Northeastern University
Khoury College of
Computer Sciences

Recap

DS 4400 | Machine Learning and Data Mining I

Zohair Shafi

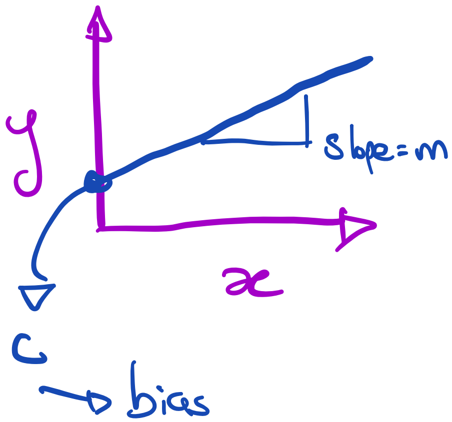
Spring 2026

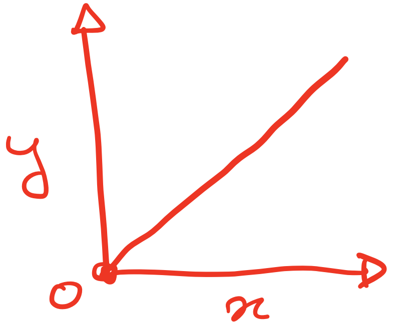
Monday | January 26, 2026

Linear Regression

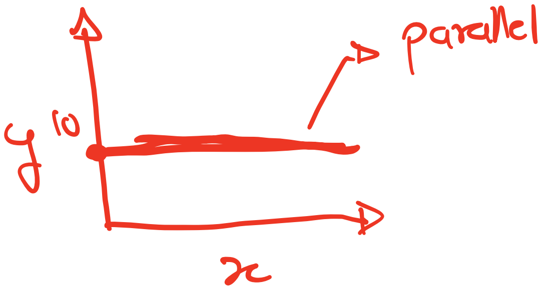
$y = mx + c$

slope   intercept



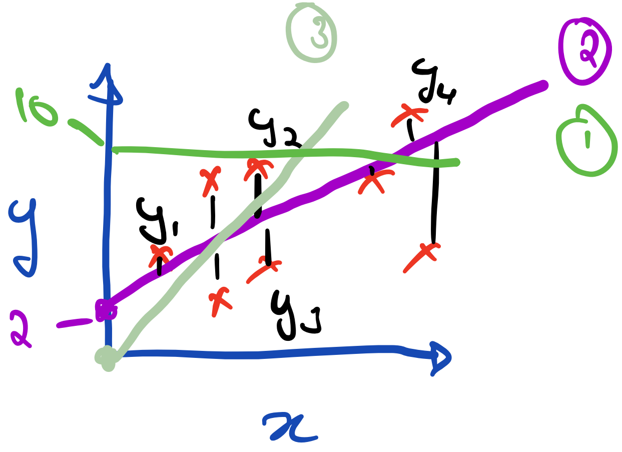


$y = mx + c$
$(1)x + 0$



$y = mx + c$
$y = 0 \cdot x + c$

Linear Regression



$m = 9$

$$y = \underbrace{m}_{\theta_1}x + \underbrace{c}_{\theta_0}$$

$$\hat{y} = \theta_1 x + \theta_0$$

input

predicted/output

$$\text{Loss} = \frac{1}{m}\sum_{i=0}^{m=9}\left(y_i - \hat{y}_i\right)^2$$

Cost

Error

$\mathcal{L}$, J , C

1 $\hat{y} = \theta_1 x + \theta_0$ — $\theta_1 = 0$, $\theta_0 = 10$

2 $\hat{y} = \theta_1 x + \theta_0$ — $\theta_1 = 2$, $\theta_0 = 2$

3 $\hat{y} = \theta_1 x + \theta_0$: $\theta_0 = 0$, $\theta_1 = 4$

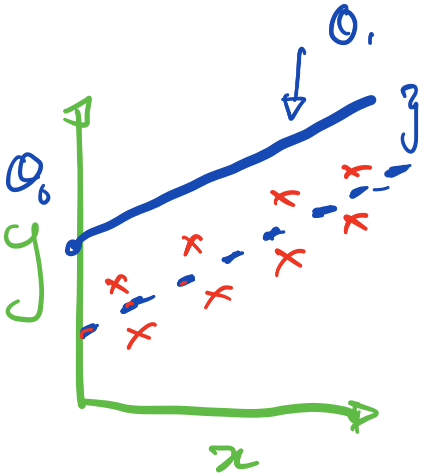
Linear Regression

Model: Line $\longrightarrow$ $\hat{y} = \theta_0 + \theta_1 \cdot x$

function — $f_\theta(x)$

θ — parameters

x — input



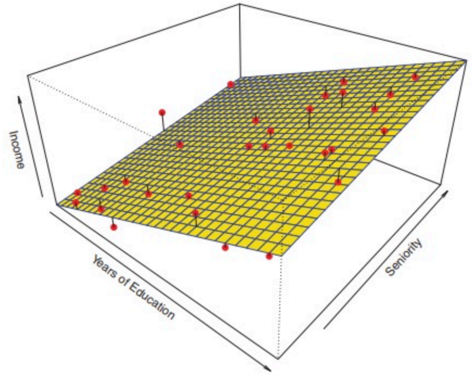
Minimize

$\boxed{\text{Loss:}}$ $L_\theta(x) = \frac{1}{m}\sum_{i=0}^{m}(y-\hat{y})^2$

$= \frac{1}{m}\sum_{i=0}^{m}\left(y-(\theta_0+\theta_1 x)\right)^2$

1. Compute derivative
2. Set deriative to zero
3. Solve for θ

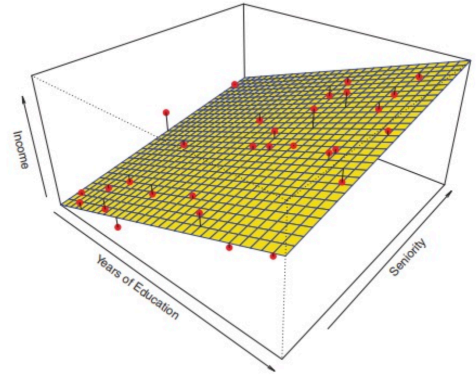
Linear Regression



- Linear Model

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

Linear Regression

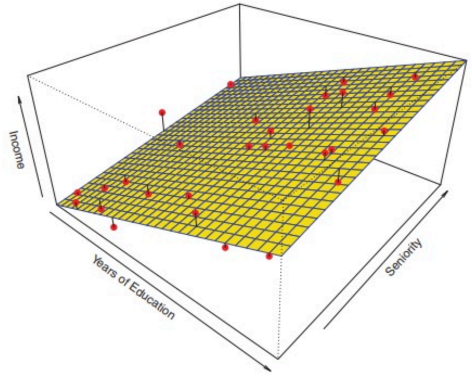


- Linear Model

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

Learnable parameters

Linear Regression



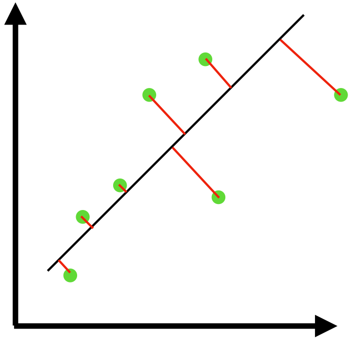
- Linear Model

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

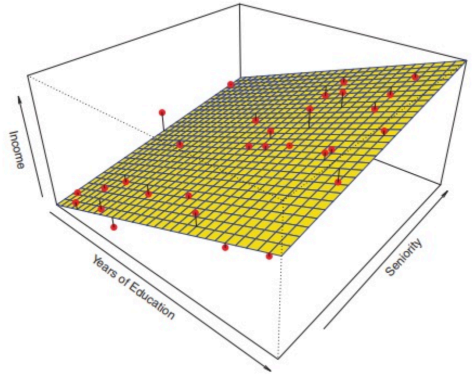
- Loss Functions (also called Cost Functions)

The red lines are called residuals

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



Linear Regression



- Linear Model

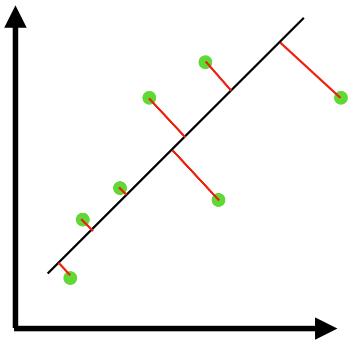
$$f_\theta(x) = \theta_0 + \theta_1 x_0 + \theta_2 x_1$$

- Loss Functions (also called Cost Functions)

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

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

The red lines are called residuals



Linear Regression

- Linear Model

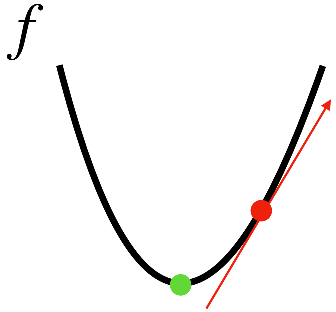
$$f_\theta(x) = \theta_0 + \theta_1 x$$

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

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

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

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



$\nabla f(\bullet) = 0$

Linear Regression

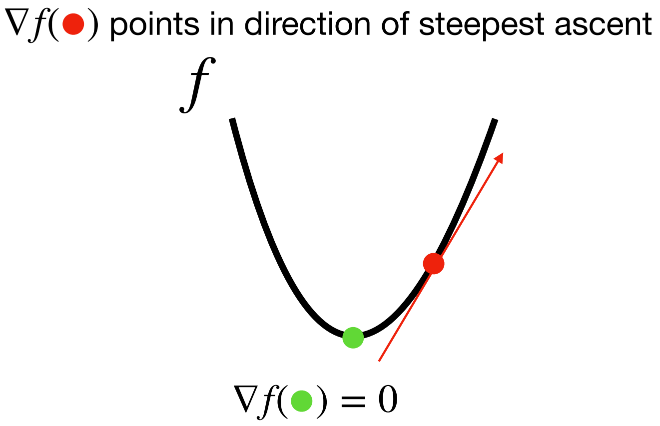
$\theta_0 \quad \theta_1$

$$\frac{\partial}{\partial \theta_1}(\theta_1 x) = x$$

$$L_{\theta}(x) = \frac{1}{m} \sum \underbrace{\left(y - (\theta_0 + \theta_1 x)\right)^2}$$

$$\frac{\partial L}{\partial \theta_0} = 2 \cdot \frac{1}{m} \sum \left(y - (\theta_0 + \theta_1 x)\right)(-1)$$

$$\frac{\partial L}{\partial \theta_1} = 2 \cdot \frac{1}{m} \sum \left(y - (\theta_0 + \theta_1 x)\right)(x)$$



