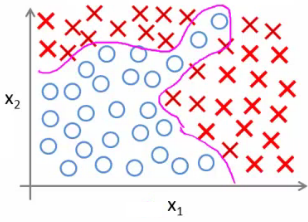
|  |  |
| --- | --- |
| |  | | --- | | Bayesian Statistics  In the [What is Science](https://sites.google.com/site/skepticalmedicine/what-is-science) section, we learned a little bit about Bayesian methods for assessing the probability of a hypothesis.  We saw that the probability of a hypothesis in light of new evidence is proportional to its prior probability times the strength of the new evidence.  Bayesian statistics allow one to make an estimate about the likelihood of a claim and then update these estimates as new evidence becomes available.  In non-mathematical terms, Baye's Theorem can be expressed like this:  The probability of a claim (given the currently available data) is proportional to its probability before the new data multiplied by the strength of the new data. Thus, we put new data into perspective with our prior understanding.  This form of statistical methodology stands in contrast to traditional statistics called [frequentist probability](http://en.wikipedia.org/wiki/Frequentism), whereby the probability of an event is determined only after counting its frequency after collecting a body of data. In Frequentism, conclusions tend to be fairly binary (we either reject the null hypothesis in favor of the pet hypothesis, or we do not). Statistical significance is said to be met if the [p value](https://sites.google.com/site/skepticalmedicine/statistics-and-risk#TOC-p-Value) is below a certain limit, usually set as p < 0.05).  Current medical studies tend to utilize frequentist probablility. As discussed in the [EBM vs. SBM](https://sites.google.com/site/skepticalmedicine/ebm-vs-sbm) section, such statistics only count frequencies of events to determine 'significance'. Plausibility of the hypothesis in question does not really play much of a role.  With Bayesian statistics, plausibility determines the starting point for determining a claims probability. Thus, we have the SBM call for using Bayes especially when dealing with highly implausible claims.  History  [https://sites.google.com/site/skepticalmedicine/_/rsrc/1488148702859/bayesian-methodology/Thomas_Bayes.png?height=300&width=400](https://sites.google.com/site/skepticalmedicine/bayesian-methodology/Thomas_Bayes.png?attredirects=0)  History was sluggish and frankly unkind to Baye's Theorem. Sharon Mcgrayne's book, [*The Theory That Would Not Die*](http://www.amazon.com/The-Theory-That-Would-Not/dp/0300169698/ref=sr_1_1?ie=UTF8&qid=1337478692&sr=8-1), describes a remarkable evolution of such a simple idea.  Thomas Bayes (1701 - 1761) was a British mathematician and a minister. He is credited for describing a process for adjusting and updating the likelihood of an event based on data -- as the data is generated. Remember, in frequentist statistics, probabilities are determined only after all of the data is collected. He did not publish this as a mathematical theorem, but rather described the basic idea in a posthumous publication, "[*An Essay towards solving a Problem in the Doctrine of Chances*](http://en.wikipedia.org/wiki/An_Essay_towards_solving_a_Problem_in_the_Doctrine_of_Chances#Outline)*".*  In 1814, the French mathematician, astronomer and statistician Pierre-Simon Laplace published *[Essai philosophique sur les probabilités](http://www.cambridge.org/gb/knowledge/isbn/item2708892/?site_locale=en_GB" \t "_blank).*Here, he described mathematically the same idea that Bayes had described earlier.  Until recently, statisticians have shunned Bayesian methodology due to the fact that it begins with uncertainty (and corrects with each new observation toward the likely truth). In her book, Mcgrayne describes those that discovered the power of Bayesian statistics over the twentieth century. Perhaps the most notable was [Alan Turing](http://en.wikipedia.org/wiki/Turing). During World War II, Turning used Bayesian methodology and early computers to decipher Nazi u-boat codes. Historians acknowledge that Turing's work likely helped turn the tide of the war in the Allies' favor. His work and methods, however, were classified by Winston Churchill after the war. The world and statisticians would not know of the success until decades later.  In the era of modern computers, Bayesian methodology has been rediscovered and updated. Prior probabilities are now calculated from our large body of prior scientific knowledge, specialized expertise or by recently developed statistical techniques. However, there is still some inherent subjectivity to the starting point.  Prior Probability  Ideas and theories start out with a baseline probability of being true based on previously established knowledge. Bayesians call this the prior probability, or just "the prior". The prior can be somewhat subjective, as it is sometimes an unknown quantity based on limited data and the biases of the person assigning the value. It is a measure of belief that the claim is true. Bayes' allows multiple hypotheses to be combined and ranked by probability. The prior is usually written as P(H). Each hypothesis for a given problem is distinguished with a number (H1, H2, etc). The "[null hypothesis](https://sites.google.com/site/skepticalmedicine/what-is-science#TOC-Absence-of-Evidence-is-Not-Necessarily-Evidence-of-Absence)" is notated as Ho.  Skeptics often employ [Occam's Razor](https://sites.google.com/site/skepticalmedicine/skeptic#TOC-Occam-s-Razor-:-) when assigning relative values to priors. Before new data is collected or observed, we can rank competing hypotheses according to the number of assumptions required by each one. Hypotheses that only incorporate established knowledge are given higher probabilities than those that either require us to accept unknowns or to reject established knowledge.  For instance, explanations that suggest a violation of physics (eg. [homeopathy](https://sites.google.com/site/skepticalmedicine/pseudoscience-in-health-care/homeopathy)) or the presence of an unknown vital force (eg. [acupuncture](https://sites.google.com/site/skepticalmedicine/pseudoscience-in-health-care/acupuncture), [reiki](https://sites.google.com/site/skepticalmedicine/pseudoscience-in-health-care/reiki-and-therapeutic-touch), [chiropractic](https://sites.google.com/site/skepticalmedicine/pseudoscience-in-health-care/chiropractic)) have more built-in assumptions than explanations that only require acceptance of known processes (eg. phychology, statistical trends, natural history of disease). Plausible explanations that incorporate the fewest unknowns and assumptions are ranked with higher priors than those with such assumptions. The more assumptions made lead to lower the ranking of prior probability. Also, the more violations of established knowledge, the lower the prior probability.  When determining the prior probability for events that have a known background rate, we may use this background rate (or "base rate") as our prior. When determining the likelihood of a disease in a patient, we can begin with the base rate of the disease in the community in which the patient is a member. If a patient belongs to a community that has a base rate of HIV infection of 1%, then we use 1% as our prior probability when determining the likelihood of the patient's status (even before gathering any information about the patient).  It must be acknowledged that many will find explanations with lower priors to be appealing due to biases, beliefs and world views. We may find comfort in explanations that appeal to unseen forces, new-age ideas and dogma. Appealing as they may be, priors should be assigned with Occam's Razor in mind.  New Data, Priors and Posterior Probability  The next step in the Bayesian approach involves the consideration of new data obtained through experimentation or observation. We must ask,  'what is the probability of obtaining a set of data if the hypothesis were true?" In other words, how much does the data support the given hypothesis. This is commonly written as P(D/H), or the probability of the data given the hypothesis.  Here we assign a probability that the new data could be accounted for by the hypothesis in question. Studies are generally designed to test a given hypothesis. It is here that we must account for the strength of the new evidence based on the quality of the study and the magnitude of the results.  Study quality was examined in the section on [Scientific Studies in Medicine](https://sites.google.com/site/skepticalmedicine/types-of-scientific-studies). Larger studies are typically more powerful than smaller ones. Randomized, double-blinded controlled trials typically are better than observational studies, etc. Results that are unambiguously positive lend more support to a hypothesis than those that barely reach statistical significance.  In Bayes', we update a the probability of a hypothesis by multiplying the prior probability with the strength of the new data. The new, updated probability is called the posterior probability, or just 'the posterior'. However, there is a denominator in the calculation. This is the sum total of probabilities of all possible relevant hypotheses. This should be considered to be a constant that, if all hypothesis are truly accounted for are summed, should equal 1. (see below).  The posterior then becomes the new prior and the process may repeat. In other words, the probability of an idea is updated in light of new information and thus should become closer to the truth.  The posterior may be higher or lower than the prior and closer to the truth (one may consider "the Truth with a capital T" to be approachable, but ultimately unknowable.)  One can see how this method complies with both Occam's Razor and the [Extraordinary Claims rule](https://sites.google.com/site/skepticalmedicine/skeptic#TOC-Extraordinary-Claims-Require-Extraordinary-Evidence-:-), which states, "extraordinary claims require extraordinary evidence." Occam's Razor helps to define just how extraordinary a claim is. Bayes' rule simply helps us to rank hypotheses with these concepts.  Bayes' in Everyday Medicine  Let's consider how we can put Bayes' Theorem to practical use in everyday medical decision making.  Let's say that a disease is known to have a prevalence of 1 out of every 1000 people in a given population. This gives us our "prior". (Note - this number is obtained through frequentist statistics. Sometimes the prior probability can be known, other times it comes from an educated guess.)  Let's also say that we have a test for the disease that is very sensitive, but has an inherent false positive rate of 10%.  Now, we need to use the test on 1000 individuals to determine if they have the disease.  The pre-test probability that any one of the individuals has the disease is 1/1000, or .001.  When we test 1000 people, we may find (on average) the 1 person in 1000 that has the disease. But we also find 100 people who test positive but do not have the disease (10% false positive x 1000 people = 100 false positives).  So, we may expect on average to find 101 people that test positive out of the 1000 people tested (1 true positive and 100 false positives). Therefore, of these 101 suspects, each of them only have a 1/101 chance of having the disease. The denominator, 101, represents the sum of our 2 possible hypotheses (101 positive tests = true positives and false positives). The perspective that Bayesian analysis gives us allows us to make better sense of data.  Without understanding that we need to combine the new data with the prior probability, a clinician may tend to raise an unwarranted amount of fear in these 101 patients even though it is likely that only 1 of them has the disease. Without the Bayesian perspective, these 101 people will likely all become convinced that they have the disease.  This exercise demonstrates how we gain a much clearer perspective about test results by combining prior knowledge with new data and updating our position. It demonstrates how we arrive at the positive predictive value of a test. We can similarly determine the [predictive value of a test](https://sites.google.com/site/skepticalmedicine/statistics-and-risk#TOC-Positive-Predictive-Value-).  Modern Formulation  Bayes' Theorem is formally notated as:  **P(H/D) = P(D/H) x P(H)**  **P(D)**  where...  **P(H/D)** is the probability of the hypothesis (H) given the data (D),  **P(D/H)** is the probability of the data (D) given the hypothesis (H),  **P(H)** is the probability of the Hypothesis prior to the new data (also called the "prior probability" or just the "prior"), and  **P(D)** is the probability of obtaining the data (D).  The  P(D) is the sum of all of the probabilities of obtaining the data for each hypothesis. In hypothesis testing, it can be expressed as the sum of the probabilities of obtaining the data (D) given the hypothesis (H) and the null hypothesis (Ho). This is notated as:  **P(D) = P(D/H) x P(H)  +  P(D/Ho) x P(Ho)**  So, we can then write Bayes' Theorem as:  **[https://sites.google.com/site/skepticalmedicine/_/rsrc/1472858148662/bayesian-methodology/Bayes%27%20calc%203.png](https://sites.google.com/site/skepticalmedicine/bayesian-methodology/Bayes%27%20calc%203.png?attredirects=0)**  Applying Bayesian Methods  Putting these last parts together, we can see some of the true power of Baye's Theorem. Let's put Bayes' in terms of commonly used statistical values ([p value](https://sites.google.com/site/skepticalmedicine/statistics-and-risk#TOC-p-Value), [prevalence](https://sites.google.com/site/skepticalmedicine/statistics-and-risk#TOC-Prevalence), [sensitivity](https://sites.google.com/site/skepticalmedicine/statistics-and-risk#TOC-Sensitivity) and [specificity](https://sites.google.com/site/skepticalmedicine/statistics-and-risk#TOC-Specificity)).  In frequentist statistics, the highly touted p-value is really the probability of obtaining the data if the null hypothesis were true. In other words,  p-value = P(D/Ho).  Also,  we can express the p-value as the probability of obtaining a false negative result. In other words,  p-value = (1-specificity).  The prevalence of a condition is its frequency within a population. Therefore, the probability of it not being present is therefore 1 - prevalence.  This essentially is the same as saying the probability of the null hypothesis. In other words,  (1 - Prevalence) = P(Ho).  By combining our previous knowledge about a condition (prevalence), and the frequentist parameters of our tests (sensitivity, specificity), we can do some Bayesian analysis, just as we did [in the exercise above](https://sites.google.com/site/skepticalmedicine/bayesian-methodology#TOC-Bayes-in-Everyday-Medicine).  [https://sites.google.com/site/skepticalmedicine/_/rsrc/1472858146192/bayesian-methodology/Bayes%27%20calc%202.png](https://sites.google.com/site/skepticalmedicine/bayesian-methodology/Bayes%27%20calc%202.png?attredirects=0)  where PPV is the "[Positive Predictive Value](https://sites.google.com/site/skepticalmedicine/statistics-and-risk#TOC-Positive-Predictive-Value-)".  Knowledgeable clinicians can use the "Positive Predictive Value" and "Negative Predictive Value" of tests and procedures to make better decisions.  Bayes Factor  This is the ratio of the probabilities of the data in light of the hypothesis (H) and the null hypothesis (Ho).  Bayes Factor  =  P(D/H)                               P(D/Ho)  [https://sites.google.com/site/skepticalmedicine/_/rsrc/1472858147635/bayesian-methodology/Bayes%27%20calc%204.png](https://sites.google.com/site/skepticalmedicine/bayesian-methodology/Bayes%27%20calc%204.png?attredirects=0)  This is often used synonymously with the **Likelihood Ratio**.  The Bayes Factor tells us about the data and how useful it is given the hypothesis being tested.  If the Bayes Factor is 1, then the data would be useless. In other words, data obtained would be equally consistent with H and Ho.  If Bayes Factor is 1 to 2, then the data would be slightly interesting, but likely does not mean a big difference between H and Ho.  If Bayes Factor is 2 to 5, then the data would represent a mild difference between H and Ho.  If Bayes Factor is 5 to 10, then the data would represent a moderate difference between H and Ho.  If Bayes Factor is over 10, then the data would represent a large difference between H and Ho.  Bayesians vs Frequentists (SBM vs EBM Revisited)  Medical science is currently evaluated mainly with frequentist statistics. Individual studies and meta-studies are look evaluate data from the prospective of the p-value. To a Bayesian, the p-value is really the probability of the data given the null hypothesis, or P(D/Ho). If the p-value is less than 0.05, we traditionally say the the data is "statistically significant" and we then reject the null hypothesis.  Frequentists have complained that Bayes' uses subjective values for initial priors (although, modern techniques can be used to reduce the subjectivity). However, the tradtitional cut off for statistical significance of p < 0.05 is also somewhat subjective. Some studies produce results with a p-value of 0.04. Are these results more significant than those with a p-value of 0.06? Frequentist statistics can lead to a binary mode of thinking. Significant or not significant. Reject or accept.  Typically, studies with a p-value close to 0.05 are labeled 'inconclusive' and more studies are called for. The modern world of Evidence Based Medicine tends to operate on frequentist statistics. This is generally fine when analyzing claims with reasonable plausibility.  When analyzing low probability claims with respect only to the p-value, we may miss the forest through the trees.  In Bayesian statistics, the p-value is only one part of the denominator. The "prior" is in the numerator and holds a major role in the calculation of the posterior. Therefore, when evaluating claims with low prior probabilities ("extraordinary claims"), Bayesian methodology automatically puts such claims in perspective. Thus, "Science Based Medicine" (SBM) takes a Bayesian approach to extraordinary claims. Bayesians and SBM proponents tend to talk in terms of low and high probability, rather than rejecting or accepting a hypothesis based on the p-value (see [EBM vs SBM](https://sites.google.com/site/skepticalmedicine/ebm-vs-sbm)).  More Bayes in Everyday Life -- "21"  Many experienced gamblers employ Bayes' Theorem everyday in casinos. Blackjack, or "21", is a game that lends itself perfectly to Bayesian methodology. Unlike roulette, craps or slot machines, 21 is not as subject to the "[gambler's fallacy](https://sites.google.com/site/skepticalmedicine/logical-fallacies#TOC-Gambler-s-Fallacy-)" in that the odds of the next hand do not reset each time. There are a finite number of cards, and a known number of "high" and "low" cards. The cards are not reshuffled with each hand and not until all of the cards have been dealt (in casinos, a card shoe usually contains 4 decks). Each player only plays against the dealer, who has a slight, inherent advantage. It is generally known that low cards work to the dealer's advantage, whereas high cards work to the player's.  Smart gamblers have developed systems of '[card counting](http://en.wikipedia.org/wiki/Card_counting)'. They assign a +1 value to low cards (2's through 6's), 0 value to middle cards (7's, 8's and 9's) and -1 value to high cards (10's through Aces). When each card is dealt and ultimately revealed, the card counter makes a mental addition or subtraction (+1 or -1). With each hand, the card counter calculates a new tally based on the cards already dealt. Tallies that are strongly positive mean that there are more high cards left in the shoe than low cards. This gives the player the advantage and a signal to start placing bigger bets. Values that are strongly negative mean that there are more low cards left, which gives the dealer the advantage.  Prior to any cards being dealt, the tally is 0. This is akin to the initial "prior" in a Bayesian analysis. The card counter then updates this prior probability in light of new data with each hand (tally from the cards that have been dealt). This is all done to determine the probabilities of 3 competing hypotheses (H1 = win, H2 = lose, H3 = break even). Without even realizing it, professional Blackjack players are natural Bayesians.  The 2008 movie, "21" was loosely based on a group of MIT students who used this technique to win big in Las Vegas (they were eventually banned from the casinos). One of the real-life players discussed the technique to promote the movie, as seen [here](http://www.youtube.com/watch?v=J30WuLODVoM&feature=endscreen&NR=1).  Conclusion  By combining the Bayes Factor with the Prior Probability of a hypothesis, we get  a very clear picture of just how probable a hypothesis is in light of new data.  In this way, the probability of an idea being true may always be adjusted in light of its prior probability and new data. We may have a better understanding over time, in light of multiple studies, of the truth of an idea.  Currently, we tend to judge the significance of a study on the p-value. As more studies are done, each study basically starts from scratch and comes up with new p-values. Each study may have biases and methodological flaws. Commonly, multiple studies - with their flaws - are combined into a meta-analysis to make one big study with its own p-value. This works fairly well for ideas that have high prior probabilities (in other words, the ideas are plausible in light of prior scientific knowledge). However, skeptical doctors are also concerned with ideas and claims that have low prior probabilities.  For ideas with low prior probabilities, it may be more useful to use a  Bayesian approach (using the Bayes Factor and prior probability to determine a new probability of an idea in light of new data). Instead of combining data from multiple studies into a meta-analysis, each study should add to the probability. The Bayes Factor of the study would determine the value of the data produced by the study. Poor studies would have low Bayes Factors, and contribute less to the big picture than would good studies.  John Byrne, M.D.  References and Links  "The Theory That Would Not Die: How Bayes' Rule Cracked the ..." 2012.  <<http://www.amazon.com/dp/0300188226>>  "An Essay towards solving a Problem in the Doctrine of Chances ..." 2011.  <<http://en.wikipedia.org/wiki/An_Essay_towards_solving_a_Problem_in_the_Doctrine_of_Chances>>  "Essai philosophique sur les probabilités - Cambridge Books Online ..." 2011.  <<http://ebooks.cambridge.org/ebook.jsf?bid=CBO9780511693182>>  "Alan Turing - Wikipedia, the free encyclopedia." 2003.  <<http://en.wikipedia.org/wiki/Alan_Turing>>  "Science-Based Medicine » Prior Probability: The Dirty Little Secret of ..." 2011.  <<http://www.sciencebasedmedicine.org/index.php/prior-probability-the-dirty-little-secret-of-evidence-based-alternative-medicine-2/>> | |

