Artificial intelligence, machine learning, and deep neural networks. These are terms that can spark your imagination of a future where robots are thinking and evolving creatures. In this video, we're going to look at reinforcement learning, or RL, as I'll sometimes abbreviate it. It's a type of machine learning that has the potential to solve some really hard control problems.

You may have heard that the AI company, Deep Mind, created a program called AlphaGo. It's an AI that uses reinforcement learning to beat the world's best go players, and then they recently created Alpha Star, which is poised to dominate the StarCraft 2 scene. So naturally, you might be thinking, well, if it can do that, why can't I use reinforcement learning to control my robot or cool my data center or stabilize a drone in a highly dynamic and turbulent flow?

Well, let's talk about that. Now, a quick warning before we begin. This is not intended to be an exhaustive look at reinforcement learning. I wouldn't be qualified to explain it all to you anyway. Instead, I want to introduce this topic from the point of view of a traditionally trained controls engineer, and I hope to show you that there's actually a lot of overlap with control theory.

By the end of this series, I think you'll be better prepared to answer questions like, what is reinforcement learning and why should I consider it when solving my control problem? How do I set up and solve the reinforcement learning problem? And what are some of the benefits and drawbacks of reinforcement learning compared to a traditional controls approach?

All right, so with the scope of this series set, let's get to it. I'm Brian, and welcome to a MATLAB Tech Talk. Let's start by thinking about the complexity of building a walking robot from the perspective of a traditional controls approach. We might use cameras to view the environment and then extract image features that can be converted into signals like position or locations of obstacles.

We could combine those observations with other sensors that complete the state estimation, which we then use along with a model of the plant and environment to design the control system. And more than likely, this would consist of multiple control loops that all interact with each other. For example, there would be low-level motor controllers and high-level controllers that are managing the leg trajectories or the robot trunk trajectory. And maybe a higher-level controller that is managing the balance or off-nominal behavior.

And everything has to work together in an uncertain environment to generate this complex movement of walking, which can be really challenging. Instead of all this

complexity, let's squeeze it down into a single black box that simply takes in observations and outputs the low-level motor commands directly. If we were infinitely smart, we could sit down and design a function that could get a robot to walk without concerning ourselves with all of the internal steps along the way. But since we're not, that's where machine learning comes in.

Broadly speaking, machine learning can be subdivided into three categories: unsupervised learning, supervised learning, and reinforcement learning. Unsupervised learning is used to find patterns or hidden structures and datasets that have not been categorized or labeled. For example, imagine you collected information on 100,000 animals--like a bunch of physical attributes and social tendencies.

Then you can use unsupervised learning to group the animals or cluster them into similar features. This can be something obvious like grouping them into mammals and birds, or to group them by patterns that might not be as obvious like finding correlations between physical traits and social behaviors that you didn't know about ahead of time. Supervised learning, on the other hand, is different in a subtle way in that we train the computer to apply a label to a given input.

For example, let's say that one of the columns of our dataset of animal features is the species. We can then treat species as the label and the rest of the data as inputs into a mathematical model. Then we can use supervised learning to train our model to correctly label each set of animal features by inputting them in one at a time, letting the model guess the species, and then systematically tweaking the model based on whether that guess was correct or not.

And if we had enough training data to get a reliable model, we could then send through the input features for a new animal, one that we don't have labeled, and our trained model would apply the most probable species label to it. And supervised learning is probably the type of machine learning that most people are familiar with because it's what allows computers to recognize pictures of cats or your friends in photos.

And fundamentally applying a label to an image is exactly the same problem as applying a label to a dataset of animal features. We input a bunch of training images into the model, and then we tweak the model based on whether it guessed correctly or not until it's accurate. The difference is that the input data for an image is just a stream of numbers representing pixel intensities, so it's not as straightforward to understand how that relates to a cat.

This is what deep learning is good at. By representing the model as a deep neural network, we have an efficient way to input thousands of numbers and then tweak it during trainings that it can identify features within an array of pixel intensities that will ultimately allow it to apply the correct label. Reinforcement learning is a different beast altogether. Unlike the other two learning frameworks which work with a static dataset, RL works with a dynamic environment and the goal is not to cluster data or label data, but to find the best sequence of actions that will generate the optimal outcome.

Optimal in this sense means to collect the most reward. It does this by allowing a piece of software called an agent to explore, interact with, and learn from the environment. The agent can take an action which affects the environment, changing its state, and the environment then produces a reward for that action. In using this information, the agent can adjust which action to take in the future. It can learn from this process.

And although you're presumably not a piece of software, you learn in essentially the same way a software agent learns with the reinforcement learning framework. You can be thought of as an agent and the world around you is the environment that you can interact with, observe its state, and collect rewards. You get rewarded by the environment by taking actions that are good, like you went to college--action--and you got a job--state--and that job pays well--reward.

Or you looked both ways before crossing the street, action. You got to the other side, state. And you didn't get run over in the process, reward. Alternatively, you get low or negative rewards for taking actions that are bad. Like you stayed up late before an exam, action. You're tired, state. And you received a poor grade, reward.

Within the agent there is a brain that takes in state observations, the inputs, and maps them to actions, the outputs. And in RL nomenclature, this mapping is called the policy. Given a set of observations, the policy decides which action to take, and just like with supervised learning, we can represent the policy as a deep neural network, which we'll see later allows our agent to input thousands of states at once and still be able to come up with a meaningful action.

