Machine Learning

Summary of Andrew Ng' course on coursera.com

by Mehrdad Alemzadeh

- In statistical learning the aim is to find the relation between X and Y, not to predict Y precisely.
- Large data set is an important factor in ML
- Modeling for inference: i.e. increase in production of which product will increase the sell. Lassano and least square are some good choices because they result in highly interpretable models with less flexibility on the hypothesis function.
- Predictions:: i.e. house prices. Bagging, boosting, SVM are proper for predictions because it provides more flexibility in choosing h(x), and the more flexible the method the less interpretable.
- Supervised vs unsupervised learning vs semi-supervised
- Statistical learning methods:
  - Parametric: you decide on the form of the function and use the training data to find the parameters (most famous method is least squares). One problem with this approach is that you will end up having lots of parameters to fit the function properly but it can lead to overfitting.
  - Non-parametric: no assumptions are made about the function. A lot more data points are required for this method.

Model & Cost Function

- Hypothesis function: x is the future. $h_\theta(x) = \theta_0 + \theta_1 X$
- i.e. Housing prices
- Univariate and multi-variant
- Cost function is about finding the parameters, theta: $\theta = (X^T * X)^{-1} * X^T * Y = pinv(x' * x) * x'y$
  - Choose theta so that h(x) is close to y for training set
  - This is a minimization problem (difference between the horizontal values), which is equal to partial derivative of j in respect to theta

$$J(\theta_0, \theta_1) = 1/(2m) \sum_{1}^{m} (h_\theta(x^i) - y)^2 = \partial/\partial\theta_i * J(\theta)$$

  - The cost function can be plotted for the h(x) on a 2D graph or as a contour plot, and that will also provides a good visualization about how to set theta to minimize the cost

Evaluating uni-linear regression model

- Once the model is derived
  - **Assess the accuracy of the coefficients** (1. Average a huge number of observations on the hypothesis and the actual Y and see how close they are. Huge number required to avoid overfitting/underfitting; 2. Standard error of hypothesis=SE(h)^2=standard Dev of yi/n observations; 3. Compute standard error for thetas: x^2/(sum of x-X)^2; 4. t-statistics=(theta-0)/SE(theta), p-value is the probability of values greater than |t|. The smaller the p, we can reject null-hypothesis and infer that there is a relation between x and Y)
  - **Assess the accuracy of the model** (1. RSE: residual standard error=sqrt(1/n-2 * J(theta)) if it's small it fits data well, because it provides measure of lack of fit; 2. R^2=1-(J(theta)/sum(y-Y)), when it's close to zero it means the model didn't understand any relation), when adding feature R^2 increases because model fits better; when adding a new feature has less impact on R^2 it might mean that it's less significant

Multi-Linear Regression

- Suppose we have m examples and n features. $x^i = [x_1, x_2, \ldots, x_n]$
- x_j^i is the ith training example and the jth feature
- The function is: $h_\theta(x) = \theta_0 + \theta_1 x_1 + \ldots + \theta_n x_n = \theta^T X$
- Gradient descent: cost function for this algorithm should decrease through each algorithm, otherwise something like alpha could be wrong. If alpha is too small getting to min can get lengthly, and if it's too big you could overshoot and not get the min. Below is a gradient descent formula.

$$\theta_j := \theta_j - \alpha * 1/m * \sum_i^m (h_\theta(x^i) - y^i) * x_j^i$$

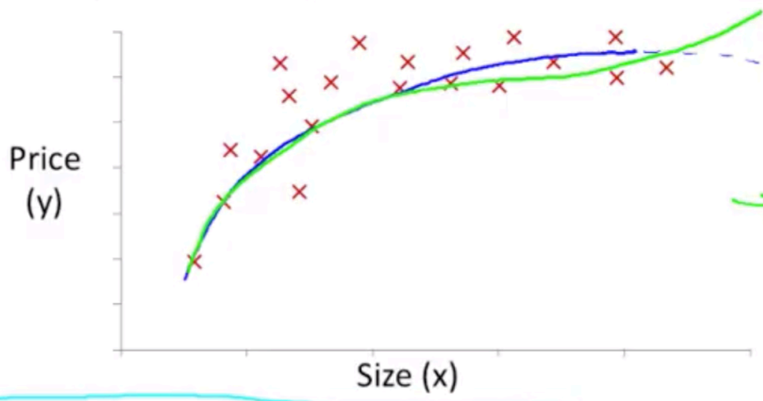
- Feature scaling: finding the min cost function with gradient descent will take a long time and it won't be an accurate model. Feature scaling is to divide each feature by it's own max amount. Or use normalization: (feature-mean of all)/(max-min)

Evaluating multi-linear regression model

- F-statistics = ((sum(y-Y)^2 - J(theta))/number of predicators) / (J(theta) / (n-p-1)); this determines if there is a relation between predictor and response. If value is close to 1, that means there are no relations.
- if p value is large we are likely to make some false discoveries. It is possible that all of the predictors are associated with the response, but it is more often the case that the response is only related to a subset of the predictors.
- We should sometimes try out different models with different features. We can use below methods to evaluate it: Mallow's Cp, Akaike information criterion (AIC), Bayesian information criterion (BIC), and adjusted R^2
  - We cannot consider all models if number of features is high (30 is high). To choose smaller subset we can use:
    - **Forward selection**: keep adding features on a linear regression that results in the lowest RSS until a threshold is met
    - **Backward selection**. We start with all variables in the model, and backward remove the variable with the largest p-value—that is, the variable that is the least statistically significant
    - **Mixed selection**: keep adding features until p value increases, then remove that feature

