[Music]

Welcome to the fourth lecture of the fifth week of the machine learning course, this lecture will be about reinforcement learning, this lecture will be divided into three parts. So we start today with part 1, the introduction. For the two first sub teams we have this so far discussed this week Explanation based learning and Inductive logic programming, these approaches stems from a tradition of computer science and logic. What they have in common with what we are going to talk about today is that they learn not in a vaccum, from a set of examples but on the brink or border of a reasonably strong like already existing domain theory. Reinforcement learning the topic of today comes from another academic tradition, actually reinforcement learning methods as I look like now were very much inspired and have been growing out from a long tradition in control theory, but as for the other system we talked about reinforcement learning is not learning in a vaccum, is actually learning on the border of an existing system with an existing domain theory. The intuitive scenario for reinforcement learning is an Agent that learns from interaction with an environment to achieve some long term related to the state. The state is a state of the environment of course also with the agent within it. So this iterative learning takes place as a series of actions on behalf of the agent and by a systemic feedback from the environment on each action. And the mapping from States to possible actions which means is actually what controls the actions in each state is called the Policy. The achievement of goals is defined by Rewards or Reward signals, those being the feedback on the actions from the environment. So this terminology is very important actually it's highly recommended I would say in this area to try to really, thoroughly learn the terminology because these few basic concepts will come back and come back in all kinds of forms when you look at very many different kinds of algorithms in this area. So apart from reward which is the atomic feedback on a single action, there is also another important concept or two related concepts, so in this area a lot of the work is centered around not just a single action by this sequence of actions and sequence of actions we typically call episode and an important concept then related to rewards is the accumulative to reward over an old episode. So it's a sum not really sum because the way you aggregate the rewards can differ but at least one can say it's the aggregated rewards overall episode, and that we call a Return. An action sequence is the set of actions from a specific state to something we call a terminal state and of course this has to be defined from problem to problem one means to be a terminal state. So the goal of most algorithm in this area is to establish a policy, policy for an agent for how to act so that it maximizes the returns across all possible action sequences. There are many examples today about reinforcement learning, actually many success stories described so on the front page you can see some slides really referring to the Alphago system (06:03) which is one of the success stories where reinforcement learning, implementation actually created a world champion level and Co playing program, also there are there examples on this slide you can see a small robot that should manage to navigate the through a maze assuming that this robot is provided with reinforcement learning system.

As I said it's highly recommended to really learn the terminology for this area because it will simplify all further work. So let's now again go through that, term by term so an environment here is a micro world defined for the particular reinforcement learning problem including the agent and many times when this is referred to and we use the letter E. Then we have an agent often designated as A. State is a particular configuration of the agent within the environment and typically here we use the letter S to refer to it. Terminal States is defined end States for a particular reinforcement learning problem, so for every domain the terminals times have to be clearly defined. And then we come to actions, so an agent selects an action based upon the current state and the policy P, which is hat (08:11)and the policy P is a mapping from states of the environment to the potential actions of an agent in those states. So policies can be deterministic which means that the policy depends only on S or it could be stochastic which means that the policy also depends on a, related to that there is another concept called Transition Probability Function which is actually the function from as an A onto a new state S prime and this specifies the probability that that environment will transitions to state S Prime if the agent takes action A instead S. An episode also sometimes called a epoque is a sequence of states actions and rewards which ends in a terminal state and the reward gives feedback from the environment on the effect of a single action A in state S is leading to S prime. Discounted reward is concept that means that when we calculate accumulate rewards over a whole episode, we want to implement some intuition that what we do so early in the sequence have more weight than that we do later in the sequence. So therefore we won't put a kind of discount factor on the later steps and the action steps in the sequence, so typically the discount factor lambda is a number between 0 & 1. The return is the accumulated reward over an episode, so finally then the value function or is the estimation of the estimation of the value or utility over state S with respect to its average return considering all possible episodes within the current policy ending as always in terminal states. So this value function must continually be re-estimated for each action taken. So finally the model of the environment to refer to two things already mentioned is that the model of the environment is considered on one hand as the relation, the function T from S and A, to a new state S Prime. And the rewards associated with all those steps.

