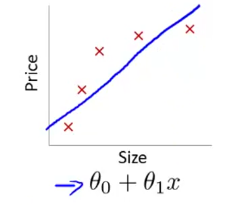
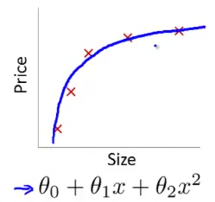
***Solving the Problem of Overfitting***

**I. THE PROBLEM OF OVERFITTING**

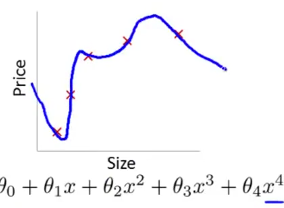
* When you apply linear + logistic regression to certain ML applications, they can run into a problem called **overfitting** that can cause them to perform very poorly.
* 1 thing we could do to our housing prices data set is fit a linear function to this data via linear regression and get a *straight* line fit to the data.



* This isn't a good model b/c as we see size increase, we see prices plateau 🡪 this algorithm does not fit well = **underfitting** = this algorithm has **high bias**.
* It's as if the algorithm has a very strong preconception/bias that prices are going to vary linearly w/ size, *despite data/evidence to the contrary* + it ends up a poor fit to the data.
* Model is too simple/uses too few features
* We could fit a quadratic function



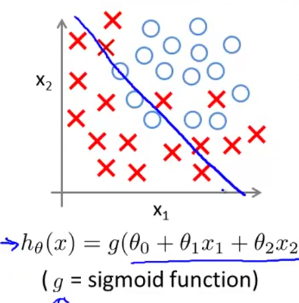
* This works well.
* We could fit a 4th-order other polynomial to the data (5 parameters)



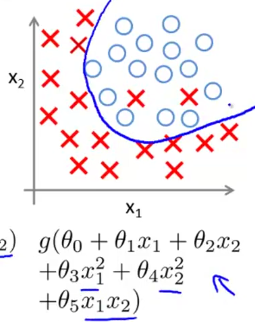
* This creates curve through all 5 training examples 🡪 seems to do a very good job fitting the training set, *but it's going all over the place + is not such a good model*
* **Overfitting** = this algorithm has **high variance**
* If fitting a high order polynomial, then it’s as if the hypothesis can fit can fit almost any function so the space of possible hypotheses is too large/variable + we don't have enough data to constrain it to give us a good hypothesis
* **Overfitting**: If we have too many features, the **learned hypothesis** may fit training data very well

)

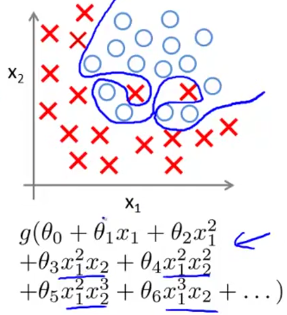
* But the function tries *too hard* to fit to the *training* set such that it fails to generalize to new examples/test data (to predict prices on new house size examples)
* **Generalize =** how well a hypothesis applies to new examples (data it has not seen in training set)
* Overfitting can apply to logistic regression as well.
* For a simple example with 2 features, x1 and x2, we could fit it w/ logistic regression w/ just a simple hypothesis + end up w/ a hypothesis with straight line to separate positive and the negative class.



* Not very good fit = underfitting = hypothesis has high bias.
* Could add quadratic terms to your features these + get a decision boundary.



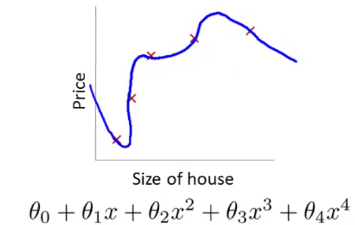
* that's a good fit to data
* If you fit a very high-order polynomial, logistical regression may contort itself to find a decision boundary that fits every single training example well.



