Northeastern University
Khoury College of
Computer Sciences

Gradient Descent

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

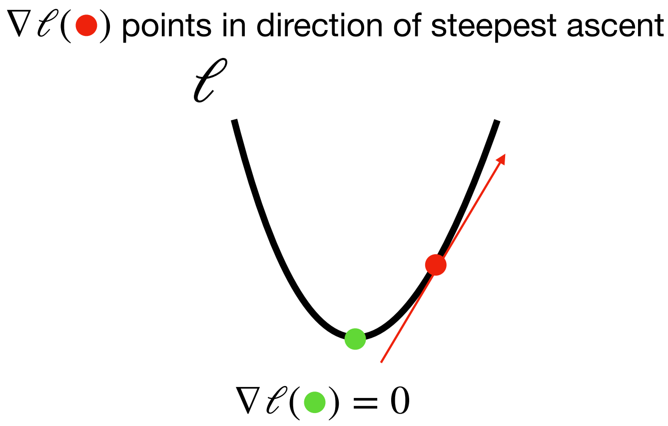
Spring 2026

Wednesday | January 21, 2026

Optimizing Loss Functions

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

$$\ell(\theta) = \frac{1}{n}\sum(y - \hat{y})^2$$



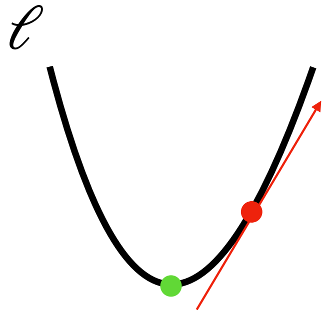
Optimizing Loss Functions

$X \in \mathbb{R}^{m \times n}$ — data, # features.

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

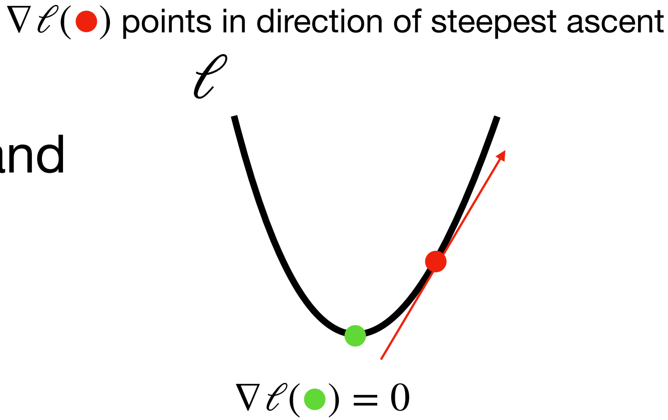
$X^T X \to n \times n$

- 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 $O(n^3)$ $\to 100^3$



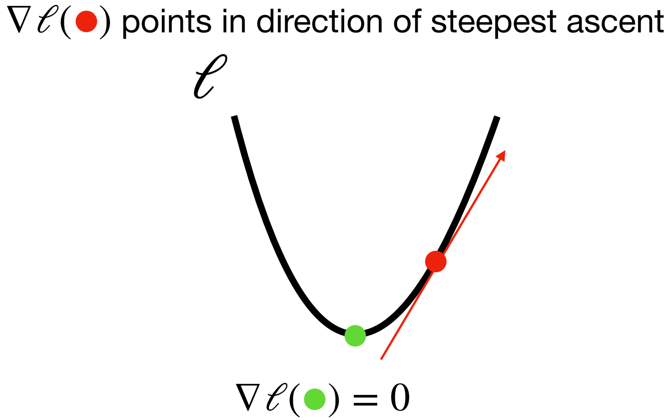
Optimizing Loss Functions

- 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)



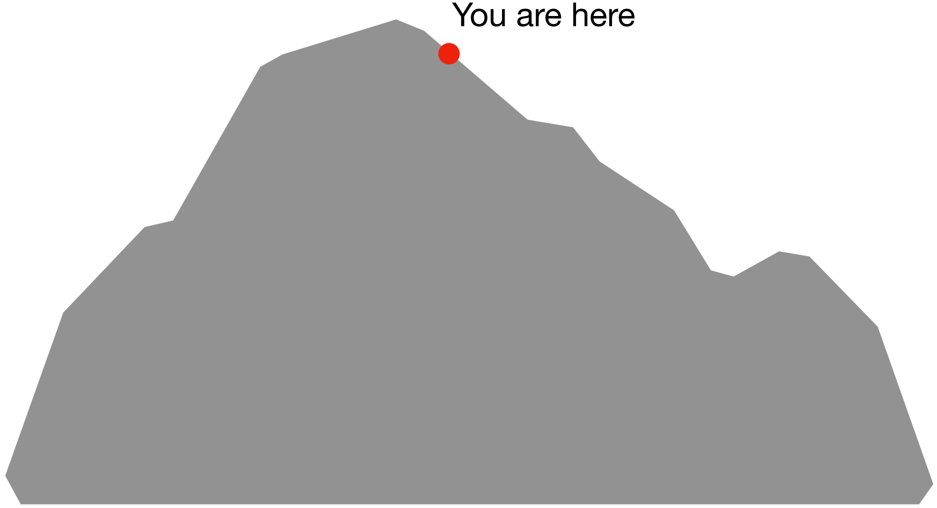
Optimizing Loss Functions

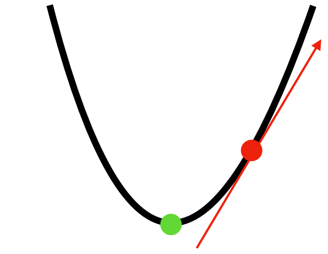
- What is gradient descent?



Optimizing Loss Functions

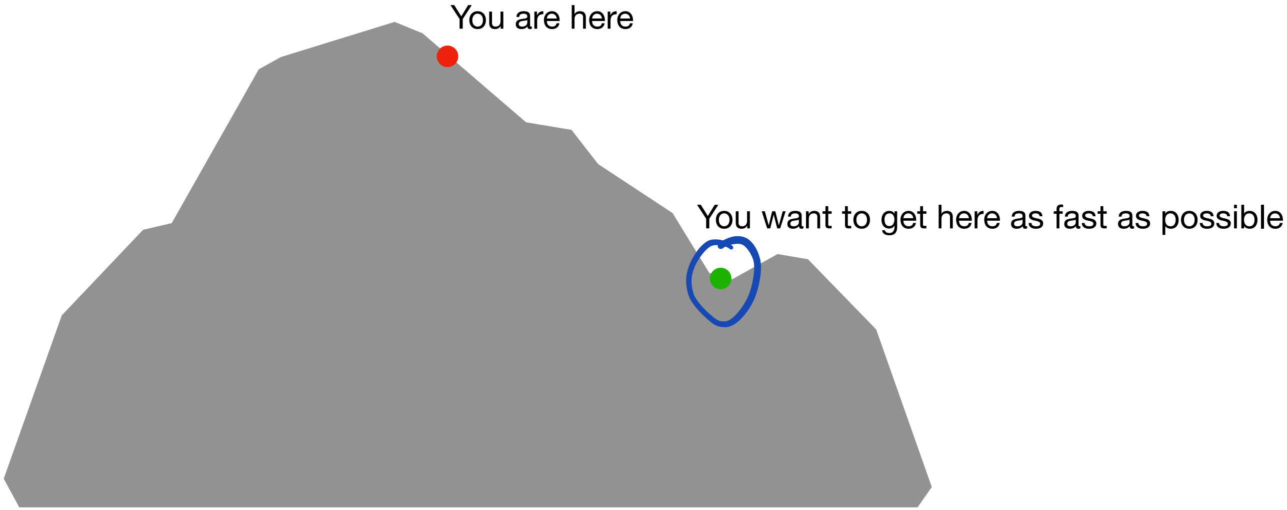
- What is gradient descent?

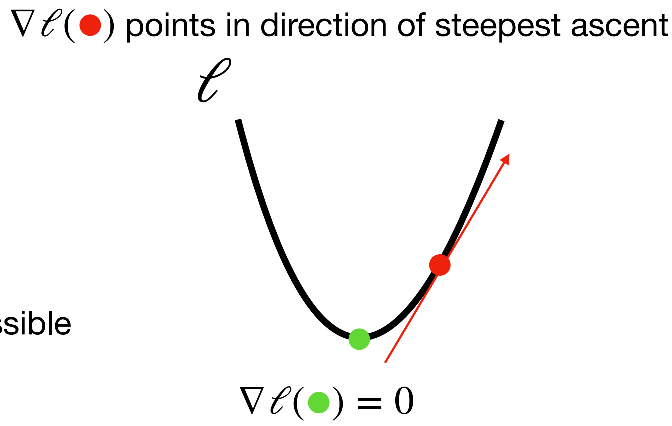




Optimizing Loss Functions

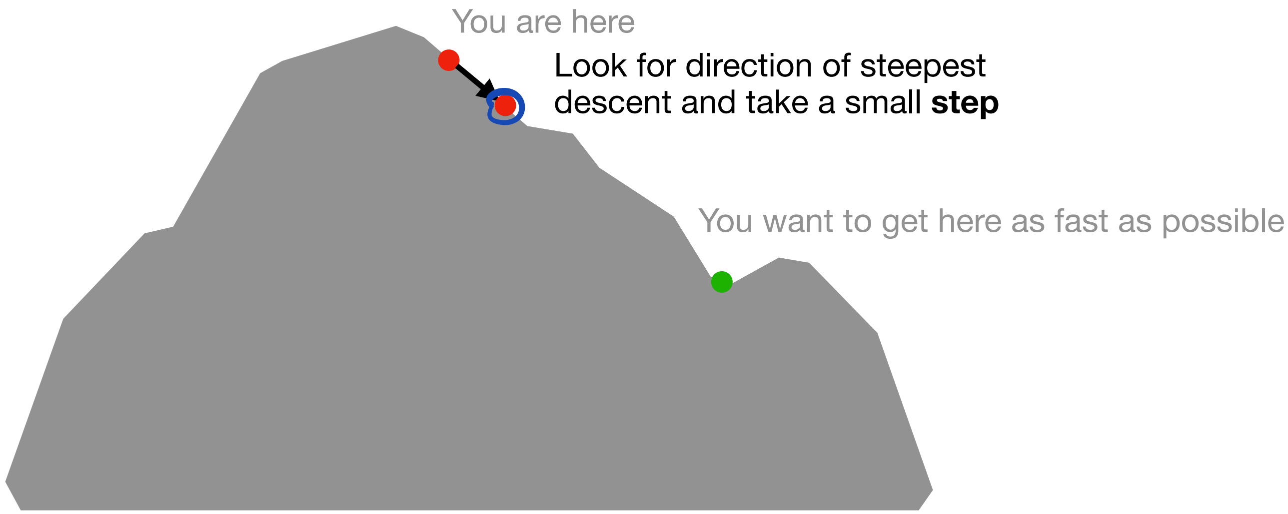
- What is gradient descent?

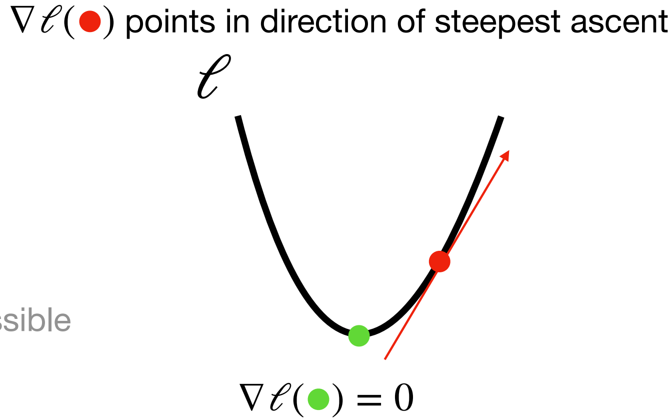




Optimizing Loss Functions

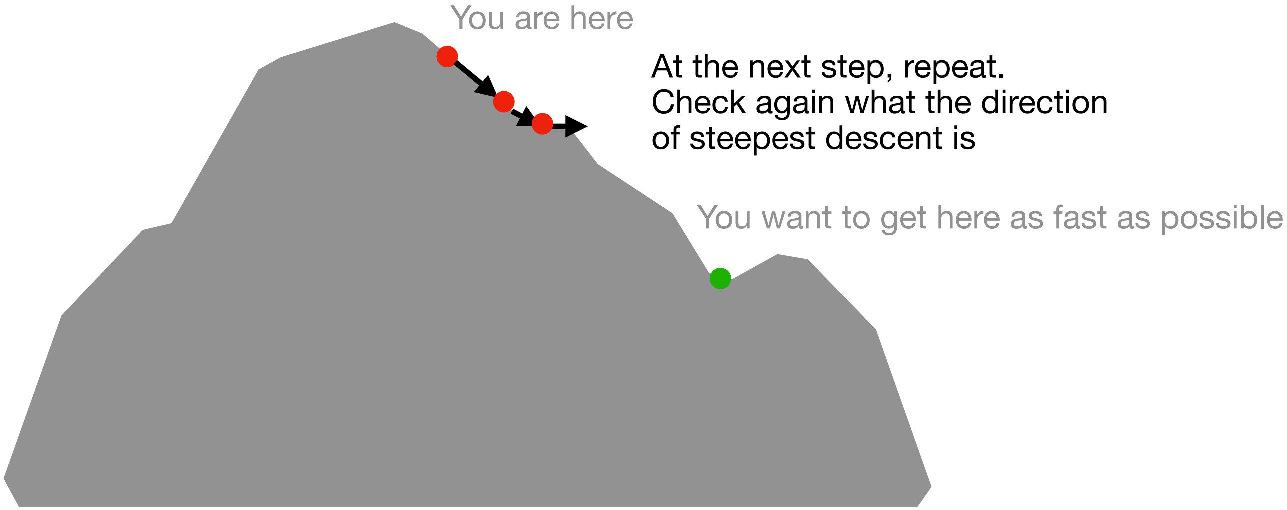
- What is gradient descent?

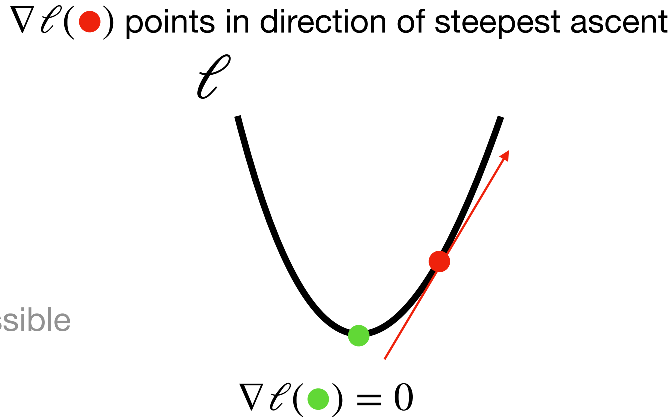




Optimizing Loss Functions

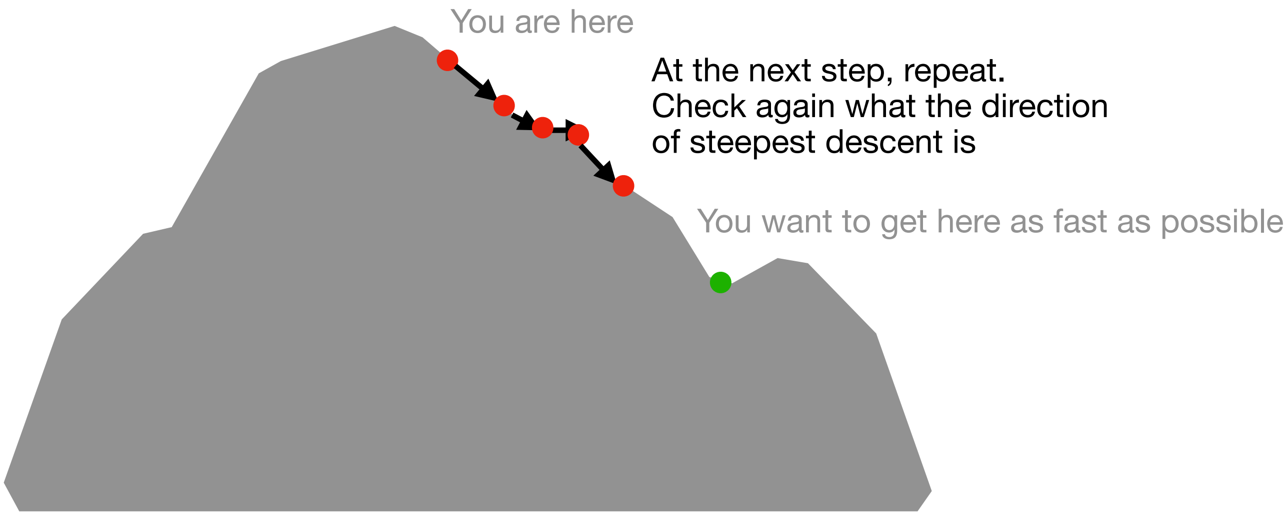
- What is gradient descent?

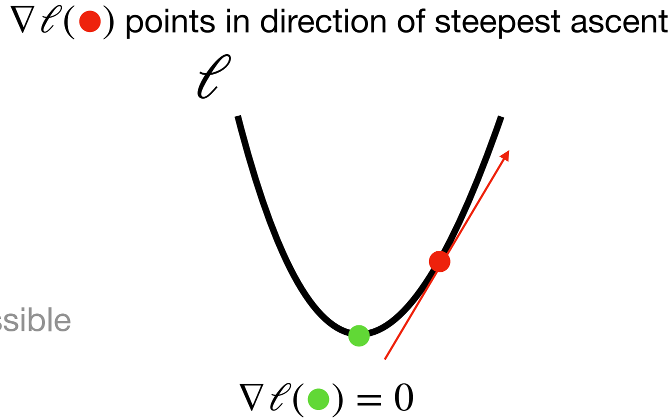




Optimizing Loss Functions

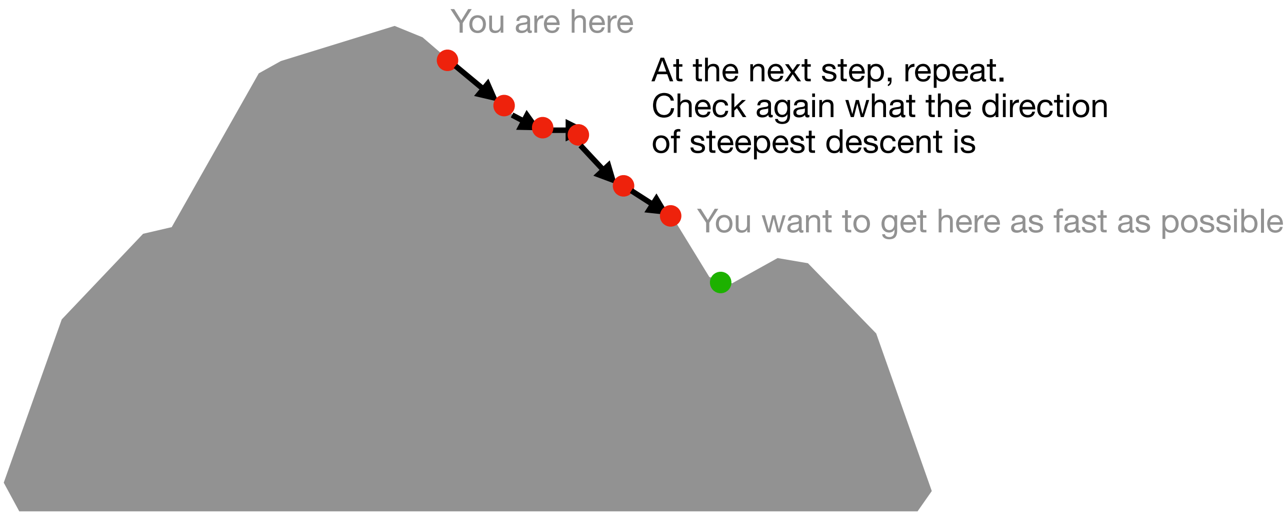
- What is gradient descent?

