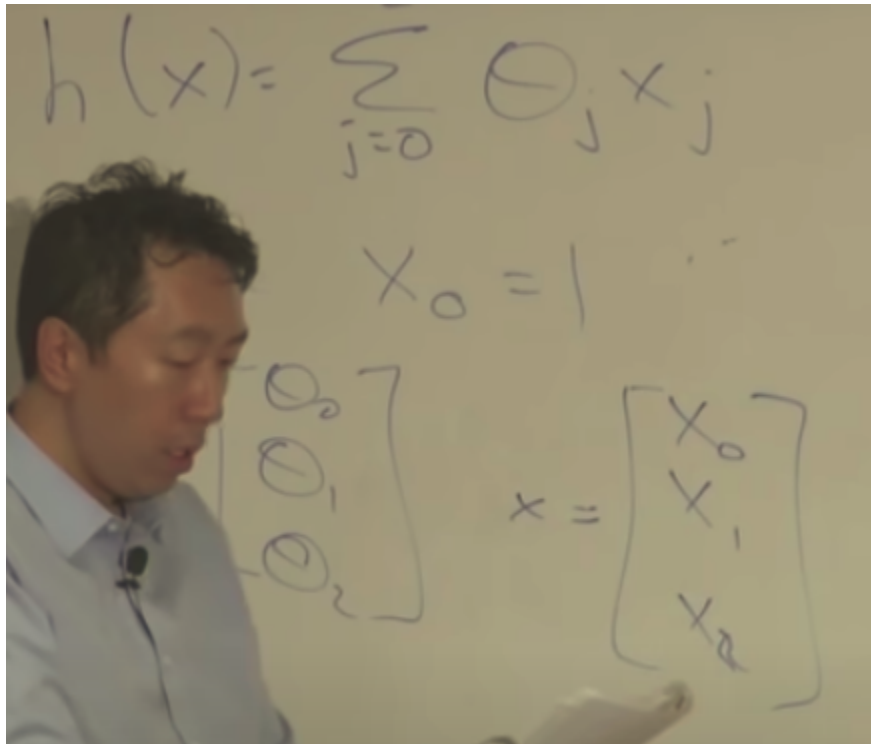
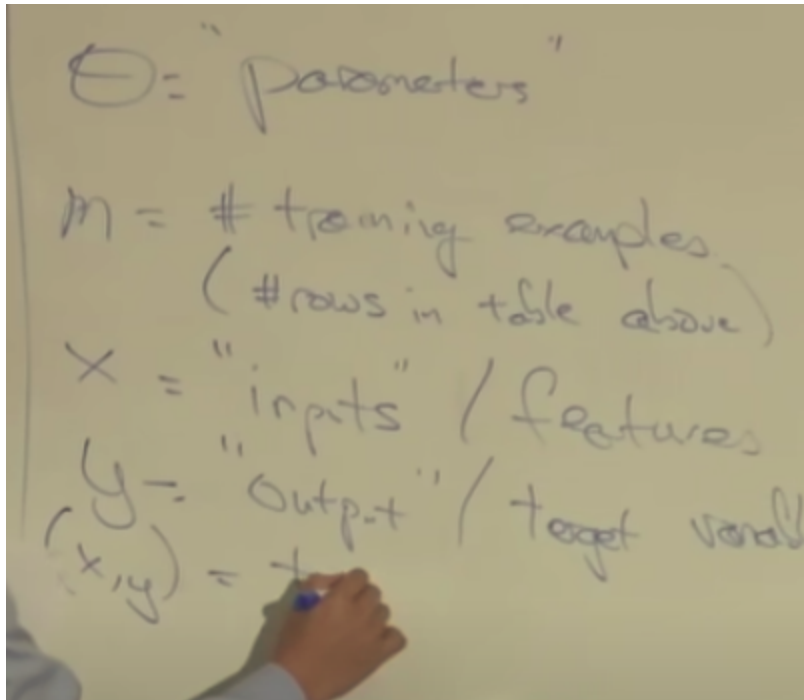


