PATRICK WINSTON: It was in 2010, yes, that's right. It was in 2010. We were having our annual discussion about what we would dump fro 6034 in order to make room for some other stuff. And we almost killed off neural nets. That might seem strange because our heads are stuffed with neurons. If you open up your skull and pluck them all out, you don't think anymore. So it would seem that neural nets would be a fundamental and unassailable topic.

But many of us felt that the neural models of the day weren't much in the way of faithful models of what actually goes on inside our heads. And besides that, nobody had ever made a neural net that was worth a darn for doing anything. So we almost killed it off. But then we said, well, everybody would feel cheated if they take a course in artificial intelligence, don't learn anything about neural nets, and then they'll go off and invent them themselves. And they'll waste all sorts of time. So we kept the subject in.

Then two years later, Jeff Hinton from the University of Toronto stunned the world with some neural network he had done on recognizing and classifying pictures. And he published a paper from which I am now going to show you a couple of examples. Jeff's neural net, by the way, had 60 million parameters in it. And its purpose was to determine which of 1,000 categories best characterized a picture.

So there it is. There's a sample of things that the Toronto neural net was able to recognize or make mistakes on. I'm going to blow that up a little bit. I think I'm going to look particularly at the example labeled container ship. So what you see here is that the program returned its best estimate of what it was ranked, first five, according to the likelihood, probability, or the certainty that it felt that a particular class was characteristic of the picture.

And so you can see this one is extremely confident that it's a container ship. It also was fairly moved by the idea that it might be a lifeboat. Now, I'm not sure about you, but I don't think this looks much like a lifeboat. But it does look like a container ship. So if I look at only the best choice, it looks pretty good.

Here are the other things they did pretty well, got the right answer is the first choice-- is this first choice. So over on the left, you see that it's decided that the picture is a picture of a mite. The mite is not anywhere near the center of the picture, but somehow it managed to find it-- the container ship again. There is a motor scooter, a couple of people sitting on it. But it correctly characterized the picture as a motor scooter. And then on the right, a Leopard. And everything else is a cat of some sort. So it seems to be doing pretty well. In fact, it does do pretty well.

But anyone who does this kind of work has an obligation to show you some of the stuff that doesn't work so well on or doesn't get quite right. And so these pictures also occurred in Hinton's paper. So the first one is characterized as a grill. But the right answer was supposed to be convertible. Oh, no, yes, yeah, right answer was convertible. In the second case, the characterization is of a mushroom. And the alleged right answer is agaric. Is that pronounced right? It turns out that's a kind of mushroom-- so no problem there.

In the next case, it said it was a cherry. But it was supposed to be a dalmatian. Now, I think a dalmatian is a perfectly legitimate answer for that particular picture-- so hard to fault it for that. And the last case, the correct answer was not in any of the top five. I'm not sure if you've ever seen a Madagascar cap. But that's a picture of one. And it's interesting to compare that with the first choice of the program, the squirrel monkey. This is the two side by side.

So in a way, it's not surprising that it thought that the Madagascar cat was a picture of a squirrel monkey-- so pretty impressive. It blew away the competition. It did so much better the second place wasn't even close. And for the first time, it demonstrated that a neural net could actually do something. And since that time, in the three years since that time, there's been an enormous amount of effort put into neural net technology, which some say is the answer.

So what we're going to do today and tomorrow is have a look at this stuff and ask ourselves why it works, when it might not work, what needs to be done, what has been done, and all those kinds of questions will emerge.

So I guess the first thing to do is think about what it is that we are being inspired by. We're being inspired by those things that are inside our head-- all 10 to the 11th of them. And so if we take one of those 10 to the 11th and look at it, you know from 700 something or other approximately what a neuron looks like. And by the way, I'm going to teach you in this lecture how to answer questions about neurobiology with an 80% probability that you will give the same answer as a neurobiologist. So let's go.

So here's a neuron. It's got a cell body. And there is a nucleus. And then out here is a long thingamajigger which divides maybe a little bit, but not much. And we call that the axon.