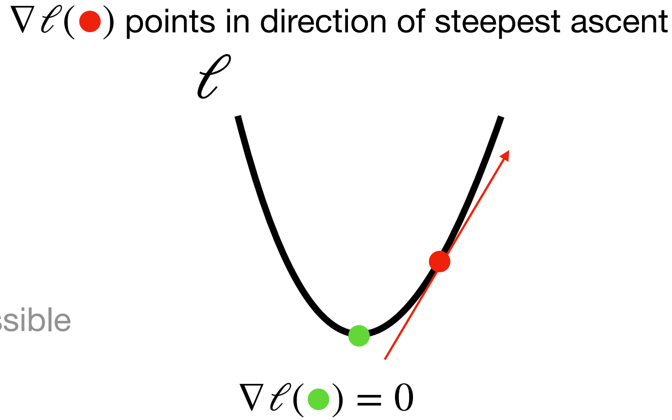




Optimizing Loss Functions

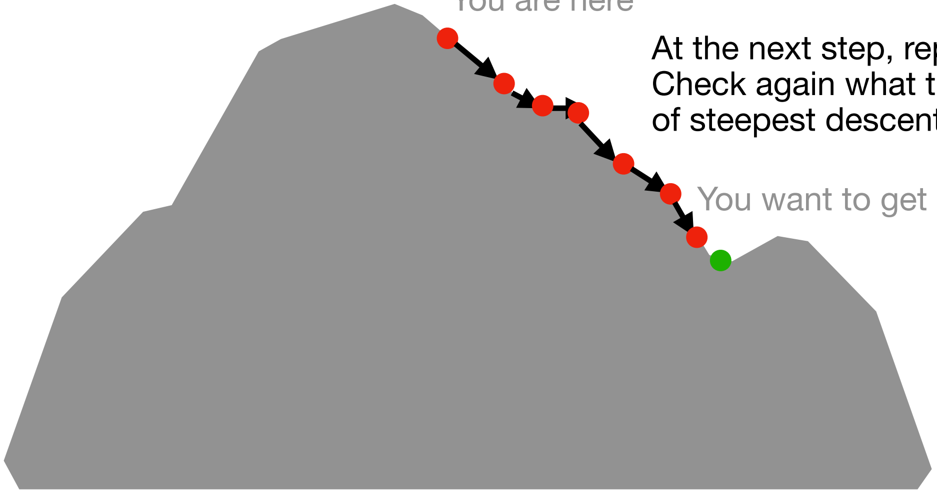
- What is gradient descent?

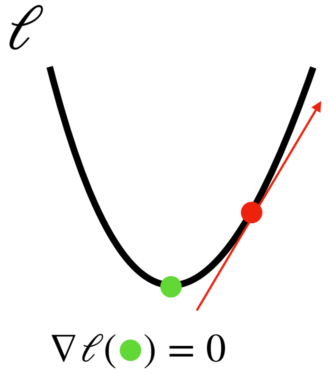




Optimizing Loss Functions

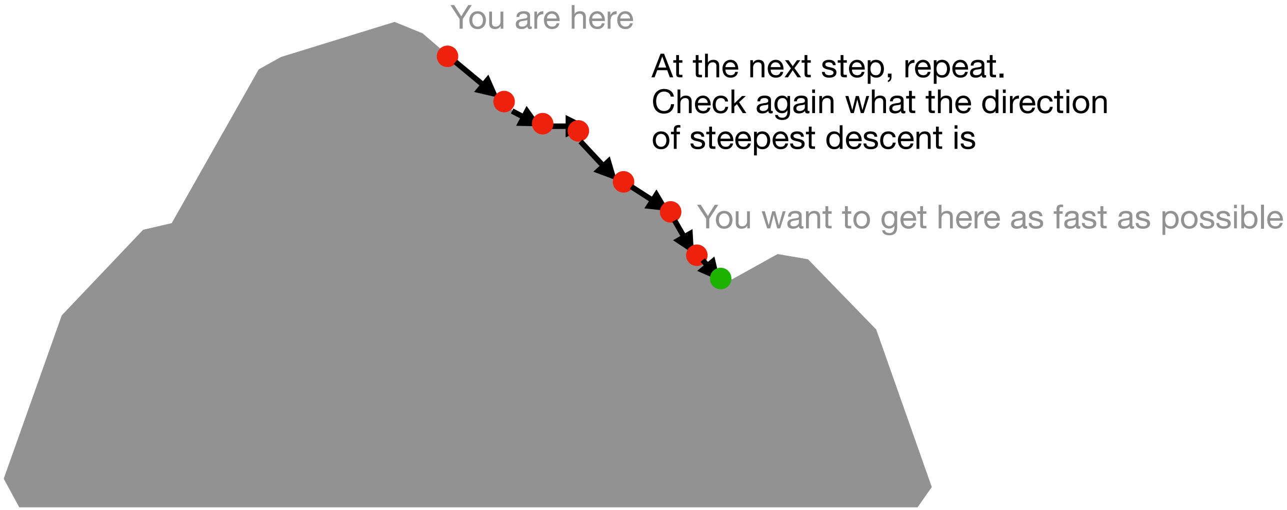
- What is gradient descent?

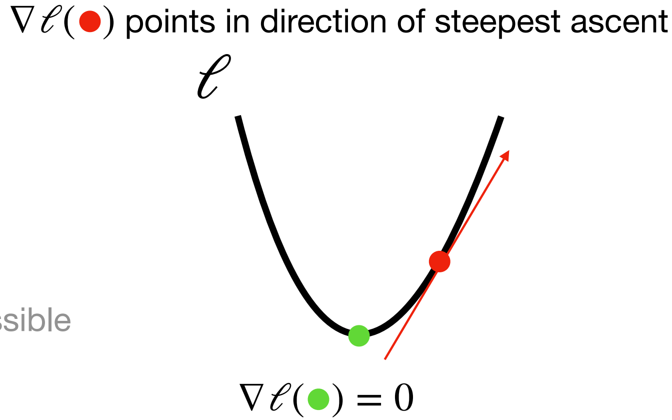




Optimizing Loss Functions

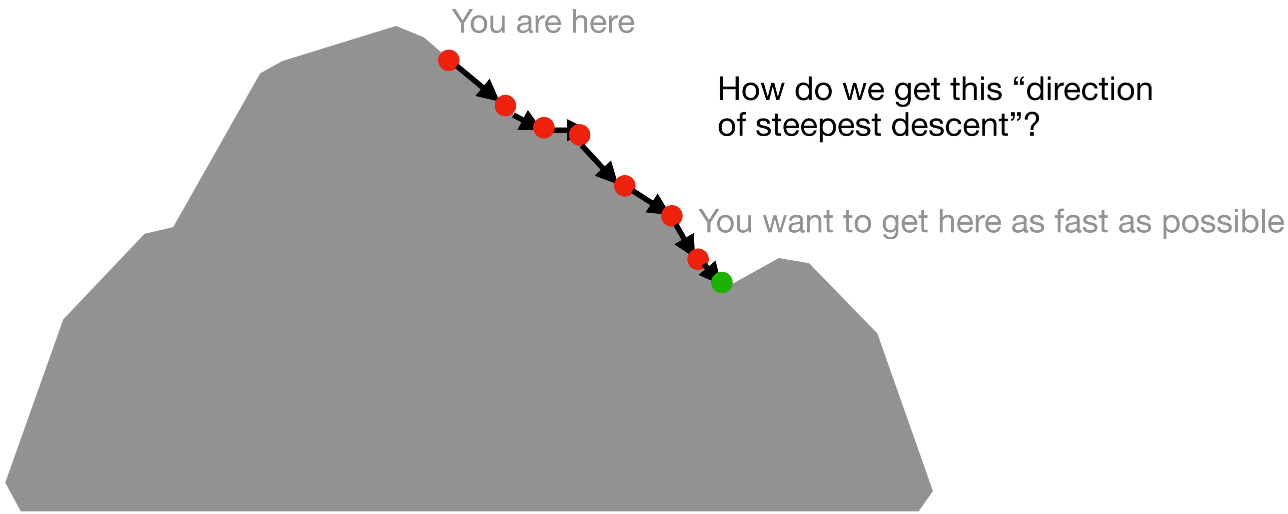
- What is gradient descent?



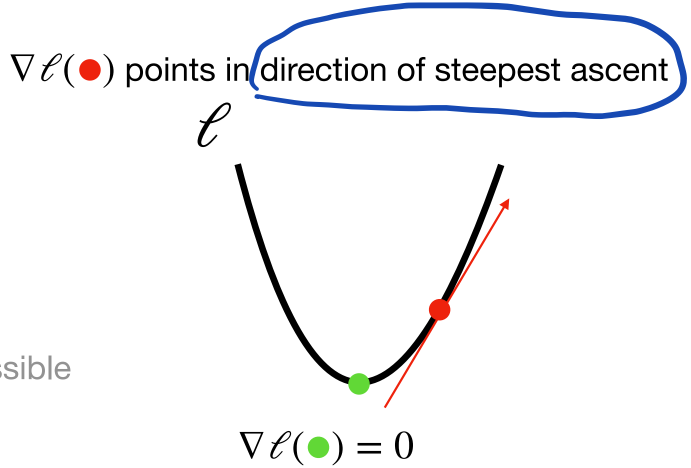


Optimizing Loss Functions

- What is gradient descent?

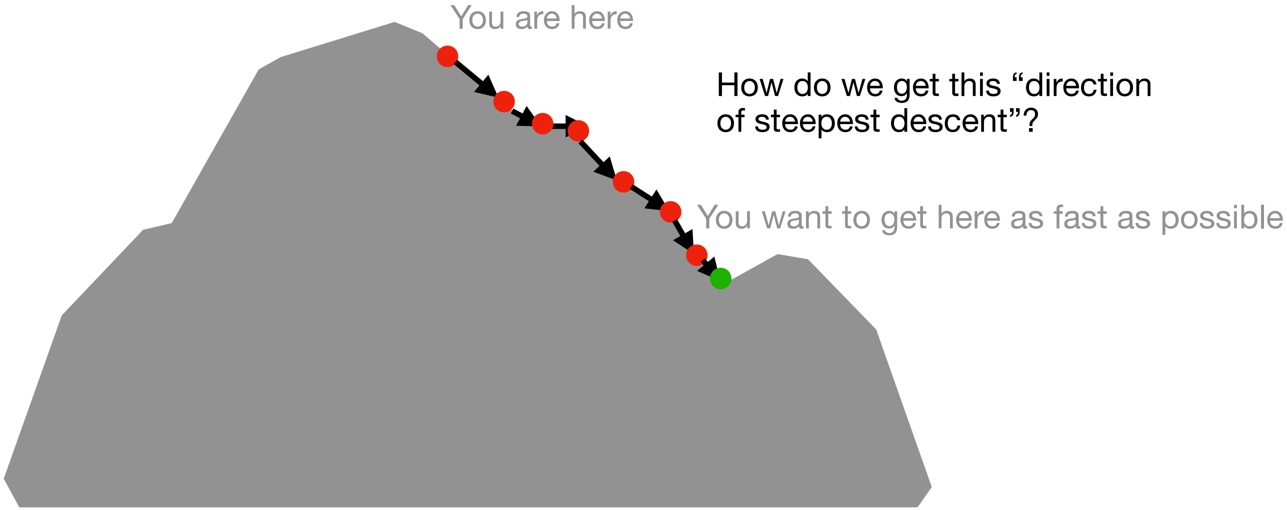


How do we get this "direction of steepest descent"?



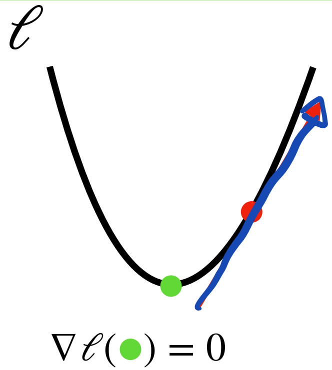
Optimizing Loss Functions

- What is gradient descent?



How do we get this "direction of steepest descent"?

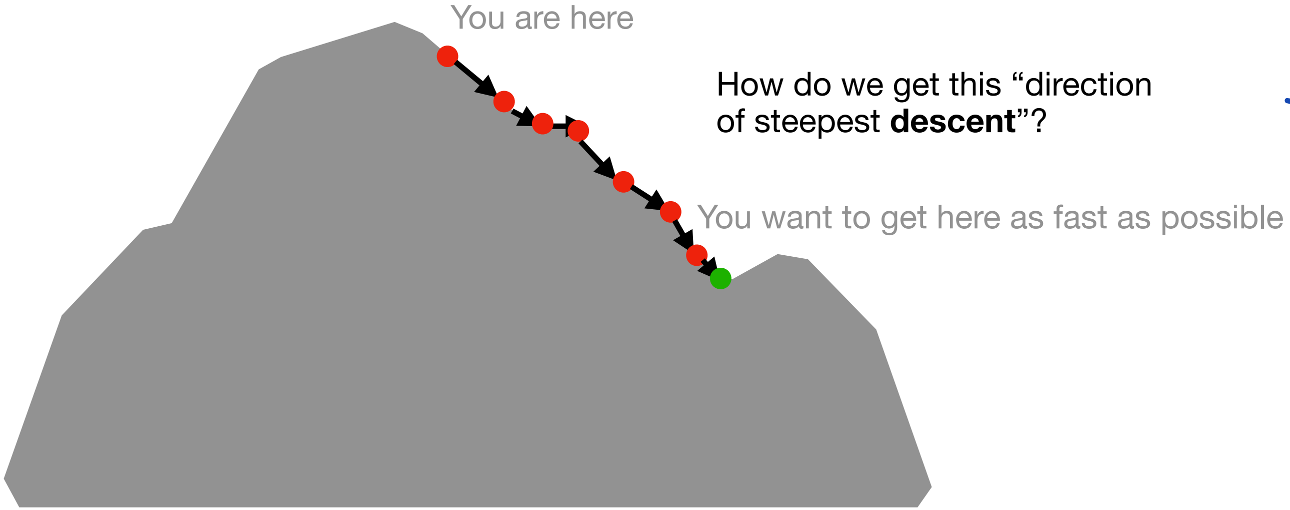
$\nabla \ell(\bullet)$ points in direction of steepest ascent



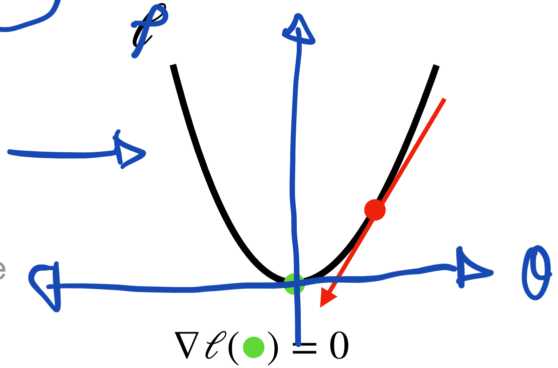
Optimizing Loss Functions

- What is gradient descent?

$-\nabla \ell($●$)$ points in direction of steepest **descent**



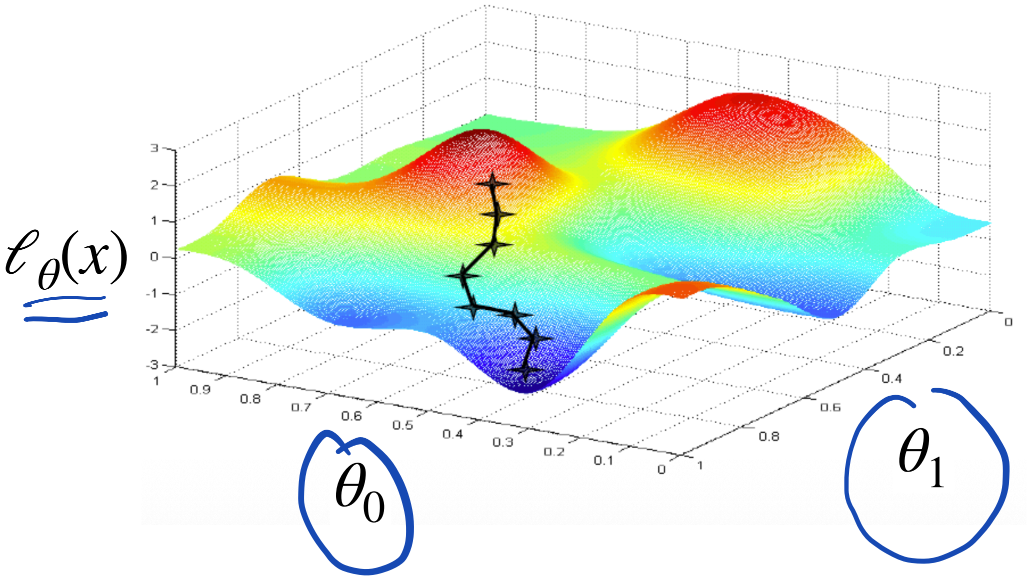
How do we get this "direction of steepest **descent**"?

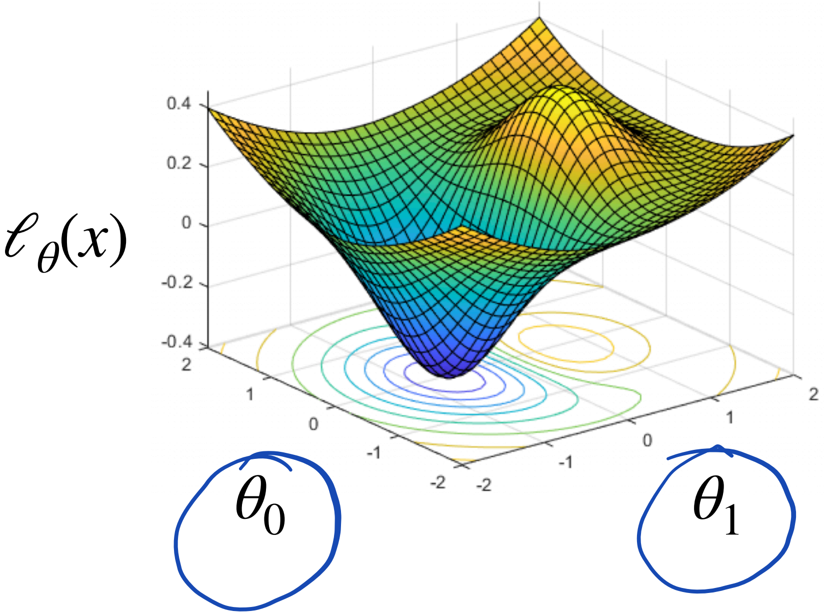


In the plot above, you have a **single** parameter θ

Optimizing Loss Functions

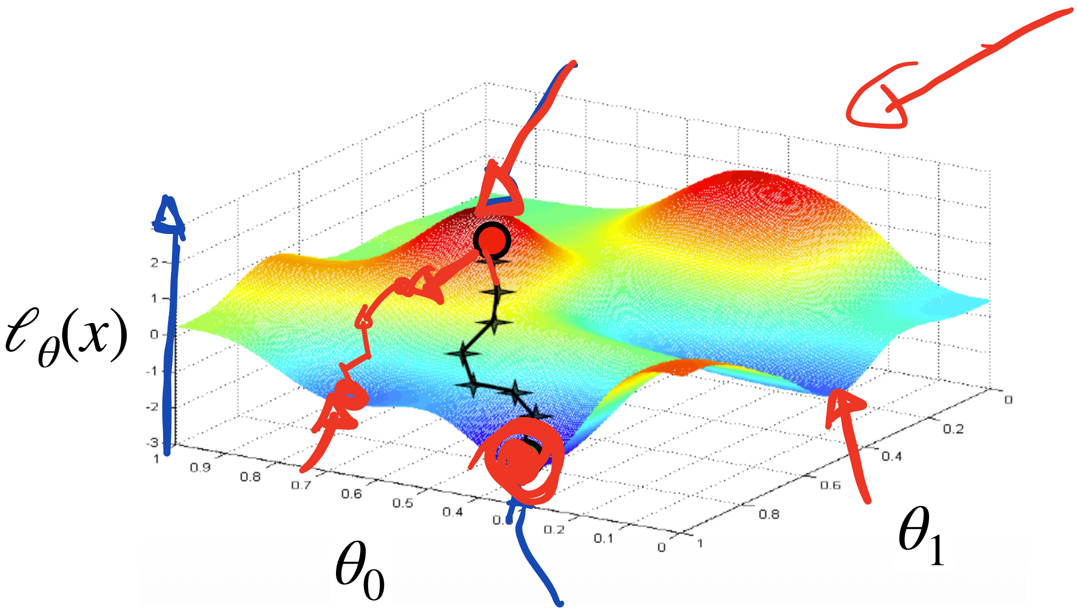
- What does the loss landscape look like with multiple learnable parameters?