[Sign in](https://www.google.com/a/UniversalLogin?continue=https://sites.google.com/site/skepticalmedicine/bayesian-methodology&service=jotspot)|[Report Abuse](https://sites.google.com/site/skepticalmedicine/system/app/pages/reportAbuse)|[Print Page](javascript:;)|Powered By [**Google Sites**](http://sites.google.com/)

**Neural networks - Overview and summary**

**Why do we need neural networks?**

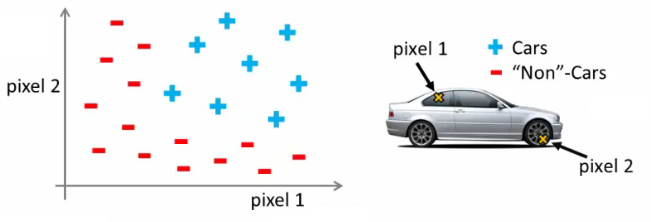
* Say we have a complex supervised learning classification problem
  + Can use logistic regression with many polynomial terms
  + Works well when you have 1-2 features
  + If you have 100 features



* e.g. our housing example
  + 100 house features, predict odds of a house being sold in the next 6 months
  + Here, if you included all the quadratic terms (second order)
    - There are lots of them (x12 ,x1x2, x1x4 ..., x1x100)
    - For the case of n = 100, you have about 5000 features
    - Number of features grows O(n2)
    - This would be computationally expensive to work with as a feature set
* A way around this to only include a subset of features
  + However, if you don't have enough features, often a model won't let you fit a complex dataset
* If you include the cubic terms
  + e.g. (x12x2, x1x2x3, x1x4x23 etc)
  + There are even more features grows O(n3)
  + About 170 000 features for n = 100
* Not a good way to build classifiers when n is large

**Example: Problems where n is large - computer vision**

* Computer vision sees a matrix of pixel intensity values
  + Look at matrix - explain what those numbers represent
* To build a car detector
  + Build a training set of
    - Not cars
    - Cars
  + Then test against a car
* How can we do this
  + Plot two pixels (two pixel locations)
  + Plot car or not car on the graph



