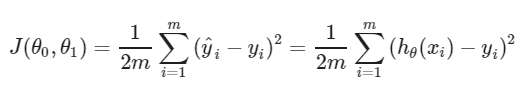
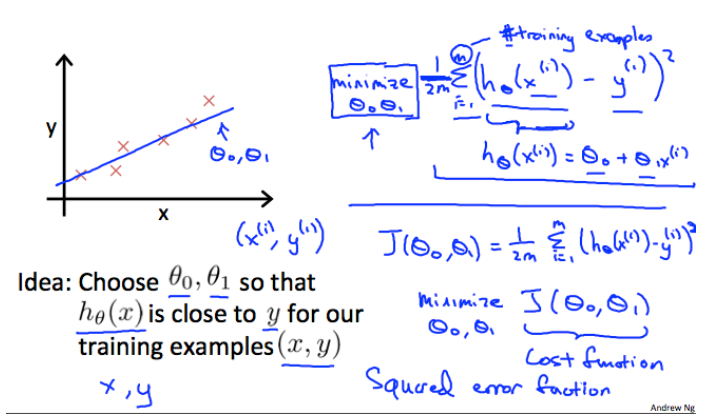
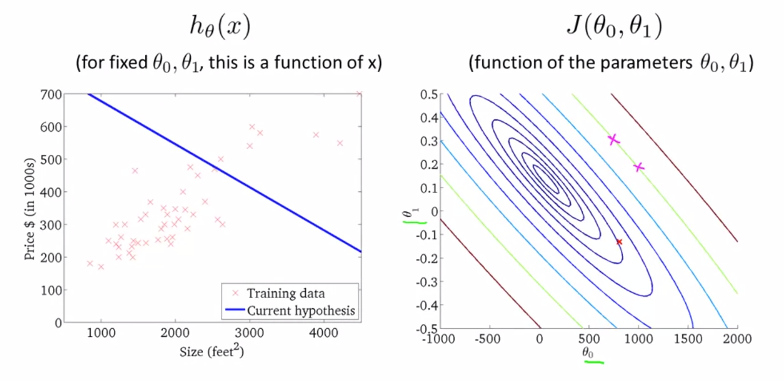
**Week 1:**

1. Algo and math not just enough, application is imp
2. ML grew out of AI
3. Examples:
   1. Click stream data used know more about users
   2. Medical record to medical knowledge
   3. Genomics
   4. Autonomous helicopter (can’t be programmed)
   5. NLP
   6. Handwriting recognition
   7. CV
   8. Recommendation systems
4. ML – ability to learn without being explicitly programmed (Checkers example)
5. Supervised Learning – labelled dataset (for every data point we had “right” answer)
   1. Classification / Regression
6. Unsupervised Learning – don’t give “right” answers
   1. Clustering
      1. Google news (cohesive news clustering)
      2. Clustering people into group from gene data
      3. Social network analysis
      4. Market segmentation
   2. Cocktail Party problem – hard to hear people in noisy party
      1. Multiple speaker recording multiple people’s audio
      2. Pass them to algo, then algo will split the audios
7. Model Representation – Regression Supervised problem (hypothesis)
8. Cost Function – Help fit best line to data (choosing parameters of model changes hypothesis function)
   1. Linear regression – we need to determine 2 parameters θ0 and θ1 so as to minimize squared error of all the training examples

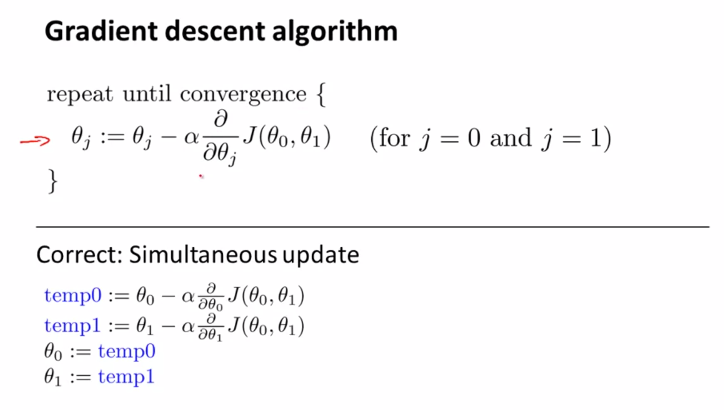




* 1. Contour plots used for better visualizing cost function minima point

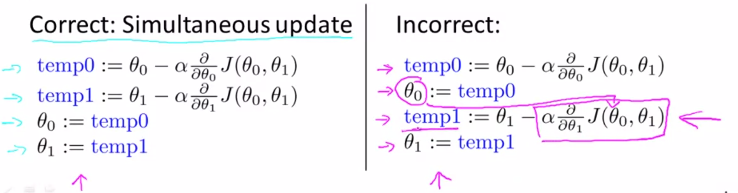


1. Gradient Descent – used to minimize cost function (in linear regression). It applied to any general cost function
   1. Vary θ0 and θ1 and go towards local minima



Alpha – learning rate, derivative of cost function (slope of tangent)

* 1. Correct way to implement gradient descent (simultaneous update)