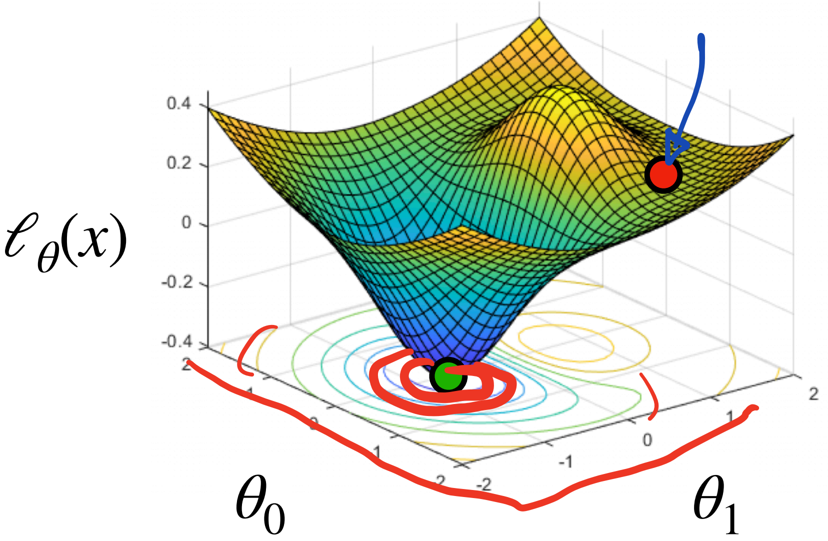




Optimizing Loss Functions

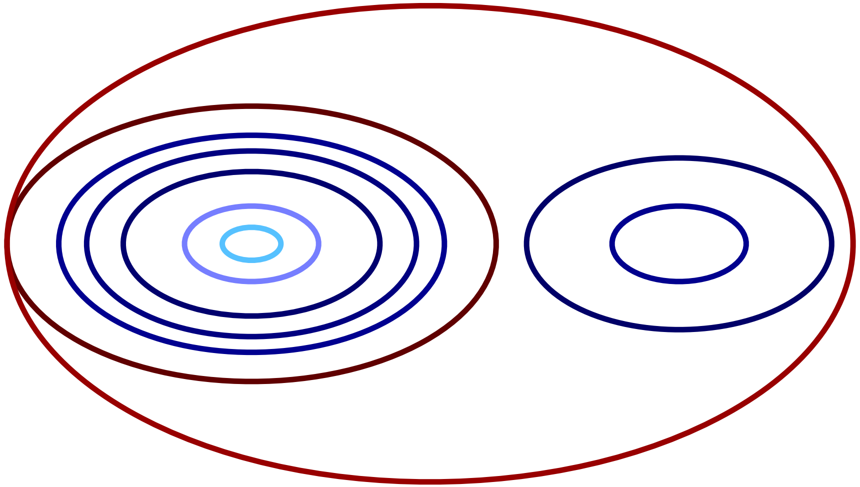
- What does the loss landscape look like with multiple learnable parameters?

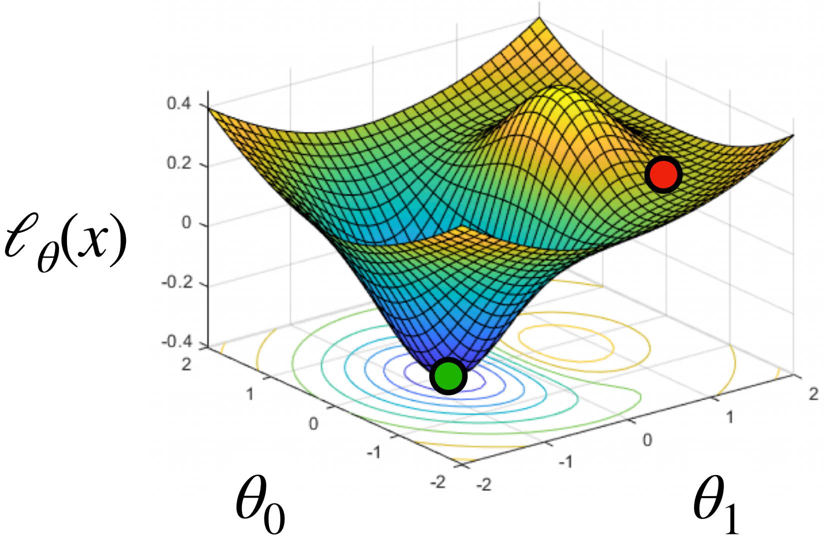




Optimizing Loss Functions

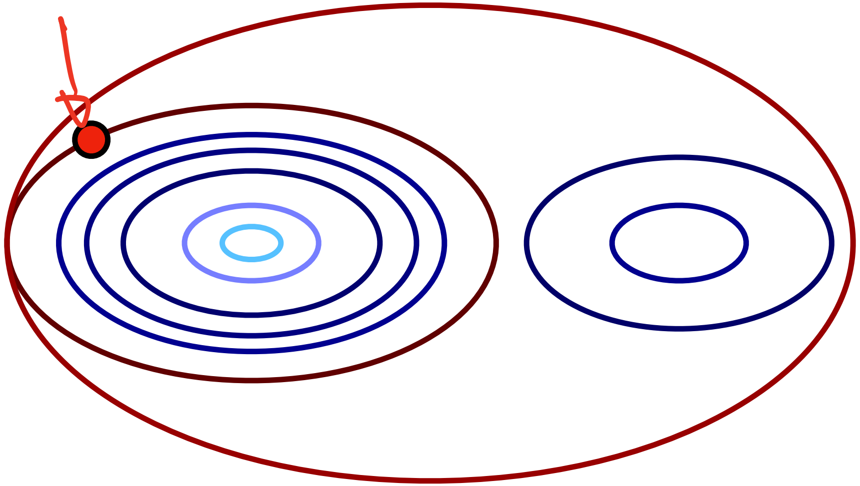
- What does the loss landscape look like with multiple learnable parameters?

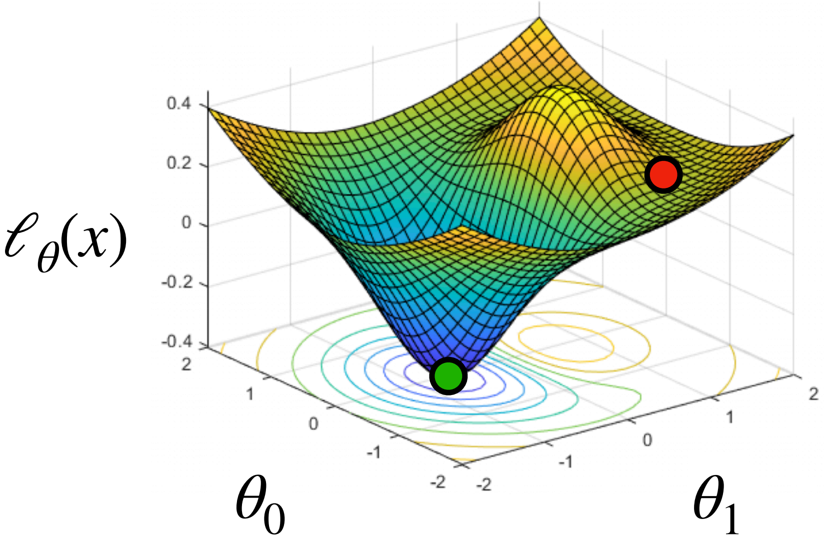




Optimizing Loss Functions

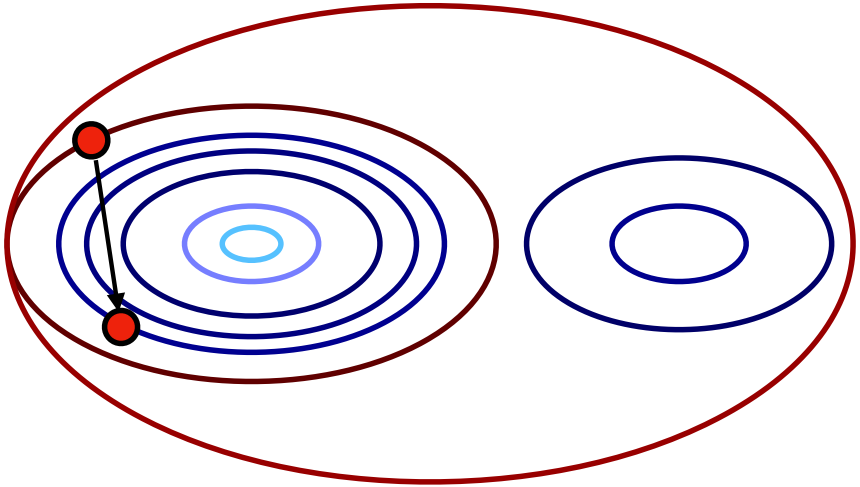
- What does the loss landscape look like with multiple learnable parameters?

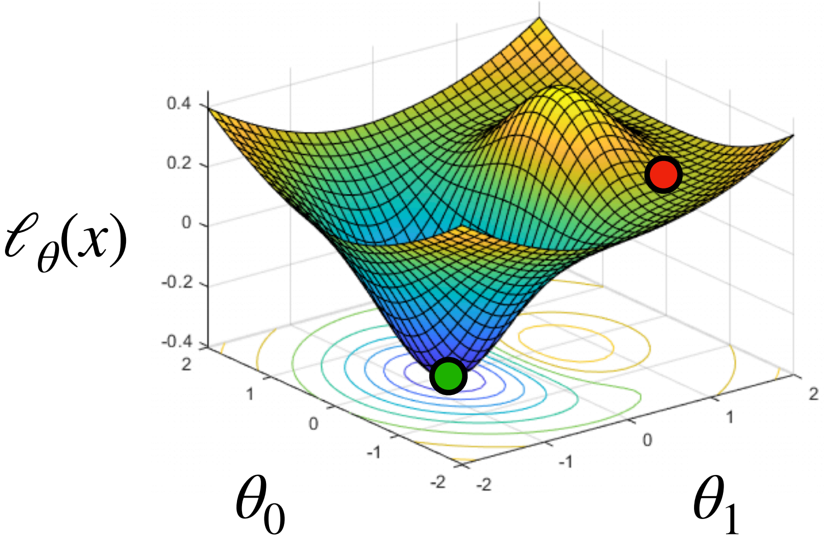




Optimizing Loss Functions

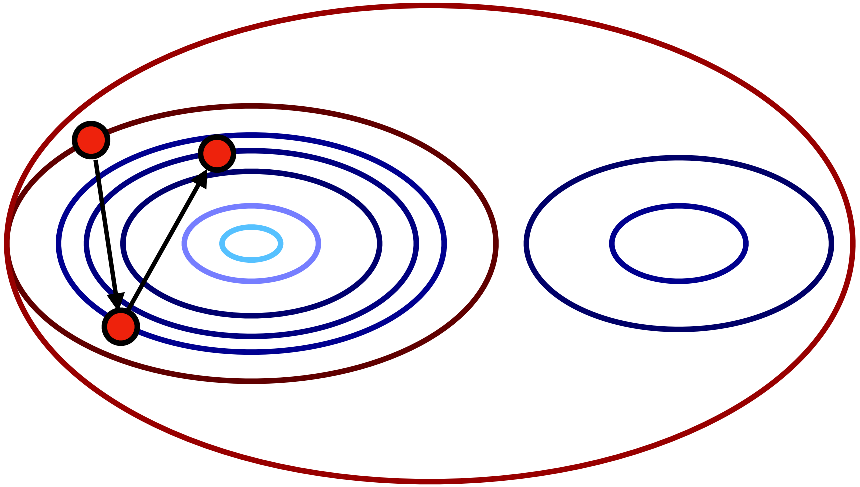
- What does the loss landscape look like with multiple learnable parameters?

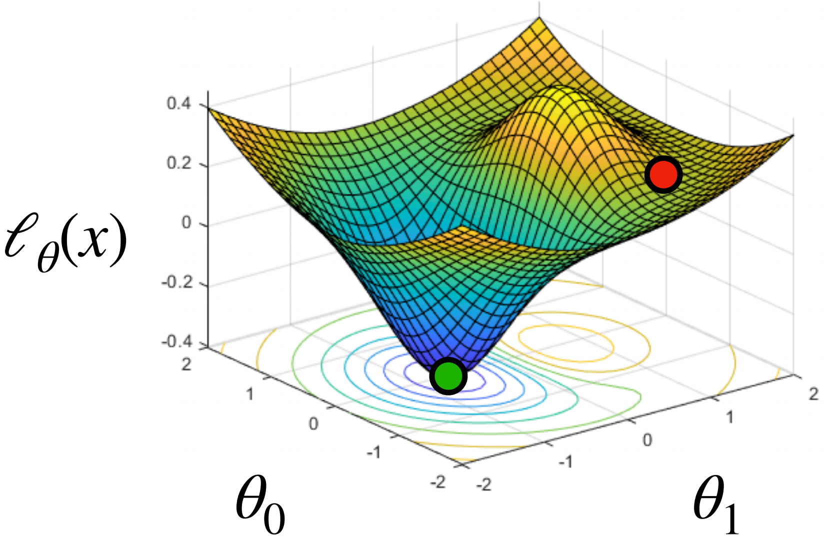




Optimizing Loss Functions

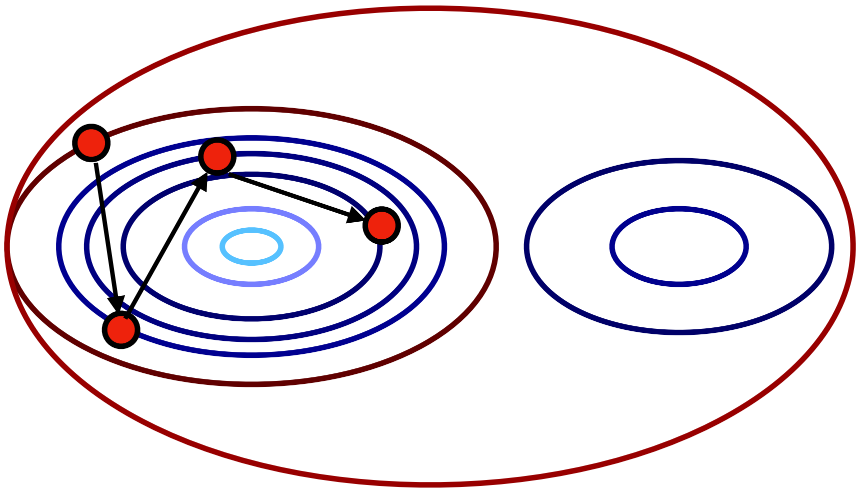
- What does the loss landscape look like with multiple learnable parameters?

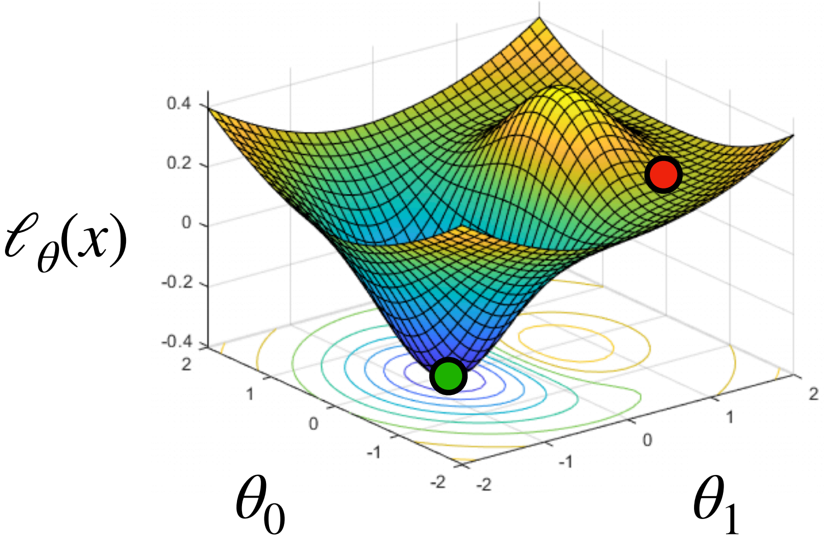




Optimizing Loss Functions

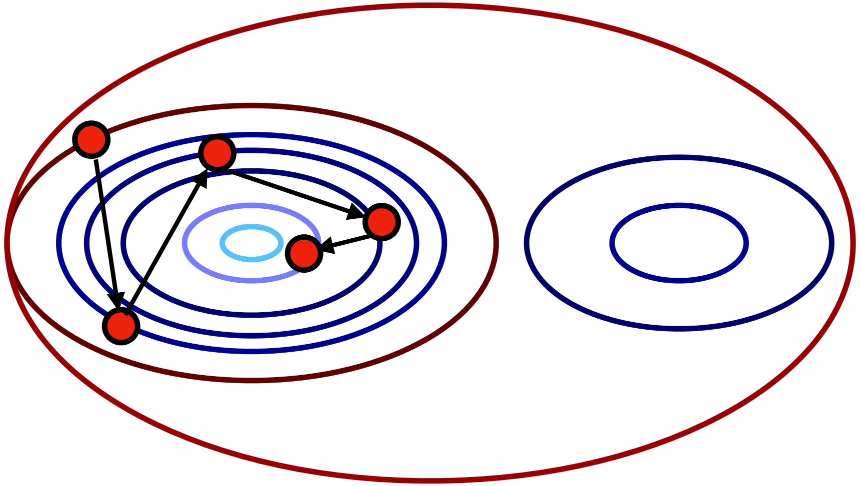
- What does the loss landscape look like with multiple learnable parameters?

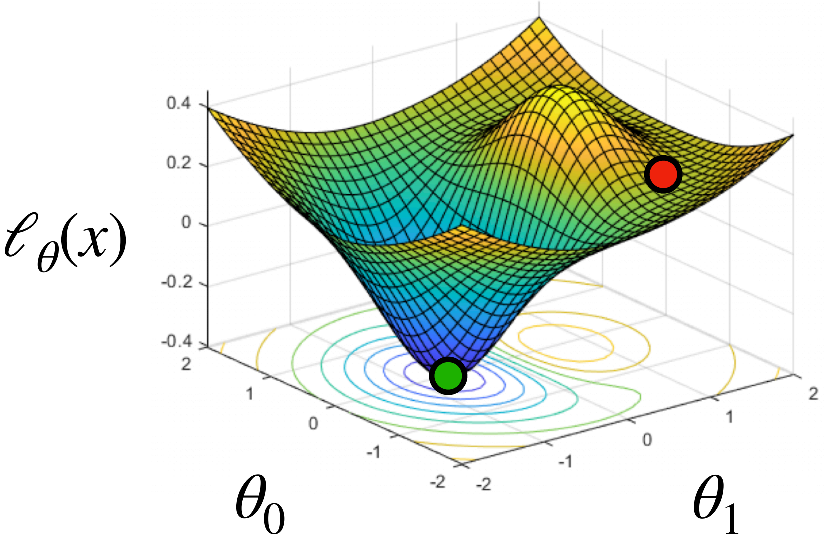




Optimizing Loss Functions

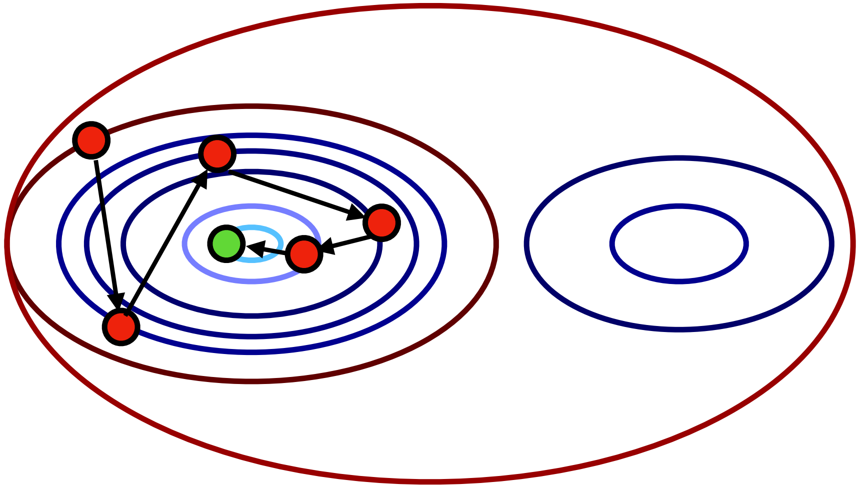
- What does the loss landscape look like with multiple learnable parameters?

