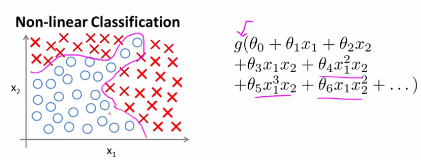
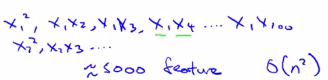
***Neural Networks***

**I. NON-LINEAR HYPOTHESIS**

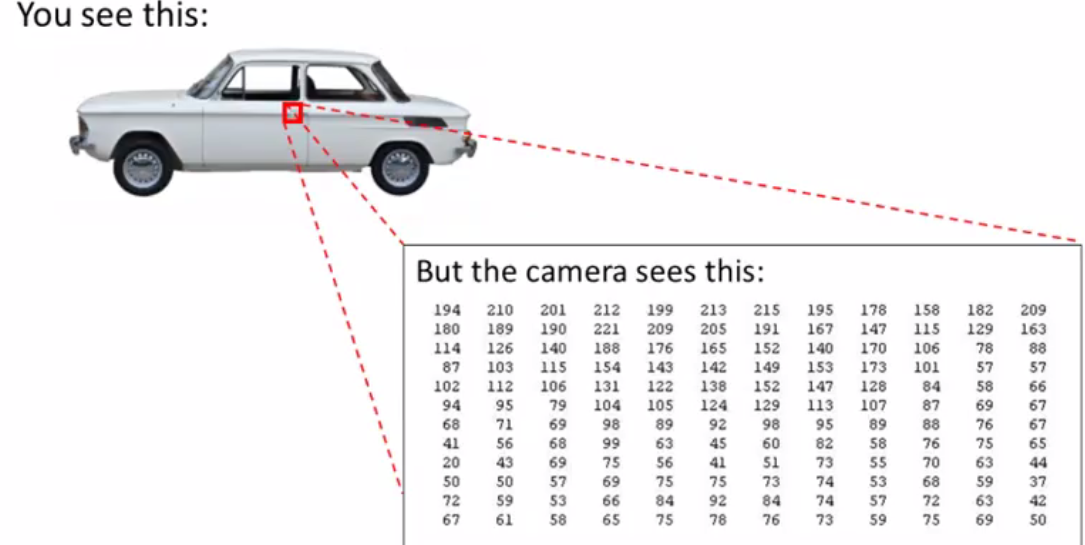
* **Neutral networks** is actually a pretty old idea, but had fallen out of favor for a while.
* But today, it is the state of the art technique for many different ML problems.
* So why do we need another learning algorithm? We already have linear regression + logistic regression
* Examples of ML problems where we need to learn complex, NON-LINEAR hypotheses.
* Supervised learning classification problem where you have a training set + you want to apply logistic regression



* Need to apply logistic regression w/ nonlinear features (where g is the sigmoid function) + lots of polynomial terms
* If you include enough polynomial terms, maybe you can get a hypotheses that separates positive + negative
* *This particular method only works well when you have, say, 2 features b/c you can then include all those polynomial terms of x1 + x2.*
* But many interesting ML problems have a lot more features than just 2.
* For housing prediction, suppose you have a housing classification problem rather than a regression problem
* You have 100 different features of a house + want to predict the odds your house will be sold w/in the next 6 months, so that will be a classification problem.
* For a problem like this, if you were to include all the quadratic (2nd-order polynomial) terms (terms that are a product of other terms starting w/ x1\*x1):

