MachineLearning-Lecture02

Instructor (Andrew Ng):All right, good morning, welcome back. So before we jump into today's material, I just have one administrative announcement, which is graders. So I guess sometime next week, we'll hand out the first homework assignment for this class.

Is this loud enough, by the way? Can people in the back hear me? No. Can you please turn up the mic a bit louder? Is this better? Is this okay? This is okay? Great.

So sometime next week, we'll hand out the first problem sets and it'll be two weeks after that, and the way we grade homework problems in this class is by some combination of TAs and graders, where graders are usually members – students currently in the class.

So in maybe about a week or so, I'll email the class to solicit applications for those of you that might be interested in becoming graders for this class, and there's usually sort of a fun thing to do. So four times this quarter, the TAs, and the graders, and I will spend one evening staying up late and grading all the homework problems.

For those of you who that have never taught a class before, or sort of been a grader, it's an interesting way for you to see, you know, what the other half of the teaching experience is. So the students that grade for the first time sort of get to learn about what it is that really makes a difference between a good solution and amazing solution. And to give everyone to just how we do points assignments, or what is it that causes a solution to get full marks, or just how to write amazing solutions. Becoming a grader is usually a good way to do that.

Graders are paid positions and you also get free food, and it's usually fun for us to sort of hang out for an evening and grade all the assignments. Okay, so I will send email. So don't email me yet if you want to be a grader. I'll send email to the entire class later with the administrative details and to solicit applications. So you can email us back then, to apply, if you'd be interested in being a grader.

Okay, any questions about that? All right, okay, so let's get started with today's material. So welcome back to the second lecture. What I want to do today is talk about linear regression, gradient descent, and the normal equations. And I should also say, lecture notes have been posted online and so if some of the math I go over today, I go over rather quickly, if you want to see every equation written out and work through the details more slowly yourself, go to the course homepage and download detailed lecture notes that pretty much describe all the mathematical, technical contents I'm going to go over today.

Today, I'm also going to delve into a fair amount – some amount of linear algebra, and so if you would like to see a refresher on linear algebra, this week's discussion section will be taught by the TAs and will be a refresher on linear algebra. So if some of the linear algebra I talk about today sort of seems to be going by pretty quickly, or if you just want to see some of the things I'm claiming today with our proof, if you want to just see some of those things written out in detail, you can come to this week's discussion section.

So I just want to start by showing you a fun video. Remember at the last lecture, the initial lecture, I talked about supervised learning. And supervised learning was this machine-learning problem where I said we're going to tell the algorithm what the close right answer is for a number of examples, and then we want the algorithm to replicate more of the same.

So the example I had at the first lecture was the problem of predicting housing prices, where you may have a training set, and we tell the algorithm what the "right" housing price was for every house in the training set. And then you want the algorithm to learn the relationship between sizes of houses and the prices, and essentially produce more of the "right" answer.

So let me show you a video now. Load the big screen, please. So I'll show you a video now that was from Dean Pomerleau at some work he did at Carnegie Mellon on applied supervised learning to get a car to drive itself. This is work on a vehicle known as Alvin. It was done sort of about 15 years ago, and I think it was a very elegant example of the sorts of things you can get supervised or any algorithms to do.

On the video, you hear Dean Pomerleau's voice mention and algorithm called Neural Network. I'll say a little bit about that later, but the essential learning algorithm for this is something called gradient descent, which I will talk about later in today's lecture. Let's watch the video. [Video plays]

Instructor (Andrew Ng):So two comments, right. One is this is supervised learning because it's learning from a human driver, in which a human driver shows that we're on this segment of the road, I will steer at this angle. This segment of the road, I'll steer at this angle. And so the human provides the number of "correct" steering directions to the car, and then it's the job of the car to try to learn to produce more of these "correct" steering directions that keeps the car on the road.

On the monitor display up here, I just want to tell you a little bit about what this display means. So on the upper left where the mouse pointer is moving, this horizontal line actually shows the human steering direction, and this white bar, or this white area right here shows the steering direction chosen by the human driver, by moving the steering wheel.

The human is steering a little bit to the left here indicated by the position of this white region. This second line here where Mamos is pointing, the second line here is the output of the learning algorithm, and where the learning algorithm currently thinks is the right steering direction. And right now what you're seeing is the learning algorithm just at the very beginning of training, and so there's just no idea of where to steer. And so its output, this little white smear over the entire range of steering directions.

And as the algorithm collects more examples and learns of a time, you see it start to more confidently choose a steering direction. So let's keep watching the video. [Video plays]

Instructor (Andrew Ng):All right, so who thought driving could be that dramatic, right? Switch back to the chalkboard, please. I should say, this work was done about 15 years ago and autonomous driving has come a long way. So many of you will have heard of the DARPA Grand Challenge, where one of my colleagues, Sebastian Thrun, the winning team's drive a car across a desert by itself.