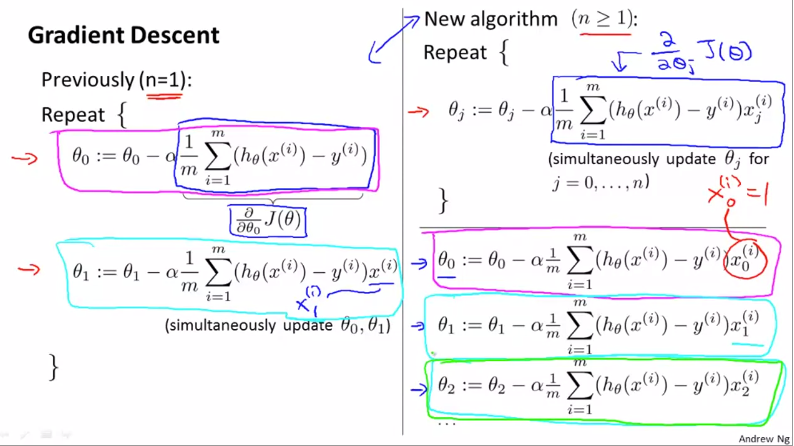


* 1. For linear regression, we will always have a convex function for cost

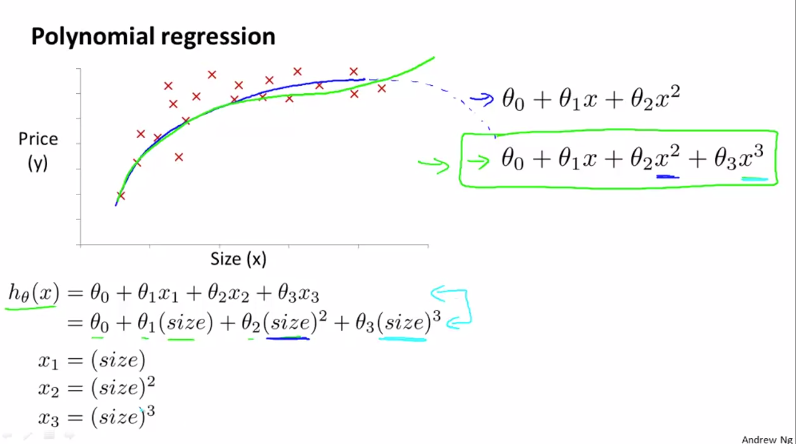
1. Linear Algebra:
   1. Matrix: dimensions
   2. Vector: n x 1 matrix, 1 indexed (1,2, 3….), 0 indexed (0,1, 2…)
   3. Matrix Algebra: addition, scalar multiplication, vector multiplication
      1. Express linear regression hypothesis function as vector multiplication
      2. Prediction = data matrix \* parameters
      3. Method to apply multiple hypothesis on dataset (linear regression)
   4. Matrix properties:
      1. A X B not equal to B X A
      2. A X (B X C) = (A X B) X C
      3. A X I = I X A = A
      4. Matrix which don’t have inverse are singular/ degenerate

**Week 2:**

1. Multiple Features (Multivariate linear regression):
   1. n = number of features, h depends on θ0 θ1 θ2 θ3
   2. To accommodate the constant term in the equation of hypothesis, theta matrix is represented in 0 indexed notation consisting of n+1 term
2. Gradient Descent for multiple variables:

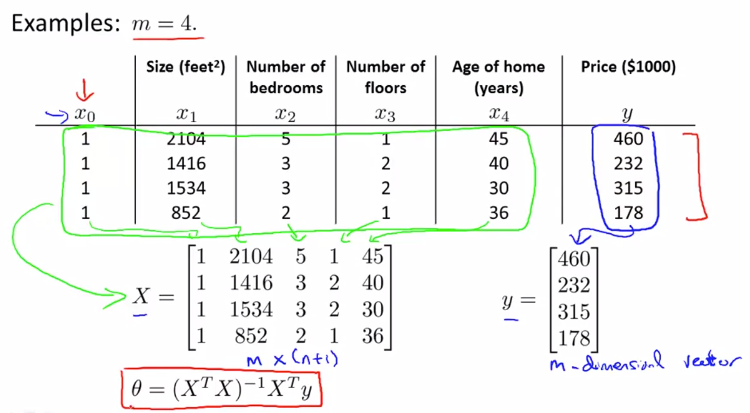


1. Feature Scaling:
   1. Convergence is fast if all features are in similar scale otherwise, we might get skewed contours
   2. It’s better to scale them in 0 to 1 (or -1 to +1) which will help in reaching global minimum faster
   3. Mean normalization: subtract all values with mean (and divide by range)
2. Learning Rate (alpha) (Practical aspect):
   1. Plot minimum cost function over iterations
   2. Ideally, the plot should decrease with each iteration
   3. When the graphs flatten, it means gradient descent has converged
   4. If the graphs do not show the usual trajectory of decrement and instead increases over iterations (or looks periodic), the gradient descent is not working properly and the alpha needs to be reduced.
   5. If alpha too small, slow convergence. If alpha is large, cost function may not converge over iterations
3. Features and Polynomial Regression:
   1. We can create new features (area), instead of using the features we already have (length, width)
   2. This can help in making better model

