GILBERT STRANG: OK, so this is an important day, and Friday was an important day. I hope you enjoyed Professor Sra's terrific lecture as much as I did. You probably saw me taking notes like mad for the section that's now to be written about stochastic gradient descent.

And he promised a theorem, if you remember, and there wasn't time. And so he was going to send it to me or still is going to send it to me. I'll report, I haven't got it yet, but I'll bring it to class, waiting to see, hopefully. And that will give us a chance to review stochastic gradient descent, the central algorithm of deep learning.

And then this today is about the central structure of deep neural nets. And some of you will know already how they're connected, what the function F, the learning function-- you could call it the learning function-- that's constructed. The whole system is aiming at constructing this function F which learns the training data and then applying it to the test data. And the miracle is that it does so well in practice. That's what has transformed deep learning into such an important application.

Chapter 7 has been up for months on the math.mit.edu/learningfromdata site, and I'll add it to Stellar. Of course, that's where you'll be looking for it. OK, and then the second, the back propagation, the way to compute the gradient, I'll probably reach that idea today. And you'll see it's the chain rule, but how is it organized.

OK, so what's the structure? What's the plan for deep neural nets? Good. Starting here, so what we have is training data. So we have vectors, x1 to x-- what should I use for the number of samples that we have in the training data? Well, let's say D for Data. OK.

And each vector, those are called feature vectors, so equals feature vectors. So each one, each x, has like m features. So maybe my notation isn't so hot here. I have a whole lot of vectors. Let me not use the subscript for those right away.

So vectors, feature vectors, and each vector has got maybe shall we say m features? Like, if

we were measuring height and age and weight and so on, those would be features. The job of the neural network is to create-- and we're going to classify. Maybe we're going to classify men and women or boys and girls. So let's make it a classification problem, just binary.

So the classification problem is-- what shall we say? Minus 1 or 1, or 0 or 1, or boy or girl, or cat or dog, or truck or car, or anyway, just two classes. So I'm just going to do two-class classification. We know which class the training data is in. For each vector x, we know the right answer.

So we want to create a function that gives the right answer, and then we'll use that function on other data. So let me write that down. Create a function F of x so that gets-- most of gets the class correct. In other words, F of x should be negative for when the classification is minus 1, and F of x should be positive when the classification is plus 1.

And as we know, we don't necessarily have to get every x, every sample, right. That may be over-fitting. If there's some sample that's just truly weird, by getting that right we're going to be looking for a truly weird data in the test set, and that's not a good idea. We're trying to discover the rule that covers almost all cases but not every crazy, weird case. OK? So that's our job, to create a function F of x that is correct on almost all of the training data. Yeah.

So before I draw the picture of the network, let me just remember to mention the site Playground. I don't know if you've looked at that, so I'm going to ask you, playground@tensorflow.org. How many of you know that site or have met with it? Just a few, OK. OK, so it's not a very sophisticated site. It's got only four examples, four examples.

So one example is a whole lot of points that are blue, B for Blue, inside a bunch of points that are another set that are O for Orange, orange, blue. OK, so those are the two classes, orange and blue. So the points x and the feature vector here is just the xy, the coordinates features are the xy coordinates of the points. And our job is to find a function that's positive on these points and negative on those points.

So there is a simple model problem, and I recommend-- well, just partly-- if you're a expert in deep learning. This is for children, but morally here, I certainly learned from playing in this playground. So you set the step size. Do you set it, or does it set it? I guess you can change it. I don't think I've changed it.

What else do you set? Oh, you set the nonlinear activation, the nonlinear activation function,

active I'll say function. And let me just go over here and say what function people now mostly use.

The activation function is called ReLU pronounced different ways. I don't know how we got into that crazy thing. For this function, that is 0 and x. So the function, ReLU is a function of x is the maximum the larger of 0 and x.

The point is, it's not linear, and the point is that if we didn't allow nonlinearity in here somewhere, we couldn't even solve this playground problem. Because if our classifiers were all linear classifiers, like support vector machines, I couldn't separate the blue from the orange with a plane. It's got to somehow create some nonlinear function which maybe the function is trying to be-- a good function would be a function of r and theta maybe, maybe r minus 5.

So maybe the distance out to that. Let's suppose that distance is 5. Then, r minus 5 will be negative on the blues, because r is small. And r minus 5 will be positive on the oranges, because r is bigger. And therefore, we will have the right signs, less than 0 or greater than 0, and it'll classify this data, this training data. Yeah.