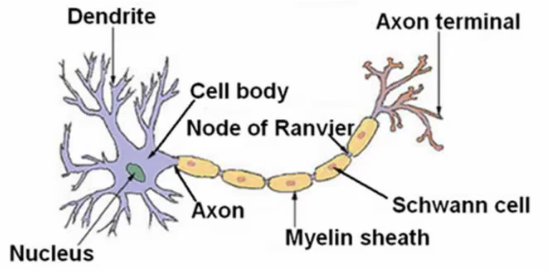
* Need a non-linear hypothesis to separate the classes
* Feature space
  + If we used 50 x 50 pixels --> 2500 pixels, so n = 2500
  + If RGB then 7500
  + If 100 x 100 RB then --> 50 000 000 features
* Too big - wayyy too big
  + So - simple logistic regression here is not appropriate for large complex systems
  + Neural networks are much better for a complex nonlinear hypothesis even when feature space is huge

**Neurons and the brain**

* **Neural networks** (**NNs**) were originally motivated by looking at machines which replicate the brain's functionality
  + Looked at here as a machine learning technique
* Origins
  + To build learning systems, why not mimic the brain?
  + Used a lot in the 80s and 90s
  + Popularity diminished in late 90s
  + Recent major resurgence
    - NNs are computationally expensive, so only recently large scale neural networks became computationally feasible
* Brain
  + Does loads of crazy things
    - Hypothesis is that the brain has a single learning algorithm
  + Evidence for hypothesis
    - Auditory cortex --> takes sound signals
      * If you cut the wiring from the ear to the auditory cortex
      * Re-route optic nerve to the auditory cortex
      * Auditory cortex learns to see
    - Somatosensory context (touch processing)
      * If you rewrite optic nerve to somatosensory cortex then it learns to see
  + With different tissue learning to see, maybe they all learn in the same way
    - Brain learns by itself how to learn
* Other examples
  + Seeing with your tongue
    - Brainport
      * Grayscale camera on head
      * Run wire to array of electrodes on tongue
      * Pulses onto tongue represent image signal
      * Lets people see with their tongue
  + Human echolocation
    - Blind people being trained in schools to interpret sound and echo
    - Lets them move around
  + Haptic belt direction sense
    - Belt which buzzes towards north
    - Gives you a sense of direction
* Brain can process and learn from data from any source

**Model representation 1**

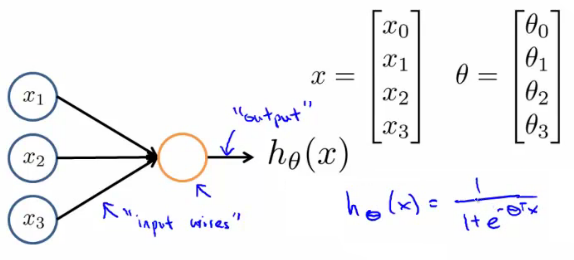
* How do we represent neural networks (NNs)?
  + Neural networks were developed as a way to simulate networks of neurones
* What does a neurone look like



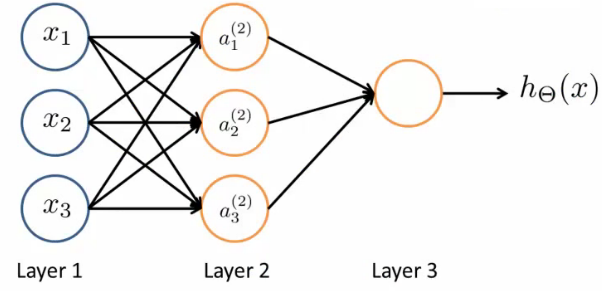
* Three things to notice
  + Cell body
  + Number of input wires (dendrites)
  + Output wire (axon)
* Simple level
  + Neurone gets one or more inputs through dendrites
  + Does processing
  + Sends output down axon
* Neurons communicate through electric spikes
  + Pulse of electricity via axon to another neurone

**Artificial neural network - representation of a neurone**

* In an artificial neural network, a neurone is a logistic unit
  + Feed input via input wires
  + Logistic unit does computation
  + Sends output down output wires
* That logistic computation is just like our previous logistic regression hypothesis calculation



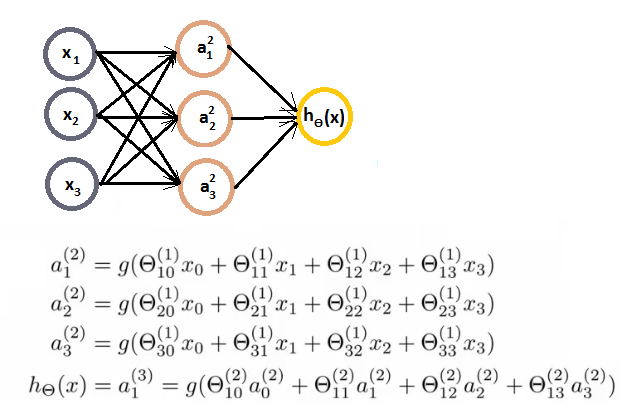
* Very simple model of a neuron's computation
  + Often good to include an x0 input - the **bias unit**
    - This is equal to 1
* This is an artificial neurone with a sigmoid (logistic) activation function
  + Ɵ vector may also be called the **weights**of a model
* The above diagram is a single neurone
  + Below we have a group of neurones strung together

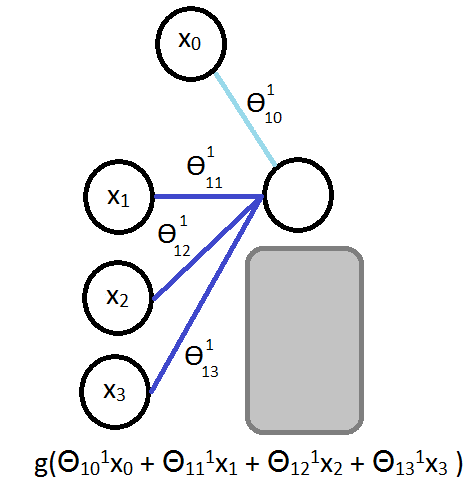


* Here, input is x1, x2and x3
  + We could also call input activation on the first layer - i.e. (a11, a21and a31)
  + Three neurones in layer 2 (a12, a22and a32)
  + Final fourth neurone which produces the output
    - Which again we \*could\* call a13
* First layer is the **input layer**
* Final layer is the **output layer** - produces value computed by a hypothesis
* Middle layer(s) are called the **hidden layers**
  + You don't observe the values processed in the hidden layer
  + Not a great name
  + Can have many hidden layers

**Neural networks - notation**

* **ai(j) - activation of unit *i*in layer *j***
  + So, a12 - is the **activation** of the 1st unit in the second layer
  + By activation, we mean the value which is computed and output by that node
* **Ɵ(j) - matrix of parameters controlling the function mapping from layer *j* to layer *j* + 1**
  + Parameters for controlling **mapping** from one layer to the next
  + If network has
    - sj units in layer *j* and
    - sj+1 units in layer j + 1
    - Then Ɵj will be of dimensions [sj+1 Xsj + 1]
      * Because
        + sj+1is equal to the number of units in layer (j + 1)
        + is equal to the number of units in layer j, plus an additional unit
  + Looking at the Ɵ matrix
    - Column length is the number of units in the following layer
    - Row length is the number of units in the current layer + 1 (because we have to map the bias unit)
    - So, if we had two layers - 101 and 21 units in each
      * Then Ɵj would be = [21 x 102]
* What are the computations which occur?
  + We have to calculate the activation for each node
  + That activation depends on
    - The input(s) to the node
    - The parameter associated with that node (from the Ɵ vector associated with that layer)
* Below we have an example of a network, with the associated calculations for the four nodes below



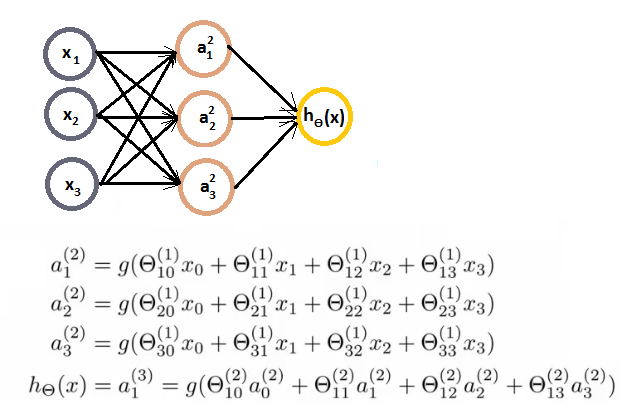
* As you can see
  + We calculate each of the layer-2 activations based on the input values with the bias term (which is equal to 1)
    - i.e. x0 to x3
  + We then calculate the final hypothesis (i.e. the single node in layer 3) using exactly the same logic, except in input is not x values, but the activation values from the preceding layer
* The activation value on each hidden unit (e.g. a12 ) is equal to the sigmoid function applied to the linear combination of inputs
  + Three input units
    - So Ɵ(1)is the matrix of parameters governing the mapping of the input units to hidden units
      * Ɵ(1) here is a [3 x 4] dimensional matrix
  + Three hidden units
    - Then Ɵ(2)is the matrix of parameters governing the mapping of the hidden layer to the output layer
      * Ɵ(2) here is a [1 x 4] dimensional matrix (i.e. a row vector)
  + One output unit
* Something conceptually important (that I hadn't really grasped the first time) is that
  + **Every input/activation goes to every node in following layer**
    - Which means each "layer transition" uses a matrix of parameters with the following significance
      * For the sake of consistency with later nomenclature, we're using j,i and l as our variables here (although later in this section we use j to show the layer we're on)
      * **Ɵ**jil
        + j (first of two subscript numbers)= ranges from 1 to the number of units in layer l+1
        + i (second of two subscript numbers) = ranges from 0 to the number of units in layer l
        + l is the layer you're moving FROM
      * This is perhaps more clearly shown in my slightly over the top example below
* 
* For example
  + Ɵ**131**= means
    - **1** - we're mapping to node 1 in layer l+1
    - **3** - we're mapping from node 3 in layer l
    - **1** - we're mapping from layer 1

**Model representation II**

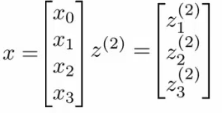
*Here we'll look at how to carry out the computation efficiently through a vectorized implementation. We'll also consider*

*why NNs are good and how we can use them to learn complex non-linear things*

* Below is our original problem from before
  + Sequence of steps to compute output of hypothesis are the equations below



* Define some additional terms
  + z12 = Ɵ101x0+ Ɵ111x1+ Ɵ121x2+ Ɵ131x3
  + Which means that
    - a12 = g(z12)
  + NB, superscript numbers are the layer associated
* Similarly, we define the others as
  + z22 and z32
  + These values are just a linear combination of the values
* If we look at the block we just redefined
  + We can vectorize the neural network computation
  + So lets define
    - x as the feature vector x
    - z2 as the vector of z values from the second layer



