[SQUEAKING]

[RUSTLING]

[CLICKING]

PROFESSOR: But yeah, we are very glad to have John Hull with us here. So John, please.

JOHN HULL: Well, thanks very much. It's a real pleasure to be here. Let me just say a little bit by way of introduction. I probably got a similar background to many of you. Math was always my best subject at high school. That was in the UK.

I, not surprisingly, ended up doing a math undergraduate at Cambridge. And after graduating, I decided that I wanted to apply my math in business. So I did a master's in operational research, which was a fairly new thing in those days.

And then an amusing little factoid, I worked for two years after graduating with my master's-- I worked for two years for a shoe retailer. Everybody bursts out laughing when they hear this. Actually, it was a large shoe retailer in the UK. It had 2,000 outlets. And actually, it was a pretty interesting job. I mean, it was all about deciding which outlets should stock which shoes in which sizes and so on, and then which outlets should be closed and where we should look for new outlets and that sort of thing. So there's lots of interesting stuff.

But after two years with this company, I moved back into the academic world. I won't bore you with all the details. But I wound up I got my PhD and ended up teaching here at University of Toronto in what we call the Joseph L. Rotman School of Management. And I've been at University of Toronto for about 30 years now.

Probably I'm best known for my work on derivatives, which I understand you guys have had quite a bit of instruction in. And most of my 30 years has been spent doing research in derivatives, but I guess it was about seven, eight years ago, I began to see that machine learning and AI were becoming really important to our students.

And so I switched my focus. And in fact, all of my teaching is now in the machine learning area. And pretty much every-- just let me advance slides. Pretty much every course that we offer at Rotman actually has either a compulsory course or an elective in machine learning. So it's not just me that teaches machine learning. I've now got quite a few colleagues who do so as well.

I wrote a book, which is now in its third edition, *Machine Learning in Business-- An Introduction to the World of Data Science.* And you can find out more information on the book from my website. But I'm not saying you should buy the book now. In fact, the reverse is true because the fourth edition will be coming out in probably about two months. The fourth edition is going to be with three co-authors who are actually the other people who teach machine learning here at Rotman.

So the fourth edition is going to deal with-- I mean, third edition came out in I think it was 2021. And since then, of course, we've had a lot of things happen in natural language processing, large language models, that sort of thing. So I wanted to bring the book up to date. And my colleagues have helped me do that.

Really, today's presentation is in three parts. I'm going to talk about the nature of machine learning. Then I'm going to move on to talk about applications of neural networks, and finally, what I've been working on quite a lot over the last little while, applications of reinforcement learning. OK. I don't know whether there's any easy way you can interrupt me to ask questions, but I'm happy to be interrupted with any questions you have as we go along. Is it possible for them to interrupt me with questions?

PROFESSOR: If they speak up with a question, you'll hear them speak up.

JOHN HULL: I will hear them?

PROFESSOR: Yes, I think you will.

JOHN HULL: Good, good. OK. No, I don't mind being interrupted at all. OK. So next slide, please.

So what is machine learning? It's a branch of AI. And the idea underlying machine learning is that we give a computer access to lots of data and just let it learn, learn about the relationships between variables and make predictions.

Some of the techniques of machine learning date back to quite a long way. The key thing that's different today is that computers are really fast. So we had all these techniques for doing what we call machine learning quite some time ago. But really computers weren't fast enough to make them a practical tool. We're now reaching the stage where all of these ideas suddenly become practical tools because of the speed of the machines.

OK. I think it's important to distinguish between machine learning and all the automation that's gone on before it. I mean, computers have been used for many years to automate many business decisions-- payroll, sending out invoices, and so on. This is what we could call the third Industrial Revolution.

OK. And machine learning is central to the fourth Industrial Revolution. If you want a bit of background here, the first Industrial Revolution was steam and water power, which was about 1760 to 1840. The second Industrial Revolution was electricity and mass production. This is not me. This is what's generally considered to be the case. And this was 1840 to 1920.

Then basically, computers and digitization were the third Industrial Revolution. And we've really had it. We now know how to use computers to automate things that human beings do. And let's say we can call that computers and digitization. We could say that went from maybe 1950 to the year 2000.

And then we're now living through the fourth Industrial Revolution. And we'll be living through it for quite some time, which is really AI, machine learning and everything we're going to be talking about today. OK. Next slide, please.

So if you want to go back to slide 6, if you want to distinguish between the third and the fourth Industrial Revolution, one example I use is with loan applications. I mean, let's suppose we've got loan officers in a company. And there are certain rules they use to decide whether a loan should be accepted or not.