Choice of a function

Polynomial regression



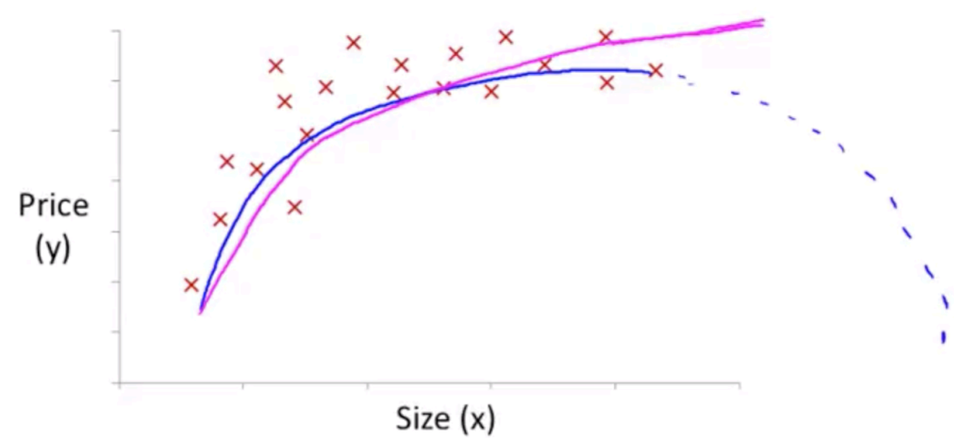
$\theta_0 + \theta_1 x + \theta_2 x^2$

$\theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_3$$
$$= \theta_0 + \theta_1 (size) + \theta_2 (size)^2 + \theta_3 (size)^3$$

$x_1 = (size)$
$x_2 = (size)^2$
$x_3 = (size)^3$

Choice of features



$$h_\theta(x) = \theta_0 + \theta_1 (size) + \theta_2 (size)^2$$

$$h_\theta(x) = \theta_0 + \theta_1 (size) + \theta_2 \sqrt{(size)}$$

Binary Classification

Logistic/Sigmoid Regression

$$h_\theta(x) = 1/(1 + e^{-\theta^T x}) = P(y = 1 \,|\, x; \theta)$$

- The possibility of y=1 when x depends on theta, for classification hypothesis based on logistic regression

- To know when h will predict 1, h should be >0.5 for sigmoid function, therefore, when $\theta^T x > 0$ then prediction is class 1. Same thing applies for non-linear

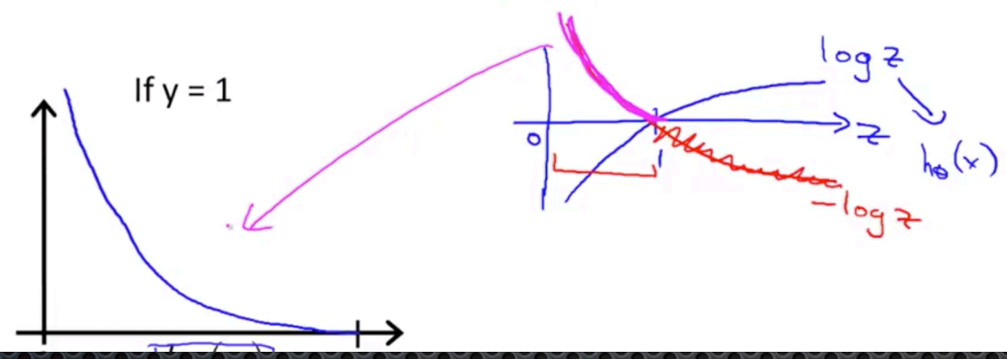
- Linear cost function doesn't work for sigmoid regression because it's not convex (it will have several local minima); a simple u shape is expected.

Binary Classification

Logistic/Sigmoid Regression - cont.

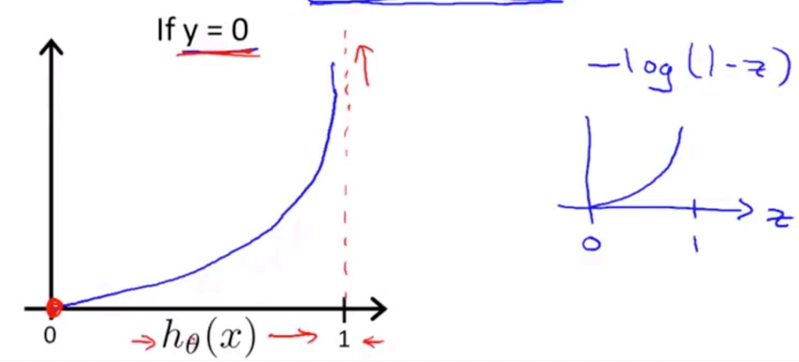
Logistic regression cost function

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} \boxed{-\log(h_\theta(x))} & \text{if } y = 1 \\ -\log(1 - h_\theta(x)) & \text{if } y = 0 \end{cases}$$



Logistic regression cost function

$$\mathrm{Cost}(h_\theta(x), y) = \begin{cases} -\log(h_\theta(x)) & \text{if } y = 1 \\ \boxed{-\log(1 - h_\theta(x))} & \text{if } y = 0 \end{cases}$$



Logistic regression cost function

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Cost}(h_\theta(x^{(i)}), y^{(i)})$$

$$= -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1 - y^{(i)}) \log\left(1 - h_\theta(x^{(i)})\right)\right]$$

Binary Classification

Logistic/Sigmoid Regression - cont.

Repeat {

$$\theta_j := \theta_j - \alpha \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)}) x_j^{(i)}$$

(simultaneously update all θ_j)

}

$$\begin{bmatrix} \vdots \\ \theta_n \end{bmatrix}$$

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

$h_\theta(x) = \frac{1}{1+e^{-\theta^T x}}$

Algorithm looks identical to linear regression!

Multi Class Classification

- 1-vs-all method: Consider class1 as 0 and remaining classes as 1 and use logistic classifier. Then we have n number of h(x) for each class. Once we get a new data point we'll compare it against each h(x) and take the highest probability.

