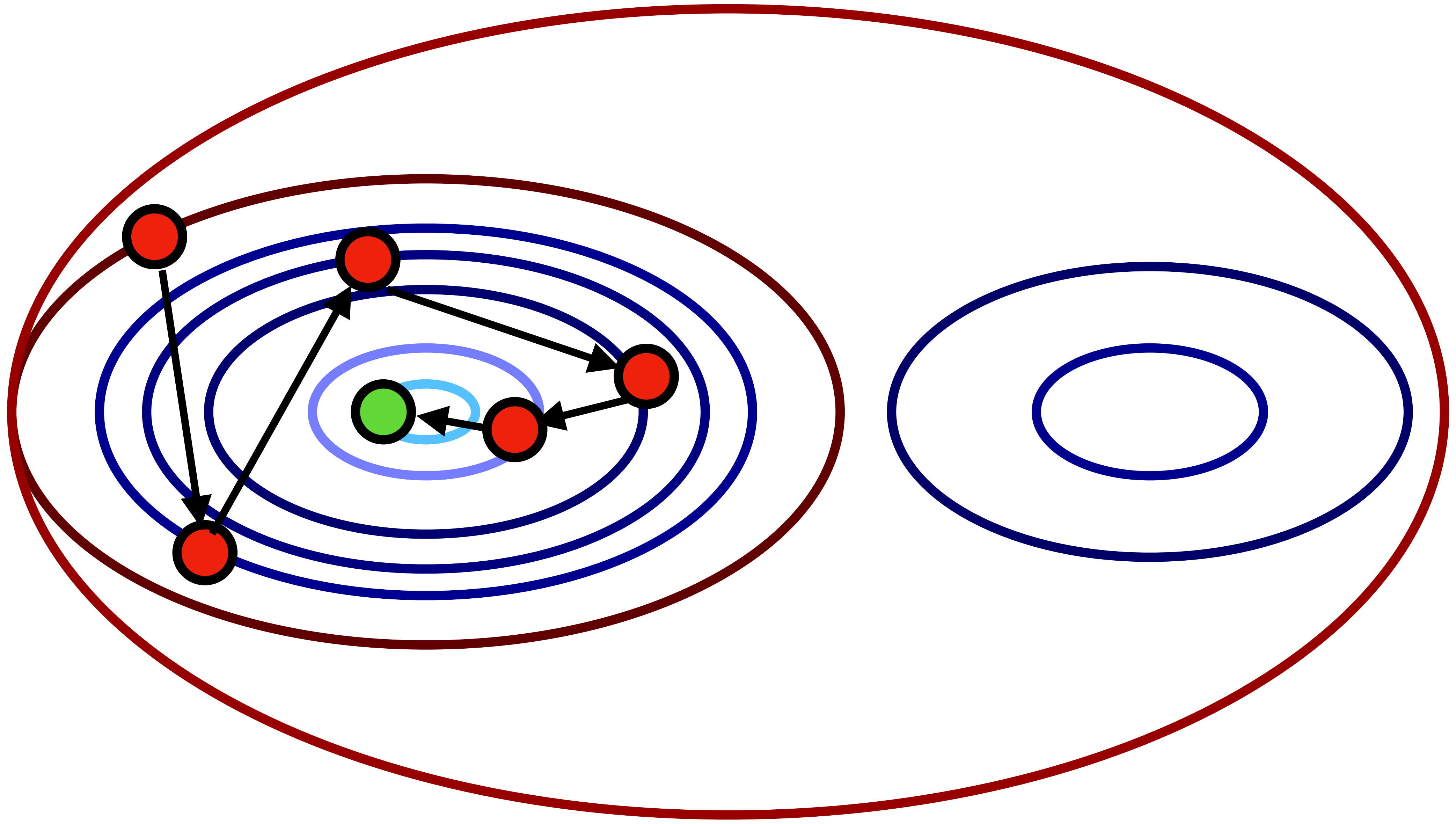


Optimizing Loss Functions

Optimizing Loss Functions

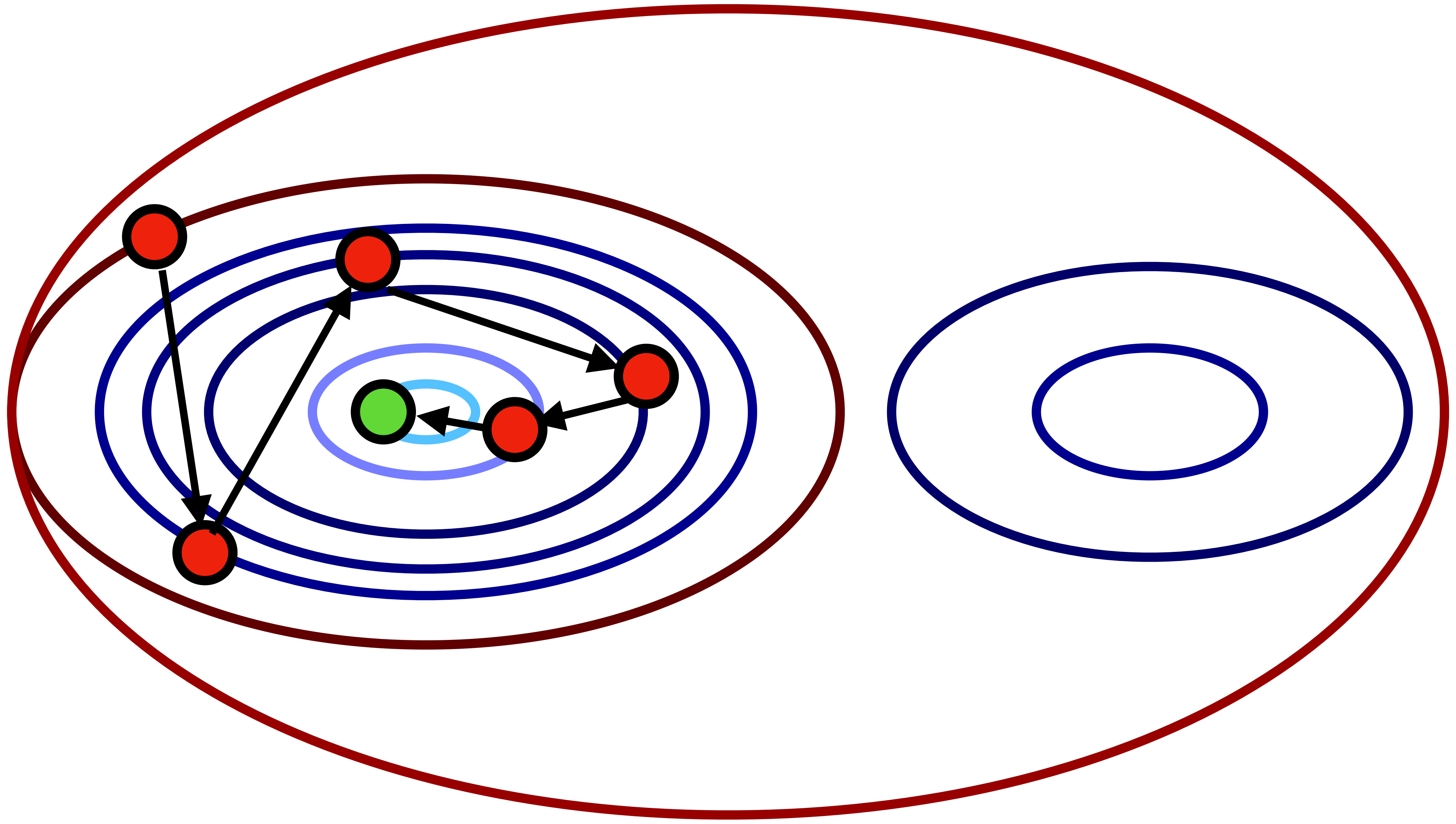
Gradient Descent - Formulation

$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$



Optimizing Loss Functions

Gradient Descent - Formulation

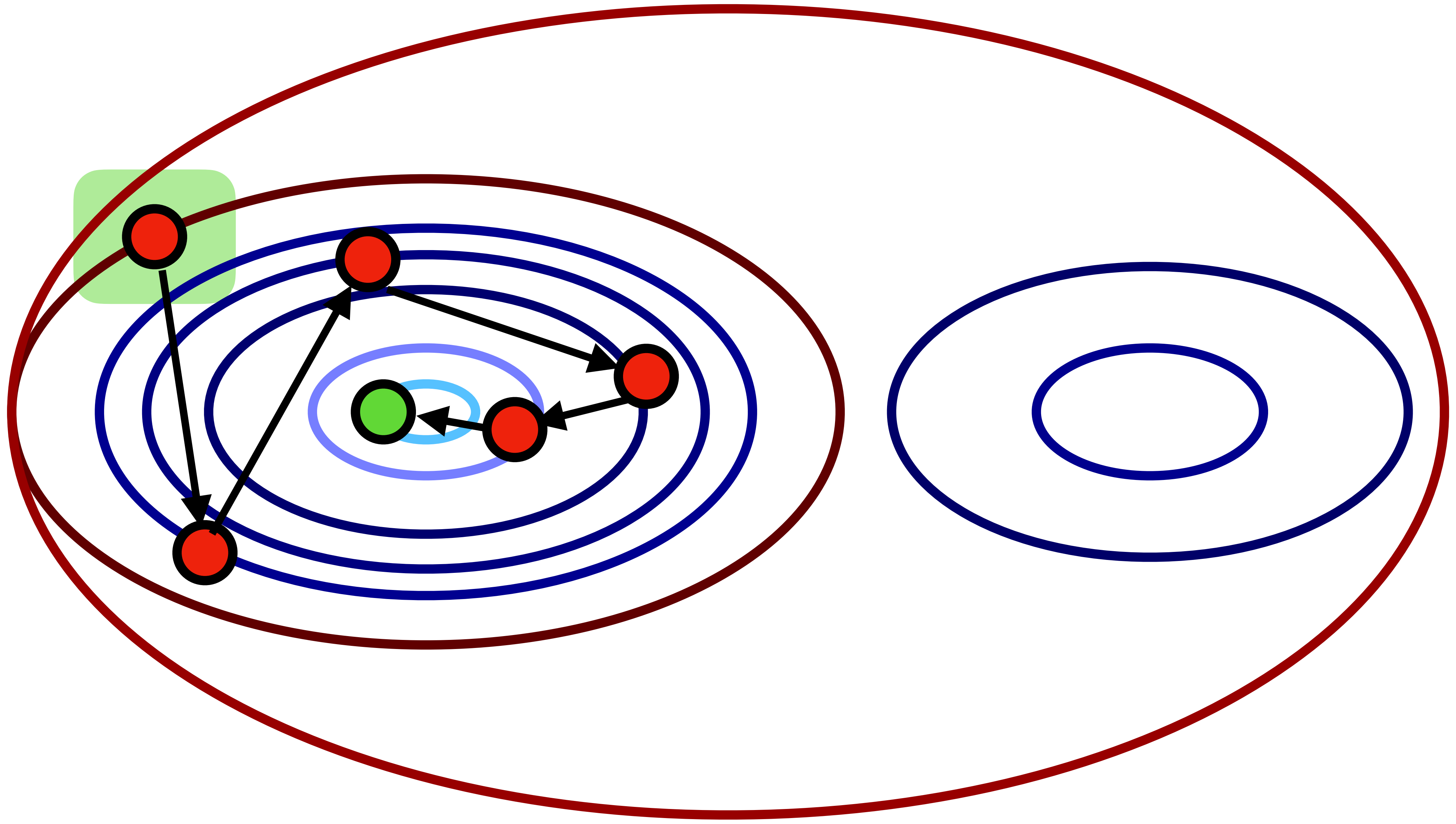


$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

Optimizing Loss Functions

Gradient Descent - Formulation



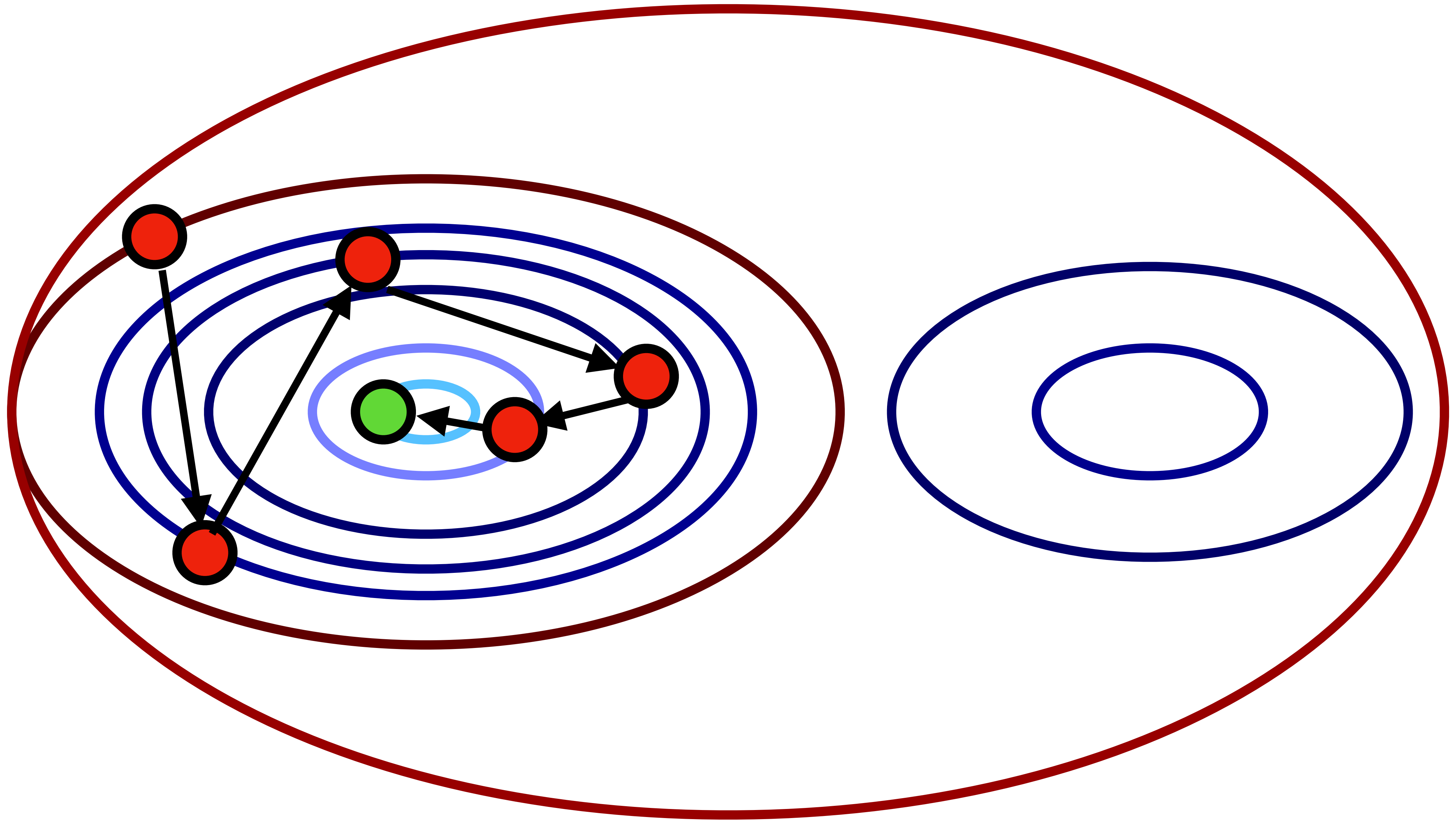
$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

This is going to be your "starting point" on the loss landscape

Optimizing Loss Functions

Gradient Descent - Formulation



$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

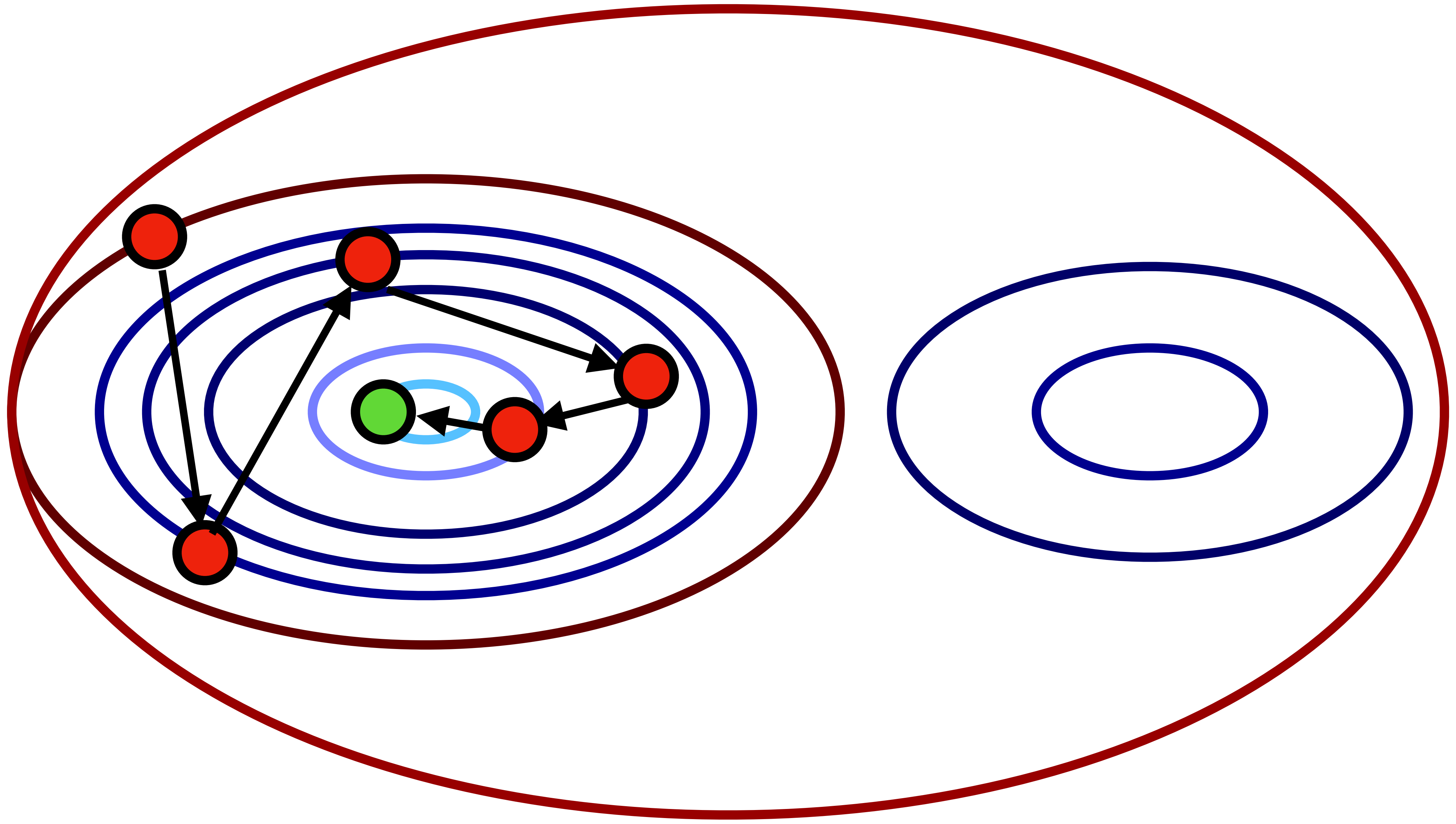
Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

Optimizing Loss Functions

Gradient Descent - Formulation



$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

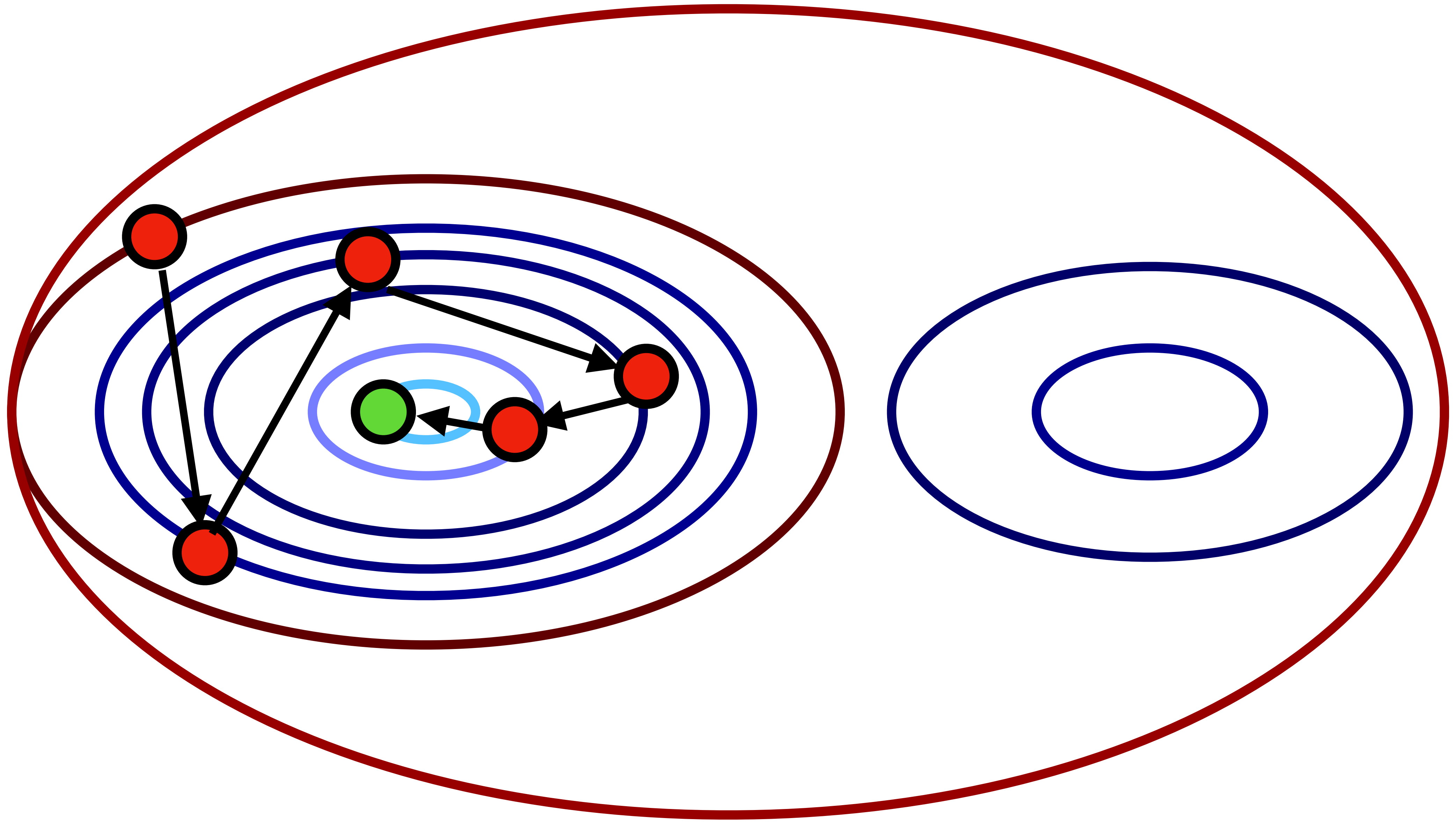
Step 2: Repeat Until Convergence

$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

Negative of partial derivative points in the direction of steepest descent

Optimizing Loss Functions

Gradient Descent - Formulation



$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

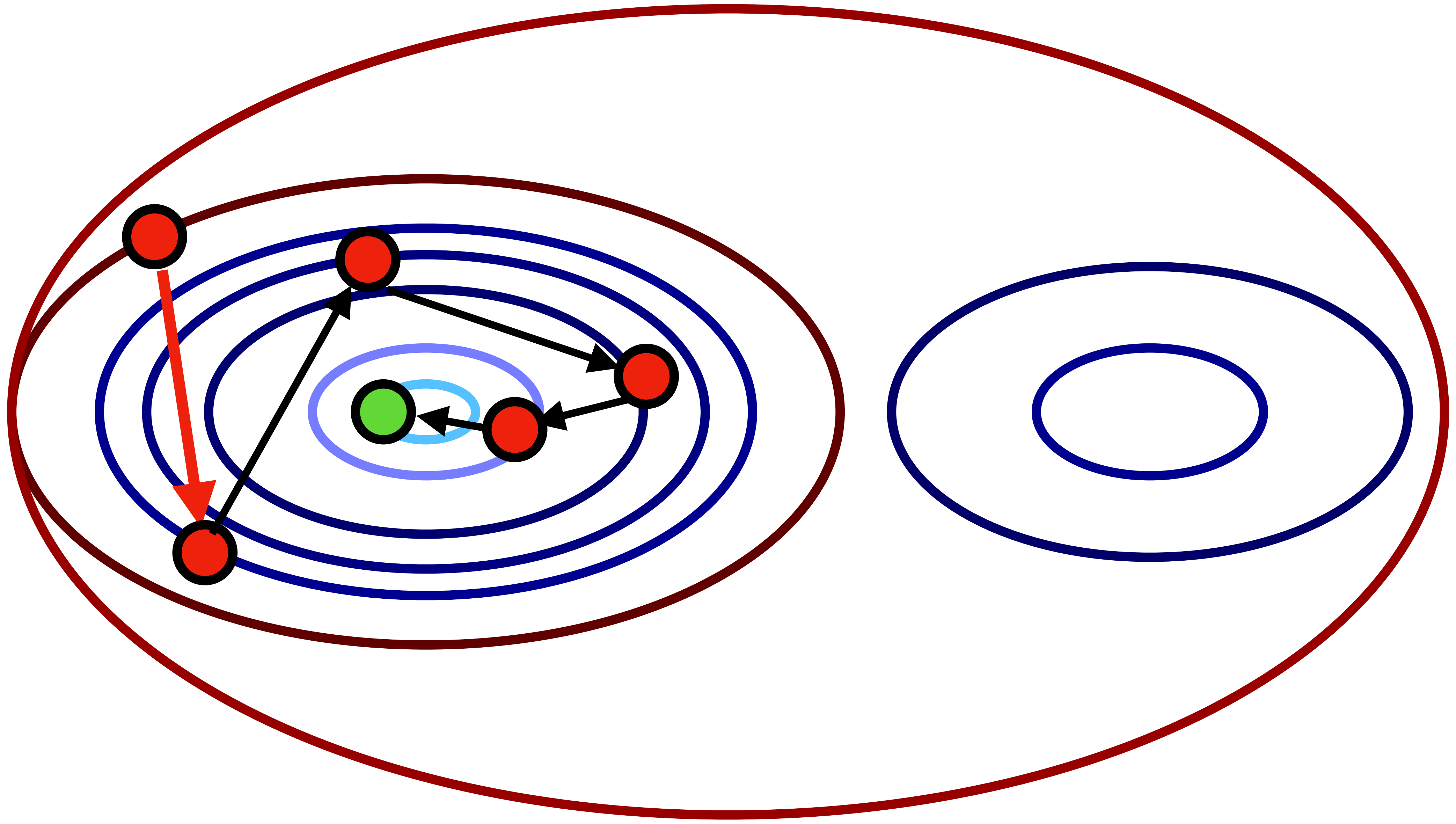
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