So it has to do that. This is not a hard one to do. There are four examples, as I say, two are trivial. It finds a good function. Well yeah, I've forgotten, they're so trivial, they shouldn't be mentioned, and then this is the medium test.

And then the hard test is when you have a sort of spiral of oranges, and inside, you have a spiral of blues. That was cooked up by a fiend. So the system is trying to find a function that's positive on one spiral and negative on the other spiral, and that takes quite a bit of time, many, many epochs.

I learned what an epoch is. Did you know what an epoch is? I didn't know whether it was just a fancy word for counting the steps in gradient descent. But it counts the steps, all right, but one epoch is the number of steps that matches the size of the training data. So if you have a million samples-- where ordinary gradient descent you would be doing a million-- you'd have a million by a million problem per step.

Of course, stochastic gradient descent just does a mini-batch of 1 or 32 or something, but anyway. So you have to do it enough mini-batches so that the total number you've covered is the equivalent of one full run through the training data, and that was an interesting point. Did you pick up that point? That in stochastic gradient descent, you could either do a mini-batch,

and then put them back in the soup, so with replacement. Or you could just put your data in some order, from one to a zillion.

So here's a first x and then more and more x's, and then just randomize the order. So you'd have to randomize the order for stochastic gradient descent to be reasonable, and then take a mini-batch and a mini-batch and a mini-batch and a mini-batch. And when you get to the bottom, you've finished one epoch. And then you'd probably randomize again, maybe, if you wanted to live right. And go through the mini-batches again, and probably do 1,000 times or more.

Anyway, so I haven't said yet what you do, what this F of x is like, but you can sort of see it on the screen. Because as it creates this function F, it kind of plots it. And what you see on the screen is the 0 set for that function. So perfect would be for it to go through 0-- if I had another color. Oh, I do have another color. Look, this is the first time the whole semester blue is up here.

OK, so if the function was positive there-- in this part, on the blues-- and negative outside that region for the oranges, that would be just what we want. Right? That would be what this little Playground site is creating. And on the screen, you'll see it. You'll see this curve, where it crosses 0.

So that curve, where it crosses 0, is supposed to separate the two sets. One set is positive, one set is negative, where 0 is in between. And the point is, it's not a straight line, because we've got this nonlinear function. This is nonlinear, and it allows us to have functions like r minus 5. And so at 5, that's where the function would be 0, and you'll see that on the screen.

You might just go to playground@tensorflow. Of course, TensorFlow is a big system. This is the child's department, but I thought it was pretty good. And then on this site, you decide how many layers there will be, how many neurons in each layer. So you create the structure that I'm about to draw.

And you won't be able to get to solve this problem to find a function F that learns that data without a number of layers and a number of neurons. If you don't give it enough, you'll see it struggling. The 0 set tries to follow this, but it gives up at some point. This one doesn't take too many layers, and the two trivial examples, just a few neurons do the job.

OK. So now, that's a little commented on one website. If you know other websites that I should

know and should call attention to, could you send me an email? I'm just not aware of everything that's out there.

Or if you know a good Convolutional Neural Net, CNN, that is available to practice on, where you could give it the training set. That's what I'm talking about here. I'd be glad to know, because I just don't know all that I should.

OK. So what does the function look like? Well, as I say, linear isn't going to do it, but linear is a very important part of it, of this function F. So the function F really has the form-- well, so we start here with a vector of one, two, three, four, m is five. This is the vector x, five components. OK, so let me erase that now.

OK, so then we have layer 1 with some number of points. Let's say, n1 is 6 neurons, and let me make this simple. I'll just have that one layer, and then I'll have the output. This will be the output layer, and it's just going to be one number.

So I'm going to have a matrix, A1, that takes me from this. A1 will be 6 by 5, because I want 6 outputs and 5 inputs. 6 by 5 matrix, so I have 30 weights to choose there. And so the y that comes out is going to be y1 will be A1 times x0. So x0 is the feature vector with 5 components.

So that's a purely linear thing, but we also want an offset function, offset vector. So that's a vector. Then, this, the y that's coming out, has 6 components. The A1 is 6 by 5, the x0 was 5 by 1, and then of course, this is 6 by 1. So these are the weights. Yeah, I'll call them all weights, weights to compute.

So these are connected. The usual picture is to show all these connections. I'll just put in some of them. So in here, we have 30 plus 6 parameters, 36 parameters, and then I'm going to close this. It's going to be a very shallow thing, so that will be just 1 by 6. Yeah. OK.

Right, so we're just getting one output. So that's just a vector at this final point, but of course, that the whole idea of deep neural nets is that you have many layers. So 36 more realistically is in the tens of thousands, and you have it multiple times. And the idea seems to be that you can separate what layer one learns about the data and from what layer two learns about the data.