And those rules have been worked out. Then we could automate their activities. And that would just be the third Industrial Revolution. If we didn't know the rules, we could actually use machine learning to determine the rules. We'd collect lots and lots of data on loans that came before the loan officers and the decisions that were made. And it would be an application of machine learning.

We could go one step further, though, and we could also say, well, let's try and improve on the rules. So then we could collect lots of data on decisions that were made by the loan officers and how those decisions worked out, and try and improve the rules for accepting or rejecting loans.

So the first bullet point here is the third Industrial Revolution, digitization, fairly straightforward, and then the last two would be applications of machine learning. OK. Next slide.

So I'm sure you've all been trained in statistics. And we all know that what we're supposed to do in statistics is sit down, comfortable chair, not look at any data, and develop a hypothesis. Then we collect data and test the hypothesis. That's the standard way of doing things in statistics.

Machine learning is different. What we do, we don't develop any hypotheses in machine learning. We go out and we collect the data. And we try and explain the data. We try different models. And we find patterns in the data or try and develop predictive tool or something like that. But the key difference between machine learning and the traditional Statistical approach is that, in machine learning, we collect the data first where, in statistics, you're not supposed to collect the data first. You're supposed to develop the hypothesis before you've even looked at the data. OK. Next slide.

So there's various types of machine learning algorithms. Unsupervised learning, which is basically just looking for patterns in the data or clusters in the data. Supervised learning, which is where we're trying to predict something or classify the data in some way. And then reinforcement learning, which is multi-stage decision making, which we'll be talking a bit about later on.

And there's other ways of subdividing the algorithms. But this is one quick classification, if you like. Machine learning has its own terminology. And actually, when I first got into machine learning, as I say, it was something like seven or eight years ago, I was pretty confused because the terminology is different from statistics.

In statistics, for example, we talk about dependent variables, independent variables, and so on, whereas, in machine learning, we talk about features and targets. Targets, as the name implies, is the thing you're trying to predict, perhaps. And features are the independent variables you're trying to predict from, things like labels, activation functions, and so on. In fact, when I wrote my book, which I showed you earlier, I have a whole section, I think it's about 10 pages long, showing all the terminology which machine learning uses, some of which is quite different from the terminology that statistics uses and other branches.

It's one of the things that just takes a bit of getting used to. And this terminology, unsupervised learning, supervised learning, for example. I mean where does that come from? Unsupervised learning, well, with unsupervised learning, you're just looking for patterns in the data. So you could say don't really need to-- you don't really need any supervision because all you're doing is just looking for patterns.

And so you just-- all you need to do is to throw the data at the software and the software will find patterns for you, whereas supervised learning, you've got to do a bit of supervision because you've got to tell-- you've got to tell the computer what you want it to do with the data. You want it to predict something, classify data in some way, and so on.

So OK. Sorry. Next slide. So just one slide on unsupervised learning then. So in a typical application, what you're asking, you give the software data and you ask it to divide the data into clusters. So the observations in the cluster have similar feature values. OK.

And so a typical application would be a company wants to understand its customers better. So it collects attributes of its customers, maybe 10 different attributes that it collects, like how much the customer spends every year, how many different products the customer buys, where the customer is located, and so on and so on. And basically, you give the software a list of your customers with those attributes. And it will classify them or cluster them.

So you say, oh yeah, we've got customers that look like this. We've got customers that look like this, we've got customers, and so on. And I'm told that this is actually quite a powerful tool because sometimes you get to find clusters of customers that you never really thought of as being a cluster of your customers before but this tool has actually enabled you to find the clusters.

OK. Next slide. So supervised learning is where you're not just trying to understand the data and create clusters from your data. You're actually trying to predict something, typically. And what you do is you divide your data into three sets, training set, a validation set, and a test set because, remember, what we're doing here is not-- we haven't got a hypothesis. We've just got a lot of data. And so, typically, you put about 60% of your data into the training set, 20% into the validation set, and 20% into the test set. And hopefully, you've got quite a lot of data.

So you develop different models using the training set. So you try out different models. And we'll see some examples of this in a moment. And you compare those models using validation set. Say, well, this model seems to do better than this other model when tested on-- the parameters of the model are determined using the training set. And then you can see how well the models work on the validation set and decide which seems to be the best model for explaining the data.