α : Learning Rate

Optimizing Loss Functions

Gradient Descent - Formulation

α **controls how big a step to take**



$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

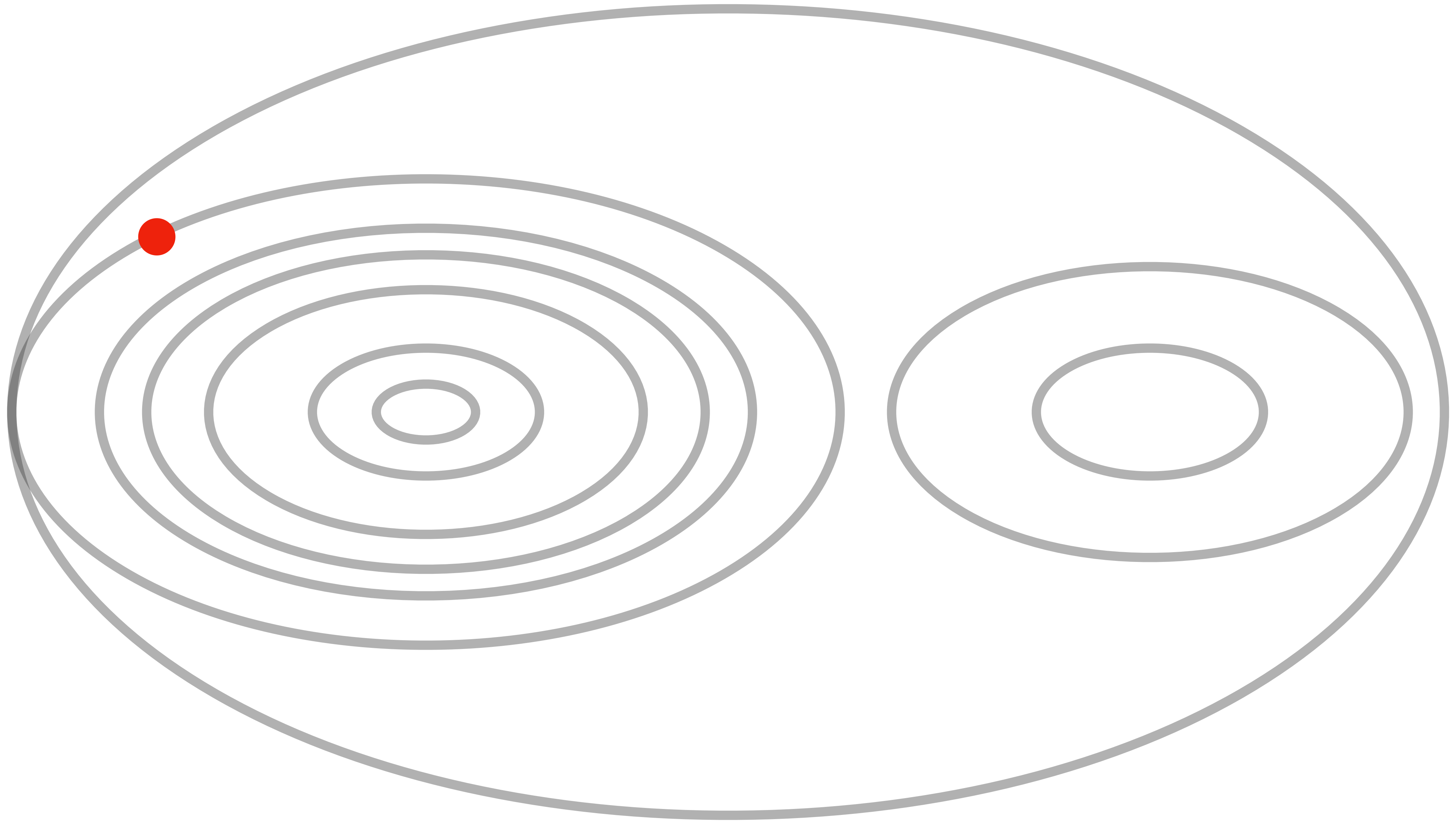
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

α : Learning Rate

Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

What happens when α is too small?
Say $\alpha = 10^{-5}$



$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

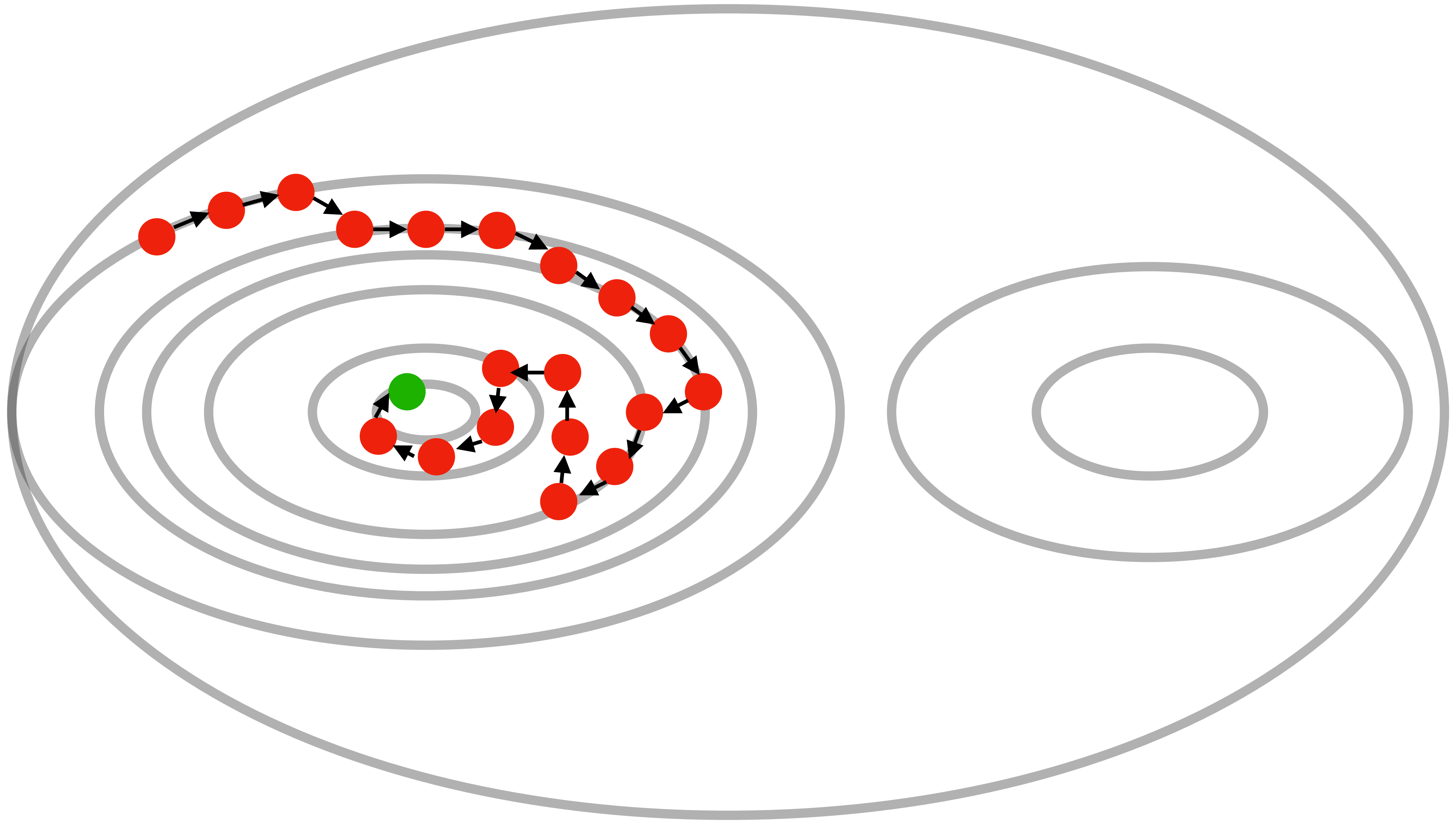
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

α : Learning Rate

Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

What happens when α is too small?
Say $\alpha = 10^{-5}$



$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

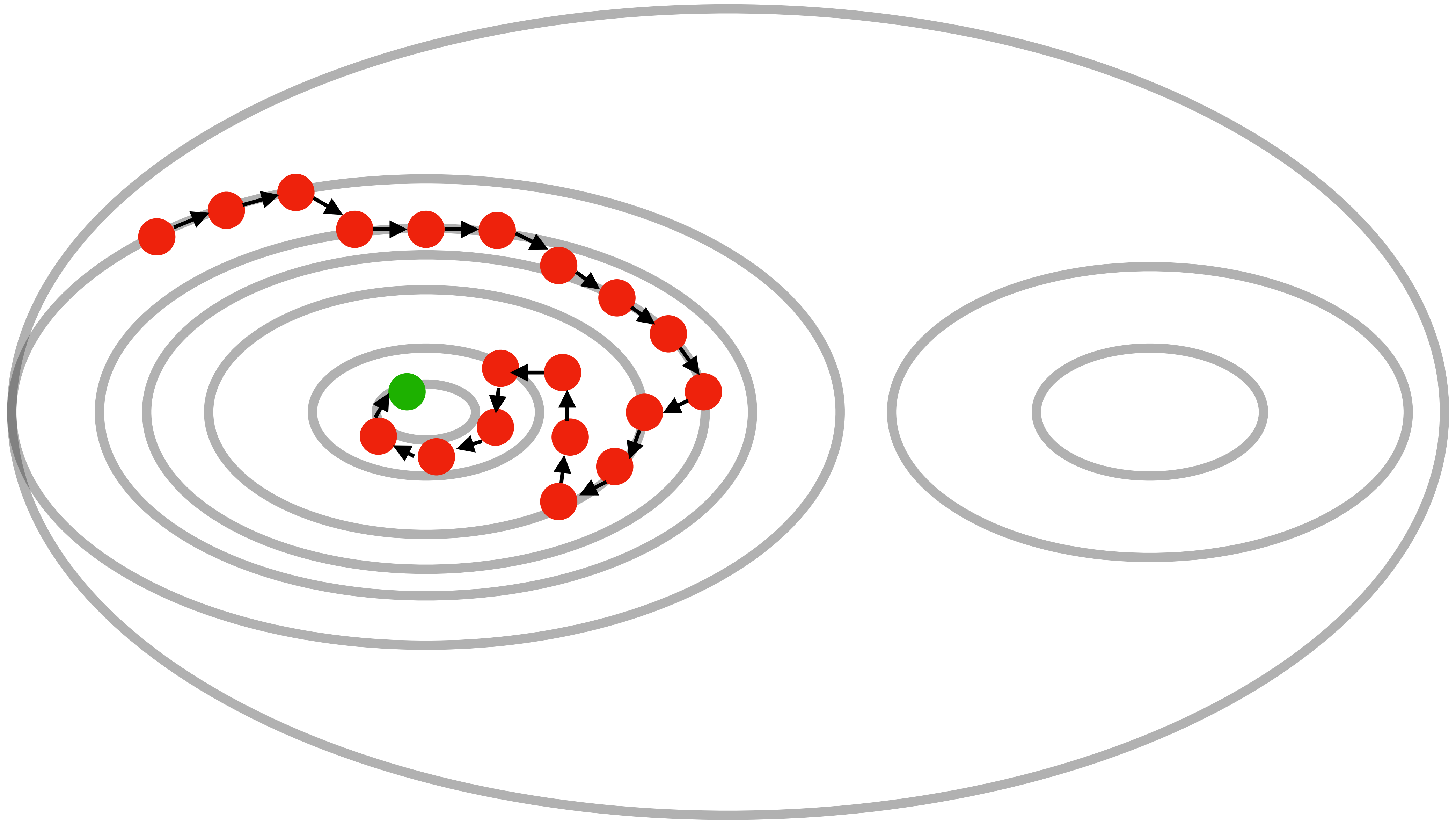
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

α : Learning Rate

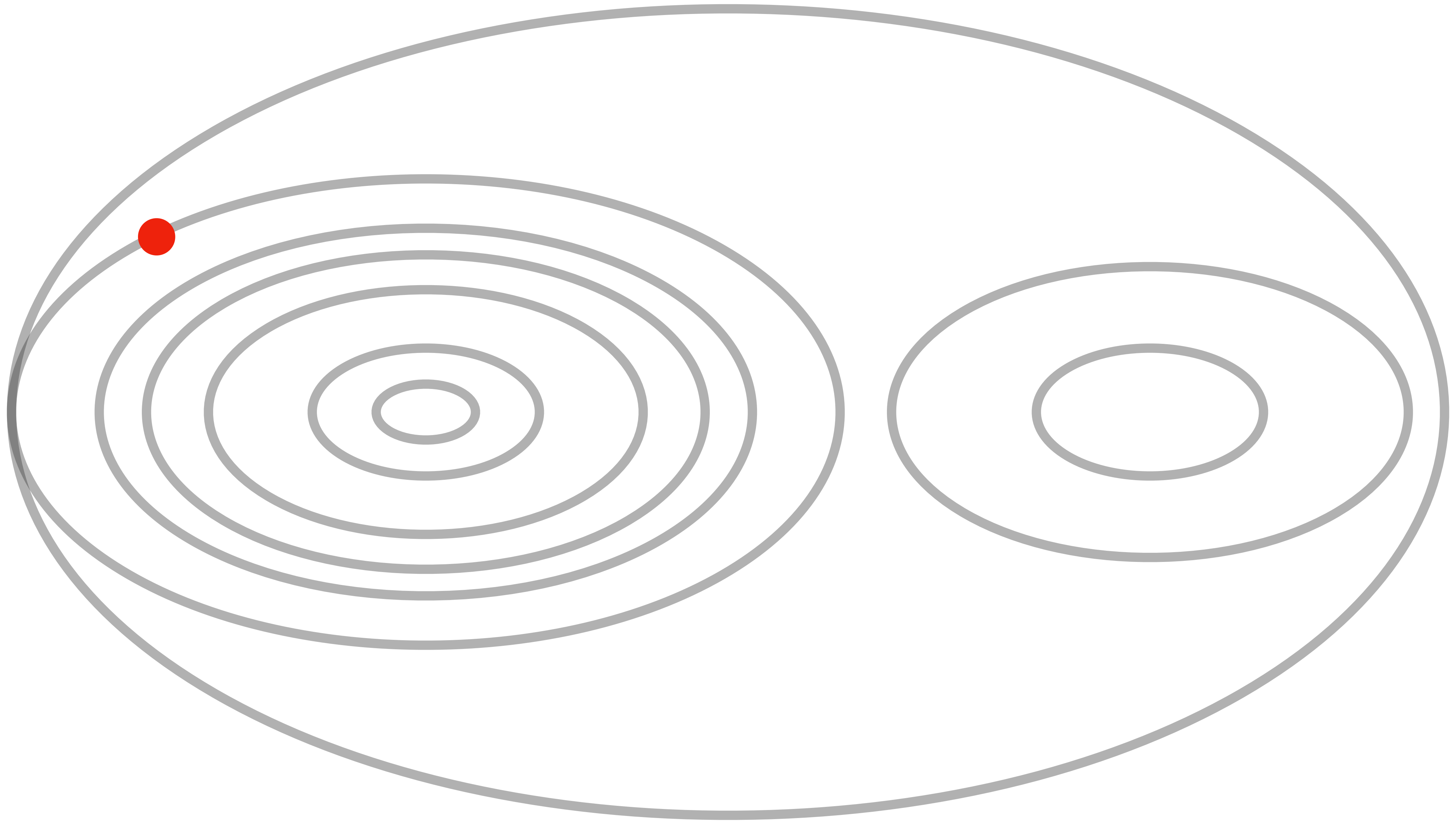
Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

What happens when α is too small?
Say $\alpha = 10^{-5}$



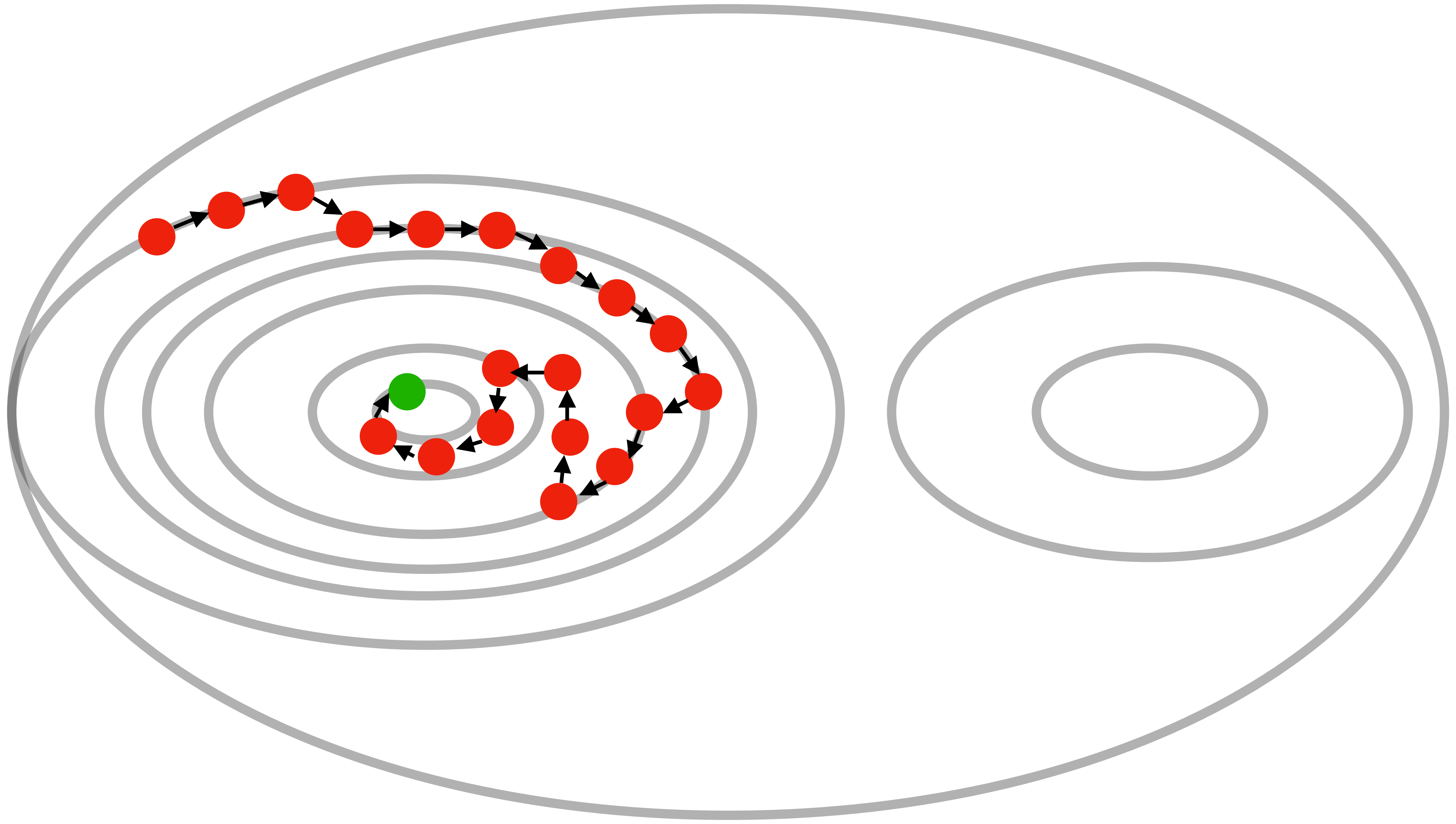
What happens when α is too large?
Say $\alpha = 10$



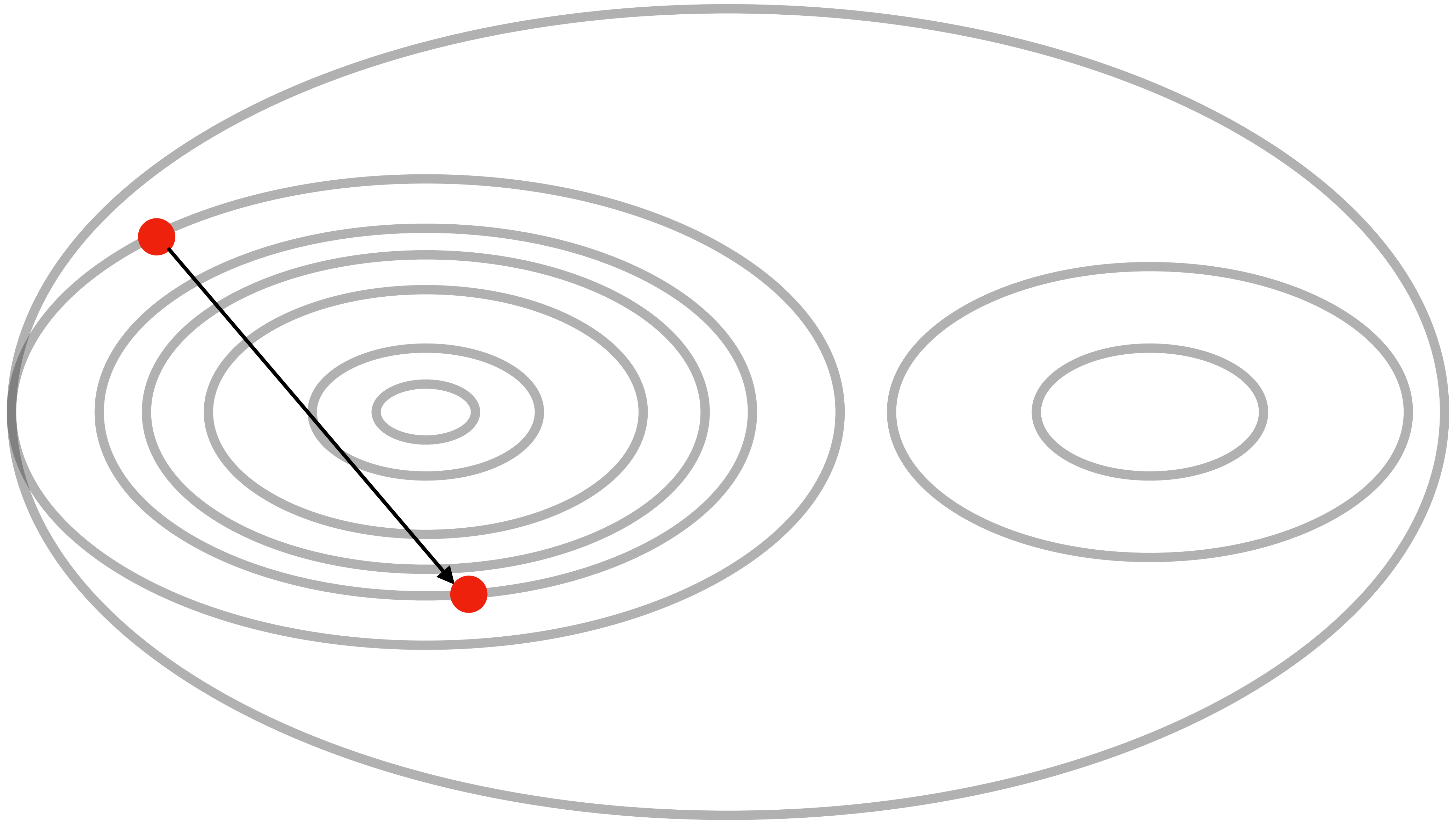
Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

What happens when α is too small?
Say $\alpha = 10^{-5}$



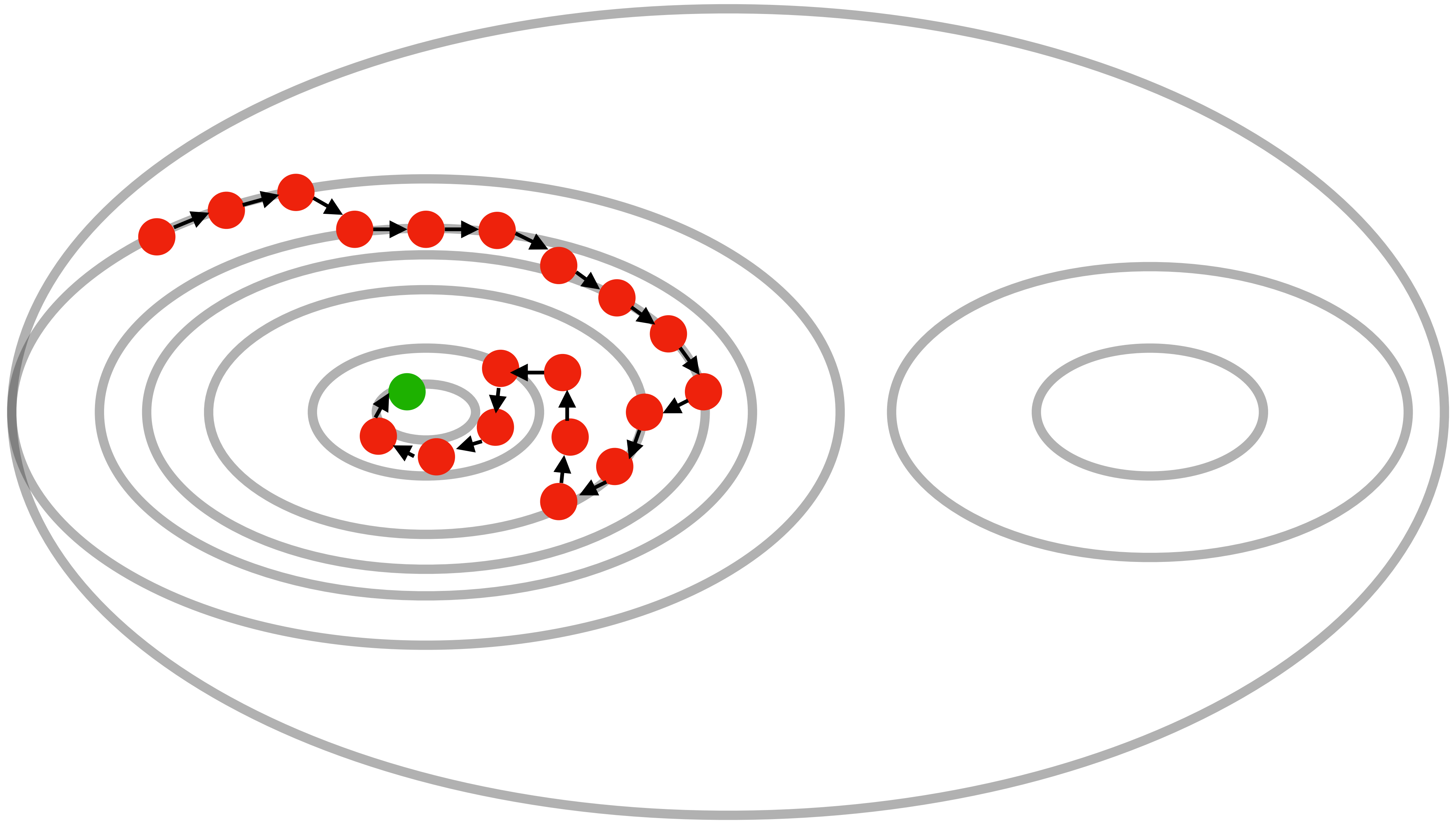
What happens when α is too large?
Say $\alpha = 10$