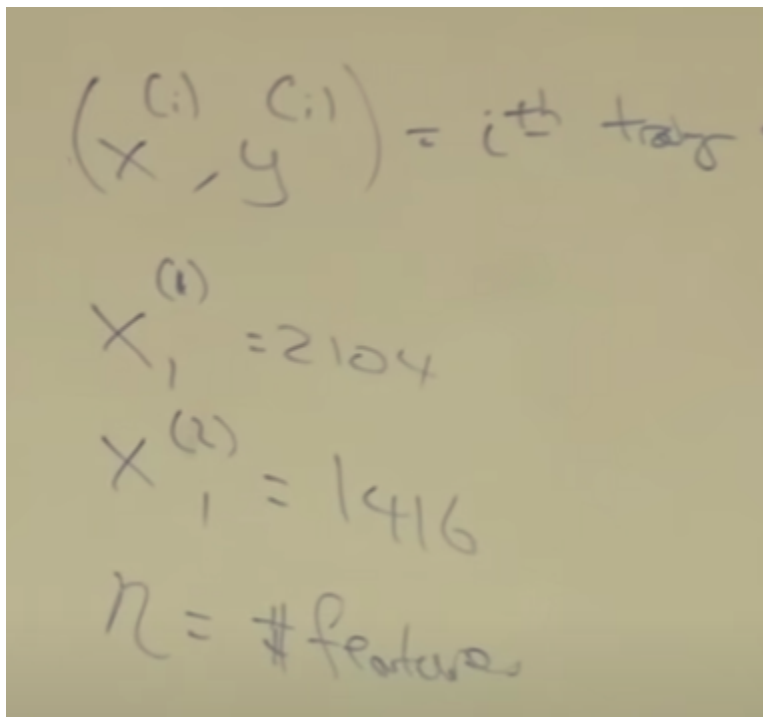
Linear Regression and Gradient Descent - Andrew Eng



- x_0 is defined as a dummy feature that always equals 1. This allows us to write the hypothesis.
- θ becomes a three dimensional parameter. Features also become a three dimensional vector where x_0 is always one, x_1 is the size of the house, and x_2 is the number of bedrooms for the house.
- θ is called the parameters of the learning algorithm. The job of the learning algorithm is to choose the right parameters (θ), that allows you to make good predictions about your prices of houses.



- (X, Y) - training example



- The superscript i represents the index that spans through the number of training examples.

$N = \text{\# of features}$ which in this case equals 2 with one dummy variable.

HOW DO YOU CHOOSE PARAMETERS

- The learning algorithm's job is to choose values for the parameters θ so it can output hypotheses.
- Choose θ so that $h(x)$ is close to y for training examples.

Choose θ st. $h(x) \approx y$ for training examples

- $H(X)$ has been written as a function for the features of the house. (size, # of bedrooms).

$$h_{\theta}(x) = h(x)$$

- This is equivalent to h subscript x letting us know that the parameters are still prominent.
(abbreviation in notation)

- We want to minimize the squared difference between what the hypothesis outputs

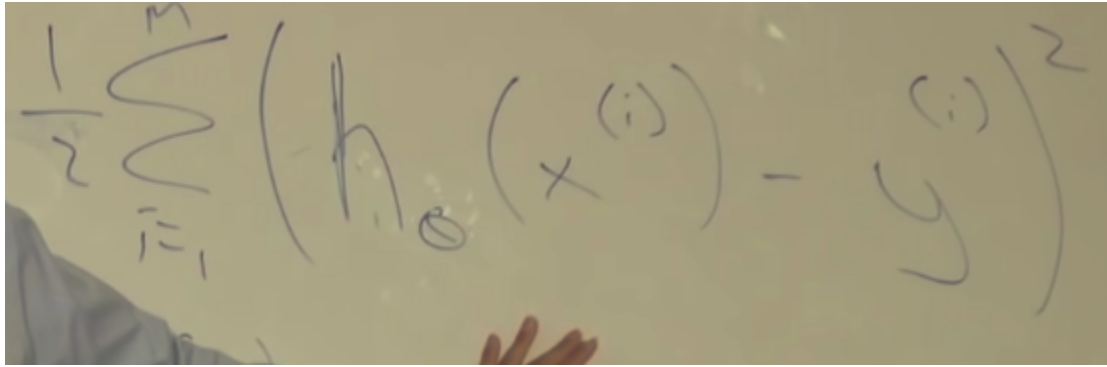
Hypothesis house price output : $h_{\theta}(x)$