Typically, you can find a really complex model, sometimes anyway, that will explain the data in the training set but then it does not do well on the validation set. So the model can be too complex. So we'll see. So you want to, typically, you want to-- a rule of thumb is you want to increase model complexity until the model starts to perform worse on the validation set.

So once you've chosen a particular model, you can increase its complexity and the parameters, and so on. But as soon as it starts to perform worse on the validation set, you know you've overfitted the training set. And then the test set is kept to one side. And it's not being involved in the selection of the model or anything like that. And so it provides a good final out-of-sample indication as to how well the chosen model works.

Yeah. Next slide, please. Thanks. So this is an example that I use in the book. It's a pretty trivial example in a way, but it's for predicting salaries for people in a certain profession in a certain area as a function of age. And we try out different models.

Didn't have a huge amount of data for this particular one. We try out different models which are just polynomials, polynomials of the person's age. So the first, the extreme left model, is a fifth order polynomial. It fits the data pretty well. But actually, what it's fitting is the first 60% of data that you use to compare your models.

OK. It doesn't fit the validation set well. So it's overfitting you find. The linear model is underfitting. And the best model actually for the data that I used was a quadratic model. So salary increases and then slightly tails off as you get older.

So it's an example of-- it depends how much data you've got, of course, for comparing your models. But it quite often happens that you wind up overfitting a model. And maybe we can just go back a slide or two to-- yeah. So training-- sorry.

So you use the training set. But you find that you have actually fitted the training set really well but the model doesn't extend well to the validation set. So you then say you've overfitted the training set. So that's an example of what I'm talking about. Increase model complexity until the model starts to perform worse on the validation set. Thanks, Peter.

Let's move on. Neural networks. Well, maybe I'll just stop there. Are there any questions that anybody really wants to ask or points Anybody wants to make before we move on? OK. If not, OK, let's move on to the next slide and the next slide.

So we're now going to talk about neural networks. And here's a picture of an artificial neural network. It's abbreviated ANN. So basically, what we're trying to do is to, in this example, predict something from certain variables.

So the input layer would be the variables that we know. And the output layer are the one or more things we're trying to predict. Now I know I've got multiple variables in the output layer here. Typically though, in many applications, there's only one thing you're trying to predict. So let's think in terms-- I mean, if there's multiple things you're trying to predict, then obviously you've got to-- in terms of your objective, you've got to weight how important is it to predict this one, how important is it to predict that one and so on. So let's just, for the moment, assume there's only one variable in the output layer.

The models that you're most familiar with are models that predict the output from the input in one go as it were. Linear regression would be a classic example. But any more complicated model is also likely to predict the output directly from the input.

The key thing that we're doing with an artificial neural network is we're not doing it all in one go. We're using the input layer, the variables that we know, to predict these variables in the first hidden layer, V1,1, V1,2, V1,3, and so on. And then we use those variables in the first hidden layer to predict variables in the second hidden layer, V2,2, V2,2, V2,3 and so on, and then the third hidden layer. So the key thing is about a neural network is that you're going through a number of stages in predicting the output from the input. It's not all done in one go. So we're not writing down a nice, neat equation for the relationship between the output and the input and predicting the parameters of that equation. We're saying there's a number of stages. We go through in getting from the input to the output. And that makes the model, of course, a lot more flexible. Next slide.

So what's the relationship between the values in one layer and the values in a previous layer? Well, what we're doing is we're relating the values at one neuron to a linear combination of values in the previous layer. And we're using these things called activation functions to do it. Maybe we go back a slide now, Peter, if we could.

And so if we look here, V1,1 is related to the variables in the input layer, the variables we know, V0,1, V0,2, V0,3. And what we do-- what we do is we apply weights to those variables. So we have a weight times V0,1 plus a weight times V02, plus a weight times V0,3 and so on.

And then once we got this weighted average of the V01, V0,2, V0,3, we apply a function to it to get V1,1. And then we do the same thing for V1,2. We take a weighted average of V0,1, V0,2, V0,3 and so on, different weighted average, and apply a function to it to get V1,2 and so on. And then to go from the V1,1's V1,2's, V1,3's to V2,1, V2,2, V2,3, we do the same thing.

So V2,1, for example, we take a weighted average of V1,1, V1,2, V1,3, V1,4 and so on, and apply a function to it, this activation function to get V2,1. And then so what are the things-- what are the parameters of this model? The parameters are the weights. What weights do we apply to the V's in going from one layer to the next layer? OK.

