Netflix demo

- Have 2 people set up new Netflix profiles - they select 3 movies they like, and it recommends new ones. Notice how it tries to make predictions on what you'll like, given the data you've inputted!
- Problem: we have information on some movies that people like, but obviously it's incomplete since most people haven't watched every movie that exists :)
- Intuition: if someone else has similar interests to you, then you might like watching other videos that they enjoy.
- If many people enjoy the same 2 movies, maybe they're related! So if you liked one of them, you might be likely to like the 2nd
- This requires learning your preferences from data! (We'll talk more about this later in the course.)

Review

- Overall, in this class we're going to learn about **techniques that allow computers to learn from data**. Basically, the idea is that once you feed the computer lots of data, it will automatically learn the patterns in the data, and use them to make well-informed and accurate predictions.
- This first week, we've talked about **regression**.
    - For each example: given a bunch of data, predict some outcome.
    - For example, given information about a house (area, does it have a garage, year built, number of bedrooms, number of bathrooms, size, etc), predict its price
        - *Draw table*
        - To learn from, some houses already have their price (training set).
        - But we want to predict the price for new houses for which we don't have that info.
- We've only discussed **linear regression** so far. This assumes that the output we're trying to predict is approximately a **linear function** of the input variables.
    - One of the simplest ML algorithms, but it demonstrates some really important concepts!
    - Also, it's already very useful in practice. Examples:
        - House price example
        - Epidemiology: predicting disease risk from certain factors. For example, linear regression was used to demonstrate that smoking led to higher mortality rates back in the 60's.
        - Economics: estimate the systematic risk of an investment, or predict GDP growth
    - Let's think about the simplest case, where we only have one piece of information **x1** (size of the house, in square feet) and we want to predict the house price. We assume that the house price **y** (what we're trying to predict) can be predicted by **y_hat = w0 + w1*x1**
    - In machine Learning, y is usually the output that we're trying to predict, y_hat is our prediction, x_i is the input features (data we have), and w_i are weights that are learned.

- But what are w0 and w1? **We learn them from data!** That's what makes it a machine learning algorithm!
- In particular, we want to find the value of w0 and w1 that **minimizes the error between our predicted price and the true price.** Let's write this down as a **loss function**. We want to find the values of w0 and w1 that minimize this function:
    - sum_(i=1)^n (predicted price - true price)^2
    - **For each house**, we square the error (distance) between the predicted and true prices. Then add all the errors up.

Convex/Concave Functions and Gradient Descent

- Great, so how in the world do we minimize this function?
- Entire field of mathematical optimization is dedicated to this - we're just going to scratch the surface :)
- First, let's introduce the idea of **convex and concave functions**. Convex and concave functions are useful since they're relatively easy to optimize. So what are they?
    - For a convex function, if you draw a straight line between any two points on the function, the line is always **higher than the function value**.
    - This means that there is only one minimum, which is the **only** point where the derivative is 0. (If there are other points where the derivative is 0, then it's not convex. For this class you don't need to prove this.)
    - Concave: similar (the line is always lower than the function)
    - *Show example of neither*
- Note: we're going to focus on finding the **minimum** of the function. (If you wanted to find the maximum, you can find the minimum of the negative of the function.)
- Let's suppose the function is convex - how do we do this?
    1. The ideal way to find the minimum is to compute the derivative of the function (how fast the function value changes in response to small changes in w), and set it to 0. Find "w" where the derivative is 0.
        a. Example: g(w) = 5 - (w-10)^2 (see annotated slides)
        b. Only works when function is convex or concave. If it's not, there could be multiple extreme points where the derivative is 0, or none.
        c. Also, it's not always possible or efficient to solve for this
    2. Let's say we can't find the minimum analytically, but we can still take the derivative of the function. How might we get to the minimum?
        - Just keep going downhill! In the 1-D case: if the function is increasing to the right, we take a step to the left. If the function is decreasing to the right, we take a step to the right.
        - Exactly the idea behind gradient descent. The gradient represents the **direction** that is the steepest downhill.
            - *Write Gradient Descent equation: w <- w - eta * (dg(w)/dw)*
        - How far do we move? It's sometimes hard to tell! If we move too far, we run the risk of overshooting the minimum and going too far. But if we don't move enough, it could take way too many steps to actually get to the minimum.

        - (Draw example of too-large step size)
        - In practice, you may need to try various step sizes to see which one works. If it's going slowly, try increasing it. But if the function value isn't improving, you may have overshot the minimum, so try reducing it.
        - Sometimes it helps to decrease step size as we get closer to the minimum, to make sure we don't miss it.
            - *Common choices: eta_t = alpha / t, alpha / sqrt(t)*
        - **When do we stop?**
            - For a convex function the derivative should be exactly 0 at the minimum
            - But it's hard to land exactly there, so usually we stop when we're close enough: the magnitude of the gradient is below some threshold (epsilon)
- What if function isn't convex?
    - Gradient descent will find a **local** minimum but it's not guaranteed to find the global minimum
    - Trick: try multiple runs starting from different places, or try taking a large step and see if it lets you escape from the local minimum.
    - In neural networks, the loss function is **not** convex. However, people still use gradient descent and it often works in practice.
    - Luckily in linear regression, the loss function is convex.

Higher dimensions

- Let's say instead of a single weight, we have multiple weights/parameters we're trying to optimize over.
- Gradient: a **vector** containing partial derivatives with respect to each variable
    - *Do example*
- Points in the direction of steepest ascent. We take the negative of the gradient to find the steepest downhill direction.