Correct price : y

$$(h_{\theta}(x) - y)^2$$

- We want to choose values of θ that minimize the squared difference.

OVERVIEW OF COST FUNCTION

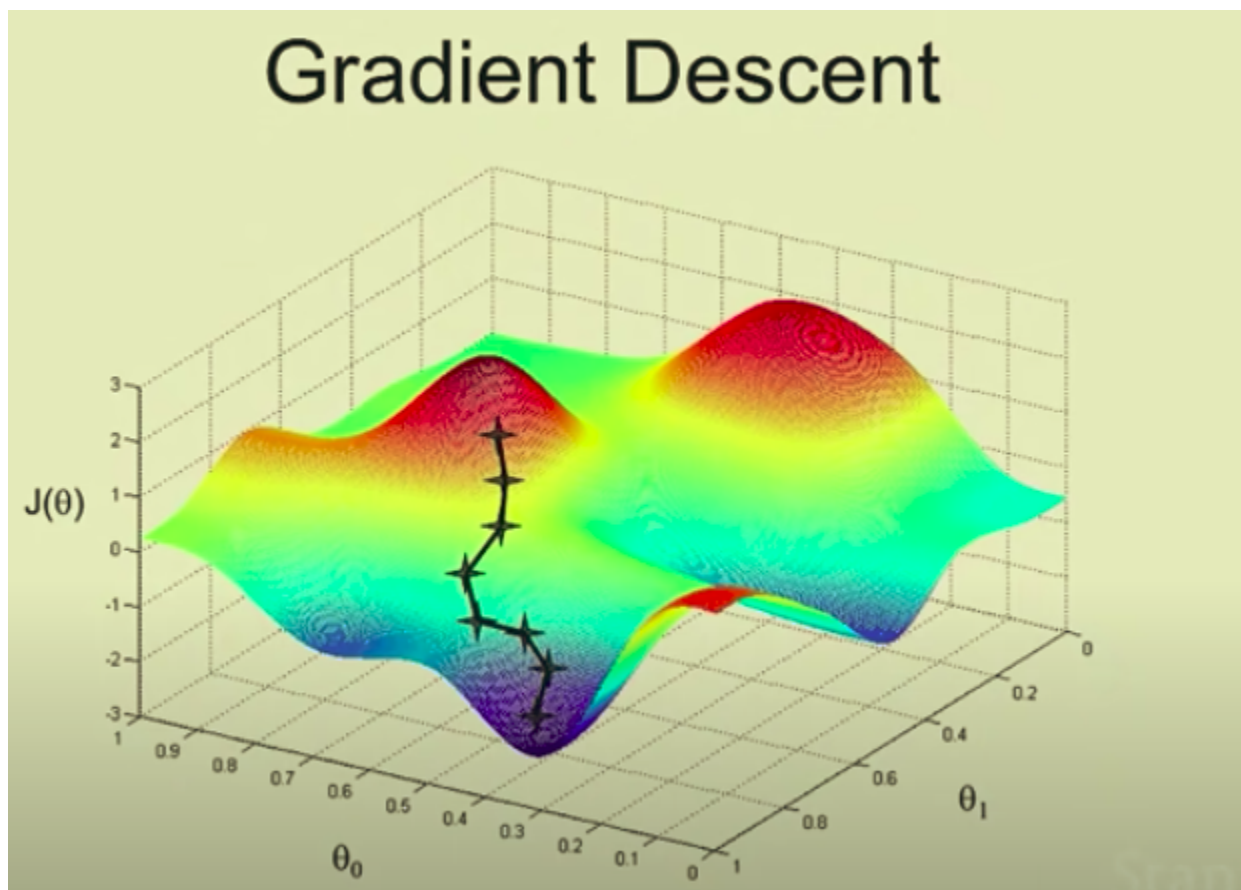


A photograph of a chalkboard with the cost function formula written in chalk. The formula is
$$\frac{1}{2} \sum_{i=1}^M (h_{\theta}(x^{(i)}) - y^{(i)})^2$$

- Given m training examples we sum through $i = 1$ to M . Say we have 50 examples we take the squared difference from what the algorithm predicts versus what the true price of the house is.
- Finally by convention we place a $\frac{1}{2}$ constant to make the math simpler.
- The cost function is known as $J(\theta)$ of θ . Therefore we find parameters that minimize the cost function.

GRADIENT DESCENT TO MINIMIZE COST FUNCTION $J(\theta)$

- θ is this 3 dimensional vector that we have to pass into the cost function.
- Say for example we have a vector of all zeros. Then we keep changing θ to reduce the cost function. $J(\theta)$



- What we want to do here is find values for θ_0 and θ_1 which minimize the height of the surface J of θ .
- We are trying to reach the lowest possible point for J of θ one tiny step at a time.
- What gradient descent will do is look around and find the route to the lowest possible elevation. Finding the steepest gradient one step at a time.
- It should be mentioned that depending on where you localize it will lead to different local optimums (endpoints).