So Alvin was, I think, absolutely amazing work for its time, but autonomous driving has obviously come a long way since then. So what you just saw was an example, again, of supervised learning, and in particular it was an example of what they call the regression problem, because the vehicle is trying to predict a continuous value variables of a continuous value steering directions, we call the regression problem.

And what I want to do today is talk about our first supervised learning algorithm, and it will also be to a regression task. So for the running example that I'm going to use throughout today's lecture, you're going to return to the example of trying to predict housing prices. So here's actually a dataset collected by TA, Dan Ramage, on housing prices in Portland, Oregon.

So here's a dataset of a number of houses of different sizes, and here are their asking prices in thousands of dollars, $200,000. And so we can take this data and plot it, square feet, best price, and so you make your other dataset like that. And the question is, given a dataset like this, or given what we call a training set like this, how do you learn to predict the relationship between the size of the house and the price of the house?

So I'm actually going to come back and modify this task a little bit more later, but let me go ahead and introduce some notation, which I'll be using, actually, throughout the rest of this course. The first piece of notation is I'm going to let the lower case alphabet M denote the number of training examples, and that just means the number of rows, or the number of examples, houses, and prices we have.

And in this particular dataset, we have, what actually happens, we have 47 training examples, although I wrote down only five. Okay, so throughout this quarter, I'm going to use the alphabet M to denote the number of training examples. I'm going to use the lower case alphabet X to denote the input variables, which I'll often also call the features. And so in this case, X would denote the size of the house they were looking at.

I'm going to use Y to denote the "output" variable, which is sometimes also called a target variable, and so one pair, x, y, is what comprises one training example. In other words, one row on the table I drew just now what would be what I call one training example, and the Ith training example, in other words the Ith row in that table, I'm going to write as XI, Y, I.

Okay, and so in this notation they're going to use this superscript I is not exponentiation. So this is not X to the power of IY to the power of I. In this notation, the superscript I in parentheses is just sort of an index into the Ith row of my list of training examples.

So in supervised learning, this is what we're going to do. We're given a training set, and we're going to feed our training set, comprising our M training example, so 47 training examples, into a learning algorithm. Okay, and our algorithm then has output function that is by tradition, and for historical reasons, is usually denoted lower case alphabet H, and is called a hypothesis. Don't worry too much about whether the term hypothesis has a deep meaning. It's more a term that's used for historical reasons.

And the hypothesis's job is to take this input. There's some new [inaudible]. What the hypothesis does is it takes this input, a new living area in square feet saying and output estimates the price of this house. So the hypothesis H maps from inputs X to outputs Y. So in order to design a learning algorithm, the first thing we have to decide is how we want to represent the hypothesis, right.

And just for this purposes of this lecture, for the purposes of our first learning algorithm, I'm going to use a linear representation for the hypothesis. So I'm going to represent my hypothesis as H of X equals theta zero, plus theta 1X, where X here is an input feature, and so that's the size of the house we're considering.

And more generally, come back to this, more generally for many regression problems we may have more than one input feature. So for example, if instead of just knowing the size of the houses, if we know also the number of bedrooms in these houses, let's say, then, so if our training set also has a second feature, the number of bedrooms in the house, then you may, let's say X1 denote the size and square feet. Let X have script two denote the number of bedrooms, and then I would write the hypothesis, H of X, as theta rho plus theta 1X1 plus theta 2X2.

Okay, and sometimes when I went to take the hypothesis H, and when I went to make this dependent on the theta is explicit, I'll sometimes write this as H subscript theta of X. And so this is the price that my hypothesis predicts a house with features X costs. So given the new house with features X, a certain size and a certain number of bedrooms, this is going to be the price that my hypothesis predicts this house is going to cost.

One final piece of notation, so for conciseness, just to write this a bit more compactly I'm going to take the convention of defining X0 to be equal to one, and so I can now write H of X to be equal to sum from I equals one to two of theta I, oh sorry, zero to two, theta I, X I. And if you think of theta as an X, as vectors, then this is just theta transpose X.

And the very final piece of notation is I'm also going to let lower case N be the number of features in my learning problem. And so this actually becomes a sum from I equals zero to N, where in this example if you have two features, N would be equal to two.

All right, I realize that was a fair amount of notation, and as I proceed through the rest of the lecture today, or in future weeks as well, if some day you're looking at me write a symbol and you're wondering, gee, what was that simple lower case N again? Or what was that lower case X again, or whatever, please raise hand and I'll answer. This is a fair

amount of notation. We'll probably all get used to it in a few days and we'll standardize notation and make a lot of our descriptions of learning algorithms a lot easier.

But again, if you see me write some symbol and you don't quite remember what it means, chances are there are others in this class who've forgotten too. So please raise your hand and ask if you're ever wondering what some symbol means. Any questions you have about any of this?