Linear Regression

$y = \boxed{\theta_0} + \boxed{\theta_1 x}$

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

$\theta_0 \rightarrow \frac{\partial L}{\partial \theta_0} = 0$

$\theta_1 \rightarrow \frac{\partial L}{\partial \theta_1} = 0$

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

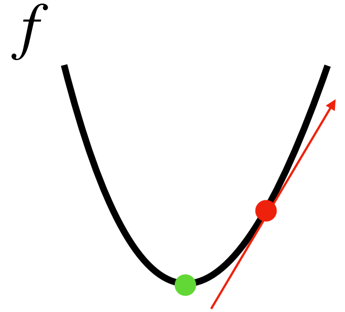
$$L(\theta) = \frac{1}{m}\sum_{i=1}^{m}[\theta_0 + \theta_1 \cdot x_i - y_i]^2$$

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

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

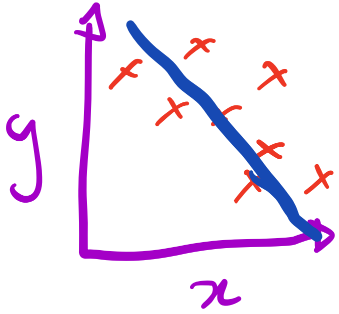
$$\frac{\partial L(\theta)}{\partial \theta_1} = \frac{2}{m}\sum_{i=1}^{m}x_i \cdot (\theta_0 + \theta_1 x_i - y_i) = 0$$

$\nabla f(\color{red}{\bullet})$ points in direction of steepest ascent



$\nabla f(\color{green}{\bullet}) = 0$

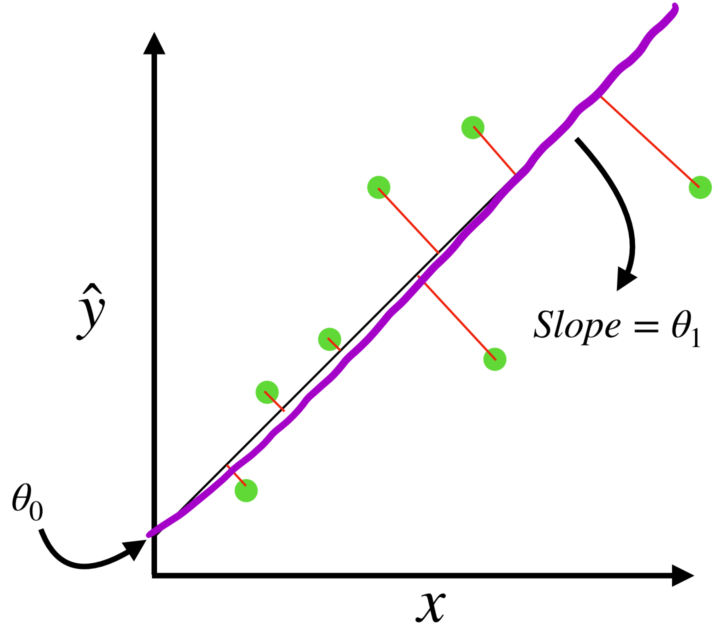
Linear Regression



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

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

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



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

Linear Regression

Solutions in Matrix Form

Loss $\rightarrow \frac{1}{m}\sum(y-\hat{y})^2$

$\frac{1}{m}\sum(y - x\theta)^2$ Predict

Loss

y	x Input
10	3
15	5

Train data.

$\hat{y} = \theta_0 + \theta_1 x$

$\hat{y} = \theta_0 + \theta_1(3) = 10$

$\hat{y} = \theta_0 + \theta_1(5) = 15$

y

$\hat{y} = \theta_0(1) + \theta_1(3) = 10$

$\hat{y} = \theta_0(1) + \theta_1(5) = 15$

$\theta_0(1) + \theta_1(3) = 10$

$\theta_0(1) + \theta_1(5) = 15$

$$\hat{y} : \begin{bmatrix}1 & 3\\ 1 & 5\end{bmatrix}\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix} = \begin{bmatrix}10\\ 15\end{bmatrix}$$

$X \in \underbrace{n}_{\#train}\times\underbrace{n}_{\#param}$

θ ~ #param × 1, n × 1

Y = #train × 1, m × 1

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 $\theta = \begin{bmatrix}\theta_0 \\ \theta_1\end{bmatrix} \in \mathbb{R}^{d\times 1}$

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

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

Quick Recap

Systems of Linear Equations - Linear Regression Example

θ_0 θ_1

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

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

$$
\begin{aligned}
(1) \cdot w_0 + (450) \cdot w_1 &= 2000 \\
(1) \cdot w_0 + (510) \cdot w_1 &= 2100 \\
(2) \cdot w_0 + (980) \cdot w_1 &= 2400 \\
(3) \cdot w_0 + (1500) \cdot w_1 &= 3000
\end{aligned}
\longrightarrow
\begin{bmatrix} 1 & 450 \\ 1 & 510 \\ 2 & 980 \\ 3 & 1500 \end{bmatrix}
\begin{bmatrix} w_0 \\ w_1 \end{bmatrix}
=
\begin{bmatrix} 2000 \\ 2100 \\ 2400 \\ 3000 \end{bmatrix}
$$

$X \in \mathbb{R}^{4\times 2}$ $\quad W \in \mathbb{R}^{2\times 1}$ $\quad y \in \mathbb{R}^{4\times 1}$

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

Linear Regression
Solution

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

$\boxed{\nabla L_\theta} \rightarrow$

We want to find the minimum so set gradient to zero

$\theta = \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$

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

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

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

$\theta_1 = \frac{\mathrm{cov}(x,y)}{\mathrm{var}(x)}$

If X^TX is invertible, then

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

Practical Issues in Linear Regression

Multicollinearity

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

$$5 \cdot \frac{1}{5} = 1$$

$$A \cdot A^{-1} = I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

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

↳ full rank matrix

↳ No linearly dependent rows or columns.

Practical Issues in Linear Regression

Multicollinearity

$x_0 \quad x_1$

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

  Cannot compute inverse.

  - $X^T X$ 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

$x_0 \rightarrow x_{10}$

- When two features are highly correlated or are linearly dependent on each other
- Why it's a problem: $\theta = (X^TX)^{-1}X^TY$

Simple Detection:
If correlation between features $\boxed{\geq 0.8}$

  - X^TX 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?