And there's a lot of weights in this example that you got in front of you there because when we're determining V1,1, we've got a weight associated with V0,1, V0,2, V0,3, and so on. But even if there are only three inputs, that's three weights. And then V1,2 is another three weights. V1,3 is another three weights and so on. So you can wind up with a lot of weights. But those are the variables that-- those weights are the variables you're trying to determine. And then the activation function is something you apply after you've applied the weights.

OK. So can we move on to the next slide now? So you see a number of different activation functions that you could use. fy equals y means that you've just applied weights to the variables in the previous layer. And then you've just haven't done anything else to them. You've just said that that's going to give you a value in the new layer. Now if, throughout the whole neural network, you just use the identity, if you think about it, you just have a linear model. OK.

So if the variables in the first layer were a linear function of the input layer, the variables in the second layer were linear functions of the variables in the first layer and so on, and you went all the way through, then it would just be a linear model. And that wouldn't be very interesting because if we want to have a linear model, we don't have to go through all this.

However, as we'll see, you often use the identity just for the final layer. You see sigmoid is a popular activation function. So once you've determined the weighted average of the variables in the previous layer, you take 1 minus 1 plus e to the minus that to get the value of the new node and so on. Hyperbolic tangent, ReLU and so on. Lots of different activation functions.

So you get the idea. You're really saying that the relationship between the outputs and the input is a kind of a convolution of a lot of these activation functions applied to weighted averages. OK. Next.

So how do we actually solve the model? Well, basically, you've got, typically, in models, you've got lots of parameter values, all the different weights that you're applying. And you say, well, you start off by assuming some values for the weights, probably not very good values, but you assume some values. And then you calculate gradients.

You say, well, perhaps we're trying to minimize some value or minimize a function. So we're trying to determine the input values that will minimize some function. So we calculate gradients to determine the direction in which the parameter values should be changed to improve things to actually make the function smaller. And then you take a step and then do the whole thing again, calculate the gradients, and move the parameter values again in a certain direction and so on. So it's a case of-- somebody's trying to call me.

So basically, what you need is you're trying to minimize some function. What you need is the partial derivative of that function with respect to all these parameters, all these weights that define your model. Now that seems a pretty tricky task. But actually, there was-- I mean, clever maybe, you could say almost obvious way of doing this is to say, what we're really talking about with this model is the final thing that we're interested in being a function of a function of a function.

In other words, if we're a function of the final weights and those final weights are a function of the weights that come immediately before the final weights and then those final weights. So each weight-- and of course, in calculus, we know how to calculate the gradient of a function of a function of a function. So there are some clever shortcuts for calculating these gradients that you need to move the parameters in the right direction. They're not starting from scratch in terms of calculating each gradient. Next slide, please.

So this is a very simple example but it illustrates the nature of what you're doing. I mean let's suppose you wanted to calculate the value of x that minimizes this function, y equals 6x squared minus 8x plus 20. Of course, we could all do this with calculus very quickly. And x equals 4 is the answer.

But suppose we were doing it in the way that I've just described. Then we'd start with some estimate of x. It might be x equals 1, not a very good estimate. And then we'd look at the gradient of this, when x equals 1. And what is it? It's 2x minus 8. So it's the gradient is minus 6.

We move the value that we're looking at in the direction, which minimizes y, which the gradient was minus 6 so we take the negative of the gradient because we're trying to minimize something and move it in the direction indicated by the gradient by a certain amount. And we see that we're then at a little bit more than 2. Then we calculate the gradient again, move it again, calculate gradient again. So you can see how this method would work in this very simple example where there's only one thing that we're having to estimate. OK.

And obviously, when there's many things we want to estimate, it's the same basic principle except we're working in many dimensions. OK. Next slide, please. So we continue training the model until the results of the validation set diverge from those of the training set. We don't want to overtrain.

Choosing the learning rate is important. The learning rate is how far we move. Once we've estimated the gradient, we have to decide how far we want to move in the direction indicated by the gradient. And well, there are some clever ways of determining the learning rate as it's called.

But clearly, if we could just go back to the previous slide a moment, I think you can see here, if you choose a really small learning rate, then the gradient at 1 would be the same but you'd move only a very short amount, and then you calculate the gradient again, and move a very short amount, and so on. You'd never actually reach the bottom of the-- you'd never in a reasonable time reach the bottom of the curve if you made the learning rate too small because the learning rate is determining how far you move once you've determined the gradient.

On the other hand, if the learning rate is too big, you could oscillate and move from above the minimum to below the minimum to above the minimum to below the minimum, and so on. So determining the learning rate is important. You can do it by trial and error. As I say, there's some clever ways of doing it as well. OK. Next slide, please.