Yeah?

Student: The variable can be anything? [Inaudible]?

Instructor (Andrew Ng):Say that again.

Student:[inaudible] zero theta one?

Instructor (Andrew Ng):Right, so, well let me – this was going to be next, but the theta or the theta Is are called the parameters. The thetas are called the parameters of our learning algorithm and theta zero, theta one, theta two are just real numbers. And then it is the job of the learning algorithm to use the training set to choose or to learn appropriate parameters theta.

Okay, is there other questions?

Student: What does [inaudible]?

Instructor (Andrew Ng):Oh, transpose. Oh yeah, sorry. When [inaudible] theta and theta transpose X, theta [inaudible].

Student: Is this like a [inaudible] hypothesis [inaudible], or would you have higher orders? Or would theta [inaudible]?

Instructor (Andrew Ng):All great questions. The answer – so the question was, is this a typical hypothesis or can theta be a function of other variables and so on. And the answer is sort of yes. For now, just for this first learning algorithm we'll talk about using a linear hypothesis class. A little bit actually later this quarter, we'll talk about much more complicated hypothesis classes, and we'll actually talk about higher order functions as well, a little bit later today.

Okay, so for the learning problem then. How do we chose the parameters theta so that our hypothesis H will make accurate predictions about all the houses. All right, so one reasonable thing to do seems to be, well, we have a training set. So – and just on the training set, our hypothesis will make some prediction, predictions of the housing prices, of the prices of the houses in the training set.

So one thing we could do is just try to make the predictions of a learning algorithm accurate on a training set. So given some features, X, and some correct prices, Y, we might want to make that theta square difference between the prediction of the algorithm and the actual price [inaudible].

So to choose parameters theta, unless we want to minimize over the parameters theta, so the squared area between the predicted price and the actual price. And so going to fill this in. We have M training examples. So the sum from I equals one through M of my M training examples, of price predicted on the Ith house in my training set. Mine is the actual target variable. Mine is actual price on the Ith training example.

And by convention, instead of minimizing this sum of the squared differences, I'm just going to put a one-half there, which will simplify some of the math we do later. Okay, and so let me go ahead and define J of theta to be equal to just the same, one-half sum from I equals one through M on the number of training examples, of the value predicted by my hypothesis minus the actual value.

And so what we'll do, let's say, is minimize as a function of the parameters of theta, this quantity J of theta. I should say, to those of you who have taken sort of linear algebra classes, or maybe basic statistics classes, some of you may have seen things like these before and seen least [inaudible] regression or [inaudible] squares.

Many of you will not have seen this before. I think some of you may have seen it before, but either way, regardless of whether you've seen it before, let's keep going. Just for those of you that have seen it before, I should say eventually, we'll actually show that this algorithm is a special case of a much broader class of algorithms. But let's keep going. We'll get there eventually.

So I'm going to talk about a couple of different algorithms for performing that minimization over theta of J of theta. The first algorithm I'm going to talk about is a search algorithm, where the basic idea is we'll start with some value of my parameter vector theta. Maybe initialize my parameter vector theta to be the vector of all zeros, and excuse me, have to correct that. I sort of write zero with an arrow on top to denote the vector of all zeros.

And then I'm going to keep changing my parameter vector theta to reduce J of theta a little bit, until we hopefully end up at the minimum with respect to theta of J of theta. So switch the laptops please, and lower the big screen. So let me go ahead and show you an animation of this first algorithm for minimizing J of theta, which is an algorithm called grading and descent.

So here's the idea. You see on the display a plot and the axes, the horizontal axes are theta zero and theta one. That's usually – minimize J of theta, which is represented by the height of this plot. So the surface represents the function J of theta and the axes of this function, or the inputs of this function are the parameters theta zero and theta one, written down here below.

So here's the gradient descent algorithm. I'm going to choose some initial point. It could be vector of all zeros or some randomly chosen point. Let's say we start from that point denoted by the star, by the cross, and now I want you to imagine that this display actually shows a 3D landscape. Imagine you're all in a hilly park or something, and this is the 3D shape of, like, a hill in some park.

So imagine you're actually standing physically at the position of that star, of that cross, and imagine you can stand on that hill, right, and look all 360 degrees around you and ask, if I were to take a small step, what would allow me to go downhill the most? Okay, just imagine that this is physically a hill and you're standing there, and would look around ask, "If I take a small step, what is the direction of steepest descent, that would take me downhill as quickly as possible?"

So the gradient descent algorithm does exactly that. I'm going to take a small step in this direction of steepest descent, or the direction that the gradient turns out to be. And then you take a small step and you end up at a new point shown there, and it would keep going. You're now at a new point on this hill, and again you're going to look around you, look all 360 degrees around you, and ask, "What is the direction that would take me downhill as quickly as possible?"