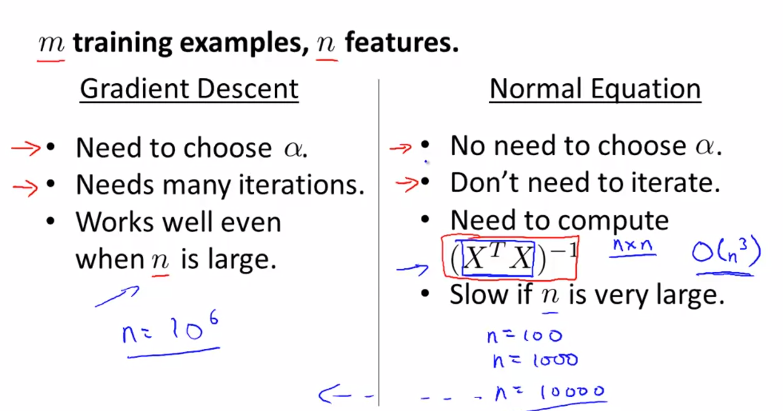


* 1. For polynomial regression, the higher order terms (square, cube) are treated as features
  2. The range will drastically change in polynomial regression, therefore scaling is important

1. Normal Equation:
   1. It is an alternative to Gradient Descent without any need of derivation
   2. Ideal way to minimize cost function is to take derivative over all thetas, equating them to zero and thus finding all thetas (features).
   3. The way to find theta that minimizes the cost function is gives as shown below



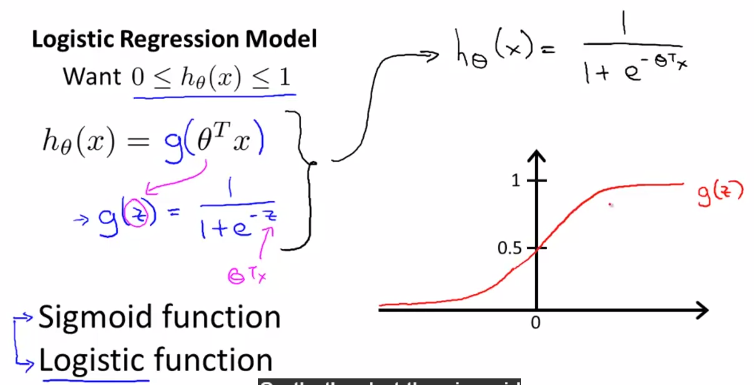
* 1. For normal equation method, it’s okay to not do feature scaling
  2. Gradient Descent v/s Normal Equation



1. Normal Equation Non-invertibility (Singular): How normal equation can be implemented for singular X transpose. X
   1. It can happen if we have redundant features (columns are linearly row)
   2. Too many features (solution – delete features, use regularization)
2. MATLAB/ Octave:
   1. Octave is used extensively for prototyping
   2. Load/ Store:
      1. who/whos = show current variables
      2. load(‘filename.dat’) = load data from files
      3. save filename.mat variable\_name = to save variables in file (compress in binary format)
      4. save filename.txt variable\_name -ascii = human readable format
   3. Basic computation:
      1. A\*B = cross product
      2. A.\*B = element wise multiplication
      3. log(A), abs(A), exp(A)
      4. A’ = A transpose
      5. A < 3 = element wise comparison (generate 1,0 pattern)
      6. find (A < 3) = return row and column vectors satisfying this condition
      7. pinv(A) = inverse of A (pseudo inverse mathematically)
   4. Visualization:
      1. Imagesc() = visualize matrices

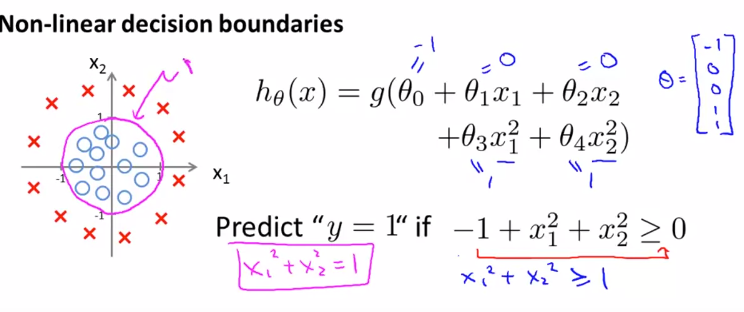
**Week 3:**

1. Classification:
   1. E.g., email spam, transaction fraud, tumour
   2. y (binary classification) can be 0 (negative class),1 (positive class)
   3. We can simply apply linear regression and put threshold as 0.5 to split it between 2 classes (not ideal)
2. Hypothesis - Logistic Regression:
   1. Classification algorithm (not regression like name suggests)
   2. We want hypothesis that gives answers between 0 and 1

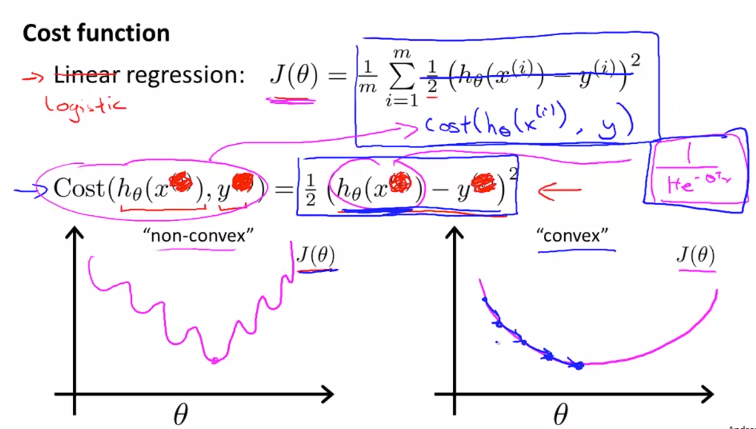


* 1. Sigmoid asymptotes 0 at x = -infinity and 1 at x = +infinity
  2. If h(x) = 0.7,
     1. it means there is a 70% chance (probability) of class 1 for given features x
     2. it means there is a 30% chance (probability) of class 0 for given features x