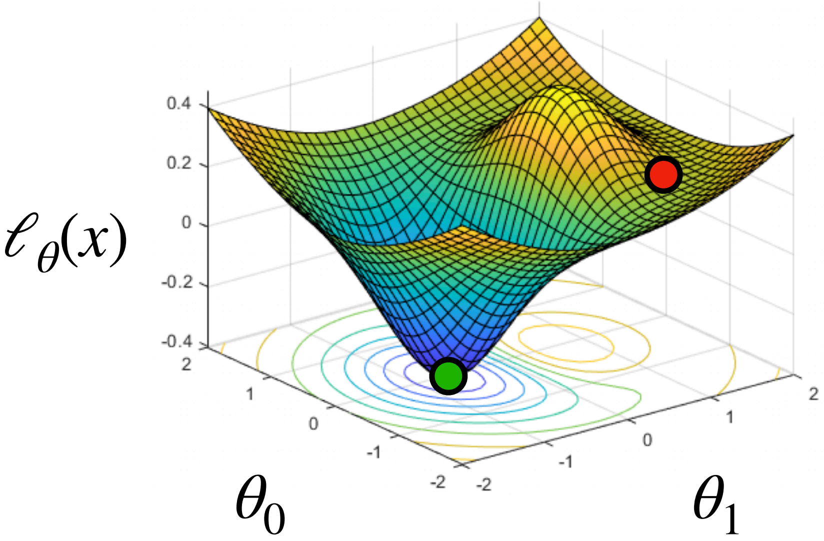




Optimizing Loss Functions

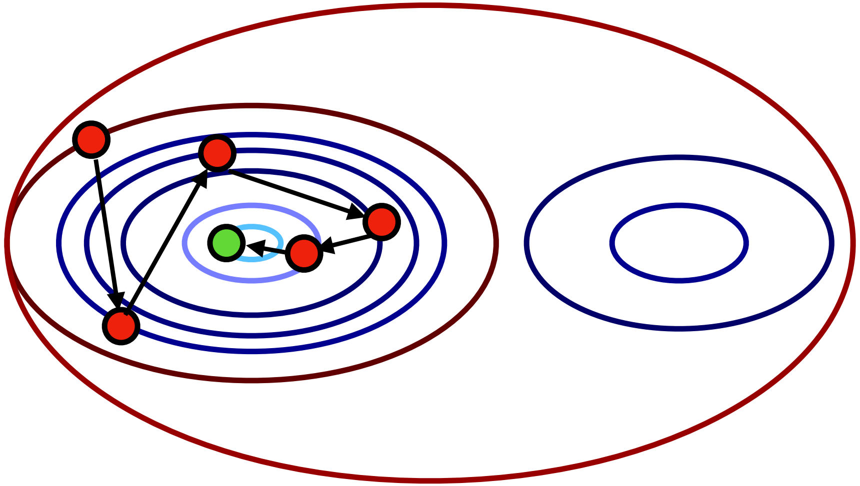
- What does the loss landscape look like with multiple learnable parameters?



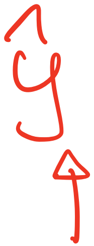


Optimizing Loss Functions

Gradient Descent - Formulation

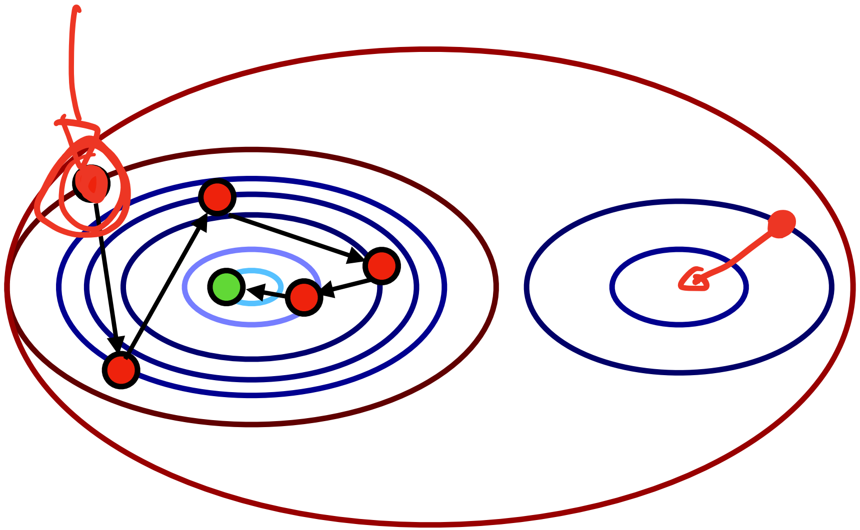


$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$



Optimizing Loss Functions

Gradient Descent - Formulation

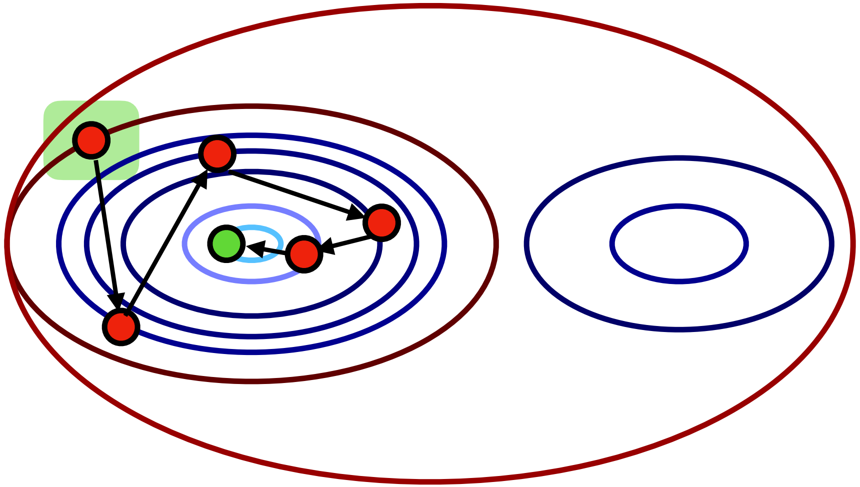


$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

Step 1: Initialize θ_0, θ_1

Optimizing Loss Functions

Gradient Descent - Formulation



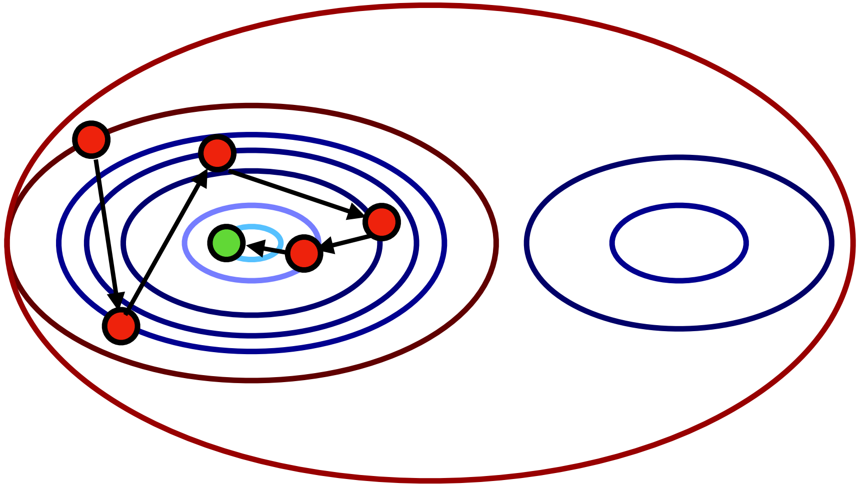
$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

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



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

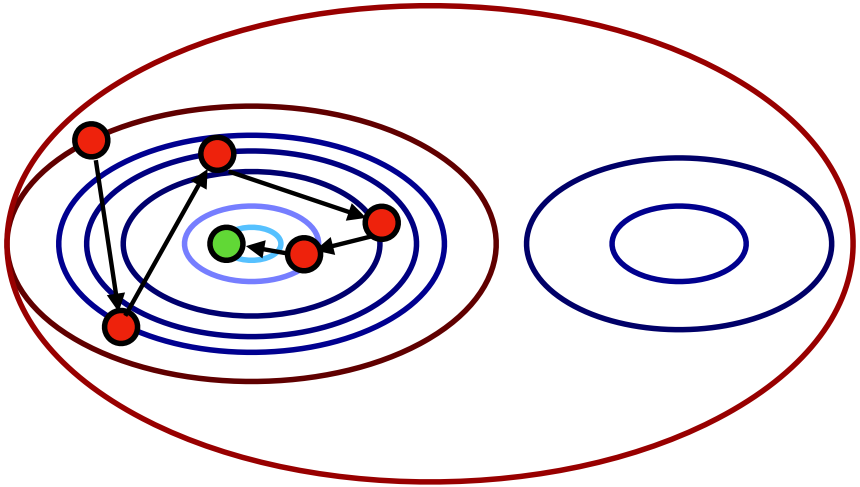
Step 1: Initialize θ_0, θ_1

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

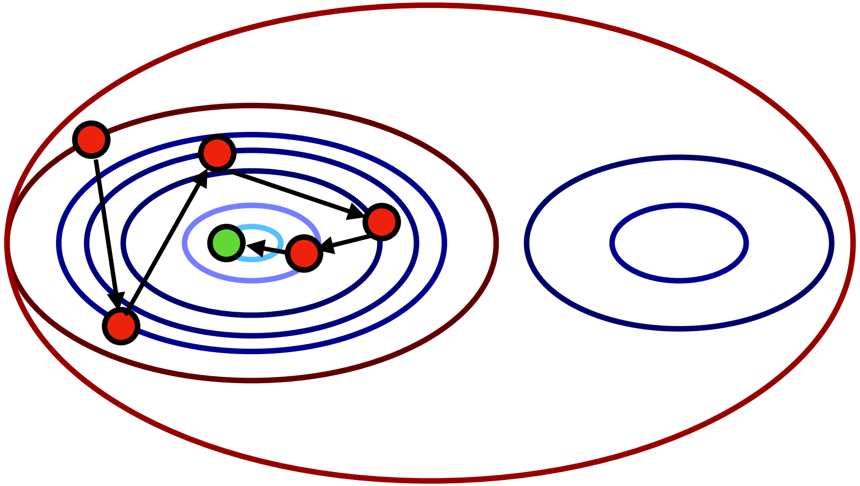
Step 2: Repeat Until Convergence

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

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

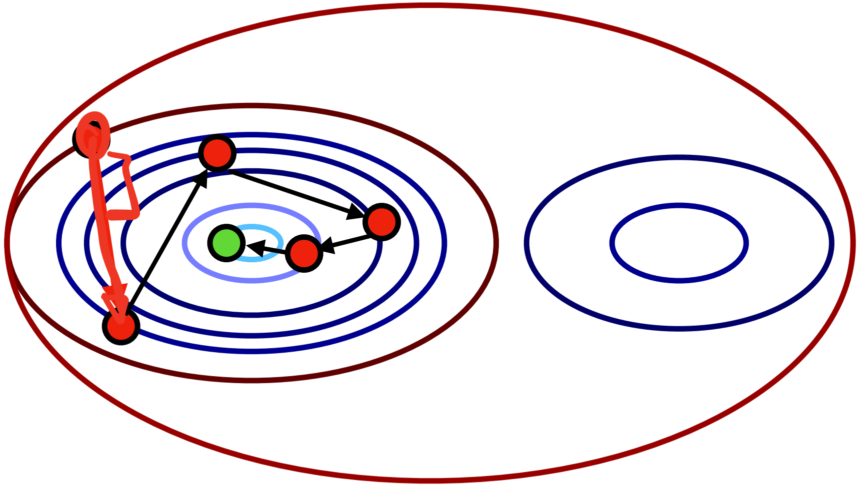
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

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

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

10^{-3}

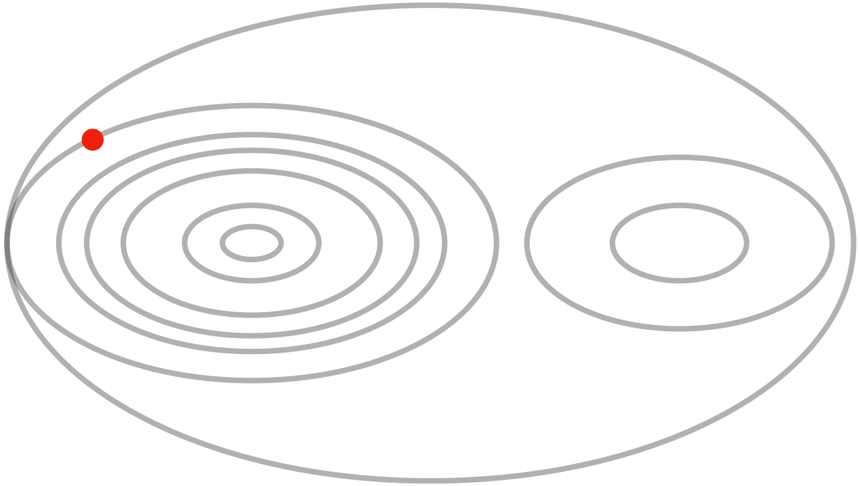
α : Learning Rate

Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

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

10^{-3}



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

Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

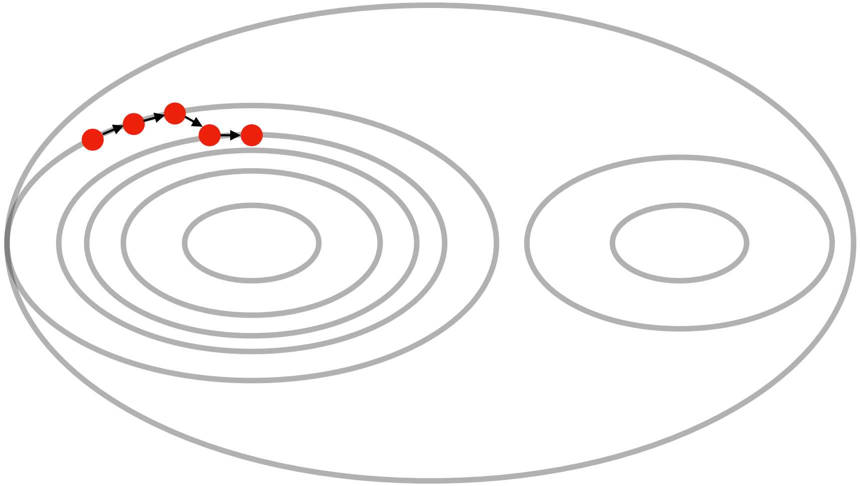
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

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

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

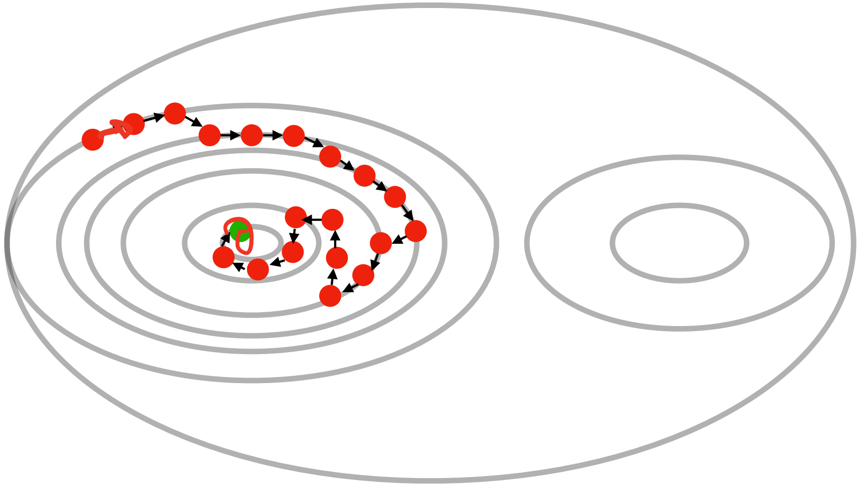
α : Learning Rate

Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

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

10^-2
10^-8



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

Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

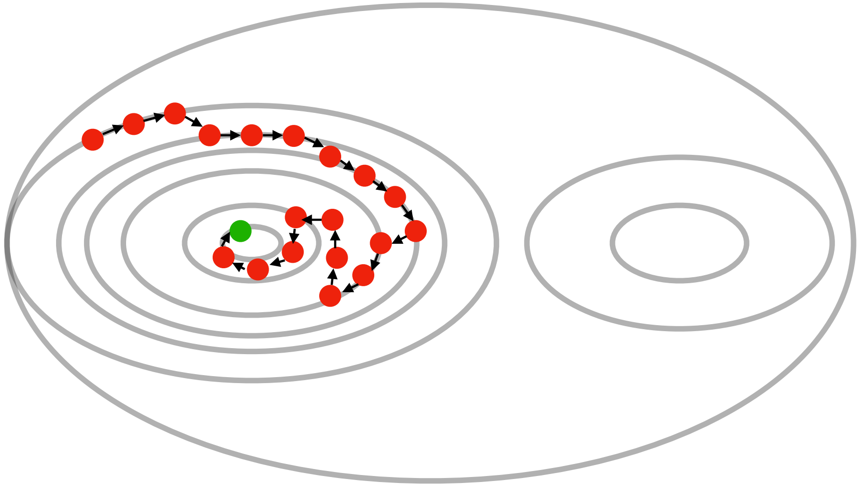
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

α : Learning Rate

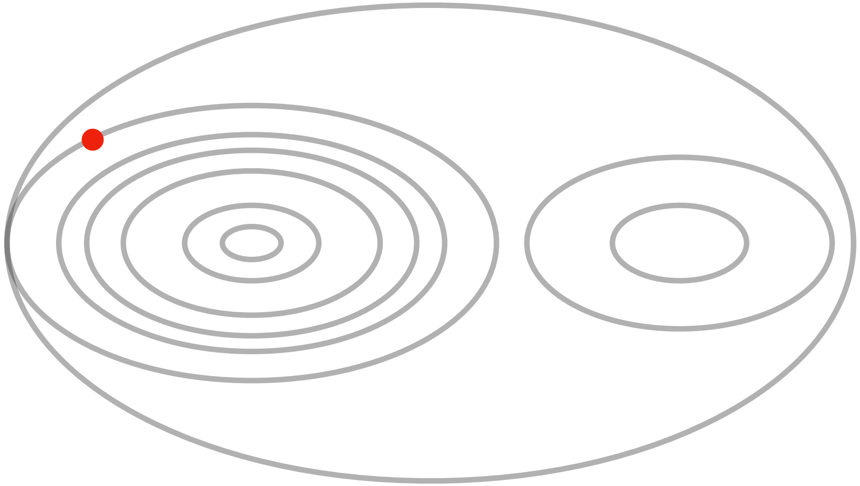
Optimizing Loss Functions

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 $\alpha = 10$

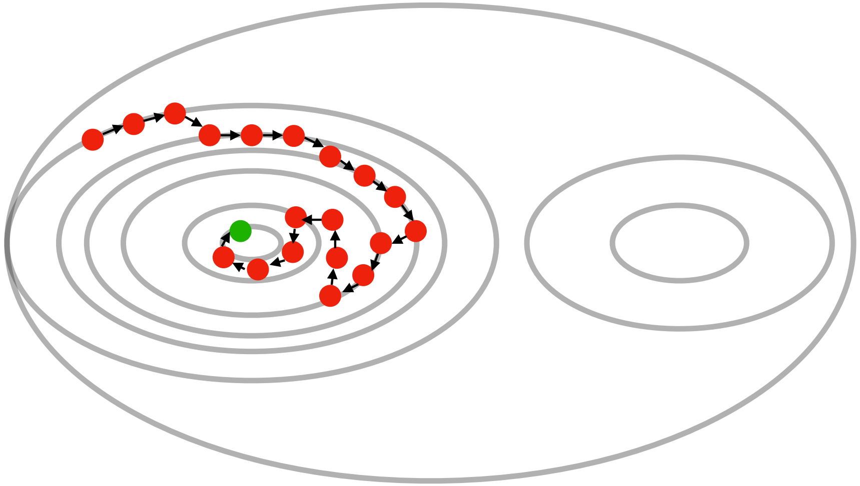


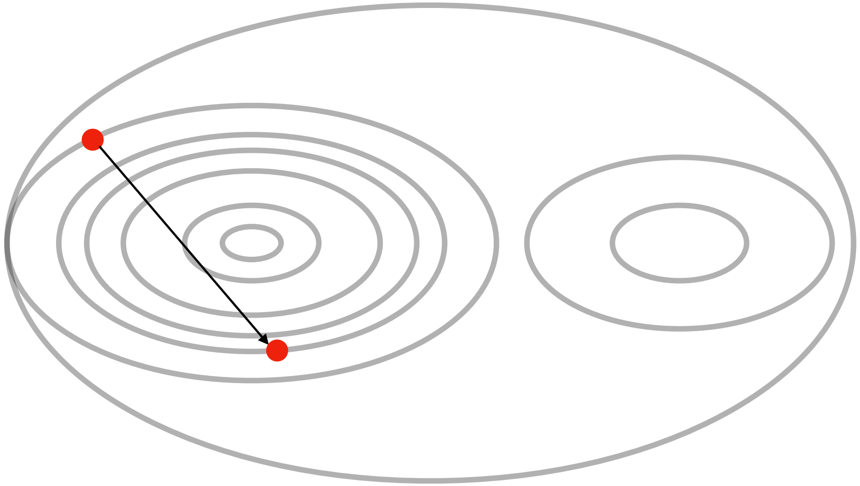
Optimizing Loss Functions

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 $\alpha = 10$

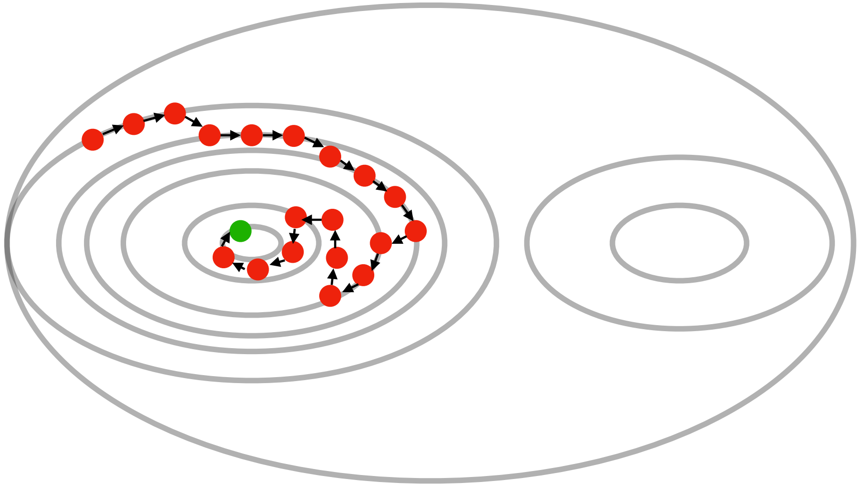




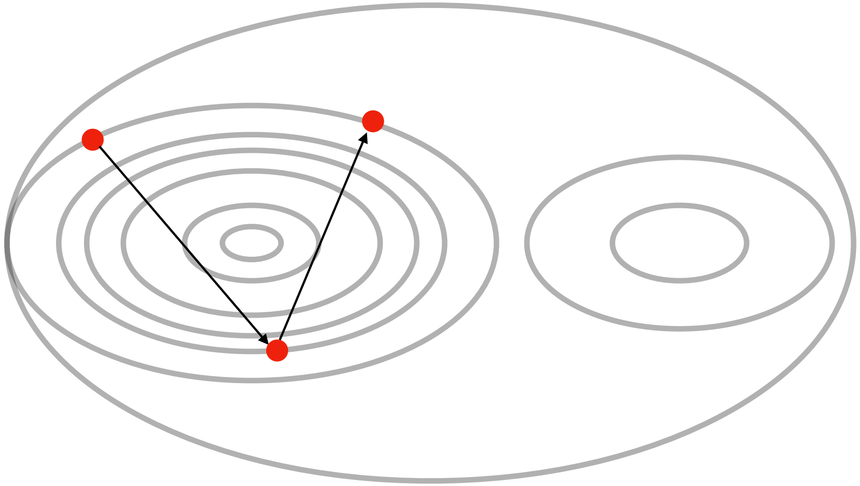
Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