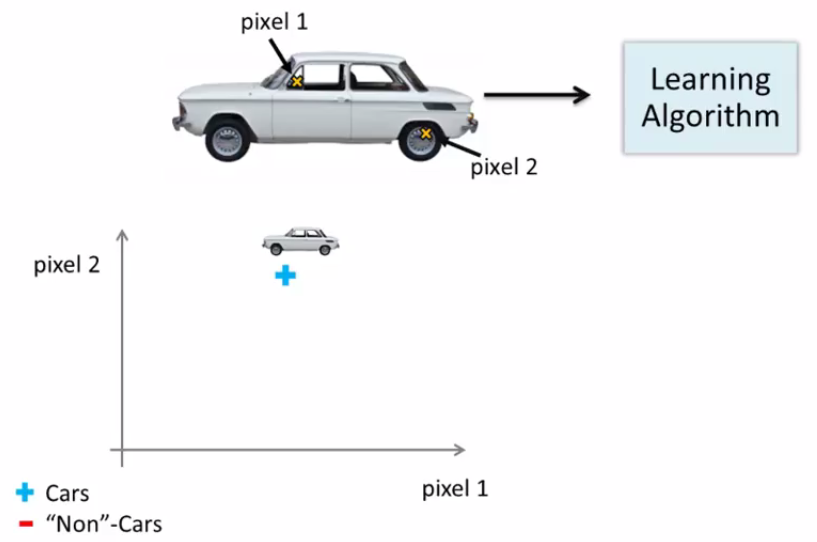
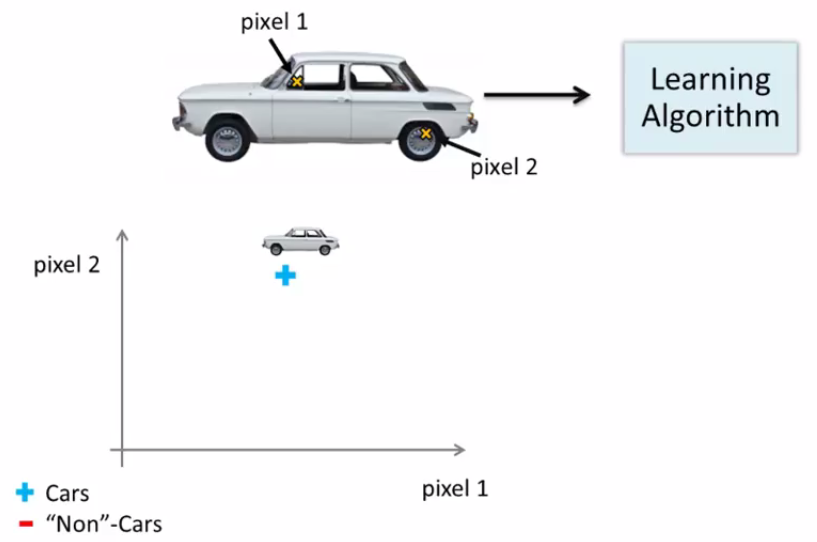


* We end up with about 5000 features.
* Asymptotically, the number of quadratic features grows roughly as **order n squared: O(n^2)** where n is the number of the original features,
* Its actually closer to n^2 / 2
* So including all the quadratic features doesn't seem like it's a good idea, b/c that’s a lot of features + you might up overfitting the training set, + it can also be computationally expensive
* 1 thing you could do is include only a subset of the quadratic features (only the squared terms for example), the number of features is much smaller 🡪 only 100 such quadratic features
* But this is not enough features to fit the data set like we have above
* In fact, if you include only these quadratic features together, you CAN actually fit very interesting hypotheses, such as axis lines of ellipses, but you certainly cannot fit a more complex data set like above.
* If you were to include the cubic, or 3rd-order polynomials of the features, we get O(n^3) 🡪 170,000 cubic features
* Including higher-order polynomial features when an original feature set is large really dramatically blows up a feature space
* Doesn't seem like a good way to come up w/ additional features w/ which to build non-linear classifiers when n is large.
* For many ML problems, n *WILL* be pretty large. Here's an example.
* Consider the problem of **computer vision** + you want to use ML to train a classifier to examine an image + tell us if the image is a car.
* To understand why CPU vision, where you + I see a car, the CPU sees a matrix/grid of pixel intensity values that signifies the brightness of each pixel in the image.
* So CPU vision problems to look at this matrix of pixel intensity values + tell us if these numbers represent, say, the door handle of a car.

