Week 4

August 18, 2021

Linear regression with multiple features

- Multiple variables = multiple features.
- x_1 - size, x_2 - number of bedrooms, x_3 - number of floors, x_4 - age of home.
- n - number of features (n = 4).
- m - number of examples (i.e. number of rows in a table).
- x^i - vector of the input (in our example a vector of the four parameters for the i^{th} input example), where i is the training set's index.
- x_j^i the value of j^{th} feature in the i^{th} training set. For example x_2^3 represents the number of bedrooms in the third house.

Previously, our hypothesis had the following form:

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

Now we have multiple features:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3 + +\theta_4 x_4$$

For convenience of notation, we can introduce $x_0 = 1$. As a result, your feature vector is now n + 1 dimensional and indexed from 0.

$$h_\theta(x) = \theta^T X$$

Gradient descent for multiple variables

$$J(\theta_0, \theta_1, ..., \theta_n) = \frac{1}{2m} \sum_{m}^{i=1} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Algorithm 1 Gradient Descent

1: **while** not converged **do**

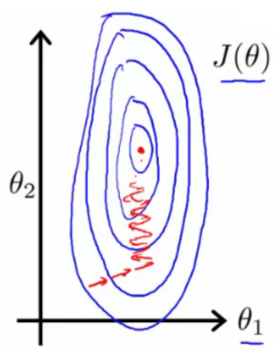
2: $\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, ..., \theta_n)$

3: $(for\ j = 0, ..., n)$

- For each θ_j (0 until n) we make an simultaneous update.
- θ_j is now equal to it's previous value subtracted by learning rate (α) times the partial derivative of of the θ vector with respect to θ_j.

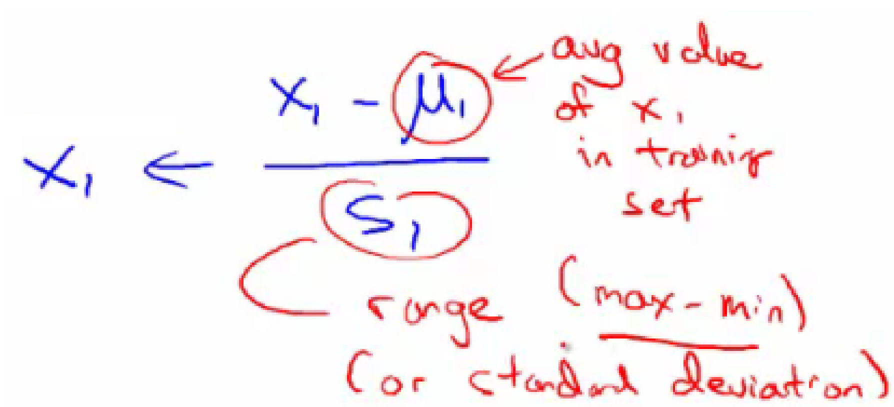
Gradient Decent in practice: Feature Scaling

- If you have a problem with multiple features, make sure they all have a comparable scale.
- For example, x1 = size (0 - 2000 feet) and x2 = number of bedrooms (1-5). Means the contours generated if we plot θ_1 vs. θ_2 give a very tall and thin shape due to the huge range difference.
- Finding the global minimum using gradient descent on this type of cost function might take a long time.



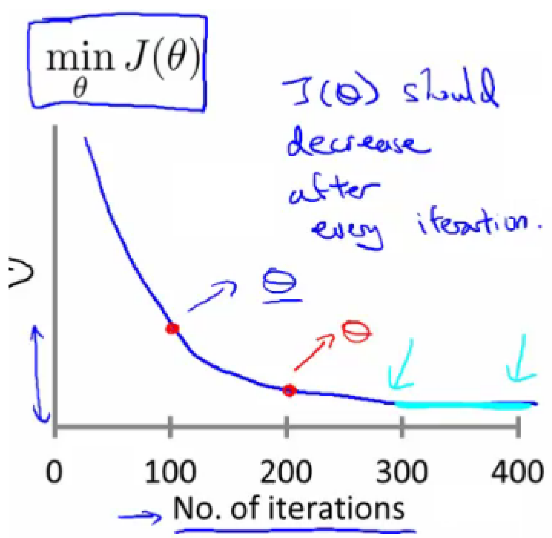
Mean normalization

- Take a feature x_i.
- Replace it by $\frac{x_i - mean}{max}$.
- So your values all have an average of about 0.