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



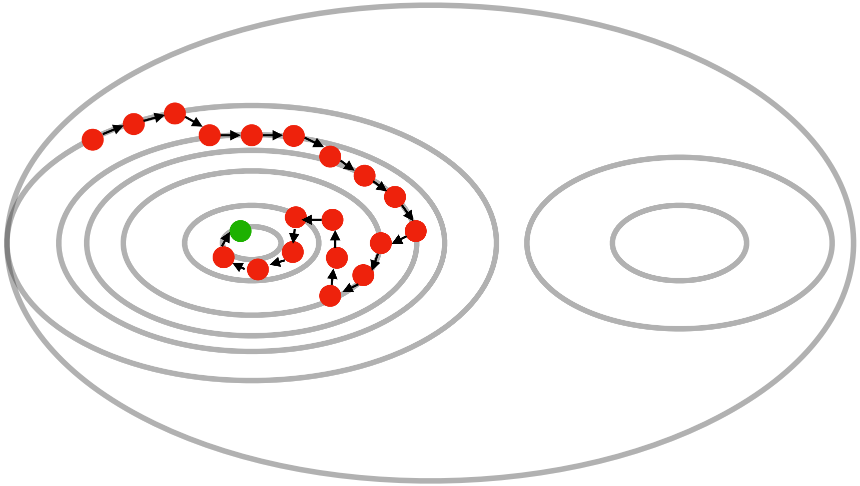
What happens when α is too large?
Say $\alpha = 10$



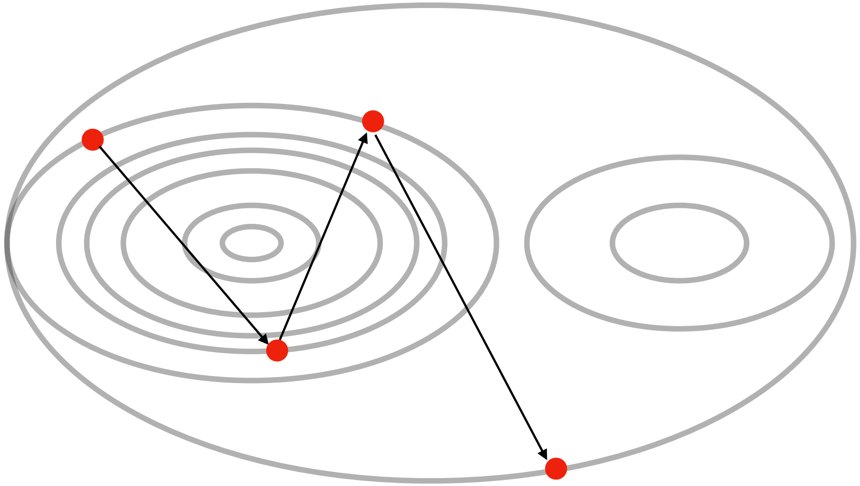
Optimizing Loss Functions

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 $\alpha = 10$

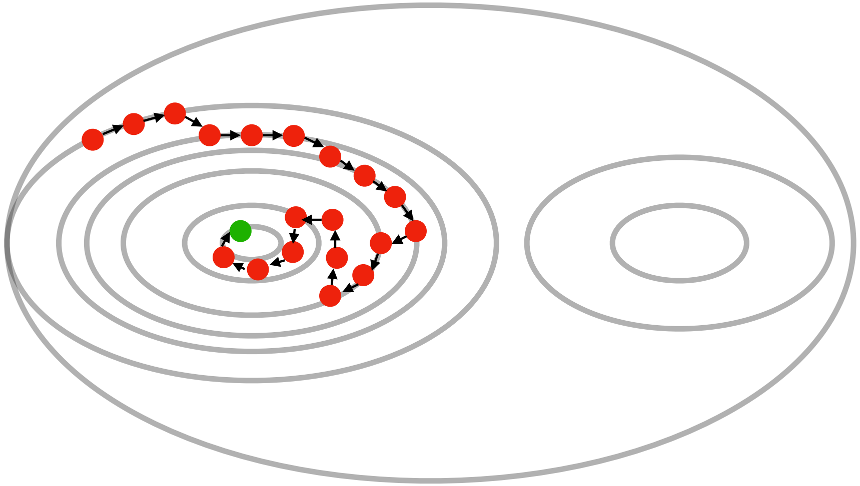


Optimizing Loss Functions

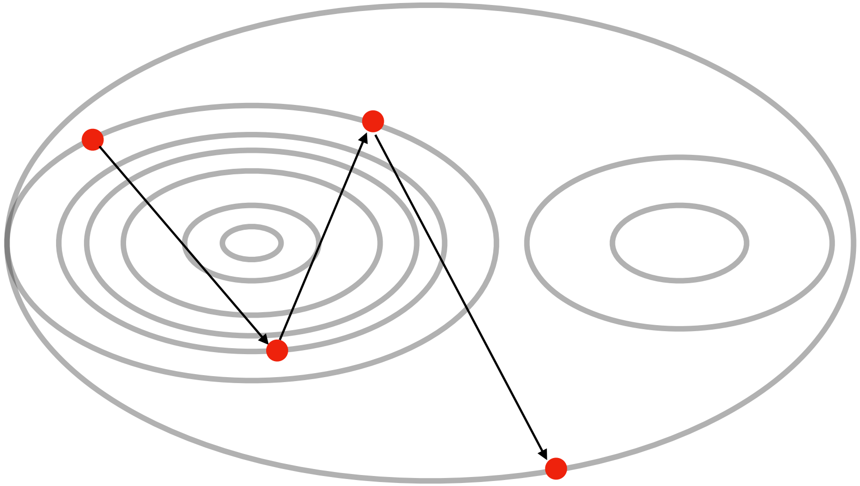
Gradient Descent - Effect of Learning Rate

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 $\alpha = 10$

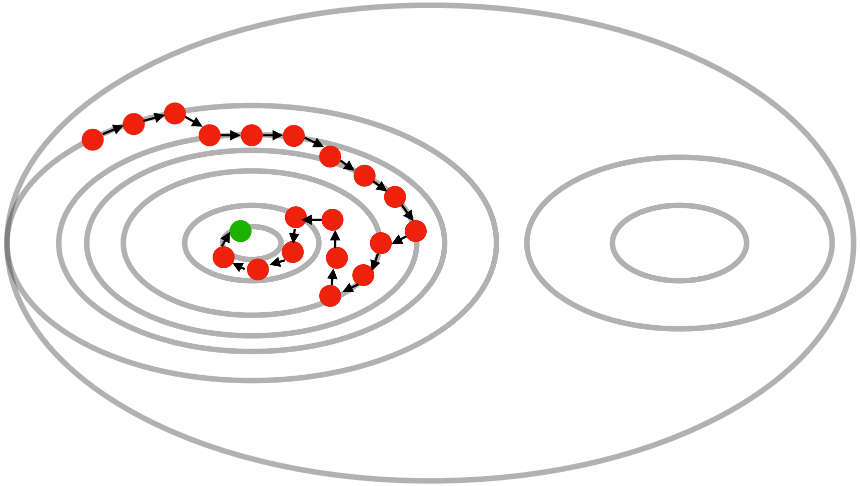


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

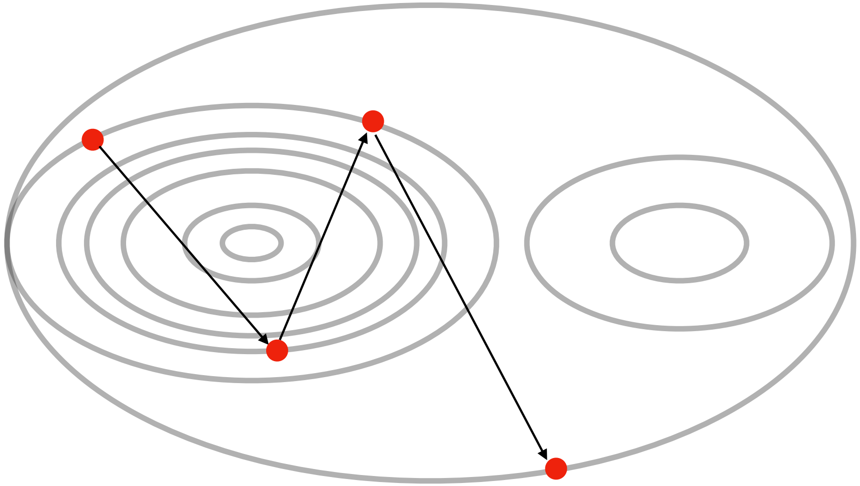
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

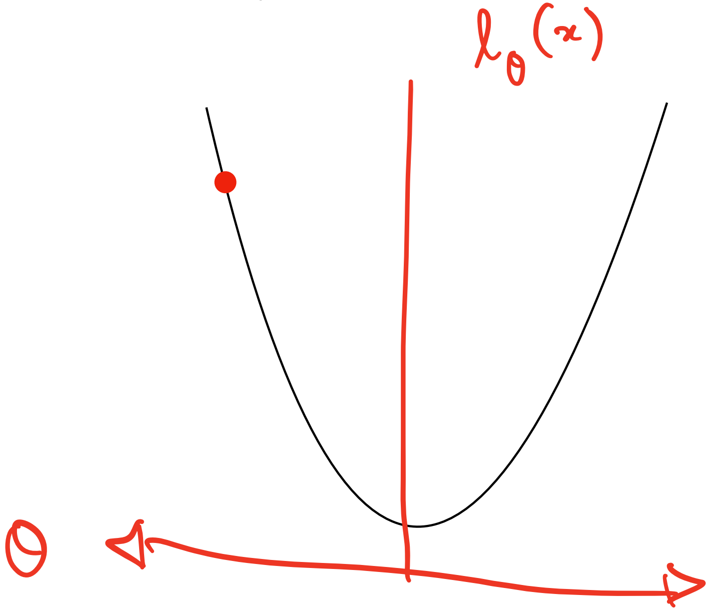


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

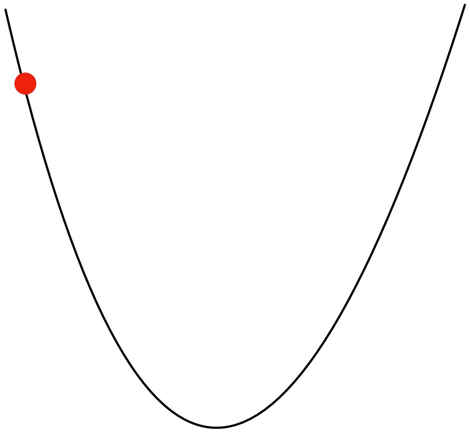
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

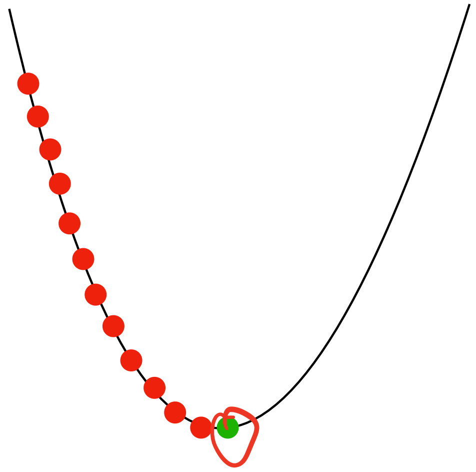


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

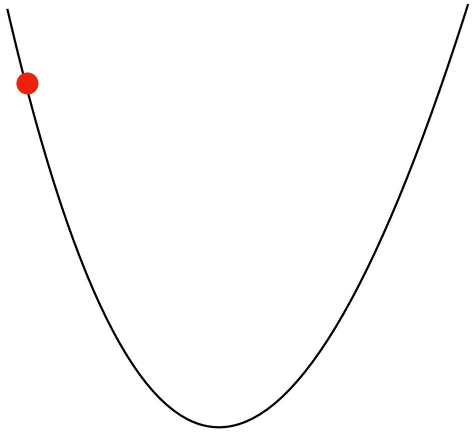
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

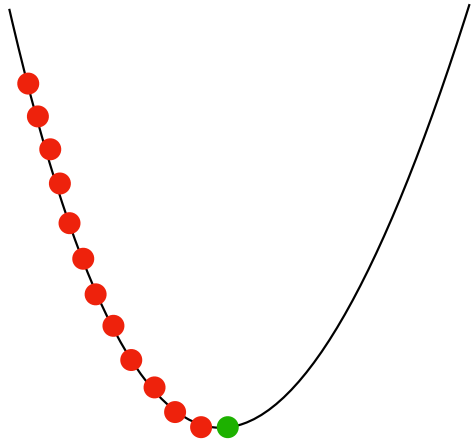


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

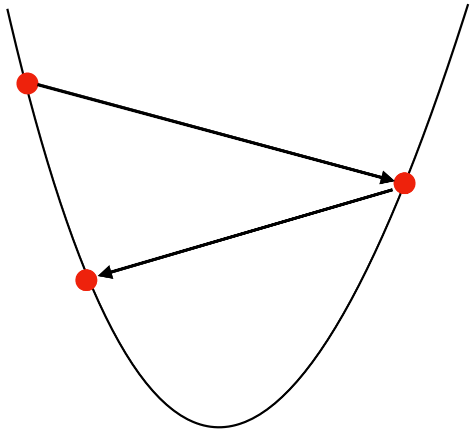
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

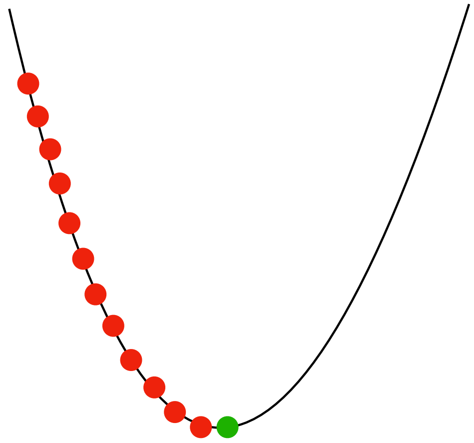


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

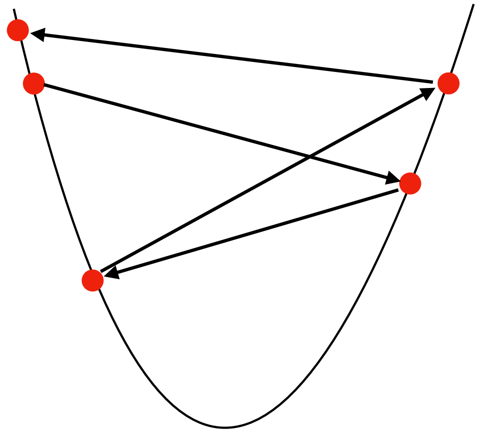
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

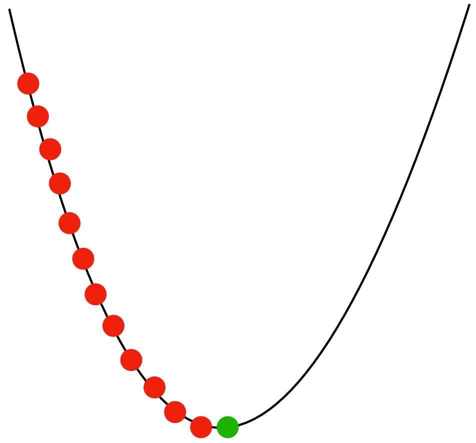


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

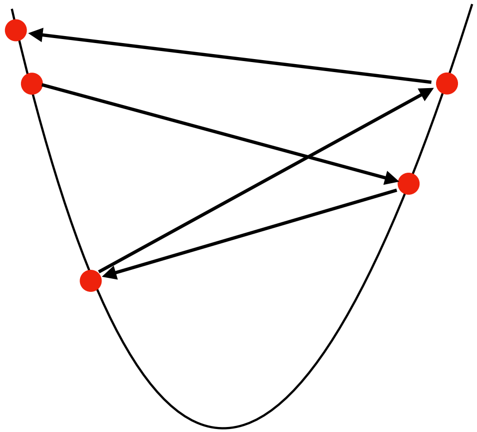
What happens when α is too small?
Say $\alpha = 10^{-5}$

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



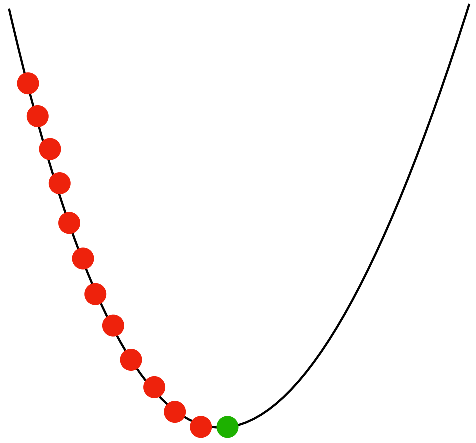
You might not always diverge, but converging might still not be possible

Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

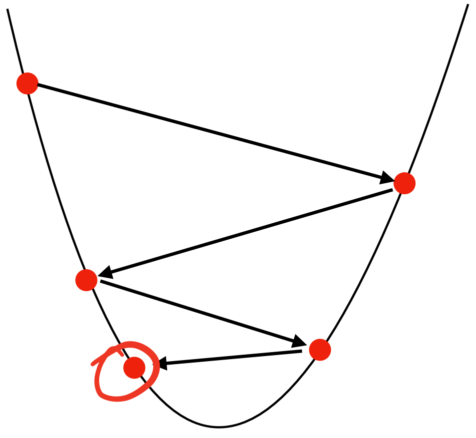
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

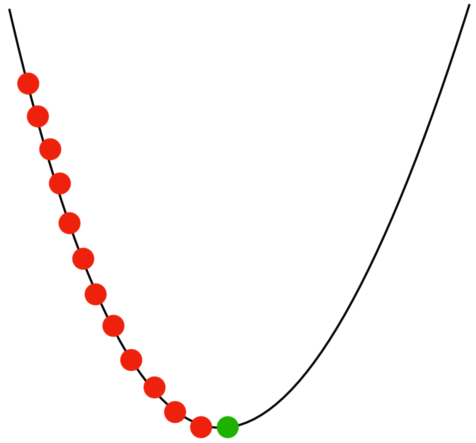


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

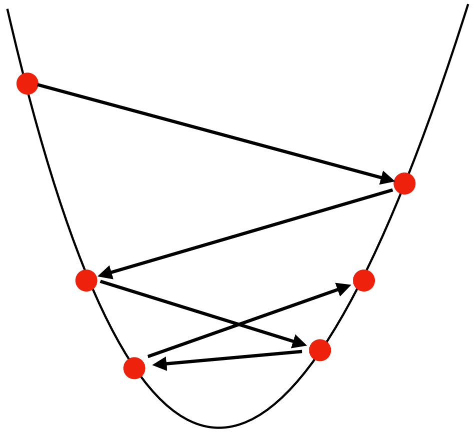
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

