Gradient Descent

Raguvir Kunani

Data 100, Discussion 7

March 6, 2019

Last Time: Big Picture View of Modeling

What does modeling mean?

GD helps us do these steps

1. Identify a target variable that you want to predict
2. Gather some observational data about that target variable
3. Propose some relationship between the columns in your observational data and the target variable
4. Use ~ machine learning ~ to see how well that relationship actually predicts the target variable
5. Repeat steps 3 and 4 for multiple different relationships
6. Choose the relationship that best predicts the target variable

Clarification on Notation

1. Input data points are called x_i, and their corresponding output values are called y_i.

Example: How long will it take you to walk to class?

- x_i might represent 3 columns of data: how far the class is, how much time is left until the class starts, how long it took you last time
- y_i is how long it *actually* took you to walk to class

2. $L(\theta)$ is the *average* loss on the entire dataset. $l(\theta)$ is the loss on one point.

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

$$L(\theta) = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 = \frac{1}{n} \sum_{i=1}^{n} l_i(\theta)$$

Clarification on Parameters

Step 3 says to propose some relationship between the columns in your data and the target. This is called **proposing a model**.

Example: Walking to Class

parameters

$$\vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix}$$

how long to walk to class

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$

data

- θ_i are the **parameters** of this model (sometimes θ_i is called w_i).
- We often collect the parameters into a vector $\vec{\theta}$ (or $\vec{w}$)

This model is *parametric* and *linear*. We will only study these models.

For simplicity, we write the above equation as $\hat{y} = f_\theta(\vec{x})$.

We have a model, but is the model good?

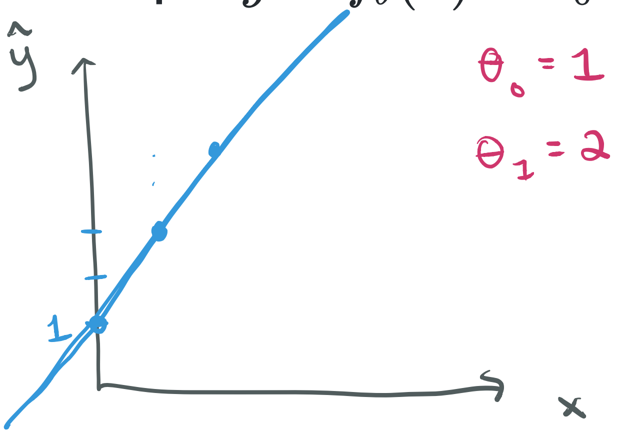
We have the loss function which tells us how *bad* our model is, but we can't use it just yet. Why?

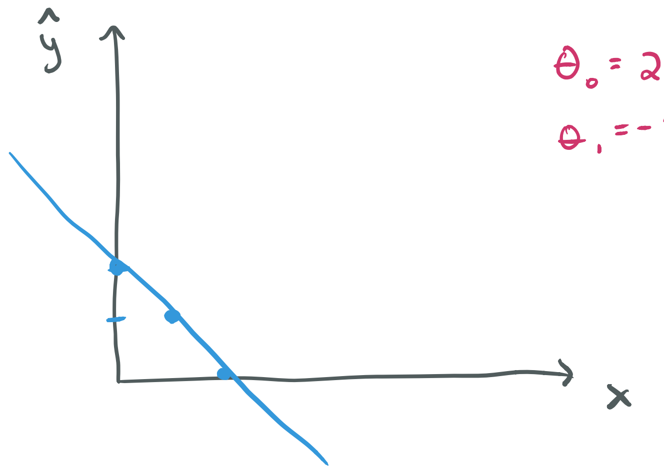
> Hint: What are the knowns and unknowns in $\hat{y} = f_{\theta}(\vec{x})$?

unknown!

We need to choose a value for $\vec{\theta}$! This will fully define our model.

Example: $\hat{y} = f_{\theta}(\vec{x}) = \theta_0 + \theta_1 x$





How to Choose a Value for $\vec{\theta}$?

Whichever $\vec{\theta}$ minimizes loss seems natural since we want to find a model that gives accurate predictions (i.e. small loss).