And we want to go downhill as quickly as possible, because we want to find the minimum of J of theta. So you do that again. You can take another step, okay, and you sort of keep going until you end up at a local minimum of this function, J of theta. One property of gradient descent is that where you end up – in this case, we ended up at this point on the lower left hand corner of this plot.

But let's try running gradient descent again from a different position. So that was where I started gradient descent just now. Let's rerun gradient descent, but using a slightly different initial starting point, so a point slightly further up and further to the right. So it turns out if you run gradient descent from that point, then if you take a steepest descent direction again, that's your first step.

And if you keep going, it turns out that with a slightly different initial starting point, you can actually end up at a completely different local optimum. Okay, so this is a property of gradient descent, and we'll come back to it in a second. So be aware that gradient descent can sometimes depend on where you initialize your parameters, theta zero and theta one.

Switch back to the chalkboard, please. Let's go ahead and work out the math of the gradient descent algorithm. Then we'll come back and revisit this issue of local optimum. So here's the gradient descent algorithm.

We're going to take a repeatedly take a step in the direction of steepest descent, and it turns out that you can write that as [inaudible], which is we're going to update the parameters theta as theta I minus the partial derivative with respect to theta I, J of Theta. Okay, so this is how we're going to update the I parameter, theta I, how we're going to update Theta I on each iteration of gradient descent.

Just a point of notation, I use this colon equals notation to denote setting a variable on the left hand side equal to the variable on the right hand side. All right, so if I write A colon equals B, then what I'm saying is, this is part of a computer program, or this is part of an algorithm where we take the value of B, the value on the right hand side, and use that to overwrite the value on the left hand side.

In contrast, if I write A equals B, then this is an assertion of truth. I'm claiming that the value of A is equal to the value of B, whereas this is computer operation where we overwrite the value of A. If I write A equals B then I'm asserting that the values of A and B are equal.

So let's see, this algorithm sort of makes sense – well, actually let's just move on. Let's just go ahead and take this algorithm and apply it to our problem. And to work out gradient descent, let's take gradient descent and just apply it to our problem, and this being the first somewhat mathematical lecture, I'm going to step through derivations much more slowly and carefully than I will later in this quarter. We'll work through the steps of these in much more detail than I will later in this quarter.

Let's actually work out what this gradient descent rule is. So – and I'll do this just for the case of, if we had only one training example. Okay, so in this case we need to work out what the partial derivative with respect to the parameter theta I of J of theta. If we have only one training example then J of theta is going to be one-half of script theta, of X minus Y, script. So if you have only one training example comprising one pair, X, Y, then this is what J of theta is going to be.

And so taking derivatives, you have one-half something squared. So the two comes down. So you have two times one-half times theta of X minus Y, and then by the [inaudible] derivatives, we also must apply this by the derivative of what's inside the square. Right, the two and the one-half cancel. So this leaves [inaudible] times that, theta zero, X zero plus [inaudible].

Okay, and if you look inside this sum, we're taking the partial derivative of this sum with respect to the parameter theta I. But all the terms in the sum, except for one, do not depend on theta I. In this sum, the only term that depends on theta I will be some term here of theta I, X I. And so we take the partial derivative with respect to theta I, X I – take the partial derivative with respect to theta I of this term theta I, X I, and so you get that times X I.

Okay, and so this gives us our learning rule, right, of theta I gets updated as theta I minus alpha times that. Okay, and this Greek alphabet alpha here is a parameter of the algorithm called the learning rate, and this parameter alpha controls how large a step you take. So you're standing on the hill. You decided what direction to take a step in, and so this parameter alpha controls how aggressive – how large a step you take in this direction of steepest descent.

And so if you – and this is a parameter of the algorithm that's often set by hand. If you choose alpha to be too small than your steepest descent algorithm will take very tiny steps and take a long time to converge. If alpha is too large then the steepest descent may actually end up overshooting the minimum, if you're taking too aggressive a step.

Yeah?

Student:[Inaudible].

Instructor (Andrew Ng):Say that again?

Student: Isn't there a one over two missing somewhere?

Instructor (Andrew Ng):Is there a one-half missing?

Student:I was [inaudible].

Instructor (Andrew Ng):Thanks. I do make lots of errors in that. Any questions about this?

All right, so let me just wrap this property into an algorithm. So over there I derived the algorithm where you have just one training example, more generally for M training examples, gradient descent becomes the following. We're going to repeat until convergence the following step.

Okay, theta I gets updated as theta I and I'm just writing out the appropriate equation for M examples rather than one example. Theta I gets updated. Theta I minus alpha times the sum from I equals one to M. Okay, and I won't bother to show it, but you can go home and sort of verify for yourself that this summation here, this is indeed the partial derivative with respect to theta I of J of theta, where if you use the original definition of J of theta for when you have M training examples.