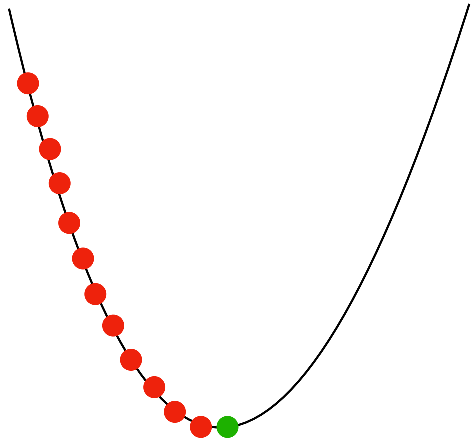


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

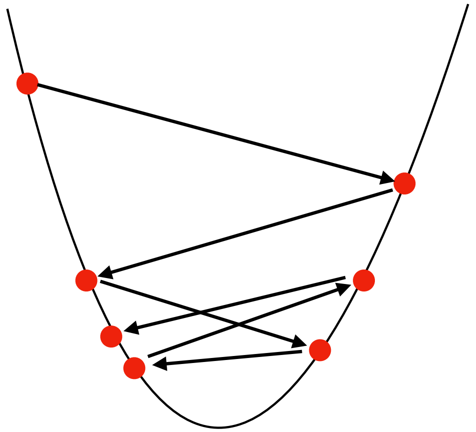
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

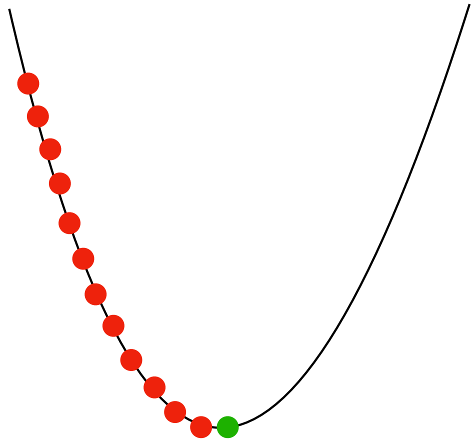


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

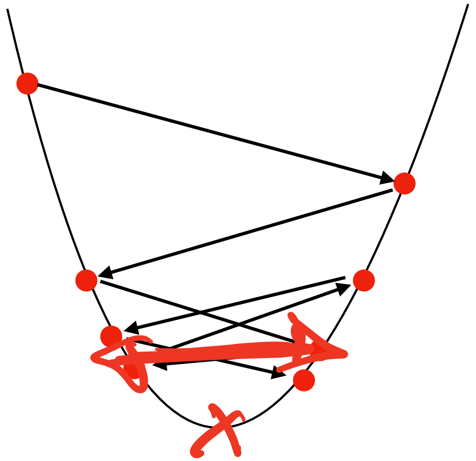
What happens when α is too small?
Say $\alpha = 10^{-5}$

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

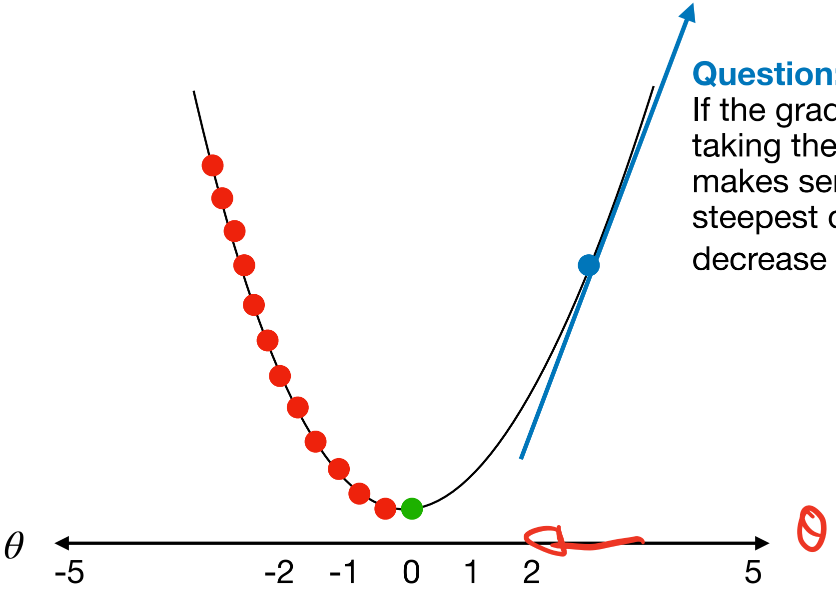


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

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

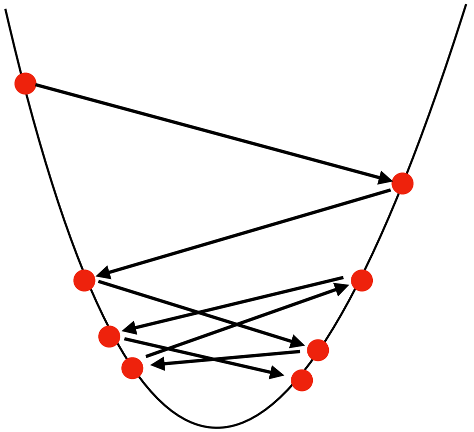
With a small learning rate α, if the loss function is convex, the optimization will eventually **converge**



Question:
If the gradient here is positive, taking the negative of the gradient makes sense to get direction of steepest descent - i.e., we decrease the value of θ

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



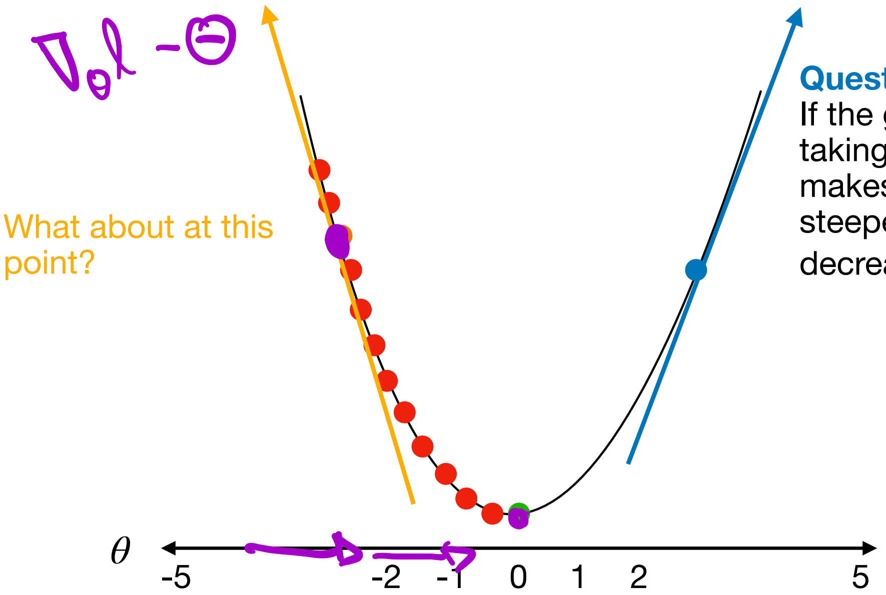
Optimizing Loss Functions

Gradient Descent - 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**

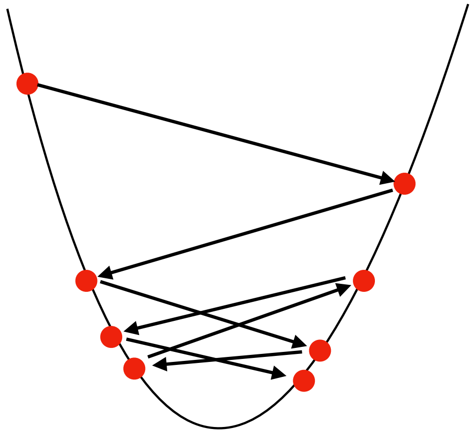
What about at this point?



Question:
If the gradient here is positive, taking the negative of the gradient makes sense to get direction of steepest descent - i.e., we decrease the value of θ

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

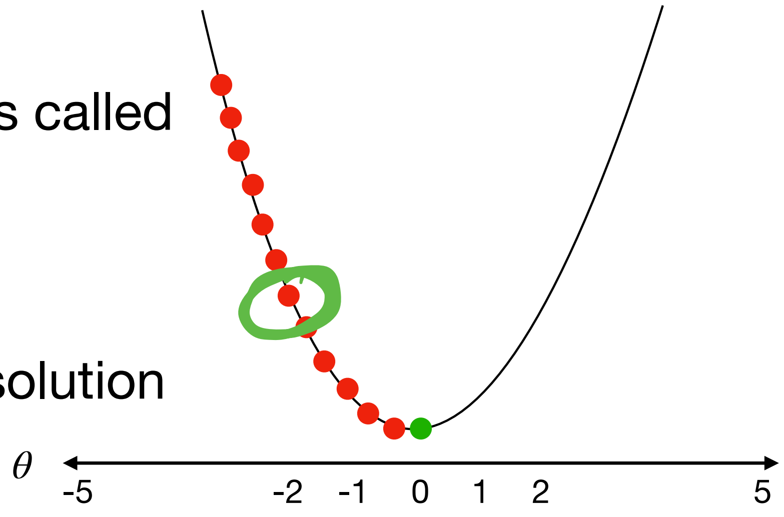
With a large learning rate α, if the loss function is convex, the optimization could possibly start **diverging** and never converge



Optimizing Loss Functions

Gradient Descent - Stopping Criterion

- 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



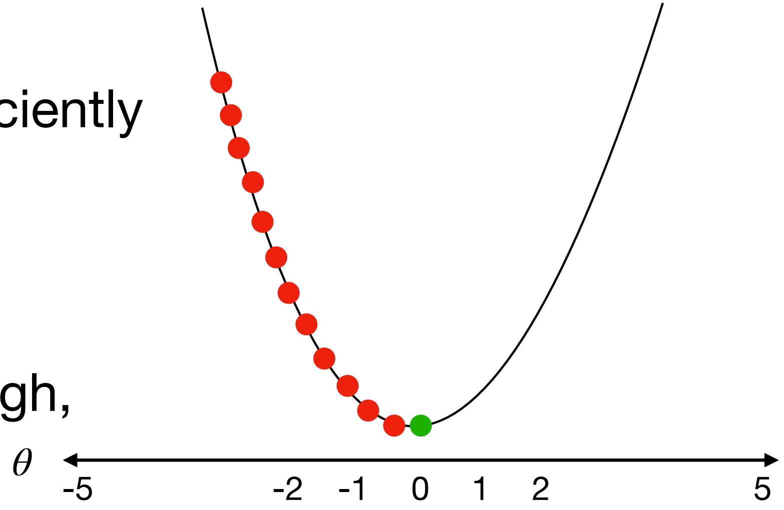
Optimizing Loss Functions

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

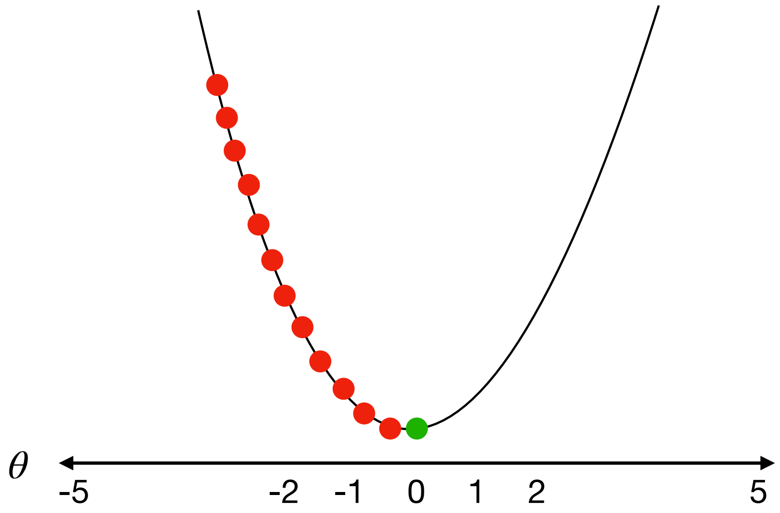


Optimizing Loss Functions

Gradient Descent - Stopping Criterion

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

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



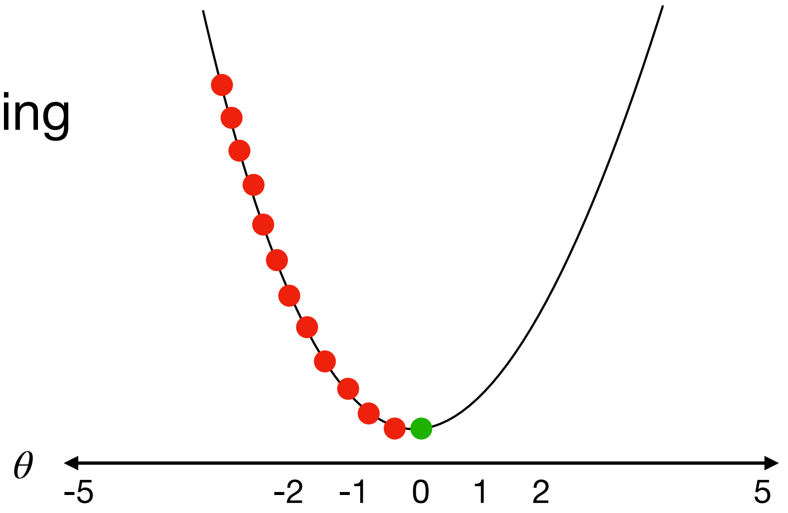
Optimizing Loss Functions

Gradient Descent - Stopping Criterion

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

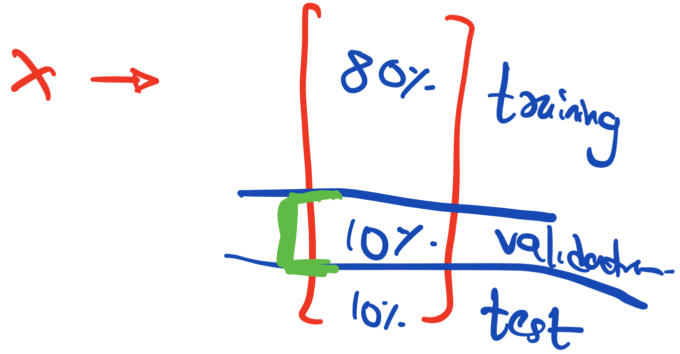
$$|\theta_t - \theta_{t-1}| \leq \epsilon$$

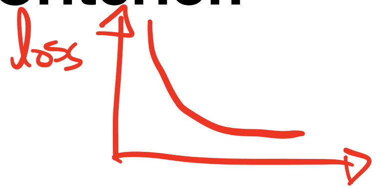
prev slide → $|\ell(\theta_t) - \ell(\theta_{t-1})|$



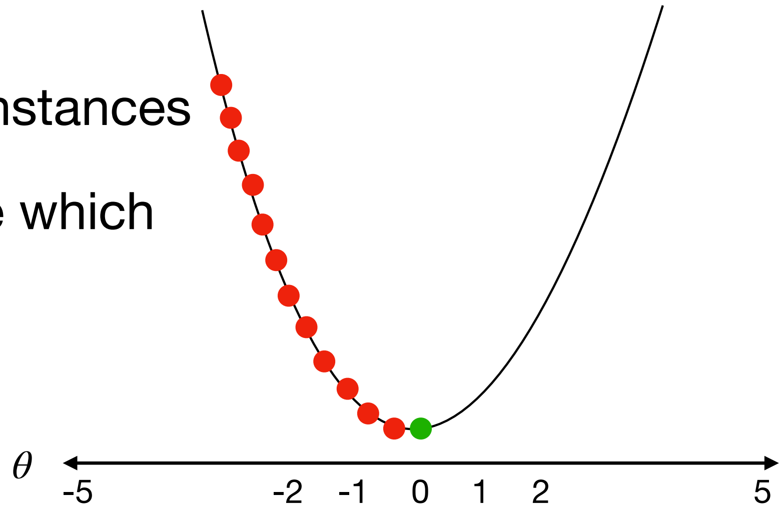
Optimizing Loss Functions

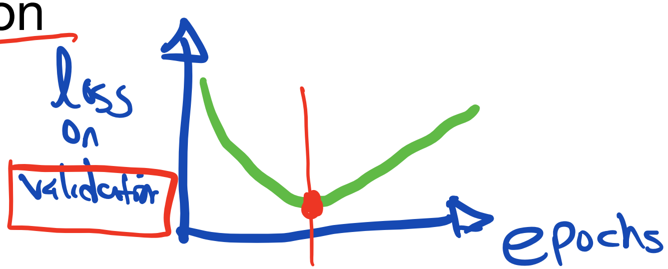
Gradient Descent - Stopping Criterion





- 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



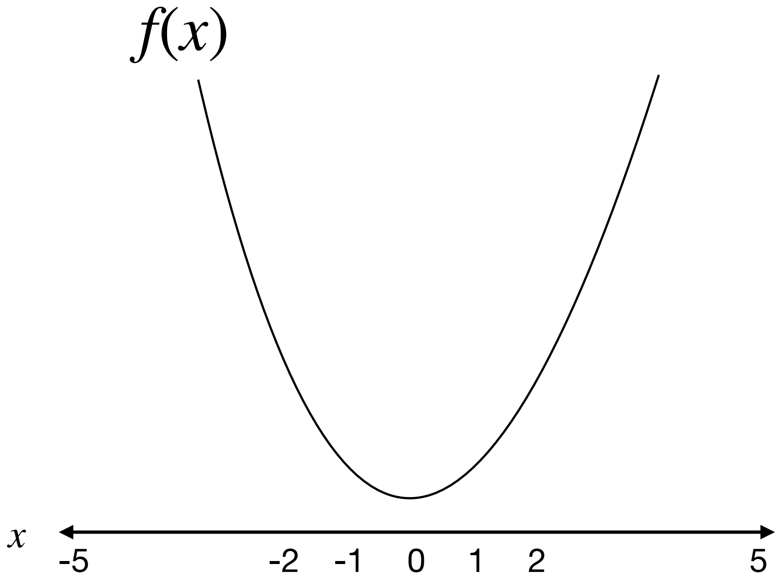


Optimizing Loss Functions

Gradient Descent - Convexity

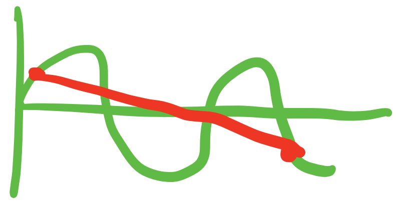
- A function f is convex if for all points in its domain (input) and for all $\lambda \in [0,1]$

$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$



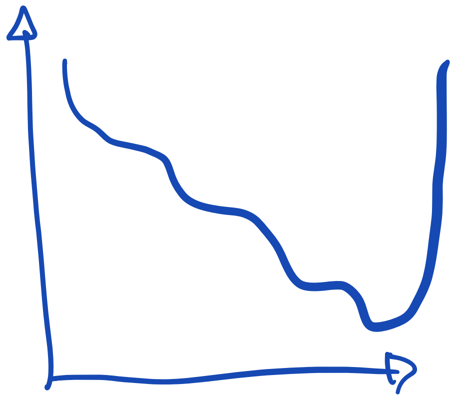
Optimizing Loss Functions

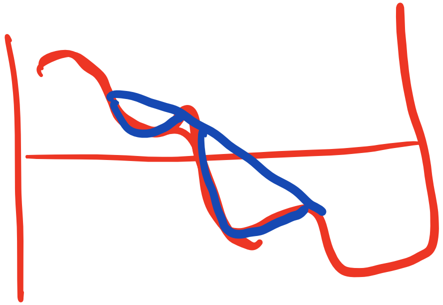
Gradient Descent - Convexity

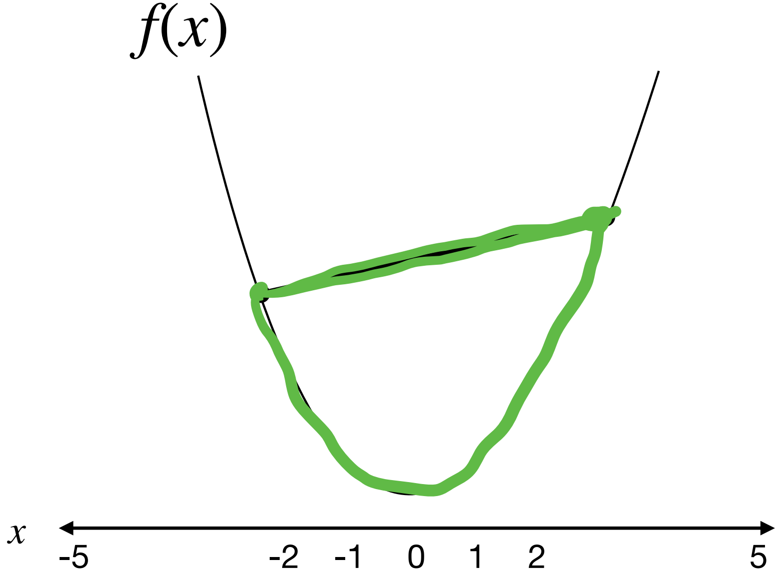


- A function f is convex if for all points in its domain (input) and for all $\lambda \in [0,1]$