Learning Rate α

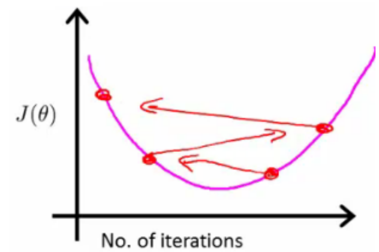
- Plot the $minJ(\theta)$ vs. the number of iterations (i.e. plot $J(\theta)$ throughout the course of gradient descent).
- $J(\theta)$ should decrease after each iteration if gradient descent is operating.
- Can also show if you're not making significant progress after a specific amount of days.
- If necessary, heuristics can be used to decrease the number of iterations.
- If, after 1000 iterations, $J(\theta)$ stops decreasing, you may choose to only conduct 1000 iterations in the future.
- DON'T hard-code thresholds like these in and then forget why they're there!



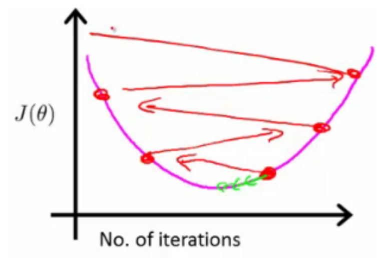
Automatic convergence tests

- Check if $J(\theta)$ changes by a small threshold or less.
- Choosing this threshold is hard.
- It's easier to check for a straight line.
- Why? - Because we're seeing the straightness in the context of the whole algorithm.

If you plot $J(\theta)$ vs iterations and see the value is increasing - means you probably need a smaller α.



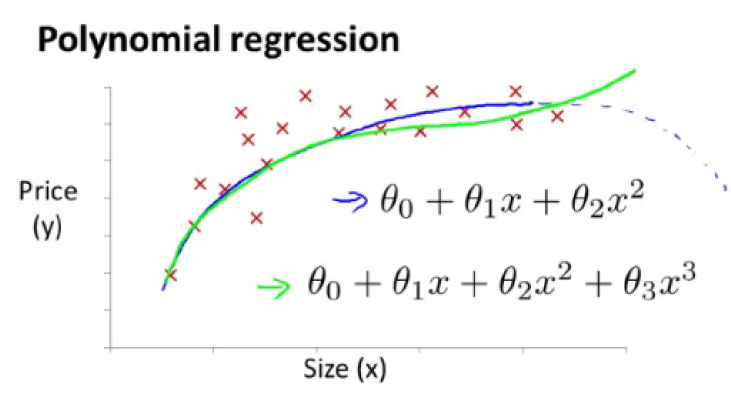
However, you might overshoot, therefore lower your learning rate so that you can reach the minimum (green line).



BUT, if α is too small then rate is too slow.

Features and polynomial regression

- May fit the data better.
- Overfitting vs underfitting.



Normal equation

- The normal equation is a superior approach for some linear regression problems.
- We've been using a gradient descent - iterative method that takes steps to converge.
- Normal equation gives us θ analytically.

How does it work?

- Here θ is an n+1 dimensional vector of real numbers.
- Cost function is a function that takes that vector as an argument.
- How do we minimize this function?
- Take the partial derivative of $J(\theta)$ with respect θ_j and set to 0 for every j. Solve for θ_0 to θ_n.

Example

Size (feet^2)	Number of bedrooms	Number of floors	Age of home (years)	Price ($1000)
x_1	x_2	x_3	x_4	y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178

Steps

- $m = 4$, $n = 4$.
- Add an extra column (x_0 feature).
- Construct a matrix (X - the design matrix) which contains all the training data features in an $[m \times n + 1]$ matrix.
- Construct a column vector y vector $[mx1]$ matrix.
- Use the following equation for θ:

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

$$\left(\begin{bmatrix} 1 & 1 & 1 & 1 \\ 2104 & 1416 & 1534 & 852 \\ 5 & 3 & 3 & 2 \\ 1 & 2 & 2 & 1 \\ 45 & 40 & 30 & 36 \end{bmatrix} \text{X} \begin{bmatrix} 1 & 2104 & 5 & 1 & 45 \\ 1 & 1416 & 3 & 2 & 40 \\ 1 & 1534 & 3 & 2 & 30 \\ 1 & 852 & 2 & 1 & 36 \end{bmatrix} \right)^{-1} \text{X} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2104 & 1416 & 1534 & 852 \\ 5 & 3 & 3 & 2 \\ 1 & 2 & 2 & 1 \\ 45 & 40 & 30 & 36 \end{bmatrix} \text{X} \begin{bmatrix} 460 \\ 232 \\ 315 \\ 178 \end{bmatrix}$$

If you compute this, you get the value of theta which minimize the cost function.

Gradient descent vs normal equation

Gradient Descent	Normal Equation
Need to choose learning rate	No need to choose a learning rate
Needs many iterations - could make it slower	No need to iterate, check for convergence etc.
Works well even when n is massive (millions)	Slow of n is large