PROFESSOR: Today we are introducing an exciting new pledge in 6034. Anyone who has already looked at any of the neural net problems will have easily been able to see that even though Patrick only has them back up to 2006 now, there's still-- well out of four tests, perhaps two or three different ways that the neural nets were drawn. Our exciting new pledge is we're going to draw them in a particular way this year. And I will show you which way, assuming that this works. Yes.

We are going to draw them like the one on the right. The one on the left is the same as the one on the right. At first, not having had to explain the difference between the two of them, you might think you want the one on the left. But you really want the one on the right, and I'll explain why. The 2007 quiz was drawn, roughly similarly, to this. Although if you somehow wind up in tutorial or somewhere else doing one of the older quizzes, a lot of them were drawn exactly like this.

In this representation, one thing I really don't like, is that the inputs are called x's, and the outputs are called y's, but there's two x's, so the inputs are not x and y, and then they often correspond to x's of a graph, and then people get confused. Additional issues that many people have are the fact that the summation and the multiplication with the weight is implied. The weights are written on the edges, where outputs and inputs go, and the summation of the two inputs into the node are also implied.

But take a look here. This is the same net. These w's here would be the w's that are written onto these lines are here. Actually the better way to draw it would be like so, since each of these can have their own w, which is different. So each of the w's that are down here, are being explicitly set to a multiplier. Where as here, you just had to remember to multiply the weight by the input that was coming by. Here you see an input, comes to a multiplier, you multiply by the weight, then once you multiplied all the inputs by the weight, then you send them to a sum, so the sigma is just a sum, you sum them, add them all together, send the result of that into the sigmoid function, our old buddy, 1 over 1 plus e to the negative whatever our input was, with a weight for an offset, and then we send the result of that into more multipliers with more weights, more sums, more sigmoids.

So this is how it's going to look like on the quiz. And this is a conversion guide from version 0.9 data into version 1.0. So if you see something that looks like this, on one of the old quizzes that you're doing, see if you can convert it, and then solve the problem. Chances are if you can convert it, you're probably going to do fine. We'll start off not only with this conversion guide, but also-- I'll leave that up here-- also I'm going to work out the formulas for you guys one more time.

These are all the formulae that you're going to need on the quiz. And then we're going to decide what will change in the formulae, if, and this is a very likely if, there seems to be good amount of times that this happens, is that the sigmoid function in those neurons out there was ever changed into some other kind of function. Hint. It's changed into a plus already in the problem we're about to do. People change it all the time into some bizarro function. I've seen arc tangent, I think. So here we go.

Let's look at the front of it. First of all, sigmoid. Well our old buddy, sigmoid, I just said it a moment ago, sigmoid is 1 over 1 plus e to the minus x. Also, fun fact about sigmoid, the derivative of sigmoid, is itself-- the derivative of sigmoid is-- let's say that the sigmoid-- we'll just turn sigmoid into like the letter say y. Y is the result, right? So if you say y equals 1 over 1 plus e to the negative x, then the derivative of sigmoid is y times 1 minus y. You can also write out the whole nasty thing, it's 1 over 1 plus e to the negative x times 1 minus 1 over 1 plus e to negative x. So the nice property of sigmoid it's going to be important for us in the very near future, and that future begins now.

So now the performance function. This is a function we used to tell neural nets when they inevitably act up and give us really crappy results. At first we tell them just how long they are, with our performance function. The first function can be any sane function that gives you a better score, where better can be decided as lower or higher, if you feel like, that gives you a better score, if your answers are closer to the answer you're looking for. However, in this case, we have, for a very sneaky reason, chosen the performance function to be 1/2 d, which is the desired output, minus o, the actual output squared. So we want a small, well it's negative, So we want a small negative or 0. That would mean we performed well.