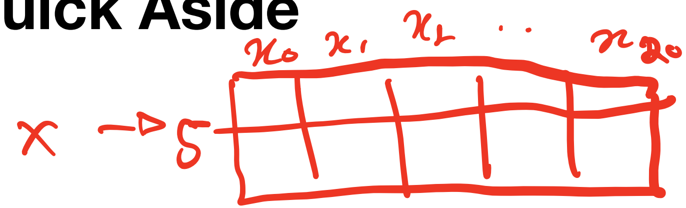
$$X \in \mathbb{R}^{m \times n}$$

m: Number of training examples

n: Number of parameters in the model

Practical Issues in Linear Regression

Quick Aside



x = $\underbrace{5 \times 20}$

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

m: Number of training examples

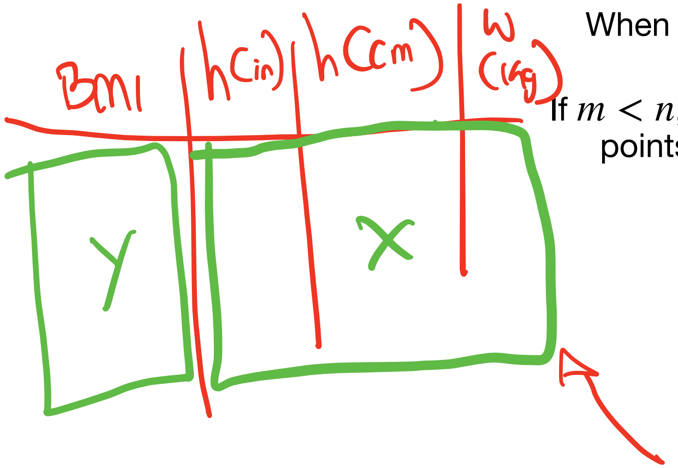
n: Number of parameters in the model

$rank(X) = min(m, n)$

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

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



1 in = 2.54cm

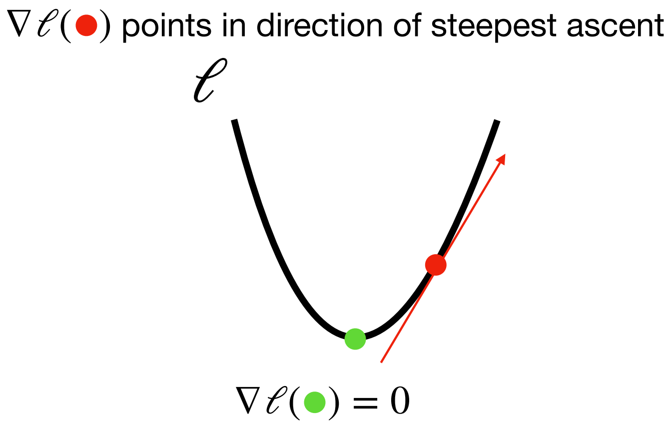
$\underbrace{y = 2.54x}$

cm

in

Gradient Descent: Optimizing Loss Functions

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



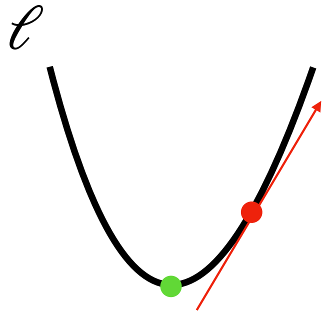
Optimizing Loss Functions

100

$\underbrace{(X^T X)^{-1}} \rightarrow O(n^3)$

- 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 $O(n^3)$

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



$\nabla \ell$(●) $= 0$

Optimizing Loss Functions

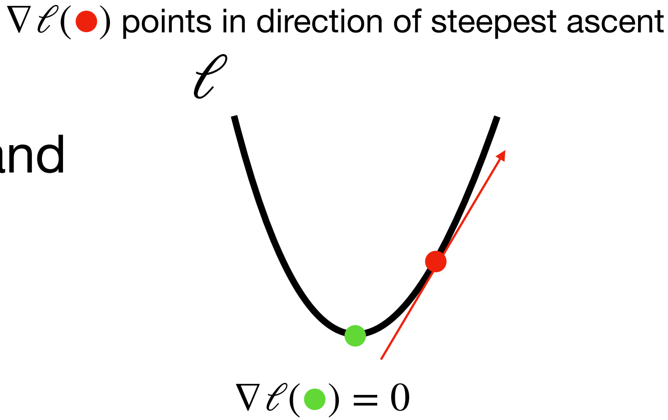
$O(m\,T\,n)$ # params

data # iterations

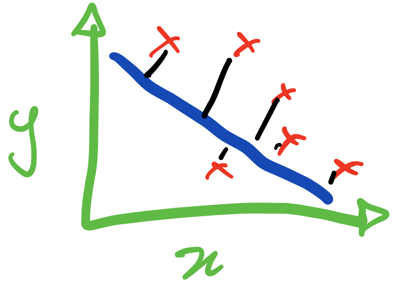
$\boxed{n^3}$

MSE: $\frac{1}{m}\sum(y-\hat{y})^2$

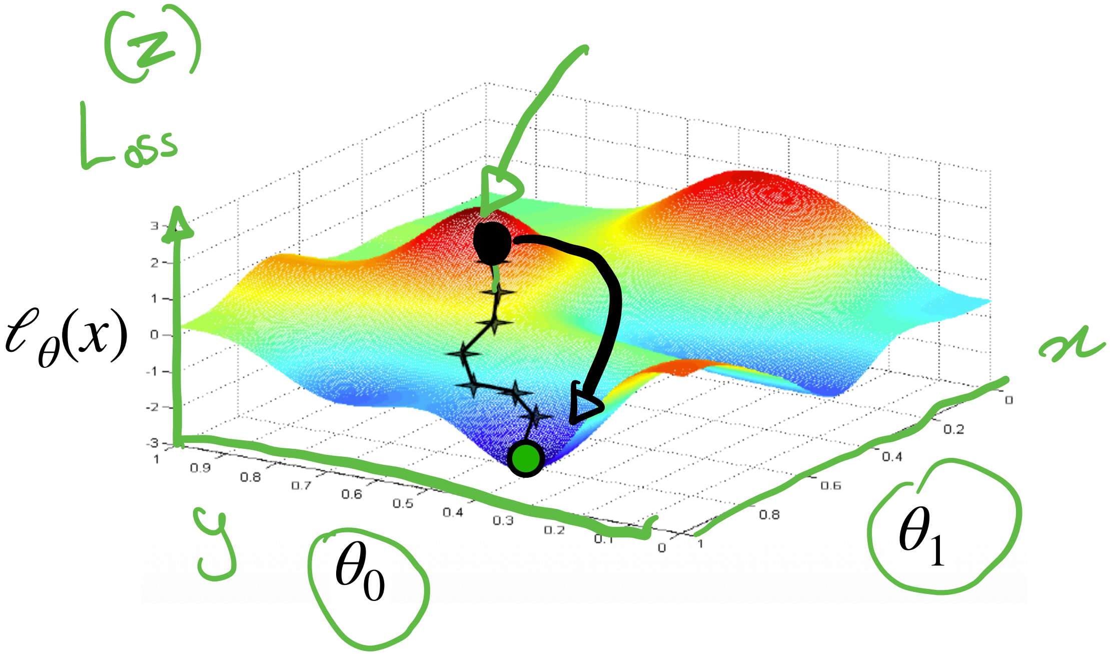
- This is where Gradient Descent comes in
  - Practical and efficient - has $\boxed{O(mTn)}$ where m is number of training points, T is number of epochs and n is number of features
  - Generally applicable to different loss functions
  - Convergence guarantees for certain types of loss functions (e.g.,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

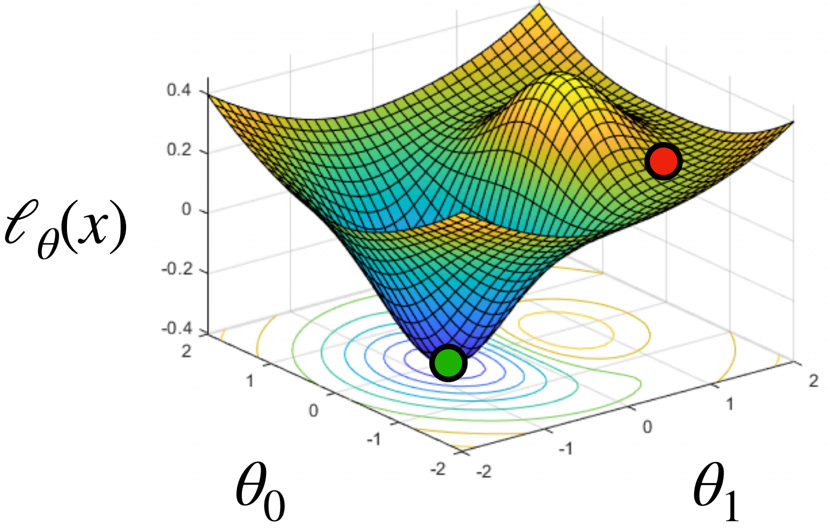


Optimizing Loss Functions

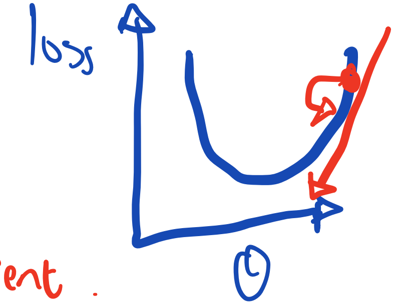


- What does the loss landscape look like with multiple learnable parameters?





Optimizing Loss Functions



1. Define loss function
2. Compute derivative.
3. Take a single step in the direction of the gradient.

(1) $L_\theta(x) = \frac{1}{m}\sum\left(y^{true} - \overbrace{(\theta_0+\theta_1 x)}^{\hat{y}}\right)^2$

(2) $\frac{\partial L_\theta(x)}{\partial \theta} = \nabla_\theta L(x)$

$\nabla_{\theta_0} L(x) = \frac{-2}{m}\sum\left(y-(\theta_0+\theta_1 x)\right)$

$\nabla_{\theta_1} L(x) = \frac{-2}{m}\sum\left(y-\theta_0+\boxed{\theta_1 x}\right)\cdot x$

(3) $\theta_{\boxed{0}} \leftarrow \theta_{\boxed{0}} - \alpha\cdot\nabla_{\theta_0} L(x)$

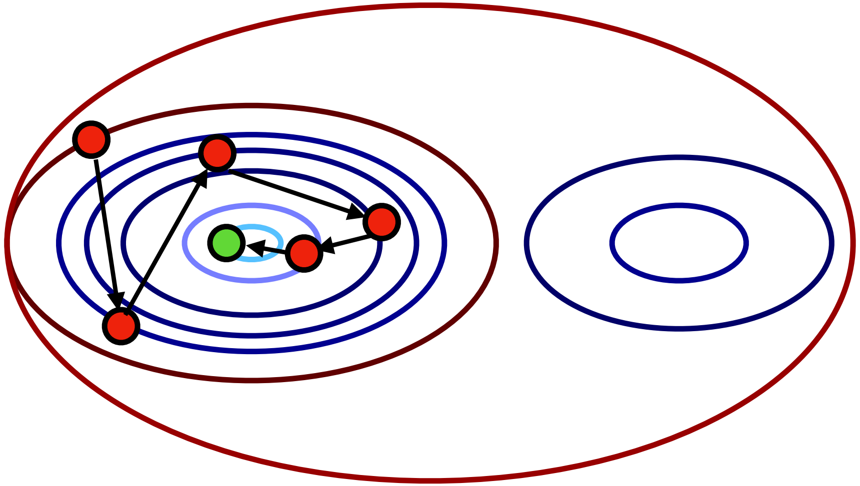
$\theta_1 \leftarrow \theta_1 - \alpha\cdot\nabla_{\theta_1} L(x)$

$\begin{bmatrix}\theta_0\\ \theta_1\end{bmatrix} \leftarrow \theta_{\boxed{t}} \leftarrow \theta_{t-1} - \alpha\cdot\nabla_{\theta_{t-1}} L(x)$

```
for t in range(0, 100)
    theta = theta - alpha * deriv.
```