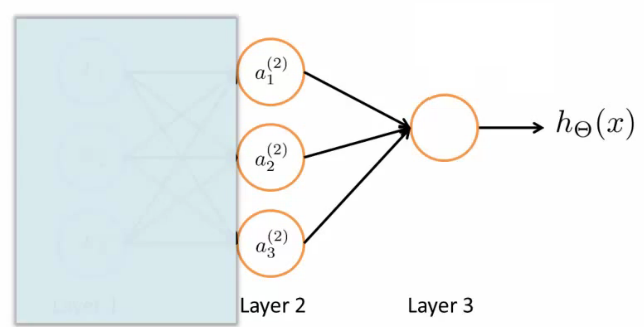
* z2 is a 3x1 vector
* We can vectorize the computation of the neural network as as follows in two steps
  + z2 = Ɵ(1)x
    - i.e. Ɵ(1) is the matrix defined above
    - x is the feature vector
  + a2 = g(z(2))
    - To be clear, z2 is a 3x1 vecor
    - a2 is also a 3x1 vector
    - g() applies the sigmoid (logistic) function element wise to each member of the z2 vector
* To make the notation with input layer make sense;
  + a1 = x
    - a1 is the activations in the input layer
    - Obviously the "activation" for the input layer is just the input!
  + So we define x as a1 for clarity
    - So
      * a1 is the vector of inputs
      * a2 is the vector of values calculated by the g(z2) function
* Having calculated then z2 vector, we need to calculate a02 for the final hypothesis calculation

http://www.holehouse.org/mlclass/08_Neural_Networks_Representation_files/Image%20%5b9%5d.png

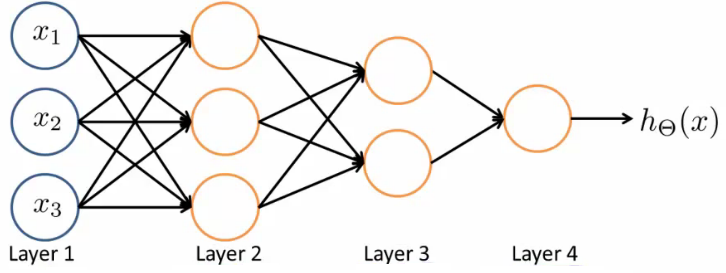
* To take care of the extra bias unit add a02 = 1
  + So add a02 to a2 making it a 4x1 vector
* So,
  + z3 = Ɵ2a2
    - This is the inner term of the above equation
  + hƟ(x) = a3 = g(z3)
* This process is also called **forward propagation**
  + Start off with activations of input unit
    - i.e. the x vector as input
  + Forward propagate and calculate the activation of each layer sequentially
  + This is a vectorized version of this implementation

**Neural networks learning its own features**

* Diagram below looks a lot like logistic regression



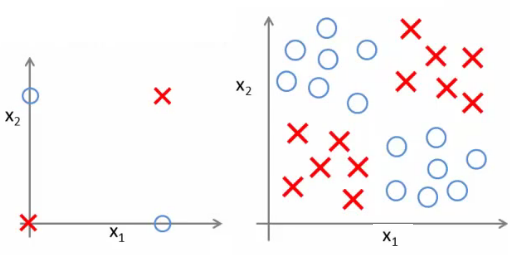
* Layer 3 is a logistic regression node
  + The hypothesis output = g(Ɵ102a02 + Ɵ112a12 + Ɵ122a22 + Ɵ132a32)
  + This is just logistic regression
    - The only difference is, instead of input a feature vector, the features are just values calculated by the hidden layer
* The features a12, a22, and a32 are calculated/learned - not original features
* So the mapping from layer 1 to layer 2 (i.e. the calculations which generate the a2 features) is determined by another set of parameters - Ɵ1
  + So instead of being constrained by the original input features, a neural network can learn its own features to feed into logistic regression
  + Depending on the Ɵ1 parameters you can learn some interesting things
    - Flexibility to learn whatever features it wants to feed into the final logistic regression calculation
      * So, if we compare this to previous logistic regression, you would have to calculate your own exciting features to define the best way to classify or describe something
      * Here, we're letting the hidden layers do that, so we feed the hidden layers our input values, and let them learn whatever gives the best final result to feed into the final output layer
* As well as the networks already seen, other architectures (topology) are possible
  + More/less nodes per layer
  + More layers
  + Once again, layer 2 has three hidden units, layer 3 has 2 hidden units by the time you get to the output layer you get very interesting non-linear hypothesis



* Some of the intuitions here are complicated and hard to understand
  + In the following lectures we're going to go though a detailed example to understand how to do non-linear analysis

**Neural network example - computing a complex, nonlinear function of the input**

* Non-linear classification: XOR/XNOR
  + x1, x2 are binary



* Example on the right shows a simplified version of the more complex problem we're dealing with (on the left)
* We want to learn a non-linear decision boundary to separate the positive and negative examples

y = x1 XOR x2

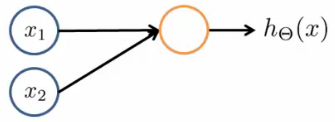
      x1 XNOR x2

Where XNOR = NOT (x1 XOR x2)

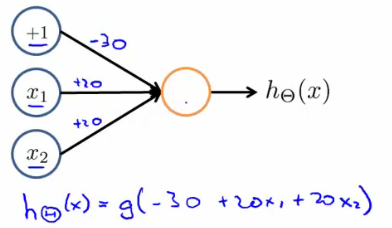
* Positive examples when both are true and both are false
  + Let's start with something a little more straight forward...
  + Don't worry about how we're determining the weights (Ɵ values) for now - just get a flavor of how NNs work

**Neural Network example 1: AND function**

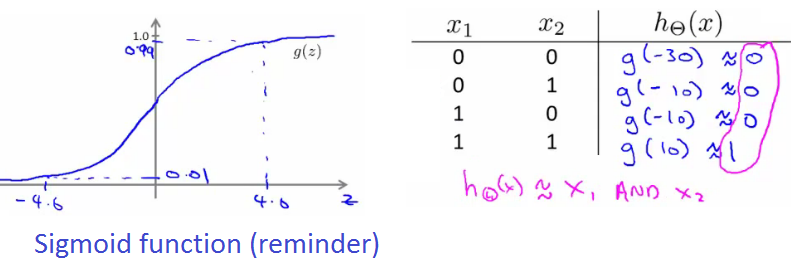
* Simple first example



* Can we get a one-unit neural network to compute this logical AND function? (*probably...*)
  + Add a bias unit
  + Add some weights for the networks
    - What are weights?
      * Weights are the parameter values which multiply into the input nodes (i.e. Ɵ)



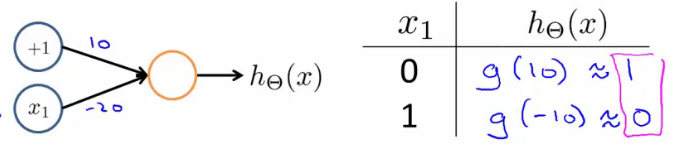
* Sometimes it's convenient to add the weights into the diagram
  + These values are in fact just the Ɵ parameters so
    - Ɵ101 = -30
    - Ɵ111 = 20
    - Ɵ121 = 20
  + To use our original notation
* Look at the four input values



* So, as we can see, when we evaluate each of the four possible input, only (1,1) gives a positive output

**Neural Network example 2: NOT function**

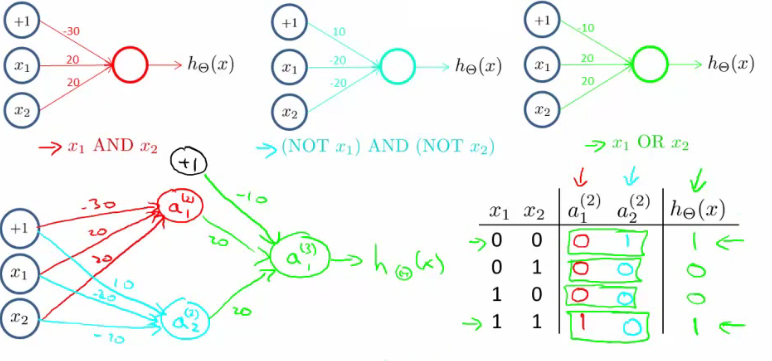
* How about negation?



* Negation is achieved by putting a large negative weight in front of the variable you want to negative

**Neural Network example 3: XNOR function**

* So how do we make the XNOR function work?
  + XNOR is short for NOT XOR
    - i.e. NOT an exclusive or, so either go big (1,1) or go home (0,0)
  + So we want to structure this so the input which produce a positive output are
    - AND (i.e. both true)  
      **OR**
    - Neither (which we can shortcut by saying not only one being true)
* So we combine these into a neural network as shown below;



* Simplez!

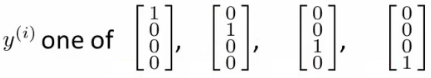
**Neural network intuition - handwritten digit classification**

* Yann LeCun = machine learning pioneer
* Early machine learning system was postcode reading
  + Hilarious music, impressive demonstration!

**Multiclass classification**

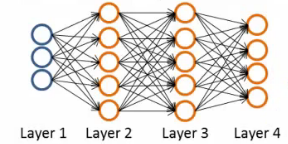
* Multiclass classification is, unsurprisingly, when you distinguish between more than two categories (i.e. more than 1 or 0)
* With handwritten digital recognition problem - 10 possible categories (0-9)
  + How do you do that?
  + Done using an extension of one vs. all classification
* Recognizing pedestrian, car, motorbike or truck
  + Build a neural network with four output units
  + Output a vector of four numbers
    - 1 is 0/1 pedestrian
    - 2 is 0/1 car
    - 3 is 0/1 motorcycle
    - 4 is 0/1 truck
  + When image is a pedestrian get [1,0,0,0] and so on
* Just like one vs. all described earlier
  + Here we have four logistic regression classifiers



* Training set here is images of our four classifications
  + While previously we'd written y as an integer {1,2,3,4}
  + Now represent y as
* 

**Neural network cost function**

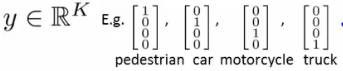
* NNs - one of the most powerful learning algorithms
  + Is a learning algorithm for fitting the derived parameters given a training set
  + Let's have a first look at a neural network cost function
* Focus on application of NNs for classification problems
* Here's the set up
  + Training set is {(x1, y1), (x2, y2), (x3, y3) ... (x*n*, y*m*)
  + *L* = number of layers in the network
    - In our example below L = 4
  + sl = number of units (not counting bias unit) in layer l



* So here
  + l   = 4
  + s1 = 3
  + s2 = 5
  + s3 = 5
  + s4 = 4

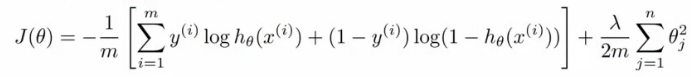
**Types of classification problems with NNs**

* Two types of classification, as we've previously seen
* **Binary classification**
  + 1 output (0 or 1)
  + So single output node - value is going to be a real number
  + k = 1
    - NB k is number of units in output layer
  + s*L* = 1
* **Multi-class classification**
  + k distinct classifications
  + Typically k is greater than or equal to three
  + If only two just go for binary
  + s*L* = k
  + So y is a k-dimensional vector of real numbers

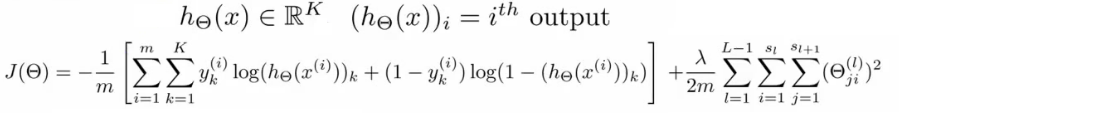


**Cost function for neural networks**

* The (regularized) logistic regression cost function is as follows;



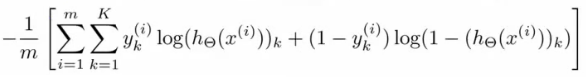
* For neural networks our cost function is a generalization of this equation above, so instead of one output we generate *k* outputs



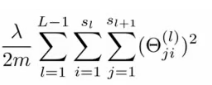
* Our cost function now outputs a *k* dimensional vector
  + hƟ(x) is a k dimensional vector, so hƟ(x)*i* refers to the ith value in that vector
* Costfunction J(Ɵ) is
  + [-1/m] times a sum of a similar term to which we had for logic regression
  + But now this is also a sum from k = 1 through to K (K is number of output nodes)
    - Summation is a sum over the k output units - i.e. for each of the possible classes
    - So if we had 4 output units then the sum is k = 1 to 4 of the logistic regression over each of the four output units in turn
  + This looks really complicated, but it's not so difficult
    - We don't sum over the bias terms (hence starting at 1 for the summation)
      * Even if you do and end up regularizing the bias term this is not a big problem
    - Is just summation over the terms

**Woah there - lets take a second to try and understand this!**

* There are basically two halves to the neural network logistic regression cost function

**First half**  


* This is just saying
  + For each training data example (i.e. 1 to m - the first summation)
    - Sum for each position in the output vector
* This is an average sum of logistic regression

**Second half**  


* This is a massive regularization summation term, which I'm not going to walk through, but it's a fairly straightforward triple nested summation
* This is also called a **weight decay** term
* As before, the lambda value determines the important of the two halves
* The regularization term is similar to that in logistic regression
* So, we have a cost function, but *how* do we minimize this bad boy?!