Measure the quality of fit

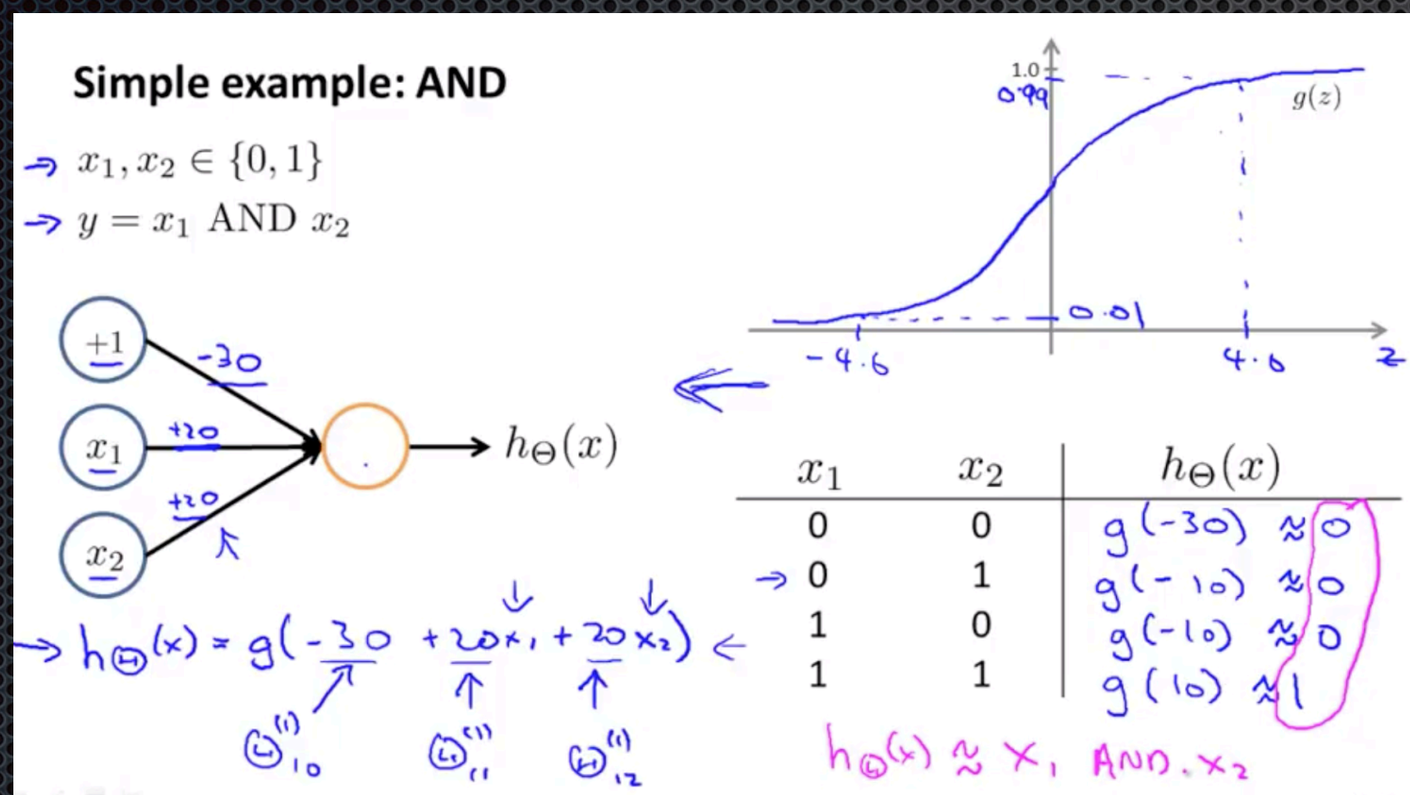
- Mean square error (MSE) for regression models

Neural Networks

- Solving non-linear problems
- X0 is always 1, called biased unit
- Theta are the parameters or weights in NN
- The hypothesis is similar to logistic regression. The difference is that, the input values x1…xn are not the direct values used in h(x), they are wired to hidden layers and the NN will feed its learned features to a sigmoid hypothesis. You can have even more complex features in multiple hidden layers.

Neural Networks

Logical networks



x_1	x_2	$h_\Theta(x)$
0	0	$g(-30) \approx 0$
0	1	$g(-10) \approx 0$
1	0	$g(-10) \approx 0$
1	1	$g(10) \approx 1$

Neural Networks

Cost function and regularization

Cost function

Logistic regression:

$$J(\theta) = -\frac{1}{m}\left[\sum_{i=1}^{m} y^{(i)} \log h_\theta(x^{(i)}) + (1-y^{(i)}) \log(1-h_\theta(x^{(i)}))\right] + \frac{\lambda}{2m}\sum_{j=1}^{n}\theta_j^2$$

Θ_0

Neural network:

$$h_\Theta(x) \in \mathbb{R}^K \quad (h_\Theta(x))_i = i^{th} \text{ output}$$

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right]$$

$$+\frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{ji}^{(l)})^2$$

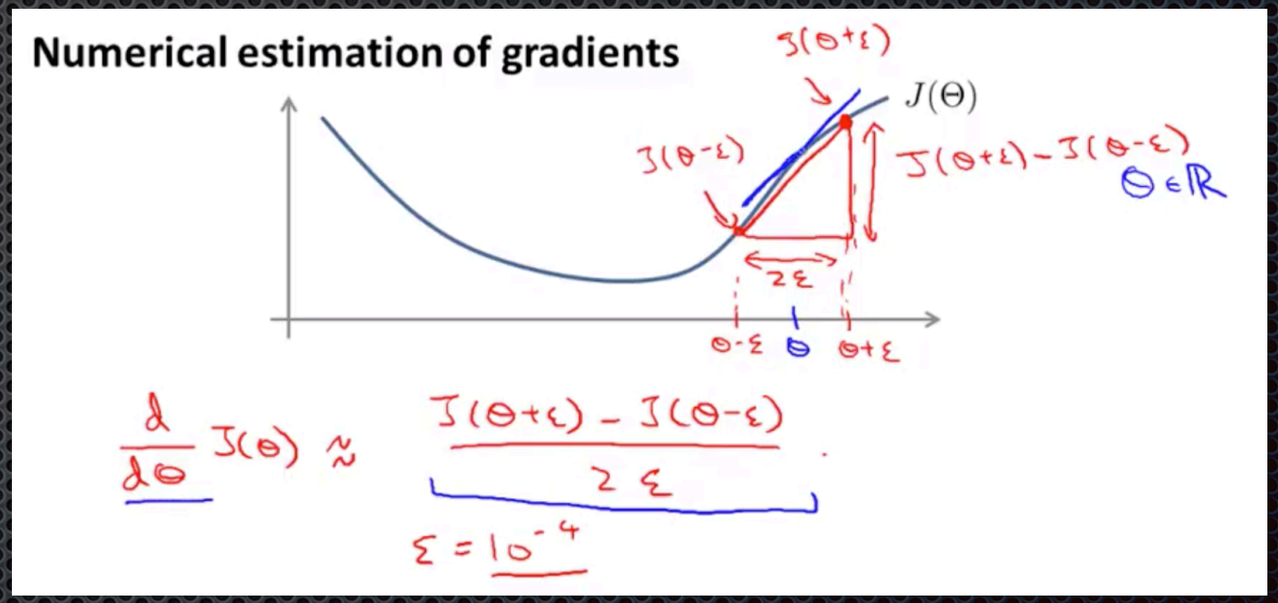
Neural Networks

Back-Propagation

- Based on the results of your initial learning, you can find the error at the end layer, just like we have done so far; h(x)-y. The only difference is that we'll continue this error assessment every layer backwards to the first layer.
- The results of back propagation is exactly the partial derivative of the cost function in respect to the parameters/weights.
- Steps to setup NN:
  - Randomly set the weights
  - Implement forward propagation to calculate h(x) for all x
  - Implement cost func J(theta); this will not be convex so could have local-minima issue
  - Implement back-propagation to compute spatial derivate of J(theta)
  - If your J of cross validation is higher than J of training for a NN, then should we add more hidden layers to address the issue? No because it's suffering from high variance.

Gradient checking

- Sometimes back-propagation seems to be working and the J(theta) seems to be decreasing however the model might have some errors. Below method is called 2-sided difference. NN weights should be randomly initialized for symmetry breaking

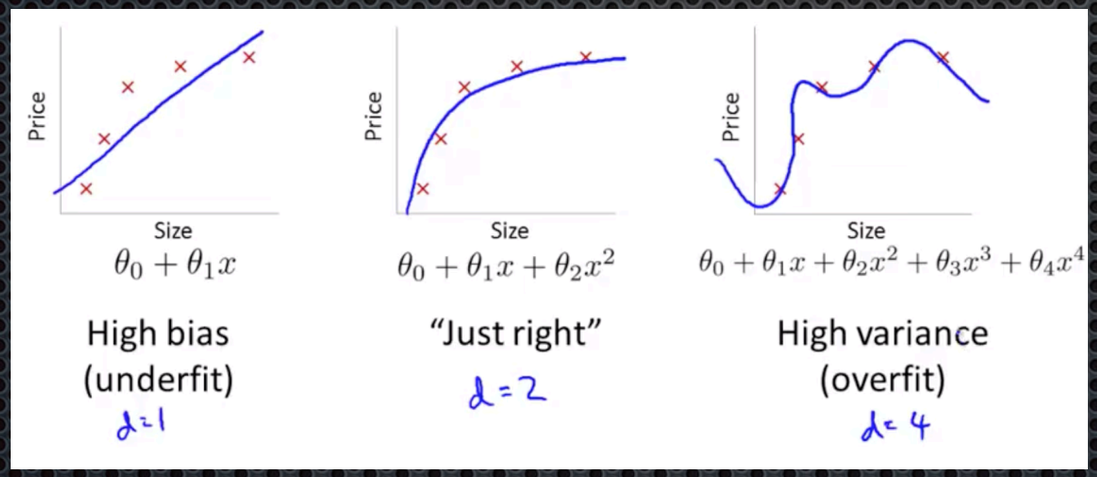


-

Improve Learning

- Sometimes your learning algorithm doesn't predict well
- One way is to get more data, but not always
- Another way is to get more features, but too many features can contribute to bad predictions
- You can choose different learning parameters or algorithms like polynomial features and etc. but these may be unnecessary
-

Evaluate the Hypothesis

- Randomly choose 60% of your data for training, 20% for cross validation (CV) and 20% for testing
- For different models (linear, different degrees of polynomial) calculate vector of theta and then evaluate J(theta) for each one on the CV set of data and choose the one with least error
- Choose the regularization parameter lambda from 0 to 10 or higher (moving up in multiplies of two and 2 decimal points 0, 0.01, 0.02, 0.04,...). Calculate the J on CV and choose the min.
- 

- Get more training examples → fixes high variance
- Try smaller sets of features → fixes high variance
- Try getting additional features → fixes high bias
- Try adding polynomial features $(x_1^2, x_2^2, x_1 x_2, \text{etc})$ → fixes high bias.
- Try decreasing λ → fixes high bias
- Try increasing λ → fixes high variance

Variance (overfit) vs Bias (under fitting)

- Goal is low variance and low bias
- Variance means, ideally a small change in training data shouldn't impact h(x).
  - Root cause:
    - large number of features and little number of training sets.
    - Regularization: add it to cost function

$$J(\theta) = 1/2m[\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2 + \lambda \sum_{i=1}^{n}\theta_j^2]$$

$$\theta_j = \theta_j - \alpha[1/m \sum_{i=1}^{m}(h_\theta(x^i) - y^i)x_j^i + \lambda/m * \theta_j]$$

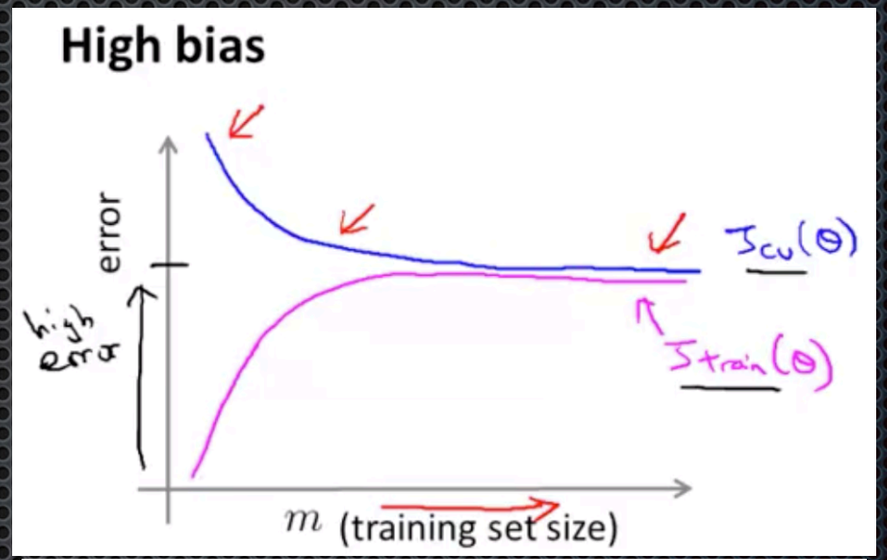
$$\theta = [X^T X + \lambda * matrix(n + 1 * n + 1)]^{-1} * X^T y$$