It's also turns out to be best to scale the variables when you're doing all of this. And backpropagation is used to calculate the partial derivatives. That's what I mentioned before, which is really just saying the partial derivatives. You're talking about taking the partial derivative of something, which is a function of a function. We've known how to do that in calculus for quite a long time.

OK. Next slide, please. So here's one application that we've used it for. May seem a little artificial, but actually it's not, as I'll explain in a minute. But basically, I generated 10,000 call option prices using the Black-Scholes-Merton model, and added a normally distributed error. So I generated a lot of data. I said that the asset prices, the strike price K, the risk free rate, the volatility sigma and the time to maturity.

You can see in the second bullet point there what I chose as the ranges there. So we sampled parameters randomly and then added a normally distributed error term to the Black-Scholes-Merton model and then said, let's see how well the model can actually determine what the function is and actually used three hidden layers, 20 neurons per layer and sigmoid activation function except the final layer.

And this is a point. If I use the sigmoid activation function everywhere, it wouldn't have worked because sigmoid activation function gives you a value between 0 and 1. And of course, the Black-Scholes-Merton price is not between 0 and 1. So I used the. identity activation function for the final layer.

And that's not uncommon to use the identity activation function for the final layer because the activation functions you're using for earlier layers may be constraining the answer you get. And divided the observation between the training set, and validation set, and the test set in the way that we were discussing earlier, 6,000 for the training set, 2,000 for the validation set, 2,000 for the test set. We have the next slide, please, Peter.

So you can see what happens. The epochs of the training set is increased, which just basically means we continue to try and match the training set data as closely as possible. It improves things for the validation set up to a certain point and then starts to get worse. See what's going on better. Next slide.

We can take a moving average. And this is-- so you can see what's going on here. We keep trying to improve the model. But we stop after 2,575 epochs. Just think about the epoch as a trial. So you can see that things improve for the validation set up to a certain point. And then they seem to get worse so it's best to stop.

So the model that we actually wound up with was the model after 2,575 epochs. Next slide, please. So we only had 10,000 observations. But we found that the neural network actually imitated the Black-Scholes-Merton model very well. In other words, the random noise that we added to the Black-Scholes-Merton prices didn't-- it managed to overcome that random noise pretty well. OK. There's more details about that in my book. So the next slide.

Now you may say, well, that's a pretty artificial example. We don't need a model to-- we don't need to use neural networks to determine the Black-Scholes model because we have a formula. But it turns out that, actually, derivatives companies are using neural networks to value exotic derivatives.

There are some derivatives which can only be valued using Monte Carlo simulation. And the Monte Carlo simulation is slow. It can take three or four minutes to come up with a price. If you've got a client on the end of the line, that may not be very good.

So basically, instead of doing this for the Black-Scholes-Merton model, you could do it for some exotic derivatives model. I mean, you run your Monte Carlo simulation many, many times to create data, which relates the price to the inputs. And then you create a neural network to replicate the prices as closely as possible.

And then after you've done that, you have a really fast way-- I mean, obviously you don't-- if you're doing this in the way that I'm describing, you wouldn't actually add an error term to it. You just want to create the prices as accurately as possible. And once you've got a neural network that will actually create these exotic option prices, you can then very, very quickly calculate any new price you want to because all you're doing to work to calculate a new price is working forward through the neural network. So instead of being three or four minutes, it's almost instantaneous because, as I say, you work forward through the neural network to obtain the price.

And this can also be useful for scenario analysis because scenario analysis can be very slow if you're having to calculate this exotic option over and over again using Monte Carlo simulation. But if you've constructed a neural network to produce a price, then it's much faster and scenario analysis is perfectly feasible.

Can we have the next slide, please? Another thing we've looked at as part of our research is we've looked at the relationship between volatilities and how volatilities change as the return on the asset changes.

The standard way of doing hedging, of course, is to say, well, I mean, let's talk about delta hedging, but it applies to other hedging as well. With delta hedging, you say, well, how does the price of the option change when the price of the underlying asset changes?

And we'll assume that the implied volatility remains the same. That's the standard assumption. Implied volatility remains the same. And you just calculate delta as the rate of change of the option price with respect to the asset price. But perhaps there's a tendency for the volatility to change as the asset price changes. In other words, if there's a positive return on the asset so the asset price goes up, there's less overall-- the company is less highly levered.

At least in the market value sense, the company is less highly levered and so maybe the implied volatility goes down whereas, if the return on the asset goes down, the company becomes more highly levered and therefore this tendency for the volatility to goes up. So maybe we should take this into account when calculating delta. So can we have the next slide, please, Peter?