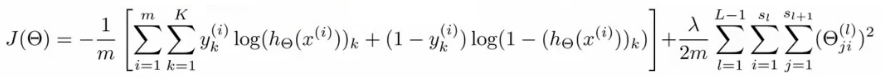
**Summary of what's about to go down**

*The following section is, I think, the most complicated thing in the course, so I'm going to take a second to explain the general idea of what we're going to do;*

* We've already described **forward propagation**
  + This is the algorithm which takes your neural network and the initial input into that network and pushes the input through the network
    - It leads to the generation of an output hypothesis, which may be a single real number, but can also be a vector
* We're now going to describe **back propagation**
  + Back propagation basically takes the output you got from your network, compares it to the real value (y) and calculates how wrong the network was (i.e. how wrong the parameters were)
  + It then, using the error you've just calculated, back-calculates the error associated with each unit from the preceding layer (i.e. layer *L -* 1)
  + This goes on until you reach the input layer (where obviously there is no error, as the activation is the input)
  + These "error" measurements for each unit can be used to calculate the **partial derivatives**
    - Partial derivatives are the bomb, because gradient descent needs them to minimize the cost function
  + We use the partial derivatives with gradient descent to try minimize the cost function and update all the Ɵ values
  + This repeats until gradient descent reports convergence
* A few things which are good to realize from the get go
  + There is a Ɵ matrix for each layer in the network
    - This has each node in layer l as one dimension and each node in l+1 as the other dimension
  + Similarly, there is going to be a Δ matrix for each layer
    - This has each node as one dimension and each training data example as the other

**Back propagation algorithm**

* We previously spoke about the neural network cost function
* Now we're going to deal with **back propagation**
  + Algorithm used to minimize the cost function, as it **allows us to calculate partial derivatives**!



* The cost function used is shown above
  + We want to find parameters Ɵ which minimize J(Ɵ)
  + To do so we can use one of the algorithms already described such as
    - Gradient descent
    - Advanced optimization algorithms
* To minimize a cost function we just write code which computes the following
  + **J(Ɵ)**
    - i.e. the cost function itself!
    - Use the formula above to calculate this value, so we've done that
  + **Partial derivative terms**
    - So now we need some way to do that
      * This is not trivial! Ɵ is indexed in three dimensions because we have separate parameter values for each node in each layer going to each node in the following layer
      * i.e. each layer has a Ɵ matrix associated with it!
        + We want to calculate the partial derivative Ɵ with respect to a single parameter   
          http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b7%5d.png
    - Remember that the partial derivative term we calculate above is a REAL number (not a vector or a matrix)
      * Ɵ is the input parameters
        + Ɵ1 is the matrix of weights which define the function mapping from layer 1 to layer 2
        + Ɵ101 is the real number parameter which you multiply the bias unit (i.e. 1) with for the bias unit input into the first unit in the second layer
        + Ɵ111 is the real number parameter which you multiply the first (real) unit with for the first input into the first unit in the second layer
        + Ɵ211 is the real number parameter which you multiply the first (real) unit with for the first input into the second unit in the second layer
        + As discussed, Ɵijl i

i here represents the unit in layer l+1 you're mapping to (destination node)

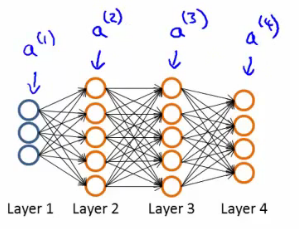
j is the unit in layer l you're mapping from (origin node)

l is the layer your mapping from (to layer l+1) (origin layer)

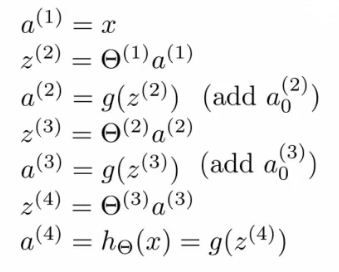
NB

*The terms destination node, origin node and origin layer are terms I've made up!*

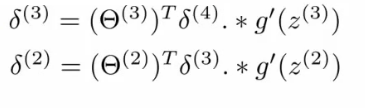
* + - So - this partial derivative term is
      * The partial derivative of a 3-way indexed dataset with respect to a real number (which is one of the values in that dataset)
  + **Gradient computation**
    - One training example
    - Imagine we just have a single pair (x,y) - entire training set
    - How would we deal with this example?
    - The forward propagation algorithm operates as follows
      * **Layer 1**
        + a1 = x
        + z2 = Ɵ1a1
      * **Layer 2**
        + a2 = g(z2) (add a02)
        + z3 = Ɵ2a2
      * **Layer 3**
        + a3 = g(z3) (add a03)
        + z4 = Ɵ3a3
      * **Output**
        + a4 = hƟ(x) = g(z4)

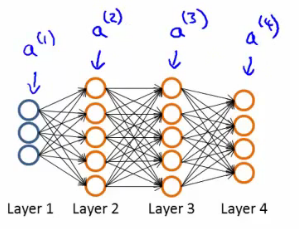


* + This is the vectorized implementation of forward propagation
    - Lets compute activation values sequentially (below just re-iterates what we had above!)



**What is back propagation?**

* Use it to compute the partial derivatives
* Before we dive into the mechanics, let's get an idea regarding the intuition of the algorithm
  + For each node we can calculate (δjl) - this is **the error of node j in layer l**
    - If we remember, ajl is the activation of node j in layer l
    - Remember the activation is a totally calculated value, so we'd expect there to be some error compared to the "real" value
      * The delta term captures this error
      * But the problem here is, "what is this 'real' value, and how do we calculate it?!"
        + The NN is a totally artificial construct
        + The only "real" value we have is our actual classification (our y value) - so that's where we start
* If we use our example and look at the fourth (output) layer, we can first calculate
  + δj4= aj4- yj
    - [Activation of the unit] - [the actual value observed in the training example]
    - We could also write aj4as hƟ(x)j
      * Although I'm not sure why we would?
  + This is an individual example implementation
* Instead of focussing on each node, let's think about this as a vectorized problem
  + δ4= a4- y
    - So here δ4is the vector of errors for the 4th layer
    - a4is the vector of activation values for the 4th layer
* With δ4calculated, we can determine the error terms for the other layers as follows;  
  
* Taking a second to break this down
  + Ɵ3is the vector of parameters for the 3->4 layer mapping
  + δ4is (as calculated) the error vector for the 4th layer
  + g'(z3) is the first derivative of the activation function g evaluated by the input values given by z3
    - You can do the calculus if you want (...), but when you calculate this derivative you get
    - g'(z3) = a3**. \*** (1 - a3)
  + So, more easily
    - **δ3= (Ɵ3)Tδ4. \*(a3. \*** **(1 - a3))**
  + **. \*** is the element wise multiplication between the two vectors
    - Why element wise? Because this is essentially an extension of individual values in a vectorized implementation, so element wise multiplication gives that effect
    - We highlighted it just in case you think it's a typo!

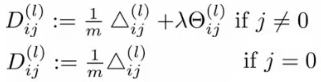
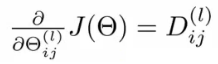
**Analyzing the mathematics**

* And if we take a second to consider the vector dimensionality (with our example above [3-5-5-4])
  + **Ɵ3** = is a matrix which is [4 X 5] (if we don't include the bias term, 4 X 6 if we do)
    - **(Ɵ3)**T = therefore, is a [5 X 4] matrix
  + **δ4** = is a 4x1 vector
  + So when we multiply a [5 X 4] matrix with a [4 X 1] vector we get a [5 X 1] vector
  + Which, low and behold, is the same dimensionality as the **a3**vector, meaning we can run our pairwise multiplication
* For δ3when you calculate the derivative terms you get  
  a3**. \*** (1 - a3)
* Similarly For δ2 when you calculate the derivative terms you get  
  a2**. \*** (1 - a2)
  + So to calculate δ2 we do  
    **δ2= (Ɵ2)Tδ3****. \*(a2. \* (1 - a2))**
* There's no δ1 term
  + Because that was the input!

**Why do we do this?**

* We do all this to get all the δ terms, and we want the δ terms because through a very complicated derivation you can use δ to get the partial derivative of Ɵ with respect to individual parameters (if you ignore regularization, or regularization is 0, which we deal with later)
* http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b12%5d.png = ajlδi(l+1)
* By doing back propagation and computing the delta terms you can then compute the **partial derivative terms**
  + We need the partial derivatives to minimize the cost function!

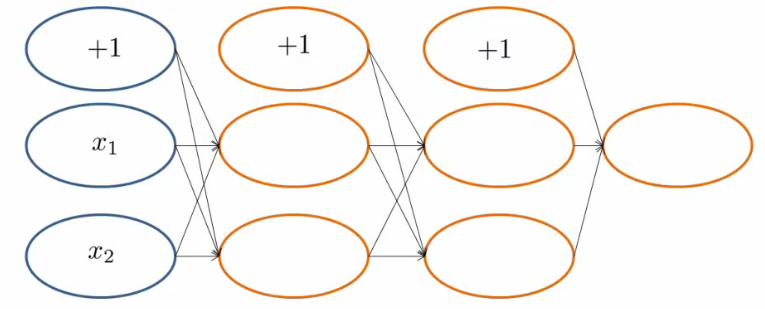
**Putting it all together to get the partial derivatives!**

* What is really happening - lets look at a more complex example
* Training set of m examples  
  http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b13%5d.png
* **First**, set the delta values  
  http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b14%5d.png
  + Set equal to 0 for all values
  + Eventually these Δ values will be used to compute the partial derivative
    - Will be used as accumulators for computing the partial derivatives
* **Next**, loop through the training set  
  http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b15%5d.png
  + i.e. for each example in the training set (dealing with each example as (x,y)
  + Set a1(activation of input layer) = xi
  + **Perform** **forward propagation** to compute alfor each layer (l = 1,2, ... L)
    - i.e. run forward propagation
  + **Then**, use the output label for the specific example we're looking at to calculate δL where δL= aL- yi
    - So we initially calculate the delta value for the output layer
    - Then, using **back propagation** we move back through the network from layer L-1 down to layer
  + Finally, use Δ to accumulate the partial derivative terms  
    http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b16%5d.png
  + Note here
    - l = layer
    - j = node in that layer
    - i = the error of the affected node in the target layer
  + You can vectorize the Δ expression too, as  
    http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b17%5d.png
* **Finally**
  + After executing the body of the loop, exit the for loop and compute   
    
    - When j = 0 we have no regularization term
* At the end of ALL this
  + You've calculated all the *D* terms above using Δ
    - NB - each D term above is a real number!
  + We can show that each D is equal to the following
    - 
  + We have calculated the partial derivative for each parameter
    - We can then use these in gradient descent or one of the advanced optimization algorithms
* Phew!
  + What a load of hassle!

