# Gradient Descent (Reading: Ch 11)

**Learning goals:**

- See how gradients generalize derivatives.
- Learn the batch and stochastic gradient descent algorithms.
- Understand how GD is affected by learning rate, convexity, and loss function.

---

**UC Berkeley Data 100 Summer 2019**
**Sam Lau**

(Slides adapted from Sandrine Dudoit and Joey Gonzalez)

# Announcements

- HW4 due **today**
- HW5 out today, due **Friday**

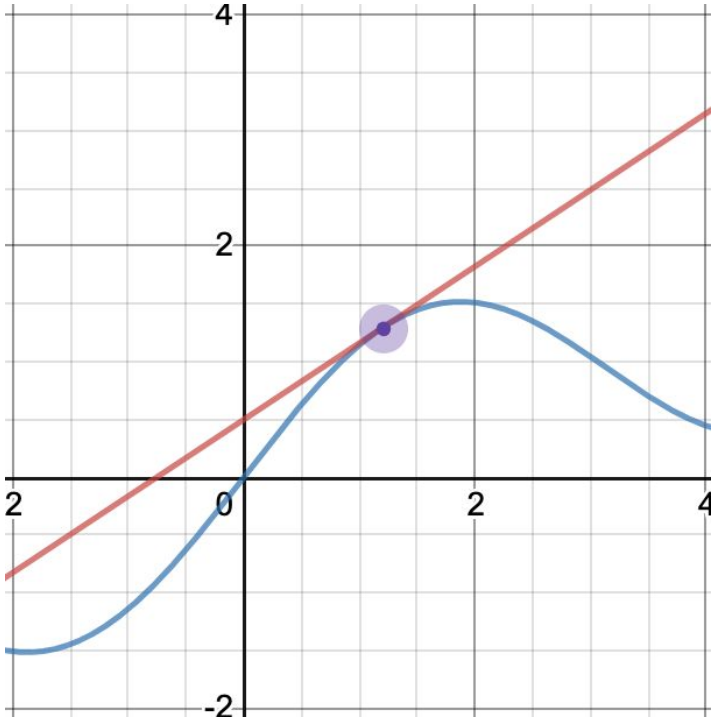
# So Far:

- Modeling pipeline involves picking a model, picking a loss function, and fitting model to loss.
- We fit model by taking derivative of loss, setting derivative equal to 0, then solving for parameters.
- Doesn't work for complicated models or loss functions!
    - E.g. Normal equations take too long to solve.
- Today: Learn **gradient descent**, a general technique for loss minimization.

# Gradients

# Derivatives

- The derivative of a univariate function gives the slope at a given point.
- We can also interpret the slope as:
  - If I nudge x, how does y change?
- [Desmos demo]



# Partial Derivative

- What about multivariable functions? E.g:

$$f_\theta(\boldsymbol{x}) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_p x_p$$

- Intuition: Pretend that multivariable function is univariate, then take derivative as normal. This is a **partial derivative**.

$$\frac{\partial}{\partial x_1} f_\theta(\boldsymbol{x}) = \theta_1 \qquad \frac{\partial}{\partial \theta_0} f_\theta(\boldsymbol{x}) = 1$$

- Desmos demo

# Gradients

- The gradient extends the derivative to multiple variables.
  - Vector-valued function (always outputs a vector).
  - The gradient of f(**θ**) w.r.t. **θ** is a vector. Each element is the partial derivative for component of **θ**.

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}) = \begin{bmatrix} \frac{\partial}{\partial \theta_0}(f) \\ \frac{\partial}{\partial \theta_1}(f) \\ \vdots \\ \frac{\partial}{\partial \theta_p}(f) \end{bmatrix}$$

# You Try:

Find the gradient of f(**θ**, **x**) w.r.t **θ**. Then find the gradient w.r.t **x**.

$$f(\boldsymbol{\theta}, \boldsymbol{x}) = \boldsymbol{\theta} \cdot \boldsymbol{x}$$

# You Try:

$$f(\boldsymbol{\theta}, \boldsymbol{x}) = \boldsymbol{\theta} \cdot \boldsymbol{x} = \theta_0 x_0 + \ldots + \theta_p x_p$$

$$\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}, \boldsymbol{x}) = \begin{bmatrix} \frac{\partial}{\partial \theta_0}(f) \\ \frac{\partial}{\partial \theta_1}(f) \\ \vdots \\ \frac{\partial}{\partial \theta_p}(f) \end{bmatrix} = \begin{bmatrix} x_0 \\ x_1 \\ \vdots \\ x_p \end{bmatrix} = \boldsymbol{x}$$

By a similar argument,

$$\nabla_{\boldsymbol{x}} f(\boldsymbol{\theta}, \boldsymbol{x}) = \boldsymbol{\theta}$$

Notice how the gradient looks like taking a normal derivative?

$$f(x) = ax$$

$$\frac{\partial}{\partial x} f(x) = a$$

This happens a lot! But you should always start with assembling a vector like we just did.