Layer one-- this A1, apparently by just looking after the computation-- this learns some basic facts about the data. The next, A2 which would go in here, would learn more detail, and then A3 would learn more details. So we would have a number of layers, and it's that construction

that has made neural net successful. But I haven't finished, because right now, it's only linear. Right now, I just have, I'll call it A2 in here.

Right now, I would just have a matrix multiplication apply A1 and then apply A2, but in between there is a 1 by 1 action on each by this function. So that function acts on that number to give that number back again or to give 0. So in there is ReLU. In this comes ReLU on each, 6 copies of ReLU acting on each of those 6 numbers. Right? So really x1 comes from y1 by applying ReLU to it.

Then, that gives the x. So here are the y's from the linear part, and here are the x-- that's y1. That's a vector y1 from just the linear plus an affine map. Linear plus constant, that's affine. And then the next step is component by component we apply this function, and we get x1, and then do it again and again and again.

So do you see the function? How do I describe now the function F of x? So the learning function which depends on the weights, on the A's and b's. So I start with an x, I apply A1 to it. Yeah, let me do this.

This is the function F of x. F of x is going to be F3, let's say, of F2 of F1 of x, one, two, three, parentheses, right? OK, so it's a chain, you could say. F is a-- what's the right word for a chain of functions, if I take a function of a function? The reason I use the word chain is that the chain rule gives the derivative.

So a function of a function of a function, that's called composition, composing function. So this is a composition. I don't know if there's a standard symbol for starting with F1 and do some composition and do some composition. And now what are those separate F's? So the separate F's are the-- F1 of a vector would be-- it includes the ReLU part, the nonlinear part, of A1, x0 plus b1.

So two parts, you do the linear or affine map on your feature vector, and then component by component you apply that nonlinear function. And it took some years before that nonlinear function became a big favorite. People imagined that it was better, it was important, to have a smooth function. So the original functions were sigmoids, like S curves, but of course, it turned out that experiments showed that this worked even better.

Yeah, so that would be F1, and then F2 would have the same form, and F3 would have the same form. So maybe this had 36 weights, and the next one would have another number and

the next another number. You get quite complicated functions by composition, by like e to the sine of x, or e to the sign of the logarithm of x, or things like that. Pure math has asked, what functions can you get? Try to think of them all.

Now, what kind of functions do we have here? What can I say about F of x as a function, as a math person? What kind of a function is it? So it's created out of matrices and vectors, out of a linear or affine map, followed by a nonlinear, by that particular nonlinear function. So what kind of a function is it?

Well, I've written those words down up here, and F of x is going to be a continuous piecewise linear function. Because every step is continuous, that's a continuous function. Linear functions are a continuous functions, so we're taking a composition of continuous function, so it's continuous. And it's piecewise linear, because part of it is linear, and part of it is piecewise linear. So this is some continuous, piecewise, linear function of x, x in m dimensions. OK.

So one little math question which I think helps to understand, to like to swallow the idea of a chain, of the kind of chain we have here, of linear followed by ReLU. So here's my question. This is the question I'm going to ask. And by the way, back propagation is certainly going to come Wednesday rather than today. That's a major topic in itself.

So let me keep going with this function. Could you get any function whatsoever this way? Well, no, you only get continuous, piecewise, linear functions. It's an interesting case.

Let me just ask you. One of the exercises says, if I took two continuous, piecewise, linear functions-- the next 20 minutes are an attempt to give us a picture of the graph of a piecewise, linear function in say a function of two variables. So I have m equal to 2, and I draw its graph.

OK, help me to draw this graph. So this would be a graph of F of x1, x2, and it's going to be continuous and piecewise linear. So what does its graph look like? That's the question. What's the graph of a piecewise, linear function looks like?

Well, it's got flat pieces in between the change from-- I do say piecewise, that means it's got different pieces. But within a piece, it's linear, and the pieces with each other, because it's continuous. So I visualize, well, it's like origami. This is the theory of origami almost. So right, origami, you take a flat thing, and you fold it along straight folds.

So what's different from origami? Maybe not much. Well, maybe origami allows more than we

allow here, or origami would allow you to fold it up and over. So origami would give you a multi-valued thing, because it's got a top and a bottom and other folds. This is just going out to infinity in flat pieces, and the question will be, how many pieces?

So let me ask you that question. How many pieces do I have? Do you see what I mean by a piece? So I'm thinking of a graph that has these flat pieces, and they're connected along straight edges. And those straight edges come from the ReLU operation.