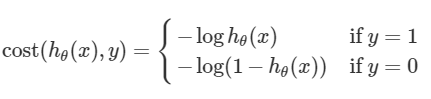
1. Decision Boundary – Logistic Regression
   1. The 0.5 on the sigmoid is the point where we make decision for the classifier
   2. Whenever it is (theta)’\*X >=0, it is class 1 otherwise class 0 (this line is the decision boundary)
   3. Decision boundary is the property of the hypothesis and not of the dataset (need dataset for fitting)
   4. We can add higher order terms to develop for non-linear decision boundaries

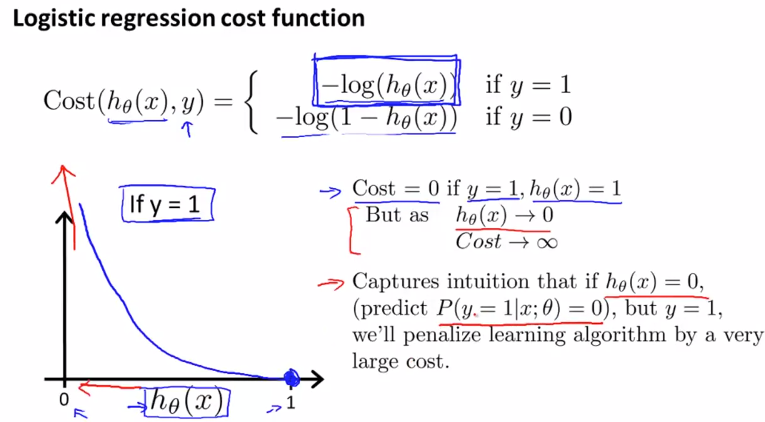


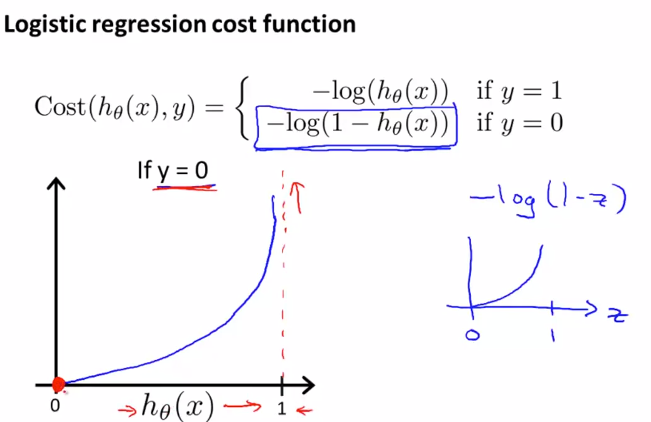
1. Cost function: optimization function to fit the hypothesis parameters
   1. If we use the cost (squared error) from linear regression for logistic regression, the cost function will be non-convex meaning that it will have multiple local minima and therefore there is no guarantee to reach global minima by gradient descent (by plugging sigmoid in linear regression cost equation)
   2. To overcome this, we need J(theta) to be a convex function of theta despite having a non-linear term of sigmoid in equation of hypothesis, which squared error does not allow



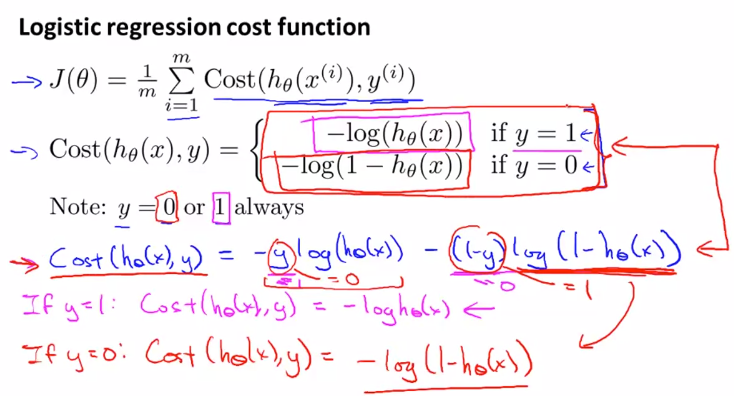
* 1. Cost equation must have a logarithmic term (unlike squared error in linear regression) so as to compensate for the non-linear sigmoid hypothesis and therefore to eventually have a convex cost function.