So then over here, we've got this much more branching type of structure that looks maybe a little bit like so. Maybe like that-- and this stuff branches a whole lot. And that part is called the dendritic tree.

Now, there are a couple of things we can note about this is that these guys are connected axon to dendrite. So over here, they'll be a so-called pre-synaptic thickening. And over here will be some other neuron's dendrite. And likewise, over here some other neuron's axon is coming in here and hitting the dendrite of our the one that occupies most of our picture.

So if there is enough stimulation from this side in the axonal tree, or the dendritic tree, then a spike will go down that axon. It acts like a transmission line. And then after that happens, the neuron will go quiet for a while as it's recovering its strength. That's called the refractory period.

Now, if we look at that connection in a little more detail, this little piece right here sort of looks like this. Here's the axon coming in. It's got a whole bunch of little vesicles in it. And then there's a dendrite over here. And when the axon is stimulated, it dumps all these vesicles into this inner synaptic space. For a long time, it wasn't known whether those things were actually separated. I think it was Raamon and Cahal who demonstrated that one neuron is actually not part of the next one. They're actually separated by these synaptic gaps.

So there it is. How can we model, that sort of thing? Well, here's what's usually done. Here's what is done in the neural net literature. First of all, we've got some kind of binary input, because these things either fire or they don't fire. So it's an all-or-none kind of situation. So over here, we have some kind of input value. We'll call it x1. And is either a 0 or 1.

So it comes in here. And then it gets multiplied times some kind of weight. We'll call it w1. So this part here is modeling this synaptic connection. It may be more or less strong. And if it's more strong, this weight goes up. And if it's less strong, this weight goes down. So that reflects the influence of the synapse on whether or not the whole axon decides it's stimulated.

Then we got other inputs down here-- x sub n, also 0 or 1. It's also multiplied by a weight. We'll call that w sub n. And now, we have to somehow represent the way in which these inputs are collected together-- how they have collective force. And we're going to model that very, very simply just by saying, OK, we'll run it through a summer like so.

But then we have to decide if the collective influence of all those inputs is sufficient to make the neuron fire. So we're going to do that by running this guy through a threshold box like so. Here is what the box looks like in terms of the relationship between input and the output. And what you can see here is that nothing happens until the input exceeds some threshold t.

If that happens, then the output z is a 1. Otherwise, it's a 0. So binary, binary out-- we model the synaptic weights by these multipliers. We model the cumulative effect of all that input to the neuron by a summer. We decide if it's going to be an all-or-none 1 by running it through this threshold box and seeing if the sum of the products add up to more than the threshold. If so, we get a 1.

So what, in the end, are we in fact modeling? Well, with this model, we have number 1, all or none-- number 2, cumulative influence-- number 3, oh, I, suppose synaptic weight. But that's not all that there might be to model in a real neuron. We might want to deal with the refractory period.

In these biological models that we build neural nets out of, we might want to model axonal bifurcation. We do get some division in the axon of the neuron. And it turns out that that pulse will either go down one branch or the other. And which branch it goes down depends on electrical activity in the vicinity of the division. So these things might actually be a fantastic coincidence detectors. But we're not modeling that. We don't know how it works. So axonal bifurcation might be modeled.

We might also have a look at time patterns. See, what we don't know is we don't know if the timing of the arrival of these pulses in the dendritic tree has anything to do with what that neuron is going to recognize-- so a lot of unknowns here. And now, I'm going to show you how to answer a question about neurobiology with 80% probability you'll get it right. Just say, we don't know. And that will be with 80% probability what the neurobiologist would say.

So this is a model inspired by what goes on in our heads. But it's far from clear if what we're modeling is the essence of why those guys make possible what we can do. Nevertheless, that's where we're going to start. That's where we're going to go. So we've got this model of what a neuron does.