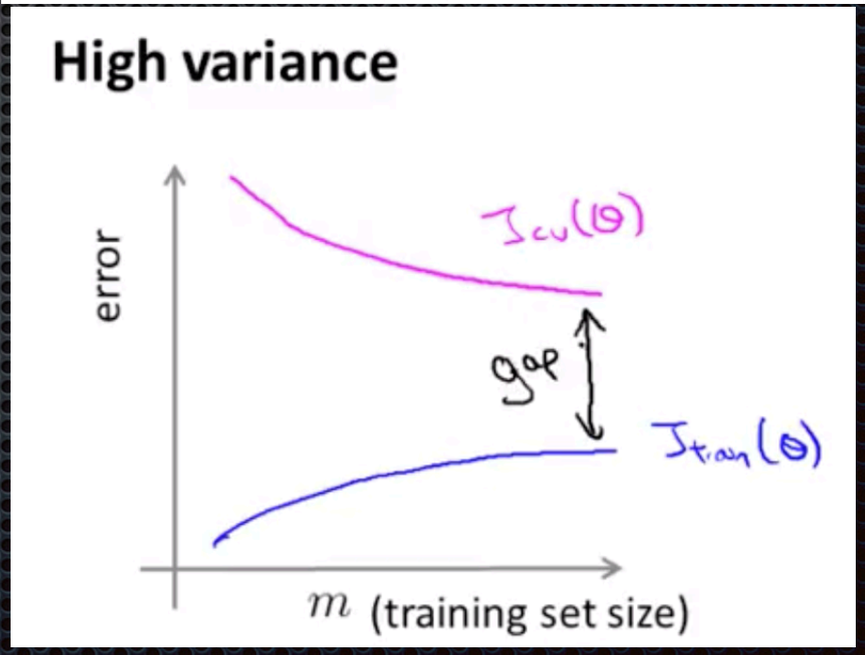
    - Lambda is the bias rate, matrix(n+1,n+1) is a zero matrix with diagonal 1 except for first element.
- Bias is high when h(x) is too simple for training data

Variance (overfit) vs Bias (under fitting) cont.

- You can figure out bias vs variance based on the number of training examples. If both J for training and CV are high it's a bias situation and increasing examples won't help. You need to add more features not training examples.

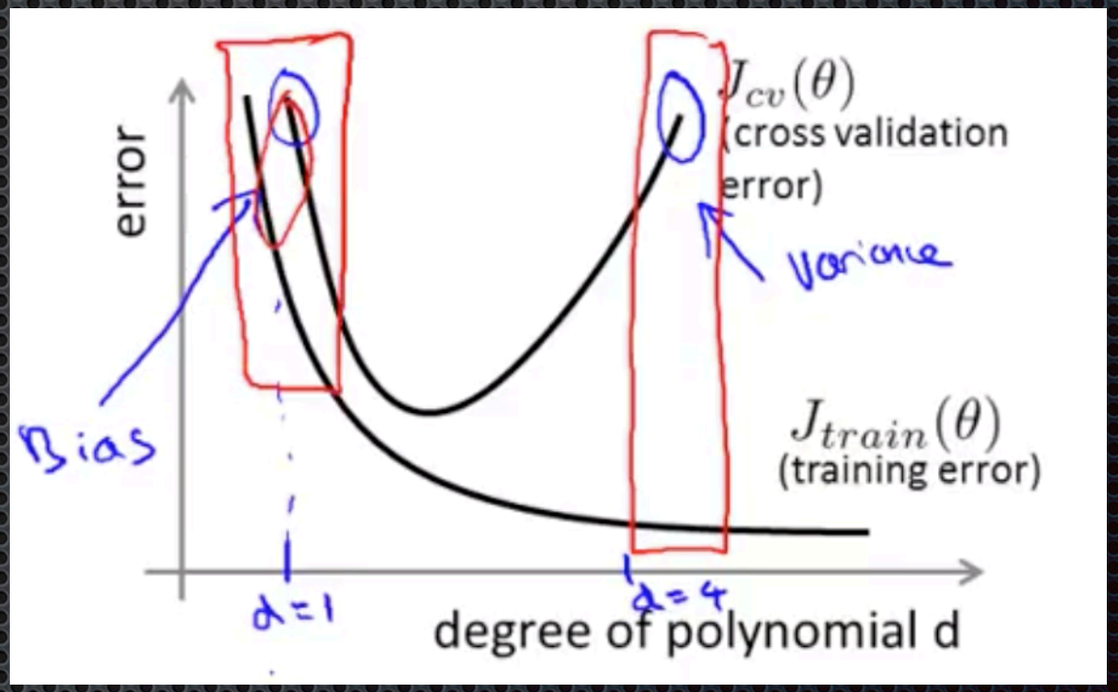
- In case of high variance, there is a bigger gap between J's. Increasing examples could be helpful in this case.