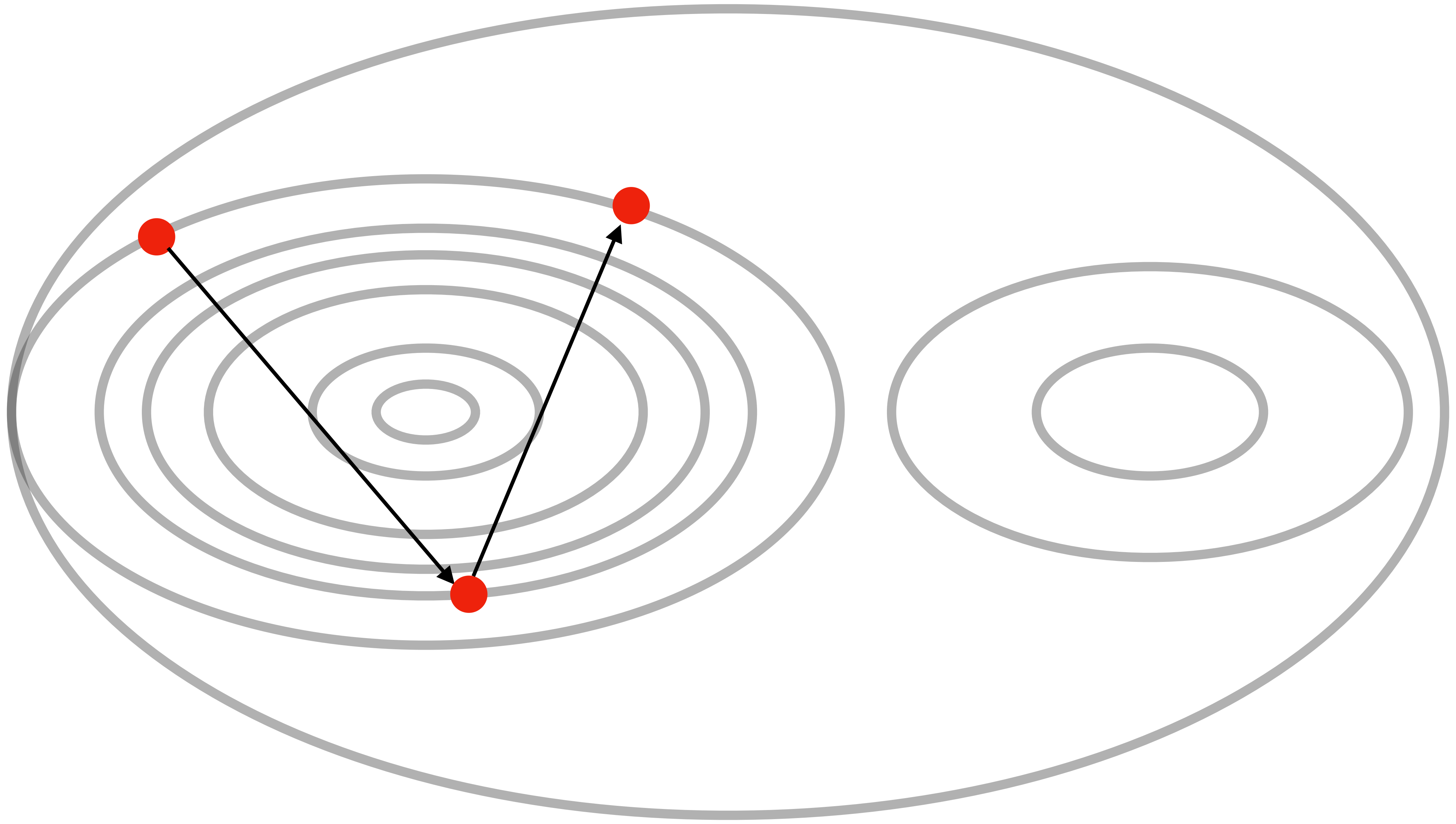
Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

What happens when α is too small?
Say $\alpha = 10^{-5}$



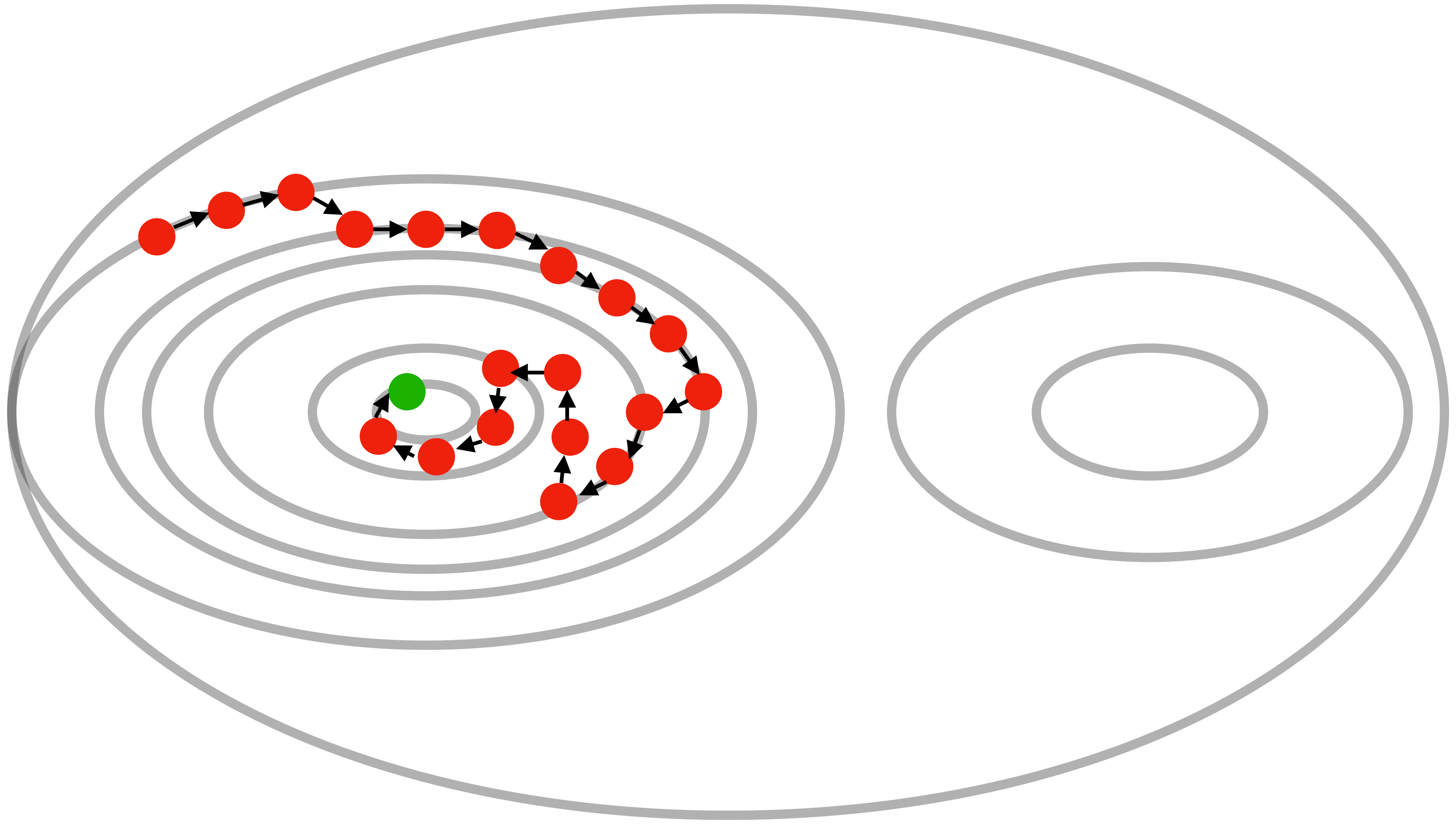
What happens when α is too large?
Say $\alpha = 10$



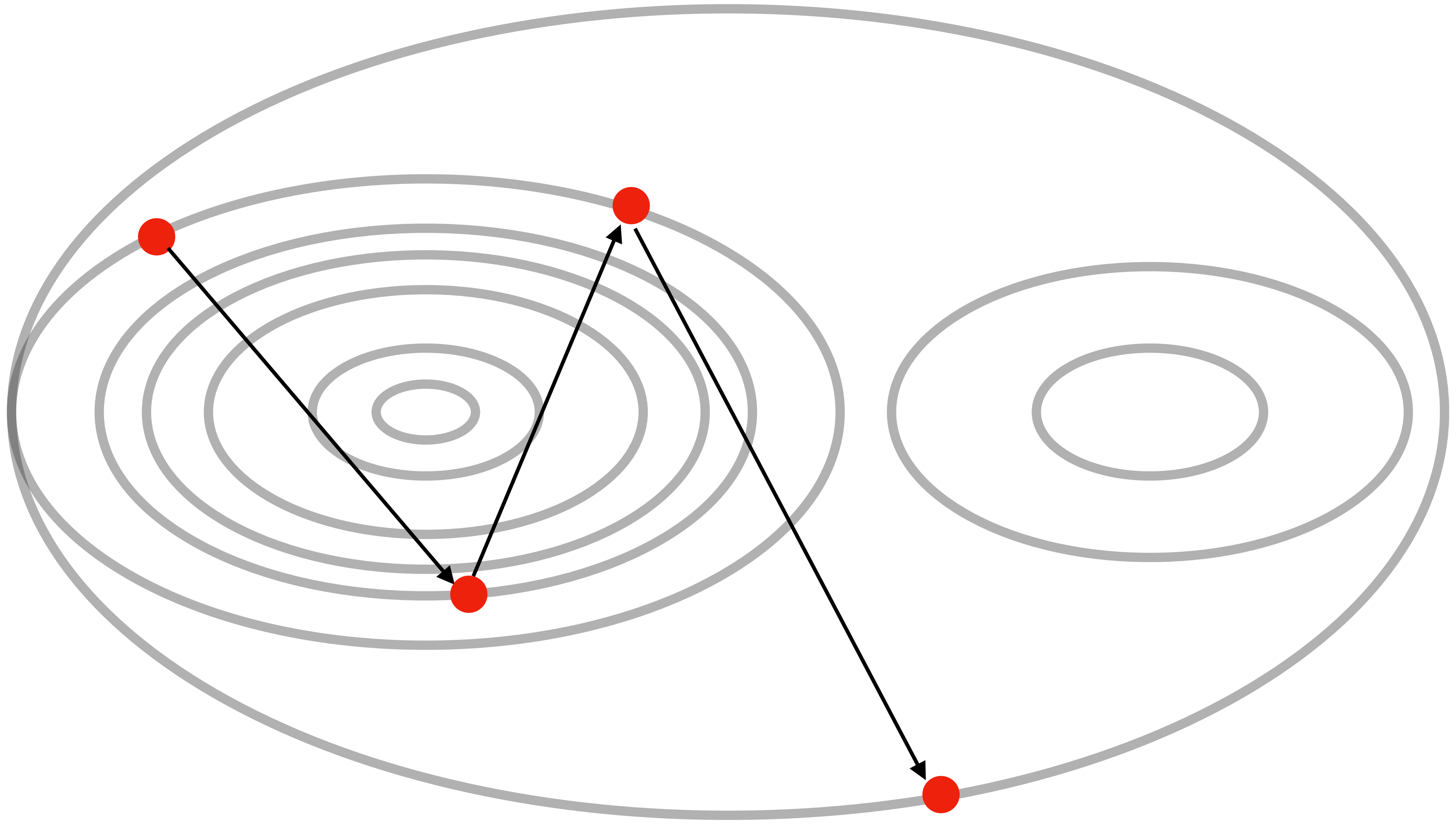
Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

What happens when α is too small?
Say $\alpha = 10^{-5}$



What happens when α is too large?
Say $\alpha = 10$

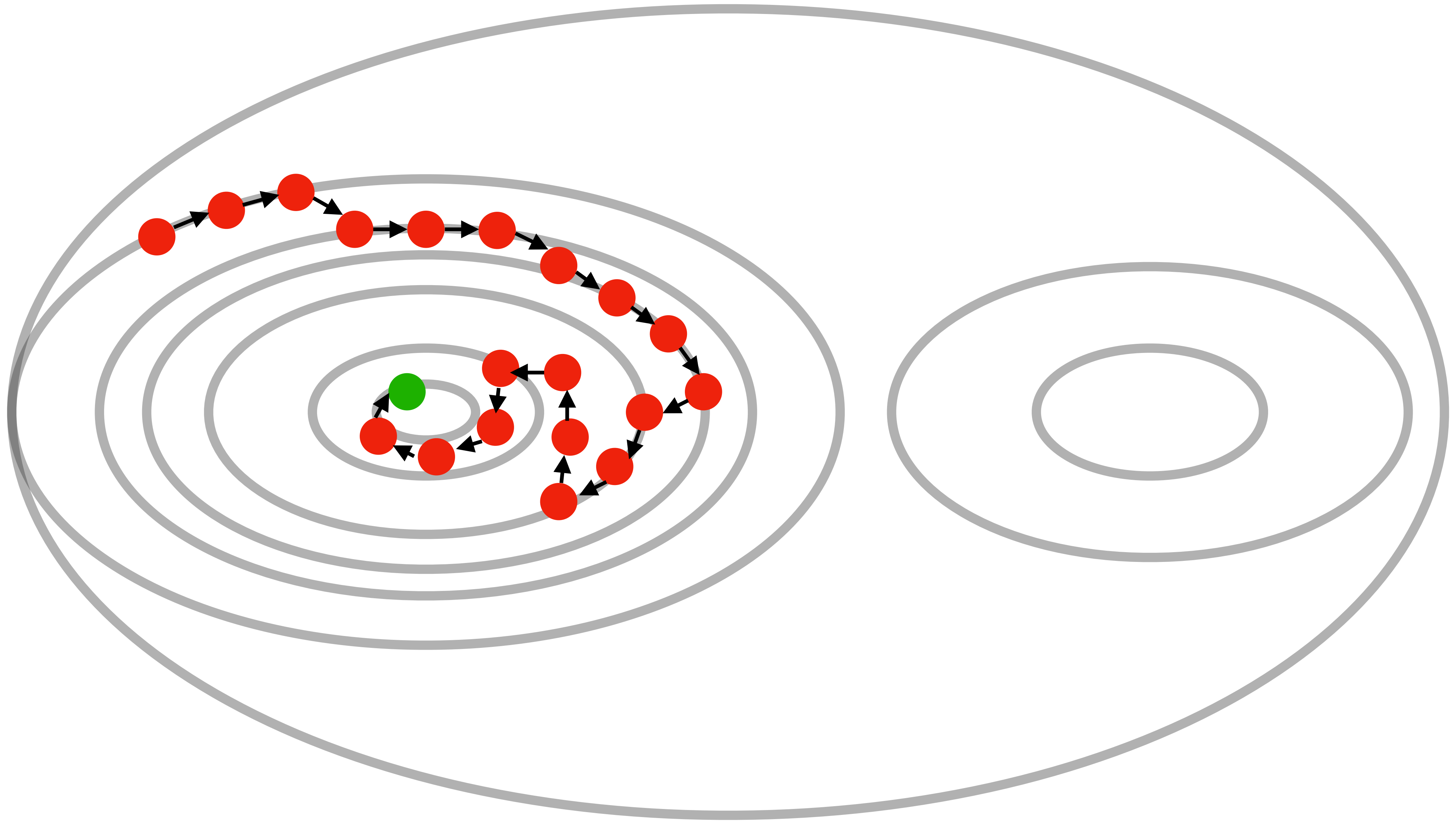


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

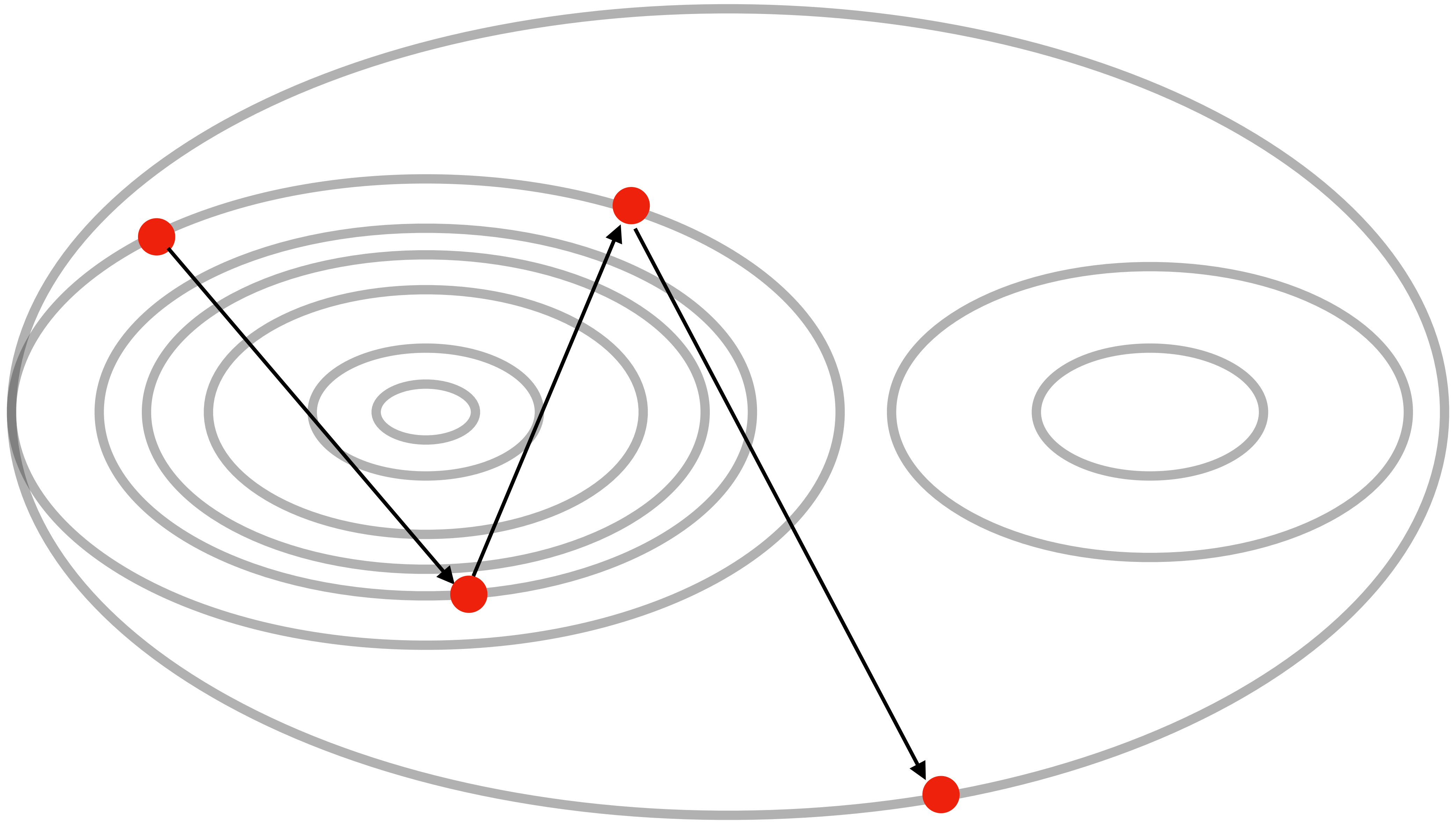
What happens when α is too small?
Say $\alpha = 10^{-5}$

With a small learning rate α, if the loss function is convex, the optimization will eventually **converge**



What happens when α is too large?
Say $\alpha = 10$

With a large learning rate α, if the loss function is convex, the optimization could possibly start **diverging** and never converge



Optimizing Loss Functions

Gradient Descent - Stopping Criterion

Maximum Iteration

Gradient Norm Threshold

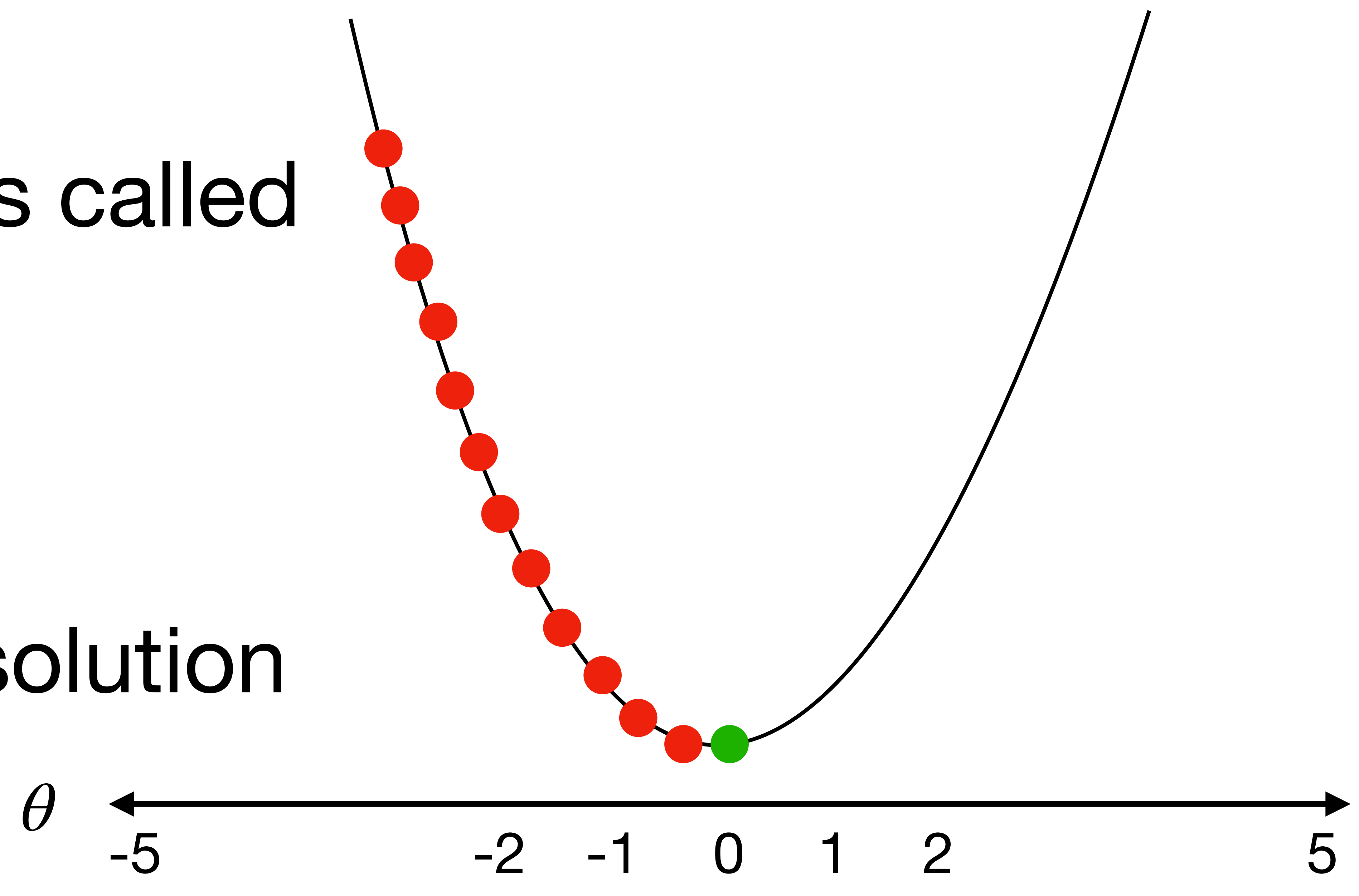
Function Value Change

Parameter Value Change

Optimizing Loss Functions

Gradient Descent - Stopping Criterion

- When do you stop your iterations?
  - Maximum Iteration
    - Each iteration through the training dataset is called an "epoch"
    - Terminate after a fixed number of epochs
    - Simple, but provides no guarantees about solution quality



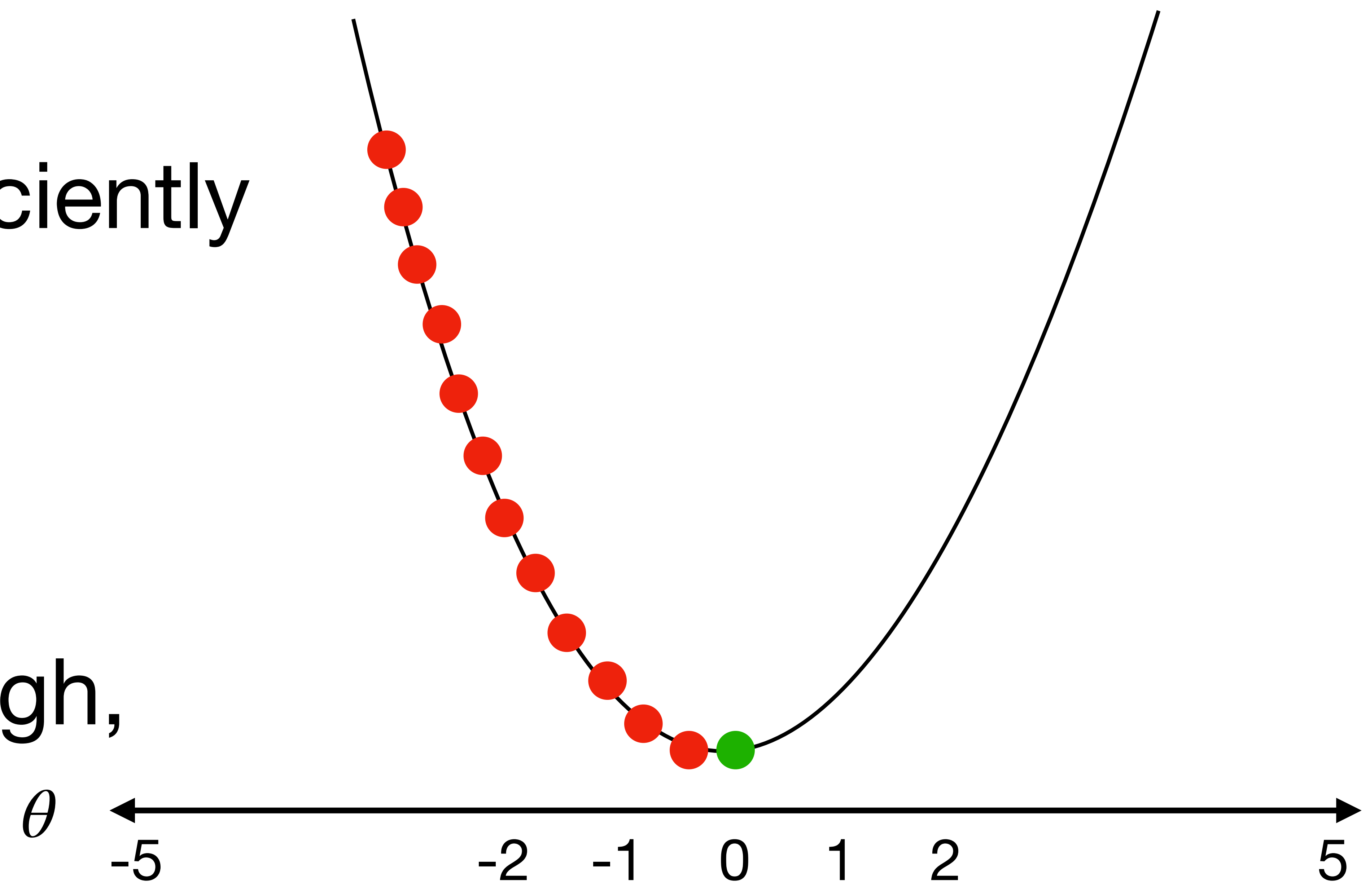
Optimizing Loss Functions

Gradient Descent - Stopping Criterion

- When do you stop your iterations?
  - Gradient Norm Threshold
    - Terminate when the gradient becomes sufficiently small

$$||\nabla \ell_\theta(x)||_2 \leq \epsilon$$

    - At this point, if the gradients are small enough, the parameters won't move much anyway