So what about what does a collection of these neurons do? Well, we can think of your skull as a big box full of neurons. Maybe a better way to think of this is that your head is full of neurons. And they in turn are full of weights and thresholds like so. So into this box come a variety of inputs x1 through xm. And these find their way to the inside of this gaggle of neurons. And out here come a bunch of outputs c1 through zn. And there a whole bunch of these maybe like so. And there are a lot of inputs like so.

And somehow these inputs through the influence of the weights of the thresholds come out as a set of outputs. So we can write that down a little fancier by just saying that z is a vector, which is a function of, certainly the input vector, but also the weight vector and the threshold vector. So that's all a neural net is. And when we train a neural net, all we're going to be able to do is adjust those weights and thresholds so that what we get out is what we want. So a neural net is a function approximator. It's good to think about that. It's a function approximator.

So maybe we've got some sample data that gives us an output vector that's desired as another function of the input, forgetting about what the weights and the thresholds are. That's what we want to get out. And so how well we're doing can be figured out by comparing the desired value with the actual value.

So we might think then that we can get a handle on how well we're doing by constructing some performance function, which is determined by the desired vector and the input vector-- sorry, the desired vector and the actual output vector for some particular input or for some set of inputs. And the question is what should that function be? How should we measure performance given that we have what we want out here and what we actually got out here?

Well, one simple thing to do is just to measure the magnitude of the difference. That makes sense. But of course, that would give us a performance function that is a function of the distance between those vectors would look like this. But this turns out to be mathematically inconvenient in the end. So how do you think we're going to turn it up a little bit?

AUDIENCE: Normalize it?

PATRICK WINSTON: What's that?

AUDIENCE: Normalize it?

PATRICK WINSTON: Well, I don't know. How about just we square it? And that way we're going to go from this little sharp point down there to something that looks more like that. So it's best when the difference is 0, of course. And it gets worse as you move away from 0. But what we're trying to do here is we're trying to get to a minimum value. And I hope you'll forgive me. I just don't like the direction we're going here, because I like to think in terms of improvement as going uphill instead of down hill.

So I'm going to dress this up one more step-- put a minus sign out there. And then our performance function looks like this. It's always negative. And the best value it can possibly be is zero. So that's what we're going to use just because I am who I am. And it doesn't matter, right? Still, you're trying to either minimize or maximize some performance function.

OK, so what do we got to do? I guess what we could do is we could treat this thing-- well, we already know what to do. I'm not even sure why we're devoting our lecture to this, because it's clear that what we're trying to do is we're trying to take our weights and our thresholds and adjust them so as to maximize performance. So we can make a little contour map here with a simple neural net with just two weights in it. And maybe it looks like this-- contour map.

And at any given time we've got a particular w1 and particular w2. And we're trying to find a better w1 and w2. So here we are right now. And there's the contour map. And it's a 6034. So what do we do?

AUDIENCE: Climb.

PATRICK WINSTON: Simple matter of hill climbing, right? So we'll take a step in every direction. If we take a step in that direction, not so hot. That actually goes pretty bad. These two are really ugly. Ah, but that one-- that one takes us up the hill a little bit. So we're done, except that I just mentioned that Hinton's neural net had 60 million parameters in it. So we're not going to hill climb with 60 million parameters because it explodes exponentially in the number of weights you've got to deal with-- the number of steps you can take.

So this approach is computationally intractable. Fortunately, you've all taken 1801 or the equivalent thereof. So you have a better idea. Instead of just taking a step in every direction, what we're going to do is we're going to take some partial derivatives. And we're going to see what they suggest to us in terms of how we're going to get around in space.

So we might have a partial of that performance function up there with respect to w1. And we might also take a partial derivative of that guy with respect to w2. And these will tell us how much improvement we're getting by making a little movement in those directions, right? How much a change is given that we're just going right along the axis.

So maybe what we ought to do is if this guy is much bigger than this guy, it would suggest we mostly want to move in this direction, or to put it in 1801 terms, what we're going to do is we're going to follow the gradient. And so the change in the w vector is going to equal to this partial derivative times i plus this partial derivative times j. So what we're going to end up doing in this particular case by following that formula is moving off in that direction right up to the steepest part of the hill.