Optimizing Loss Functions

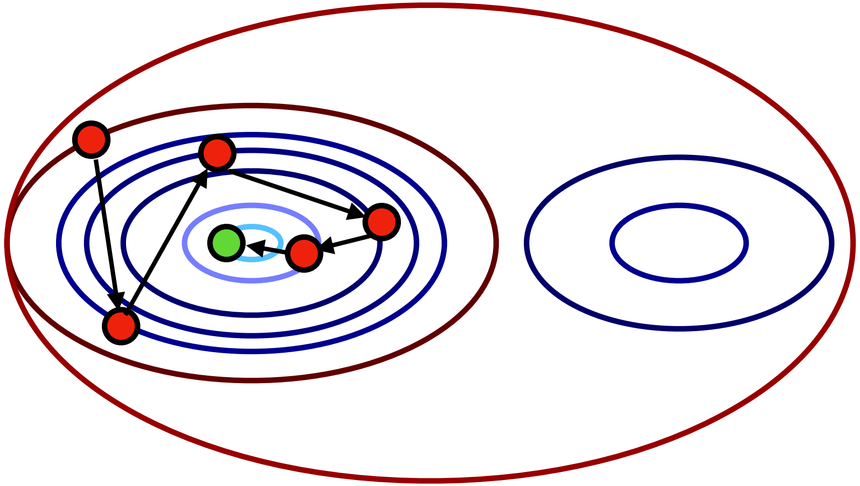
Gradient Descent - Formulation



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

Optimizing Loss Functions

Gradient Descent - Formulation

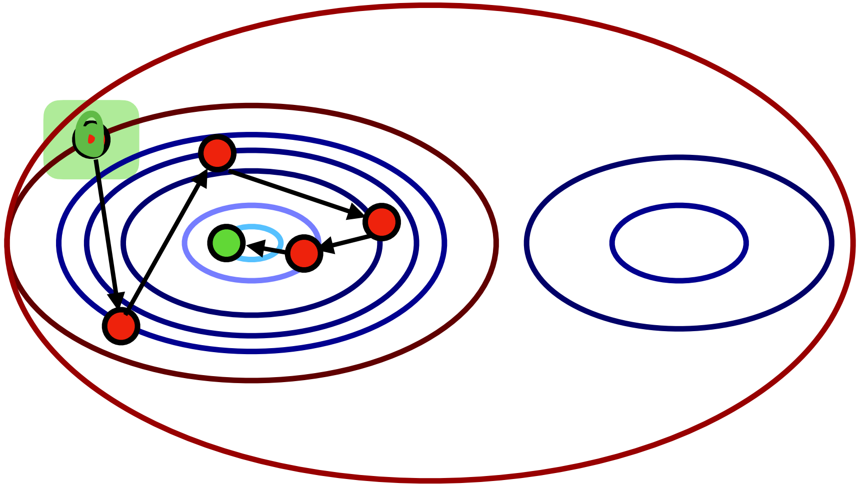


$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \overset{0.5}{\theta_0} - \overset{10}{\theta_1} x_i)^2$$

Step 1: Initialize θ_0, θ_1

Optimizing Loss Functions

Gradient Descent - Formulation



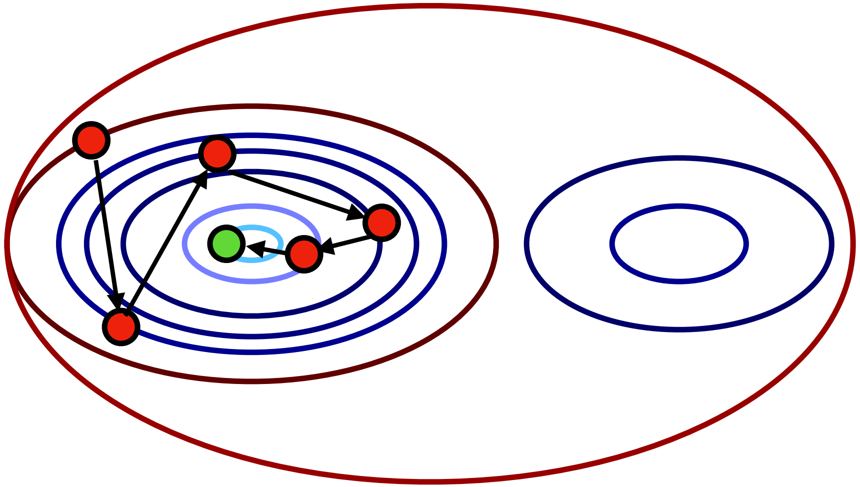
$$\ell_\theta(x) = \frac{1}{m}\sum_i (y_i - \theta_0 - \theta_1 x_i)^2$$

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



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

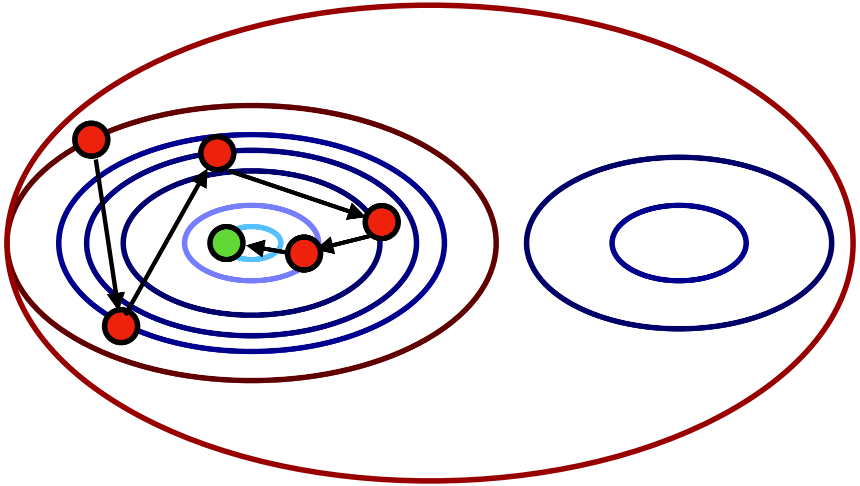
Step 1: Initialize θ_0, θ_1

Step 2: Repeat Until Convergence

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

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

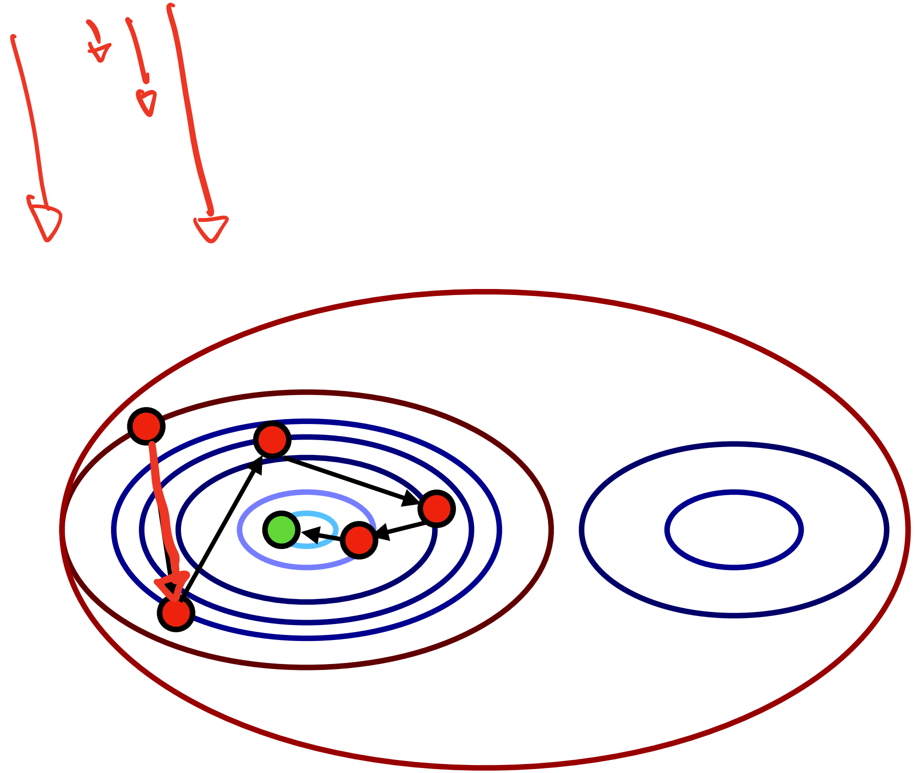
Step 2: Repeat Until Convergence

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

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

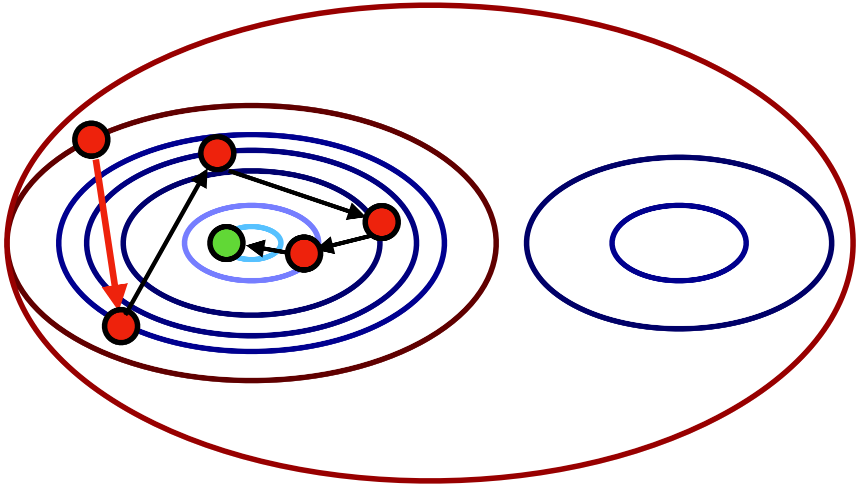
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

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

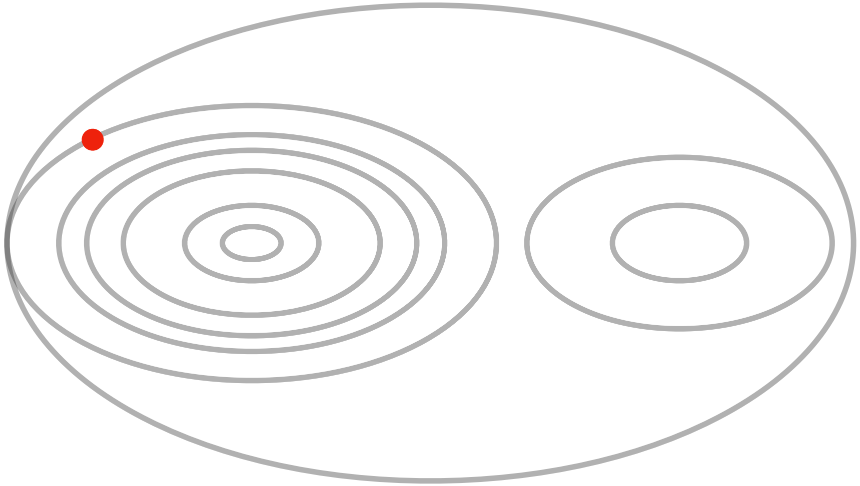
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