So why this? Well the main reason is ddx of performance is, the 2 comes down, the o is the variable that we're actually, so maybe I should say ddo, that negative comes out, we get a simple d minus o. And yeah, we're using derivatives here. So those are fine. These are two assumptions. They could be changed on your test. We're going to figure out what happens, if we change them, if we change the performance, if we change the sigmoid, that is if we change the sigmoid to some other function, what's going to happen to the next three functions, which are basically the only things that you need to know to do backpropagation.

So let's look at that. First, w prime. This is the formula for a new weight. After one step of backpropagation. A new weight in any of these positions that you can see up here on this beautiful neural net. That w-- each of the w's will have to change step by step. That's, in fact, how you do the hill climbing neural nets. You change the weights incrementally. You step a little bit in the direction towards giving you your desired results until eventually, you hope, you have an intelligent neural net. And maybe you have many different training examples that you run it on, in a cycle, hoping that you don't over fit to your one sample, on a computer. But on the test, we will probably will not do that.

So let's take a look at how you calculate the weights for the next level. And then you have the weights for the current level. So first things first. New weight, weight prime equals-- starts with the old weight. That has to go there because otherwise we're just going to jump off somewhere at random. We want to make a little step in some direction, so we want to start where we are, with the weight. And then we're going to add three things. So if we're talking about the weight between some i and some j-- there's some examples of the names of weights. So this is w 1 i, that's the weight between 1 and-- so this is w 1 a, it's the weight between 1 and a. This is w 2 b, which is the weight between 2 and b . Makes sense?

Well makes sense so far, but what if it's just called w b, then it's the weight between-- these w's that only have one letter, we'll get to later. They're the bias. They're the offset. They are always attached to a negative 1. So you can pretty much treat them as being a negative 1 here, that is then fed into a multiplier with this w b, if you like. This is implied to be that. All of the offsets are implied to be that. So w plus sum of alpha-- why is this Greek letter? Where does it come from? How do we calculate it?

Well alpha is just some value told to you on the quiz. You'll find it somewhere. There's no way you're going to have to calculate alpha. You might be asked to try to give us an alpha, but probably not. Alpha is supposed to give the size of our little steps that we take when we're doing hill climbing. Very large alpha, take a huge step. Very small alpha, take tentative steps.

So alpha is there, basically, to change this answer and to make the new value either very close to w, or far from w, depending on our taste. So plus alpha times i, so i is the value coming in into the node. We're changing the weight here. So i is the value, for instance, i sub 1 here, i would be the value of WAC, i would be the value coming output of node a. WBC, i would be the output of node b. i is sometimes as little as i is the input coming in to meet that weight at the multiplier. And then it's multiplied by delta j. Your delta is the delta that belongs to these neural net nodes.

What is a delta, you said? Funny you may ask. It is a strange Greek letter. It sort of comes from the fact that we're doing some partial derivatives and stuff, but the main way you're going to figure out what the deltas are are these two formulae that I've not written in yet. So hold off on trying to figure out what the delta is until-- well right now, I'm about to tell you the delta is. So the delta is basically, think of the delta as using partial derivatives to figure out which way you're going to step, when you're doing hill climbing. Because you know when you're doing hill climbing, you look around, you figure out, OK, this is the direction of the highest increase, and then you step off in that direction.

So the deltas are telling you which way to step, with the weights. And the way they do that is by taking the partial derivative of-- basically you try to figure out how the weight that you're currently looking at is contributing to the performance of the net. Contributing to, either the good performance of the net, or the bad performance of the net. So when you're dealing with the weights, like WBC, WAC, that pretty much directly feed into the end of the net. They feed into the last node, and it then comes out. It's the output. That's pretty easy. You can tell exactly how much those weights, and the values coming from them, are contributing to the end. And we do that by essentially, remember what the partial derivative, so partial derivative here is, in fact, the way that the final weights are contributing to the performance, is just the performance function.