And how much we move is a question. So let's just have a rate constant R that decides how big our step is going to be. And now you think we were done. Well, too bad for our side. We're not done. There's a reason why we can't use-- create ascent, or in the case that I've drawn our gradient, descent if we take the performance function the other way. Why can't we use it?

AUDIENCE: Local maxima.

PATRICK WINSTON: The remark is local maxima. And that is certainly true. But it's not our first obstacle. Why doesn't gradient ascent work?

AUDIENCE: So you're using a step function.

PATRICK WINSTON: Ah, there's something wrong with our function. That's right. It's non-linear, but rather, it's discontinuous. So gradient ascent requires a continuous space, continuous surface. So too bad our side. It isn't. So what to do? Well, nobody knew what to do for 25 years. People were screwing around with training neural nets for 25 years before Paul Werbos sadly at Harvard in 1974 gave us the answer.

And now I want to tell you what the answer is. The first part of the answer is those thresholds are annoying. They're just extra baggage to deal with. What we really like instead of c being a function of xw and t was we'd like c prime to be a function f prime of x and the weights. But we've got to account for the threshold somehow. So here's how you do that.

What you do is you say let us add another input to this neuron. And it's going to have a weight w0. And it's going to be connected to an input that's always minus 1. You with me so far? Now what we're going to do is we're going to say, let w0 equal t. What does that do to the movement of the threshold? What it does is it takes that threshold and moves it back to 0. So this little trick here takes this pink threshold and redoes it so that the new threshold box looks like this.

Think about it. If this is t, and this is minus 1, then this is minus t. And so this thing ought to fire if everything's over-- if the sum is over 0. So it makes sense. And it gets rid of the threshold thing for us. So now we can just think about weights. But still, we've got that step function there. And that's not good.

So what we're going to do is we're going to smooth that guy out. So this is trick number two. Instead of a step function, we're going to have this thing we lovingly call a sigmoid function, because it's kind of from an s-type shape. And the function we're going to use is this one-- one, well, better make it a little bit different-- 1 over 1 plus e to the minus whatever the input is. Let's call the input alpha. Does that makes sense?

Is alpha is 0, then it's 1 over 1 plus 1 plus one half. If alpha is extremely big, then even the minus alpha is extremely small. And it becomes one. It goes up to an asymptotic value of one here. On the other hand, if alpha is extremely negative, than the minus alpha is extremely positive. And it goes to 0 asymptotically.

So we got the right look to that function. It's a very convenient function. Did God say that neurons ought to be-- that threshold ought to work like that? No, God didn't say so. Who said so? The math says so. It has the right shape and look and the math. And it turns out to have the right math, as you'll see in a moment.

So let's see. Where are we? We decided that what we'd like to do is take these partial derivatives. We know that it was awkward to have those thresholds. So we got rid of them. And we noted that it was impossible to have the step function. So we got rid of it. Now, we're a situation where we can actually take those partial derivatives, and see if it gives us a way of training the neural net so as to bring the actual output into alignment with what we desire.

So to deal with that, we're going to have to work with the world's simplest neural net. Now, if we've got one neuron, it's not a net. But if we've got two-word neurons, we've got a net. And it turns out that's the world's simplest neuron. So we're going to look at it-- not 60 million parameters, but just a few, actually, just two parameters.

So let's draw it out. We've got input x. That goes into a multiplier. And it gets multiplied times w1. And that goes into a sigmoid box like so. We'll call this p1, by the way, product number one. Out here comes y. Y gets multiplied times another weight. We'll call that w2. The neck produces another product which we'll call p2. And that goes into a sigmoid box.

And then that comes out as z. And z is the number that we use to determine how well we're doing. And our performance function p is going to be one half minus one half, because I like things are going in a direction, times the difference between the desired output and the actual output squared.