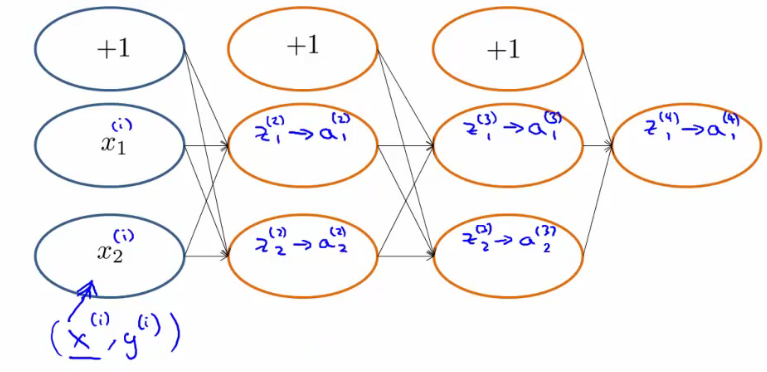
**Back propagation intuition**

* Some additionally back propagation notes
  + In case you found the preceding unclear, which it shouldn't be as it's fairly heavily modified with my own explanatory notes
* Back propagation is hard(ish...)
  + But don't let that discourage you
  + It's hard in as much as it's confusing - it's not difficult, just complex
* Looking at mechanical steps of back propagation

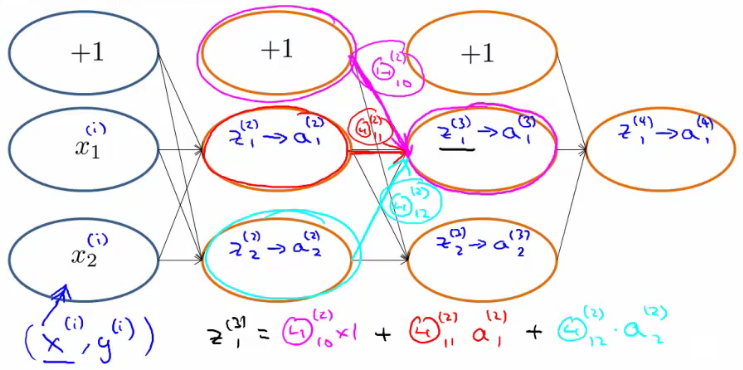
**Forward propagation with pictures!**

****

* Feeding input into the input layer (xi, yi)
  + Note that x and y here are vectors from 1 to n where n is the number of features
    - So above, our data has two features (hence x1 and x2)
* With out input data present we use **forward propagation**

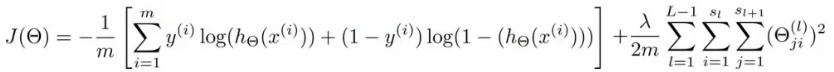


* The sigmoid function applied to the z values gives the activation values
  + Below we show exactly how the z value is calculated for an example



**Back propagation**

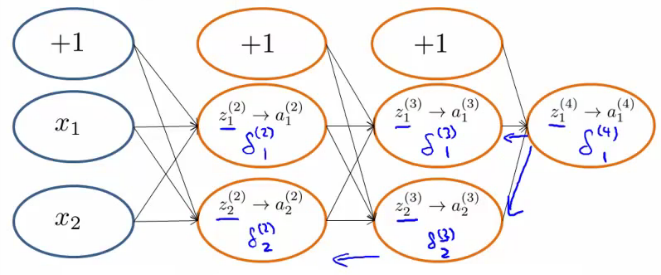
* With forwardprop done we move on to do back propagation
* Back propagation is doing something very similar to forward propagation, but backwards
  + Very similar though
* Let's look at the cost function again...
  + Below we have the cost function if there is a single output (i.e. binary classification)

****

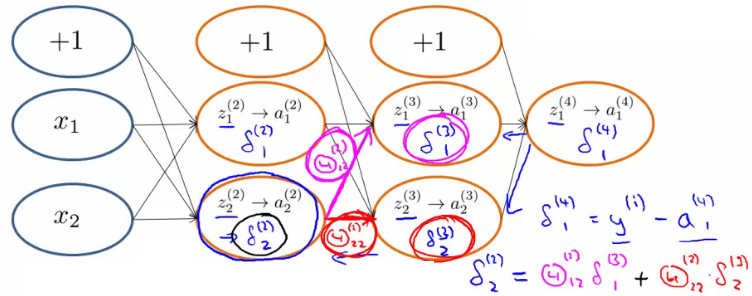
* This function cycles over each example, so the cost for one example really boils down to this

http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b24%5d.png

* Which, we can think of as a sigmoidal version of the squared difference (check out the derivation if you don't believe me)
  + So, basically saying, "how well is the network doing on example *i*"?
* We can think about a δ term on a unit as the "error" of cost for the activation value associated with a unit
  + More formally (*don't worry about this...*), δ is  
    http://www.holehouse.org/mlclass/09_Neural_Networks_Learning_files/Image%20%5b25%5d.png
    - Where cost is as defined above
    - Cost function is a function of y value and the hypothesis function
* So - for the output layer, back propagation sets the δ value as [a - y]
  + Difference between activation and actual value
* We then propagate these values backwards;



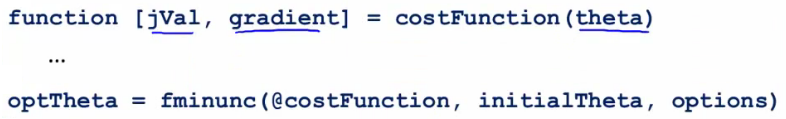
* Looking at another example to see *how* we actually calculate the delta value;



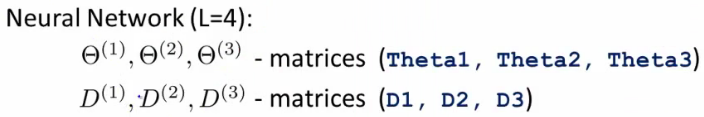
* So, in effect,
  + Back propagation calculates the δ, and those δ values are the weighted sum of the next layer's delta values, weighted by the parameter associated with the links
  + Forward propagation calculates the activation (a) values, which
* Depending on how you implement you may compute the delta values of the bias values
  + However, these aren't actually used, so it's a bit inefficient, but not a lot more!

**Implementation notes - unrolling parameters (matrices)**

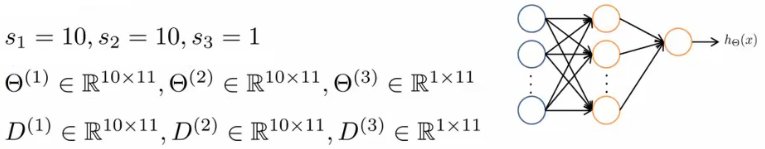
* Needed for using advanced optimization routines



* Is the MATLAB/octave code
  + But theta is going to be matrices
* fminunc takes the costfunction and initial theta values
  + These routines assume theta is a parameter vector
  + Also assumes the gradient created by costFunction is a vector
* For NNs, our parameters are matrices
  + e.g.



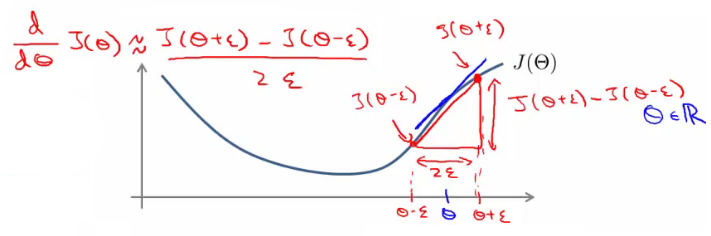
**Example**

****

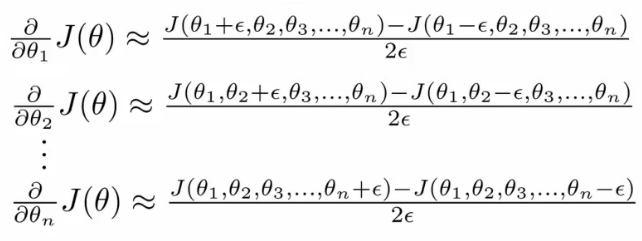
* Use the **thetaVec = [ Theta1(:); Theta2(:); Theta3(:)]**; notation to unroll the matrices into a long vector
* To go back you use
  + **Theta1 = resape(thetaVec(1:110), 10, 11)**

**Gradient checking**

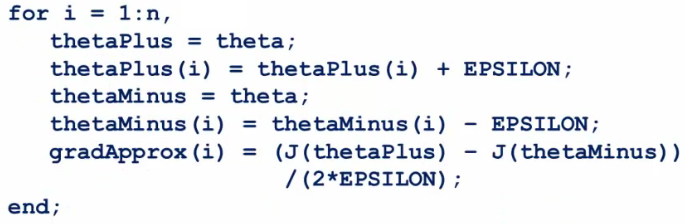
* Backpropagation has a lot of details, small bugs can be present and ruin it :-(
  + This may mean it looks like J(Ɵ) is decreasing, but in reality it may not be decreasing by as much as it should
* So using a numeric method to check the gradient can help diagnose a bug
  + Gradient checking helps make sure an implementation is working correctly
* **Example**
  + Have an function J(Ɵ)
  + Estimate derivative of function at point Ɵ (where Ɵ is a real number)
  + How?
    - Numerically
      * Compute Ɵ + ε
      * Compute Ɵ - ε
      * Join them by a straight line
      * Use the slope of that line as an approximation to the derivative



* Usually, epsilon is pretty small (0.0001)
  + If epsilon becomes REALLY small then the term BECOMES the slopes derivative
* The is the two sided difference (as opposed to one sided difference, which would be J(Ɵ + ε) - J(Ɵ) /ε
* If Ɵ is a vector with n elements we can use a similar approach to look at the partial derivatives



* So, in octave we use the following code the numerically compute the derivatives



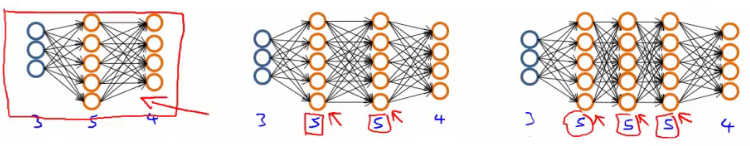
* So on each loop thetaPlus = theta except for thetaPlus(i)
  + Resets thetaPlus on each loop
* Create a vector of partial derivative approximations
* Using the vector of gradients from backprop (DVec)
  + Check that gradApprox is basically equal to DVec
  + Gives confidence that the Backproc implementation is correc
* Implementation note
  + Implement back propagation to compute DVec
  + Implement numerical gradient checking to compute gradApprox
  + Check they're basically the same (up to a few decimal places)
  + Before using the code for learning turn off gradient checking
    - Why?
      * GradAprox stuff is very computationally expensive
      * In contrast backprop is much more efficient (just more fiddly)