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

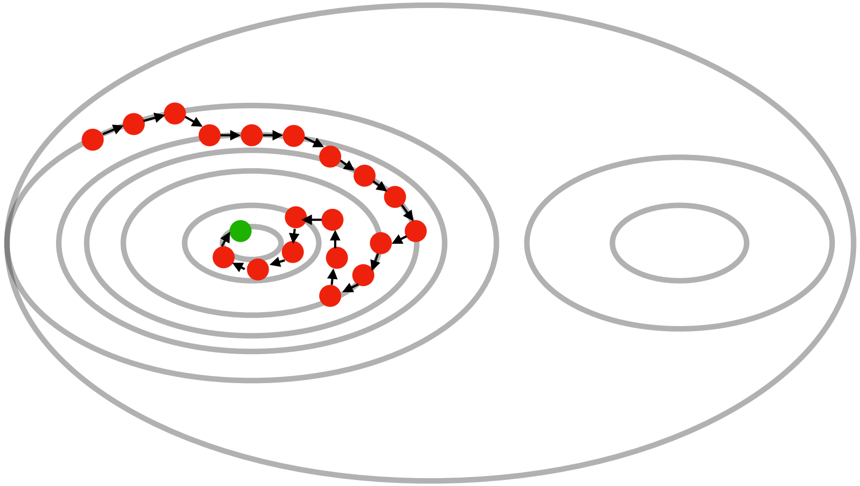
$$\theta_j \leftarrow \theta_j - \boxed{\alpha} \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

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

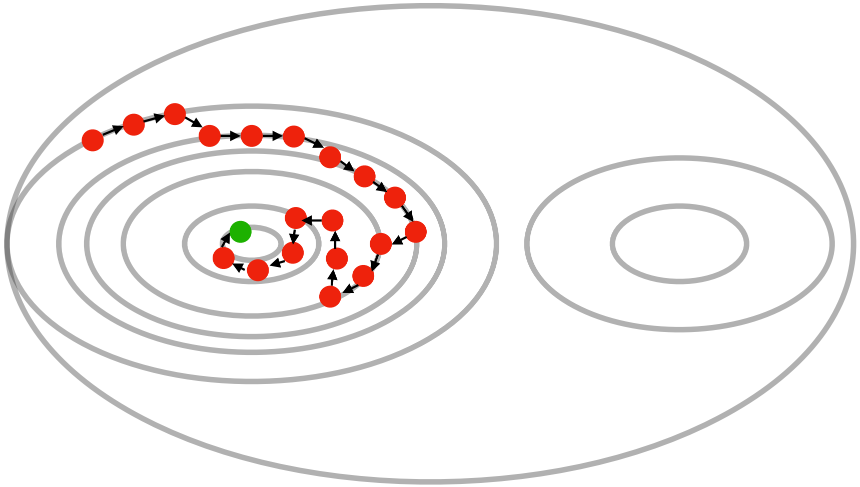
$$\theta_j \leftarrow \theta_j - \alpha \cdot \frac{\partial \ell_\theta(x)}{\partial \theta_j}$$

α : Learning Rate

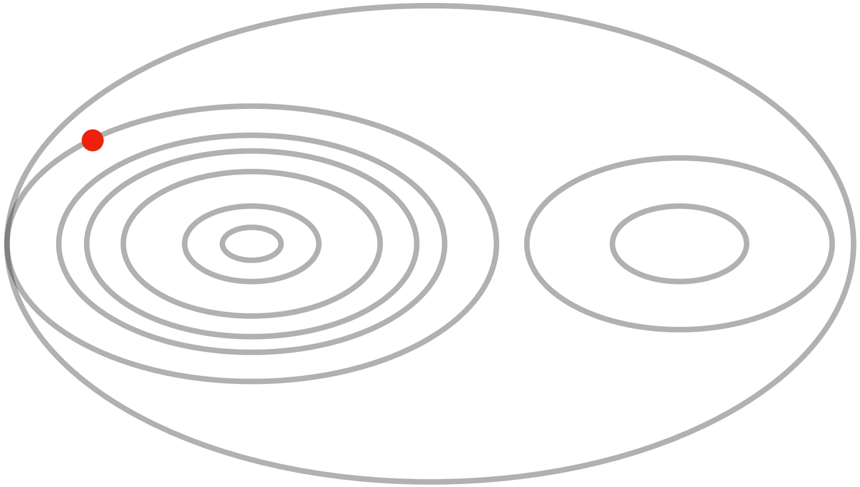
Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

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



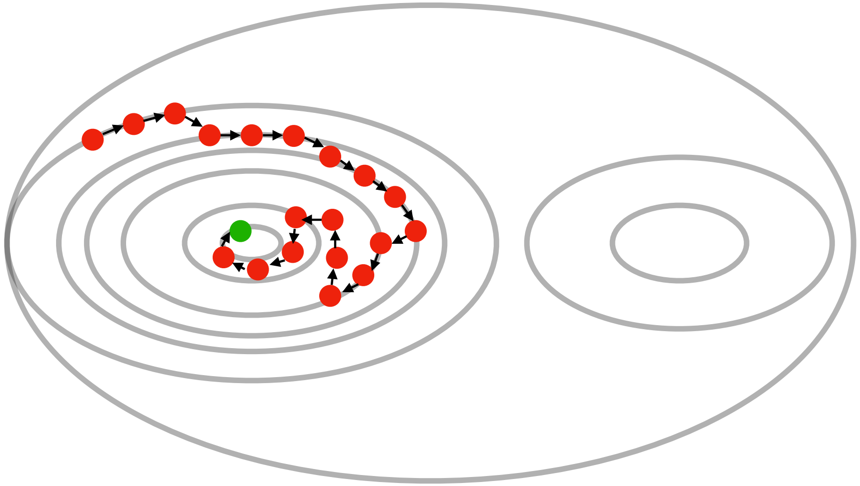
What happens when α is too large?
Say $\alpha = 10$



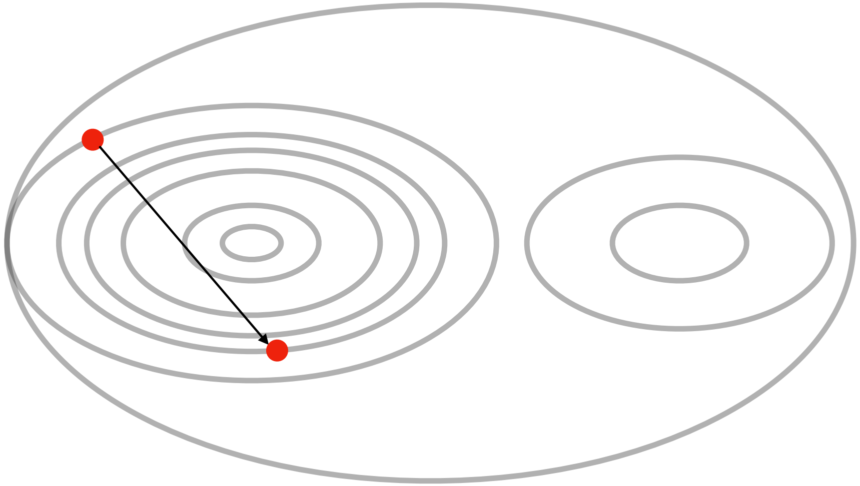
Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

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



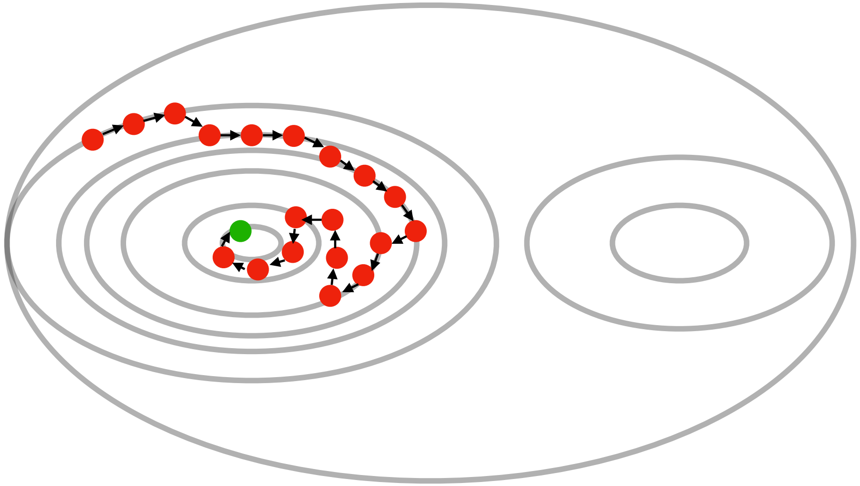
What happens when α is too large?
Say $\alpha = 10$