So now let's decide what those partial derivatives are going to be. Let me do it over here. So what are we trying to compute? Partial of the performance function p with respect to w2. OK. Well, let's see. We're trying to figure out how much this wiggles when we wiggle that.

But you know it goes through this variable p2. And so maybe what we could do is figure out how much this wiggles-- how much z wiggles when we wiggle p2 and then how much p2 wiggles when we wiggle w2. I just multiplied those together. I forget. What's that called? N180-- something or other.

AUDIENCE: The chain rule

PATRICK WINSTON: The chain rule. So what we're going to do is we're going to rewrite that partial derivative using chain rule. And all it's doing is saying that there's an intermediate variable. And we can compute how much that end wiggles with respect how much that end wiggles by multiplying how much the other guys wiggle. Let me write it down. It makes more sense in mathematics.

So that's going to be able to the partial of p with respect to z times the partial of z with respect to p2. Keep me on track here. Partial of z with respect to w2. Now, I'm going to do something for which I will hate myself. I'm going to erase something on the board. I don't like to do that. But you know what I'm going to do, don't you?

I'm going to say this is true by the chain rule. But look, I can take this guy here and screw around with it with the chain rule too. And in fact, what I'm going to do is I'm going to replace that with partial of z with respect to p2 and partial of p2 with respect to w2. So I didn't erase it after all. But you can see what I'm going to do next. Now, I'm going to do same thing with the other partial derivative. But this time, instead of writing down and writing over, I'm just going to expand it all out in one go, I think.

So partial of p with respect to w1 is equal to the partial of p with respect to z, the partial of z with respect to p2, the partial of p2 with respect to what? Y? Partial of y with respect to p1-- partial of p1 with respect to w1. So that's going like a zipper down that string of variables expanding each by using the chain rule until we got to the end. So there are some expressions that provide those partial derivatives.

But now, if you'll forgive me, it was convenient to write them out that way. That matched the intuition in my head. But I'm just going to turn them around. It's just a product. I'm just going to turn them around. So partial p2, partial w2, times partial of z, partial p2, times the partial of p with respect to z-- same thing. And now, this one. Keep me on track, because if there's a mutation here, it will be fatal.

Partial of p1-- partial of w1, partial of y, partial p1, partial of p2, partial of y, partial of z. There's a partial of p2, partial of a performance function with respect to z. Now, all we have to do is figure out what those partials are. And we have solved this simple neural net. So it's going to be easy.

Where is my board space? Let's see, partial of p2 with respect to-- what? That's the product. The partial of z-- the performance function with respect to z. Oh, now I can see why I wrote it down this way. Let's see. It's going to be d minus e. We can do that one in our head.

What about the partial of p2 with respect to w2. Well, p2 is equal to y times w2, so that's easy. That's just y. Now, all we have to do is figure out the partial of z with respect to p2. Oh, crap, it's going through this threshold box. So I don't know exactly what that partial derivative is. So we'll have to figure that out, right? Because the function relating them is this guy here.

And so we have to figure out the partial of that with respect to alpha. All right, so we got to do it. There's no way around it. So we have to destroy something. OK, we're going to destroy our neuron. So the function we're dealing with is, we'll call it beta, equal to 1 over 1 plus e to the minus alpha. And what we want is the derivative with respect to alpha of beta. And that's equal to d by d alpha of-- you know, I can never remember those quotient formulas.

So I am going to rewrite it a little different way. I am going to write it as 1 minus e to the minus alpha to the minus 1, because I can't remember the formula for differentiating a quotient. OK, so let's differentiate it. So that's equal to 1 minus e to the minus alpha to the minus 2. And we got that minus comes out of that part of it. Then we got to differentiate the inside of that expression.

And when we differentiate the inside of that expression, we get e to the minus alpha.

AUDIENCE: Dr. Winston--

PATRICK WINSTON: Yeah?

AUDIENCE: That should be 1 plus.