Partial derivative-- I've already figured out the derivative here, it's just d minus o. This is for sort of final weights, the weights in the last level. D minus o, except we're not done yet, because when we do derivatives, remember the chain rule. To get from the end to these weights, we pass through, well it should be a sigmoid, here it's not, we're going to pretend it is for the moment, we pass through a sigmoid, and since we passed through the sigmoid, we had better take the derivative of the sigmoid function. That is, y times 1 minus y. Well what is y? What is the output of the sigmoid? It's up. So that's also multiplied by o times 1 minus o.

However, there is a-- let me see, let me see, yes-- sorry, I'm carefully studying this sheet to make sure my nomenclature is exactly right for our new nomenclature, which so new and brave, that we're doing it, that we only knew for sure we're going to do it on Wednesday. So we have d minus o times o times 1 minus o. So you say, that's fine, that can get us these weights here, even this w c, how are we going to get the deltas for the new weights here? Oh, I realize-- yeah, I got it. So the delta-- by the way, this is a delta c, how is neuron c contributing to the output? Well it's directly contributing to the output , and it's got a sigmoid in it. It doesn't really, but we're pretending it does for now. d minus o times 1 minus o.

What about inner node? Node d, node a, what are we going to have to do? Well the way they contribute to the output is that they contribute to node c. So we can do this problem recursively. So let's do this recursively. First of all, as you have probably figured out, all of them are going to have an o times 1 minus o factoring from the chain rule, because they're all sigmoid, pretending that they're all sigmoids. We also have a dearth of good problems that are actually sigmoid on the web right now. There's only 2007. But here's o times 1 minus o, what are we going to do for the rest of it? How does it contribute to our final result?

Well it contributes to our final result recursively. So we're talking about delta i. I is an inner node. It's not a final node. It's somewhere along the way. So sum over j of w, going from i to j, times delta j. Now sum over all j, j such that i leads to j. I needs to have a direct path into j. So if i, in this instance, was j, everyone, the only possible j in this would be c. That's right. We would not sum over b as one of the j. i does not lead to b, or a does not lead to b, a only leads to c. Also note that c does not need to be here. That's going backwards. So you just-- to figure out which j you're looking at, look directly forwards at the next one. So if there was another d here, or that a does not go to d, a goes to c.

You only look at the next level children, and you sum over all those children, the weight between them, multiplied by the child's delta. That makes sense, right? Because the way we affect, if the child's delta is the way the child affects the output, calling these children for a moment, and then if this one directly affects the output, then the way this one affects it is-- it affects it because it affects this, but it's also multiplied by it's weight. So in fact, for instance, if the weight between a and c were 0, then a doesn't affect the output at all, right? Because its weight is 0, and when we do this problem, we go this times 0, and then we try to add it in there, doesn't affect anything. It's weight is very high, it's going to really dominate c, and that is taken into account here, and then multiply by the delta for the right node.

So the following question, and since I spent a lot of time with formulae and not that much time starting on the problem, I will not call on someone at random, but rather take a volunteer. If no one volunteers, I'll eventually tell you, which is, we've got some nice formulae on the bottom three. If we change the sigmoid function, what has to change? That's the only thing that changes in this crazy assed problem right here, which by the way, changes the sigmoid functions into adders, is that we take all of the o times 1 minus o in delta f and the delta i, and we change it to a new derivative. We then do the exact same thing that we would've done. Correct.

And on a similar note, if you change the performance function, how many of these equations at all have to change out of the bottom three. Yeah. That's right, just one, just delta f. Take the d minus o, make it the new derivative of the new performance function. And in fact, delta i doesn't change at all. Does everyone see that? Because it is very common for something to be replaced, I think three of the four the quizzes that we have, replaced in some-- changed something in some way.

All right. Let's go. We're going to do 2008 quiz, because it has a part of the end that screwed up everyone, and so let's make sure we get to that part. That's going to be the part that you probably care about the most at this point. So these are all adders instead of sigmoids. That means that they simply add up everything as normal, for a normal neural net, and then there's no sigmoid threshold. They just give some kind of value. Question?

STUDENT: So we talked about those multiplier things, we don't have those in nodes?