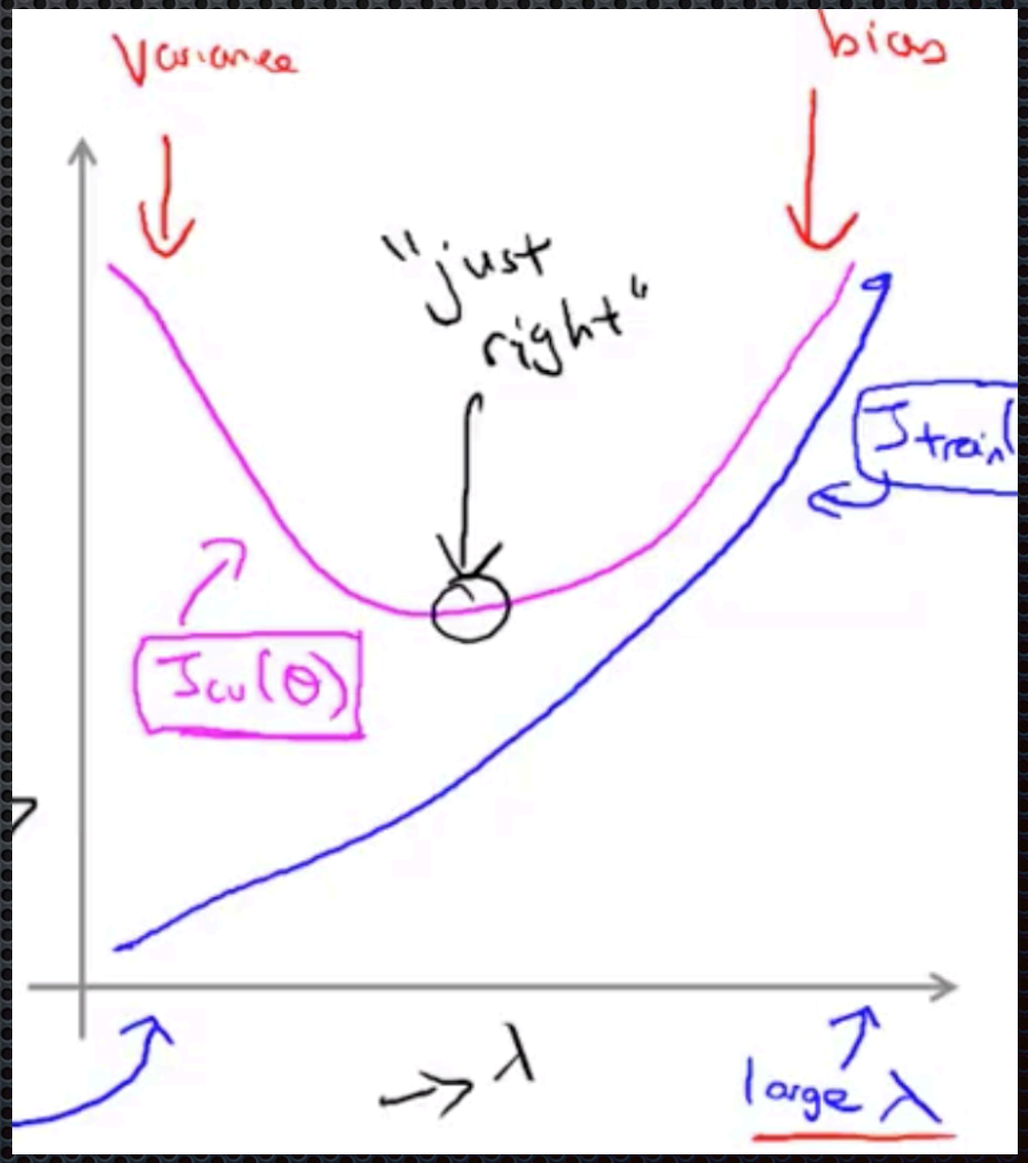
Variance (overfit) vs Bias (under fitting) cont.

- If we increase the degree of polynomial, error will decrease for training set. See the behavior in comparison with J(theta) of cross validation or test set. To understand if the issue is a bias or variance issue, you can visualize the relation between J's. A 20% difference is considered a lot (variance issue)



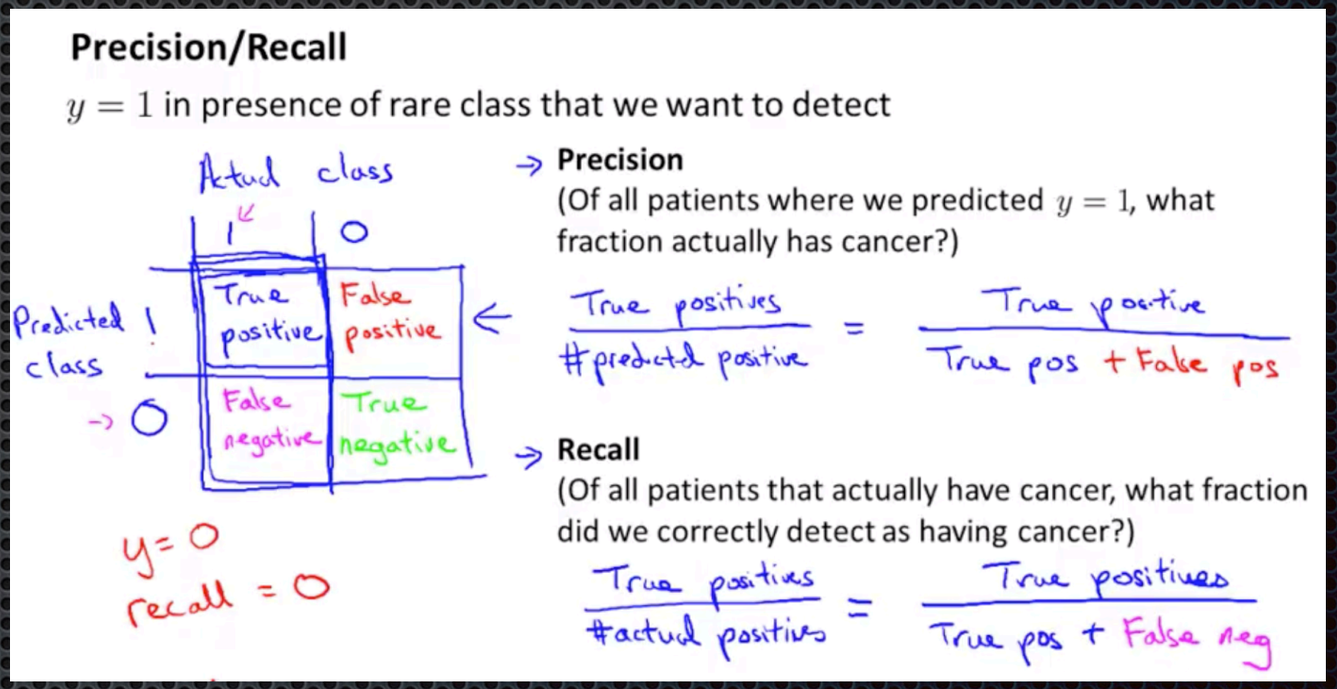
Variance (overfit) vs Bias (under fitting) cont.