PATRICK WINSTON: Oh, sorry, thank you. That was one of those fatal mistakes you just prevented. So that's 1 plus. That's 1 plus here too. OK, so we've differentiated that. We've turned that into a minus 2. We brought the minus sign outside. Then we're differentiating the inside. The derivative and the exponential is an exponential. Then we got to differentiate that guy. And that just helps us get rid of the minus sign we introduced. So that's the derivative.

I'm not sure how much that helps except that I'm going to perform a parlor trick here and rewrite that expression thusly. We want to say that's going to be e to the minus alpha over 1 plus e to the minus alpha times 1 over 1 plus e to the minus alpha. That OK? I've got a lot of nodding heads here. So I think I'm on safe ground.

But now, I'm going to perform another parlor trick. I am going to add 1, which means I also have to subtract 1. All right? That's legitimate isn't it? So now, I can rewrite this as 1 plus e to the minus alpha over 1 plus e to the minus alpha minus 1 over 1 plus e to the minus alpha times 1 over 1 plus e to the minus alpha. Any high school kid could do that. I think I'm on safe ground.

Oh, wait, this is beta. This is beta.

AUDIENCE: That's the wrong side.

PATRICK WINSTON: Oh, sorry, wrong side. Better make this beta and this 1. Any high school kid could do it. OK, so what we've got then is that this is equal to 1 minus beta times beta. That's the derivative. And that's weird because the derivative of the output with respect to the input is given exclusively in terms of the output. It's strange. It doesn't really matter. But it's a curiosity.

And what we get out of this is that partial derivative there-- that's equal to well, the output is p2. No, the output is z. So it's z time 1 minus e. So whenever we see the derivative of one of these sigmoids with respect to its input, we can just write the output times one minus alpha, and we've got it. So that's why it's mathematically convenient. It's mathematically convenient because when we do this differentiation, we get a very simple expression in terms of the output.

We get a very simple expression. That's all we really need. So would you like to see a demonstration? It's a demonstration of the world's smallest neural net in action. Where is neural nets? Here we go.

So there's our neural net. And what we're going to do is we're going to train it to do absolutely nothing. What we're going to do is train it to make the output the same as the input. Not what I'd call a fantastic leap of intelligence. But let's see what happens.

Wow! Nothing's happening. Well, it finally got to the point where the maximum error, not the performance, but the maximum error went below a threshold that I had previously determined. So if you look at the input here and compare that with the desired output on the far right, you see it produces an output, which compared with the desired output, is pretty close.

So we can test the other way like so. And we can see that the desired output is pretty close to the actual output in that case too. And it took 694 iterations to get that done. Let's try it again.

To 823-- of course, this is all a consequence of just starting off with random weights. By the way, if you started with all the weights being the same, what would happen? Nothing because it would always stay the same. So you've got to put some randomization in in the beginning. So it took a long time. Maybe the problem is our rate constant is too small. So let's crank up the rate counts a little bit and see what happens. That was pretty fast. Let's see if it was a consequence of random chance. Run. No, it's pretty fast there-- 57 iterations-- third try-- 67. So it looks like at my initial rate constant was too small.

So if 0.5 was not as good as 5.0, why don't we crank it up to 50 and see what happens. Oh, in this case, 124-- let's try it again. Ah, in this case 117-- so it's actually gotten worse. And not only has it gotten worse. You'll see there's a little a bit of instability showing up as it courses along its way toward a solution. So what it looks like is that if you've got a rate constant that's too small, it takes forever. If you've get a rate constant that's too big, it can of jump too far, as in my diagram which is somewhere underneath the board, you can go all the way across the hill and get to the other side.

So you have to be careful about the rate constant. So what you really want to do is you want your rate constant to vary with what is happening as you progress toward an optimal performance. So if your performance is going down when you make the jump, you know you've got a rate constant that's too big. If your performance is going up when you make a jump, maybe you want to increase-- bump it up a little bit until it doesn't look so good. So is that all there is to it? Well, not quite, because this is the world's simplest neural net. And maybe we ought to look at the world's second simplest neural net.