PROFESSOR: They're not neural net nodes. That is one of the reasons why that other form that you can see over there is elegant. It only has the actual nodes on it. It is very compact. It's one of the front we've used in the previous tests. The question is, do those multipliers count as nodes? However by not putting in the multipliers, we feel it sometimes confuses people of explicitness. The ones that are nodes will always have a label, like a or here, you see there's a sigmoid and an L1. The multipliers are there for your convenience, to remind you to multiply, and also those, if you look those sigmoids that are over there, are there for your convenience to remind you to add.

In fact, the only thing that counts as a node in the neural net-- and that's a very good question-- is usually the sigmoids, here it's the adders. We've essentially taken out the sigmoids. These adders are the-- oh, here's the way to tell. If it's got a threshold weight associated with it, then it's one of the actual nodes. A threshold weight. I guess the multipliers look like they have a weight, but this is just the weight that is being multiplied in. This is our witness be multiplied in with the input, but if it has a threshold weight, like wa, wb-- oh, I promised I would tell you guys the difference between the two weights.

So let's do that very quickly. The kinds of weights that, say w2b or w1a, our weight the comes between input 1 and a or between a and c, then mentally multiplying the input by this weight, and then eventually that's added together. The threshold weights, they just have like wb, wa, wc. They are essentially to decide the threshold for a success or failure, for a 1 or a 0, or anything in between, at any of the given nodes. So the idea is maybe you at some node want to have a really high cut off, you have to very high value coming in, or else it's a 0. So you put a high threshold. The weight is multiplied by negative 1. And in fact, the threshold weight won't-- one could consider if you wanted to that the threshold weight times negative 1. was also added in it that sum, instead of putting at the same location as the node. If that works better for you, when you're converting it, you can also think of it that way. Because the threshold weight is essentially multiplied by negative 1 and added in at that same sum over there. So that's another way to do it.

There's a lot of ways to visualize these neural nets. Just make sure you have a way that makes sense to you, and that you can tell pretty much whatever we write, as long as it looks vaguely like that, how to get it in your mind, into the representation that works for you. Because once you have the representation right for you, you're more than halfway to solving these guys. They aren't that bad. They just look nasty. They don't bite. OK.

These are just adders. So if it's just an adder, then that means that, if we take all the x inputs coming in-- let's do x and y for the moment, so we can figure out the derivative-- then what comes out after we just add up the x, what comes out, y equals x, right? We're just adding it up. Adding up all the input, we're not doing anything to it. Y equals x is what this node does. You people see that? So the derivative is just one. So that's pretty easy, because the first problem says, what is the new formula, delta f. So I'll just tell you. You guys probably figured it out. o times 1 minus o. Because we replaced d minus o with 1. OK? Makes sense so far? Please ask questions along the way, because I'm not going to be asking you guys. I'll do it myself. Question?

STUDENT: Why to we use d minus o of 1?

PROFESSOR: That's a good question. The reason is because I did the wrong thing. So see, it's good that you guys are asking questions. It actually should be replaced with o times 1 minus o with 1. The answer is delta f equals d minus o. So yes, perhaps I did it to trick you. No, I actually messed up. But yes, please ask questions along the way. Again, I don't have time to call on you guys at random to figure out if you guys are following along. So I'll do it myself. We're placing the o times 1 minus o with 1 because of the fact that the sigmoid is gone, and we get just delta f equals d minus o.

So great. We now want to know what the equation is for delta i, at the node a. So delta a. Well let's take a look. The o times 1 minus o is gone. Now we just have the sum over j, which you guys already told me is, only c of WAC times delta c. We know that delta c is d minus o. The answer is delta a is just WAC times d minus o. That time, I got it right. I see the answer here. Though it's written in a very different format from the old quiz. Any questions on that?

Well that's part a that we finished out of c. Let's go to part b. Part b is doing one step backpropagation. There's almost always going to be one of these in here. So the first thing it asks is to figure out what the output o is for this neural net if all weights are initially 1 except that this guy right here is negative 0.5. All the other ones start off as 1. Let's do a step-- oh, let's see what are the inputs. The inputs are also all 1. Desired output is also 1. And in fact, the rate constant alpha is also 1. This is the only thing that isn't 1, folks.