Okay, so I'm just going to show – switch back to the laptop display. I'm going to show you what this looks like when you run the algorithm. So it turns out that for the specific problem of linear regression, or ordinary release squares, which is what we're doing today, the function J of theta actually does not look like this nasty one that I'll show you just now with a multiple local optima.

In particular, it turns out for ordinary release squares, the function J of theta is – it's just a quadratic function. And so we'll always have a nice bow shape, like what you see up here, and only have one global minimum with no other local optima.

So when you run gradient descent, here are actually the contours of the function J. So the contours of a bow shaped function like that are going to be ellipses, and if you run gradient descent on this algorithm, here's what you might get. Let's see, so I initialize the

parameters. So let's say randomly at the position of that cross over there, right, that cross on the upper right.

And so after one iteration of gradient descent, as you change the space of parameters, so if that's the result of one step of gradient descent, two steps, three steps, four steps, five steps, and so on, and it, you know, converges easily, rapidly to the global minimum of this function J of theta.

Okay, and this is a property of [inaudible] regression with a linear hypothesis cost. The function, J of theta has no local optima. Yes, question?

Student: Is the alpha changing every time? Because the step is not [inaudible].

Instructor (Andrew Ng):So it turns out that – yes, so it turns out – this was done with a – this is with a fake value of alpha, and one of the properties of gradient descent is that as you approach the local minimum, it actually takes smaller and smaller steps so they'll converge. And the reason is, the update is – you update theta by subtracting from alpha times the gradient. And so as you approach the local minimum, the gradient also goes to zero.

As you approach the local minimum, at the local minimum the gradient is zero, and as you approach the local minimum, the gradient also gets smaller and smaller. And so gradient descent will automatically take smaller and smaller steps as you approach the local minimum. Make sense?

And here's the same plot – here's actually a plot of the housing prices data. So here, lets you initialize the parameters to the vector of all zeros, and so this blue line at the bottom shows the hypothesis with the parameters of initialization. So initially theta zero and theta one are both zero, and so your hypothesis predicts that all prices are equal to zero.

After one iteration of gradient descent, that's the blue line you get. After two iterations, three, four, five, and after a few more iterations, excuse me, it converges, and you've now found the least square fit for the data. Okay, let's switch back to the chalkboard. Are there questions about this? Yeah?

Student:[Inaudible] iteration, do we mean that we run each sample – all the sample cases [inaudible] the new values?

Instructor (Andrew Ng):Yes, right.

Student: And converged means that the value will be the same [inaudible] roughly the same?

Instructor (Andrew Ng):Yeah, so this is sort of a question of how do you test the convergence. And there's different ways of testing for convergence. One is you can look

at two different iterations and see if theta has changed a lot, and if theta hasn't changed much within two iterations, you may say it's sort of more or less converged.

Something that's done maybe slightly more often is look at the value of J of theta, and if J of theta – if the quantity you're trying to minimize is not changing much anymore, then you might be inclined to believe it's converged. So these are sort of standard heuristics, or standard rules of thumb that are often used to decide if gradient descent has converged.

Yeah?

Student:I may have missed something, but especially in [inaudible] descent. So one feature [inaudible] curve and can either go this way or that way. But the math at incline [inaudible] where that comes in. When do you choose whether you go left, whether you're going this way or that way?

Instructor (Andrew Ng):I see. It just turns out that – so the question is, how is gradient descent looking 360 around you and choosing the direction of steepest descent. So it actually turns out – I'm not sure I'll answer the second part, but it turns out that if you stand on the hill and if you – it turns out that when you compute the gradient of the function, when you compute the derivative of the function, then it just turns out that that is indeed the direction of steepest descent.

By the way, I just want to point out, you would never want to go in the opposite direction because the opposite direction would actually be the direction of steepest ascent, right. So as it turns out – maybe the TAs can talk a bit more about this at the section if there's interest. It turns out, when you take the derivative of a function, the derivative of a function sort of turns out to just give you the direction of steepest descent.

And so you don't explicitly look all 360 degrees around you. You sort of just compute the derivative and that turns out to be the direction of steepest descent. Yeah, maybe the TAs can talk a bit more about this on Friday.

Okay, let's see, so let me go ahead and give this algorithm a specific name. So this algorithm here is actually called batch gradient descent, and the term batch isn't a great term, but the term batch refers to the fact that on every step of gradient descent you're going to look at your entire training set. You're going to perform a sum over your M training examples.

So [inaudible] descent often works very well. I use it very often, and it turns out that sometimes if you have a really, really large training set, imagine that instead of having 47 houses from Portland, Oregon in our training set, if you had, say, the U.S. Census Database or something, with U.S. census size databases you often have hundreds of thousands or millions of training examples.