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$







Optimizing Loss Functions

Gradient Descent - Convexity

$f(\theta_0, \theta_1, \theta_2)$

$\nabla f_\theta : \left[\frac{\partial^2 f}{\partial^2 \theta_1} \quad \frac{\partial^2 f}{\partial^2 \theta_2} \quad \cdots \right]$

- A function f is convex if for all points in its domain (input) and for all $\lambda \in [0,1]$

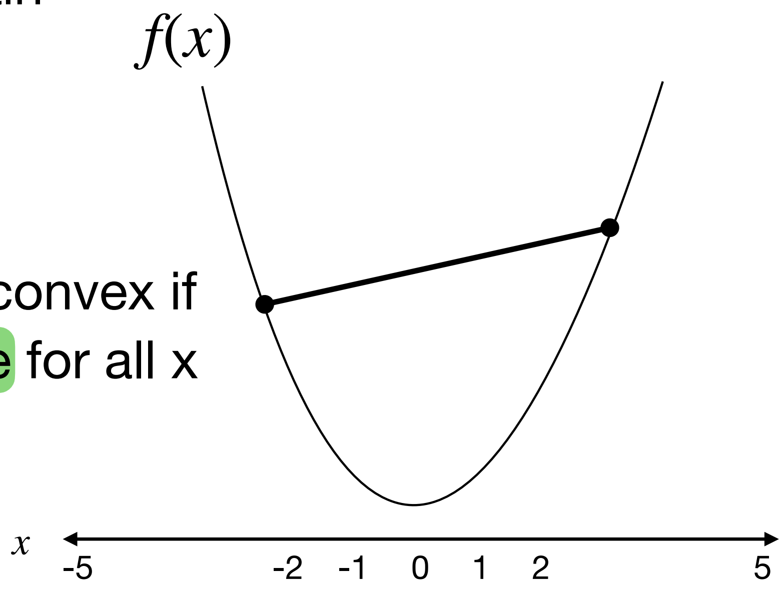
$$f(\lambda x + (1 - \lambda)y) \leq \lambda f(x) + (1 - \lambda)f(y)$$

- For more complicated functions, a function f is convex if the Hessian matrix $H(x)$ is positive semi-definite for all x

Second order derivative or derivative of the Jacobian

A matrix is positive semi-definite if and only if all of its eigenvalues are strictly greater than 0

symmetric



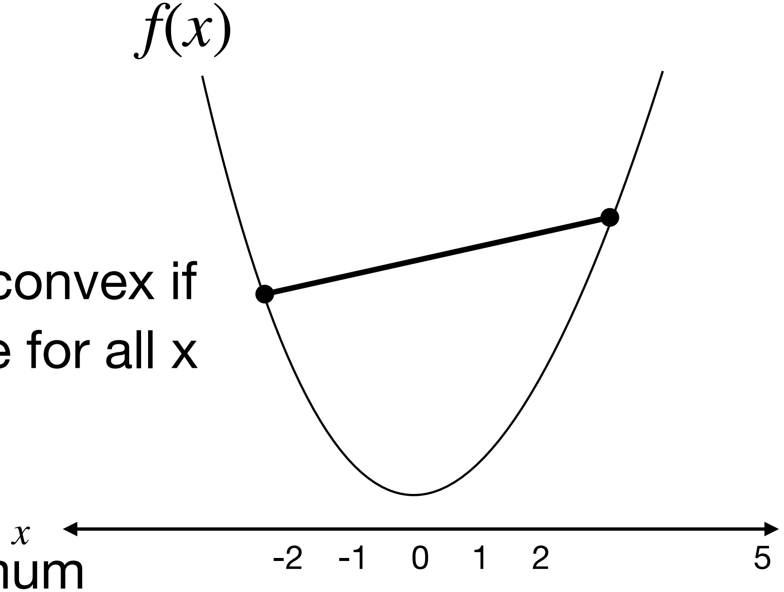
Optimizing Loss Functions
Gradient Descent - Convexity

- A function f is convex if for all points in its domain (input) and for all $\lambda \in [0,1]$

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda)f(y)$$

- For more complicated functions, a function f is convex if the Hessian matrix $H(x)$ is positive semi-definite for all x

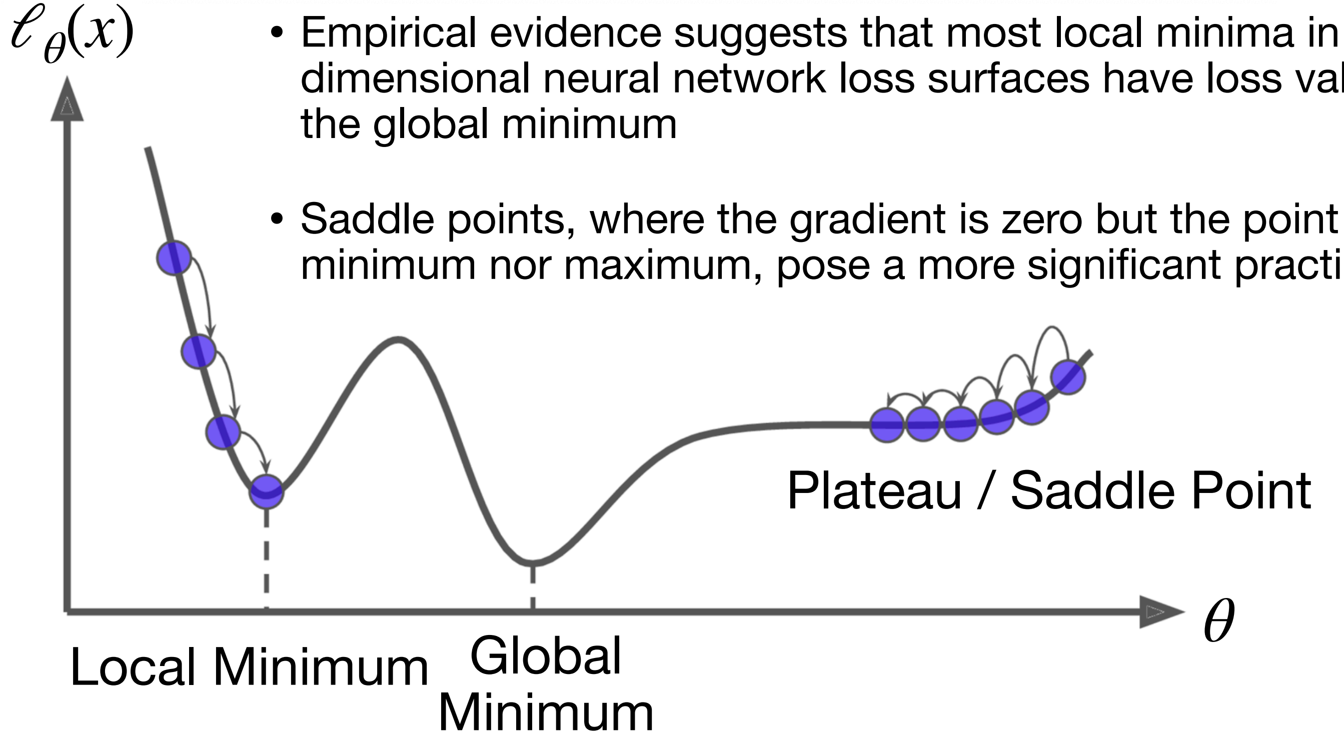
- If a function is convex, gradient descent is guaranteed to converge given the right learning rate since every local minimum is a global minimum



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.



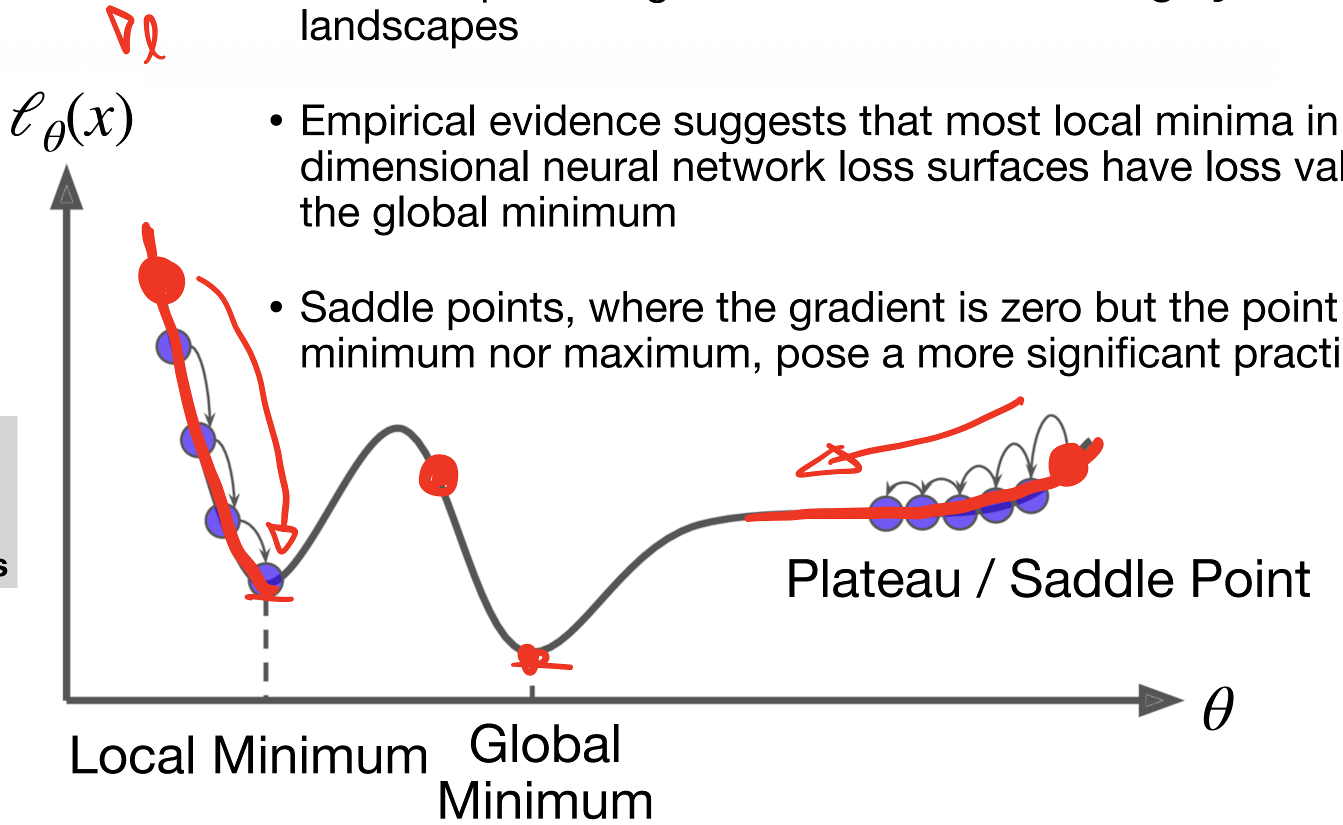
Optimizing Loss Functions

Gradient Descent - 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.

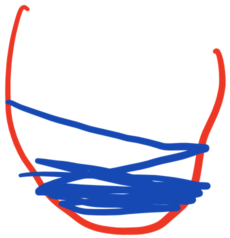
Initialization is an issue.

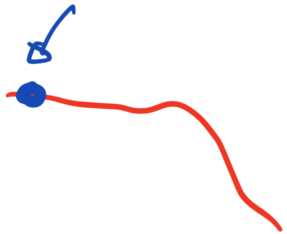
We will talk about it when we get to neural networks



Optimizing Loss Functions

Gradient Descent - Convergence Issues





- **Oscillation**: When the learning rate is **too large** or the loss surface has regions of high curvature, the algorithm oscillates around the minimum rather than converging smoothly.
- **Slow convergence in flat regions**: When gradients are small, parameter updates become negligible, leading to extremely slow progress.
- **Divergence**: If the learning rate exceeds a certain value for convex functions, the algorithm can **diverge** entirely, with the loss increasing without bound.
- Saddle points: In high dimensions, saddle points are ubiquitous. The **gradient at a saddle point is zero**, causing standard gradient descent to stall.

Optimizing Loss Functions

Gradient Descent - Practical Fixes

$$x \leftarrow \frac{x - \mu}{\sigma}$$

mean (μ)

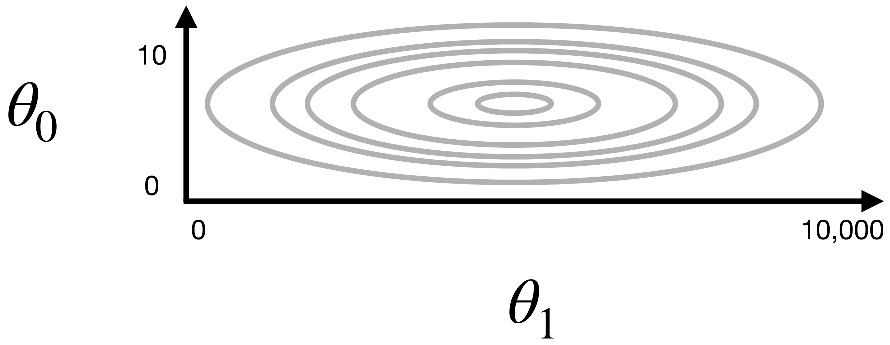
variance (σ)

- 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).

Optimizing Loss Functions

Gradient Descent - Practical Fixes

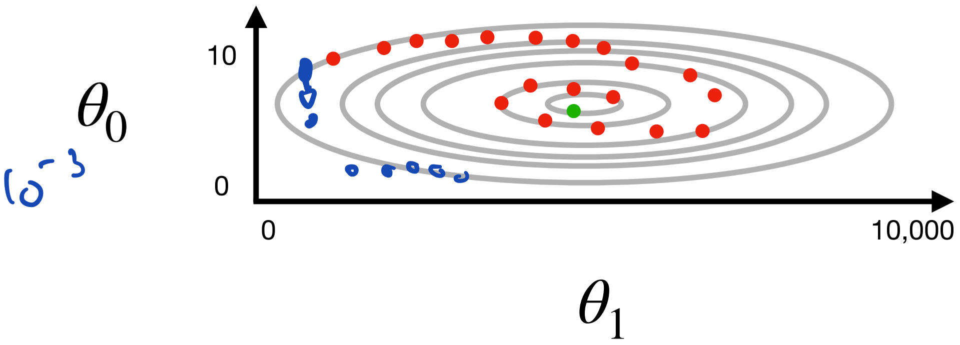
- Feature Scaling
  - Remember we want all input features $x_1, x_2 \cdots x_n$ to be in similar ranges
  - When features have different scales, the loss surface becomes elongated (ill-conditioned).



Optimizing Loss Functions

Gradient Descent - Practical Fixes

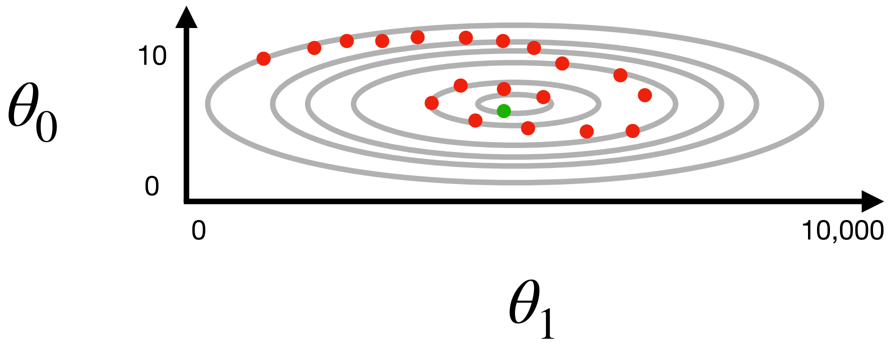
- Feature Scaling
  - Remember we want all input features $x_1, x_2 \cdots x_n$ to be in similar ranges
  - When features have different scales, the loss surface becomes elongated (ill-conditioned).

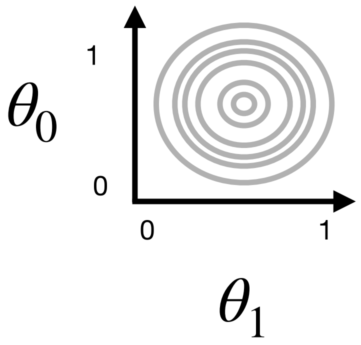


Optimizing Loss Functions

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).

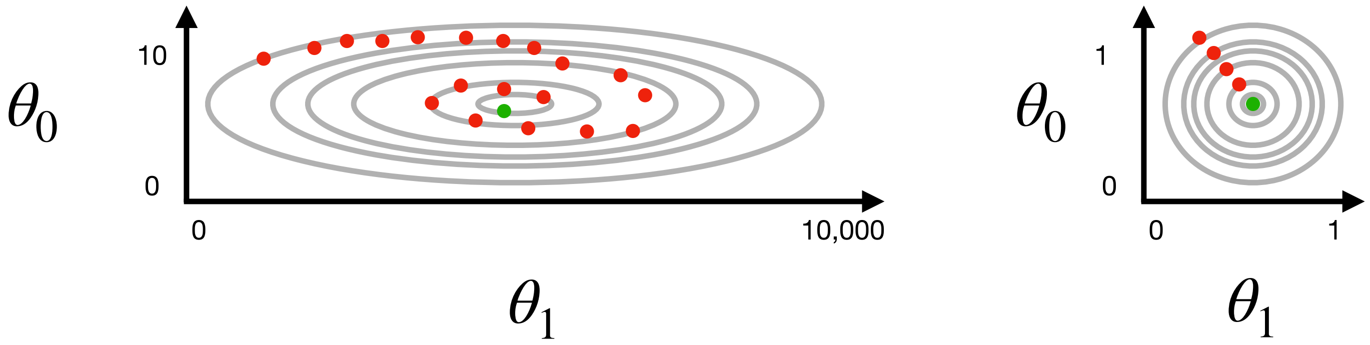




Optimizing Loss Functions

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

$x_{train} \leftarrow \frac{x - \mu_{train}}{\sigma_{train}}$

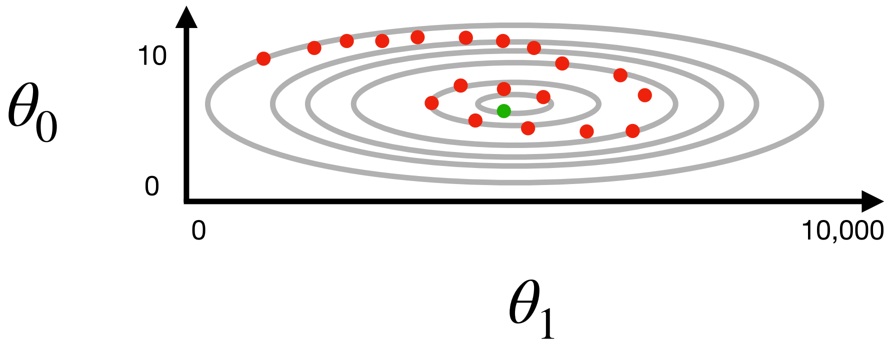
$x_{test} \leftarrow \frac{x - \mu_{train}}{\sigma_{train}}$

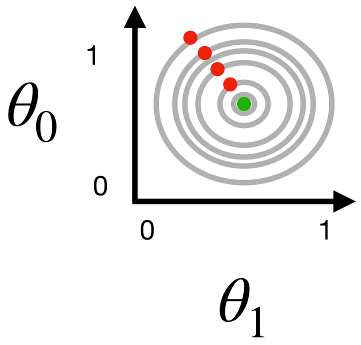
- Feature Scaling

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).