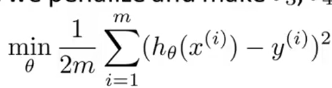
* This doesn't look like a very good hypothesis for making predictions as it tries too hard to fit every training example well
* It’s **overfitting** = hypothesis has **high variance** = unlikely to generalize well to new examples.
* In previous examples, we had 1 or 2 dimensional data, so we could just plot hθ(x) + see what was going on + select the appropriate degree of polynomial.
* So plotting hθ(x) could be 1 way to try to decide what degree polynomial to use, but that doesn't always work.
* More often than not, we have a lot of features + it’s not just a matter of selecting what degree polynomial to use
* W/ too many features, it becomes much harder to plot data + to decide what features to keep
* *If we have a lot of features + very little training data, overfitting can become a problem.*
* To address over fitting, there are 2 main options
* Try to reduce the # of features
* *Manually* look through list of features + decide which are more important + which to ignore
* But, by throwing away features, we also throw away info we have about the problem
* Maybe, ALL features are indeed useful for predicting housing prices so, we don't want to throw some of our info/features away.
* Model Selection *algorithm*
* **Regularization**
* Keep all features but reduce the **magnitude**/values of the parameters θ(j)
* This works well when we have a lot of features, each of which contributes a little bit to predicting the value of y

**II. COST FUNCTION**

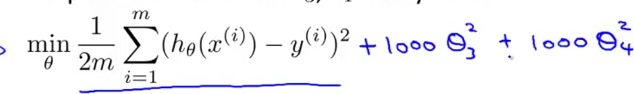
* So, remember that if we were to fit a quadratic function to data, we can get a pretty good fit
* Whereas, if fitting an overly-high-ordered polynomial, we end up w/ a curve that fits a training set very well, but overfits the data + does not generalize well.



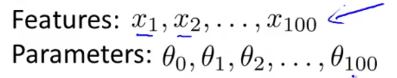
* Looking at the effect of penalizing 2 of the parameter values being large, suppose we penalize + make the parameters θ3 and θ4 really small.
* *\*\*\*\*\*Reduce the weight that some of the terms in our function carry by increasing their cost.*
* Our **optimization problem**, where we minimize our usual squared error cause function:



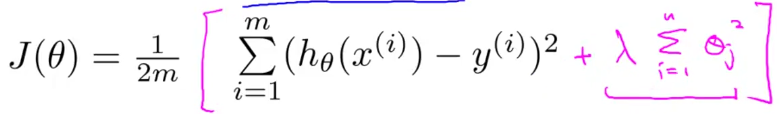
* Say we modify this objective + add to 1000\*θ3^2 + 1000\*θ4^2



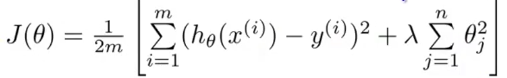
* The only way to make this *new* cost function small is if θ3 and θ4 are small (close to 0), as if we're getting rid of these 2 terms at the end of the polynomial.
* If θ3 + θ4 are close to 0, we’re left w/ a quadratic function plus tiny contributions from the very small terms θ3 + θ4 which gives a good fit to the data
* Idea behind **regularization:**
* *Having small values for the parameters θ0-θn usually corresponds to having a “simpler” hypothesis*
* If we penalize just θ3 + θ4 to make them close to 0 + wound up w/ a much simpler hθ(x), which was essentially a quadratic function.
* But more broadly, if we penalize ALL parameters usually, think of it as trying to give us a simpler hypothesis
* More generally, it’s possible to show that having smaller values of parameters corresponds to smoother, more simple functions (usually), which are therefore, less prone to overfitting.
* Ex: For housing price prediction, we may have 100 features +, unlike the polynomial example, we don't know that θ3 + θ4 are the high order polynomial terms.

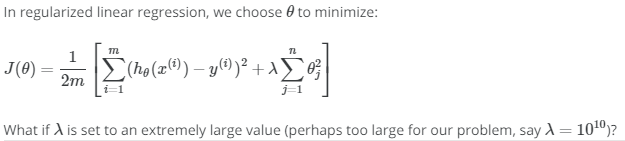


* So, if we have just a set of a 100 features, it's hard to pick in advance which are the ones less likely to be relevant so we don’t know which parameters to shrink.
* In regularization, we take our cost function for linear regression + modify it to shrink ALL parameters (b/c we don't know which to shrink) via a new **regularization term** at the end.



* W/ this extra regularization term, we shrink every single parameter from θ1- θn (not penalizing θ0 because the sum is from i = 1 to m)
* By convention, the summation for the regularization term starts from 1 so we’re don’t penalize θ0 being large.
* makes very little difference θ0 or not in the summation in practice,



