Artificial Intelligence

Prof. Mausam
Department of Computer Science and Engineering
Indian Institute of Technology-Delhi

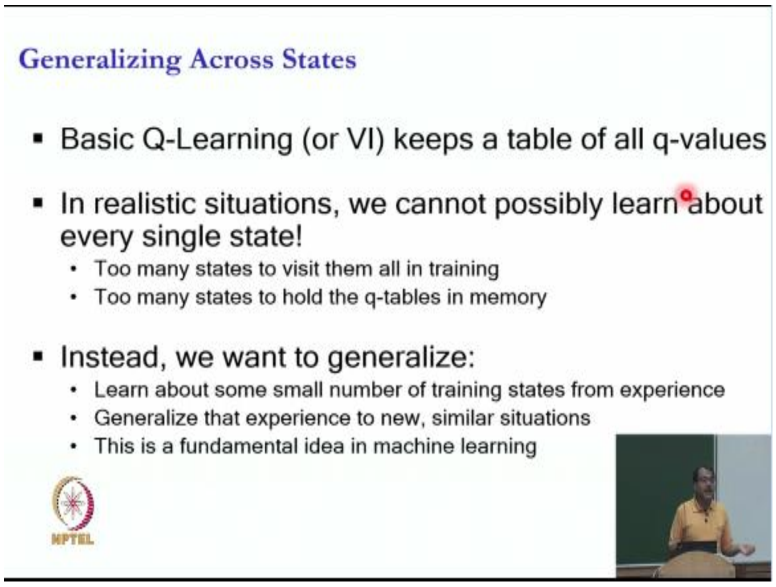
Lecture - 83
Reinforcement Learning: Generalization in RL

Now I have given you lots of algorithms, TD learning, Q learning, this that. None of them work in practice. Absolutely, none of them work in practice. In practice is an important word. Why do they not learn? See if you go back and you know this you should have been asking me not me reminding you that you should ask me when we started doing search we said that the search space is usually extremely large.

It is so large that we cannot even represent it in the memory. We cannot even run a Dijkstra's algorithm which is why even shortest path computation we do using a star and what not where we are expanding only a small number of states. But then when I came to uncertainty I said there is a state space. There is action space, there is a policy that is a mapping from states to actions. I am always maintaining the state space.

You should ask me. How can I do this? State space is in state space too large and the answer is, is it too large? Absolutely, it is too large. It is always going to be in practice some they will have some features in your problem. And then your state space will become you know exponential in that feature space. So all this business about memorizing a policy, doing value iteration, doing Q learning. These are good for the books. But they do not work.

(Refer Slide Time: 01:55)



Generalizing Across States

- Basic Q-Learning (or VI) keeps a table of all q-values
- In realistic situations, we cannot possibly learn about every single state!
  - Too many states to visit them all in training
  - Too many states to hold the q-tables in memory
- Instead, we want to generalize:
  - Learn about some small number of training states from experience
  - Generalize that experience to new, similar situations
  - This is a fundamental idea in machine learning

And the reason they do not work is because in realistic situations, we cannot possibly learn about every single state. Just not going to happen. Too many states, too many states to visit them in training, too many states to hold the Q table in memory. So what do we do? Now let us look at the state of the class. The state of the class is where Vaibhav is sitting on chair A5 and Shaheen is sitting on chair B19 and Parth is sitting on chair C16, etc., etc.

And I am standing in front of these guys. This is my state of the class. Some people are absent, of course. Does that mean I should teach you differently? No. Right? If tomorrow Sian decides to not come or come and sit in chair A15 should I teach you differently? No. It is possible that I cannot make some jokes because I have to make jokes on specific people and that person is no longer in the class.

So I might make small changes in my narrative. It is possible that there is some big event happening because of Facebook and nobody shows up in my class, then do I teach differently? Yes, maybe only four people show up. Maybe I will say let us skip today's class because more people will suffer if I teach today. So I will decide what to do based on you and us the state of our class.

And suppose my throat is bad. I just cannot speak. No voice is coming out of my mouth. Will I teach differently? Yes. I might not teach. I might show you a video. I might say no class today. So some features are more important than other features. My ability to speak, my physical presence in the class is unfortunately more important than yours, right. But individual small changes in the state do not really matter very much.

Some things like where you are sitting completely does not matter. Some things regarding how many of you are present do matter a little bit. So actually I do not need to train myself doing many classes again and again and again with each of you sitting differently and trying to teach again and seeing what is the reward. That is just too much to expect.

If I have to learn the value of every action in every state independent of the other state I am being making a fool of myself. It is too many states, too many actions, infinite samples of s, a pairs do not make sense in practice. So what do I need to do? What do I need to do? You want to generalize. Imagine you do training with some classes. Let us say I taught once UL 3, 3, 3 class. I taught another COL 3, 3, 3 class.