**Random initialization**

* Pick random small initial values for all the theta values
  + If you start them on zero (which does work for linear regression) then the algorithm fails - all activation values for each layer are the same
* So chose random values!
  + Between 0 and 1, then scale by epsilon (where epsilon is a constant)

**Putting it all together**

* **1) - pick a network architecture**
  + Number of
    - **Input units** - number of dimensions x (dimensions of feature vector)
    - **Output units** - number of classes in classification problem
    - **Hidden units**
      * Default might be
        + 1 hidden layer
      * Should probably have
        + Same number of units in each layer
        + Or 1.5-2 x number of input features
      * Normally
        + More hidden units is better
        + But more is more computational expensive
  + We'll discuss architecture more later



* **2) - Training a neural network**
  + **2.1)** Randomly initialize the weights
    - Small values near 0
  + **2.2)**Implement forward propagation to get hƟ(x)i for any xi
  + **2.3)**Implement code to compute the cost function J(Ɵ)
  + **2.4)**Implement back propagation to compute the partial derivatives
  + General implementation below

**for i = 1:m {**

**Forward propagation on (xi, yi) --> get activation (a) terms**

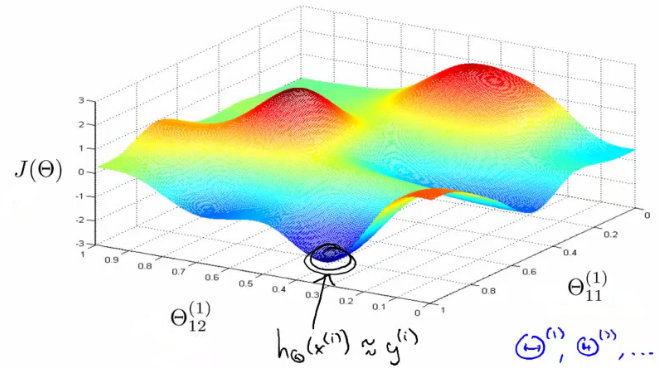
**Back propagation on (xi, yi) --> get delta (δ) terms**

**Compute Δ := Δl + δl+1(al)*T***

**}**

**With this done compute the partial derivative terms**

* + Notes on implementation
    - Usually done with a for loop over training examples (for forward and back propagation)
    - *Can* be done without a for loop, but this is a much more complicated way of doing things
    - Be careful
* **2.5)**Use gradient checking to compare the partial derivatives computed using the above algorithm and numerical estimation of gradient of J(Ɵ)
  + Disable the gradient checking code for when you actually run it
* **2.6)** Use gradient descent or an advanced optimization method with back propagation to try to minimize J(Ɵ) as a function of parameters Ɵ
  + Here J(Ɵ) is non-convex
    - Can be susceptible to local minimum
    - In practice this is not usually a huge problem
    - Can't guarantee programs with find global optimum should find good local optimum at least



* e.g. above pretending data only has two features to easily display what's going on
  + Our minimum here represents a hypothesis output which is pretty close to y
  + If you took one of the peaks hypothesis is far from y
* Gradient descent will start from some random point and move downhill
  + Back propagation calculates gradient down that hill

**10: Advice for applying Machine Learning**

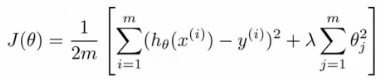
[Previous](http://www.holehouse.org/mlclass/09_Neural_Networks_Learning.html) [Next](http://www.holehouse.org/mlclass/11_Machine_Learning_System_Design.html) [Index](http://www.holehouse.org/mlclass/index.html)

**Deciding what to try next**

* We now know many techniques
  + But, there is a big difference between someone who knows an algorithm vs. someone less familiar and doesn't understand how to apply them
  + Make sure you know how to chose the best avenues to explore the various techniques
  + Here we focus deciding what avenues to try

**Debugging a learning algorithm**

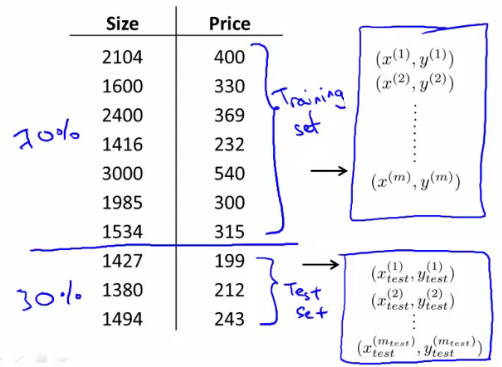
* So, say you've implemented regularized linear regression to predict housing prices

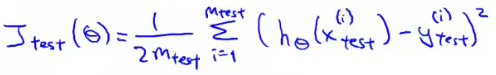
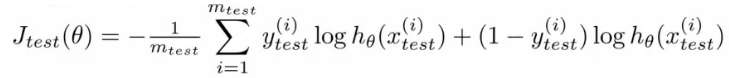
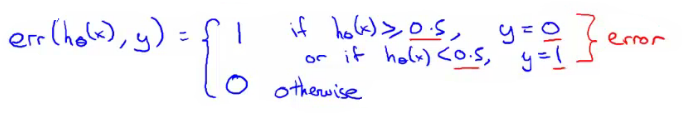
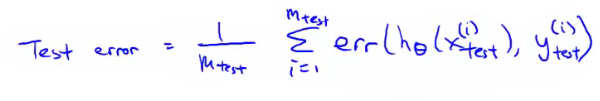


* + Trained it
  + But, when you test on new data you find it makes unacceptably large errors in its predictions
  + :-(
* What should you try next?
  + There are many things you can do;
    - **Get more training data**
      * Sometimes more data doesn't help
      * Often it does though, although you should always do some preliminary testing to make sure more data will actually make a difference (discussed later)
    - **Try a smaller set a features**
      * Carefully select small subset
      * You can do this by hand, or use some dimensionality reduction technique (e.g. PCA - we'll get to this later)
    - **Try getting additional features**
      * Sometimes this isn't helpful
      * LOOK at the data
      * Can be very time consuming
    - **Adding polynomial features**
      * You're grasping at straws, aren't you...
    - **Building your own, new, better features** based on your knowledge of the problem
      * Can be risky if you accidentally over fit your data by creating new features which are inherently specific/relevant to your training data
    - **Try decreasing or increasing λ**
      * Change how important the regularization term is in your calculations
  + These changes can become MAJOR projects/headaches (6 months +)
    - Sadly, most common method for choosing one of these examples is to go by gut feeling (randomly)
    - Many times, see people spend huge amounts of time only to discover that the avenue is fruitless
      * No apples, pears, or any other fruit. Nada.
  + There are some simple techniques which can let you rule out half the things on the list
    - Save you a lot of time!
* **Machine learning diagnostics**
  + Tests you can run to see what is/what isn't working for an algorithm
  + See what you can change to improve an algorithm's performance
  + These can take time to implement and understand (week)
    - But, they can also save you spending months going down an avenue which will *never* work

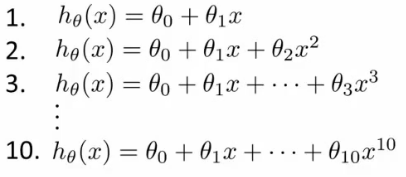
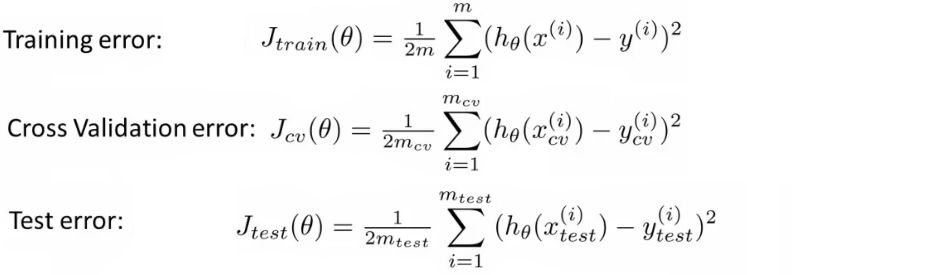
**Evaluating a hypothesis**

* When we fit parameters to training data, try and minimize the error
  + We might think a low error is good - doesn't necessarily mean a good parameter set
    - Could, in fact, be indicative of overfitting
    - This means you model will fail to generalize
  + How do you tell if a hypothesis is overfitting?
    - Could plot hθ(x)
    - But with lots of features may be impossible to plot
* Standard way to evaluate a hypothesis is
  + Split data into two portions
    - 1st portion is **training set**
    - 2nd portion is **test set**
  + Typical split might be 70:30 (training:test)



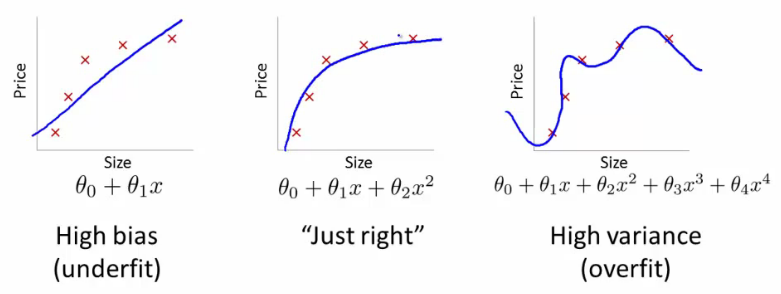
* NB if data is ordered, send a random percentage
  + (Or randomly order, then send data)
  + Data is typically ordered in some way anyway
* So a typical **train and test scheme** would be
  + 1) Learn parameters θ from training data, minimizing J(θ) using 70% of the training data]
  + 2) Compute the test error
    - Jtest(θ) = average square error as measured on the test set  
      
  + This is the definition of the **test set error**
* What about if we were using logistic regression
  + The same, learn using 70% of the data, test with the remaining 30%  
    
  + Sometimes there a better way - misclassification error (0/1 misclassification)
    - We define the error as follows  
      
    - Then the test error is  
      
      * i.e. its the fraction in the test set the hypothesis mislabels
* These are the standard techniques for evaluating a learned hypothesis

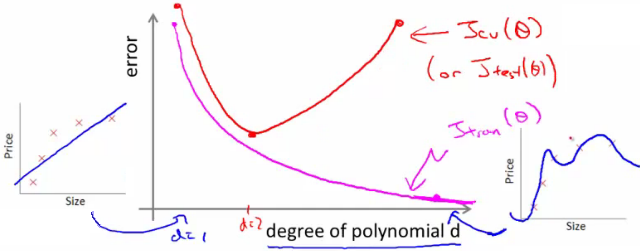
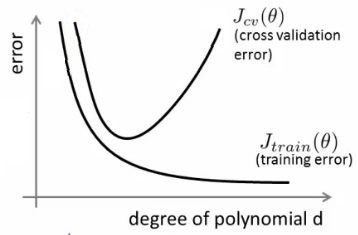
**Model selection and training validation test sets**