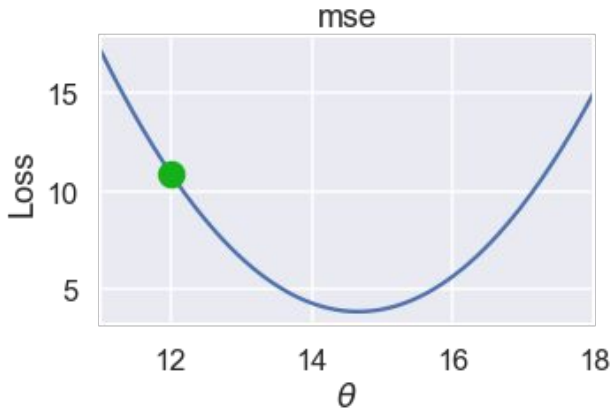
# How to Interpret Gradients

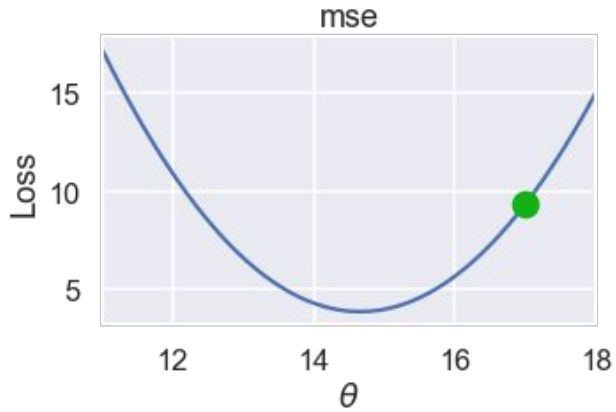
- Soon we take gradients of loss functions.
- You should read these gradients as:
  - If I nudge the 1st model weight, what happens to loss?
  - If I nudge the 2nd, what happens to loss?
  - Etc.
- Since the gradient generalizes the derivative, will say “gradient” from now on instead of “derivative”.

# Gradient Descent

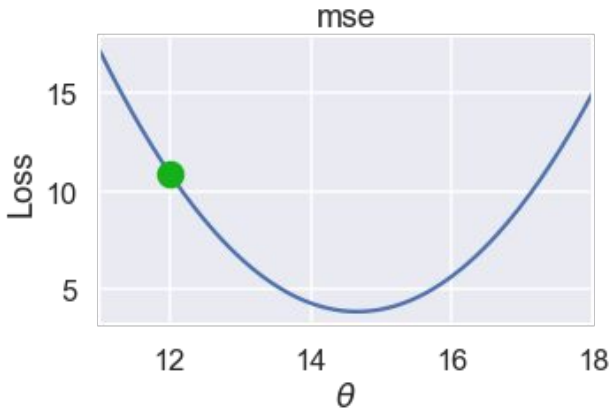
# Using the Gradient

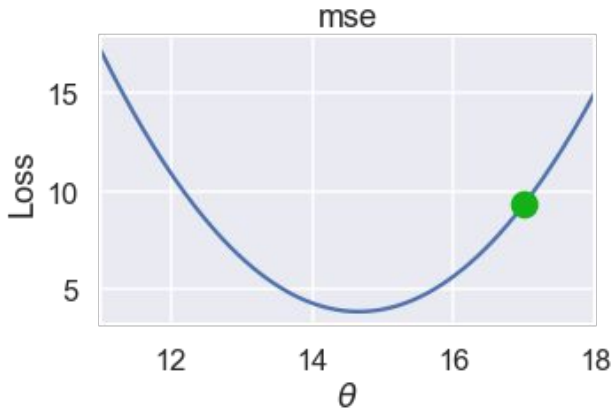
- Goal: Fit model by minimizing loss.
- Recall that loss is a function of model weights.
- So, gradient w.r.t **θ** encodes where **θ** should “move”.
  - Left: θ should move in the positive direction
  - Right: θ should move in negative direction



# Using the Gradient




- Left plot: gradient -, want θ +
- Right plot: gradient +, want θ -
- Idea: Move θ in negative direction of gradient.

# Batch Gradient Descent

- **Gradient descent** algorithm: nudge θ in negative gradient direction until θ converges.
- Batch gradient descent update rule:

Next value for θ

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}, \boldsymbol{X}, \boldsymbol{y})$$

Gradient of loss wrt θ

Learning rate

θ: Model weights  L: loss function

$\alpha$: Learning rate, usually small constant

y: True values from training data

# Gradient Descent Algorithm

- Initialize model weights to all zero
  - Also common: initialize using small random numbers
- Update model weights using update rule:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}, \boldsymbol{X}, \boldsymbol{y})$$

- Repeat until model weights don't change (convergence).
  - At this point, we have $\hat{\theta}$, our minimizing model weights

# Gradient Descent for MSE

- For constant model using MSE loss, we have:

$$L(\theta, \boldsymbol{x}) = \frac{1}{n}\sum(x_i - \theta)^2 \qquad \nabla_{\boldsymbol{\theta}} L(\theta, \boldsymbol{x}) = -\frac{2}{n}\sum(x_i - \theta)$$

- So the update rule is:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \left(-\frac{2}{n}\sum(x_i - \theta)\right)$$

**(demo)**

# The Learning Rate

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}, \mathbf{y})$$

- $\alpha$ called the **learning rate** of gradient descent.
  - Higher values can converge faster since θ changes more each iteration.
  - But $\alpha$ too high might cause GD to never converge!
  - $\alpha$ too low makes GD take many iterations to converge.