Well, that's got two pieces. Actually, we could do it 1D. In 1D, we could count the number of pieces pretty easily. So what would be a piecewise linear? Let me put it over here on the side and erase it soon. OK.

So here's m equal 1, a continuous, piecewise, linear F. I'll just draw its graph. So OK, so it's got straight pieces, straight pieces like so. Yeah, you've got the idea. It's a broken line type. Sometimes, people say broken line, but I'm never sure that's a good description of this.

Piecewise, linear, continuous, so it's continuous because the pieces meet, and it's piecewise, linear, obviously. OK, so that's the kind of picture I have for a function of one variable. Now, my question is-- as an aid to try to visualize this function in 2D-- is to see if we can count the pieces, see if we can count the pieces. Yes. So that's in the notes. I found it in a paper by five authors for a meeting. So actually, the whole world of neural nets, it's the conferences every couple of years that everybody prepares for, submits more than one paper.

So it's kind of a piecewise, linear conference, and those are the big conferences. OK. So this is the back propagation section, and I want to look at the-- OK. So this is a paper by Kleinberg and four others. Kleinberg, he's a computer science guy at Cornell. He was a PhD from here in math, and he's a very cool and significant person, not so much on neural networks as just this whole part of computer science. Right.

So anyway, they and other people too have asked this same problem. Suppose I'm in two variables. So what are you imagining now for the surface, the graph of F of x and y? It has these lines, fold lines, right? I'm thinking it has fold lines.

So I can start with a complete plane, and I fold it along one line. So now, it's like ReLU. It's one half plane there going into a different half plane there. Everybody with it? And now, I take that function, that surface which just has two parts, and I put in another fold.

OK, how many parts have I got now? I think four, am I right? Four parts, yes, because this will

be different from this, because it was folded along that line. So these will be four different pieces. They have the same value at the center there, and they match along the lines.

So the number of flat pieces is four for this. So that's with two folds, and now I just want to ask you, with m folds how many pieces are there? Can I get up to three folds? So I'm going to look for the number of folds.

So let me just use a notation, maybe r. r is the number of flat pieces, and m is the dimension of x. In my picture, it's two, and N is the number of folds.

So let me say it again. I'm taking a plane. I'll fold that plane-- because the dimension was two-- I'll fold it N times. How many pieces? How many flat pieces?

This would be a central step in understanding how close the function-- what freedom you have in the function F. For example, can you approximate any continuous function by one of these functions F by taking enough folds? Seems like the answer should be yes, and it is yes. For pure math, that's one question. Is this class of functions universal?

So the universality theorem would be to say that any function-- sine x, whatever-- could be approximated as close as you like by one of these guys with enough folds. And over here, we're kind of making it more numerical. We're going to count the number of pieces just to see how quickly do they grow.

So what happens here? So I have four folds. Right now, I have N equal 2. m is 2 here in this picture. And I'm trying to draw this surface, in here I've put in 2.

Did I take N? Yeah, two folds, and now I'm going to go up to three folds. OK. So let me fold it along that line. How many pieces of I got now?

Let's see, can I count those pieces? Is it seven? So what is a formula? What if I do another fold? Yeah, let's pretend we do another fold. Yeah?

AUDIENCE: [INAUDIBLE]

GILBERT STRANG: Uh, yeah. Well, maybe that's going to be it. It's a kind of nice question, because it asks you to visualize this thing. OK. So what happened? How many of those lines will be-- if I put in a fourth line-- how many? Yeah, how many new folds do I create?

That's kind of the question, and I'm assuming that fourth line doesn't go through any of these

points. It's sort of in general position. So I put it in a fourth line, da-da-da-da, there it is. OK, so what happened here? How many new ones did it create?

How many new ones did it create? Let me make that one green, because I'm distinguishing that's the guy that's added after the original. We had seven. We had seven pieces, and now we've got more. Was it seven?

It was, wasn't it? One, two, three, four, five, six, seven, but now how many pieces have I got? Or how many pieces did this new line create? We want to build it up, use a recursion. How many pieces did this new-- well, this new line created one new piece there. Right? One new piece there, one new piece there, one new piece there, so there are four new pieces. OK.

Yes, so there's some formula that's going to tell us that, and now what would the next one create? Well, now I have one, two, three, four lines. So now, I'm going to put through a fifth line, and that will create a whole bunch of pieces. I'm losing the thread of this argument, but you're onto it. Right? Yeah, so any suggestions? Yeah.