* How to chose regularization parameter or degree of polynomial (**model selection problems**)
* We've already seen the problem of overfitting
  + More generally, this is why training set error is a poor predictor of hypothesis accuracy for new data (generalization)
* Model selection problem
  + Try to chose the degree for a polynomial to fit data  
    
  + d = what degree of polynomial do you want to pick
    - An additional parameter to try and determine your training set
      * d =1 (linear)
      * d=2 (quadratic)
      * ...
      * d=10
    - Chose a model, fit that model and get an estimate of how well you hypothesis will generalize
  + You could
    - Take model 1, minimize with training data which generates a parameter vector θ1 (where d =1)
    - Take mode 2, do the same, get a *different* θ2 (where d = 2)
    - And so on
    - Take these parameters and look at the test set error for each using the previous formula
      * Jtest(θ1)
      * Jtest(θ2)
      * ...
      * Jtest(θ10)
  + You could then
    - See which model has the lowest test set error
  + Say, for example, d=5 is the lowest
    - Now take the d=5 model and say, how well does it generalize?
      * You could use Jtest(θ5)
      * BUT, this is going to be an optimistic estimate of generalization error, because our parameter is fit to that test set (i.e. specifically chose it because the test set error is small)
      * So not a good way to evaluate if it will generalize
  + To address this problem, we do something a bit different for model selection
* Improved model selection  
  + Given a training set instead split into three pieces
    - 1 - **Training set** (60%) - m values
    - 2 - **Cross validation** (CV) set (20%)mcv
    - 3 - **Test set** (20%) mtest
  + As before, we can calculate
    - Training error
    - Cross validation error
    - Test error  
      
  + So
    - Minimize cost function for each of the models as before
    - Test these hypothesis on the cross validation set to generate the cross validation error
    - Pick the hypothesis with the lowest cross validation error
      * e.g. pick θ5
    - Finally
      * Estimate generalization error of model using the test set
* Final note
  + In machine learning as practiced today - many people will select the model using the test set and then check the model is OK for generalization using the test error (which we've said is bad because it gives a bias analysis)
    - With a MASSIVE test set this is maybe OK
  + But considered much better practice to have separate training and validation sets

**Diagnosis - bias vs. variance**

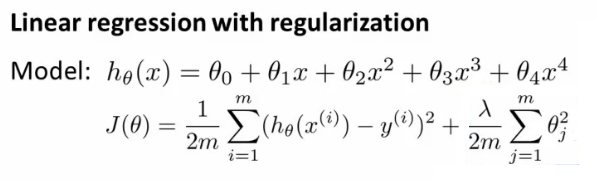
* If you get bad results usually because of one of
  + **High bias** - under fitting problem
  + **High variance** - over fitting problem
* Important to work out which is the problem
  + Knowing which will help let you improve the algorithm
* Bias/variance shown graphically below



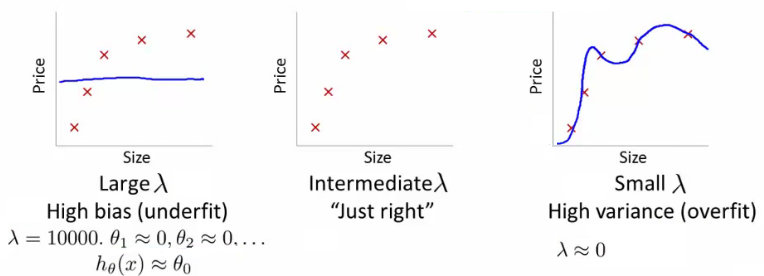
* The degree of a model will increase as you move towards overfitting
* Lets define training and cross validation error as before
* Now plot
  + x = degree of polynomial d
  + y = error for both training and cross validation (two lines)
    - CV error and test set error will be very similar   
      
    - This plot helps us understand the error
  + We want to minimize both errors
    - Which is why that d=2 model is the sweet spot
* How do we apply this for diagnostics
  + If cv error is high we're either at the high or the low end of d  
    
  + if d is too small --> this probably corresponds to a high bias problem
  + if d is too large --> this probably corresponds to a high variance problem
* **For the high bias case, we find both cross validation and training error are high**
  + Doesn't fit training data well
  + Doesn't generalize either
* **For high variance, we find the cross validation error is high but training error is low**
  + So we suffer from overfitting (training is low, cross validation is high)
  + i.e. training set fits well
  + But generalizes poorly

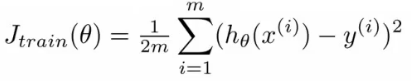
**Regularization and bias/variance**

* How is bias and variance effected by regularization?



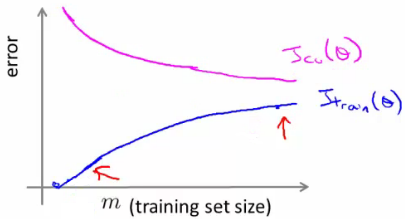
* The equation above describes fitting a high order polynomial with regularization (used to keep parameter values small)
  + Consider three cases
    - **λ = large**
      * All θ values are heavily penalized
      * So most parameters end up being close to zero
      * So hypothesis ends up being close to 0
      * So **high bias -> under fitting data**
    - **λ = intermediate**
      * Only this values gives the fitting which is reasonable
    - **λ = small**
      * Lambda = 0
      * So we make the regularization term 0
      * So **high variance -> Get overfitting** (minimal regularization means it obviously doesn't do what it's meant to)

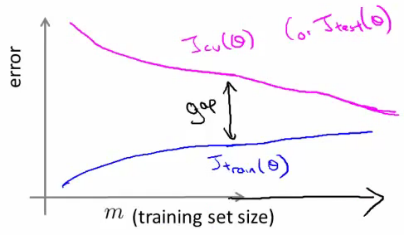


* How can we automatically chose a good value for λ?
  + To do this we define another function Jtrain(θ) which is the optimization function *without* the regularization term (average squared errors)  
    
  + Define cross validation error and test set errors as before (i.e. without regularization term)
    - So they are 1/2 average squared error of various sets
* **Choosing λ**
  + Have a set or range of values to use
  + Often increment by factors of 2 so
    - model(1)= λ = 0
    - model(2)= λ = 0.01
    - model(3)= λ = 0.02
    - model(4) = λ = 0.04
    - model(5) = λ = 0.08  
      .  
      .  
      .
    - model(p) = λ = 10
  + This gives a number of models which have different λ
  + With these models
    - Take each one (pth)
    - Minimize the cost function
    - This will generate some parameter vector
      * Call this θ(p)
    - So now we have a set of parameter vectors corresponding to models with different λ values
  + Take all of the hypothesis and use the cross validation set to validate them
    - Measure average squared error on cross validation set
    - Pick the model which gives the lowest error
    - Say we pick θ(5)
  + Finally, take the one we've selected (θ(5)) and test it with the test set
* **Bias/variance as a function of λ**
  + Plot λ vs.
    - Jtrain
      * When λ is small you get a small value (regularization basically goes to 0)
      * When λ is large you get a large vale corresponding to high bias
    - Jcv  
      * When λ is small we see high variance
        + Too small a value means we over fit the data
      * When λ is large we end up underfitting, so this is bias
        + So cross validation error is high
  + Such a plot can help show you you're picking a good value for λ

**Learning curves**

* A learning curve is often useful to plot for algorithmic sanity checking or improving performance
* What is a learning curve?
  + Plot Jtrain (average squared error on training set) or Jcv (average squared error on cross validation set)
  + Plot against m (number of training examples)
    - m is a constant
    - So artificially reduce m and recalculate errors with the smaller training set sizes
  + Jtrain
    - Error on smaller sample sizes is smaller (as less variance to accommodate)
    - So as m grows error grows
  + Jcv
    - Error on cross validation set
    - When you have a tiny training set your generalize badly
    - But as training set grows your hypothesis generalize better
    - So cv error will decrease as m increases



* What do these curves look like if you have
  + **High bias**
    - e.g. setting straight line to data
    - Jtrain
      * Training error is small at first and grows
      * Training error becomes close to cross validation
      * So the performance of the cross validation and training set end up being similar (but very poor)
    - Jcv
      * Straight line fit is similar for a few vs. a lot of data
      * So it doesn't generalize any better with lots of data because the function just doesn't fit the data
        + No increase in data will help it fit
    - The problem with high bias is because cross validation and training error are both high
    - Also implies that if a learning algorithm as high bias as we get more examples the cross validation error doesn't decrease
      * **So if an algorithm is already suffering from high bias, more data does not help**
      * So knowing if you're suffering from high bias is good!
      * In other words, high bias is a problem with the underlying way you're modeling your data
        + So more data won't improve that model
        + It's too simplistic
  + **High variance**
    - e.g. high order polynomial
    - Jtrain
      * When set is small, training error is small too
      * As training set sizes increases, value is still small
      * But slowly increases (in a near linear fashion)
      * Error is still low
    - Jcv
      * Error remains high, even when you have a moderate number of examples
      * Because the problem with high variance (overfitting) is your model doesn't generalize
    - An indicative diagnostic that you have high variance is that there's a big gap between training error and cross validation error
    - If a learning algorithm is suffering from high variance, more data is probably going to help  
      
      * **So if an algorithm is already suffering from high variance, more data will probably help**
        + Maybe
  + These are clean curves
  + In reality the curves you get are far dirtier
  + But, learning curve plotting can help diagnose the problems your algorithm will be suffering from

**What to do next (revisited)**

* How do these ideas help us chose how we approach a problem?
  + Original example
    - Trained a learning algorithm (regularized linear regression)
    - But, when you test on new data you find it makes unacceptably large errors in its predictions
    - What should try next?
  + How do we decide what to do?
    - **Get more examples** --> helps to fix high variance
      * Not good if you have high bias
    - **Smaller set of features** --> fixes high variance (overfitting)
      * Not good if you have high bias
    - **Try adding additional features** --> fixes high bias (because hypothesis is too simple, make hypothesis more specific)
    - **Add polynomial terms** --> fixes high bias problem
    - **Decreasing λ** --> fixes high bias
    - **Increases λ** --> fixes high variance
* Relating it all back to neural networks - selecting a network architecture
  + One option is to use a small neural network
    - Few (maybe one) hidden layer and few hidden units
    - Such networks are prone to under fitting
    - But they are computationally cheaper
  + Larger network
    - More hidden layers
      * How do you decide that a larger network is good?
  + Using a single hidden layer is good default
    - Also try with 1, 2, 3, see which performs best on cross validation set
    - So like before, take three sets (training, cross validation)
  + More units
    - This is computational expensive
    - Prone to over-fitting
      * Use regularization to address over fitting

**11: Machine Learning System Design**

[Previous](http://www.holehouse.org/mlclass/10_Advice_for_applying_machine_learning.html) [Next](http://www.holehouse.org/mlclass/12_Support_Vector_Machines.html) [Index](http://www.holehouse.org/mlclass/index.html)

**Machine learning systems design**

* In this section we'll touch on how to put together a system
* Previous sections have looked at a wide range of different issues in significant focus
* This section is less mathematical, but material will be very useful non-the-less
  + Consider the system approach
  + You can understand all the algorithms, but if you don't understand how to make them work in a complete system that's no good!

**Prioritizing what to work on - spam classification example**

The idea of prioritizing what to work on is perhap