**(demo)**

# You Try:

Derive the gradient descent rule for a linear model with two model weights, no bias term, and MSE loss. Start by taking the gradient for a single data point.

$$f_\theta(\boldsymbol{x}) = \theta_1 x_1 + \theta_2 x_2 \qquad \ell(\boldsymbol{\theta}, \boldsymbol{x}, y_i) = (y_i - \theta_1 x_1 - \theta_2 x_2)^2$$

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \boldsymbol{x}, y_i) = ?$$

$\ell$ means loss for a single point;
L means average loss for dataset.

# You Try:

$$\ell(\boldsymbol{\theta}, \boldsymbol{x}, \boldsymbol{y}) = \frac{1}{n}\sum(y_i - \theta_1 x_1 - \theta_2 x_2)^2$$

$$\frac{\partial}{\partial \theta_1}\ell(\boldsymbol{\theta}, \boldsymbol{x}, y_i) = 2(y_i - \theta_1 x_1 - \theta_2 x_2)(-x_1)$$

$$\frac{\partial}{\partial \theta_2}\ell(\boldsymbol{\theta}, \boldsymbol{x}, y_i) = 2(y_i - \theta_1 x_1 - \theta_2 x_2)(-x_2)$$

$$\nabla_{\boldsymbol{\theta}}\ell(\boldsymbol{\theta}, \boldsymbol{x}, y_i) = -2\begin{bmatrix}(y_i - \theta_1 x_1 - \theta_2 x_2)(x_1) \\ (y_i - \theta_1 x_1 - \theta_2 x_2)(x_2)\end{bmatrix}$$

The gradient for the entire dataset is the average of the gradients for each point, so we can run GD as-is.

# In Matrix Form

There's usually a succinct matrix form for gradients. You should be able to convert back and forth.

$$\nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}, \boldsymbol{x}, y_i) = -2 \begin{bmatrix} (y_i - \theta_1 x_1 - \theta_2 x_2)(x_1) \\ (y_i - \theta_1 x_1 - \theta_2 x_2)(x_2) \end{bmatrix}$$

$$= -2(y_i - \boldsymbol{\theta} \cdot \boldsymbol{x})(\boldsymbol{x})$$

$$c \qquad c \qquad (p \times 1) = (p \times 1)$$

Once we have matrix form, we can use p dimensions (not just 2)

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}) = \frac{1}{n} \sum_{i}^{n} [-2(y_i - \boldsymbol{\theta} \cdot \boldsymbol{X_i})(\boldsymbol{X_i})]$$

$$= -\frac{2}{n}(\boldsymbol{X}^\top \boldsymbol{y} - \boldsymbol{X}^\top \boldsymbol{X} \boldsymbol{\theta})$$

Last line is bonus. Don't stress if you can't find it.

# Update Rule

$$\nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}, \boldsymbol{X}, \boldsymbol{y}) = -\frac{2}{n}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}^\top \boldsymbol{y})$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla_{\boldsymbol{\theta}} L$$

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \left(-\frac{2}{n}\right)(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}^\top \boldsymbol{y})$$

**(demo)**

Although we started off with only two variables, matrix notation extends our derivation to any number of variables.

# Break!
# Fill out Attendance:
# http://bit.ly/at-d100

# Stochastic Gradient Descent

# Stochastic Gradient Descent (SGD)

- Can perform update using one data point instead of all the data points.

Batch GD: $$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla_{\boldsymbol{\theta}} L(\boldsymbol{\theta}^{(t)}, \boldsymbol{X}, \boldsymbol{y})$$

**Stochastic GD**: $$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{(t)}, \boldsymbol{X_i}, y_i)$$

- SGD usually converges faster than BGD in practice!

# SGD Algorithm

1. Initialize model weights to zero.
2. Shuffle input data.
3. For each point in the shuffled input data, perform update:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \nabla_{\boldsymbol{\theta}} \ell(\boldsymbol{\theta}^{(t)}, \boldsymbol{X_i}, y_i)$$

4. Repeat steps 2 and 3 until convergence.

Each individual update is called an **iteration**.
Each full pass through the data is called an **epoch**.

# SGD Example

- BGD update rule for constant model, MSE loss:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot \left( -\frac{2}{n} \sum (x_i - \theta) \right)$$

- SGD update rule for constant model, MSE loss:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \alpha \cdot (-2)(x_i - \theta)$$