So if M is a few million then if you're running batch rate and descent, this means that to perform every step of gradient descent you need to perform a sum from J equals one to a

million. That's sort of a lot of training examples where your computer programs have to look at, before you can even take one step downhill on the function J of theta.

So it turns out that when you have very large training sets, you should write down an alternative algorithm that is called [inaudible] gradient descent. Sometimes I'll also call it incremental gradient descent, but the algorithm is as follows. Again, it will repeat until convergence and will iterate for J equals one to M, and will perform one of these sort of gradient descent updates using just the J training example.

Oh, and as usual, this is really – you update all the parameters data runs. You perform this update for all values of I. For I indexes and the parameter vectors, you just perform this update, all of your parameters simultaneously. And the advantage of this algorithm is that in order to start learning, in order to start modifying the parameters, you only need to look at your first training examples.

You should look at your first training example and perform an update using the derivative of the error with respect to just your first training example, and then you look at your second training example and perform another update. And you sort of keep adapting your parameters much more quickly without needing to scan over your entire U.S. Census database before you can even start adapting parameters.

So let's see, for launch data sets, so constantly gradient descent is often much faster, and what happens is that constant gradient descent is that it won't actually converge to the global minimum exactly, but if these are the contours of your function, then after you run the constant gradient descent, you sort of tend to wander around.

And you may actually end up going uphill occasionally, but your parameters will sort of tender to wander to the region closest to the global minimum, but sort of keep wandering around a little bit near the region of the global [inaudible]. And often that's just fine to have a parameter that wanders around a little bit the global minimum. And in practice, this often works much faster than back gradient descent, especially if you have a large training set.

Okay, I'm going to clean a couple of boards. While I do that, why don't you take a look at the equations, and after I'm done cleaning the boards, I'll ask what questions you have.

Okay, so what questions do you have about all of this?

Student:[Inaudible] is it true – are you just sort of rearranging the order that you do the computation? So do you just use the first training example and update all of the theta Is and then step, and then update with the second training example, and update all the theta Is, and then step? And is that why you get sort of this really – ?

Instructor (Andrew Ng):Let's see, right. So I'm going to look at my first training example and then I'm going to take a step, and then I'm going to perform the second

gradient descent updates using my new parameter vector that has already been modified using my first training example. And then I keep going.

Make sense? Yeah?

Student: So in each update of all the theta Is, you're only using –

Instructor (Andrew Ng):One training example.

Student: One training example.

Student:[Inaudible]?

Instructor (Andrew Ng):Let's see, it's definitely a [inaudible]. I believe this theory that sort of supports that as well. Yeah, the theory that supports that, the [inaudible] of theorem is, I don't remember.

Okay, cool. So in what I've done so far, I've talked about an iterative algorithm for performing this minimization in terms of J of theta. And it turns out that there's another way for this specific problem of least squares regression, of ordinary least squares. It turns out there's another way to perform this minimization of J of theta that allows you to solve for the parameters theta in close form, without needing to run an iterative algorithm.

And I know some of you may have seen some of what I'm about to do before, in like an undergraduate linear algebra course, and the way it's typically done requires [inaudible] projections, or taking lots of derivatives and writing lots of algebra. What I'd like to do is show you a way to derive the closed form solution of theta in just a few lines of algebra.

But to do that, I'll need to introduce a new notation for matrix derivatives, and it turns out that, sort of, the notation I'm about to define here just in my own personal work has turned out to be one of the most useful things that I actually use all the time, to have a notation of how to take derivatives with respect to matrixes, so that you can solve for the minimum of J of theta with, like, a few lines of algebra rather than writing out pages and pages of matrices and derivatives.

So then we're going to define this new notation first and then we'll go ahead and work out the minimization. Given a function J, since J is a function of a vector of parameters theta, right, I'm going to define the derivative of the gradient of J with respect to theta, as self of vector. Okay, and so this is going to be an N plus one dimensional vector. Theta is an n plus one dimensional vector with indices ranging from zero to N. And so I'm going to define this derivative to be equal to that.

Okay, and so we can actually rewrite the gradient descent algorithm as follows. This is batch gradient descent, and we write gradient descent as updating the parameter vector

theta – notice there's no subscript I now – updating the parameter vector theta as the previous parameter minus alpha times the gradient.

Okay, and so in this equation all of these quantities, theta, and this gradient vector, all of these are n plus one dimensional vectors. I was using the boards out of order, wasn't I? So more generally, if you have a function F that maps from the space of matrices A, that maps from, say, the space of N by N matrices to the space of real numbers. So if you have a function, F of A, where A is an N by N matrix.

So this function is matched from matrices to real numbers, the function that takes this input to matrix. Let me define the derivative with respect to F of the matrix A. Now, I'm just taking the gradient of F with respect to its input, which is the matrix. I'm going to define this itself to be a matrix.