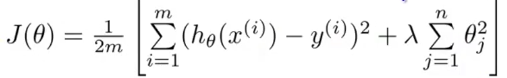
* **Lambda (λ)** = the **regularization parameter** 🡪 controls the trade-off between 2 different goals.
* 1 🡪 Would like to fit training data well (captured by 1st term in the objective/original objective)
* 2 🡪 Want to keep parameters small (captured by the 2nd term = the regularization objective)
* **λ**'s controlled trade-off between them keeps the hypothesis relatively simple to avoid overfitting
* Using the regularized objective w/ a high-ordered polynomial, you can get a curve that isn't quite a quadratic function, but is much smoother + simpler than a wiggly overfit + gives a much better hypothesis for this data.
* 
* 
* In **regularized linear regression**, if the **regularization parameter** **λ** is set to be *very large*, we end up penalizing parameters θ1-θn very highly + they end up close to 0
* This is basically getting rid of all terms in hθ(x), we're just left w/ a hθ(x) = θ0
* This is akin to fitting a *flat horizontal straight line* to the data = **underfitting**



* Another way of saying this is that this hθ(x) has *too strong a preconception* or *too high bias* that housing prices are equal to θ0, despite clear data to the contrary
* For regularization to work well, some care should be taken to choose a good **λ**

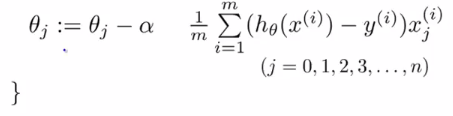
**III. REGULARIZED LINEAR REGRESSION**

* For linear regression, we have previously worked out 2 learning algorithms: 1 based on **gradient descent**, 1 based on the **normal equation**
* Optimization objective for regularized linear regression.

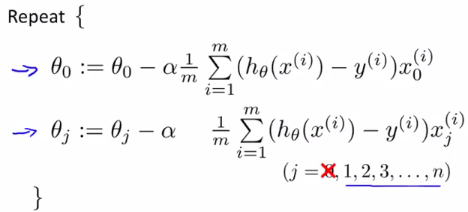


* 1st part = usual objective for linear regression, 2nd part = additional regularization term w/ regularization parameter **λ**
* We want to find parameters θ that minimizes this regularized cost function J(θ).
* Previously, we were using gradient descent for the original cost function *w/out* a regularization term

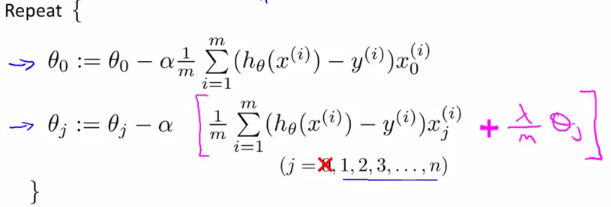




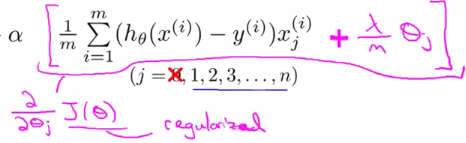
* We would repeatedly update the parameters θj for j0-j(n)
* Write the case for θ0 separately.



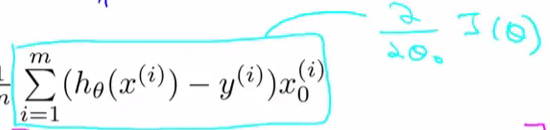
* We haven't changed anything yet, right.
* Remember that for regularized linear regression, *we don't penalize θ0.*
* So, when we modify this algorithm for gradient descent for regularized linear regression, we're going to end up treating θ0 slightly differently.
* Concretely, if we want to modify this algorithm to use the regularized objective, all we need to do is add λ/m times θj to the bottom term



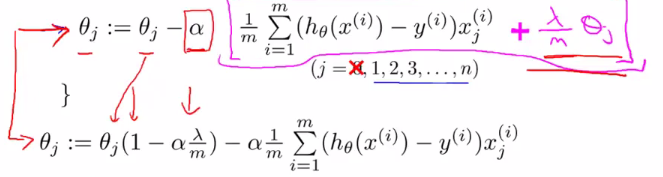
* If you implement this, then you have gradient descent for trying to minimize the regularized J(θ).
* W/in the pink square brackets is the partial derivative w/ respect to J(θ) using the *new* definition of J(θ) w/ the regularization term.



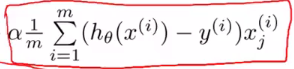
* Similarly, up on we have the partial derivative of J(θ) w/ respect of θ0



* If you look at the update for θj, it gets updated as θj – **α**\*this other term that depends on θj.
* So if you group all the terms that depend on θj together, you can show that this update can be written equivalently as:



* The term **(1 – α\*(λ/m))** is a pretty interesting term + has a pretty interesting effect.
* This term is going to be a number that is usually a little bit less than 1, because **α\*(λ/m)** is going to be positive, + if **α** is small + if m is large, it’s usually pretty small.
* Think of it as a number like 0.99
* Now say that θj gets replaced by θj\*0.99 🡪 this has the effect of *shrinking θj* a little bit towards 0.
* More formally, this makes the **squared norm** of θj a little bit smaller.
* After that, the second term is the same as the original gradient descent update we had before we added all the regularization stuff.