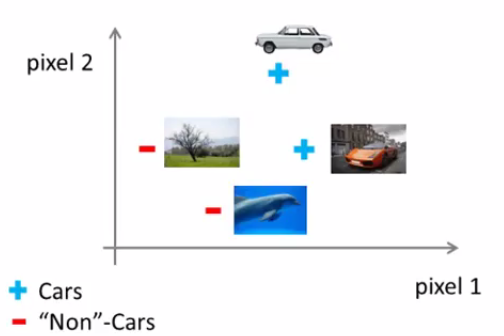


* Concretely, when we use ML to build a car detector, we come up w/ a *labelled* training set w/ examples of cars + examples of things that are not cars
* Then we give our training set to the learning algorithm, train, a classifier, and test it by showing a new image + asking "What is this new thing?" + hopefully it will recognize that that is a car.

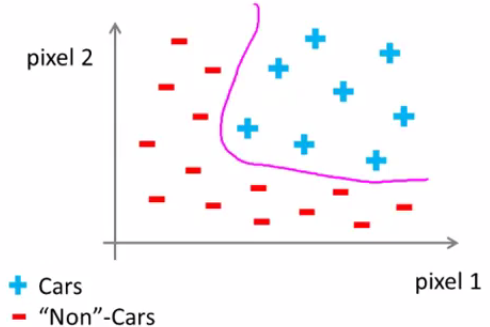
* To understand why we need **nonlinear hypotheses**, take a look at some of the images of cars + maybe non-cars we might feed to our learning algorithm.
* Pick a couple of pixel locations in an images + plot this car image at a certain point, depending on the intensities of pixel 1 and pixel 2:

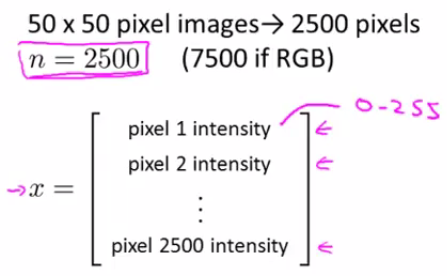
* Do this w/ a few other images: a different example of a car +some non-cars

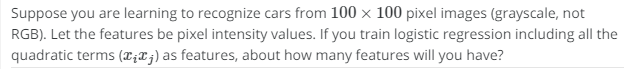


* If we do this for more + more examples, using pluses to denote cars + minuses to denote non-cars, we'll find that cars + non-cars end up lying in different regions of the space
* What we need therefore is some sort of non-linear hypotheses to try to separate out the 2 classes.



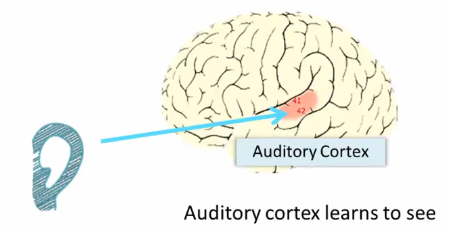
* Suppose we were to use just 50x50 pixel images (pretty small), we’d have 2500 pixels
* The dimension of our feature space will be n = 2500 where our feature vector x is a list of all the pixel intensities (brightness of pixel 1, brightness of pixel 2, down to the brightness of the last pixel)



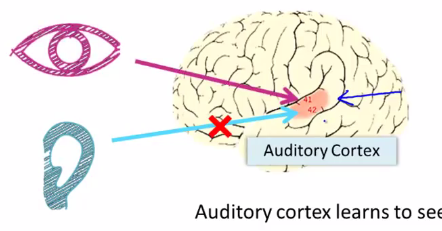
* In a typical CPU representation, each of these may be values between 0-255, if it gives us the grayscale value.
* If using RGB images (separate red, green + blue values), we would have n = 7500.
* If we were to try to learn a nonlinear hypothesis by including all quadratic features like before, w/ 2500 pixels, we’d end up w/ a total of 3 million features (2500^2 / 2)
* So, simple logistic regression together w/ adding in quadratic or cubic features is just not a good way to learn complex nonlinear hypotheses when n is large, b/c you just end up w/ too many features.
* 
* **50 million (5\*10^7)**