This is where the term deep reinforcement learning comes from. In a walking robot example, the observations might be the state of every joint and the thousands of pixels from a camera sensor. The policy would take in all of these observations and output the actuated commands. And if the robot stays upright and continues walking, the environment would generate a reward, telling the agent

exactly how well that very specific combination of actuator commands did.

Of course, the policy might not be mapped correctly to take the best actions or the environment might be slowly changing, and so the mapping is no longer optimal. And this is where reinforcement learning algorithms come in. They change the policy based on the actions that were taken, the observations from the environment, and the amount of reward collected. In this way, the goal of the overall agent is to use reinforcement learning algorithms to modify its policy as it interacts with the environment, so that eventually, given any state, it will always take the most advantageous action, the one that will produce the most reward in the long run.

For example, if you were tired for your exam and you received a bad grade, well, you learn from it, and you adjust your policies so that you won't stay up late before the next exam. Now, at its heart, reinforcement learning is an optimization problem, but there are some very interesting concepts that set reinforcement learning apart from other optimization techniques. First is the idea of value.

Reward is the instantaneous benefit of being in a specific state, whereas value is the total reward that an agent can expect to collect from that state and onwards into the future. Assessing the value of a state rather than assessing the reward helps the agent choose the action that will collect the most reward over time, rather than a short-term benefit. For example, imagine our agent is in this situation and is trying to collect the most reward within three steps.

If the agent looks directly at the reward for each action, then it will step left first to get a higher reward and then right and then left again to ultimately collect plus 1. However, if the agent is able to estimate the value of a state, then it will see that going right has a higher value than going left, and will ultimately end up with plus-8 reward. Of course, often the promise of a high reward in the future still might not mean that the action is the best, and there's at least two good reasons for this.

One, like with the financial market, money in your pocket now can be better than a little more money in your pocket a year from now. And two, your prediction of rewards further into the future become less reliable, and therefore, that high reward might not be there by the time you reach it. In both of these cases, it's more advantageous to be a little more short-sighted when estimating value.

And in RL, we can control this by discounting rewards by a larger amount the further they are in the future. Another critical aspect of reinforcement learning is

the trade-off between exploration and exploitation when interacting with the environment. This is the trade between collecting the most rewards that you already know about versus exploring areas of the environment that you haven't visited yet.

For example, let's say the agent only knows about the two rewards immediately adjacent to it. If it took the greedy approach by exploiting the environment, it would only go after the highest reward it knows about, and so it would go left to collect the 1. However, if we occasionally let the agent explore the state space--even at the risk of collecting fewer rewards--it can fill out more of its value function, and it opens up the possibility of finding higher rewards that it didn't know about.

And this is part of our normal learning process as humans as well. A simple example is deciding on which restaurant you want to eat at. Do you choose a restaurant you know you like and therefore exploit your knowledge? Or do you venture out and explore a restaurant you've never been to before, increasing your knowledge?

Now, trying a new restaurant gives you the opportunity to find a new favorite place, but it also increases your chances of getting a meal that you don't like. It's tricky to settle on the perfect balance between exploring and exploiting, however, at the very least, RL algorithms provide a simple way to set that balance. OK, it's starting to feel like RL has a completely different goal than what control engineers are trying to do when we design control systems, but it is pretty much exactly the same problem.

We're trying to figure out how to design the controller, or the policy, that maps the observed state of the plant, or the environment, to the best actuator commands, the actions. And when we design a controller, we're basically doing a one-time policy update. And one of the ways that we can design an optimal controller is by minimizing a cost function, like we do with LQR, and cost is just the negative of reward, so by maximizing the reward, we're solving the same problem as minimizing cost.

The difference is that with reinforcement learning, the computer tries to learn the optimal behavior over time, rather than have the designer solve for it explicitly. It's like the adjustment mechanism in an adaptive controller, where it's tweaking the parameters at each sample time. In this way, we can essentially design a controller, which is a policy, without knowing anything about the system itself. And without having to solve any of the traditional control problems, we just let the

computer learn the right parameters on its own through a process that you can think of as fancy trial and error.

Now, even with the learning algorithm doing most of the work for us, we can't into this process completely ignorant. We have to know several things before we start, and the first is that we need to understand our system that we're trying to control and determine whether it's better to solve the problem with traditional control techniques or with reinforcement learning. And if we choose the learning path, then we need to set up the policy so that it has enough parameters--and in the right structure--so that it can be tweaked successfully. It won't do us any good if we're hoping to control a multidimensional system but only give it a single parameter.

We also need to know what a successful result would be, and reward the controller for doing well. This requires crafting a reward function so that the learning algorithm understands when it's getting better and ultimately settles on the result that you're actually looking for. And third, we need to apply an efficient algorithm that looks at the reward and the system state and knows how to tweak the parameters so that the process converges with any reasonable amount of time.

This is where we would also set the parameters for exploration and exploitation and the discounting on future rewards. In the next few videos, we'll expand on all of this by looking at the workflow of reinforcement learning in more detail. We'll look at the structure of the policy and introduce neural networks, and we'll talk about how crafting the proper reward function impacts your final result, and we'll look at a very high overview of some interesting learning algorithms.