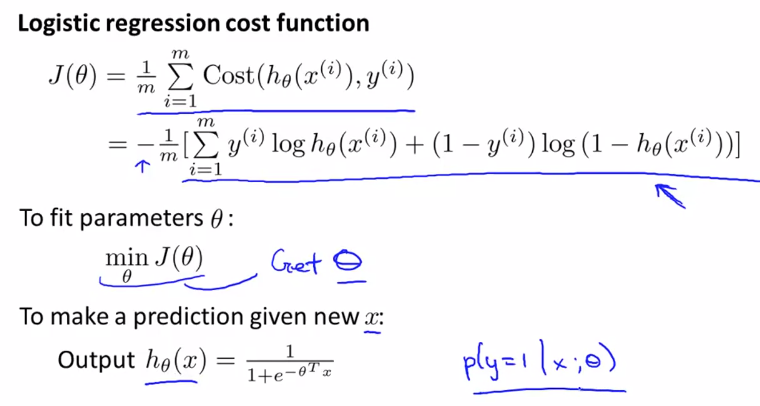




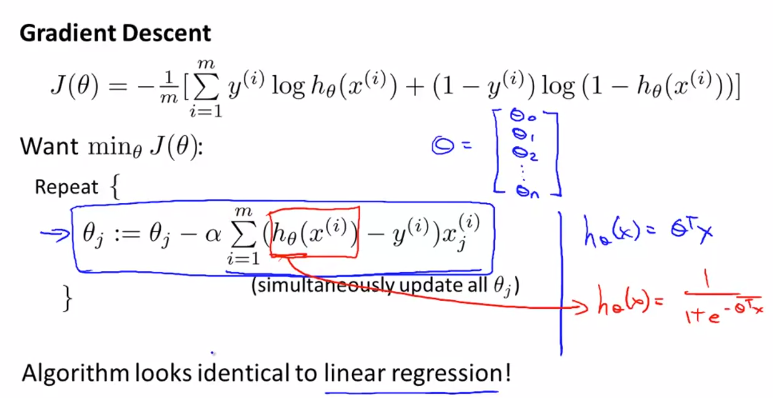
1. Simplified cost function and gradient descent:
   1. Combining the cost function equation as shown below:



* 1. This particular cost function has been selected from a statistical perspective of Maximum Likelihood Estimation which helps in selection theta for different models and is convex

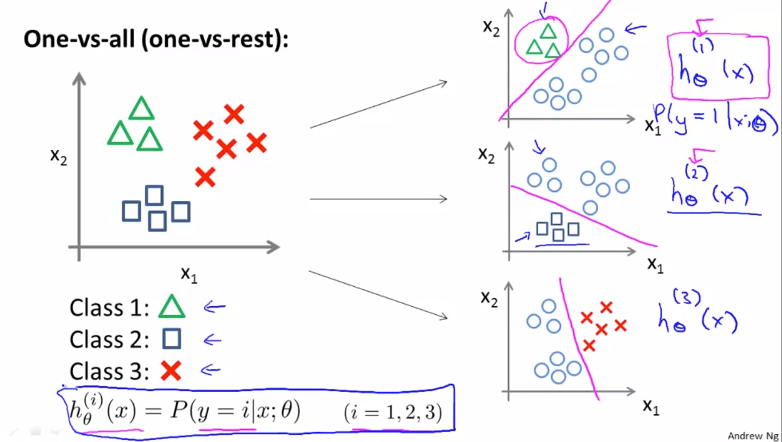


* 1. Minimization of J(theta) will help us find optimum values of theta to develop the hypothesis and it will be done by gradient descent as shown. The derivative of J when put into the theta update formula looks exactly like linear regression, the difference between the two is the hypothesis function

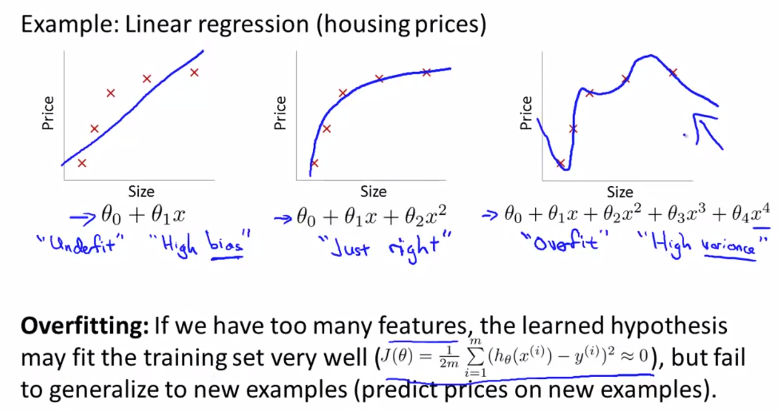


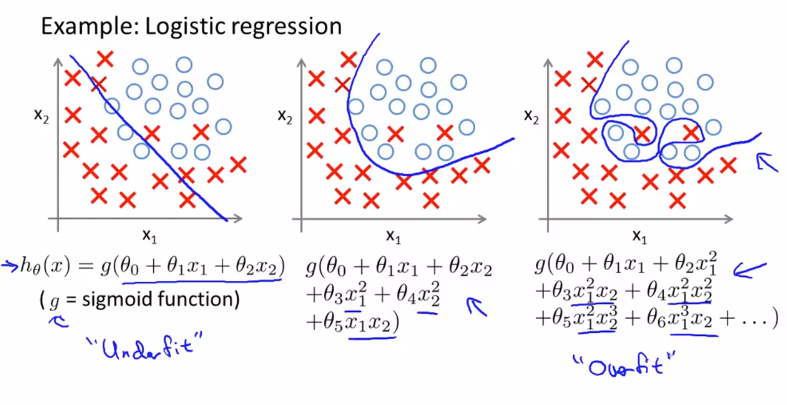
* 1. For proper convergence, plot J as a function of number of iterations (like in linear regression) and check if it is decreasing or not.
  2. We can apply feature scaling like in linear regression to make convergence faster