**II. NEURONS AND THE BRAIN**

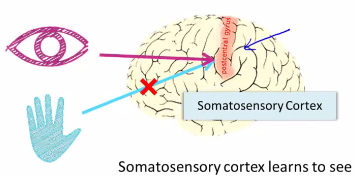
* **Neural Networks** are a pretty old algorithm originally motivated by the goal of having machines can mimic the brain + they work really well for different ML problems
* The origins of Neural Networks were algorithms that try to mimic the brain w/ the idea that if we want to build learning systems, why not mimic the most amazing learning machine we know about?
* NN’s came to be very widely used throughout the 1980's + 1990's
* For various reasons, popularity diminished in the late 90's, but more recently, NN’s have had a major recent resurgence.
* 1 reason for this resurgence is that NN’s are somewhat more computationally expensive algorithms
* It was only somewhat more recent that CPU’s became fast enough to really run large scale NN’s
* B/c of that as well as a few other technical reasons, modern NNs today are the state of the art technique for many applications.
* When you think about mimicking the brain, the human brain does many amazing things, right?
* It can learn to see process images, to hear, to process our sense of touch, to do calculus
* It does so many different + amazing things, so it seems like if you want to mimic the brain you have to write lots of different pieces of software to mimic all of these different fascinating things the brain does for us
* But there’s this fascinating hypothesis that the way the brain does all these different things is not w/ a thousand *different* programs, but instead, the brain does w/ *just a single learning algorithm*.
* This is just a hypothesis but here’s some evidence for this:



* The ear takes sound signals + routes them to the auditory cortex to allow you to understand them
* Neuroscientists have done experiments where you cut the wire from the ears to the auditory cortex + re-wire a brain so that the signal from the eyes to the optic nerve eventually gets routed to the auditory cortex.



* *If you do this it turns out, the auditory cortex will learn to see* (in every sense of the word “see” as we know it)
* So, if you do this to the animals, the animals can perform visual discrimination tasks + as they look at images, make appropriate decisions based on the images w/ that piece of brain tissue.



* Here's another example.
* The somatosensory cortex is how you process your sense of touch.
* If you do a similar re-wiring process to the one above, then the somatosensory cortex will learn to see.
* B/c of this and other similar experiments, these are called **neuro-rewiring experiments**.
* There's this sense that if the same piece of physical brain tissue can process sight OR sound OR touch, then maybe there is 1 learning algorithm that can process sight or sound or touch.
* Then, instead of needing to implement a thousand different programs/algorithms to do the thousand things the brain does, maybe what we need is to figure out some approximation to whatever the brain's learning algorithm is
* We’d implement that + the brain would learn how to process different types of data by itself

* To a surprisingly large extent, it seems as if we can plug in almost any sense to almost any part of the brain +, w/in reason, the brain will learn to deal with it.
* Here are a few more examples:



* On the upper left is an example of learning to see with your tongue.
* This is actually a system called BrainPort that’s undergoing FDA trials to help blind people see
* The way it works is, you strap a camera to your forehead, facing forward, that takes a low-resolution grayscale image of what's in front of you
* You then run a wire to an array of electrodes at you place on your tongue, such that each pixel gets mapped to a location on your tongue where maybe a high voltage corresponds to a dark pixel and a low voltage corresponds to a bright pixel
* W/ this sort of system, you + I will be able to learn to see w/ out tongues in 10’s of minutes
* 2nd example = human echo location or human sonar.
* There’s are 2 ways you can do this 🡪 either snap your fingers or click your tongue.
* There are blind people today that are actually being trained in schools to do this + learn to interpret the pattern of sounds bouncing off the environment (sonar).
* There is a kid who tragically had his eyeballs removed due to cancer, but by snapping his fingers, he can walk around + never hit anything, ride a skateboard, + shoot a basketball
* 3rd example: The Haptic Belt
* Have a strap around your waist w/ a ring of buzzers + always have the northern-most one buzzing to give a human a sense of direction similar to how birds can sense where north is
* 4th example: if you plug a 3rd eye into a frog, the frog will learn to use that eye as well.
* It's pretty amazing as to what extent you can plug in almost any sensor to the brain + the brain's learning algorithm will just figure out how to learn from + deal w/ that data.
* There's a sense that if we can figure out what the brain's learning algorithm is + implement some approximation of that algorithm on a CPU, maybe that would be our best shot at making real progress towards building truly intelligent machines + AI