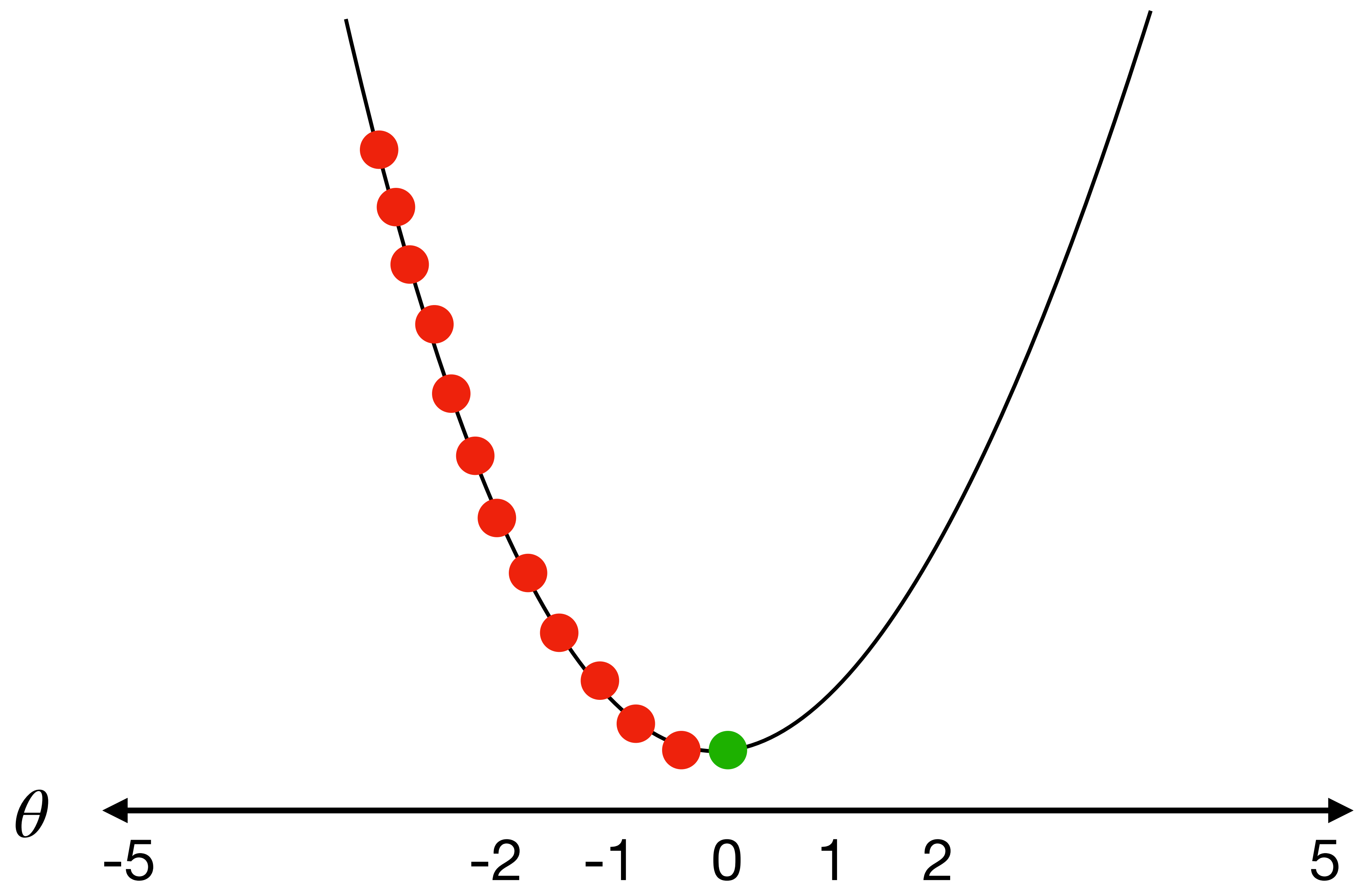


Optimizing Loss Functions

Gradient Descent - Stopping Criterion

- When do you stop your iterations?
  - Function Value Change
    - Terminate when the loss stops changing meaningfully

$$|\ell_{\theta_t}(x) - \ell_{\theta_{t-1}}(x)| \leq \epsilon$$

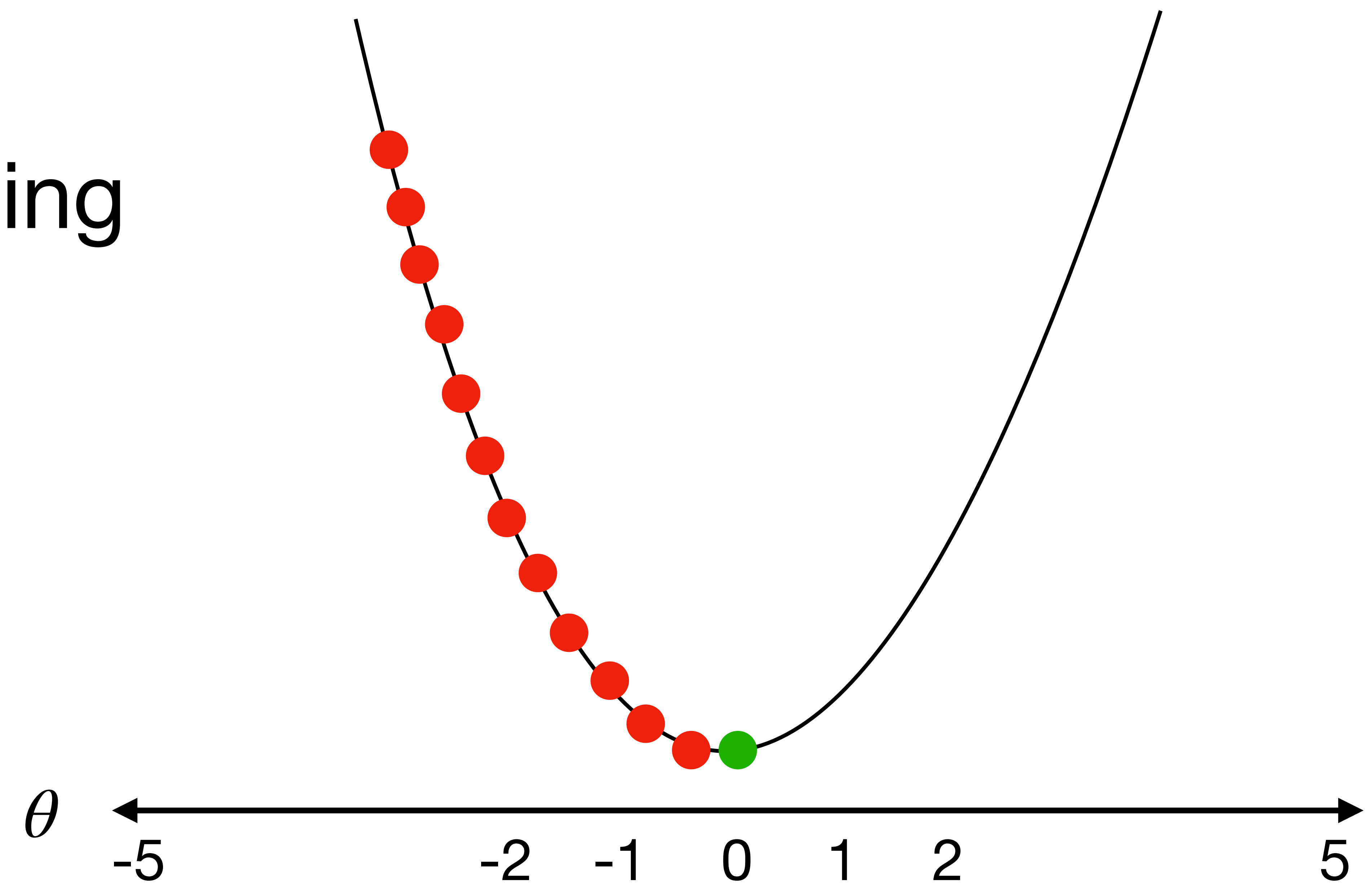


Optimizing Loss Functions

Gradient Descent - Stopping Criterion

- When do you stop your iterations?
  - Parameter Value Change
    - Terminate when the parameters stop changing meaningfully

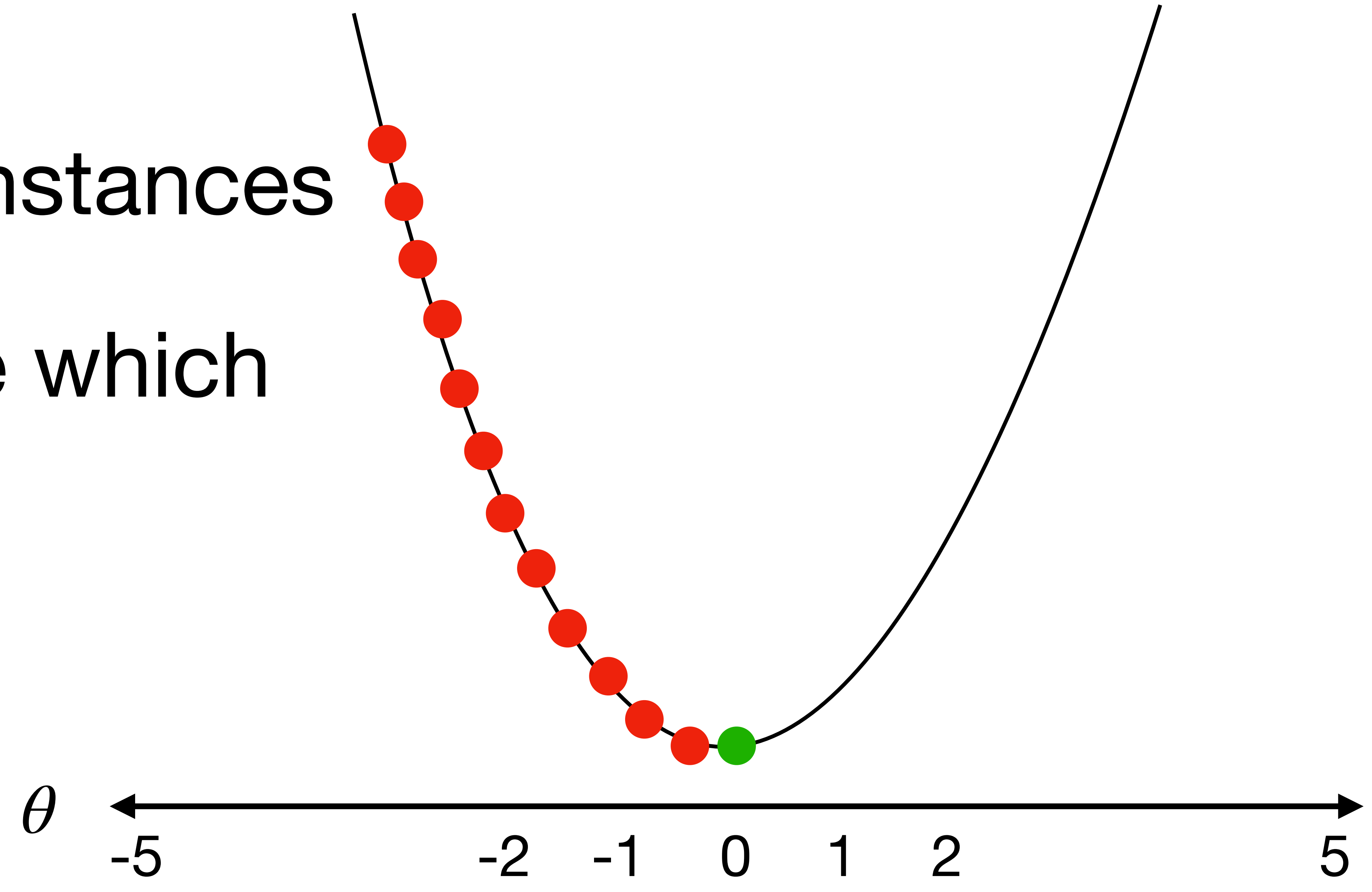
$$|\theta_t - \theta_{t-1}| \leq \epsilon$$



Optimizing Loss Functions

Gradient Descent - Stopping Criterion

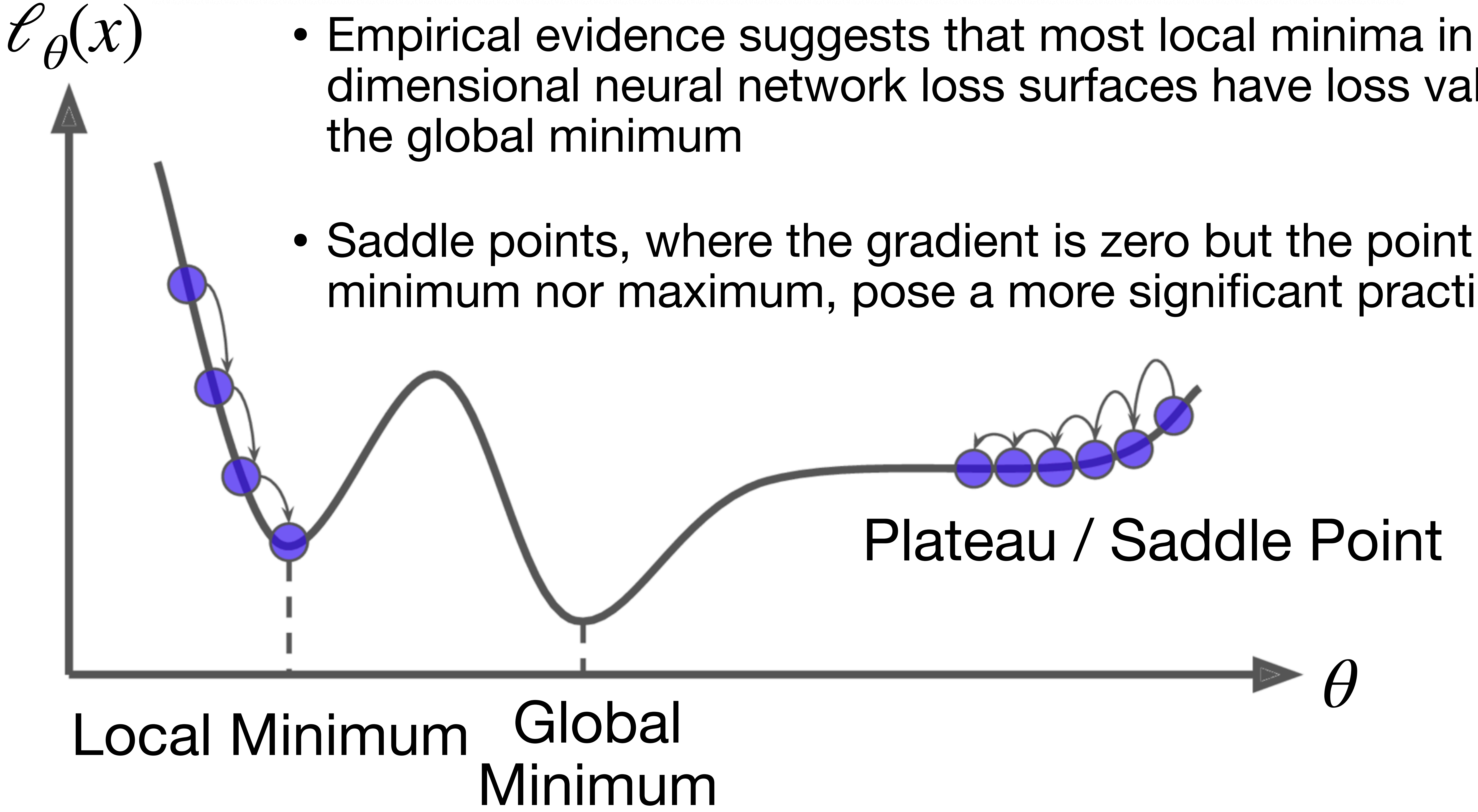
- When do you stop your iterations?
  - Validation Based Stopping (Early Stopping)
    - Monitor performance on a validation set of instances
    - Stop when validation loss begins to increase which signals overfitting
    - Serves as both stopping criterion and regularization



Optimizing Loss Functions

Gradient Descent - More Complicated Functions

- Most deep learning models however have **highly non-convex** loss landscapes
- Empirical evidence suggests that most local minima in high-dimensional neural network loss surfaces have loss values close to the global minimum
- Saddle points, where the gradient is zero but the point is neither a minimum nor maximum, pose a more significant practical challenge.



Optimizing Loss Functions

Gradient Descent - Momentum

Optimizing Loss Functions

Gradient Descent - Momentum

Optimizing Loss Functions

Gradient Descent - Momentum

- Standard gradient descent can oscillate in ravines
    - Areas where the surface curves **more steeply in one dimension** than another
    - Or they can get stuck in plateau / saddle points
- Momentum helps accelerate gradient descent by accumulating velocity in **directions of consistent gradient**

$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

$$\theta_t = \theta_{t-1} - \alpha \nabla \ell_{\theta_{t-1}}$$

Optimizing Loss Functions

Gradient Descent - Momentum

- Momentum helps accelerate gradient descent by accumulating velocity in **directions of consistent gradient**

$$\theta_t = \theta_{t-1} - \alpha \nabla \ell_{\theta_{t-1}}$$

With Momentum

$$v_t = \beta \cdot v_{t-1} + \nabla \ell_{\theta_{t-1}}$$

$$\theta_t = \theta_{t-1} - \alpha \cdot v_t$$

Optimizing Loss Functions

Gradient Descent - Momentum

- Momentum helps accelerate gradient descent by accumulating velocity in **directions of consistent gradient**

$$\theta_t = \theta_{t-1} - \alpha \nabla \ell_{\theta_{t-1}}$$

With Momentum

Velocity Vector $\boxed{v_t} = \beta \cdot v_{t-1} + \nabla \ell_{\theta_{t-1}}$

$$\theta_t = \theta_{t-1} - \alpha \cdot \boxed{v_t}$$

Optimizing Loss Functions

Gradient Descent - Momentum

- Momentum helps accelerate gradient descent by accumulating velocity in **directions of consistent gradient**

$$\theta_t = \theta_{t-1} - \alpha \nabla \ell_{\theta_{t-1}}$$

With Momentum

β is the momentum coefficient, typically set to 0.9

$$v_t = \beta \cdot v_{t-1} + \nabla \ell_{\theta_{t-1}}$$

$$\theta_t = \theta_{t-1} - \alpha \cdot v_t$$

Optimizing Loss Functions

Gradient Descent - Momentum

- Momentum helps accelerate gradient descent by accumulating velocity in **directions of consistent gradient**

$$\theta_t = \theta_{t-1} - \alpha \nabla \ell_{\theta_{t-1}}$$

With Momentum

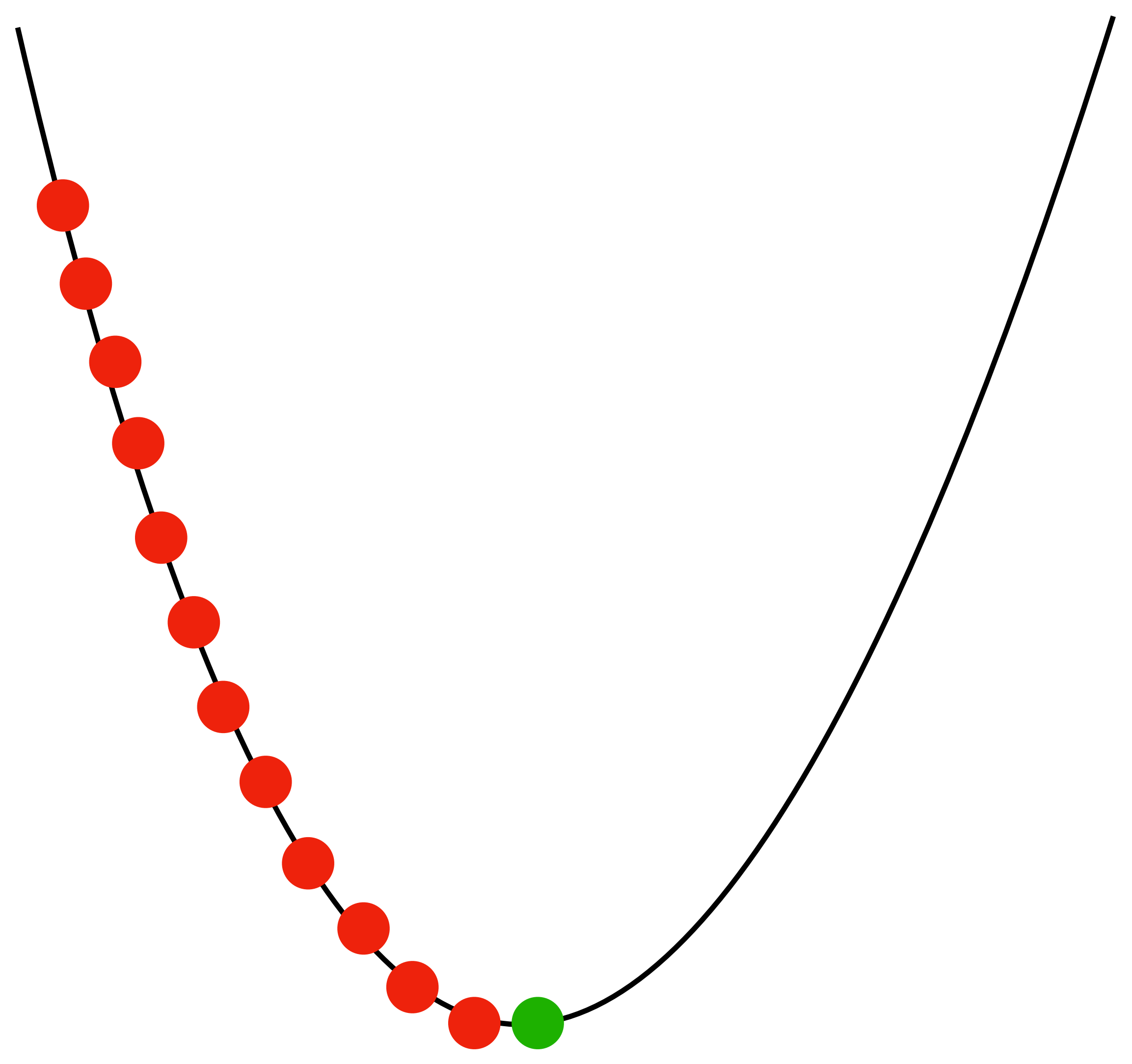
If $\beta = 0$, you get back standard gradient descent