So we calculated a neural network to investigate this and to see what's the relationship between the daily asset price return and the change in the implied volatility. And so input layer had moneyness, time to maturity, daily asset price return. And the output layer had change in implied volatility. Next slide, please.

These were all the details. In the interest of time, let's move on. Next slide, please. And we can see that we actually, in this particular case, found that validation sets started to get worse after 5,826 epochs. You kind of see that a little bit in this diagram so we stopped the training after 5,826 epochs. Next slide.

At first, we were pretty disappointed with the result of this research because the test set, which is final result, it actually only gave an 11% improvement over a simpler analytic model that my colleague Alan White and I had worked on in 2017. So it didn't give a hugely better relationship between the return on the asset and the volatility.

However, we played around with it a bit. And when the VIX index was used as a feature, there's a further improvement of 60% And what we found was that the behavior of the volatility surface was different in high and low volatility environments.

In other words, if you want to predict what's going to happen to the volatility surface so you know how to change the implied volatility when you change the underlying asset price, you really want to know whether you're in a high volatility environment or low volatility environment. So this actually gave kind of an interesting result, which you'd be unlikely to get just playing around with the data. Next slide, please.

We've got 20 minutes left. So let's use the last 20 minutes to talk about reinforcement learning, which has been a subject of quite a lot of my work over the last little while. Next slide, please. So what is reinforcement learning? It's concerned with finding a strategy for taking a series of decisions, rather than just one.

So you're in some sort of an environment. It's changing in an unpredictable way. And you have to decide-- well, you have to decide what decision you're going to take today, but you know you're going to have to take another decision tomorrow and another the next day. You know you're going to be taking a series of decisions, not a single one.

And this has actually been a hugely successful-- it's kind of like a trial and error kind of approach to this, but it's been hugely successful. I'm sure many of you have heard of the success that software has had in playing games like chess and go. It's actually we developed software that can beat the best human players of chess and go. I used to play a lot of chess when I was younger. And in a way, it's disappointing. But computers can now do this so much better than human beings. Next slide, please.

So the general model is you have actions, states, and rewards. So you take an action. You start with a certain state, which I call S0. You take a certain action at time 0. And that moves you to a new state and maybe you get a reward. Then you take another action, which will move you to yet another state, perhaps give you a reward, and so on.

So you've got to look ahead and think of all the actions you're going to have to take, not just at time 0, but at future times. And the states at future times are somewhat uncertain. Next slide, please.

So one of the fundamental-- I mean, this is really a trial and error process. But if you-- and obviously, you don't know anything when you start. So you're going to be choosing some of a random strategy. And then slowly, over time, you learn that doing this in this situation works out well, whereas doing something else in that situation works out badly. And this is where exploration versus exploitation comes in.

At any given time, there's a certain probability that you'll randomly choose a decision. OK. That's referred to as exploration. And there's a certain probability that you'll choose the decisions that actually works out best so far, which is called exploitation.

So initially, the probability of exploration is 1 because you don't know it. So you're going to randomly choose a decision. And then over time, you'll slowly reduce the probability of exploration and increase the probability of exploitation. And after a sufficient number of trials, you'll probably decide that, actually, the probability of exploitation should be one. In other words, you've pretty much decided that you know what the best decision is in all different situations. Next slide, please.

So I don't know whether any of you have come across this game Nim. OK. But it's a very simple game, but it's a good game for illustrating how reinforcement learning works.

So suppose we've got two players. So I'm playing against you. And there's a pile of matches on the table. And we take turns. Each of us can pick up one, two, or three matches. And you lose the game if you have to pick up the last match. OK. So pile of matches. Take turns picking up one, two, or three matches. And the person who has to pick up the last match loses.

Well, if you are being good mathematicians, you'd work out how to play this game very quickly. Basically, I mean, let's suppose there are only five matches on the table and it's your go. There are five matches on the table and it's your go. I'm going to win because if you pick up one match, I'll pick up three matches.

And you have to pick up the last match. If you pick up two matches, I'll pick up two matches and you have to pick up the last match and so on. So if I can leave you with five matches on the table, I win. Then we can go one stage beyond that. Let's suppose I can leave you with 9 matches on the table. Then I can make sure I leave you with five matches next time, because, again, there are matches on the table, you pick up one, I pick up three.