1. Advanced Optimization:
   1. The concept of optimization of parameters involves computation of cost function and its derivative
   2. Some sophisticated algorithms (other than gradient descent) are: Conjugate gradient, BFGS and L-BFGS. They are much complex (disadvantage) but we do not need to pick learning rate manually and also, they are much faster than gradient descent (advantage)
   3. For practical implementation:
      1. Define your own cost function which takes input as theta vector and outputs value of J and gradient for each theta (no of parameters (theta) X 1 vector)
      2. Pass them to optimset function, fminunc for their optimal values
2. Multiclass classification: One v/s all algorithm
   1. If you have N classes, divide them into N different binary classification problems
   2. Then, train a logistic regression model and make a decision boundary for one particular class (v/s all)
   3. Eventually we will have N different equations for h(x) for each of the class
   4. While making prediction we will assign the class corresponding to the maximum value of h(theta)



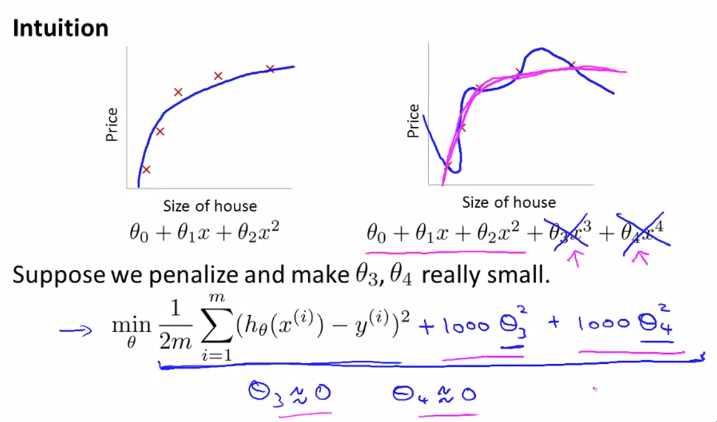
1. Overfitting:
   1. Problem is overfitting, can be solved/ ameliorated by regularization
   2. Underfitting means that the hypothesis does not fit the training very well and has high bias
   3. Overfitting means that the hypothesis has high variance, variance is too much, not enough data to fit





* 1. Addressing overfitting:
     1. Reduce number of features
        1. Manually remove features
        2. Model selection algorithm
     2. Regularization
        1. Keep all features but reduce magnitude/value of parameter theta

1. Regularization:
   1. We achieve it by making the higher order thetas very small close to zero so as to not have their effect

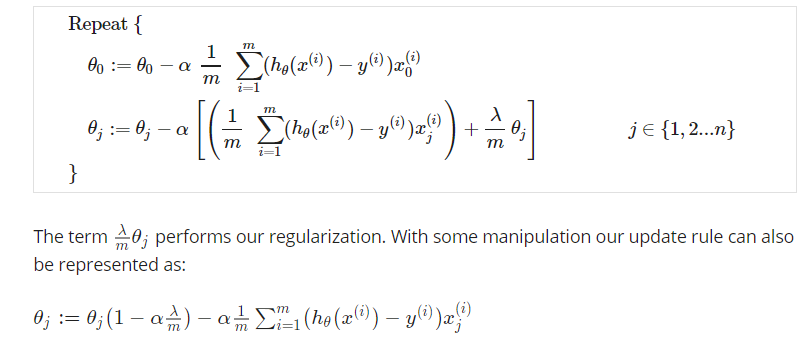


* 1. Idea – having smaller values of theta will result in developing simpler hypothesis and thus smoother
  2. We add a regularization term to the cost function equation
  3. Lambda does the job of keeping the parameters small 🡪 simple hypothesis 🡪 avoid overfitting

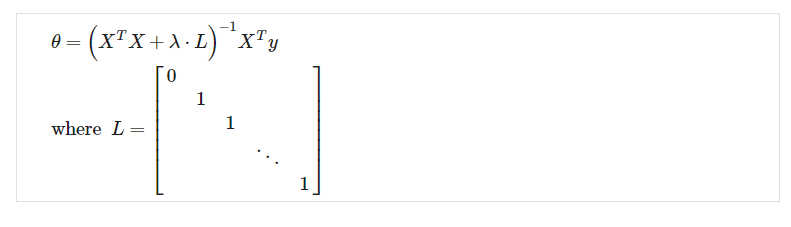


* 1. Very large lambda will underfit the hypothesis because of very high bias
  2. Therefore, the choice of lambda (regularization parameter) is crucial

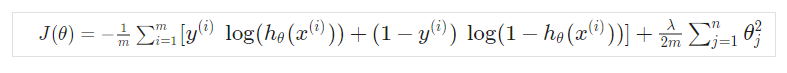
1. Regularized linear regression:
   1. Using new definition of J(theta), we will find a new derivative term to find theta update equations