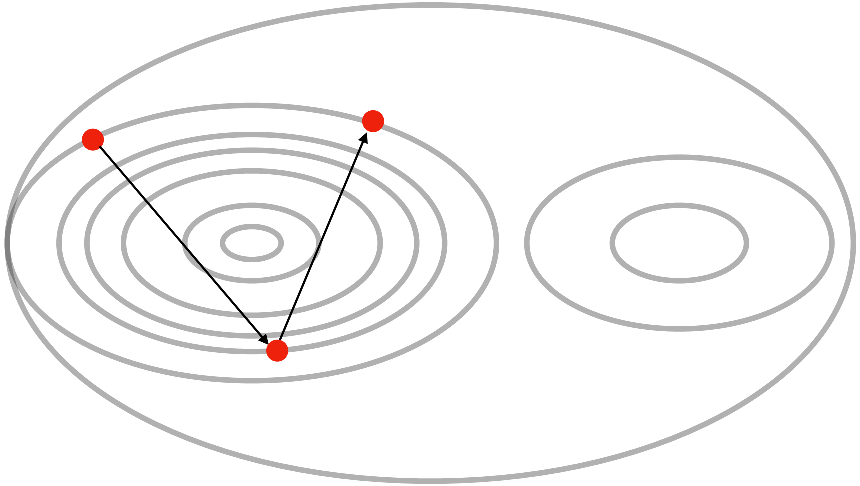
Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

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



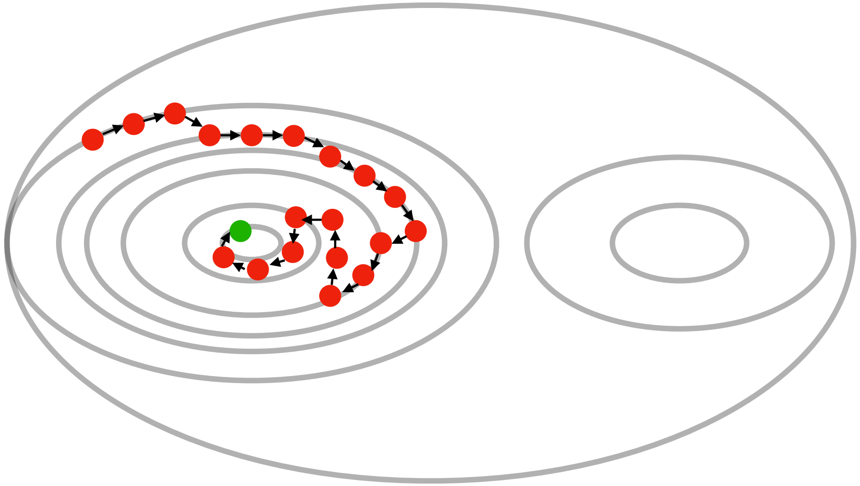
What happens when α is too large?
Say $\alpha = 10$



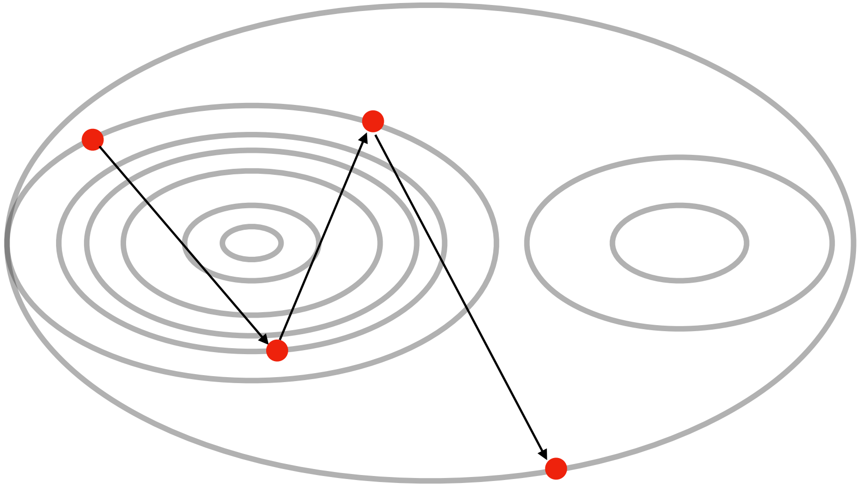
Optimizing Loss Functions

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 $\alpha = 10$

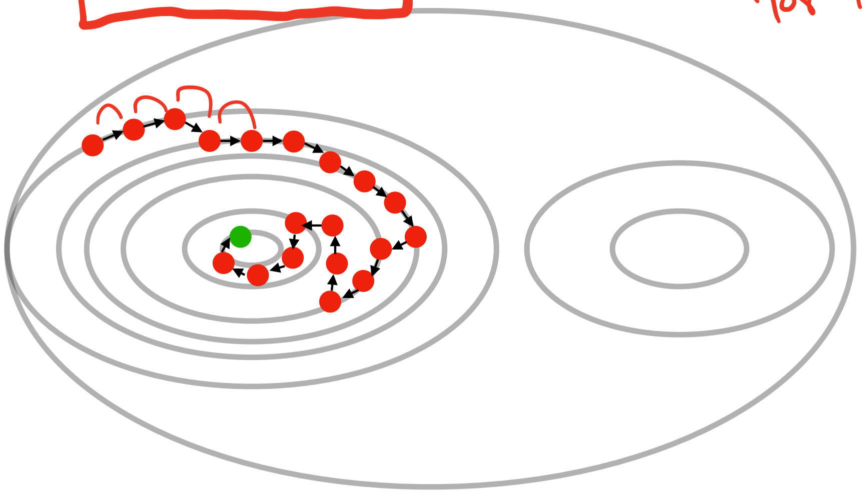


Optimizing Loss Functions

Gradient Descent - Effect of Learning Rate

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

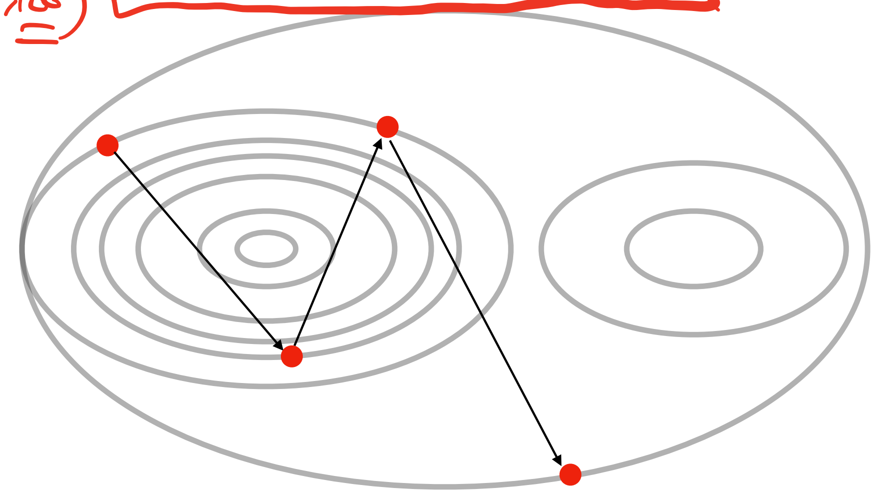
With a small learning rate α, if the loss function is convex, the optimization will eventually **converge**



for i in (0,100)

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



$$\theta_t \leftarrow \theta_{t-1} \underbrace{-\underbrace{\alpha}_{10^{-3}} \cdot \nabla L_{\theta_{t-1}}(x)}_{\nabla}$$

Optimizing Loss Functions

Gradient Descent - Stopping Criterion

Maximum Iteration

for i in range(1000)

epochs

1 whole pass through dataset.

Gradient Norm Threshold

$$\|\nabla L_{\theta_{t-1}}(x)\|_2 < \varepsilon \quad (\varepsilon \to 10^{-2})$$

Function Value Change

Loss.

$$|L_{\boxed{\theta_t}} - L_{\theta_{t-1}}| \leq \varepsilon \quad (\varepsilon \to 10^{-2})$$

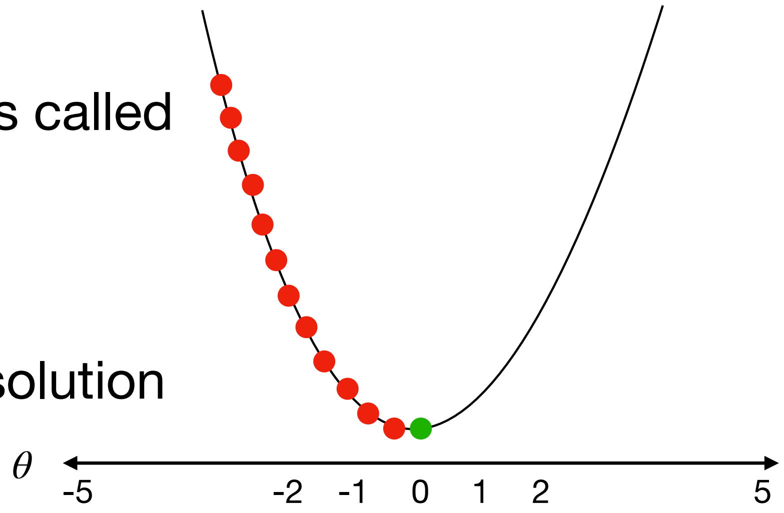
Parameter Value Change

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

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



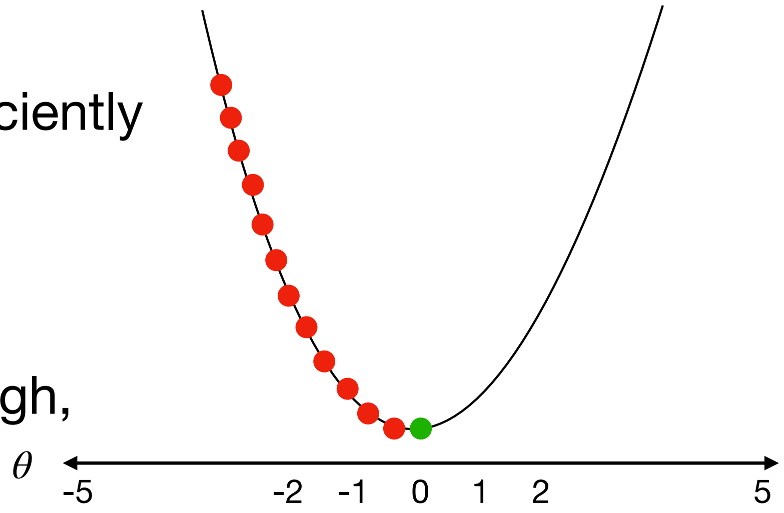
Optimizing Loss Functions

Gradient Descent - Stopping Criterion

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

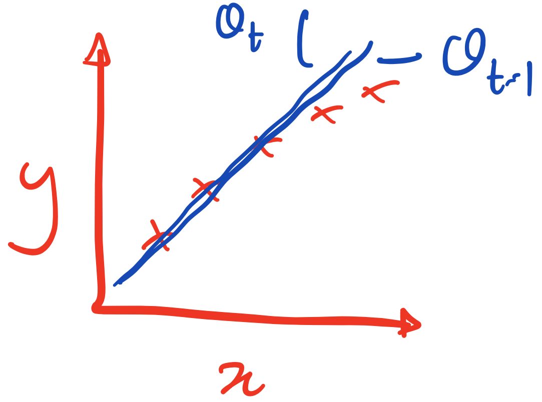
$$\|\nabla \ell_\theta(x)\|_2 \leq \epsilon$$