AUDIENCE: Yeah, I think you add essentially the number of lines that you have each time you add a line at most.

GILBERT STRANG: OK. Yes. That's right. So there is a recursion formula that I want to know, and I learned it from Kleinberg's paper. And then we have an addition to do, so the recursion will tell me how much it goes up with each new function, and then we have to add. OK. So the recursion formula, let me write that down.

So this is r of N and m that I'd like to find a formula for. It's the number of flat pieces with an m dimensional surface-- well, we're taking m to be 2-- and N folds. So N equal 1, 2, 3. Let's write down the numbers we know.

With one fold, how many pieces? Two, good, so far so good. With one fold, there were two pieces. So this is the count r, and then with two folds, how many? Oh, we've gone past that point. So can we get back to just those two? Was it four?

AUDIENCE: Yes.

GILBERT STRANG: OK, thanks. Now, when I put in that third fold, how many did I have without the green line yet? Seven, was it seven? And when the fourth one went in, that green one, how many have I got in this picture? So the question is how many new ones did I create, I guess. So that line got

chopped into that piece, that piece, that piece, that piece, four pieces for the new line.

Four pieces for the new line, and then each of those pieces like added a flat bit. Because that piece from here to here separated these two which were previously just one piece, one flat piece. I folded on that line. I folded on this. I folded there. I think it went up by 4 to 11.

So now, we just have to guess a formula that matches those numbers, and then of course, we really should guess it for any m and any N. And I'll write down the formula that they found. It involves binomial numbers. Everything in the world involves binomial numbers, because they satisfy every identity you could think of.

So here's their formula. r with N folds, and we're in m dimensions. So we've really in our thinking had m equal to 2, but we should grow up and get m to be five dimensional. So we have a five dimensional-- let's not think about that. OK.

So it turns out it's binomial numbers-- N 0, N 1, up to N m. So for m equals 2, which is my picture, it's N 0 plus N 1 plus N 2, and what are these? What does that N 2 mean, for example? That's a binomial number. I don't know if you're keen on binomial numbers. Some people, their whole lives go into binomial numbers.

So it's something like-- is it N factorial divided by N minus 2 factorial and 2 factorial? I think that's what that number means. That's the binomial number. So at this point, I'm hoping to get the answer seven, I think. I'm in m equal to-- I've gone up to 2.

Yeah, so I think I've obviously allowed for three cuts, and the r, when we had just three, was 7. So this is now I'm taking N to be 3, and I'm hoping for answer is 7. So I add these three things. So what is 3, the binomial number 3 with 2? I've forgotten how to say that.

I'm ashamed to admit. 3 choose 2, thanks. I knew there was a good way. So what is 3 choose 2? Well, put in 3, and 2 is in there already, so that'd be 6 over 1 times 2. This would be 3. Would that be 3? And what is 3 choose 1?

AUDIENCE: 3.

GILBERT STRANG: How do you know that? You're probably right. 3, I think, yeah. Oh yeah, probably a theorem that if these add 3. Yeah, so I'm doing N equals 3 here.

OK. So yeah, I agree. That's 3, and what about N to 0? That you have to live with 0 factorial,

but 0 factorial is by no means 0. So what is 0 factorial? 1, yeah.

I remember when I was an undergraduate having a bet on that. I won, but he didn't pay off. Yeah, so it's 3. This is 3 factorial over 3 factorial times 0 factorial. So it's 6 over 6 times 1.

So it's 1. Yeah, 1 and 3 and 3 make 7. So that proves the formula. Well, it doesn't quite prove the formula, but the way to prove it is by an induction.

If you like this stuff, the recursion that you use induction on. Which is just what we did now, what we did here. Here comes in a number 4, and it cuts through, and then we just counted the 4 pieces there. So yeah, so let me just tell you what the r then and m.

The number we're looking for is the number that we had with one less cut. So that's the previous count of flat pieces plus the number that was here was 4, the number of pieces that cut that. And that's r of N minus 1, m minus 1. Yeah, and I won't go further.

Time's up, but that rule for recursion is proved in the section 7.1 taken from the paper by Kleinberg and others. Yeah. So OK, I think this is-- I don't know what you feel. For me, this like gave me a better feeling that I was understanding what kind of functions we had here. And so then the question is-- with this family of functions, we want to choose the A's and the weights, the A's and b's, to match the training data. So that we have a problem in minimizing the total loss, and we have a gradient descent problem.

So we have to find the gradient, so that Wednesday's job. Wednesday's job is to find the gradient of F, and that's back propagation. Good. Thank you very much. 7.1 is done.