To rehearse a little on the terminology once step more, we introduce a little example called four times three world, which is actually a little table and it's a board of four times three positions indexed by two coordinates, so the agent has a start position in one point 1, the proposition two point two (2.2) is excluded it's like it's forbidden, there are also two rewards, I mean actually that can be in reward for any step taken in this this kind of domain but in some domains it's more common that you get a reward in the end, so maybe one can say that two categories here there are these kinds of domains where you gather a reward when you reach the terminal State or there are these domains where you get rewards more or less for every step you take, so both those variants are possible. So here in this case you can see it's it's one position for 4.3 that has a positive reward of plus one and there is no one another one that has a negative reward or minus one. Actually there is no fixed rules for the ranges of rewards, so that can be it's up to the designer and design for a particular problem, so for all other reaching all of the positions give a reward zero. And the action is to move up down left and right, of course the board as itself restricts what actions to that can be taken. So the policy is deterministic which was already mentioned I mentioned earlier here so and just to repeat it is I mean in the sense that every action in a specific take can only lead to one other state. In the stochastic case one action can lead to different states and that can then need to be some probability for in which state it's the same action and up with in. You can also see in this example some exemplification of what an episode is, and you can see what return those episodes gives in the end, I mean here it's very simple because only the last step gives the reward.

As I've already said there is a strong inspiration on the error reinforcement learning from a trophy (15:19) or even if the definition or characterization of what reinforcement learning is has been developed within the machine learning and artificial intelligence feel the kind of the models used and the methods used bear strong resemblance to those traditionally used in control theory, so therefore it's not surprising that the way of describing and modeling reinforcement learning scenario is inspired by such one model coming from the control theory field which is Markov decision process but we here abbreviate MDP. So in a Markov decision process we actually then have that kind of setup that we more or less already described for reinforcement learning, so we can say that an MDP is like a 4-tuple of state or States, actions, rewards and transitions where S is the set of states, A is the set of actions possible for each state that the agent can choose from defined by a policy P and then we have the reward function which is their feedback or on each action leading to some other state and then we have a transition probability function specifying all these probabilities from S to state S given in action A. And there is one thing that there is important to note that the coupled to this model is this assumption that when we look at the probability, the state probability we can forget all the paths that led to this state, so the Markov property says that the transition probabilities depend only on the states, not on which path that led to that state. So also in this kind of framework we have the goal for all kinds of algorithm with this is to find policies P that maximizes the return, which is the expected future cumulated possibly discounted reward for episodes starting from one state and moving to a terminal state and to the right you can see an exemplification of the state and she said well you see the probability is just an example of probabilities or specific, for a specific case yeah and it's exactly the same example that we looked at and this four times three board example.

So now it's time to come to a very important distinction for this area. So in one MDP scenario and I would say this default one, we have a complete an exact model on the Markov design process, in the sense that the transition function from states and actions to new states and the reward function from states and specific action are fully defined. So as also the various domain description available that that can completely specify those two functions or relations and so in this case where we have complete knowledge available the MDP problem is a planning problem, that can be exactly solved by use of example techniques like dynamic program. However in many cases these two relations T and R are not completely known. So this means that this information is not available from the domain with meaning that the model of the MDP is not complete, and this case is that what we truly call reinforcement learning, that the complete case is more of a standard problem in control theory which can be handled by standard techniques like dynamic programming and is essentially a planning problem while the case but with not complete knowledge is a learning problem.

As a reference point let us first look into how the case with complete knowledge is handled so that we have a reference point. So as I said the standard technique for handling this situation is called Dynamic Programming which is an element design technique for optimization developed by Richard Bellman in the 1950s. So like divide and conquer dynamic promise simplifying a complicated problem by breaking it down into simpler subproblems in a recursive manner combining the solutions to the subproblems to form the total solution. If a problem can be optimally solved by breaking it recursively to subproblems and then form the solution from optimal solution to the subproblems then is said to have optimal substructure. However there are other ways or methods for breaking problems into pieces like divide and conquer in that case the problems are supposed to be totally independent in dynamic problem and can allow subproblems to be to some way to in some sense dependent on each other. So this way of thinking about reducing a problem into subproblems also affects the way we think when we want to estimate, for example the value function, the utility function of a certain state in a Markov decision process. So typically we define the value function recursively in terms of the value function for the remaining steps of an episode, and also typically we do the same when we find a way of calculating estimate some of the policy function needed. There is also an important equation that defines what is an optimal value function in this complete knowledge situation and that equation is called the Bellman equation.