$\frac{x - x_{min}}{x_{max} - x_{min}}$



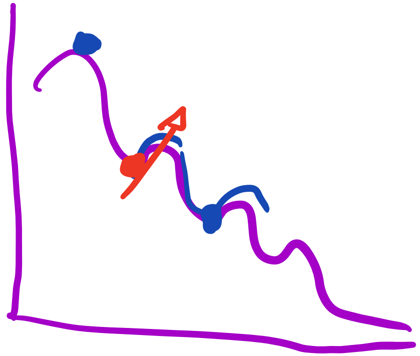


This dramatically accelerates the optimization process

This also allows having one single learning rate for all parameters

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

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

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

v_t

0.9

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

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

Velocity Vector

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

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

$V_{10} = \beta V_9 + \nabla \ell_{\theta_9}$

$V_9 = \beta V_8 + \nabla \ell_{\theta_8}$

$V_1 = \beta V_0 + \nabla \ell_0$

$V_2 = \beta V_1 + \nabla \ell_1$

$V_2 = \beta(\beta V_0 + \nabla \ell_0) + \nabla \ell_1$

$V_3 =$

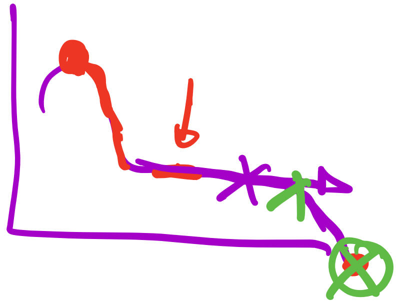
$V_{10} = \beta(V_9 + \cdot \nabla \ell\ \beta(V_8 + \cdot\cdot \nabla \ell\ \beta(V_7 + \cdot\ \nabla \ell)))$

Optimizing Loss Functions

Gradient Descent - Momentum

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

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

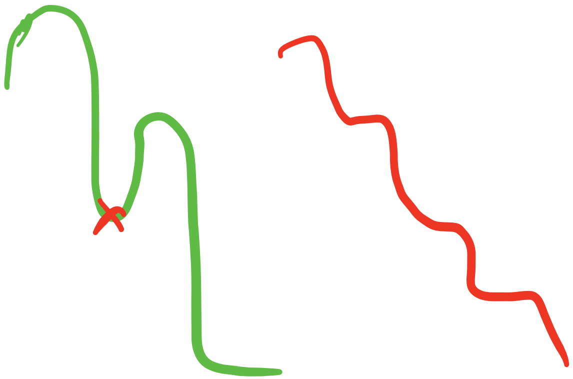


With Momentum

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**

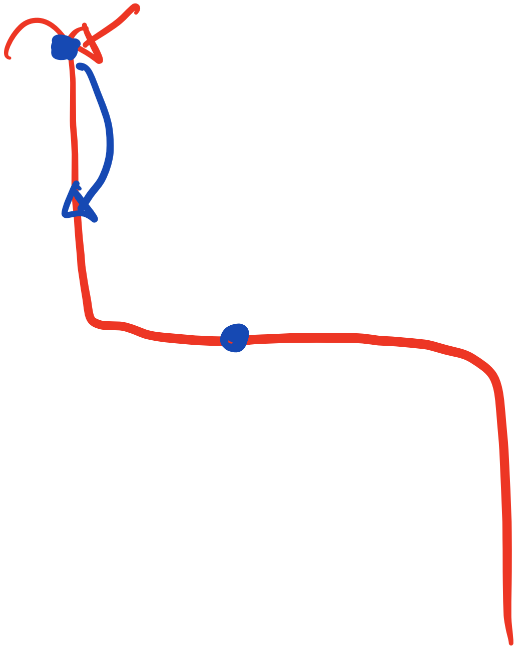
$$\theta_t = \theta_{t-1} - \alpha \nabla \ell_{\theta_{t-1}}$$

With Momentum

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

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

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



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

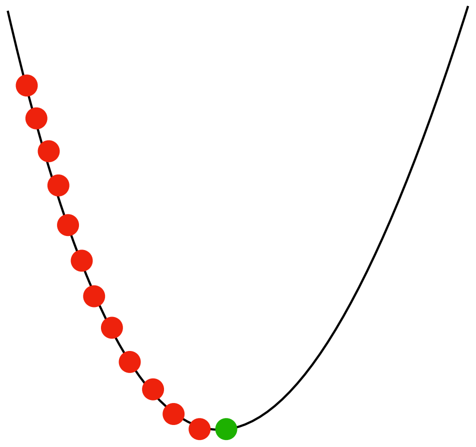
$$v_t = \beta \cdot v_{t-1} + \nabla \ell_{\theta_{t-1}}$$

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

Think of momentum as gravity pulling a ball down a hill, the momentum will carry the ball through any flat or even small uphill regions

Optimizing Loss Functions

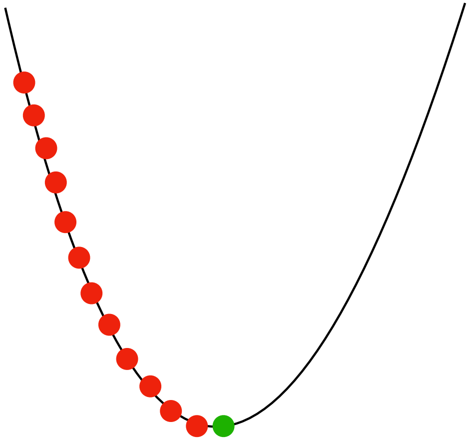
Gradient Descent - Adaptive Step Sizes



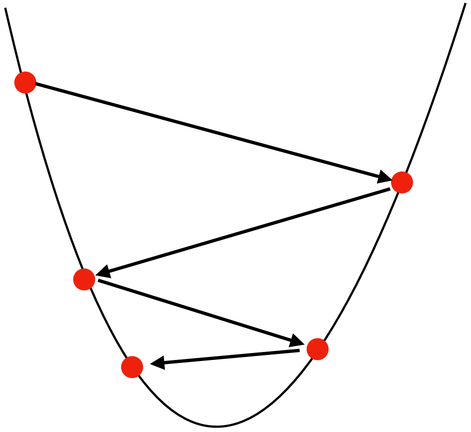
α is too small
Finds the optimal but too slow

Optimizing Loss Functions

Gradient Descent - Adaptive Step Sizes



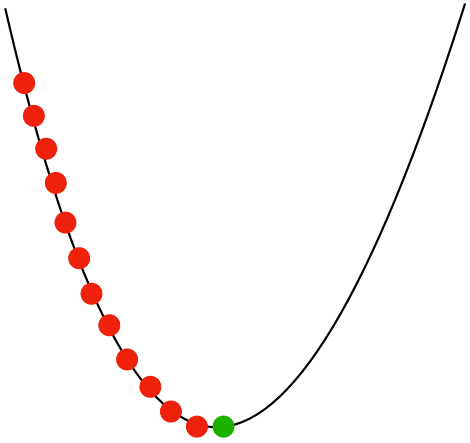
α is too small
Finds the optimal but too slow



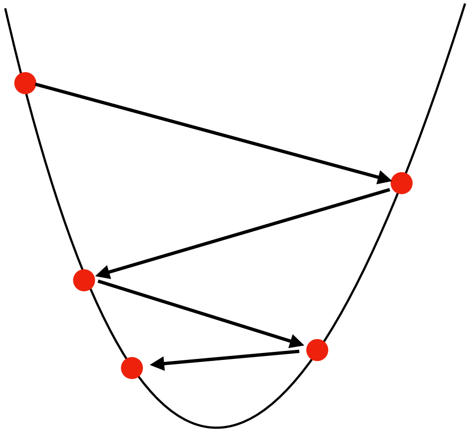
α is too large
Might not find optimal
Could even begin to diverge

Optimizing Loss Functions

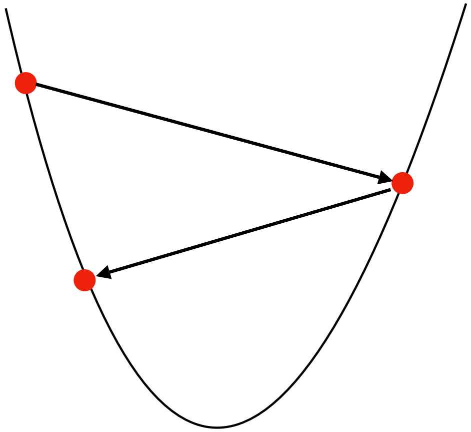
Gradient Descent - Adaptive Step Sizes



α is too small
Finds the optimal but too slow



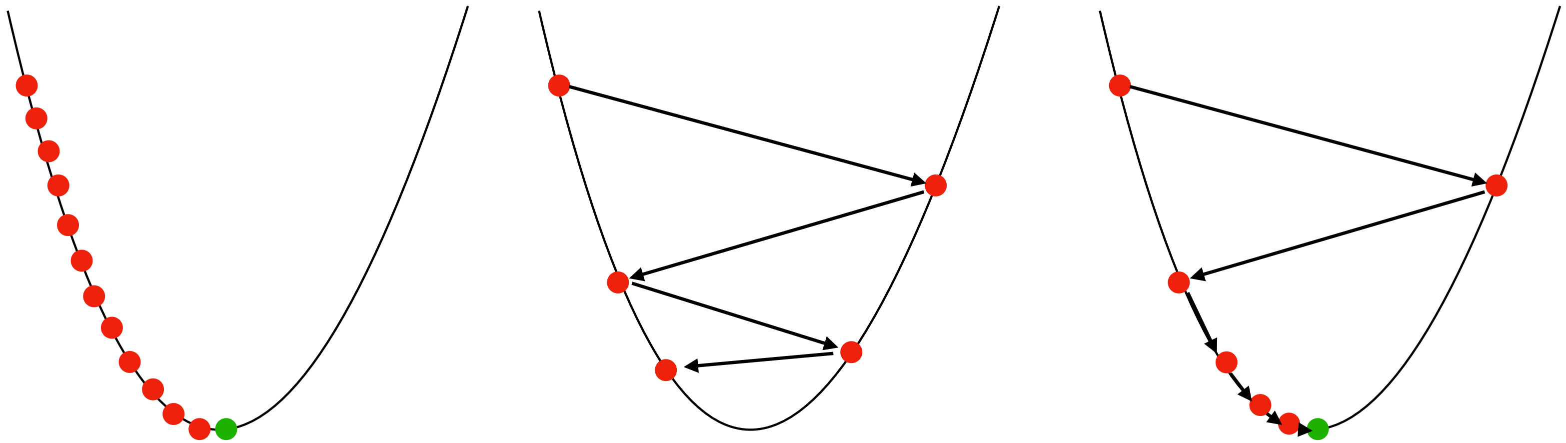
α is too large
Might not find optimal
Could even begin to diverge



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

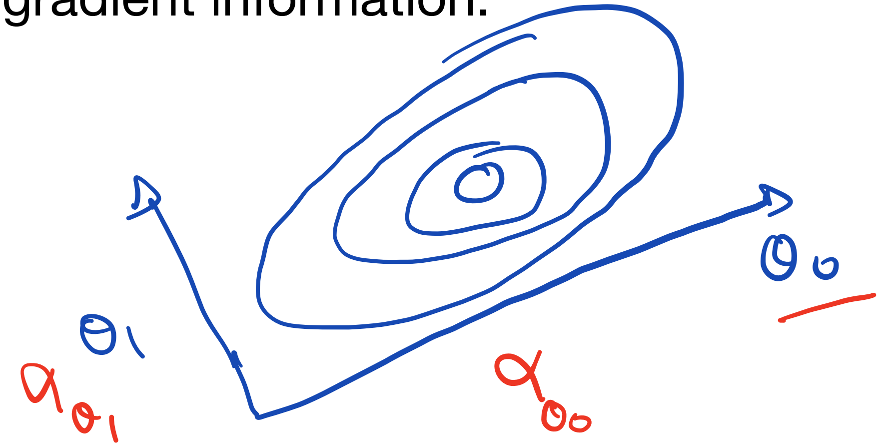
α is too large
Might not find optimal
Could even begin to diverge

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

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

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

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

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 large historical gradients receive smaller updates
  - Parameters with small historical gradients receive larger 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)

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

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

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

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

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

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

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

$x = {}_{1000}[\ ]$

- 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

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

Sum over the whole training dataset

$w_t \leftarrow w_{t-1} + \alpha \cdot \nabla l_{w_{t-1}}$

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

$x \in$ 1000

- 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

m=3

# bed	sq.ft.	y
5	5000	5000
4	4000	4900
1	650	2000

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

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

- Mini-Batch Gradient Descent
  - A compromise between batch and stochastic variants
  - Use a small batch of randomly sampled training data points
  - Typical batch sizes are $B = 32,\ 64,\ 128,\ 256,\ 512,\ 1024$

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

1000 [100] [100] [100]

1 epoch → 10 updates

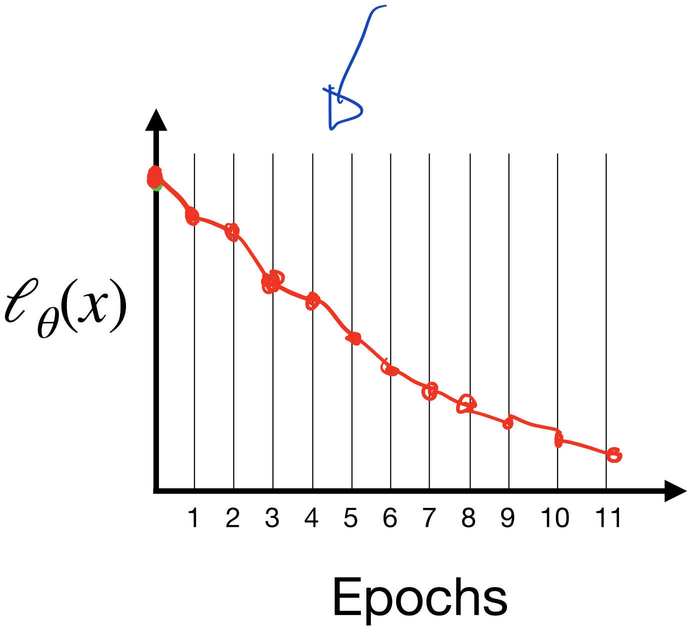
m/B

m: 1000

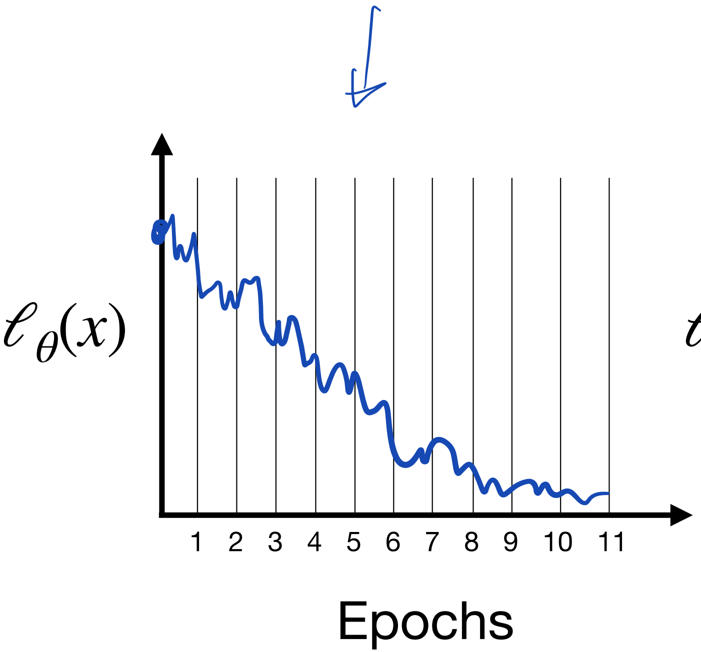
B = 100

Gradient Descent

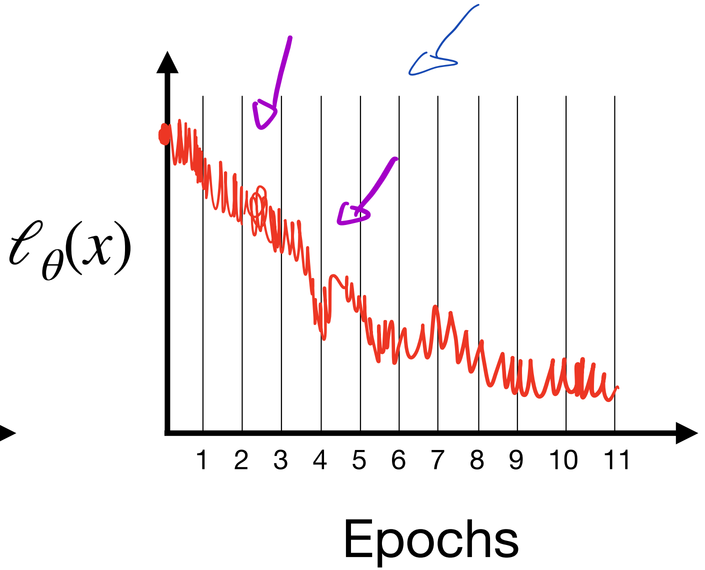
Batch vs Mini-Batch vs Stochastic Gradient Descent



Batch GD



Mini-Batch GD



SGD

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

Batch
Pros:

Stable Convergence: No noise in gradient estimates means smooth, predictable progress toward the minimum

Guaranteed Descent: Each update is guaranteed to reduce the loss (with appropriate learning rate)

Simple learning rate selection: The lack of noise means you can often use larger learning rates without instability

Parallelizable Gradient Computation: The sum over all samples can be computed in parallel across multiple processors

Stochastic
Pros:

Fast Updates: Each parameter update is computationally cheap, allowing rapid initial progress.

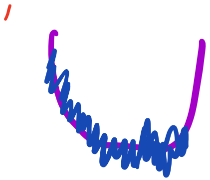
Memory Efficient: Only one sample needs to be in memory at a time.

Escapes Local Minima: The inherent noise helps the algorithm escape shallow local minima and saddle points. The stochasticity acts as implicit regularization

Online Learning: Can naturally incorporate new data as it arrives - just perform an update on each new sample

Better Generalization: The noise can prevent overfitting to the training set.

Gradient Descent



Batch vs Mini-Batch vs Stochastic Gradient Descent

Batch Cons:

Computationally Expensive: For large datasets, computing the full gradient is very slow. A dataset with 10 million samples requires processing all 10 million before a single update.

Memory Intensive: The entire dataset must fit in memory.

Redundant Computation: Many datasets contain redundant or similar samples. BGD computes gradients for all of them even when a subset would provide nearly the same information.

Poor Escape From Local Minima: The **deterministic** nature means the algorithm follows the same path every time and can get permanently stuck in local minima or saddle points.

Slow for Online Learning: Cannot incorporate new data without reprocessing everything.

Stochastic Cons:

High Variance: Individual gradient estimates can be very noisy, causing erratic updates.

Unstable Convergence: The loss curve is noisy. The algorithm may step away from the minimum even when near it.

Requires Learning Rate Decay: To converge to a minimum (rather than oscillating around it), the **learning rate must decrease** over time, adding hyperparameters.

Poor Hardware Utilization: Modern GPUs are optimized for **parallel operations on batches,** not sequential single-sample operations. SGD fails to exploit this.

Sensitive to Sample Ordering: The order in which samples are presented can affect results, requiring careful shuffling.

Gradient Descent

Batch vs Mini-Batch vs Stochastic Gradient Descent

Mini-Batch

Variance Reduction: Averaging over B samples reduces gradient variance by a factor of B compared to pure SGD, while still maintaining some beneficial noise

Hardware Efficiency: GPUs perform matrix operations in parallel. A batch size of 64 is nearly as fast as a batch size of 1 on modern hardware, giving essentially 64× speedup over SGD

Memory-Computation Tradeoff: Batch size can be tuned to maximize GPU memory utilization without requiring the full dataset

Balances Exploration and Exploitation: Enough noise to escape poor regions, enough signal to make consistent progress.

Gradient Descent

Gradient Descent vs Closed Form

$m < n^3$

$m < n^2$

Gradient Descent

+ Linear increase in m (# training data) and n (# features)

+ Generally applicable to multiple models

+ Guaranteed to reach global optimum for convex functions and appropriate learning rate

- Need to choose learning rate α and stopping conditions

- Need to choose optimization method (Adam, RMSProp etc..)

- Might get stuck in local optima / saddle point

- Needs feature scaling

Closed Form

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

+ No parameter tuning

+ Gives global optimum

- Not generally applicable to any learning algorithm

- Slow computation - scales with n^3 where n is number of features

Summary and Next Class

- Summary
  - We saw how gradient descent works
  - We saw issues with gradient descent and how to address them
  - We saw multiple optimizers commonly used in gradient descent
  - We saw types of gradient descent (batch, mini-batch, stochastic)
- Next Class - Classification, cross-validation and logistic regression