But how do we minimize the loss? There are 2 ways:

1. Take the gradient of the loss, set it to 0, and solve for $\hat{\theta}$.
2. Use an algorithm that can find minimum values of functions.

Second method is gradient descent!

Relating back to the steps of modeling, we're now entering Step 4: see how well the proposed actually predicts the target variable.

Gradient Descent Algorithm (One Dimension)

our choice of θ at time $t=0$

1. $\theta^{(0)}$ = [any choice will work for $\theta^{(0)}$ if the loss function is convex]

$L = \frac{1}{n}\sum_{i=1}^{n}(y - f_\theta(x))^2$

2. for $t = 0$ until you reach the minimum:
   a) Find the derivative of the loss, denoted $\frac{dL}{d\theta}$
   b) Evaluate the derivative at $\theta = \theta^{(t)}$, denoted $\left.\frac{dL}{d\theta}\right|_{\theta=\theta^{(t)}}$
   c) Update: $\theta^{(t+1)} = \theta^{(t)} - \alpha \left.\frac{dL}{d\theta}\right|_{\theta=\theta^{(t)}}$
3. Return $\theta^{(T)}$, the final θ after the for loop is done

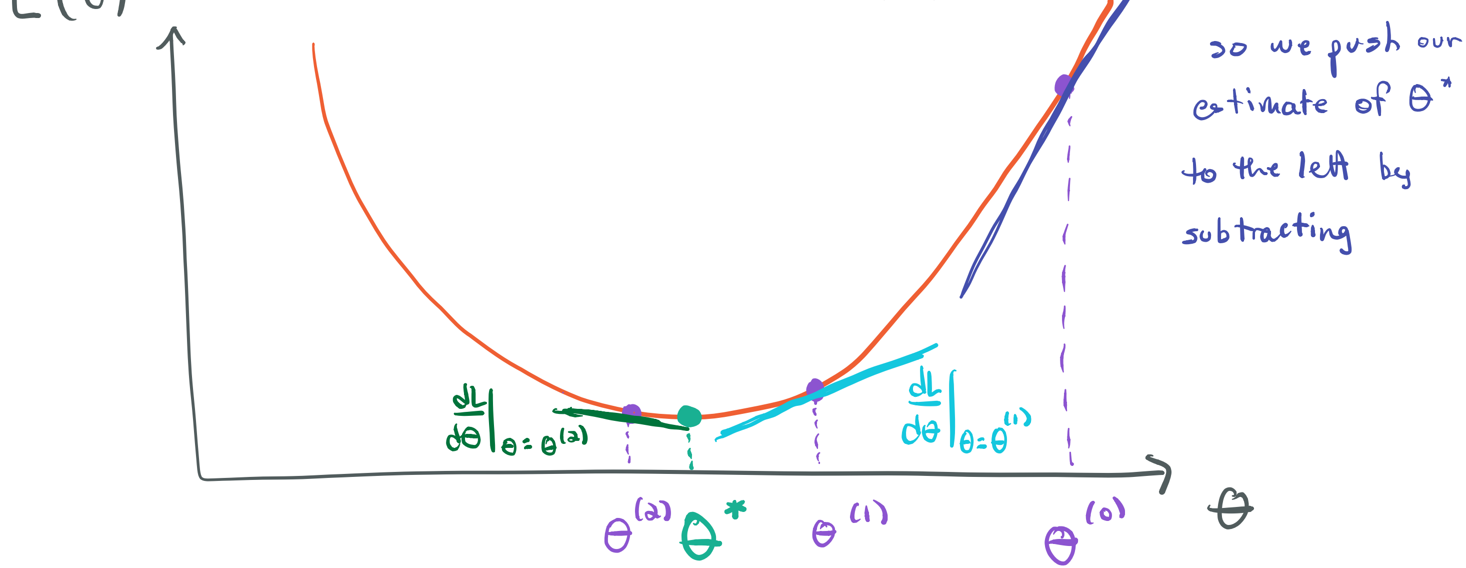
$\theta^{(T)}$ is the θ that minimizes the loss $L(\theta)$!