Now, let's call this-- well, let's call this x. What we're going to do is we're going to have a second input. And I don't know. Maybe this is screwy. I'm just going to use color coding here to differentiate between the two inputs and the stuff they go through. Maybe I'll call this z2 and this z1 and this x1 and x2.

Now, if I do that-- if I've got two inputs and two outputs, then my performance function is going to have two numbers in it-- the two desired values and the two actual values. And I'm going to have two inputs. But it's the same stuff. I just repeat what I did in white, only I make it orange.

Oh, but what happens if-- what happens if I do this? Say put little cross connections in there. So these two streams are going to interact. And then there might be some-- this y can go into another multiplier here and go into a summer here. And likewise, this y can go up here and into a multiplier like so. And there are weights all over the place like so.

This guy goes up in here. And now what happens? Now, we've got a disaster on our hands, because there are all kinds of paths through this network. And you can imagine that if this was not just two neurons deep, but three neurons deep, what I would find is expressions that look like that. But you could go this way, and then down through, and out here. Or you could go this way and then back up through here. So it looks like there is an exponentially growing number of paths through that network. And so we're back to an exponential blowup. And it won't work.

Yeah, it won't work except that we need to let the math sing to us a little bit. And we need to look at the picture. And the reason I turned this guy around was actually because from a point of view of letting the math sing to us, this piece here is the same as this piece here. So part of what we needed to do to calculate the partial derivative with respect to w1 has already been done when we calculated the partial derivative with respect to w2.

And not only that, if we calculated the partial wit respect to these green w's at both levels, what we would discover is that sort of repetition occurs over and over again. And now, I'm going to try to give you an intuitive idea of what's going on here rather than just write down the math and salute it. And here's a way to think about it from an intuitive point of view.

Whatever happens to this performance function that's back of these p's here, the stuff over there can influence p only by going through, and influence performance only going through this column of p's. And there's a fixed number of those. So it depends on the width, not the depth of the network. So the influence of that stuff back there on p is going to end up going through these guys. And it's going to end up being so that we're going to discover that a lot of what we need to compute in one column has already been computed in the column on the right.

So it isn't going to explode exponentially, because the influence-- let me say it one more time. The influences of changes of changes in p on the performance is all we care about when we come back to this part of the network, because this stuff cannot influence the performance except by going through this column of p's. So it's not going to blow up exponentially. We're going to be able to reuse a lot of the computation. So it's the reuse principle.

Have we ever seen the reuse principle at work before. Not exactly. But you remember that little business about the extended list? We know that we've seen-- we know we've seen something before. So we can stop computing. It's like that. We're going to be able to reuse the computation. We've already done it to prevent an exponential blowup. By the way, for those of you who know about fast Fourier transform-- same kind of idea-- reuse of partial results.

So in the end, what can we say about this stuff? In the end, what we can say is that it's linear in depth. That is to say if we increase the number of layers to so-called depth, then we're going to increase the amount of computation necessary in a linear way, because the computation we need in any column is going to be fixed. What about how it goes with respect to the width?

Well, with respect to the width, any neuron here can be connected to any neuron in the next row. So the amount of work we're going to have to do will be proportional to the number of connections. So with respect to width, it's going to be w-squared. But the fact is that in the end, this stuff is readily computed. And this, phenomenally enough, was overlooked for 25 years. So what is it in the end? In the end, it's an extremely simple idea. All great ideas are simple. How come there aren't more of them? Well, because frequently, that simplicity involves finding a couple of tricks and making a couple of observations.

So usually, we humans are hardly ever go beyond one trick or one observation. But if you cascade a few together, sometimes something miraculous falls out that looks in retrospect extremely simple. So that's why we got the reuse principle at work-- and our reuse computation. In this case, the miracle was a consequence of two tricks plus an observation. And the overall idea is all great ideas are simple and easy to overlook for a quarter century.