$$v_t = \boxed{\beta} \cdot v_{t-1} + \nabla \ell_{\theta_{t-1}}$$

$$\theta_t = \theta_{t-1} - \alpha \cdot v_t$$

Optimizing Loss Functions

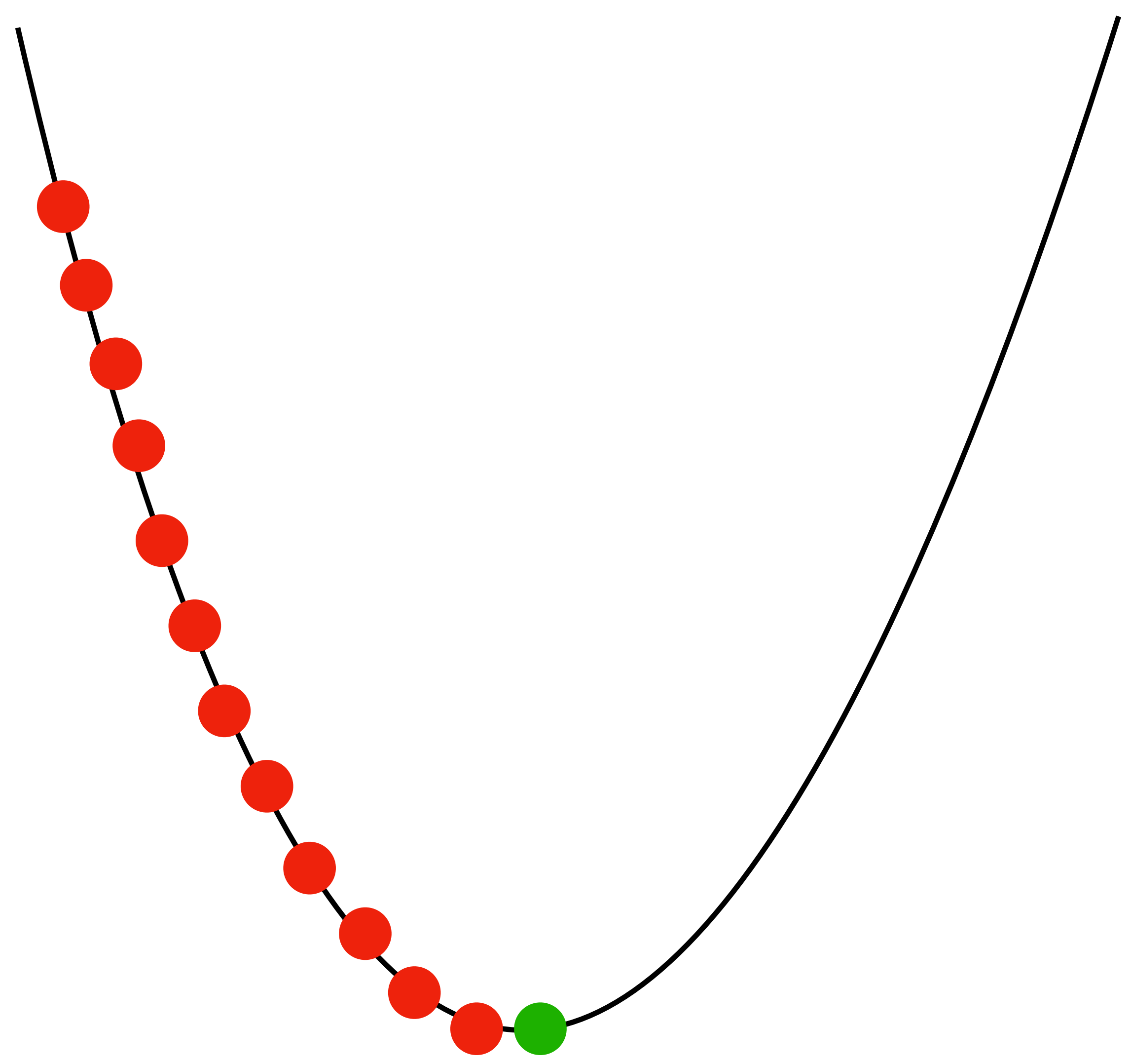
Gradient Descent - Adaptive Step Sizes



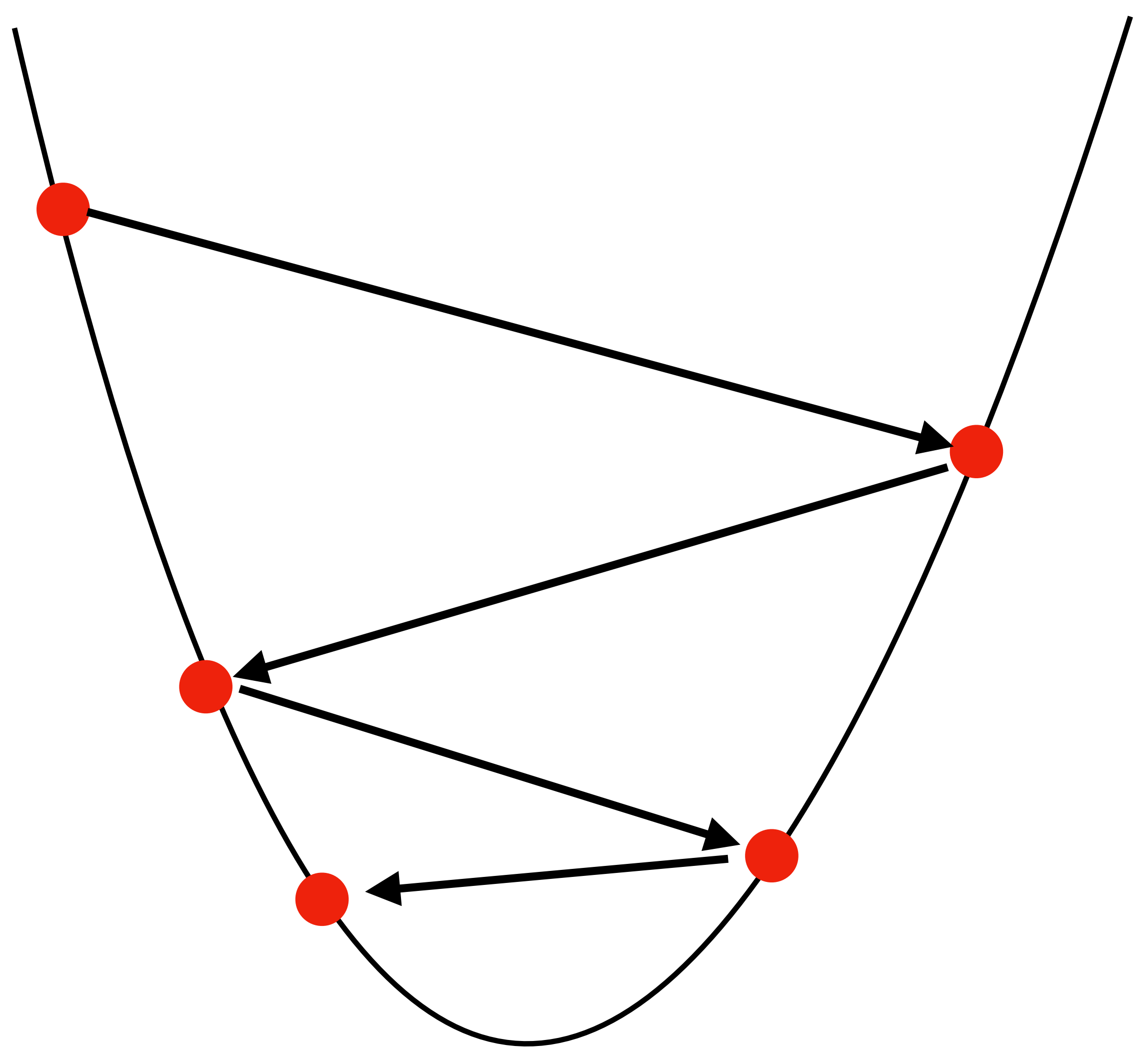
α is too small
Finds the optimal but too slow

Optimizing Loss Functions

Gradient Descent - Adaptive Step Sizes



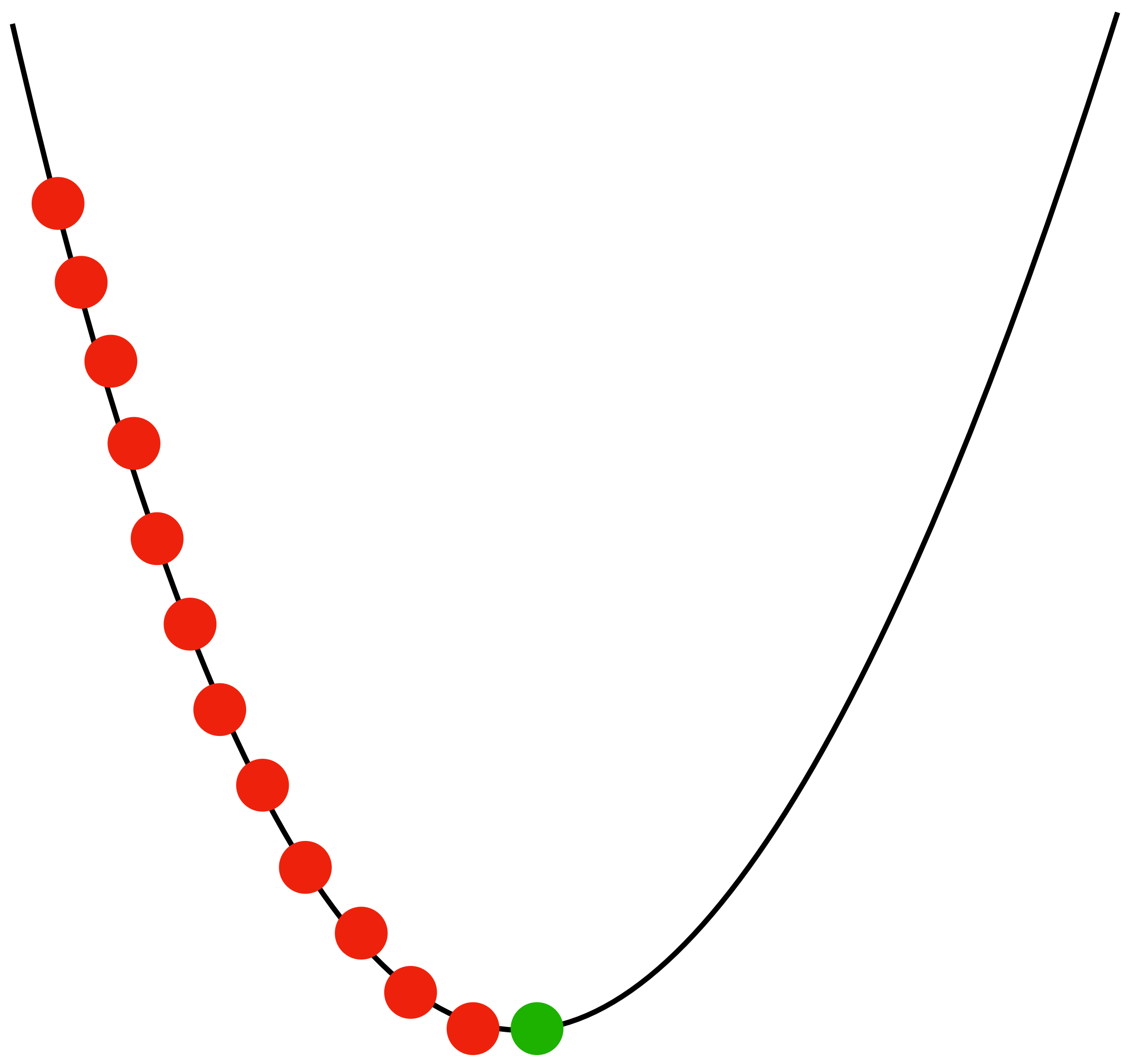
α is too small
Finds the optimal but too slow



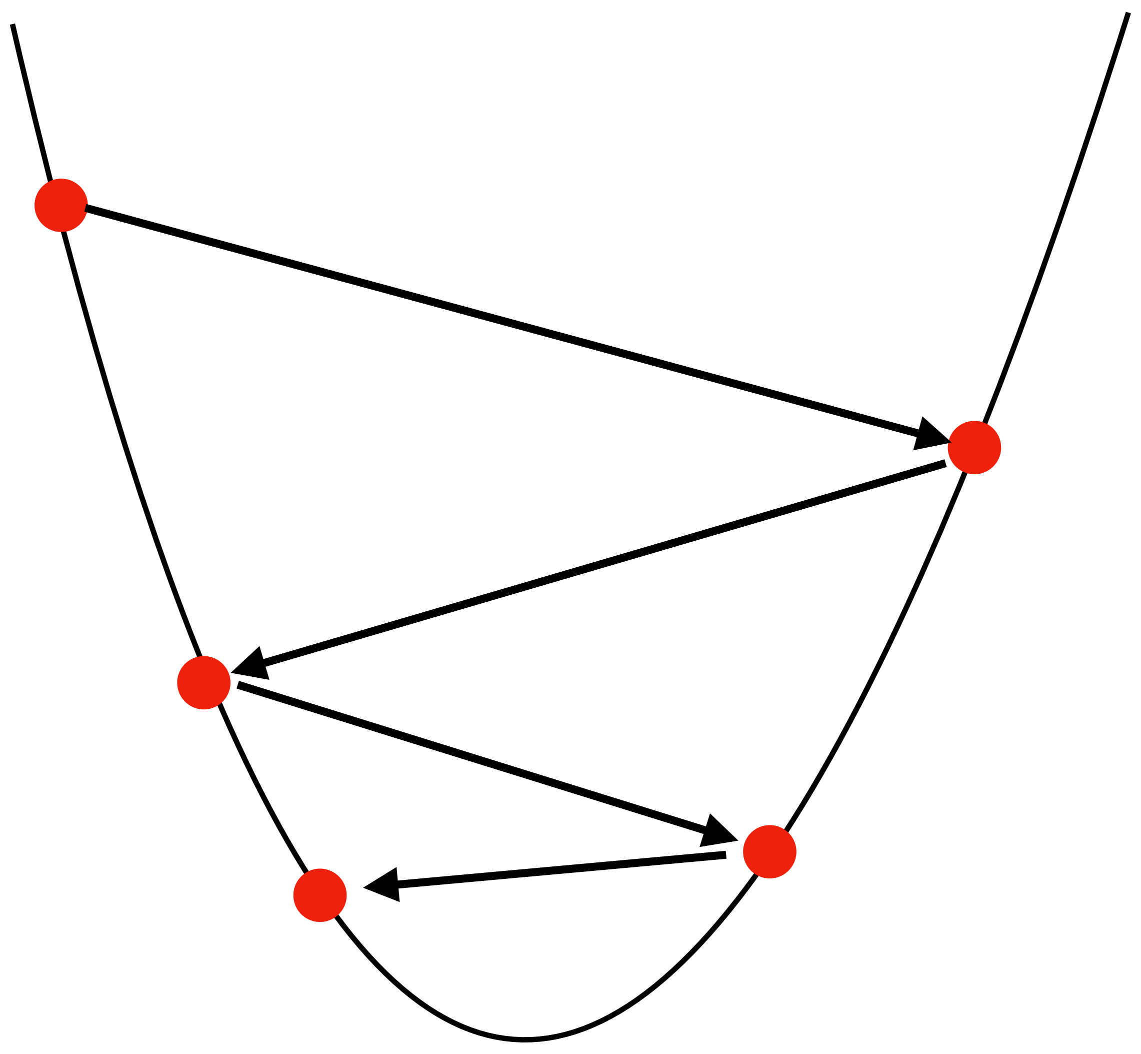
α is too large
Might not find optimal
Could even begin to diverge

Optimizing Loss Functions

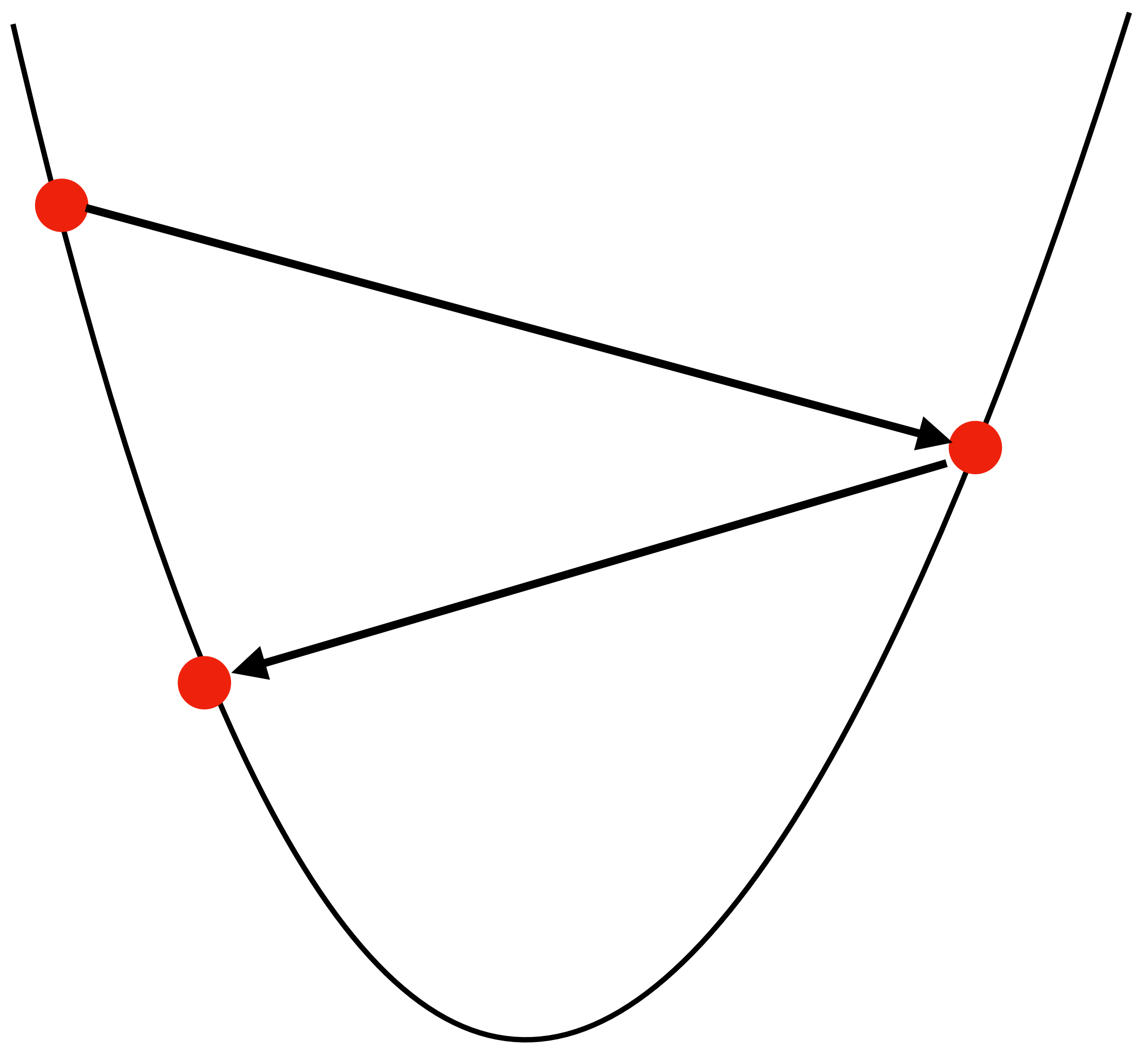
Gradient Descent - Adaptive Step Sizes



α is too small
Finds the optimal but too slow



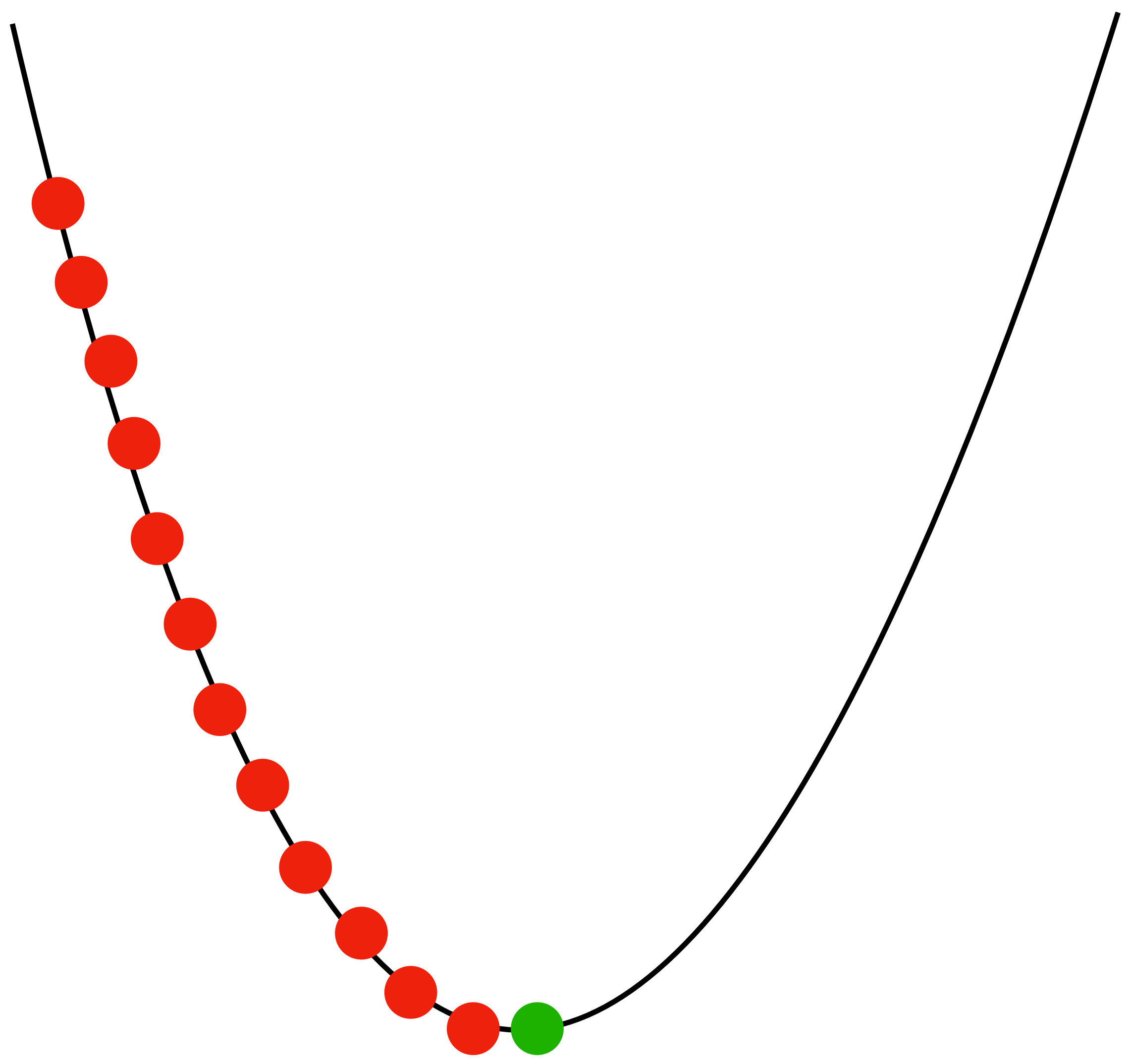
α is too large
Might not find optimal
Could even begin to diverge



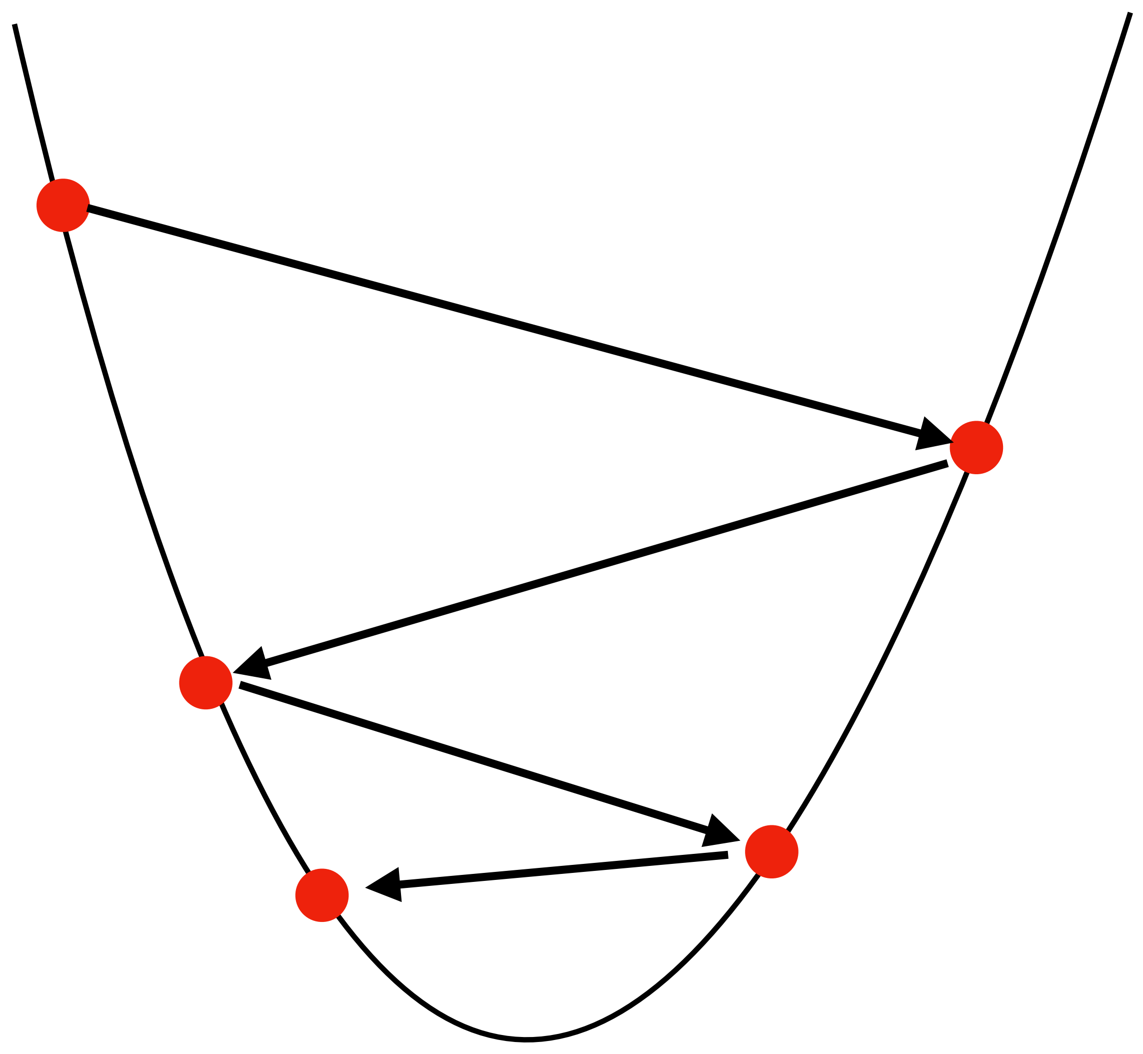
What if you set α to be large initially?