So let's see what happens. 1 times 1 is 1, then this is a negative 1 times 1 is negative 1. That's 0. The exact same thing happens here because it's symmetrical. So these are both 0. 0 times 1 is 0, 0 times 1 is 0. Then this is negative 1 times negative 0.5 is positive 0.5, so 0 plus 0 plus a positive 0.5, the output is positive 0.5. Does everyone see that? If not, you can convince yourself that it is positive 0.5. That would be a good exercise for you, run through one forward run. The output is definitely positive 0.5. First time around. OK?

Now we have to do one step of backpropagation. To do that, let's calculate all the delta so that we can calculate all the new weights, the the new weight primes. So delta c. That's easy. You guys can tell me what delta c is. We figured out what the new delta c is going to be. So simple addition or subtraction problem? Everyone, delta c is?

STUDENT: 0.5.

PROFESSOR: 0.5, one half, yes. All right. We know that delta a and delta b are just WAC times delta c, and WBC times delta c. So they are?

STUDENT: One half.

PROFESSOR: Also one half, because all the weights were 1. Easy street. OK. We've got all of the deltas are one half. And all but a few of the weights are 1. So let's figure out what the new weights are. New WAC, OK. Yeah, so let's see. What's going to be the new WAC? So the new WAC is going to be old WAC, which is 1, because all of them are 1 except for wc, plus the rate constant which is 1, times the input coming in here, but remember that was 0, so actually it's just going to be the same as the old WAC. This is a metrical problem between b and a, at the moment, this is going to be the same.

All right. Somethings are going to change though. What about wc, that was the one that was actually not 1? OK. So new wc, remember, the i for wc, the i that we use in this equation is always negative 1 because it's a threshold. So we have the old wc, which is negative 0.5, plus 1 times negative 1 times delta c, which is one half. So we have negative 0.5 plus negative 0.5 equals negative 1. w 1 a, well we've got w 1 a starts out as 1. Then we also know that w 1 a is going to be equal to 1 plus 1 times the input, which is 1, times delta of a, which is one half, so 1.5. And since it's symmetrical between a and b, then w 2 b is also 1.5. And then finally, wa and wb, the offsets here, well they start at 1 plus 1 times negative 1 times 0.5. So they're both, everyone?

STUDENT: One half.

PROFESSOR: One half. That's right. That's right. Because negative 1 is their i. Negative 1 times one half plus positive 1 is just one half. That's one full step. Maybe a mite easier than you might be used to seeing, but there's a full step. And it asks what's going to be the output after one step of backpropagation? We can take a look. So we have 1 times the new wa, which is 1.5, you've got 1.5, then the new wa is just 0.5, now is 0.5, that's a 1 coming into an adder. We've got another 1 coming in here because it's symmetrical. So 1 and a 1, 1 times WAC is 1. 1 times WBC is 1. So we have two 1s coming in here, they're added, that's 2. Then this has become negative 1, in fact, at this point. So negative 1 times negative 1, that's 3, and the output is 3. All right. Cool. We've now finished part b, which is over half of everything. Oh no, we've not. One more thing.

These are adders. They're not sigmoids. What if we train this entire neural net to try to learn this data, so that it can draw a line on a graph, or draw some lines, or do some kind of learning, to separate off the minuses from all the pluses. You've seen, maybe, and if not, you are about to in a second, because it asks you to do this in detail, than neural nets can usually draw one line on the graph for each of these, sort of, nodes in the net, because each of the nodes has some kind of threshold. And you can do some logic between them like ands or ors.