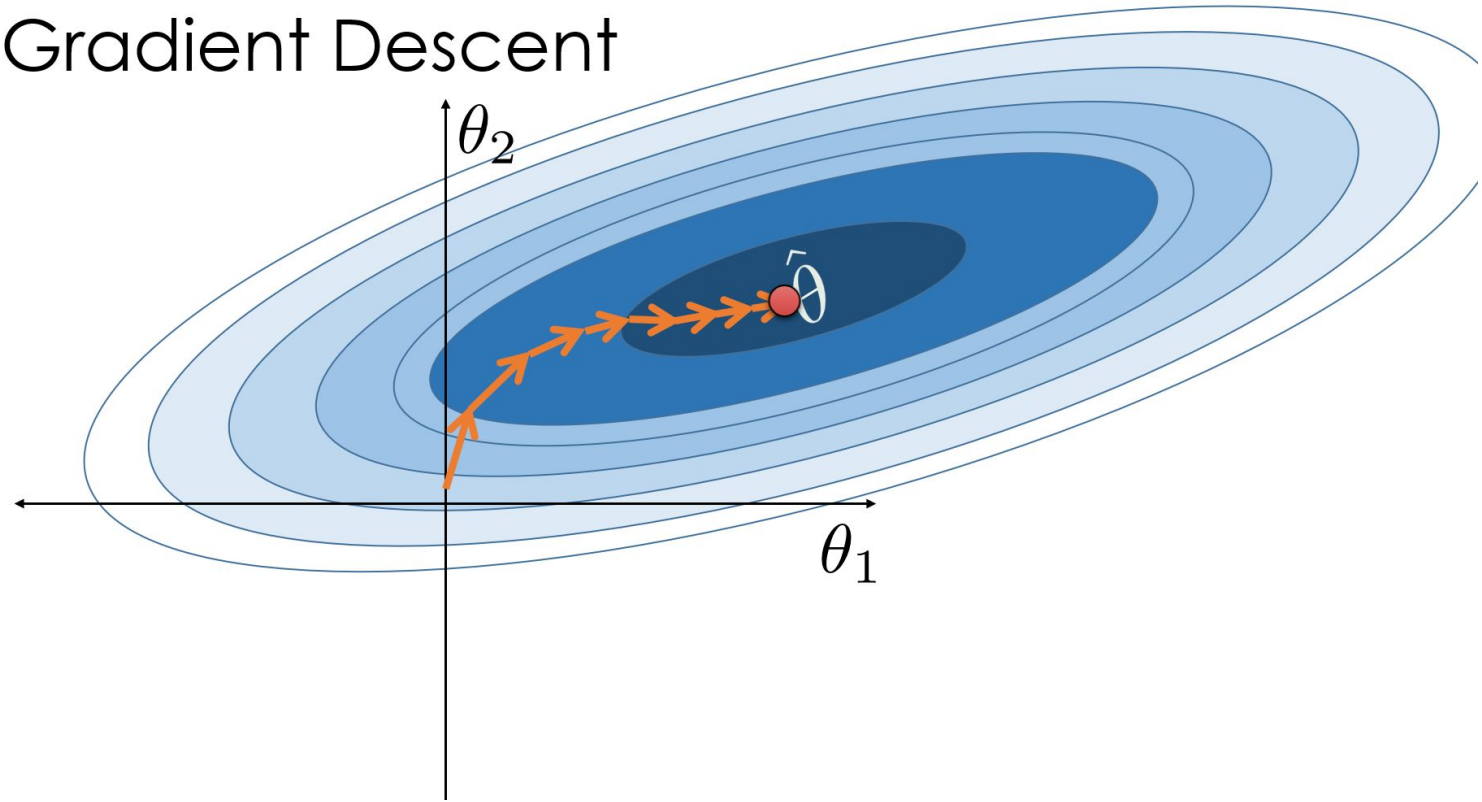
SGD update rules look like BGD update rules without taking the average.

**(demo)**

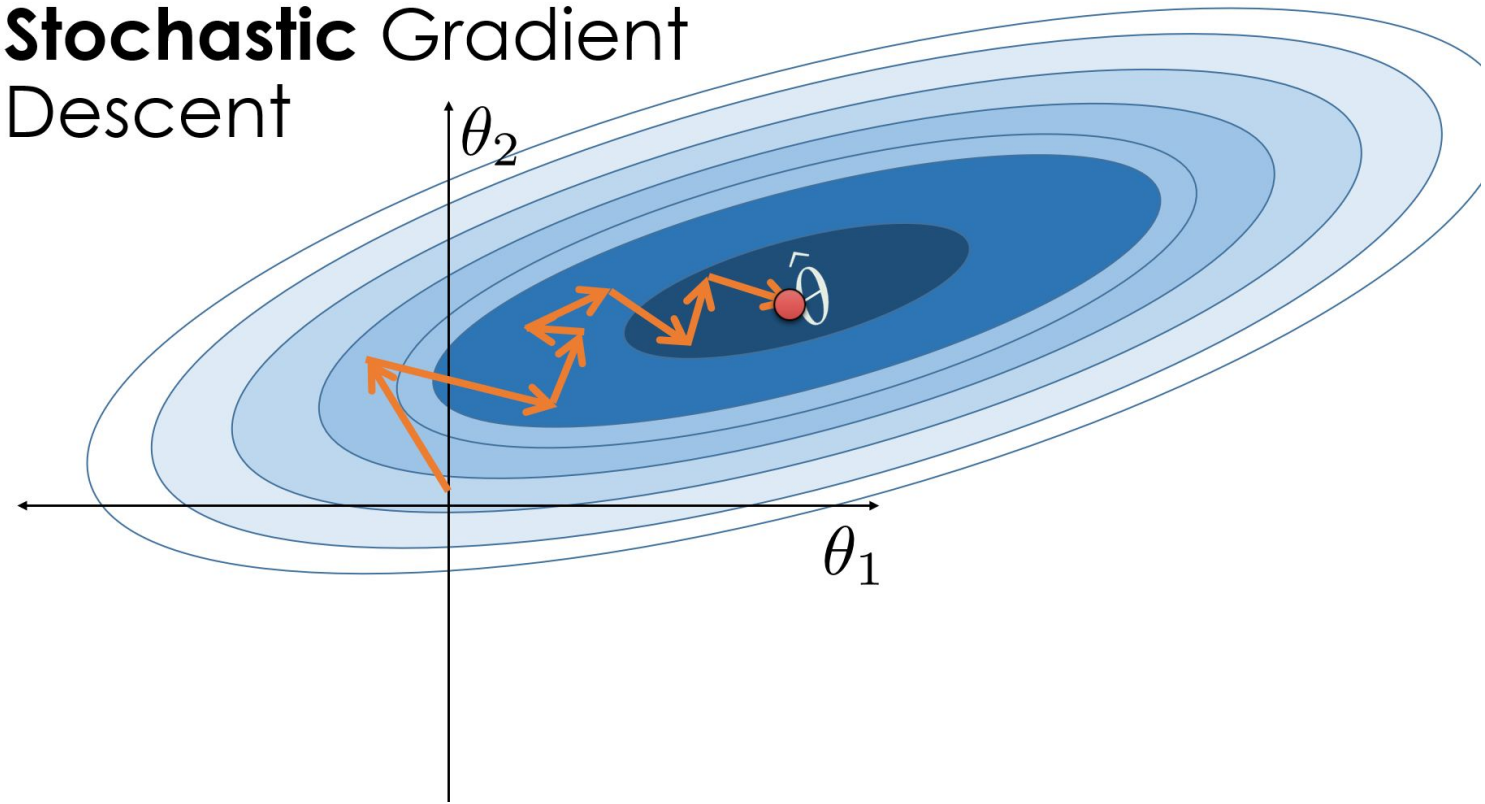
# Why does SGD perform well?