1D Gradient Descent Example

$\nearrow$ 1 parameter $\qquad\qquad\qquad\qquad \nearrow$ squared loss

Our model is $\hat{y} = \theta x$ and loss is $L(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \theta x_i)^2$.

$$\theta^{(t+1)} = \theta^{(t)} - \alpha \frac{dL}{d\theta}\bigg|_{\theta=\theta^{(t)}}$$



Conceptually, gradient descent is what you would do if you had to get to the bottom of a hill but you can only see a small distance around you.

$f(\vec{x}) = f(x_1, \cdots, x_n)$

Gradients: Derivatives in Multiple Dimensions

The **gradient** of a function f that takes a vector $\vec{x}$ as input is $\nabla_{\vec{x}} f(\vec{x})$.

$$\vec{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

How to Find the Gradient of a Function:

1. For every element x_i of the input $\vec{x}$ to f:

   a. Find the *partial derivative* with respect to x_i, denoted $\frac{\partial f}{\partial x_i}$.

2. Collect all the $\frac{\partial f}{\partial x_i}$ into a vector:

This vector is the gradient of $f(\vec{x})$: $\nabla_{\vec{x}} f(\vec{x})$.

Example: Worksheet Problem 1 and 2

Gradient Descent Algorithm (Multiple Dimensions)

1. $\vec{\theta}^{(0)}$ = [any choice will work for $\vec{\theta}^{(0)}$ if the loss function is convex]
2. for $t = 0$ until you reach the minimum:
   a) Find the gradient of the loss (denoted $\nabla L(\vec{\theta})$)
   b) Evaluate the gradient at $\vec{\theta} = \vec{\theta}^{(t)}$ (denoted $\nabla L(\vec{\theta})\Big|_{\vec{\theta}=\vec{\theta}^{(t)}}$
   c) Update: $\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} + \alpha \nabla L(\vec{\theta})\Big|_{\vec{\theta}=\vec{\theta}^{(t)}}$
3. Return $\vec{\theta}^{(T)}$, the final $\vec{\theta}$ after the for loop is done

$\vec{\theta}^{(T)}$ is the $\vec{\theta}$ that minimizes the loss $L(\vec{\theta})$!

Nothing changed except I put some $\rightarrow$ symbols on top of the θ!

Example: Worksheet Problem 3B

Learning Rates

Do Problem 3A

There are 2 types of learning rates:

1. **Constant learning rates**: simple, but prone to overshooting near the minimum
2. **Decaying learning rates**: learning rate is a decreasing function of t

Challenge Question: Why would we want decaying learning rates?

Each update we make to θ gets us closer to θ^*, but there is the danger that a step size too big will make us overshoot θ^*. Thus, we want a learning rate that is smaller as we get closer to θ^*. This is exactly what a decaying learning rate does.

Closer Look at the Gradient Descent Update Rule

We ideally stop updating $\vec{\theta}$ when the gradient is 0, but in practice this does not always occur. To fix this, we can either:

1. choose a number of times to update (epochs)
2. stop updating when $\vec{\theta}^{(t+1)}$ is really close to $\vec{\theta}^{(t)}$

Both of these fixes correspond to changing the for loop condition.

1. for $t=0$ to $t=T$:

   ⋮

2. while $|\vec{\theta}^{(t+1)} - \vec{\theta}^{(t)}| \geq 0.1$:

   ⋮

Stochastic Gradient Descent

Computing $\nabla L(\vec{\theta})$ requires computing n different gradients (1 for each data point) and averaging them. To see why, note: $\nabla L(\vec{\theta}) = \nabla \left[\frac{1}{n}\sum_{i=1}^{n} l(\vec{\theta}, \vec{x_i})\right]$ by definition of $L(\vec{\theta})$.

Instead of computing $\nabla L(\vec{\theta})$, we can instead **randomly** choose a point $\vec{x_i}$ and compute $\nabla l(\vec{\theta}, \vec{x_i})$. Then the gradient descent update rule is:

$$\vec{\theta}^{(t+1)} = \vec{\theta}^{(t)} + \alpha \nabla l(\vec{\theta}, \vec{x_i}))\Big|_{\vec{\theta}=\vec{\theta}^{(t)}}$$

Anonymous Feedback Form

tinyurl.com/raguvirTAfeedback