Optimizing Loss Functions

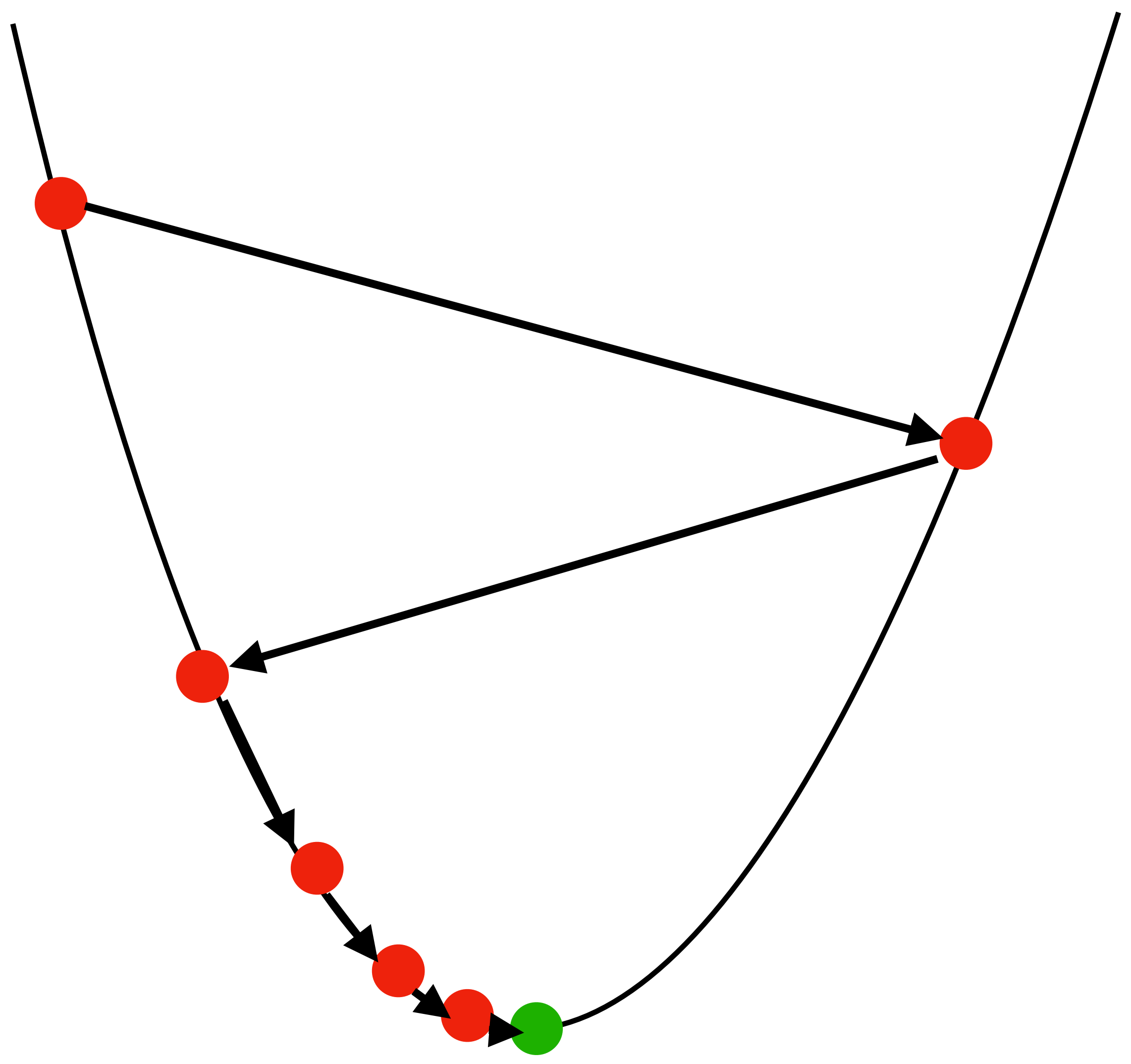
Gradient Descent - Adaptive Step Sizes



α is too small
Finds the optimal but too slow



α is too large
Might not find optimal
Could even begin to diverge



And keep reducing α as
number of epochs increases?

Optimizing Loss Functions

Gradient Descent - Per Parameter Adaptive Learning Rates

- A single global learning rate may be suboptimal
  - Some parameters might benefit from larger updates while others need smaller ones.
  - Adaptive methods adjust the learning rate for each parameter individually based on historical gradient information.

Optimizing Loss Functions
Gradient Descent - AdaGrad

Optimizing Loss Functions

Gradient Descent - AdaGrad + RMSProp

Optimizing Loss Functions

Gradient Descent - AdaGrad

- AdaGrad adapts the learning rate for **each parameter** based on the sum of squared historical gradients

$$G_t = G_{t-1} + (\nabla \ell_{\theta_{t-1}})^2$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{G_t} + \epsilon} \cdot \nabla \ell_{\theta_{t-1}}$$

  - Parameters with <u>large</u> historical gradients receive <u>smaller</u> updates
  - Parameters with <u>small</u> historical gradients receive <u>larger</u> updates
  - The limitation is that the accumulated sum G_t grows monotonically, eventually making the learning rate vanishingly small.

Optimizing Loss Functions

Gradient Descent - AdaGrad

- AdaGrad adapts the learning rate for **each parameter** based on the sum of squared historical gradients

$$G_t = G_{t-1} + (\nabla \ell_{\theta t-1})^2$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{G_t + \epsilon}} \cdot \nabla \ell_{\theta_{t-1}}$$

  - Parameters with <u>large</u> historical gradients receive <u>smaller</u> updates
  - Parameters with <u>small</u> historical gradients receive <u>larger</u> updates
  - **The limitation is that the accumulated sum G_t grows monotonically, eventually making the learning rate vanishingly small.**

Optimizing Loss Functions

Gradient Descent - RMSProp

- RMSprop addresses AdaGrad's diminishing learning rate by using an exponentially decaying average of squared gradients

$$G_t = \rho \cdot G_{t-1} + (1 - \rho) \cdot (\nabla \ell_{\theta t-1})^2$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{G_t} + \epsilon} \cdot \nabla \ell_{\theta_{t-1}}$$

  - The decay rate ρ is typically set to 0.9.
  - This prevents the learning rate from decaying to zero while still adapting to the gradient scale.

Optimizing Loss Functions

Gradient Descent - ADAM

- Adam (**Ada**ptive **M**oment Estimation) combines the benefits of **momentum** (first moment) with the adaptive learning rates of **RMSProp** (second moment)

Optimizing Loss Functions

Gradient Descent - ADAM

- Adam (**Ada**ptive **M**oment Estimation) combines the benefits of **momentum** (first moment) with the adaptive learning rates of **RMSProp** (second moment)

Optimizing Loss Functions

Gradient Descent - ADAM

- Adam (**Ada**ptive **M**oment Estimation) combines the benefits of **momentum** (first moment) with the adaptive learning rates of **RMSProp** (second moment)

Adam maintains **two** moving averages

First Moment (mean): $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla \ell_{\theta_{t-1}}$

Second Moment (variance): $v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla \ell_{\theta_{t-1}})^2$

Optimizing Loss Functions
Gradient Descent - ADAM

- Adam (**Ada**ptive **M**oment Estimation) combines the benefits of **momentum** (first moment) with the adaptive learning rates of **RMSProp** (second moment)

Adam maintains **two** moving averages

First Moment (mean): $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla \ell_{\theta_{t-1}}$

Second Moment (variance): $v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla \ell_{\theta_{t-1}})^2$

Update: $\theta_t = \theta_{t-1} - \dfrac{\alpha}{\sqrt{v_t} + \epsilon} \cdot m_t$

Optimizing Loss Functions

Gradient Descent - ADAM

- Adam (**Ada**ptive **M**oment Estimation) combines the benefits of **momentum** (first moment) with the adaptive learning rates of **RMSProp** (second moment)

Adam maintains **two** moving averages

First Moment (mean): $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla \ell_{\theta_{t-1}}$

Second Moment (variance): $v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla \ell_{\theta_{t-1}})^2$

Update: $\theta_t = \theta_{t-1} - \dfrac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$

Bias Correction: Important for early iterations when estimates are biased towards 0

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Optimizing Loss Functions

Gradient Descent - ADAM

- Adam (**Ada**ptive **M**oment Estimation) combines the benefits of **momentum** (first moment) with the adaptive learning rates of **RMSProp** (second moment)

Adam maintains **two** moving averages

First Moment (mean): $m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla \ell_{\theta_{t-1}}$

Second Moment (variance): $v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla \ell_{\theta_{t-1}})^2$

Update: $\theta_t = \theta_{t-1} - \dfrac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \cdot \hat{m}_t$

Default Hyperparameters: $\beta_1 = 0.9,\ \beta_2 = 0.999,\ \alpha = 10^{-3}$

Bias Correction: Important for early iterations when estimates are biased towards 0

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t}$$

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

- Batch Gradient Descent
  - Use **entire training set per epoch**
  - The whole training dataset is used to compute a single parameter update

$$\theta_t = \theta_{t-1} - \alpha \frac{1}{m} \sum_{i=1}^{m} \nabla \ell_{\theta_{t-1}}(x_i, y_i)$$

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

- Batch Gradient Descent
    - Use **entire training set per epoch**
    - The whole training dataset is used to compute a single parameter update
    - One epoch leads to **one** parameter update

$$\theta_t = \theta_{t-1} - \alpha \frac{1}{m} \boxed{\sum_{i=1}^{m}} \nabla \ell_{\theta_{t-1}}(x_i, y_i)$$

Sum over the whole training dataset

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

- Stochastic Gradient Descent
  - Use **one** randomly selected training data point at each step
  - Parameters are updated after looking at each data point
  - One epoch leads to **m** parameter updates

$$\theta_t = \theta_{t-1} - \alpha \nabla \ell_{\theta_{t-1}}(x_i, y_i)$$

Train / Test Splits

- Generally data is split into a training dataset and a testing data
- Rough rule of thumb is that this is an 80-20 split

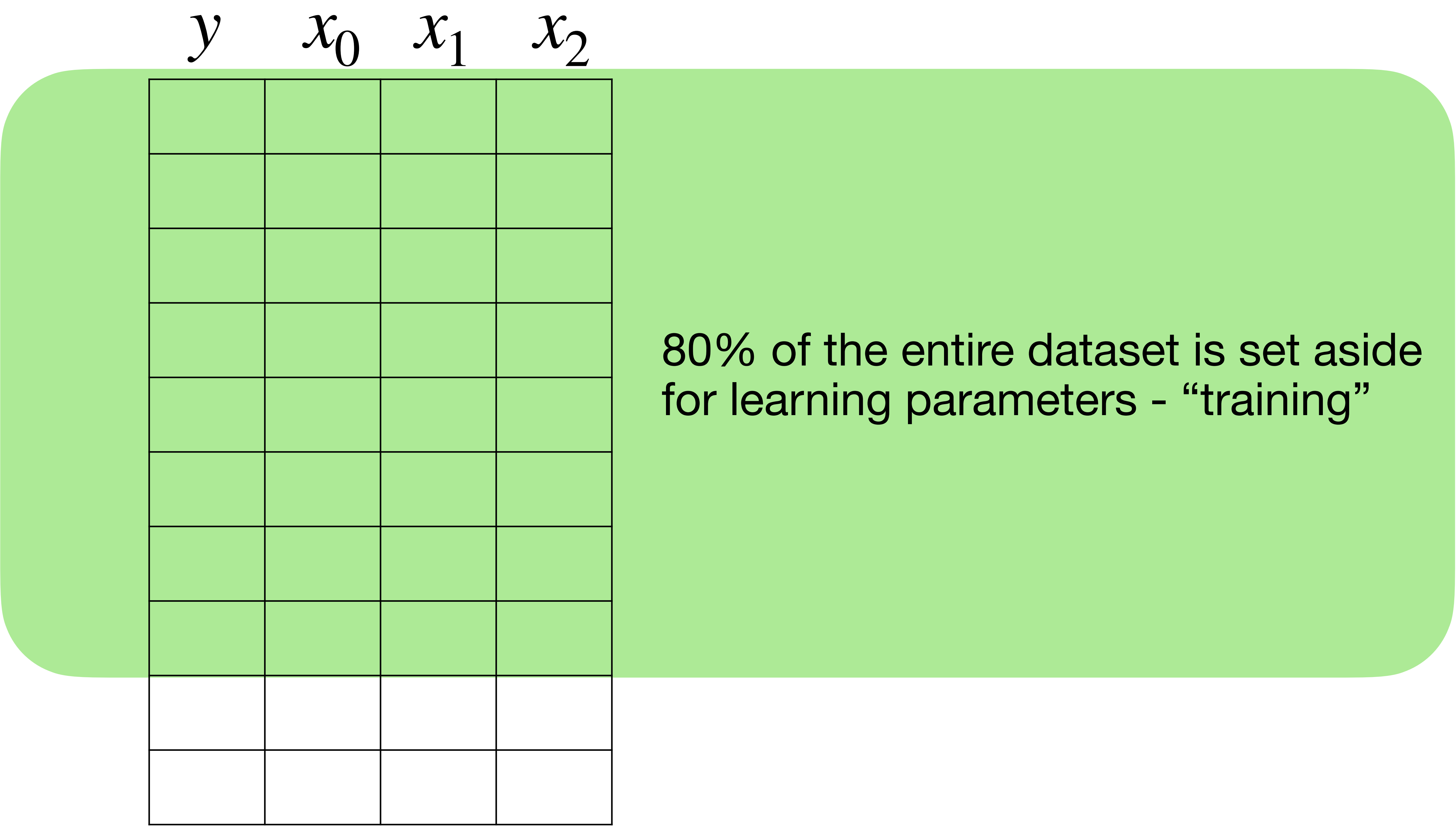
Train / Test Splits

- Generally data is split into a training dataset and a testing data
- Rough rule of thumb is that this is an 80-20 split

y	x_0	x_1	x_2

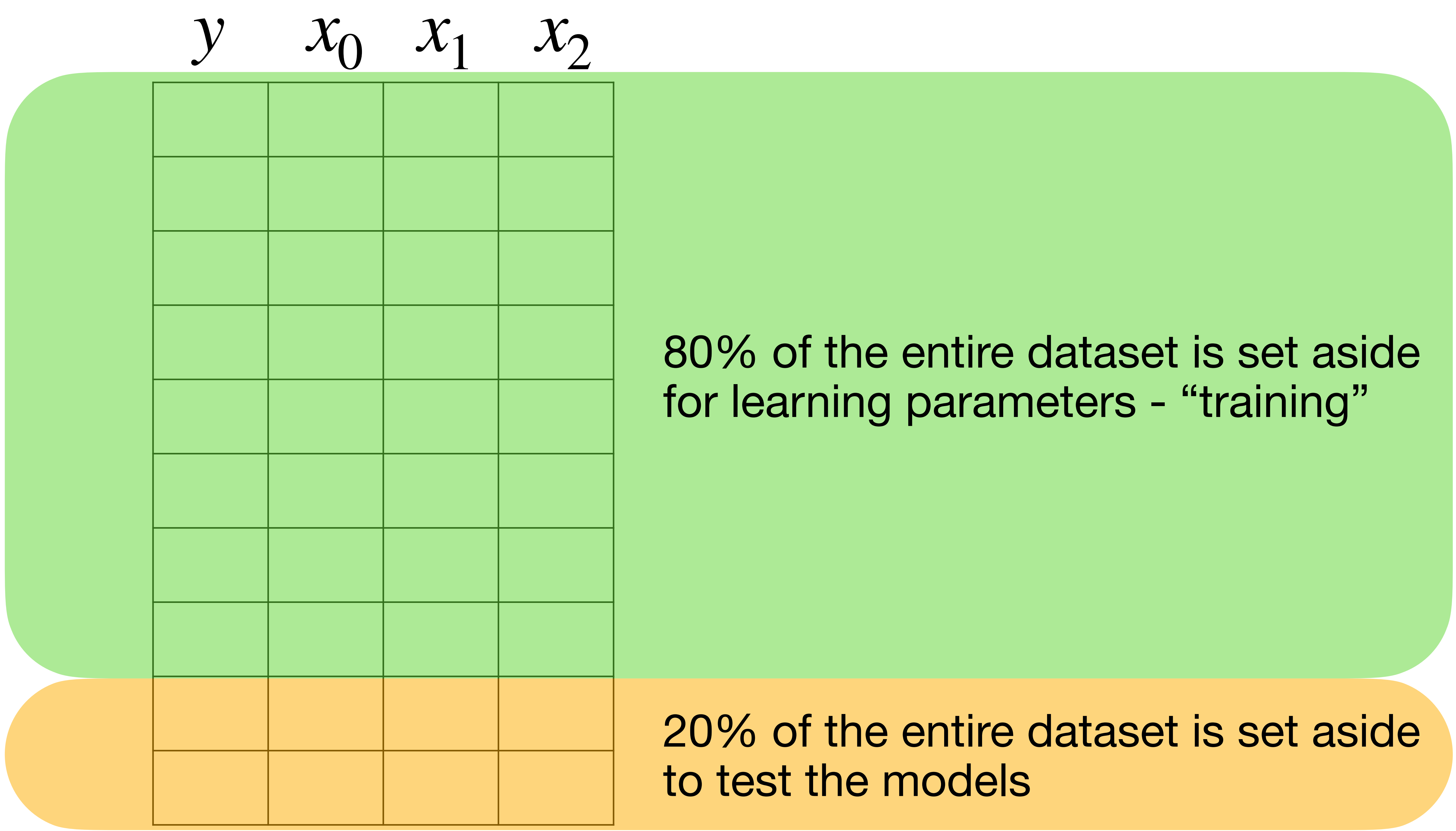
Train / Test Splits

- Generally data is split into a training dataset and a testing data
- Rough rule of thumb is that this is an 80-20 split



Train / Test Splits

- Generally data is split into a training dataset and a testing data
- Rough rule of thumb is that this is an 80-20 split



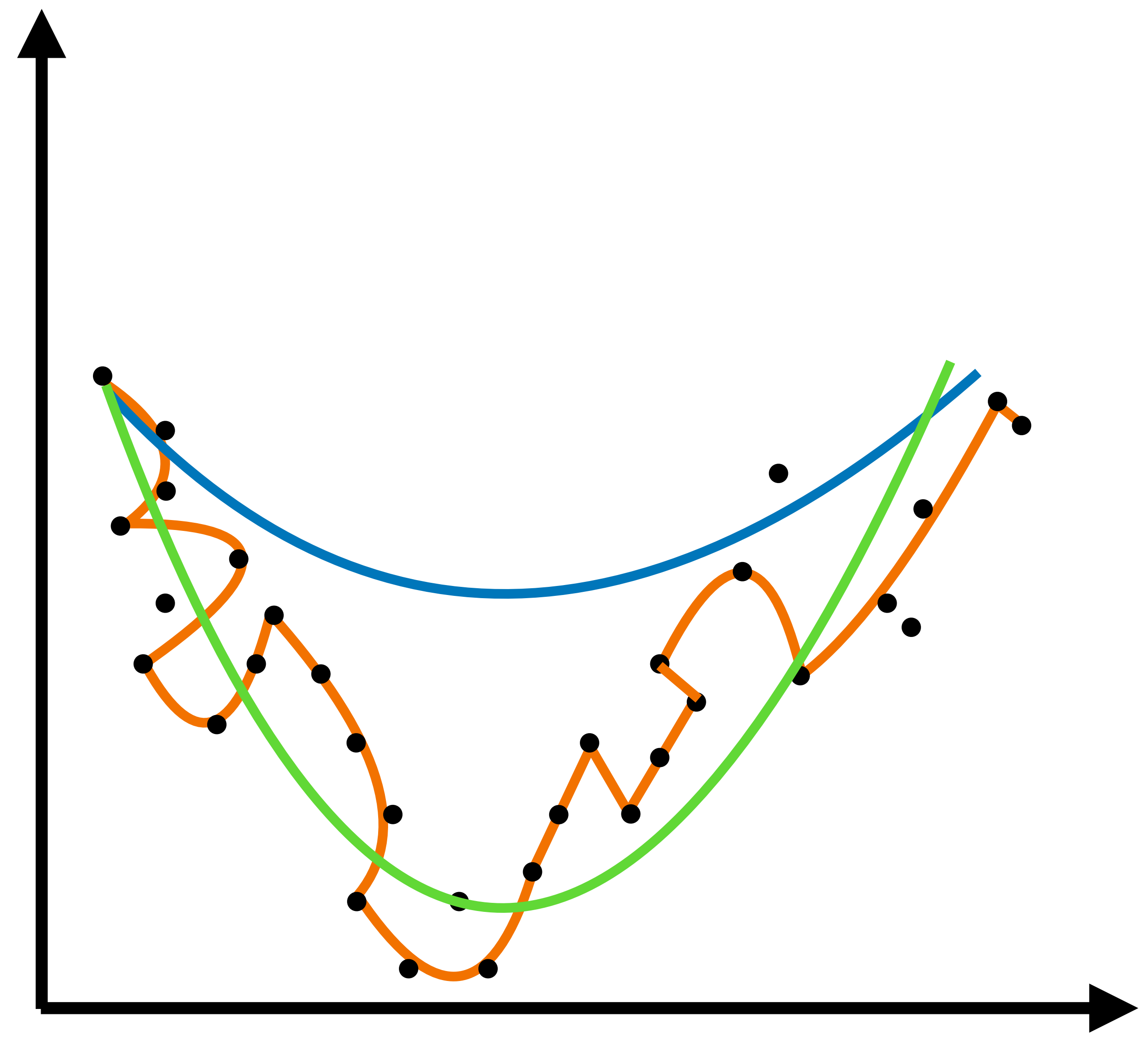
This is **unseen** data and tells you if the model can generalize well

Train / Test Splits

- However, in practice, if you are given only one train and test set, its easy to accidentally pick model architectures that work well on the test set, even though test set data is unseen
- To counter this, we use two unseen datasets - “validation” set and “test” set
- The split is generally of the form 80-10-10 where 80% is training data, 10% is validation data and 10% is test data

Practical Issues in Linear Regression

Overfitting vs Underfitting



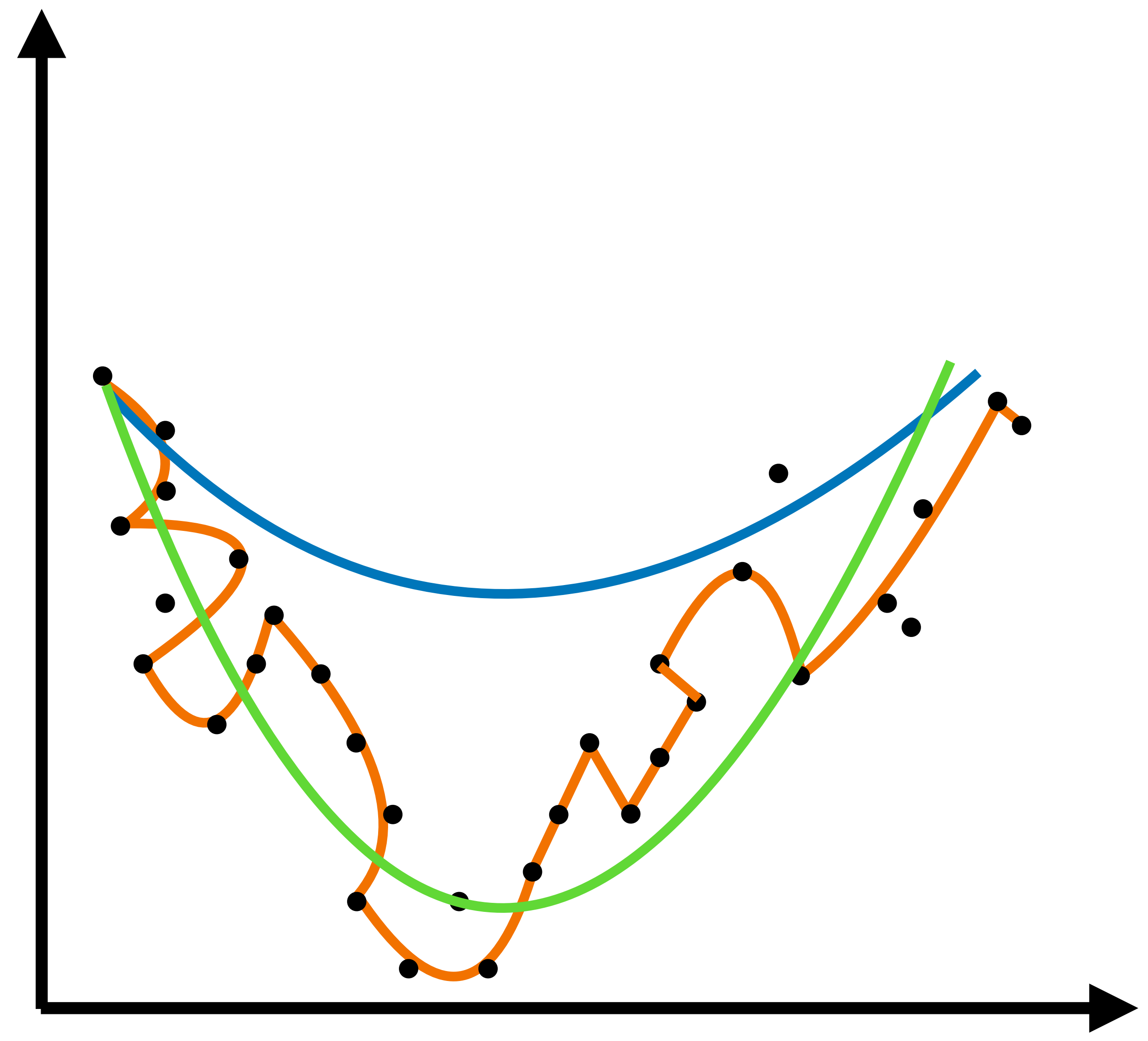
Practical Issues in Linear Regression

Overfitting vs Underfitting

The blue model is **underfitting** the data

The orange model is **overfitting** the data

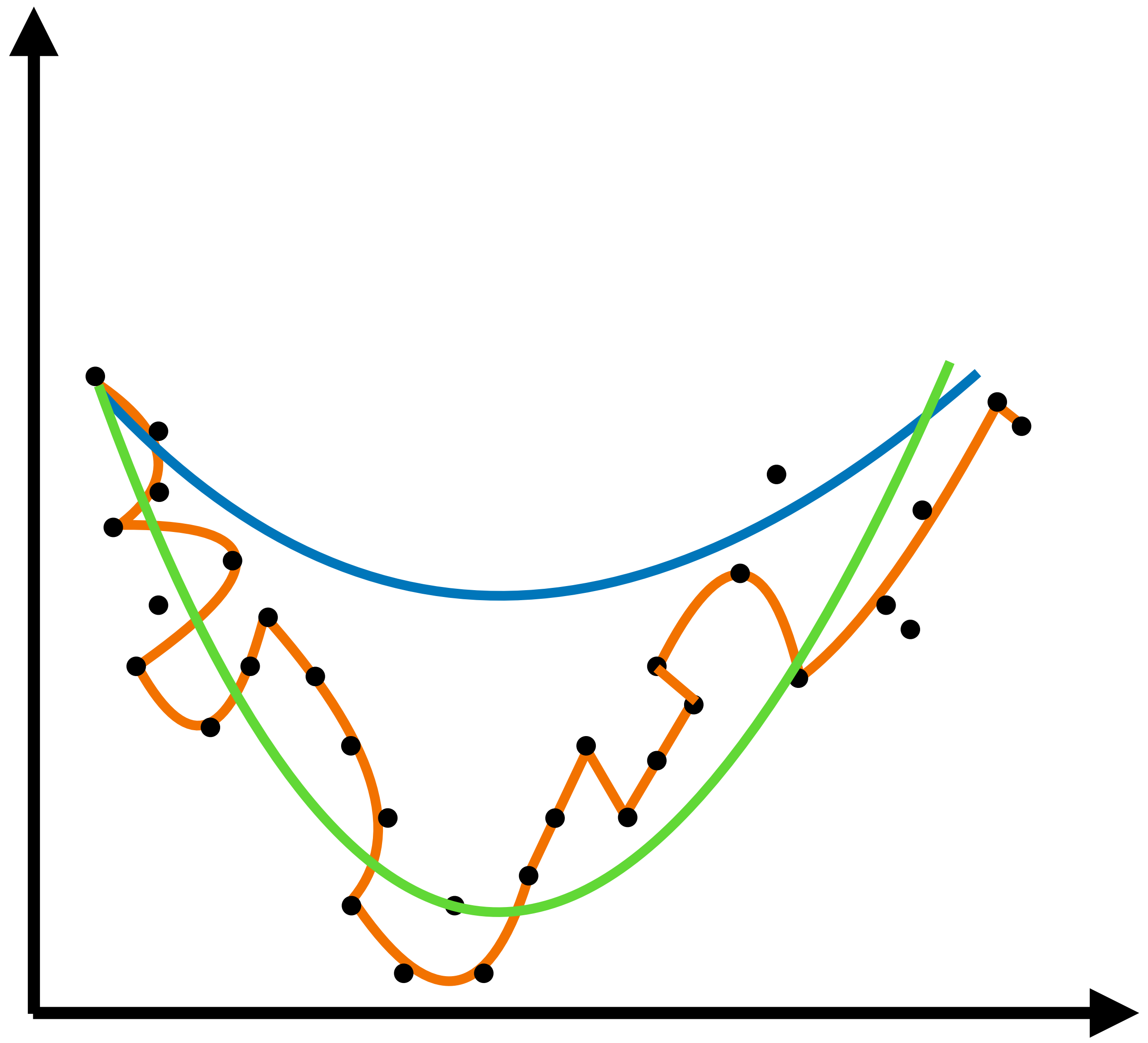
The green model is a good fit of the data