What do you guys think this net is going to draw? Anyone could volunteer, I'm not going to ask anyone to give this answer. That's a little bit tricky, because usually if you had this many nodes, you could easily draw a box and box off the minuses from the pluses. However, it draws this. And it asks what is the error? The error is-- oh yeah, it even tells you the error is 1/8, because why? These are all adders. You can't actually do anything logical. This entire net boils down to just one node, because it just adds up every time. It never takes a threshold at any point. So you can't turn into logical ones and zeroes, because it's basically not digital at all, its analog. It's giving us some very high number. So it all boils down to one cut off. And that's the best one. The one that I drew right here. OK.

Did that not make sense to you? That's OK. This problem is much harder. And putting them both on the same quiz, was a bit brutal, but by the time you're done with this, you'll understand what a neural net can do or not. I put these in simplified form because of the fact that we don't care about their values or anything like that. But inside of these little circles is a sigmoid, the multipliers and the summers are implied. I think in the simplified form when we're not actually doing backpropagation is easier to view it, and see how many nodes there are. For the same reason you asked your question about how many there are. So all of those big circles are node. And in those nodes is a sigmoid now, not those crazy adders.

We have the following problem. We have to try to match each of a, b, c, d, e, f to 1, 2, 3, 4, 5, 6, using each of them only once. That's important, because some of the more powerful networks in here can do a lot of these. So it's like yes, the powerful networks could do some of the easier problems here, but we want to match each net to a problem it can do, and there is exactly one mapping that will map-- that is one to one, and maps exactly, uses all six of the nets to solve all six of these problems here.

So some of you may be going like, what? How am I going to solve these problems? I gave away a hint before, which is that each node in the neural net, each sigmoid node can usually draw one line on the-- it can draw one line into the picture. The line can be diagonal if that nodes receives both of the inputs, which is here, i 1 and i 2. See there is an i 1 and an i 2 axis. Like x- and a y-axis. The node has to be horizontal, or vertical, if-- sorry, the line has to be horizontal or vertical if the node only receives one of the inputs. And then, if you have a deeper level, these secondary level nodes can sort of do a logical, can do some kind of brilliant thing like and or or of the first two, which can help you out. All right. And so let's try to figure it out.

So right off the bat, and I hope that people will help and call this out, because I know we don't have enough time that I can force you guys to all get it. But right off the bat, which one of these looks like it's the easiest one?

STUDENT: Six.

PROFESSOR: Six. That's great. Six is definitely the easiest one. It's a single line. So this is just how I would have solved this problem, is find the easiest one. Which of these is the crappiest net?

STUDENT: A.

PROFESSOR: A is the crappiest net. But there's no way in hell that A is going to be able to get any of these except for six. So let's, right off the bat, say that six is A. All right. Six is A. That's A. We don't have to worry about A. OK. Cool. Now let's look at some other ones that are very interesting. All the rest of these draw two lines, well these three draw two lines. These three draw three lines. They draw a triangle. So despite the fact that this c is a very powerful node, that indeed, with three whole levels here of sigmoids, it looks like there's only two that's in our little stable of nets that are equipped to handle number one and two. And those are? E and F, because E and F have three nodes at the first level. They can draw three lines. And then they can do something logical about those lines, like for instance, maybe, if it's inside all of those lines. There's a way to do that. You just-- basically you can give negative and positive weights as you so choose to make sure that it's under certain ones, above other ones, and then make the threshold such that it has to follow all three of your rules. So between E and F, which one should be two and which one should be one. Anyone see?

Well let's look at two and one. Which one is easier to do? Between two and one. Two. It's got a horizontal and a vertical. One has all three diagonal. And which one of these is a weaker net, between E and F. F. F has one node that can only do a horizontal, and one node that can only do a vertical line. So which one is F going to have to do? Two. And E does what? Good job, guys. Good job, you got this.

So now let's look at the last three. Number three is definitely the hardest. It's an exceller.