Okay, so the derivative of F with respect to A is itself a matrix, and the matrix contains all the partial derivatives of F with respect to the elements of A. One more definition is if A is a square matrix, so if A is an n by n matrix, number of rows equals number of columns, let me define the trace of A to be equal to the sum of A's diagonal elements. So this is just sum over I of A, I, I.

For those of you that haven't seen this sort of operator notation before, you can think of trace of A as the trace operator applied to the square matrix A, but it's more commonly written without the parentheses. So I usually write trace of A like this, and this just means the sum of diagonal elements.

So here are some facts about the trace operator and about derivatives, and I'm just going to write these without proof. You can also have the TAs prove some of them in the discussion section, or you can actually go home and verify the proofs of all of these.

It turns out that given two matrices, A and B, the trace of the matrix A times B is equal to the trace of B, A. Okay, I'm not going to prove this, but you should be able to go home and prove this yourself without too much difficulty. And similarly, the trace of a product of three matrices, so if you can take the matrix at the end and cyclically permeate it to the front.

So trace of A times B, times C, is equal to the trace of C, A, B. So take the matrix C at the back and move it to the front, and this is also equal to the trace of B, C. Take the matrix B and move it to the front.

Okay, also, suppose you have a function F of A which is defined as a trace of A, B. Okay, so this is, right, the trace is a real number. So the trace of A, B is a function that takes this input of matrix A and output a real number. So then the derivative with respect to the matrix A of this function of trace A, B, is going to be B transposed. And this is just another fact that you can prove by yourself by going back and referring to the definitions of traces and matrix derivatives. I'm not going to prove it. You should work it out.

Okay, and lastly a couple of easy ones. The trace of A is equal to the trace of A transposed because the trace is just the sum of diagonal elements. And so if you transpose the matrix, the diagonal, then there's no change. And if lower case A is a real number, then the trace of a real number is just itself. So think of a real number as a one by one matrix. So the trace of a one by one matrix is just whatever that real number is.

And lastly, this is a somewhat tricky one. The derivative with respect to the matrix A of the trace of A, B, A, transpose C is equal to C, A, B plus C transposed A, B transposed. And I won't prove that either. This is sort of just algebra. Work it out yourself.

Okay, and so the key facts I'm going to use again about traces and matrix derivatives, I'll use five. Ten minutes. Okay, so armed with these things I'm going to figure out – let's try to come up with a quick derivation for how to minimize J of theta as a function of theta in closed form, and without needing to use an iterative algorithm.

So work this out. Let me define the matrix X. This is called the design matrix. To be a matrix containing all the inputs from my training set. So X 1 was the vector of inputs to the vector of features for my first training example. So I'm going to set X 1 to be the first row of this matrix X, set my second training example is in place to be the second variable, and so on.

And I have M training examples, and so that's going to be my design matrix X. Okay, this is defined as matrix capital X as follows, and so now, let me take this matrix X and multiply it by my parameter vector theta. This derivation will just take two or three sets. So X times theta – remember how matrix vector multiplication goes. You take this vector and you multiply it by each of the rows of the matrix.

So X times theta is just going to be X 1 transposed theta, dot, dot, dot, down to X M, transposed theta. And this is, of course, just the predictions of your hypothesis on each of your M training examples. Then we also defined the Y vector to be the vector of all the target values Y1 through YM in my training set. Okay, so Y vector is an M dimensional vector.

So X theta minus Y contained the math from the previous board, this is going to be, right, and now, X theta minus Y, this is a vector. This is an M dimensional vector in M training examples, and so I'm actually going to take this vector and take this inner product with itself.

Okay, so we call that if Z is a vector than Z transpose Z is just sum over I, ZI squared. Right, that's how you take the inner product of a vector with a sum. So you want to take this vector, X theta minus Y, and take the inner product of this vector with itself, and so that gives me sum from I equals one to M, H, F, X, I, minus Y squared. Okay, since it's just the sum of the squares of the elements of this vector.

And put a half there for the emphasis. This is our previous definition of J of theta. Okay, yeah?

Student:[Inaudible]?

Instructor (Andrew Ng):Yeah, I threw a lot of notations at you today. So M is the number of training examples and the number of training examples runs from one through M, and then is the feature vector that runs from zero through N. Does that make sense?

So this is the sum from one through M. It's sort of theta transpose X that's equal to sum from J equals zero through N of theta J, X, J. Does that make sense? It's the feature vectors that index from zero through N where X, zero is equal to one, whereas the training example is actually indexed from one through M.

So let me clean a few more boards and take another look at this, make sure it all makes sense. Okay, yeah?

Student:[Inaudible] the Y inside the parentheses, shouldn't that be [inaudible]?