Practical Issues in Linear Regression

Underfitting

- What is happening?
    - The model is too simple to be able to capture the data
- How do you identify it?
    - Training loss is **high**
    - Test loss is **high**
- Solutions
    - Add more features
    - Add polynomial features $(x_1^2, x_2^2, x_1x_2, \cdots)$
    - Use a more complex model

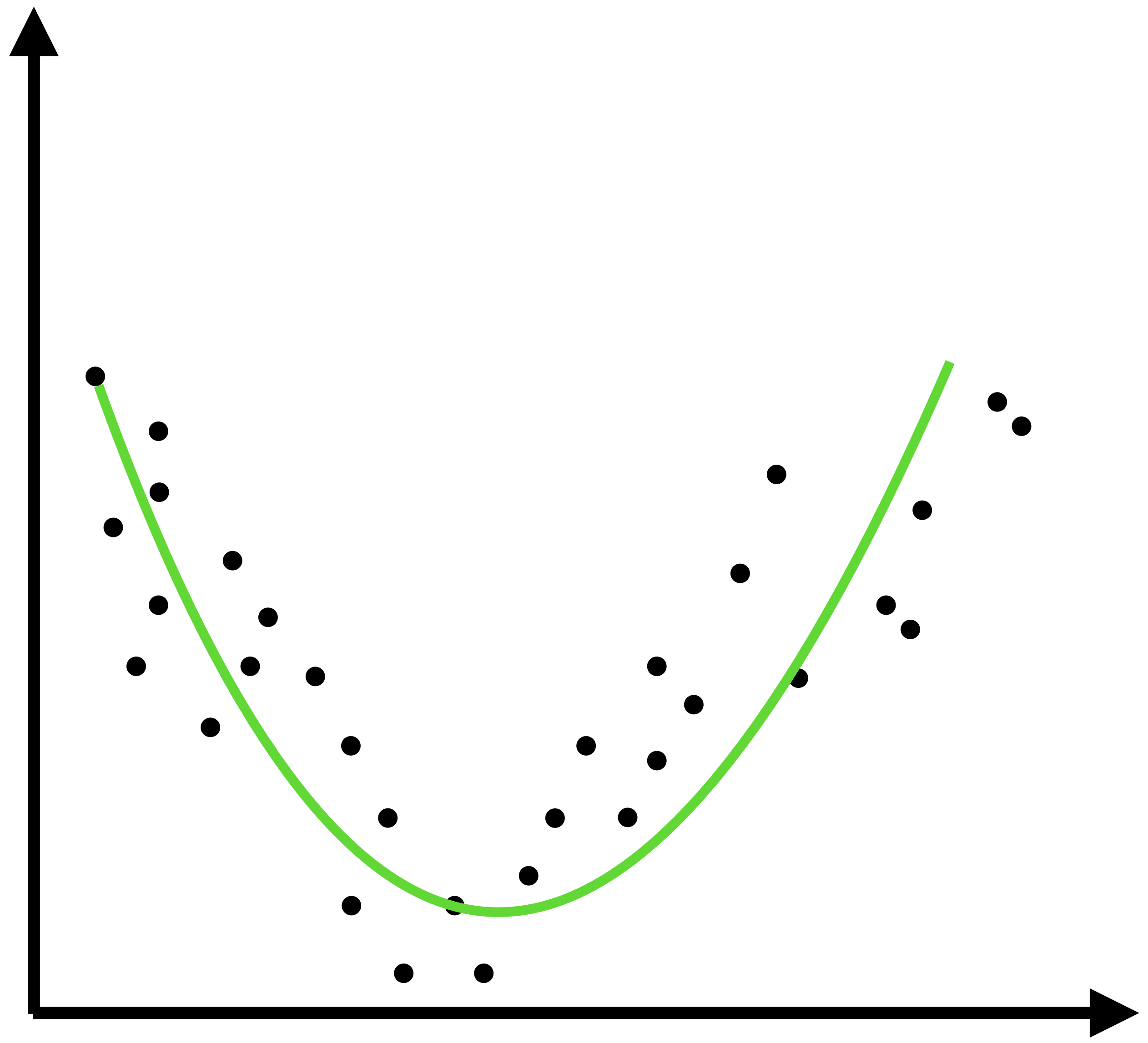


Practical Issues in Linear Regression

Quick Aside

- Add polynomial features $(x_1^2, x_2^2, x_1x_2, \cdots)$

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$

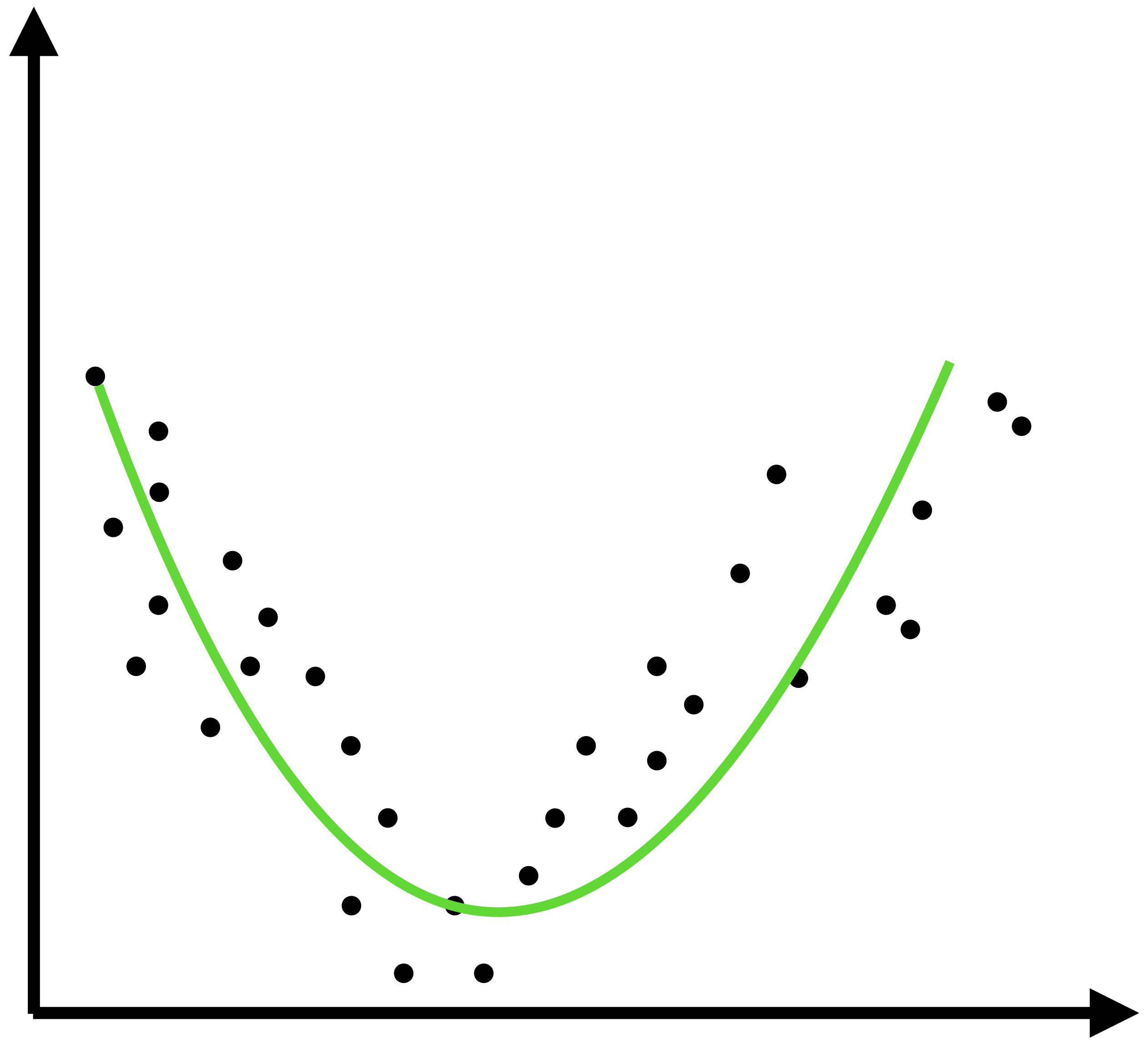


Practical Issues in Linear Regression

Quick Aside

- Add polynomial features $(x_1^2, x_2^2, x_1x_2, \cdots)$

$$f_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_1^2$$



What about these models?

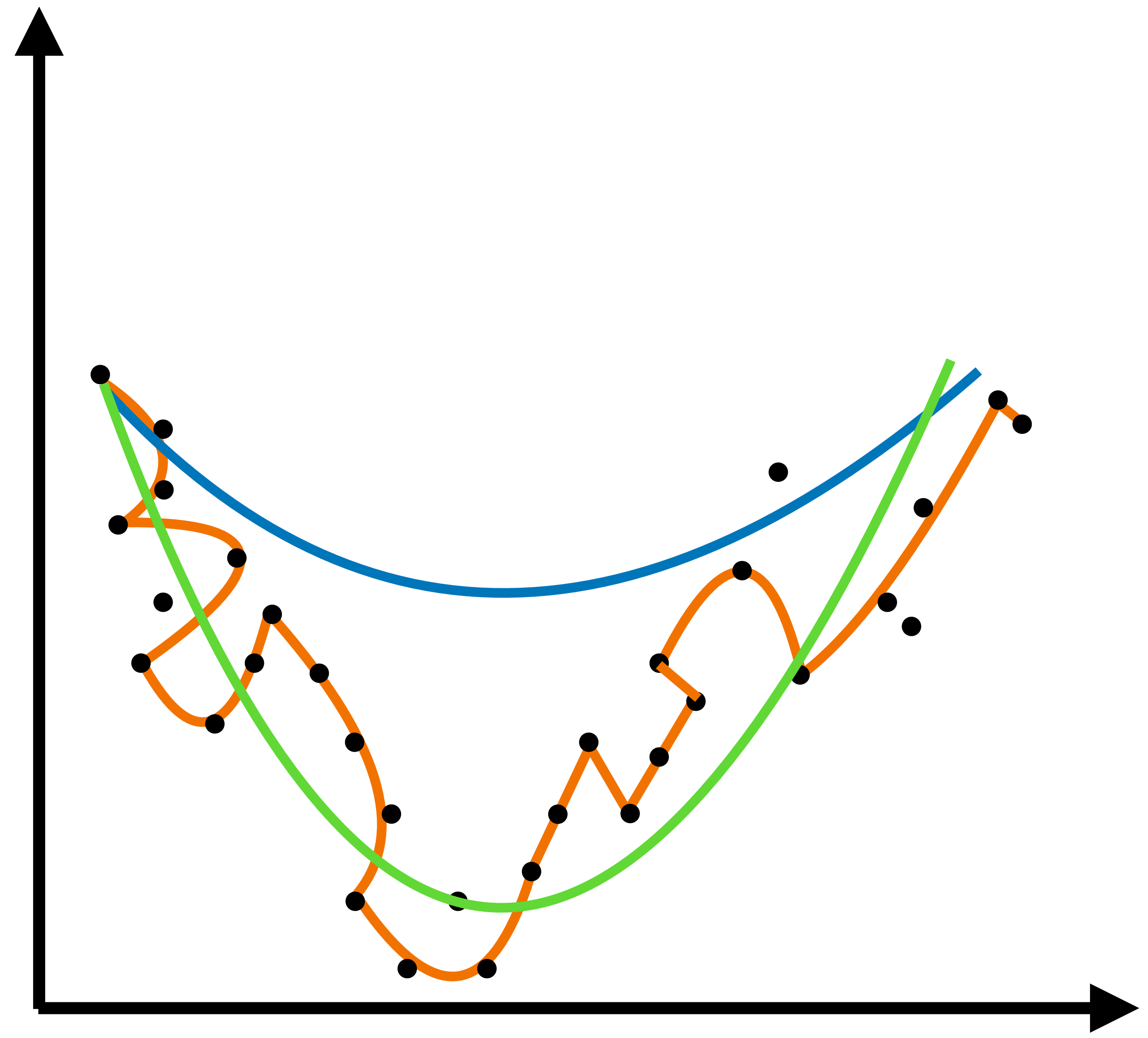
$$f_\theta(x) = \theta_0^{x_0} + \theta_1^{x_1}$$

$$f_\theta(x) = x_0^{\theta_0} + x_1^{\theta_1}$$

Practical Issues in Linear Regression

Overfitting

- What is happening?
    - The model is too complex, so it learns the noise distribution and outliers and hence does not generalize well to new data points
- How do you identify it?
    - Training loss is **low**
    - Test loss is **high**
    - Coefficients have **large** magnitudes
- Solutions
    - Regularization (L_1, L_2)
    - Cross-validation for model selection
    - Reduce number of features
    - Get more training data



Practical Issues in Linear Regression

A more mathematical look - Bias / Variance Tradeoff

Every model's prediction error/loss can be decomposed into three parts:

$$\text{Expected Loss} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

Practical Issues in Linear Regression

A more mathematical look - Bias / Variance Tradeoff

Every model's prediction error/loss can be decomposed into three parts:

$$\text{Expected Loss} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

Error from wrong assumptions due to the model being too simple

Practical Issues in Linear Regression

A more mathematical look - Bias / Variance Tradeoff

Every model's prediction error/loss can be decomposed into three parts:

Expected Loss = Bias² + Variance + Irreducible Noise

Error from high sensitivity to each data point and noise due to the model being too complex

Practical Issues in Linear Regression

A more mathematical look - Bias / Variance Tradeoff

Every model's prediction error/loss can be decomposed into three parts:

$$\text{Expected Loss} = \text{Bias}^2 + \text{Variance} + \text{Irreducible Noise}$$

Inherent randomness in data. Cannot be removed.

Practical Issues in Linear Regression

Bias / Variance Tradeoff

Why is it called a **tradeoff**?

Practical Issues in Linear Regression

Bias / Variance Tradeoff

Why is it called a **tradeoff**?

Model Complexity	Bias	Variance	Train Error	Test Error
Too Simple	High	Low	High	High

Practical Issues in Linear Regression

Bias / Variance Tradeoff

Why is it called a **tradeoff**?

Model Complexity	Bias	Variance	Train Error	Test Error
Too Simple	High	Low	High	High
Too Complex	Low	High	Low	High

Practical Issues in Linear Regression

Bias / Variance Tradeoff

Why is it called a **tradeoff**?

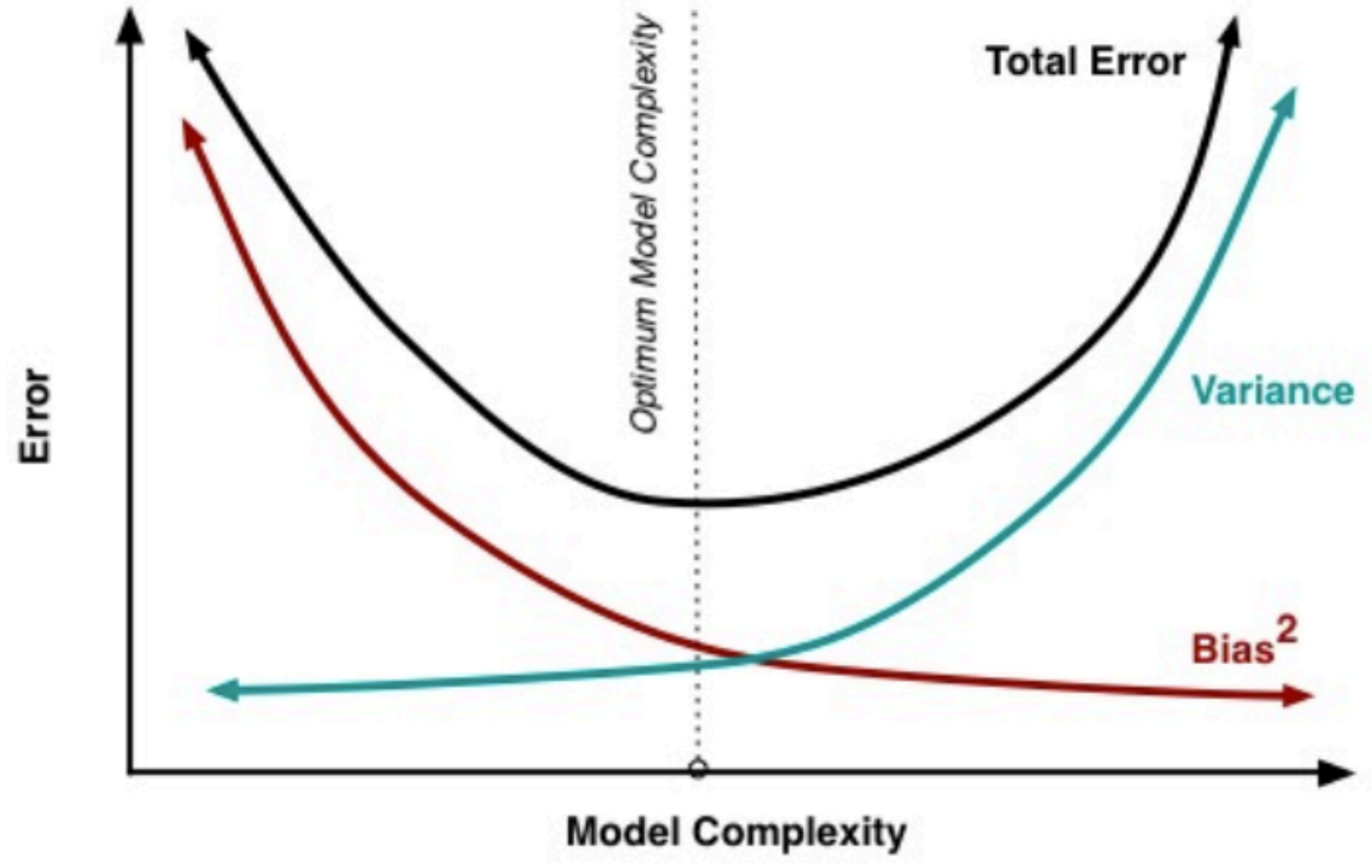
Model Complexity	Bias	Variance	Train Error	Test Error
Too Simple	High	Low	High	High
Sweet Spot	Medium	Medium	Medium	Medium
Too Complex	Low	High	Low	High

Practical Issues in Linear Regression

Bias / Variance Tradeoff

Why is it called a **tradeoff**?

Model Complexity	Bias	Variance	Train Error	Test Error
Too Simple	High	Low	High	High
Sweet Spot	Medium	Medium	Medium	Medium
Too Complex	Low	High	Low	High