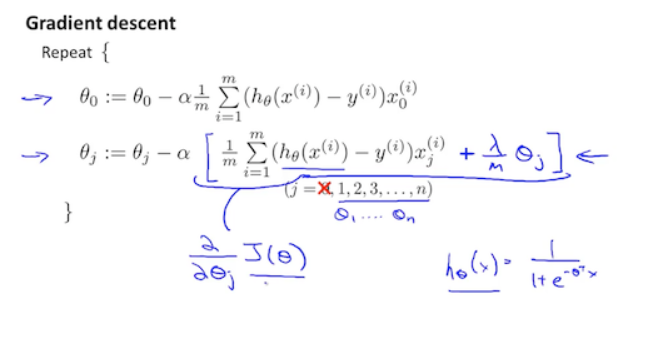


* 1. Modified normal equation:



1. Regularized Logistic Regression:
   1. Very high order polynomials added to the hypothesis can cause overfitting in logistic regression



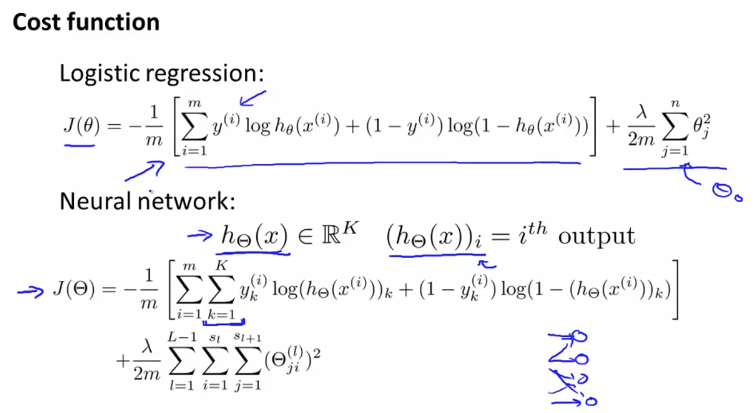


**Week 4:**

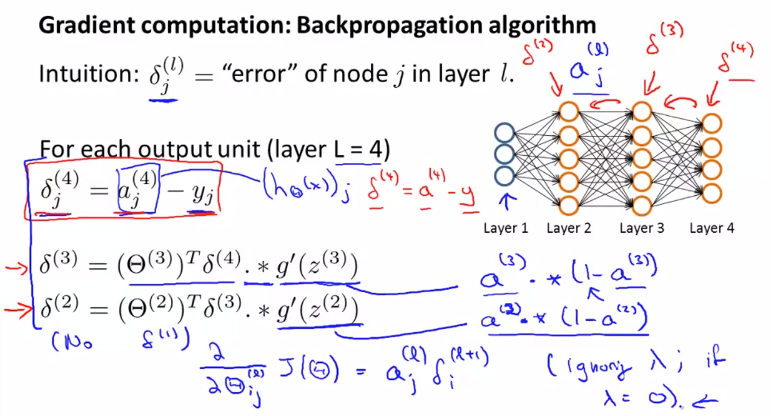
1. Non-linear hypothesis – Neural Network:
   1. One of the ways to have non-linearity in classification is to have higher order polynomials in logistic regression. But for large number of features, including higher order polynomials would be unfeasible eventually making the model overfit and also would be computationally expensive
   2. For sophisticated applications like CV, linear hypothesis can not be used thus the need for NN
2. Neuron and the Brain:
   1. Most algorithms mimic something, therefore why not go for the best and mimic the human brain
   2. Because of the neuro-rewiring experiments, it is concluded that there can be one algorithm which can work well for a large range of problems
   3. Examples (plug in any sensor and allow brain to learn):
      1. Seeing with your tongue
      2. Human echolocation
      3. Haptic belt: direction sense
      4. Implanting a third eye
3. Neural Network representation:
   1. Dendrites are inputs, axon is output, nucleus does computation on input
   2. In NN terminology, theta which are parameters of hypothesis are called weights
   3. The first layer is input layer, last layer is output layer and middle are hidden layers
   4. Forward propagation is used to compute output of NN (use vectorized method for faster implement)
   5. Architecture of NN is the way in which they are connected
4. Examples:
   1. XOR/ XNOR problem (non-linear classification problem)
   2. AND/ OR/ NOT are linear function and therefore can be realized by 1 neuron (or single layer NN)
5. Multiclass classification:

**Week 5:**

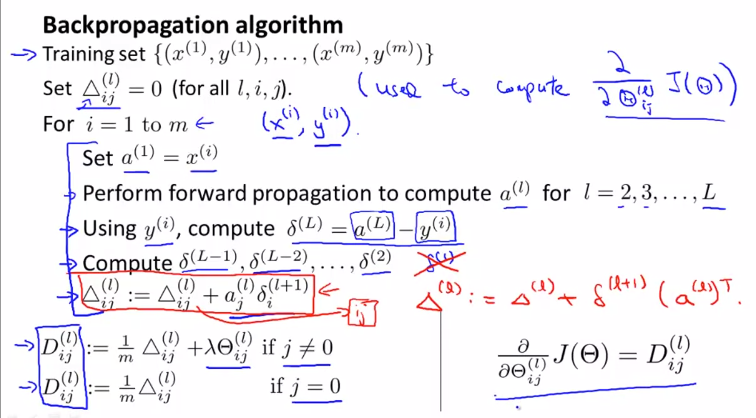
1. Cost Function for NN:
   1. L denotes total number of layers and s denotes number of units in each layer
   2. Cost function will be generalized version of regularized logistic regression