$$\theta_{j+1} := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta)$$

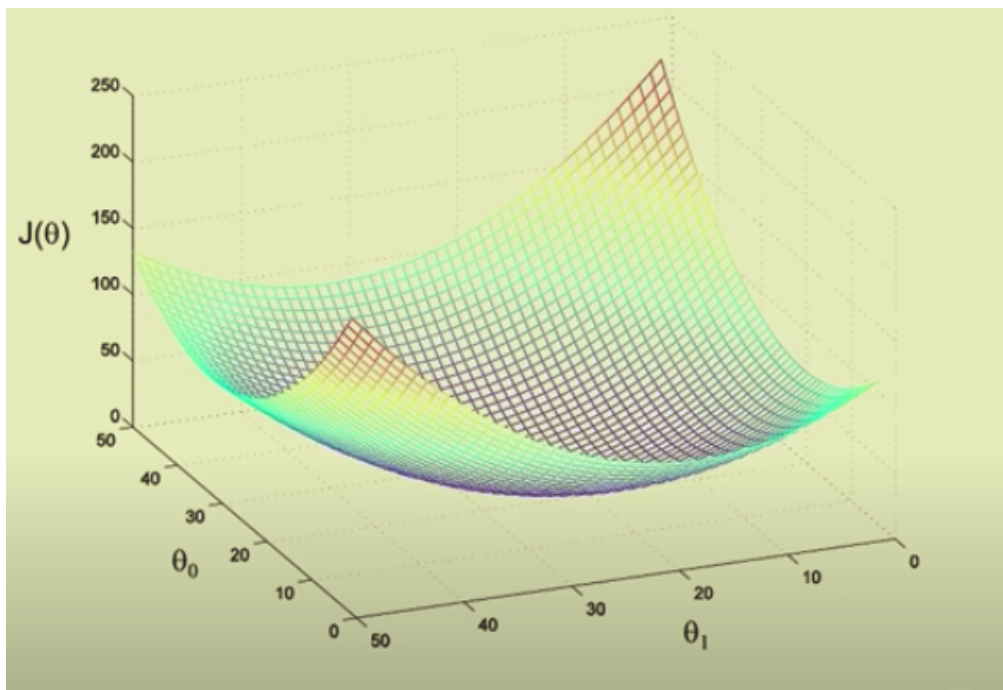
learning rate

$a+1, "a=b"$ $"a=a+1"$ $(j=0,1)$

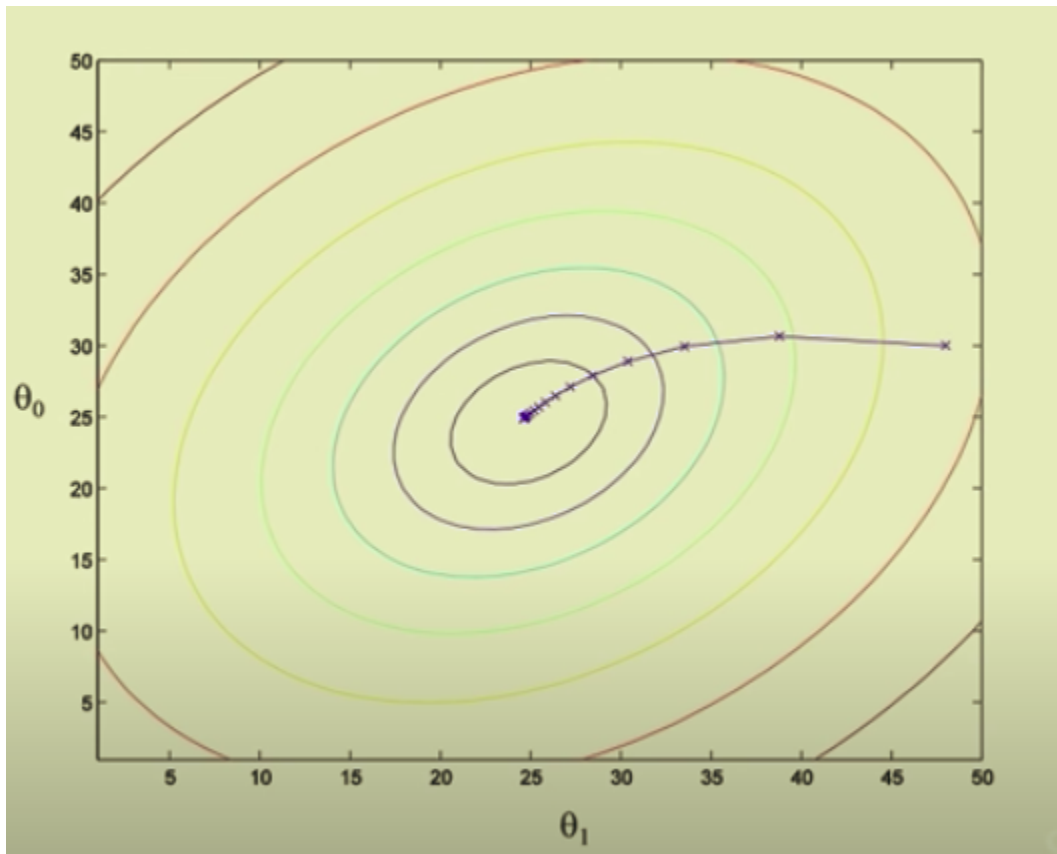
- Given examples up to n numbers, take θ_j and make it equal to θ_j minus α (learning rate) * the derivative
- In calculus the derivative of a function defines deepest descent.

LINEAR REGRESSION GRADIENT DESCENT

- When J of θ is defined as a sum of square terms the output is a quadratic function.