- Regularization is necessary to prevent overfitting or bias situations. The horizontal axis is lambda and the vertical axis is the J(theta). The optimum value of lambda is somewhere in the middle.



Precision and Recall

- The higher the precision and recall the more accurate the classification is.

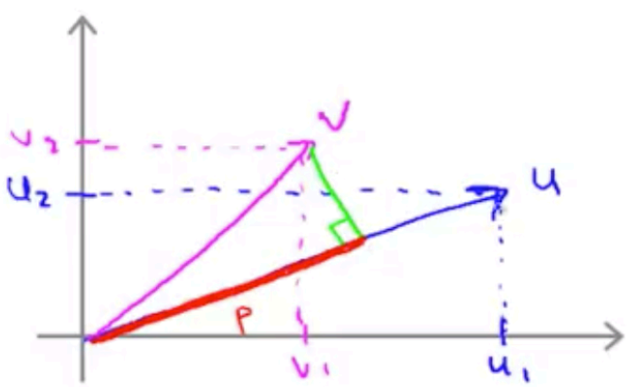


Support Vector Machine

- Find the optimal line between data sets

$$\min_{\theta} C \sum_{i=1}^{m} \left[y^{(i)} cost_1(\theta^T x^{(i)}) + (1 - y^{(i)}) cost_0(\theta^T x^{(i)}) \right] + \frac{1}{2} \sum_{i=1}^{n} \theta_j^2$$

Vector Inner Product



$$\vec{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} \quad \vec{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

$u^T v = ?$ $\quad [u_1 \;\; u_2] \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$

$\|u\|$ = length of vector u

$= \sqrt{u_1^2 + u_2^2} \in \mathbb{R}$

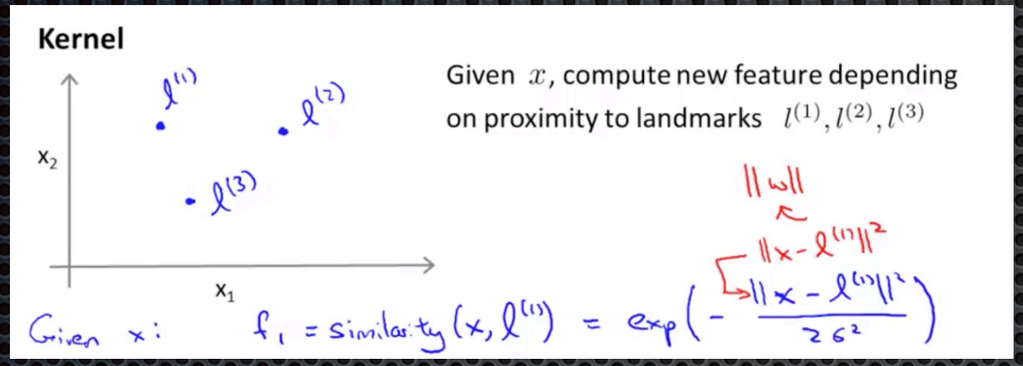
p = length of projection of v onto u.

$u^T v = p \cdot \|u\| \leftarrow \quad = v^T u$

$= u_1 v_1 + u_2 v_2 \leftarrow$

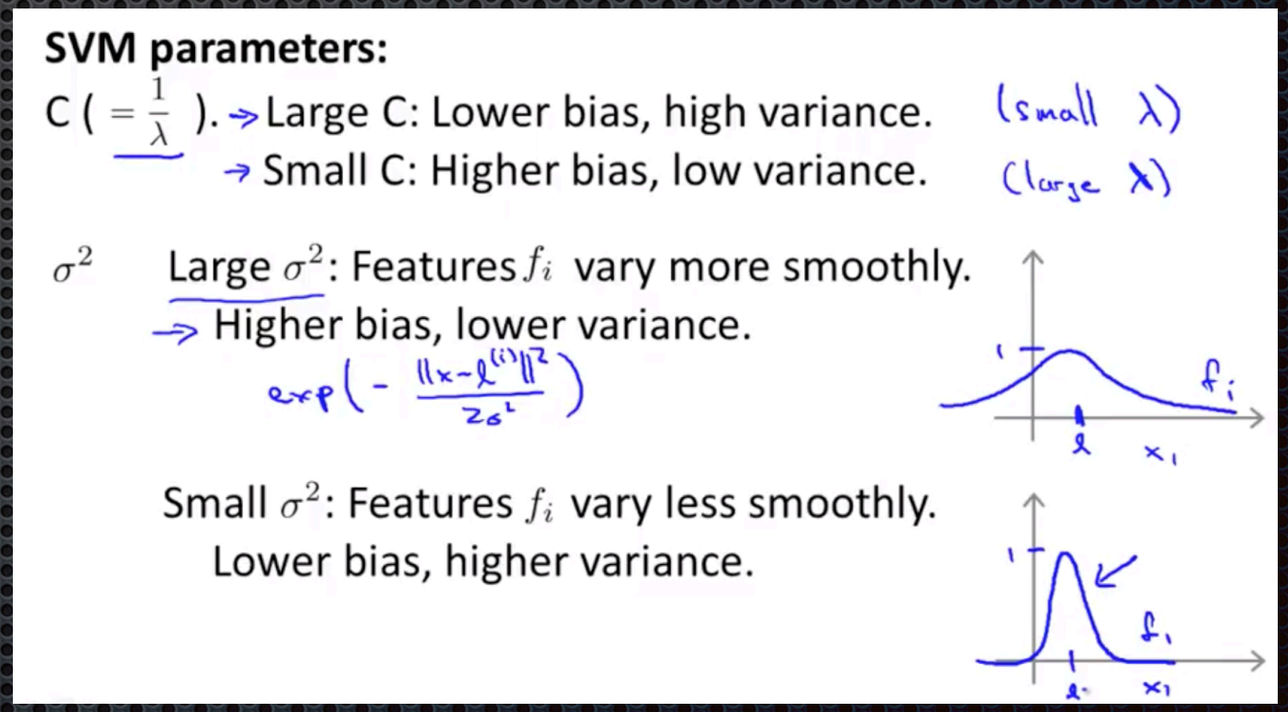
SVM Gaussian Kernel & parameterizing

- This kernel is defined by the distance between the land mark and x. If they are close then f1=exp(0)=1
- Land marks are the locations on each and every x