You're then left with five matches on the table. You pick up two, I pick up two, you're left with five matches. And very quickly you decide that, actually, the way to win this game is to always leave your opponent, if you can, with 4n plus 1 matches. So I mean, you know, suppose n is 5. 4n plus 1 is 21. So if I can leave you with 21 matches, I will win because I'll leave you with 21 matches. Then whatever you do next time round, I'll leave you with 17 matches. And then the next time round around, I'll leave you with 13 matches and so on.

OK. However, let's see how well the machine does at working out this best strategy. And I'm going to assume there's only eight matches initially on the table although you'll find on my website, you can find some software for seeing how well the machine does in a lot of other situations.

But we'll assume that the reward is plus 1 from winning and minus 1 from losing. And your opponent behaves randomly. I can change things so the opponent learns as well if I want to. So basically, we start with this state action table. OK. State is the number of matches that are left, 1 to 8. And the action is the number I picked up. Next slide, please.

So basically, as I said, it's trial and error. So let's suppose-- I mean you don't know anything. So randomly, you choose to pick up one match. Randomly, your opponent picks up three matches. Randomly, you choose to pick up one match. Randomly, your opponent picks up three matches. You win.

OK. So we can update Q8,1, 8 being the number of matches on the table and the 1 being the number picked up. So the first argument is the number of matches on the table. Second is the number picked up. So we know from that that we have a trial where we win. Q8,1 leads to a win and Q4,1 leads to a win.

So we update Q8,1 and Q4,1. OK. And we have to have an updating parameter of the formula. Formula for updating is there. And we say, well, OK, we don't want to update too much. We'll give it 5% weight.

So we have the old Q SA plus 5% of the gain minus the old Q SA. And can we have the next slide, please? On each trial, we observe certain states take action in those states. And the gain used in updating the Qs can be either the final reward, referred to as the Monte Carlo method, or it can be the value of the next state, assuming the best decision is taken.

So if I pick up one match, and you pick up one match, then the value-- and there are six matches on the table, and in terms of updating, instead of waiting to see whether I win or lose, I could just say, how good is it for me if there are just six matches left on the table? And that would be the way we would do the updating.

So we start with the Qs all 0, and then we update them in the way that I was describing. Next slide, please.

And with the Monte Carlo method, which is where we wait and see whether we win or not, you can see-- I mean, we all know that with eight matches on the table, if it's our go, we pick up three and leave our opponent with five matches on the table. That's the best decision. After 1,000 trials, it hasn't really found the best decision. It still looks as though picking up one match is marginally better. The Q value is 0.562 as opposed to 0.522. After 5,000 trials, it's definitely beginning to look that it's best to pick up three matches and after 10,000 trials for sure.

So you see what's happening here. We're just doing more and more trials. And what we did here was we had this epsilon parameter which determines whether you behave randomly or you use the best decision that seems to have worked out so far. We have epsilon starting at 1 and having this decay factor of 0.9995. So it does appear that reinforcement learning finds the best decision. Next slide, please.

And in fact, it finds it even more quickly with temporal difference learning. This is where you look to see how much does it turn out to be worth to be in the next situation that you're in after you've taken the first decision? And it turns out that, as often is the case, temporal difference learning works better than the raw Monte Carlo simulation. But it's definitely worked out that it's best to pick up three matches.

It doesn't look so bad to pick up two matches or one match because, remember, your opponent's behaving randomly. And when you've learnt a lot, you can behave much better than your opponent. I've done other tests where the opponent is learning as well, in which case picking up two matches or one match is going to allow your opponent to win. And so you have negative numbers opposite the 2 and the 1. Next slide, please.

This Nim game illustrates, admittedly in a fairly simple situation, how reinforcement learning works. Very often you've got a much more complex state action table to be filled in. You have to combine reinforcement learning with artificial neural networks because you've got this state action table. You're able to fill in some of the numbers in the state action table but not all of them. But you know there's some general non-linear function which is describing the whole table. And you have to do the best you can with the numbers that you do have on the table. Next slide, please.

OK. So five minutes left. What I've been spending a lot of my time in the last little while on is saying, can we use reinforcement learning for hedging derivatives? The traditional approach to hedging is, of course, Greek letters where you calculate delta, gamma, vega and so on. And you're really looking at a very short period of time ahead because you're saying what's the partial derivative of the value of my portfolio with respect to whatever it is, the underlying asset price, volatility, or whatever.

Reinforcement learning has a different approach. It says let's look several periods ahead and try and take a decision which will work well over our time horizon. So what have we found? Next slide.