1. Backpropagation:
   1. Delta term calculates error at all nodes in each layer



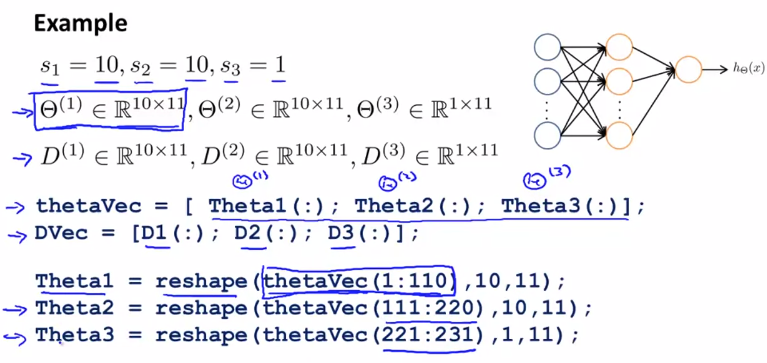
* 1. Algorithm:



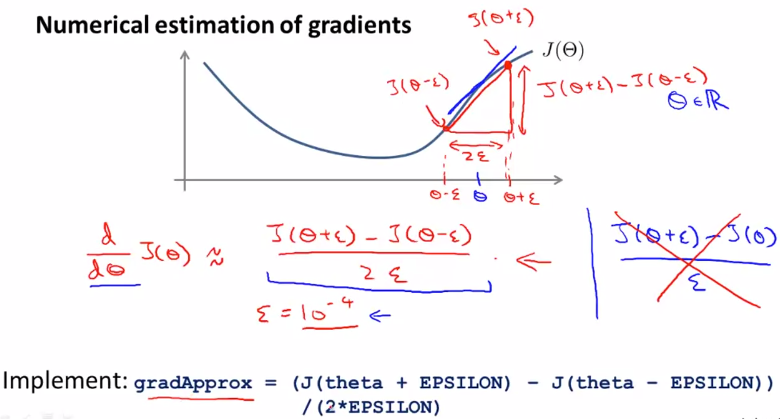
* 1. Back Propagation intuition:
     1. For intuition, the delta terms can be calculated similar to the FP by multiplying weights and output to generate corresponding delta at input side.



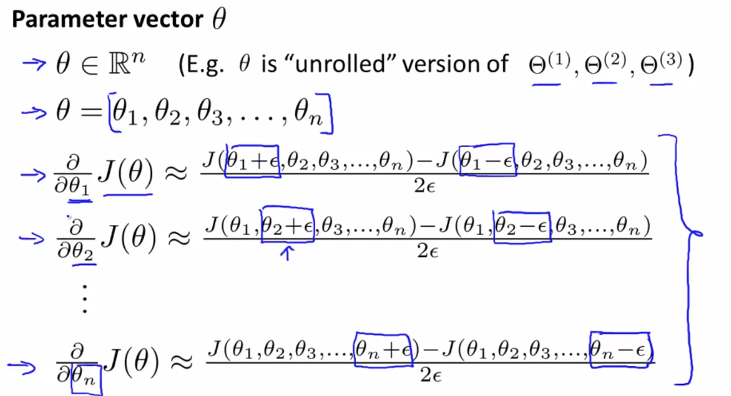
* 1. Unrolling parameters:
     1. Earlier we passed theta vectors to optimization functions like fminunc, now our parameters are matrices
     2. Method to make matrix to vector and vice versa so that it is compatible with optimization function



* 1. Gradient Checking:
     1. Gradient checking helps in accurate implementation of forward and back prop for complicated neural networks
     2. Method to compute approximate gradient is as shown



* + 1. We can use this method of numeric approximation can be used to calculate partial derivative of cost function with respect to all thetas as shown. We can compare the actual gradient of back propagation and these numerically computed ones and must be almost equal.



* 1. Random Initialization:
     1. We can’t initialize thetas as zeros for NN because then all neurons of hidden layers will act as identical
     2. To break this symmetry, we implement random initialization
  2. Steps:
     1. Chose architecture (no layers, i/p, o/p, etc)
        1. No of input = feature dimensions
        2. No of output = no of classes in multiclass classification
        3. Hidden = take 1 (common), have same no. of neurons in hidden (usually)
     2. Randomly initialize weights
     3. Implement forward propagation to get h for any x
     4. Implement cost function to calculate J
     5. Implement backpropagation to compute partial derivation (iterate over examples)
        1. Get activation and delta terms
        2. Get del and partial derivative terms (with regularization)
     6. Use gradient checking to compare backprop and approx. numeric estimation method
     7. Use gradient descent/ optimization method along with backprop to minimize J

1. Example of NN:
   1. Autonomous driving
   2. Training car by driving it
   3. Every 2 sec footage is used as training image and used as input for NN
   4. After 2min of training, the algo learns accurately to mimic driver
   5. To validate, 12 image/ sec sampling is done and multiple NN are used to give decisions and most confident network results are used to control the vehicle