Those of you who've played around with double o 2 kind of stuff, or even just logic, probably know that there is no way to make a sort of simple linear combination in one level of logic to create an x or. x or is very difficult to create. There are some interesting problems involving trying to teach an exceller to a neural net. Because a neural net is not to be able to get the x or, because of the fact that you can tell it, OK, I want this one to be high, and this one to be low. That's fine. You say these both have to be high. That's fine. It's hard to say, it's pretty much impossible to say, this one or this one, but not the other, because need to be high in a single node, because of the fact that if you just play with it, you'll see.

You need to set a threshold somewhere, and it's not going to be able to distinguish between, if the threshold is set such that the or is going to work, the whole or is going to work. It's going to accept when both of them are positive as well. So how we can do x or? We need more logic. We need to use some combinations of ands and ors in a two level way. To do that we need the deepest neural net that we have. There's only one that's capable of that. And that is? It's C.

There are many different ways to do it. Let's think of a possibility. i 1 and i 2 draw these two lines. Let's call these one, two, three, four, five, node 1 and node 2 draw these two lines. And I'll just sort of draw it here for you guys. Then maybe node 3 gives value to-- yeah, let me see-- node three can give value to perhaps-- let's see-- node 3 can give value to everything that is-- there are a lot of possibilities here. Node 3 can give value to everything that is up here. Actually node 3 can give value to everything except for this bottom part, and then node 4 could give value to say-- doesn't do it yet, but there's a few-- there's a few different ways to do it if you played around. The key idea is that node 3 and node 4 can give value to some combination and or or not, and then node 5 can give value based on being above or below a certain threshold, combination of 3 and 4. You can build an exceller out of the logic gates.

I will ponder on that in the back burner for a moment, as we continue onward, but clearly C has to do number three. OK. Now we're left with four and five. I think, interestingly, five looks like it may be more complicated than four, because of the fact that it needs to do both different directions instead of two of the same direction. So however, just the idea of the one with the fewer lines, being a simpler one, may not get us through here. And there's a reason why. Look what we have left to use. We have to use D or B. What is the property of the two lines that D can draw? D being the simpler one. One horizontal, one vertical, that's right. So even though it may look simpler to just have two horizontal lines, it actually requires B. B is the only one that can draw two horizontal lines because D has to draw one horizontal and one vertical. So that leaves us with, B on this, D on this. Excellent, we have a question. I would've thought it would have been possible that we had no questions, or maybe I just explained it the best I ever have. Question.

STUDENT: I didn't get why B has to be two horizontal lines.

PROFESSOR: All right. So the question is, I don't understand why B to be two horizontal lines. The answer is, it doesn't. B can be anything, but D can't be two horizontal lines. And so by process of elimination, it's B. Well take a look at D, right. So D has three nodes, one, two, three. Node 1 and node 2 can just draw a line anywhere they want, involving the inputs they receive. What input does node 1 receive? Let's go to node 1. So it can only make a cut off based on i 1. So therefore, it can only draw by making the cut off above and below a certain point. Node 1 can only draw vertical lines. Node 2 can only draw a horizontal line, because it can only make a cut off based on where it is an i 2. Therefore they can't both draw a horizontal. That's why this is the trickiest part. This last part, because B is more powerful. B does not only have to do two horizontal lines. It can do two diagonal lines. It can do anything it wants. It just happens that it's stuck doing this somewhat easier problem, because the fact that it is the only one left that has the power to do it.

So let's see, we're done, and we'd have aced this part of the quiz that like no one got, well not no one, but very few people got, when we put it on in 2008. The only thing we have left to ask is-- let me see-- yeah, the only thing we have left to ask is what are we going to do here for this? All right. Let's see. For the x or, let's see if I can do this x or. OK. How about this one. Right. I'm an idiot. This is the easiest way. Number one draws this line. Number two draws this line. Number three ends the line, the two lines. Number three says only if both of them are true, will I accept. Number four maps the two lines. And number five ors between three and four. Thank you. No, it's not that hard. I just completely blanked, because there's another way that a lot of people like to do it. It involves drawing in a lot of lines, and then making the clef b 2. But I can't remember it at the moment. Or there any other questions? Because I think if you have a question now, like four other people have it and just aren't raising their hand. So ask any questions about this drawing thing. Question?