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



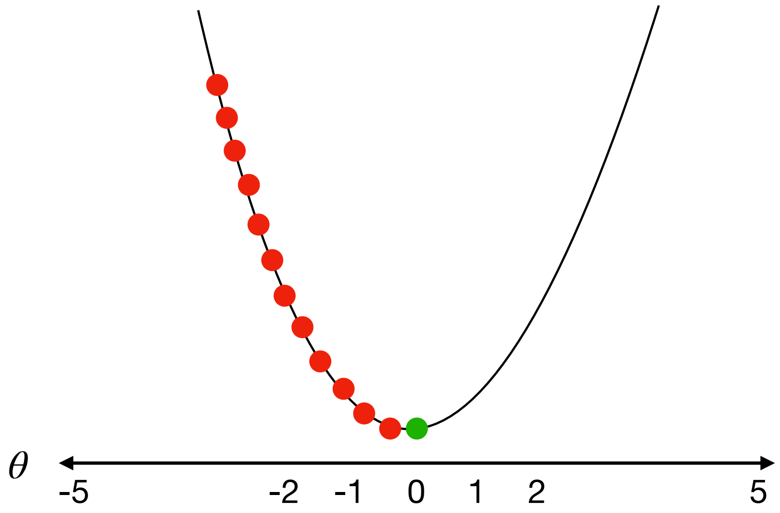
Optimizing Loss Functions

Gradient Descent - Stopping Criterion



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

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

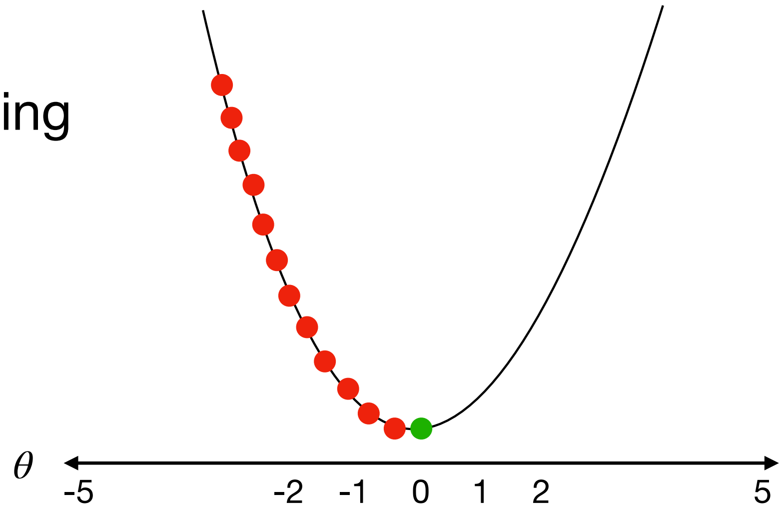


Optimizing Loss Functions

Gradient Descent - Stopping Criterion

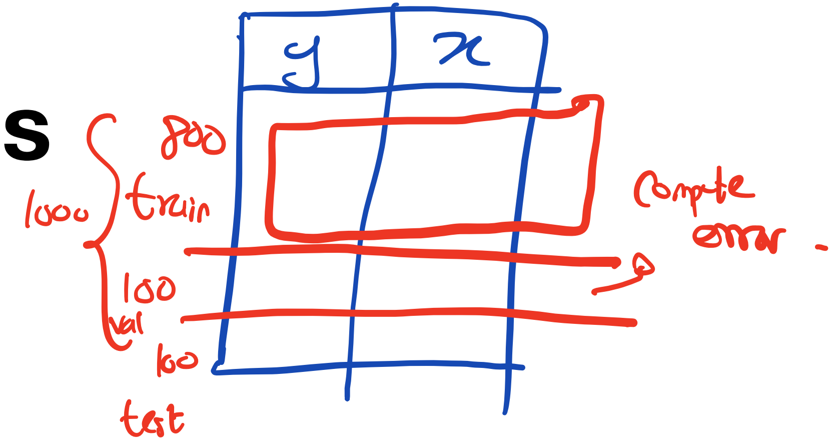
- When do you stop your iterations?
  - Parameter Value Change
    - Terminate when the parameters stop changing meaningfully

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

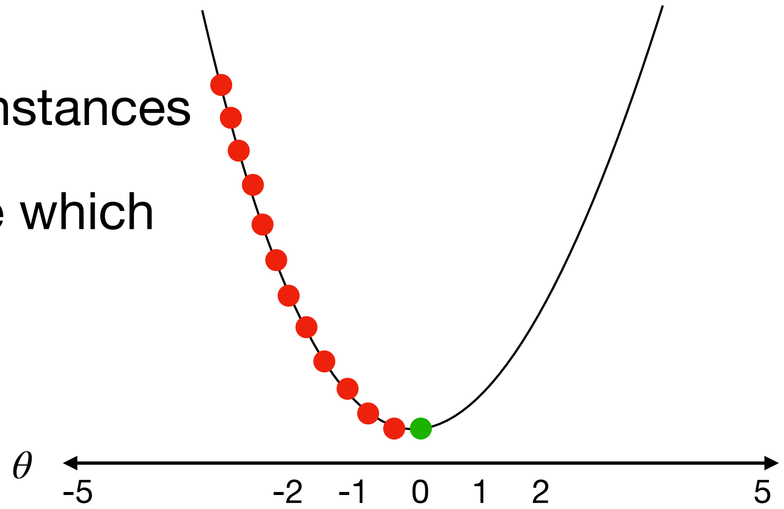


Optimizing Loss Functions

Gradient Descent - Stopping Criterion



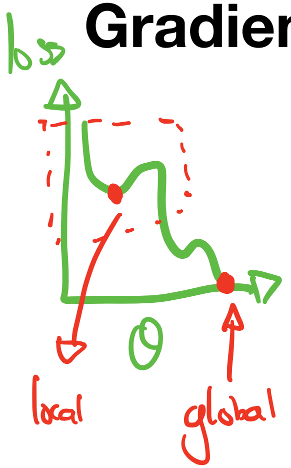
- When do you stop your iterations?
  - Validation Based Stopping (Early Stopping)
    - Monitor performance on a validation set of instances
    - Stop when validation loss begins to increase which signals overfitting
    - Serves as both stopping criterion and regularization



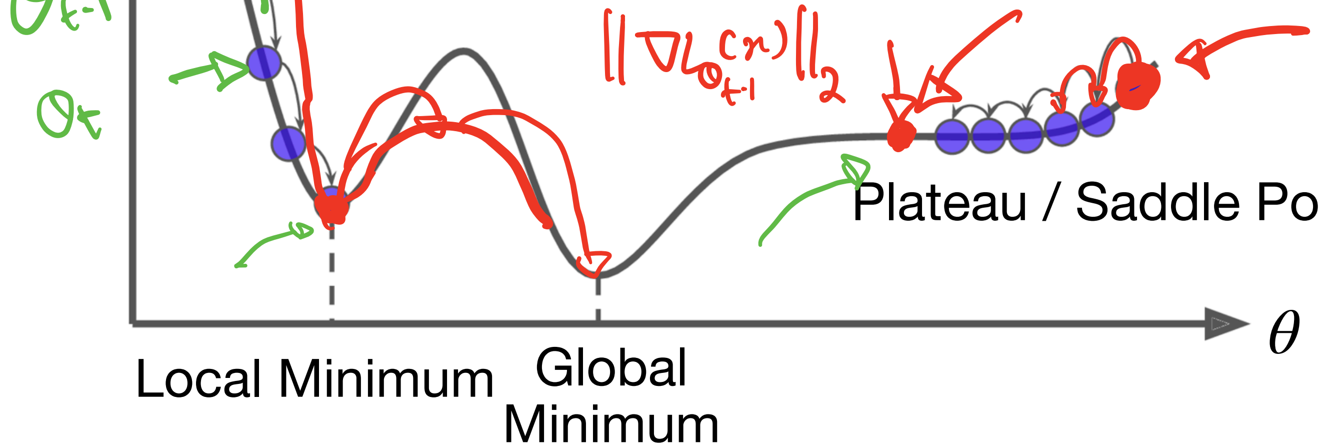
$$\theta_t \leftarrow \boxed{\theta_{t-1}} - \alpha \cdot \nabla_{\theta_{t-1}} L(x)$$

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

position at t-1

(1) G.D → $\theta_t \leftarrow \boxed{\theta_{t-1}} - \alpha \cdot \boxed{\nabla_{\theta_{t-1}} L(x)}$ → gets too small

(2) Problem → $\nabla_{\theta_{t-1}} L(x)$ gets too small.

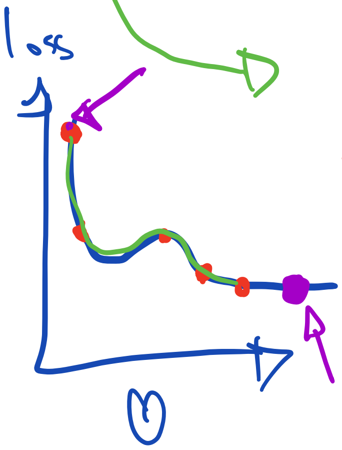
(3) Fix → Remember previous values and velocities.