* So in *regularized* linear regression, on every iteration we're multiplying θj by a number that's a little bit < 1 to shrink the parameter a little bit + *then* performing a similar update as before.
* Of course, that's just the intuition behind what this particular update is doing.
* Mathematically what its doing is *exactly* gradient descent on the regularization cost function of J(θ) + the regularization term.
* Gradient descent was just one of our two algorithms for fitting a linear regression model. The second algorithm was the one based on the normal equation, where what we did was we created the design matrix X where each row corresponded to a separate training example. And we created a vector y, so this is a vector, that's an m dimensional vector. And that contained the labels from my training set. So whereas X is an m by (n+1) dimensional matrix, y is an m dimensional vector. And in order to minimize the cost function J, we found that one way to do so is to set θ to be equal to this. Right, you have X transpose X, inverse, X transpose Y. I'm leaving room here to fill in stuff of course. And what this value for θ does is this minimizes the cost function J of θ, when we were not using regularization.
* 6:26
* Now that we are using regularization, if you were to derive what the minimum is, and just to give you a sense of how to derive the minimum, the way you derive it is you take partial derivatives with respect to each parameter. Set this to zero, and then do a bunch of math and you can then show that it's a formula like this that minimizes the cost function. And concretely, if you are using regularization, then this formula changes as follows. Inside this parenthesis, you end up with a matrix like this. 0, 1, 1, 1, and so on, 1, until the bottom. So this thing over here is a matrix whose upper left-most entry is 0. There are ones on the diagonals, and then zeros everywhere else in this matrix. Because I'm drawing this rather sloppily. But as a example, if n = 2, then this matrix is going to be a three by three matrix. More generally, this matrix is an (n+1) by (n+1) dimensional matrix. So if n = 2, then that matrix becomes something that looks like this. It would be 0, and then 1s on the diagonals, and then 0s on the rest of the diagonals. And once again, I'm not going to show this derivation, which is frankly somewhat long and involved, but it is possible to prove that if you are using the new definition of J of θ, with the regularization objective, then this new formula for θ is the one that we give you, the global minimum of J of θ.
* 8:01
* So finally I want to just quickly describe the issue of non-invertibility. This is relatively advanced material, so you should consider this as optional. And feel free to skip it, or if you listen to it and positive it doesn't really make sense, don't worry about it either. But earlier when I talked about the normal equation method, we also had an optional video on the non-invertibility issue. So this is another optional part to this, sort of an add-on to that earlier optional video on non-invertibility. Now, consider a setting where m, the number of examples, is less than or equal to n, the number of features. If you have fewer examples than features, than this matrix, X transpose X will be non-invertible, or singular. Or the other term for this is the matrix will be degenerate. And if you implement this in Octave anyway and you use the pinv function to take the pseudo inverse, it will kind of do the right thing, but it's not clear that it would give you a very good hypothesis, even though numerically the Octave pinv function will give you a result that kinda makes sense.
* 9:13
* But if you were doing this in a different language, and if you were taking just the regular inverse, which in Octave denoted with the function inv, we're trying to take the regular inverse of X transpose X. Then in this setting, you find that X transpose X is singular, is non-invertible, and if you're doing this in different program language and using some linear algebra library to try to take the inverse of this matrix, it just might not work because that matrix is non-invertible or singular. Fortunately, regularization also takes care of this for us. And concretely, so long as the regularization parameter λ is strictly greater than 0, it is actually possible to prove that this matrix, X transpose X plus λ times this funny matrix here, it is possible to prove that this matrix will not be singular and that this matrix will be invertible. So using regularization also takes care of any non-invertibility issues of the X transpose X matrix as well. So you now know how to implement regularized linear regression. Using this you'll be able to avoid overfitting even if you have lots of features in a relatively small training set. And this should let you get linear regression to work much better for many problems. In the next video we'll take this regularization idea and apply it to logistic regression. So that you'd be able to get logistic regression to avoid overfitting and perform much better as well.
* Downloads
* Lecture Videomp4
* Subtitles (English)
* WebVTT
* Transcript (English)
* txt