- Intuition: SGD “does more” with each data point.
  - E.g. Sample of size 1 million
  - BGD requires ≥1 million calculations for one update
  - SGD will have updated weights 1 million times!
- However:
  - BGD will always move towards optima
  - SGD will sometimes move **away** because it only considers one point at a time.
  - SGD still does generally better in spite of this.

# Batch GD always moves toward minimum:



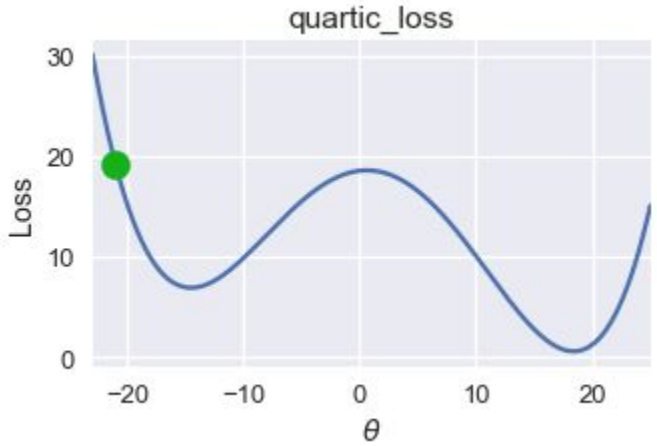
# Stochastic GD meanders



# Convexity

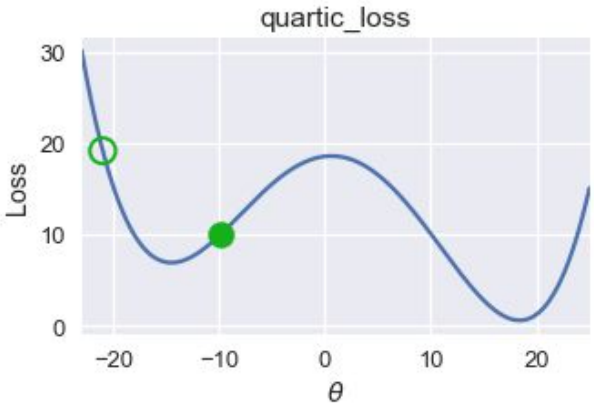
# Gradient Descent Only Finds Local Minima

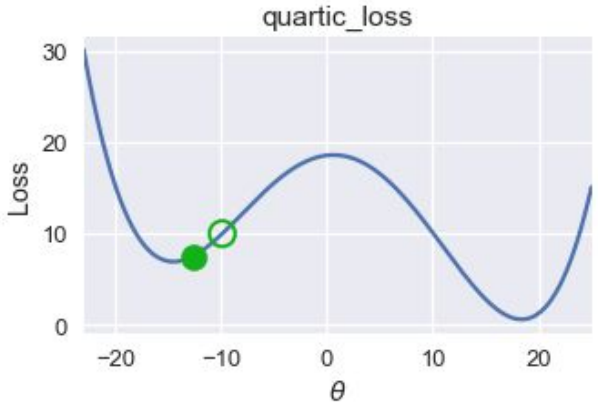
- If loss function has multiple local minima, GD is **not guaranteed** to find global minimum.
- Suppose we have this loss curve:

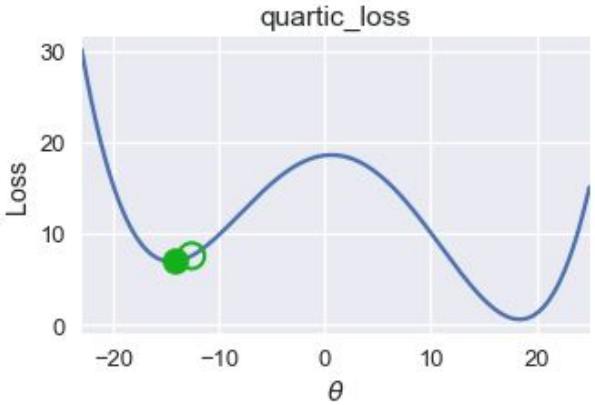


# Gradient Descent Only Finds Local Minima

- Here's how GD runs:




- GD can converge at -15 when global minimum is 18

# Convexity

- For a **convex** function f, any local minimum is also a global minimum.
  - If loss function convex, gradient descent will always find the globally optimal minimizer.
- Formally, f is convex iff:

$$tf(a) + (1-t)f(b) \geq f(ta + (1-t)b)$$

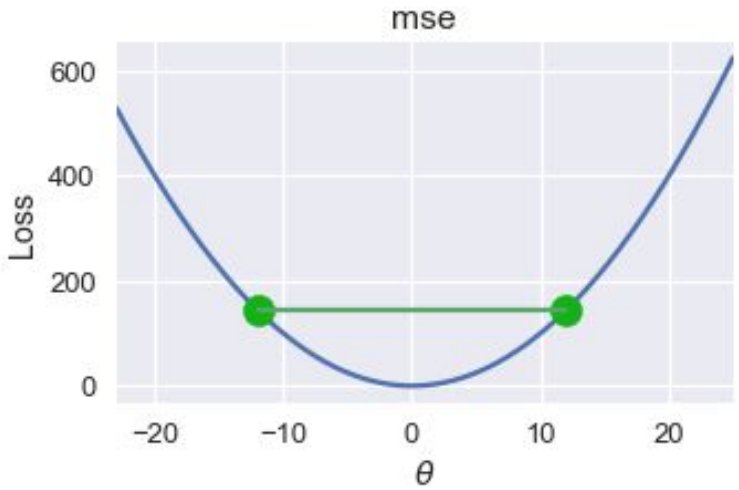
$$\text{For all } a, b \text{ in domain of } f \text{ and } t \in [0, 1]$$

# Convexity

$$tf(a) + (1-t)f(b) \geq f(ta + (1-t)b)$$

$$\text{For all } a, b \text{ in domain of } f \text{ and } t \in [0, 1]$$

- RTA: If I draw a line between two points on curve, all values on curve need to be on or below line.
- E.g. MSE loss is convex:

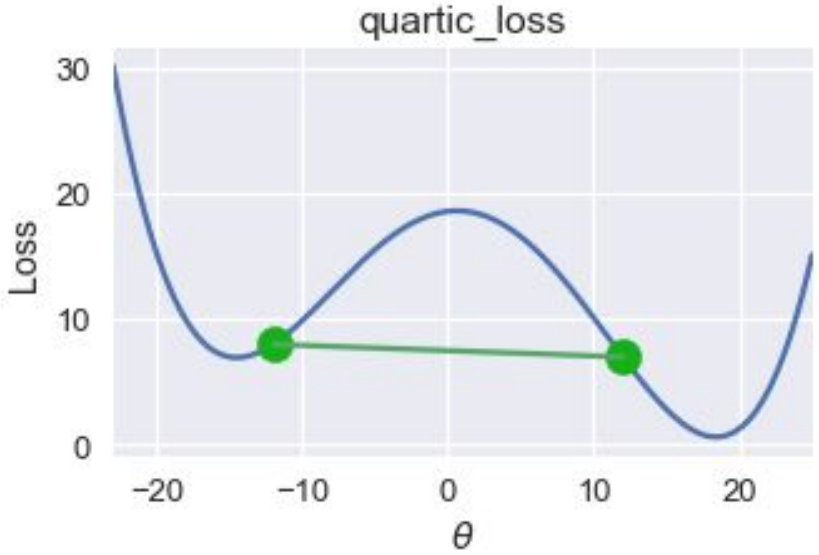


# Convexity

$$tf(a) + (1-t)f(b) \geq f(ta + (1-t)b)$$

$$\text{For all } a, b \text{ in domain of } f \text{ and } t \in [0, 1]$$

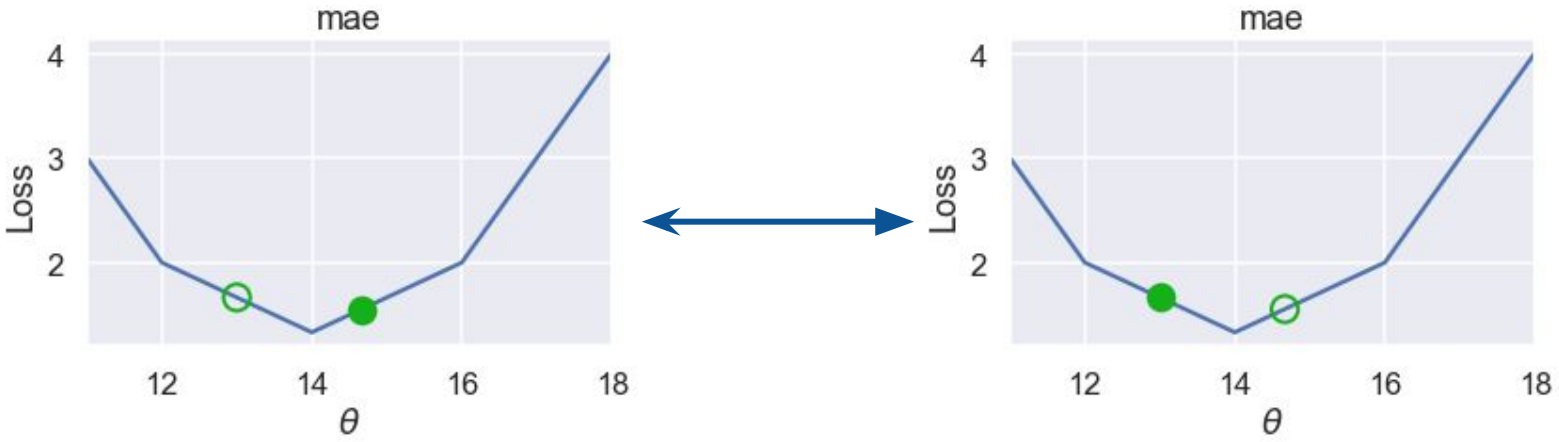
- But this loss function is not convex:



# Gradient Descent Considerations

# Choosing a Loss Function for GD

- Often we pick a loss function that “behaves well” for GD.
- MSE has larger gradients further away from minimum
  - So GD will make large steps at first, then small steps
- MAE is not differentiable at minimum.
  - So GD will seldom truly converge to minimum!



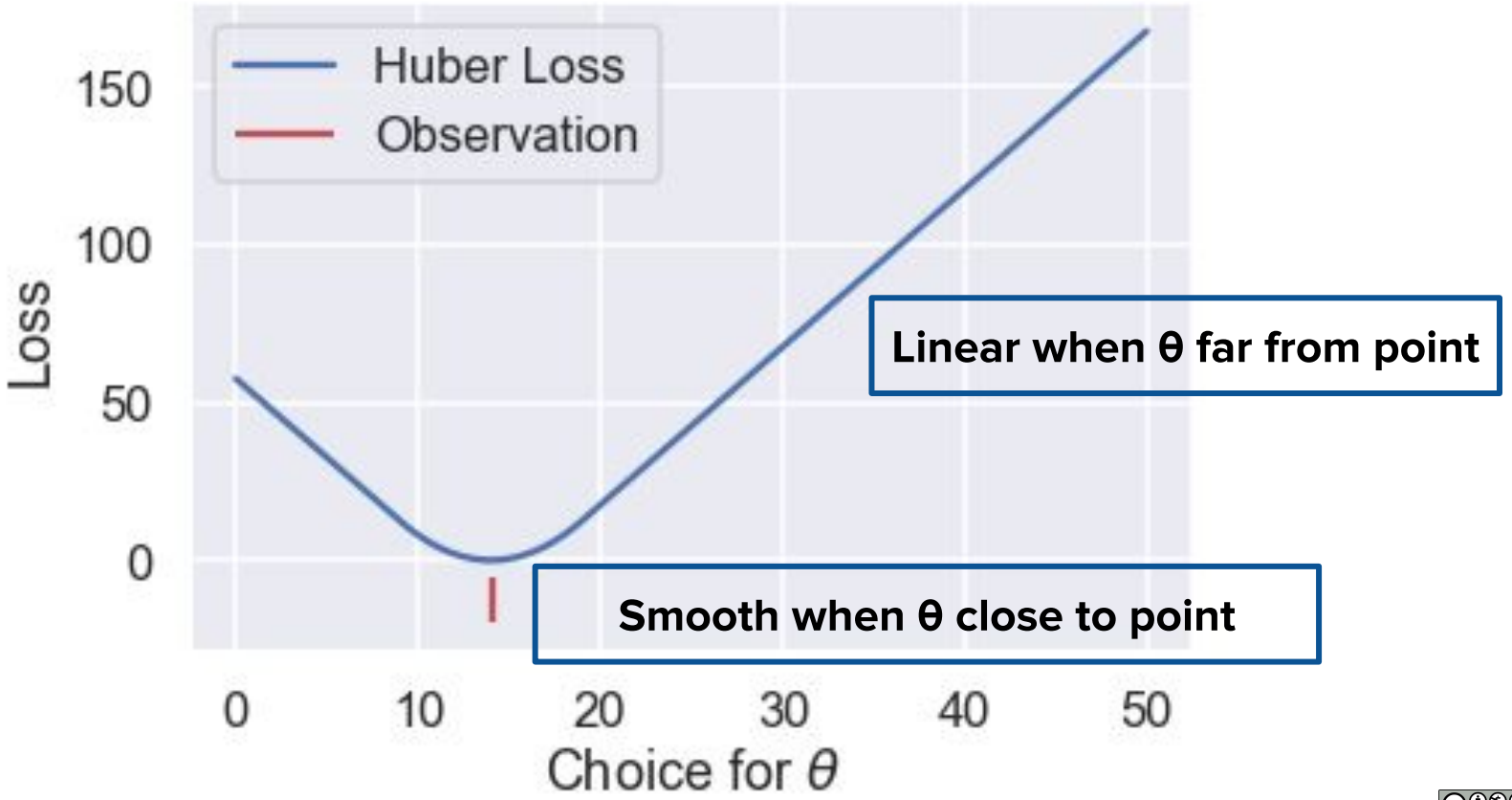
# Huber Loss

- What if we want the outlier-robust properties of MAE?
- Use **Huber Loss**:

$$L_\delta(\theta, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} \begin{cases} \frac{1}{2}(y_i - \theta)^2 & |y_i - \theta| \leq \delta \\ \delta(|y_i - \theta| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

- Behaves like MAE loss when points far from θ
  - But like MSE loss when points close to θ

# Huber Loss


Huber Loss
Observation
150
100
50
0
Loss
0
10
20
30
40
50
Choice for θ
Linear when θ far from point
Smooth when θ close to point