Before we continue I must tell you a funny story, not so long ago somebody asked me actually why is this method called dynamic programming, and then I had to say I don't know actually I always accepted the name I just know what it is but then because I got the question I really tried to search for the reason behind the name and then funnily enough I found this little piece from some old article, actually professor Bellman himself in the 50’s wrote that you know the reason I worked on a project sponsored by an by the US military and at that time it wasn't really popular to do something very theoretical or if you had funding from the state especially from the military, so therefore you always had to defend what you do so it useful for the purpose you were funded. So actually as this story goes, Bellman choose the name that looks very supposed to be look very practical so we couldn't be accused of doing something theoretical or – mathematical. So that explains why maybe this name is a little difficult to understand the relevance of.

So the main principle of dynamic programming as a way to recursively divide up a problem in smaller parts and take the solutions to the smaller problem to the subproblems and combine them to the solutions of the larger. So this way of thinking also them affect the way we run in dynamic programming plan the calculation of the various important functions. So as you can see here there are two functions of value function and the policy function, and both these functions are defined in a recursive way. So this means that the value function our state S is actually the sum of all episodes starting in S but of course via different actions leading to other states as S Prime, actually taking the probabilities for going in these various directions but for each direction take the sum of the reward for taking that direction add the discounted value of the value function in the new state. So you see you see you start from the discounted value of the state you're going into, you have the reward you multiplied by the probability to get the full picture, and in a similar way you do with the policy function now the thing is for the policy function you want advice on which argument to take, so therefore logically you look at the same kind of expression, the value function but you want the R max which is the A, that gives the position and the action A that gives the highest value of the value function, which is the reasonable action to take. But also this then equation have this recursive structure. So then we come to something called Richard bellman principle of optimality so that says that an optimal policy has the property that a variable initial state an initial decision are the remaining decision must constitute an optimal policy with regard to the state resulting. So whatever step you take initially still in order to get something totally optimal, even the rest must be optimal. On coupled to that, we also have the Bellman equation, which is an equation for specifying what could be an optimal value function and as you see for that the structure of that also is the same recursive structure.

Just because via Bellman equation have an equation that gives the criteria for finding an optimal solution it doesn't matter mean that we have one, so what remains is to find methods that that solves the bellman equation. We don't have much time to go into this at this point, so on this slide I roughly mentioned a few variant approaches something called Value Iteration where we actually through a simple procedure iterate the values, or policy iteration where we do the same for the policies. Actually it's always the case that if you have a you can always infer one from the other, so if you have an optimal value function it's pretty straightforward to infer an optimal policy and vice versa. There are other techniques you can use as linear programming, but we will not go more into this. I will know shortly introduce to you a simple example to illustrate the thinking of dynamic program, so this is actually not reinforcement learning, it's just an example to illustrate the recursive style of reducing larger problem to smaller problems and then combine the solutions, so this is actually a graph and every edge as a value or a weight and you can say it's a distance here so the task is to find the shortest path. Actually as you can see here one can easily see in this simple example the shortest path is nine, but if you have a general problem you need a method and for example if you have a greedy method the greedy method always starts where you are and take the best choice locally, but never backtrack. So in this case for a greedy method you take one from S you take the edge we wait one you come to A the then you will take the edge with weight 4 but then when you come to D you only one way to go and unfortunately you take it and need to take an edge with weight 18 so then you will go with a with a total weight of 23 which is absolutely not optimal at this point. However what you can see on the next line is that the dynamic programming approach is not greedy, it's rather more like breadth-first so actually what dynamic programming does is systematically in parallel checks all paths recursively, and actually dynamic programming in this case well in contrast to the greedy algorithm provide you with a good approximation.

So what we will do for the rest of this lecture is that we will accept that for the case where you have full knowledge you can use dynamic programming and techniques related to that but for the other case where you don't have full knowledge you have to rely on other methods or maybe reduce them to a problem where you have full knowledge and then you can continue. So we will not make an attempt to generalize the Markov decision processes in a direction that makes them more useful to the case where your lack full knowledge, however there are such extension and on this slide you can see one such attempt so it's called partially observable Markov decision process and actually what the extension is very simple but because you also introduced you notion of observation, so apart from getting a reward at every state an agent also can make an observation which is then a partial picture of the environment, actually it turns out but if you introduce this you this kind of model is more useful also for the case with not complete knowledge but for time reasons we will not go into this sidetrack either. So this is the end of part one this is soon to be continued in part two thank you