Instructor (Andrew Ng):Oh, yes, thank you. Oh is that what you meant? Yes, thank you. Great, I training example. Anything else? Cool. So we're actually nearly done with this derivation. We would like to minimize J of theta with respect to theta and we've written J of theta fairly compactly using this matrix vector notation.

So in order to minimize J of theta with respect to theta, what we're going to do is take the derivative with respect to theta of J of theta, and set this to zero, and solve for theta. Okay, so we have derivative with respect to theta of that is equal to – I should mention there will be some steps here that I'm just going to do fairly quickly without proof.

So is it really true that the derivative of half of that is half of the derivative, and I already exchanged the derivative and the one-half. In terms of the answers, yes, but later on you should go home and look through the lecture notes and make sure you understand and believe why every step is correct. I'm going to do things relatively quickly here and you can work through every step yourself more slowly by referring to the lecture notes.

Okay, so that's equal to – I'm going to expand now this quadratic function. So this is going to be, okay, and this is just sort of taking a quadratic function and expanding it out by multiplying the [inaudible]. And again, work through the steps later yourself if you're not quite sure how I did that.

So this thing, this vector, vector product, right, this quantity here, this is just J of theta and so it's just a real number, and the trace of a real number is just itself.

Student:[Inaudible].

Instructor (Andrew Ng):Oh, thanks, Dan. Cool, great. So this quantity in parentheses, this is J of theta and it's just a real number. And so the trace of a real number is just the same real number. And so you can sort of take a trace operator without changing anything. And this is equal to one-half derivative with respect to theta of the trace of – by

the second permutation property of trace. You can take this theta at the end and move it to the front.

So this is going to be trace of theta times theta transposed, X transpose X minus derivative with respect to theta of the trace of – I'm going to take that and bring it to the – oh, sorry. Actually, this thing here is also a real number and the transpose of a real number is just itself. Right, so take the transpose of a real number without changing anything.

So let me go ahead and just take the transpose of this. A real number transposed itself is just the same real number. So this is minus the trace of, taking the transpose of that. Here's Y transpose X theta, then minus [inaudible] theta. Okay, and this last quantity, Y transpose Y. It doesn't actually depend on theta. So when I take the derivative of this last term with respect to theta, it's just zero. So just drop that term.

And lastly, well, the derivative with respect to theta of the trace of theta, theta transposed, X transpose X. I'm going to use one of the facts I wrote down earlier without proof, and I'm going to let this be A. There's an identity matrix there, so this is A, B, A transpose C, and using a rule that I've written down previously that you'll find in lecture notes, because it's still on one of the boards that you had previously, this is just equal to X transpose X theta.

So this is C, A, B, which is sort of just the identity matrix, which you can ignore, plus X transpose X theta where this is now C transpose C, again times the identity which we're going to ignore, times B transposed. And the matrix X transpose X is the metric, so C transpose is equal to C.

Similarly, the derivative with respect to theta of the trace of Y transpose theta X, this is the derivative with respect to matrix A of the trace of B, A and this is just X transpose Y. This is just B transposed, again, by one of the rules that I wrote down earlier. And so if you plug this back in, we find, therefore, that the derivative – wow, this board's really bad.

So if you plug this back into our formula for the derivative of J, you find that the derivative with respect to theta of J of theta is equal to one-half X transpose theta, plus X transpose X theta, minus X transpose Y, minus X transpose Y, which is just X transpose X theta minus X [inaudible].

Okay, so we set this to zero and we get that, which is called a normal equation, and we can now solve this equation for theta in closed form. That's X transpose X theta, inverse times X transpose Y. And so this gives us a way for solving for the least square fit to the parameters in closed form, without needing to use an [inaudible] descent.

Okay, and using this matrix vector notation, I think, I don't know, I think we did this whole thing in about ten minutes, which we couldn't have if I was writing out reams of

algebra. Okay, some of you look a little bit dazed, but this is our first learning hour. Aren't you excited? Any quick questions about this before we close for today?

Student:[Inaudible].

Instructor (Andrew Ng):Say that again.

Student: What you derived, wasn't that just [inaudible] of X?

Instructor (Andrew Ng):What inverse?

Student: Pseudo inverse.

Instructor (Andrew Ng):Pseudo inverse?

Student: Pseudo inverse.

Instructor (Andrew Ng):Yeah, I turns out that in cases, if X transpose X is not invertible, than you use the pseudo inverse minimized to solve this. But it turns out X transpose X is not invertible. That usually means your features were dependent. It usually means you did something like repeat the same feature twice in your training set. So if this is not invertible, it turns out the minimum is obtained by the pseudo inverses of the inverse.

If you don't know what I just said, don't worry about it. It usually won't be a problem. Anything else?

Student: On the second board [inaudible]?

Instructor (Andrew Ng):Let me take that off. We're running over. Let's close for today and if they're other questions, I'll take them after.

[End of Audio]

Duration: 79 minutes