STUDENT: Why do we do this?

PROFESSOR: Why do we do this drawing thing? That's a very good question. The answer is so that you can see what kinds of nets you might need to use in these simple problems, to answer these simple problems. So that if Athena forbid that you have to use a neural net in a job somewhere to do some actual learning, and you see some sort of quality about the problem, you know not to make a net that's too simple, for instance. And you wouldn't want a net that is more complex than it has to be. So you can sort of see what the net's do at each level, and more visibly understand. I think a lot of people who drew problems like this just want to make sure people know, oh yeah, it's not just these numbers that we're mindlessly backpropagating from the other part of the problem to make them higher or lower.

This is what we're doing at each level. This is the space that we're looking at. Each node is performing logic on the steps before. So that if you actually have to use a neural net later on, down the road, then you'll be able to figure out what your net's going to need to look like. You'll be able to figure out what it's doing. At least as well as you can figure out what it's doing, for a neural net, since it often will start getting up these really crazy numbers, will have all sorts of nodes in it, and like a real neural net that's being used nowadays, there'll be tons of nodes, and you'll just see the numbers fluctuate wildly, and then suddenly it's going to start working or not. That's a good question. Any other questions? We still have a few minutes. Not many, but a few. Any other questions about any of this stuff? Sorry.

STUDENT: Talk about what you just asked. Just because we draw it, does the machine need to learn--

PROFESSOR: You're confused why the machine is run what, by the pictures on the right? Oh OK. Machine does not have to learn by drawing pictures and calling them in. Let me give you some real applications. My friend at the University of Maryland recently actually used neural nets because, yeah, he actually did, because of the fact that he was doing an game plan competition, where the game was not known when you were designing your AI. It had to be able to-- there was some very elegant, general game solver thing that you had be able to hook up into, and then they made up the rules, and you had a little bit of time, and then it started. Some of the AI's, what they did was, they trained, once they found out what the rules were on their own, with the rules, in his case he had a neural net, because it was so generic, you just have a web of random gook. He thought it could learn anything, and then-- he never did tell me how it went, probably didn't go well. But maybe it did. It basically tried to learn some things about the rules. Some of the other people who are more principled game players actually tried to find out fundamental properties of the space of the rules by testing a few different things, so they could view more knowledge is less search so they could do less search when the actual game playing came on. And then when the actual game playing came on, pretty much everyone did some kind of game tree based stuff.

He's telling me that a lot of Monte Carlo based game tree stuff that is this very non deterministic as what they're doing nowadays, rather than what determines the alpha beta, although he said it converges to alpha beta, if you've given enough time. That's what he told me, But that someone I know who is using neural nets. I've also in a cognitive science class I took, saw neural nets that tried to attach like qualities to objects, by having just this huge, huge number of nodes in levels in between, and then eventually it was like, a duck flies, and you're like, how's it doing this again? I'm not sure, but it is. So the basic idea is that when-- one of the main reasons that neural nets were used so much back in the day is that people on many different sides of this problem, cognitive science, AI, whatever, were all saying, wait a minute, there's networks of neurons, and they can do stuff, and we're seeing it in different places. And when you've seen it in so many different places at once, must be a genius idea that's going to revolutionize everything.

And so then everyone started using them to try to connect all these things together, which I think is a noble endeavor, but unfortunately people just stopped using it. It didn't work as they wanted. It turned out that figuring out our neurons worked in our head was not the way to solve all AI hard problems at once. And they fall into disfavor, although are still used for some reasons, like the sum is like that. So we wouldn't use it just to draw these pictures. The reason why we have these pictures is because we give you simple nets that you can work it out by hand on the quiz. Any net that is really used nowadays would make your head explode, if we tried to make you do something with it on the quiz. It would just be horrible.

So I think that's a good question. If there's no other questions, or even if they are, because we have to head out, if there's any other questions, you can see me as I'm walking out.