# Huber Loss

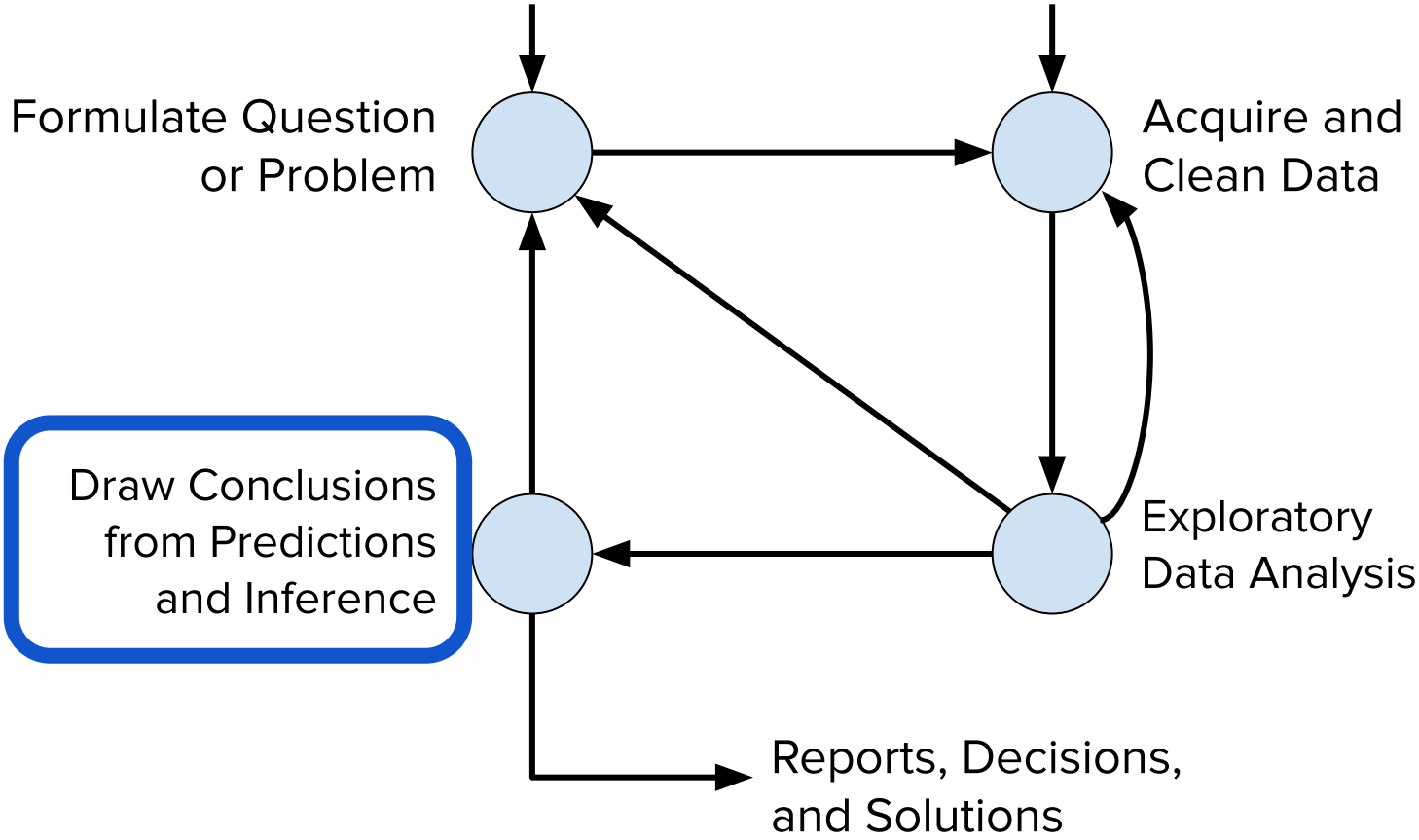
$$L_\delta(\theta, \mathbf{y}) = \frac{1}{n}\sum_{i=1}^{n} \begin{cases} \frac{1}{2}(y_i - \theta)^2 & |y_i - \theta| \leq \delta \\ \delta(|y_i - \theta| - \frac{1}{2}\delta) & \text{otherwise} \end{cases}$$

- $\delta$ is a parameter that determines where Huber loss transitions from MSE to MAE.
- No analytic solution (unlike MSE and MAE)
- But differentiable everywhere, so we can use GD!
- Try taking the derivative and running GD for practice.

# Why Use Gradient Descent?

- The beauty of GD is that it works for many types of models and loss function as long as you can take the gradient.
  - Usually, taking gradient is much easier than solving it!
  - Also, GD often finds solution faster than analytic solution even if there is one (e.g. linear regression).
- Modeling recipe:
  - Pick model
  - Pick loss function
  - Fit model by running gradient descent

# Remember the Data Science Lifecycle?



# Summary

- Gradients generalize derivatives.
- Gradient descent is a useful computational method for minimizing loss functions.
- Stochastic GD is used more often than batch GD because of its faster convergence.
- Convex loss functions behave well with GD methods.
- We now have all the pieces needed for modeling!

(demo)

$$= -\frac{2}{n}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}^\top \boldsymbol{y})$$

$$= -\frac{2}{n}(\boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{\theta} - \boldsymbol{X}^\top y)$$

Gradient of loss wrt θ