$\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \underbrace{v_t}$ → does not get too small.

$v_t \leftarrow \boxed{\beta v_{t-1}} + \underbrace{\nabla_{\theta_{t-1}} L(x)}$

Sum of all gradients seen so far

→ 0.9 → hyper parameter



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**

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

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

Optimizing Loss Functions

Gradient Descent - Momentum

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

out of bounds.

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

With Momentum

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

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

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

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

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

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

Optimizing Loss Functions

Gradient Descent - 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

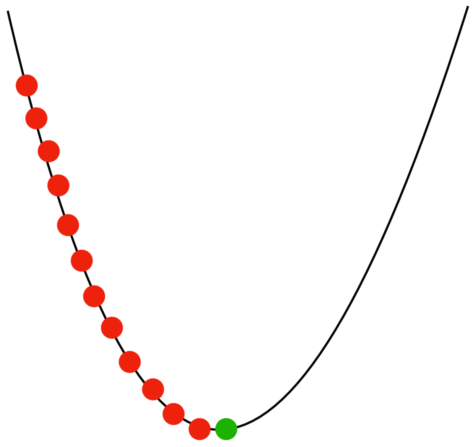
If $\beta = 0$, you get back standard gradient descent

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

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

Optimizing Loss Functions

Gradient Descent - Adaptive Step Sizes

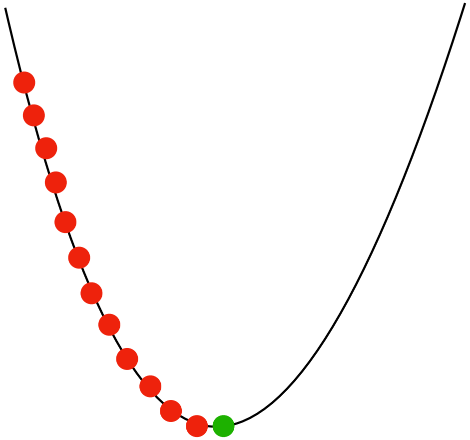


α is too small

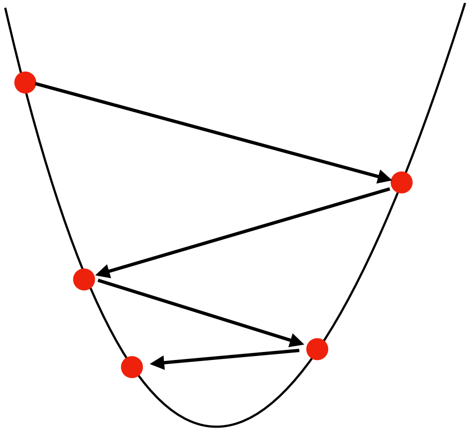
Finds the optimal but too slow

Optimizing Loss Functions

Gradient Descent - Adaptive Step Sizes



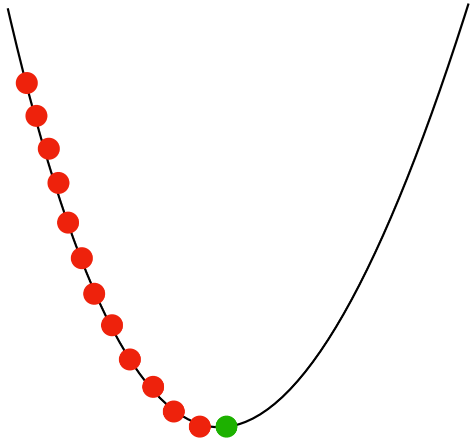
α is too small
Finds the optimal but too slow



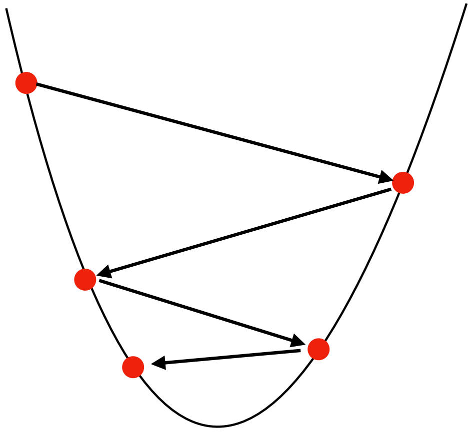
α is too large
Might not find optimal
Could even begin to diverge

Optimizing Loss Functions

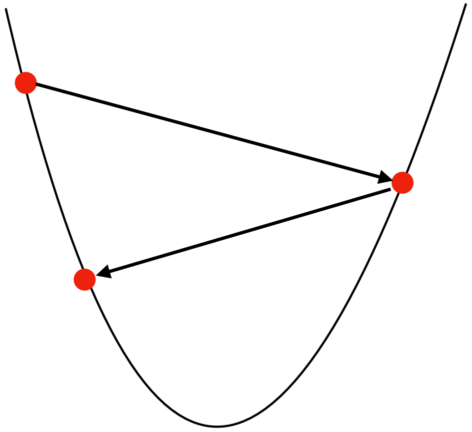
Gradient Descent - Adaptive Step Sizes



α is too small
Finds the optimal but too slow



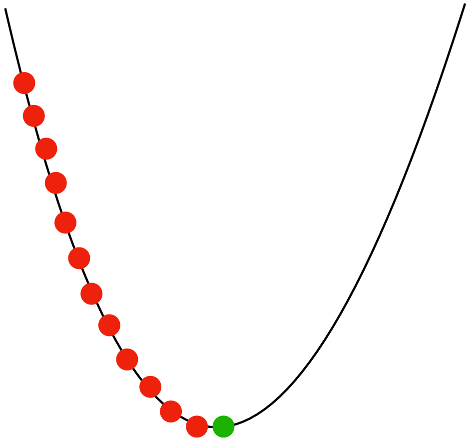
α is too large
Might not find optimal
Could even begin to diverge



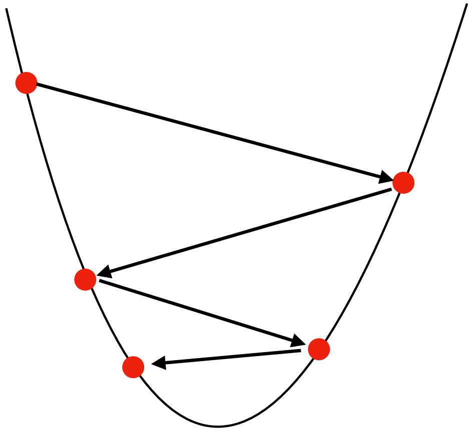
What if you set α to be large initially?

Optimizing Loss Functions

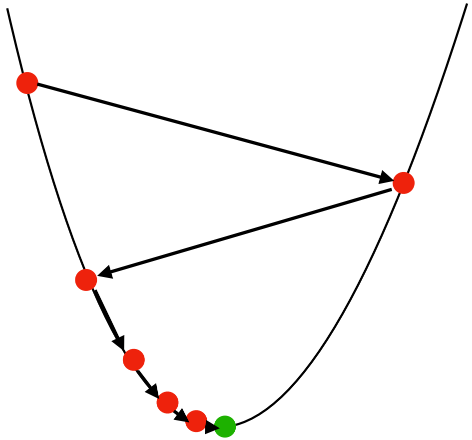
Gradient Descent - Adaptive Step Sizes



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.

$$\hat{y} = \theta_0 + \theta_1 x$$

$$\alpha$$

Optimizing Loss Functions

Gradient Descent - AdaGrad

① GD → $\theta_t \leftarrow \theta_{t-1} - \boxed{\alpha \cdot} \, \boxed{\nabla L_{\theta_{t-1}}(x)}$

momentum →

want to change.

always decreasing

$$\left[\ \theta_t \leftarrow \theta_{t-1} - \boxed{\frac{\alpha \cdot}{\sqrt{G_t + \epsilon}}} \nabla L_{\theta_{t-1}}(x) \right. \qquad \theta_0$$

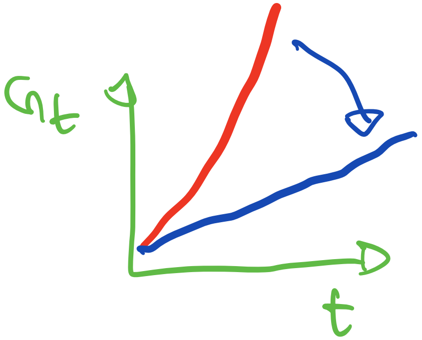
→ ?0

always increasing.

$$\boxed{G_t} \leftarrow G_{t-1} + \underbrace{\left[\nabla L_{\theta_{t-1}}(x)\right]^{2}}_{\text{always positive}}$$

Optimizing Loss Functions

Gradient Descent - $\boxed{\text{AdaGrad}}$ + RMSProp



$$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{G_t} + \epsilon} \cdot \nabla_{\theta_{t-1}} L(x)$$

$$G_t \leftarrow \beta G_{t-1} + (1-\beta)\left[\nabla_{\theta_{t-1}} L(x)\right]^2$$

↳ decay rate = 0.9

$$G_t \leftarrow 0.9 \cdot G_{t-1} + \underbrace{(0.1)}\boxed{\left[\nabla_{\theta_{t-1}} L(x)\right]^2}$$

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)

GD $\rightarrow$ $\theta_t \leftarrow \theta_{t-1} - \boxed{\alpha} \boxed{\nabla_{\theta_{t-1}} L(x)}$

want decay

Want momentum

$v_t \leftarrow \beta_1 v_{t-1} + (1-\beta_1) \nabla_{\theta_{t-1}} L(x)$

$\theta_t \leftarrow \theta_{t-1} - \frac{\alpha}{\sqrt{G_t + \epsilon}} \cdot v_t$

$G_t \leftarrow \beta_2 G_{t-1} + (1-\beta_2) [\nabla_{\theta_{t-1}} L(x)]^2$