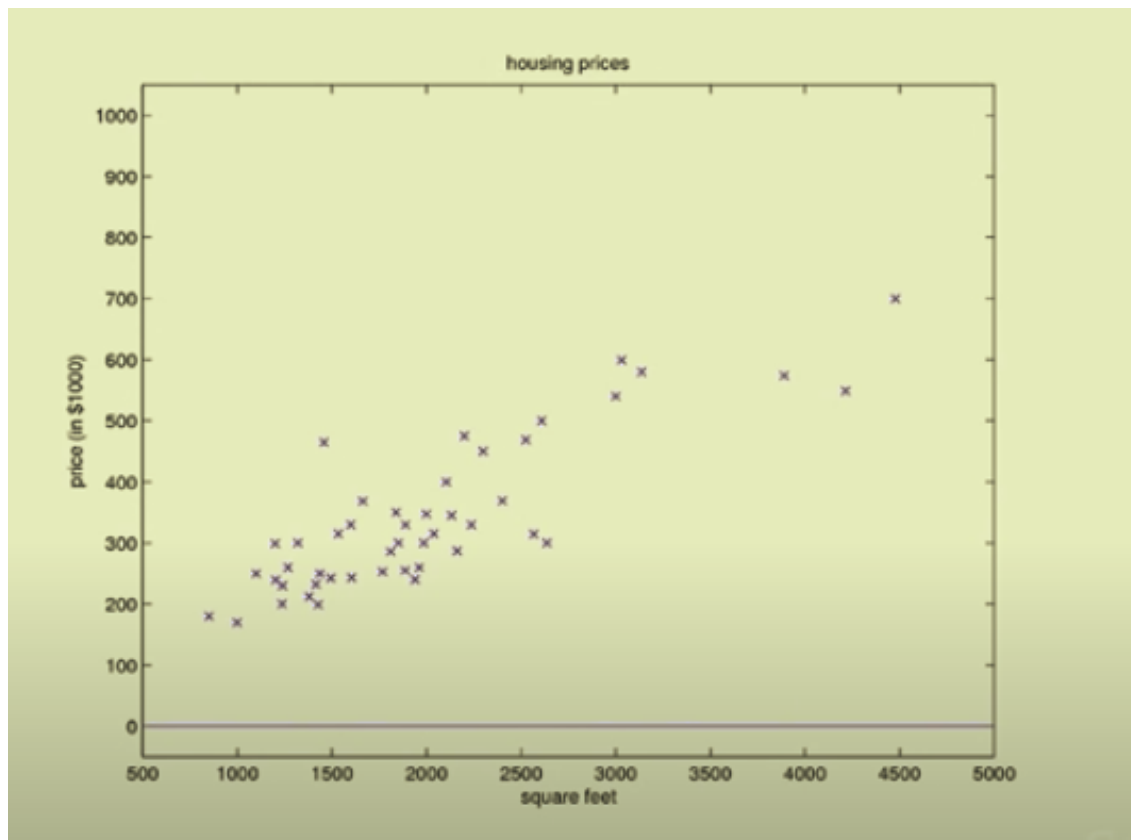
- J of theta does not have local optimum but rather global optimum. We take horizontal slices and plot the contours of the big bowl.