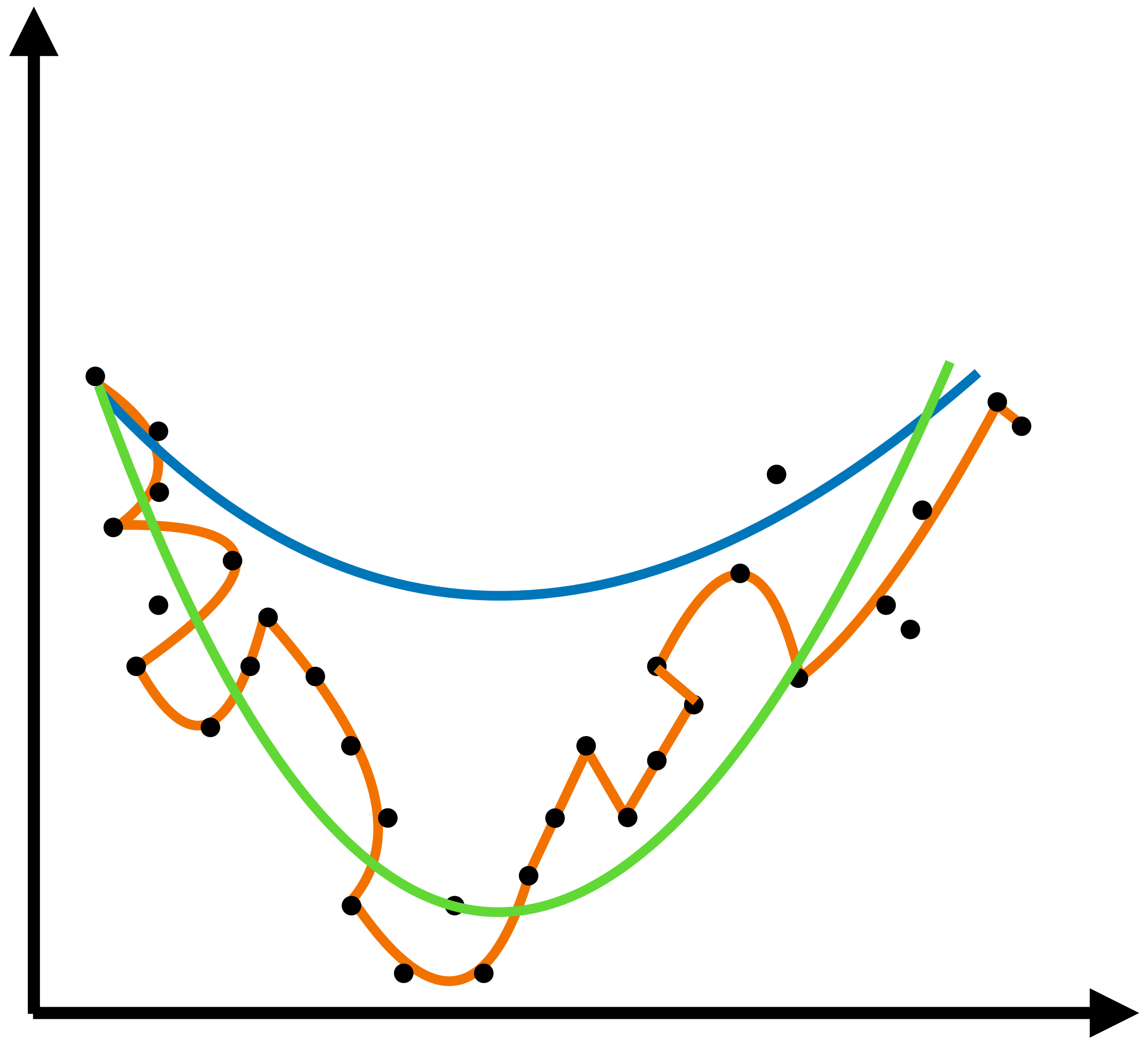


Practical Issues in Linear Regression

Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m} \sum (Y - X\theta)^2$$



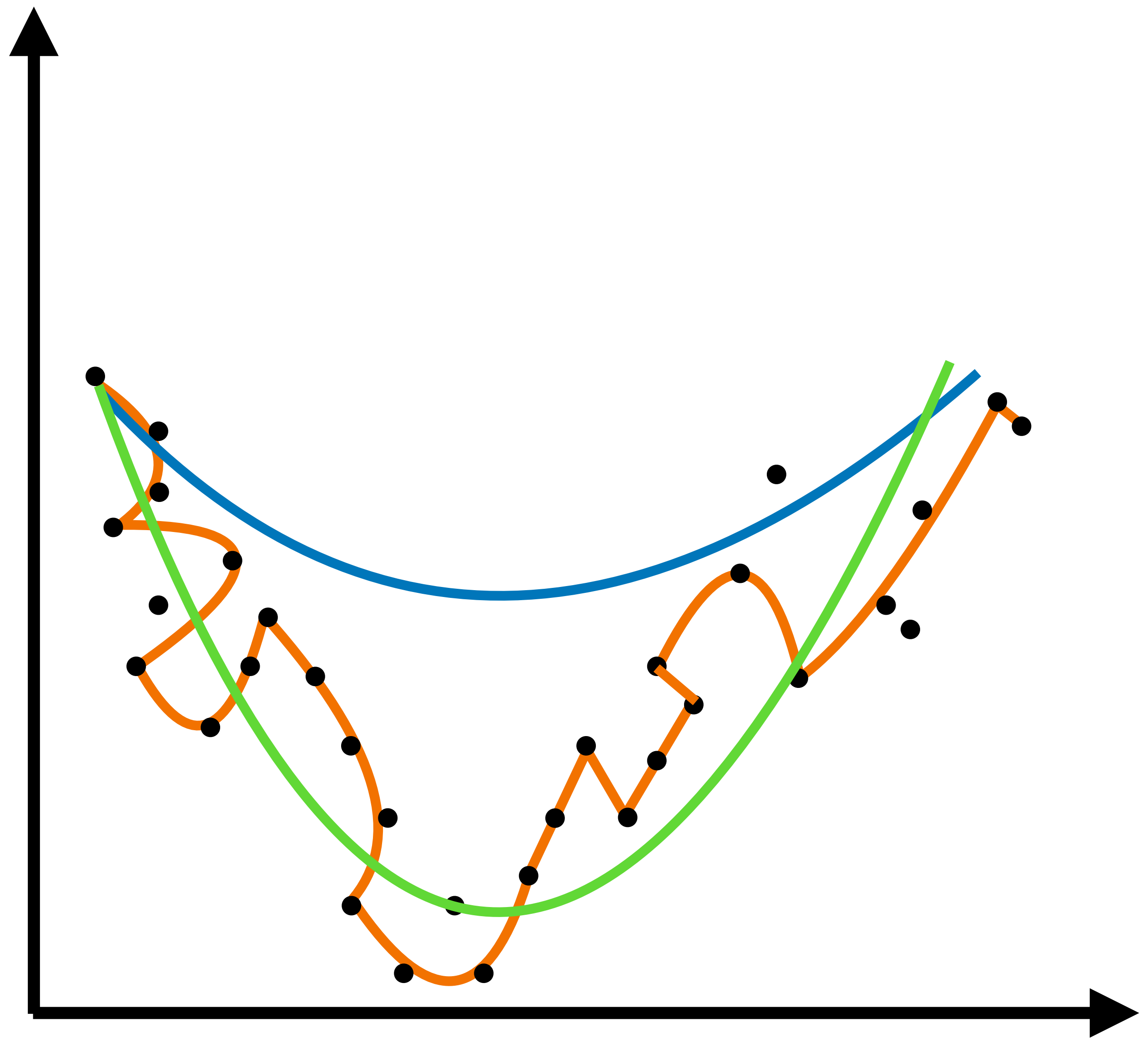
Practical Issues in Linear Regression

Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m}\sum (Y - X\theta)^2$$

$$L(\theta) = \frac{1}{m}\sum (Y - X\theta)^2 + \lambda||\theta||^2$$



Practical Issues in Linear Regression

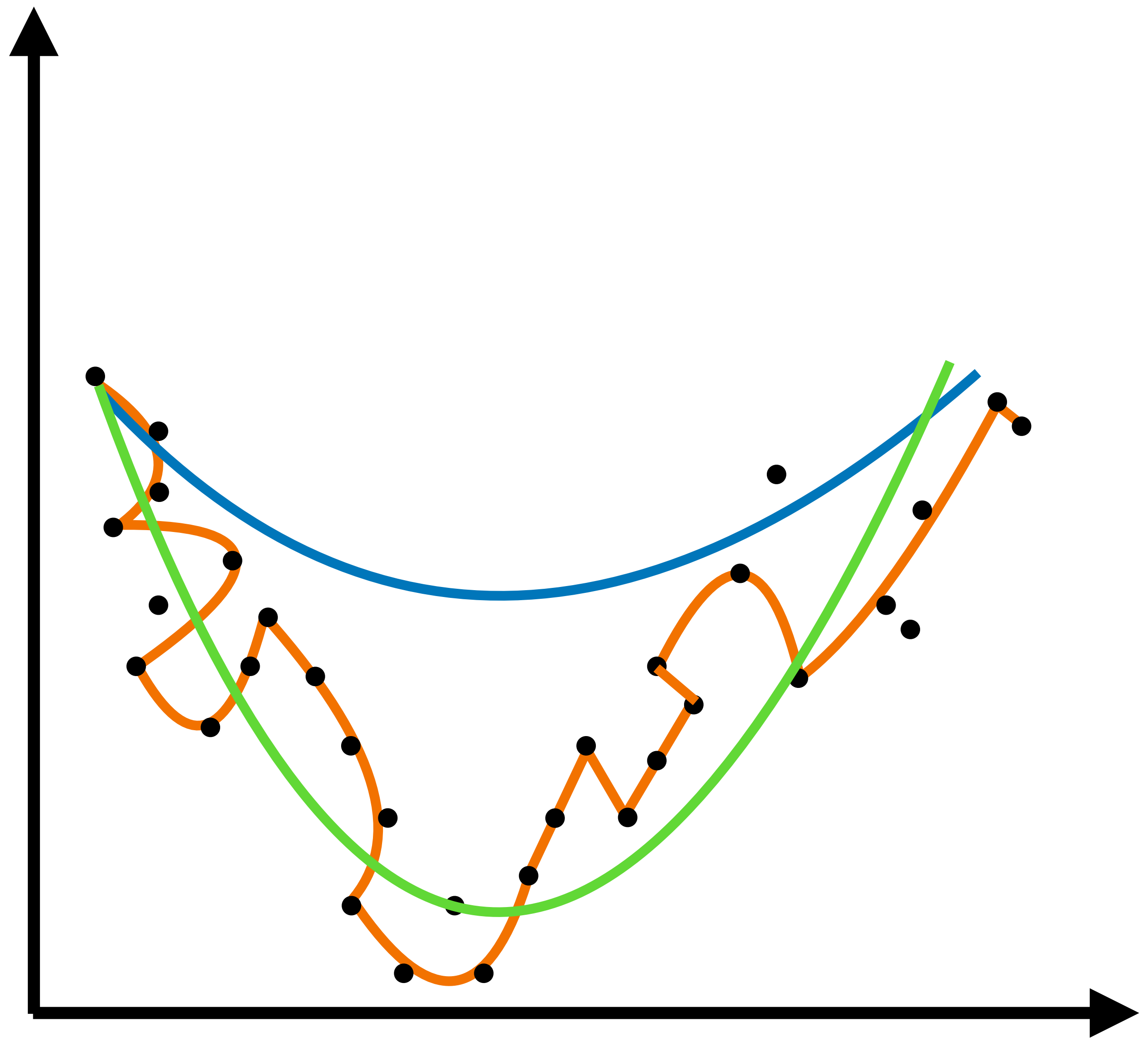
Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m}\sum(Y - X\theta)^2$$

$$L(\theta) = \frac{1}{m}\sum(Y - X\theta)^2 + \lambda||\theta||^2$$

$$\theta = (X^T X + \lambda I)^{-1} X^T Y$$



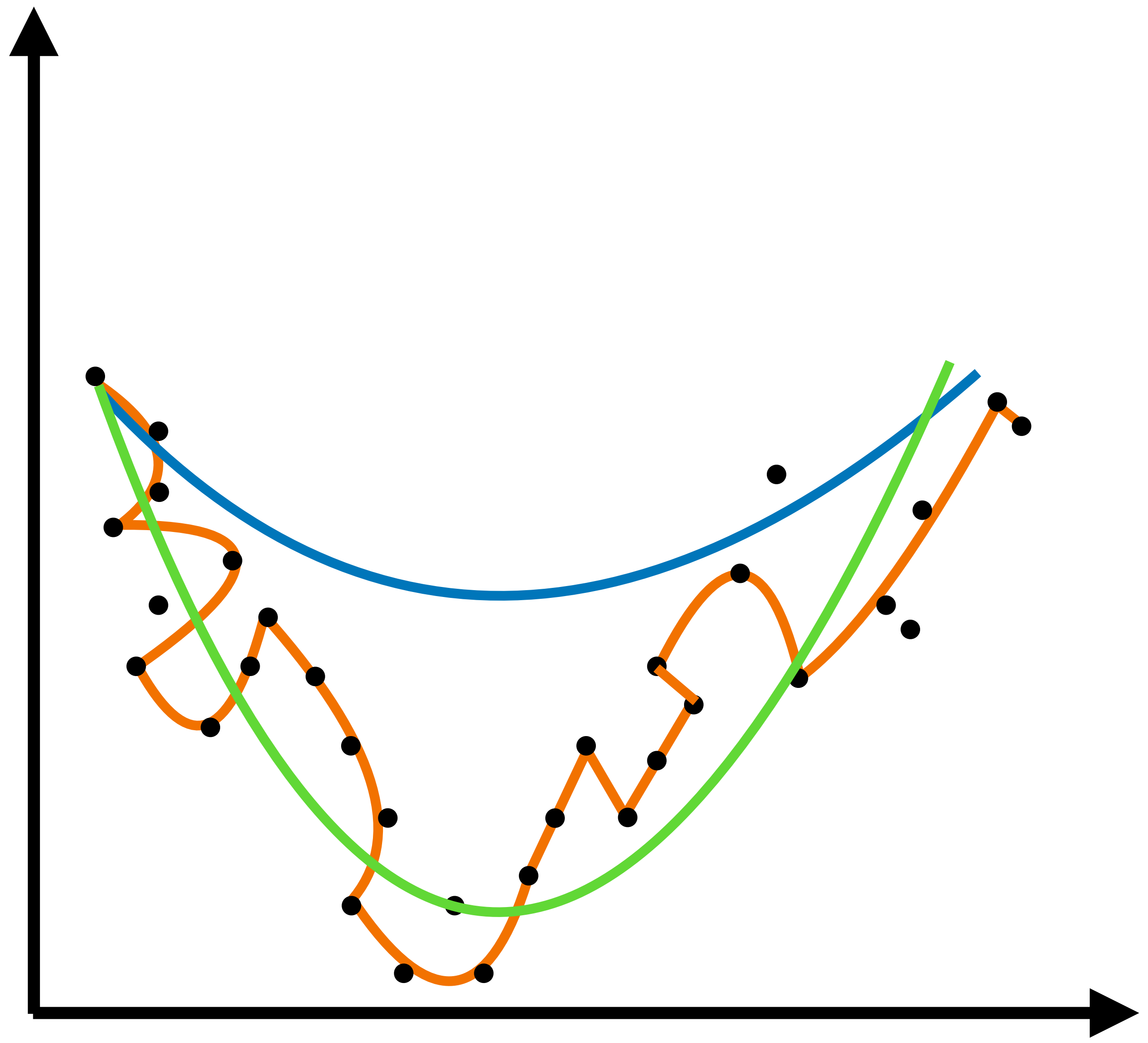
Practical Issues in Linear Regression

Regularization

- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m}\sum (Y - X\theta)^2 + \lambda \|\theta\|^2$$

$$\theta = (X^T X + \lambda I)^{-1} X^T Y$$



- As λ increases:
  - Coefficients shrink toward zero
  - Bias increases (we're constraining the model)
  - Variance decreases (less sensitive to data)
  - At some $\lambda *$, test error is minimized

Practical Issues in Linear Regression

Regularization

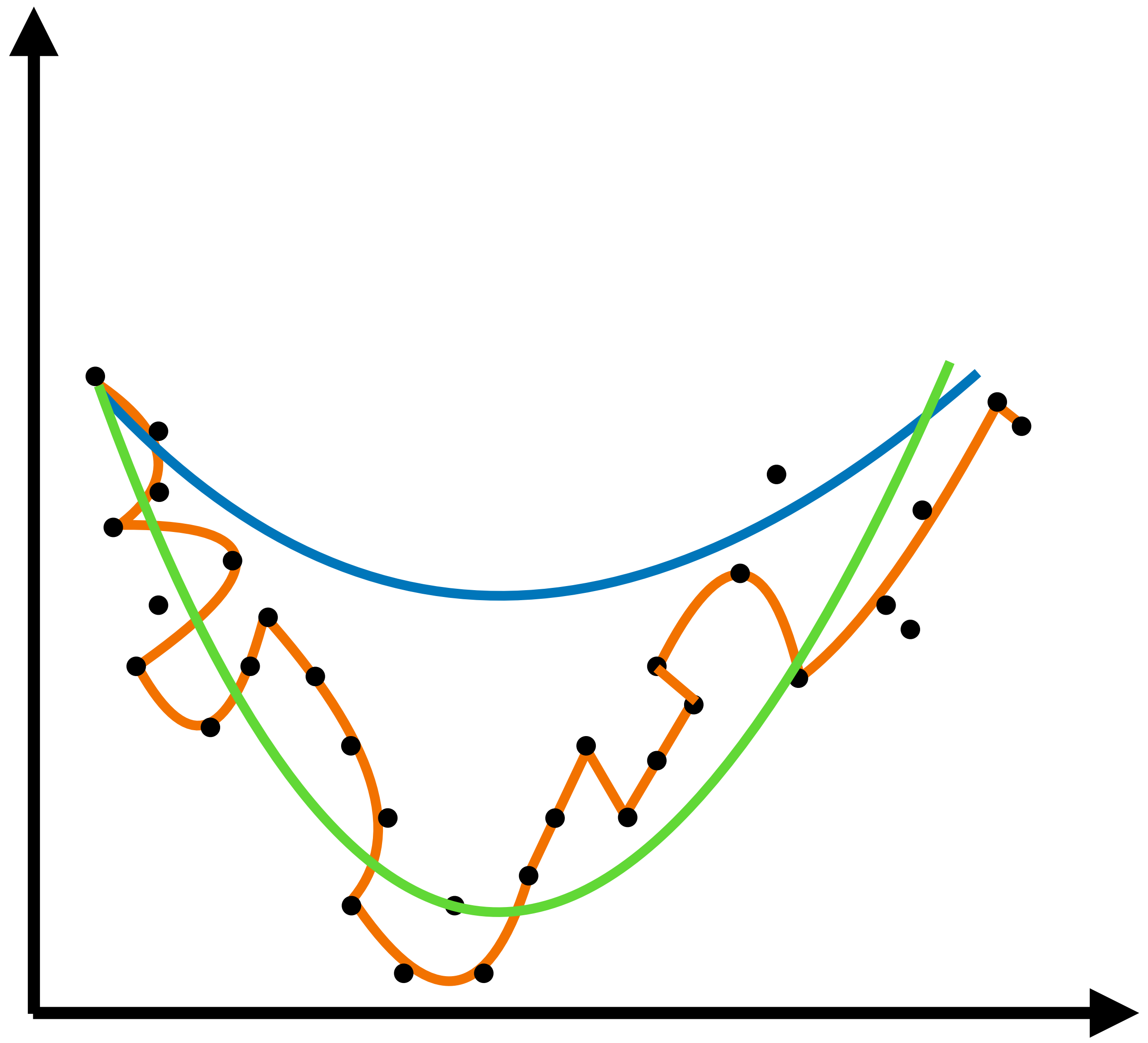
- Regularization explicitly trades bias for variance.

$$L(\theta) = \frac{1}{m}\sum(Y - X\theta)^2 + \lambda||\theta||^2$$

$$\theta = (X^T X + \lambda I)^{-1} X^T Y$$

These sort of parameters are usually called **hyper-parameters**

They are **not learnable** but are human defined



- As λ increases:
  - Coefficients shrink toward zero
  - Bias increases (we're constraining the model)
  - Variance decreases (less sensitive to data)
  - At some $\lambda*$, test error is minimized

Feature Normalization

Why Normalize?

- If feature x_1 ranges from 0 to 1 and feature x_2 ranges from 0 to 1,000,000, this could lead to numerical instability in the solving process
  - This is particular relevant to gradient descent
- Regularization unfairness
  - If x_2 is much larger, θ_2 must be much smaller to produce similar predictions.
  - The regularization penalty then affects features unequally based on arbitrary scale choices.
- Distance-based algorithms

Feature Normalization

Normalization Methods

1. Min-Max Normalization
2. Mean-Variance Normalization
3. Max-Absolute Normalization
4. Robust Normalization

Feature Normalization

Min-Max Normalization

- For every column in the input data, i.e., for each x_0, x_1, x_2, x_4 etc., this normalization method will scale each column to 0 and 1

$$x' = \frac{x - min(x)}{max(x) - min(x)}$$

- This method preserves zero entries in sparse data

- But is very sensitive to **outliers**

Feature Normalization

Mean-Variance Normalization

- For every column in the input data, i.e., for each x_0, x_1, x_2, x_4 etc., this normalization method will scale to have mean 0 and standard deviation 1

$$x' = \frac{x - \mu(x)}{\sigma(x)}$$

- Most common in practice
- Less sensitive to outliers than min-max
- Does not bound the range to 0 and 1

Feature Normalization

Max-Absolute Normalization

- For every column in the input data, i.e., for each x_0, x_1, x_2, x_4 etc., this normalization method will scale each column to -1 and 1

$$x' = \frac{x}{|max(x)|}$$

- Good for sparse data since it preserves sparsity (zeros stay zero)

Feature Normalization

Robust Normalization

- For every column in the input data, i.e., for each x_0, x_1, x_2, x_4 etc., this normalization method will scale each column as

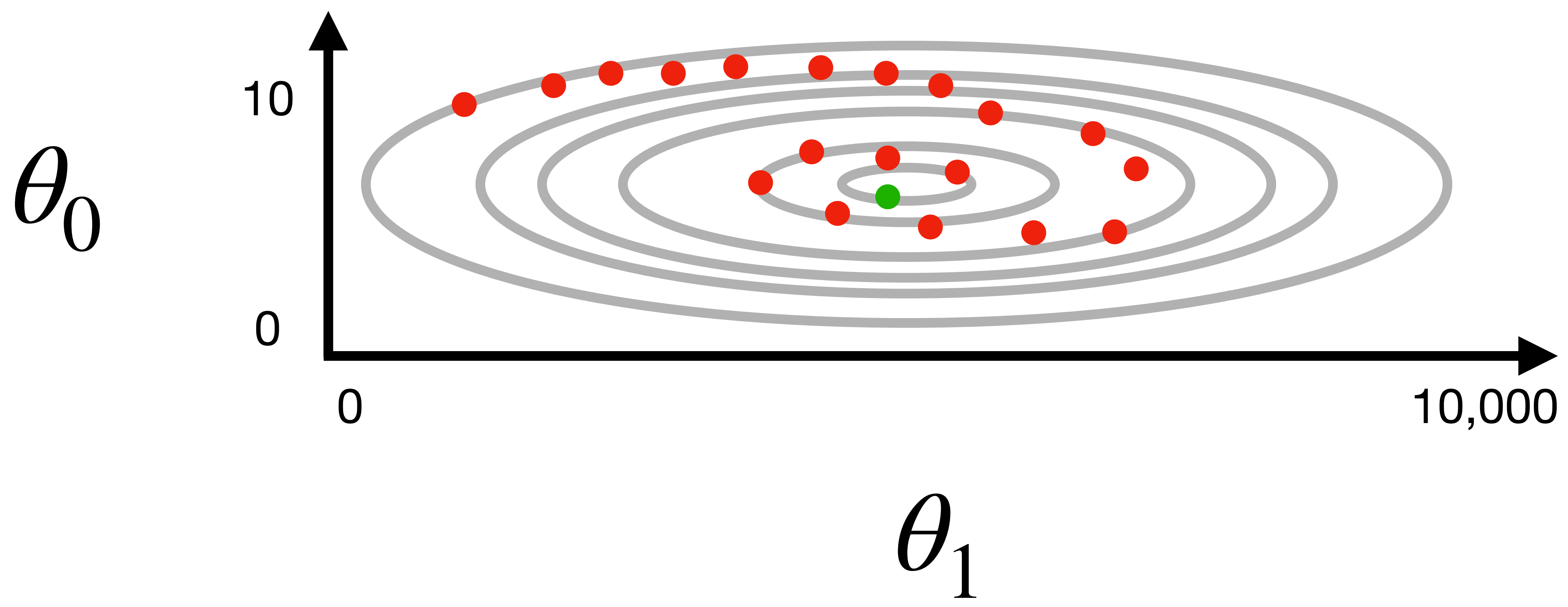
$$x' = \frac{x - median(x)}{IQR(x)}$$

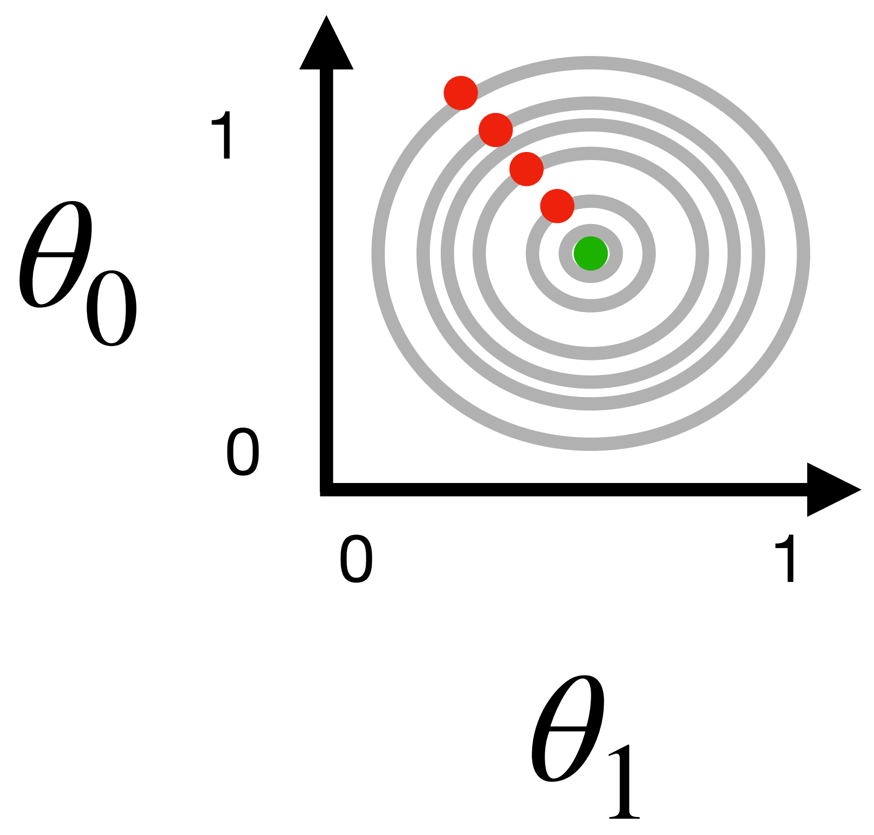
- Robust to outliers
- Use when data has many outliers

Optimizing Loss Functions

Gradient Descent - Practical Fixes

- Feature Scaling
  - Remember we want all input features $x_1, x_2 \cdots x_n$ to be in similar ranges
  - When features have different scales, the loss surface becomes elongated (ill-conditioned).





This dramatically accelerates the optimization process

This also allows having one single learning rate for all parameters

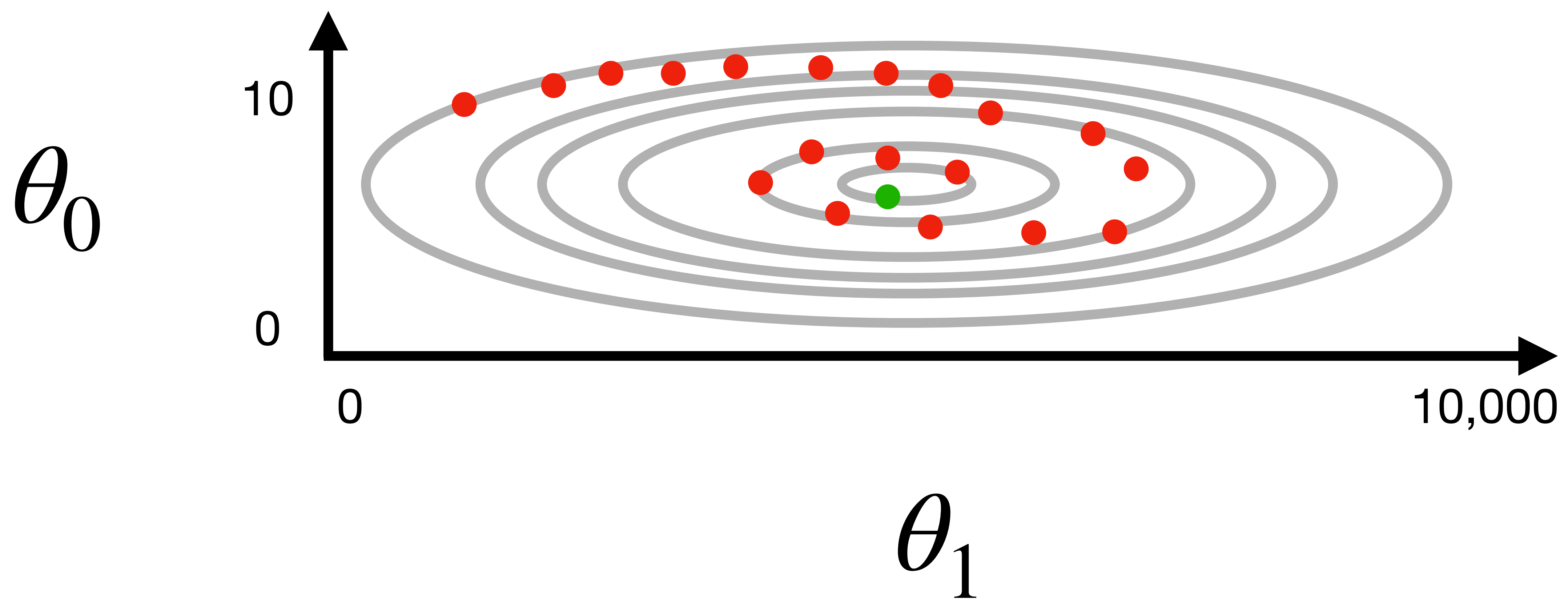
Optimizing Loss Functions

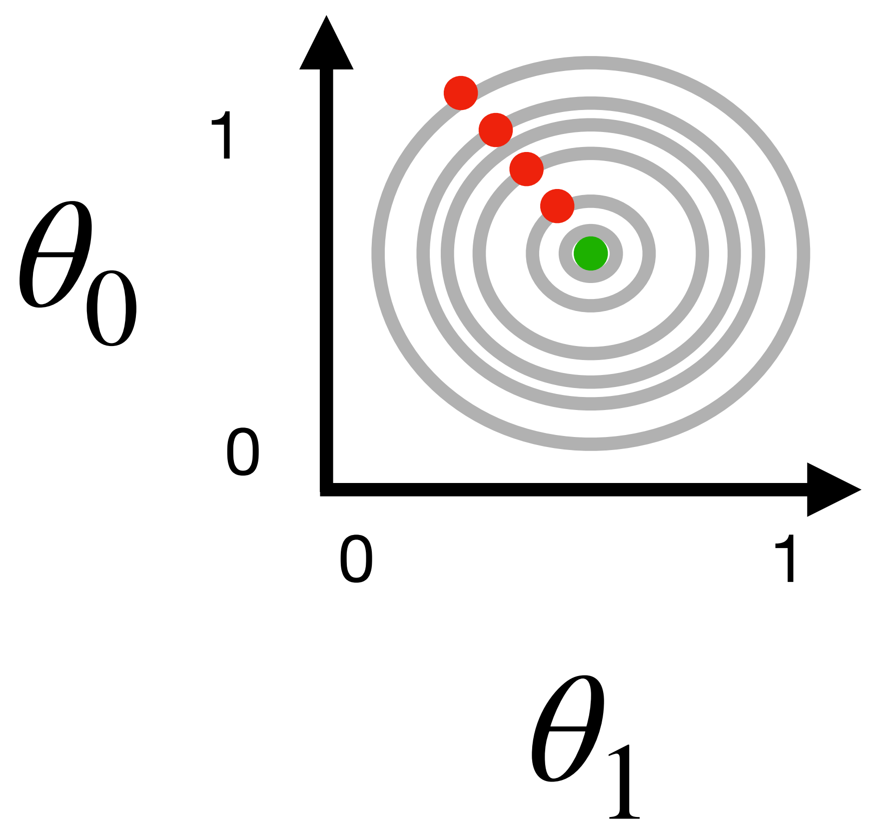
Gradient Descent - Practical Fixes

- Feature Scaling

NOTE: Scaling parameters (mean, standard deviation, min, max) must be computed **only on training data** and then applied to validation and test data to prevent data leakage.

  - Remember we want all input features $x_1, x_2 \cdots x_n$ to be in similar ranges
  - When features have different scales, the loss surface becomes elongated (ill-conditioned).





This dramatically accelerates the optimization process

This also allows having one single learning rate for all parameters

Questions