Predict "1" when

$$\theta_0 + \theta_1 f_1 + \theta_2 f_2 + \theta_3 f_3 \geq 0$$



Principal Component Analysis

- The purpose of PCA is to reduce the error; distance between data and the model describing the data (i.e. a liner line going through data)
  - With regression when we draw a line between the feature and the line, it's a vertical line to the axis of the feature. however, with PCA the line between the point and the line is actually vertical to the line (model)
- Features should be scaled and mean normalized for PCA
- PCA reduces the dimension
-

Movie Prediction: recommendation system

- Theta is the best vector describing user's rating of other movies. If he gave 5 start to romance movies, we find a theta that works for that.

$n_u = 4$, $n_m = 5$

$x_0 = 1$

$x^{(1)} = \begin{bmatrix} 1 \\ 0.9 \\ 0 \end{bmatrix}$

Content-based recommender systems

Movie	Alice (1) $\theta^{(1)}$	Bob (2) $\theta^{(2)}$	Carol (3) $\theta^{(3)}$	Dave (4) $\theta^{(4)}$	x_1 (romance)	x_2 (action)
$x^{(1)}$ Love at last 1	5	5	0	0	0.9	0
$x^{(2)}$ Romance forever 2	5	?	?	0	1.0	0.01
$x^{(3)}$ Cute puppies of love 3	? 4.95	4	0	?	0.99	0
Nonstop car chases 4	0	0	5	4	0.1	1.0
$x^{(5)}$ Swords vs. karate 5	0	0	5	?	0	0.9

$n = 2$

For each user j, learn a parameter $\theta^{(j)} \in \mathbb{R}^3$. Predict user j as rating movie i with $(\theta^{(j)})^T x^{(i)}$ stars.

$\theta^{(j)} \in \mathbb{R}^{n+1}$

$$x^{(3)} = \begin{bmatrix} 1 \\ 0.99 \\ 0 \end{bmatrix} \longleftrightarrow \theta^{(1)} = \begin{bmatrix} 0 \\ 5 \\ 0 \end{bmatrix} \qquad (\theta^{(1)})^T x^{(3)} = 5 \times 0.99 = 4.95$$

To learn $\theta^{(j)}$:

$$\min_{\theta^{(j)}} \frac{1}{2m^{(j)}} \sum_{i:r(i,j)=1} \left((\theta^{(j)})^T (x^{(i)}) - y^{(i,j)} \right)^2 + \frac{\lambda}{2m^{(j)}}$$

Movie Predictions: type of movie determination

- Suppose each user has told you what they like (action or romance, etc.)

Optimization algorithm

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(i)}$:

$$\min_{x^{(i)}} \frac{1}{2} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Given $\theta^{(1)}, \ldots, \theta^{(n_u)}$, to learn $x^{(1)}, \ldots, x^{(n_m)}$:

$$\min_{x^{(1)}, \ldots, x^{(n_m)}} \frac{1}{2} \sum_{i=1}^{n_m} \sum_{j:r(i,j)=1} \left((\theta^{(j)})^T x^{(i)} - y^{(i,j)} \right)^2 + \frac{\lambda}{2} \sum_{i=1}^{n_m} \sum_{k=1}^{n} (x_k^{(i)})^2$$

Movie Prediction: recommend other movies

- Find some features for all these movies such as romance, action, comedy and etc
- Then find the min distance between the features of movie i and j; the one with the least distance should be preferred by the user
- If we have a user with no ratings, we use mean normalization (add up all user ratings and divid but those who have rated). This will be the theta for that user

Stochastic Gradient Descent

- So far we have derived the algorithm used cost function like gradient descent to minimize that cost function.
- You can use stochastic gradient descent to update theta one at a time by going through training data one by one instead of summing them all up. This is when you have identified a biased situation through depicting J's of training and CV based on number of data sets and realized increasing data set won't help. You can also choose mini batches by iterating through 10 training data at a time instead of one
- As always we need to make sure that gradient descent is converging and minimizing over iterations. If it's not increasing or decreasing that means the algorithm isn't learning much and something needs to change. If it's increasing maybe you should decrease learning rate alpha. You can generally decrease alpha over time to make it converge better

Stochastic gradient descent

1. Randomly shuffle (reorder) training examples

2. Repeat {

   for $i := 1, \ldots, m$ {

   $$\theta_j := \theta_j - \alpha(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

   (for every $j = 0, \ldots, n$)

   }

}

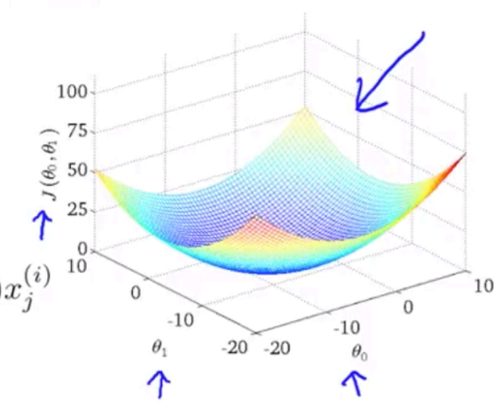
Linear regression with gradient descent

$$h_\theta(x) = \sum_{j=0}^{n} \theta_j x_j$$

$$J_{train}(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Repeat {

$$\theta_j := \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(for every $j = 0, \ldots, n$)

}