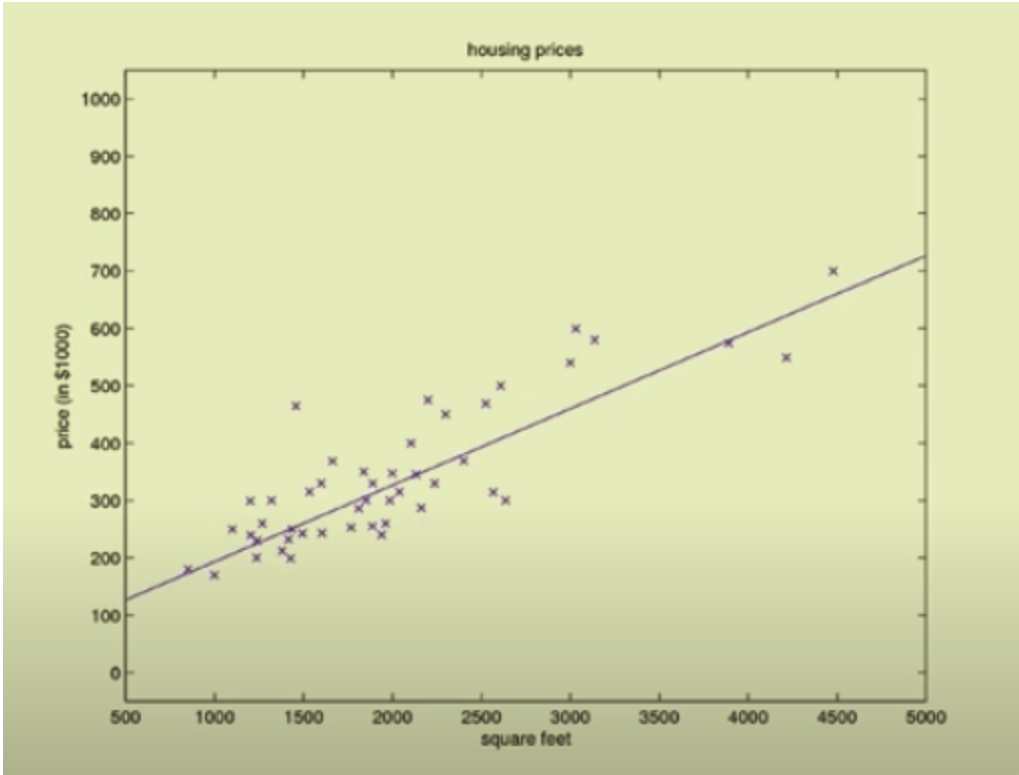
- Fun fact - the directional steepest descent is always at 90 degrees.
- As you take steps down hill because there is only one global minimum the algorithm will converge accordingly.
- If you set the learning rate α to be very large than it can overshoot, if you set it to be too small then you need a lot of iterations. What happens in practice is you try a few values, to see what value of α allows you to efficiently drive down the value for J of theta. If you see the cost function increasing rather than decreasing that is a strong signal that the learning rate is too large.

- You can try multiple values of alpha on an exponential scale. .02, .04, .08, .016 to see what values drive down the cost function the fastest.

VISUALIZATION



- There are 49 points in this dataset. If you initialize the parameters to 0 that means making your straight line fit to be zero, then your hypothesis is for any inputs of a house the estimated price is zero.
- After each iteration of gradient descent you are trying to minimize J of theta. It is trying to minimize $\frac{1}{2}$ of the sum of squared errors. (cost function).
- Eventually the end line will be portrayed as the following.



BATCH GRADIENT DESCENT

- You look at the entire training set of 49 examples and classify them as one batch. The disadvantage of batch gradient descent is that in order to make one update to the parameters (a single step of gradient descent) you must calculate the sum of the cost function for 100,000,000 training examples before you can even make one tiny step. This becomes computationally expensive.
- The alternative to batch gradient descent is Stochastic Gradient Descent.

Repeat {
For $i=1$ to m {
 $\theta_j := \theta_j - \alpha (h_{\theta}(x^{(i)}) - y^{(i)}) \cdot x^{(i)}$
}
}
Stochastic gradient

- Instead of scanning through all examples before updating parameters theta even a little bit, instead in the inner loop you loop through all examples where the derivative is taken with respect to one training example.



- In the visualization above for Stochastic Gradient: you initialize a parameter somewhere, look at your first training example and see if you can predict one house price better and modify the parameters to increase the accuracy of predicting the price for that one house.
- Because it is fitting the data for that one house, maybe you end up improving the parameters a little bit, but not quite going in the complete downhill direction. With each new house the parameters are updated. It takes a noisy path, but on average it is headed towards the global minimum. As you run stochastic it never quite converges because you are always trying to do better for that specific one house.
- With a very large dataset stochastic is used much more in practice due to computational expenses. Batch is just so slow.
- If you have a low amount of records in your dataset batch gradient descent is recommended because it is one less thing to fiddle with in regards to the parameters oscillating about with each new input.