This is my last slide. So what we found is we found that, in some situations, using reinforcement learning for hedging-- in other words, looking several periods ahead and seeing which strategies will work out best-- can be useful. For vanilla options, it does about the-- overall, it does about the same as the Greek letter hedging. But it does save transaction costs and actually saves significant amounts of transaction costs because you can think about it this way. You're hedging a vanilla option.

I mean let's just talk in terms of delta. I mean, let's suppose delta is 0.5 right now. And delta goes up to 0.7. But maybe there's a 50% chance that your next hedging date, delta will go back down to 0.5. So the question is, should you change? Should you do your hedging to reflect delta going from 0.5 to 0.7 even though you know there's a 50% chance it'll go back down to 0.5 or is it better to wait and see what happens over perhaps two time periods?

And the answer is if there's a lot of transaction costs, it's better to wait. So we find reinforcement learning can save quite a lot of transaction costs even though, when you don't take account of transaction costs, there's not a huge saving for vanilla options, typically. And of course, transaction costs are really important when it comes to vega and gamma hedging as well.

But some exciting options such as barrier options we find reinforcement learning produces superior results, even when there are no transaction costs. And one of the nice things about using reinforcement learning is that you can choose your objective function whereas, of course, you can't do that with Greek letter hedging.

So we've experimented with looking at the tail of the distribution. And VaR95 is a shorthand for you want the 95th percentile point of the loss distribution to be as high as possible. Or CVaR95 is where you want the conditional value at risk, which is the average beyond the 95th percentile of the loss distribution to be as low as possible. So you can have an objective function that focuses on the things you're most concerned about, which is that perhaps the tail of the distribution.

We find it's robust, gives good results during stress periods. And it's interesting. JP Morgan is our main competitor in terms of doing this research is JP Morgan. JP Morgan's been doing a lot of work looking at reinforcement learning for hedging. Of course, the disadvantage of reinforcement learning is that it's much more computationally demanding than traditional approaches. You've got to run these trial and error programs, which can take a few minutes, but they're getting faster.

So whereas it takes no time at all to calculate a Greek letter, it will take you a little while to calculate the best strategy for doing reinforcement learning hedging. But I think, to come back to my earlier point that computers are now sufficiently fast that it's becoming a feasible tool.

So brings me to my last slide. We're one minute over 4:00 mark. But if anybody wants to ask any questions, I'd be happy to answer them.

PROFESSOR: Well, I have a question for you, John, about the choice of activation functions. Have you found, as you did in this case, that I guess it was the sigmoid function was applied in your example or was it the ReLU?

JOHN HULL: PROFESSOR: Yeah. We've had a lot of success with the sigmoid function. That may be just chance events or maybe there's something special about the sigmoid function. I mean, of course, there's nothing to stop you trying several different activation functions and seeing which works out best in a particular situation.

And we did that in our implied volatility research. And we concluded that sigmoid function works worked out best as a matter of fact. But we tried several different activation functions. But I think, if I remember correctly, all the activation functions we tried were pretty close to each other in terms of how well they worked out. So I think choosing the activation function may not be, in many situations anyway, may not be a big deal.

What you also have to think is how big do you want the models to be, how many neurons per step, how many steps in total, and that sort of thing. I think that's very typically what happens.

PROFESSOR: Is there a way to benefit from building the neural network with, say, two layers and then use the results of that to add a third layer, or is the three layer neural network just built independently?

JOHN HULL: That's a good question. As far as I know, you've got to start from scratch when you go from, say, two layers to three layers. You can't use what you've done with two layers as a shortcut to doing three layers. I may be wrong there, but I don't think there's any way.

PROFESSOR: Right. Actually, for the class's benefit, I checked your references. And on your Toronto website, you actually have material like the Python code for various examples in your book. So I think students might find that very accessible and useful.

JOHN HULL: Yeah. Yeah. I should have mentioned this. Yeah. There's a lot on my website. I think lots of different things we've done at different times on machine learning research. So if you want to read more about it, certainly my website would be a good place to go.

PROFESSOR: How do you choose the objective functions in your examples? Was there some issues involved with choosing those specifically?

STUDENT: And what was the difference between them?

JOHN HULL: Yeah. I think every situation is different as far as that's concerned. You've got to think what objective function fits in with what I'm trying to do in the particular situation that you're working on. OK. I mean, if you've got some exotic option and you're trying to match the price as closely as possible, then, obviously, it's the difference between the price you're trying to match and the price given by the neural network is going to be most-- there's going to be an objective that you're trying to minimize.