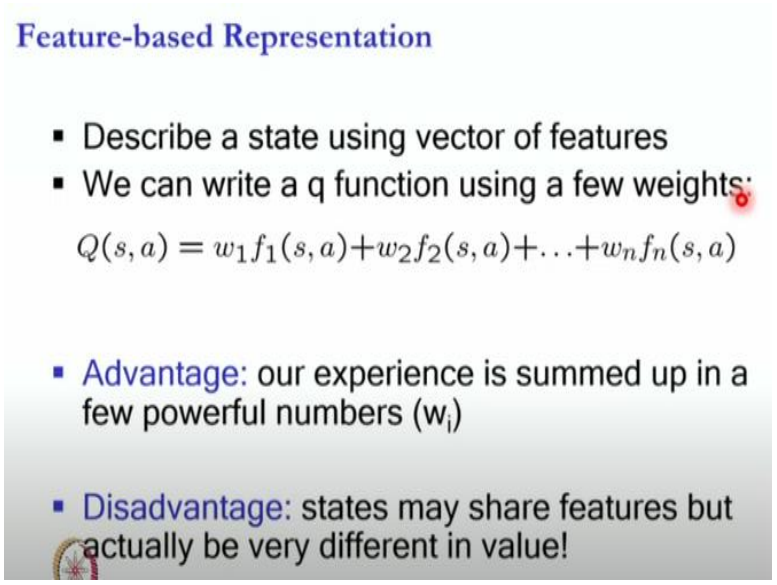
Let us say I taught this a couple of times. Now the third time I teach I have sort of figured out how do I teach, what works better, what works worse. It will still keep improving. I will still continue to learn but the specific features of the state not all of them may be important. So I need to say that this state is very similar to that state. So the action that I take in this state should be very similar to the action that I take in that state.

Or that this state is very similar to these 2, 3, 4 states. And the action that I take in the state should be very similar to the actions and an aggregate of that of what I take in those states and so on so forth. So I need to generalize my experience. In fact in real world in real life I never get the same state twice. Why? Because time is constantly moving. So if you include time in the state space, I will never get the same state twice.

But for many problems like taking an exam, the actual time does not matter only the relative time matters. So I need to generalize experience to new similar situations. This is the fundamental idea of machine learning. In machine learning I am usually given some training data and I have to learn a function to predict a particular thing. Whatever, predict whether the image has a dog in it.

Predict whether the person uttered the word box in it, whatever right in the audio, whatever classification task you have. Is this email spam or not? So we now need to say that in order to make any of this practical let us do generalization. And how do we do generalization when we use the same idea as we used in Minimax.

(Refer Slide Time: 06:58)



Feature-based Representation

- Describe a state using vector of features
- We can write a q function using a few weights:

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Advantage: our experience is summed up in a few powerful numbers (w_i)
- Disadvantage: states may share features but actually be very different in value!

So we describe a state using a vector of features. Now we can write the Q function which is what you were writing as Q s, a separately a separate parameter was being trained. Now we will hypothesize that Q function has a specific functional form. That functional form is let us say a sum of basis function representation. The basis functions are features, f 1, f 2, f 3, f n. And each feature is more important or less important.

So each of them has a weight w i for the feature f i. And we take summation w i f i as the representation for Q. Is this an approximation? Yes, and it also depends on how good your, how good you are well, so there are two things; how good your weights are and how good your features are. And in the previous version of machine learning who gave the weights? The machine and who gave the features, the human.

So we had this in the division of labor. Human came up with some features like on this chessboard how many queens do I have? How many pawns do I have? How many pawns do I have very close to you know the queening move how many etc., etc. How many checks can I give in one step? How many checks whatever can I receive in one step. All of that are features and a chess expert designs them.

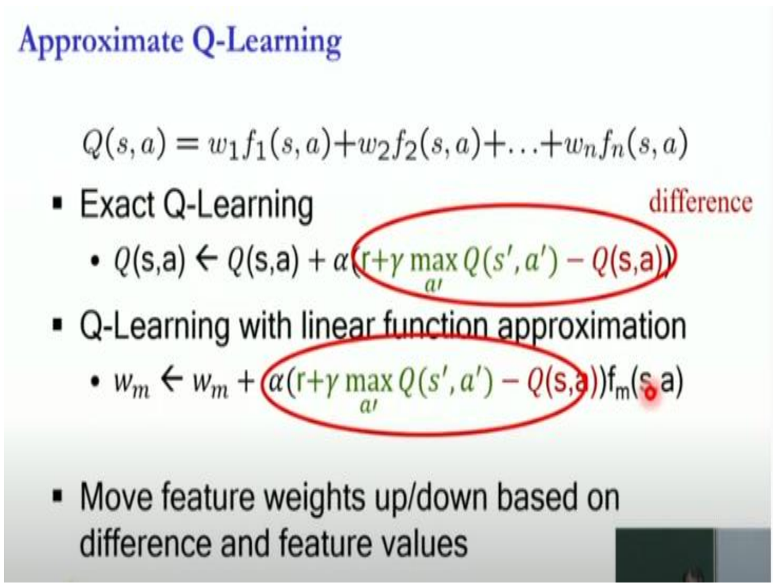
And something similar was happening in Go also. So some human expert will design those features. And then if your features are very good, then maybe you will be able to learn good weights. And that is a machine learning problem. The machine will say okay give me lot of games. Give me the features. I will try to give you the best values of w i’s. And it is possible that the, it will still be an approximation.

Because it is possible that Q function is actually not a linear function of f i’s. So that is also possible. But the advantage of this approach was that our experience was summed up in a few powerful numbers w i’s. If my human designer gave 20 features I have to learn 20 numbers, big deal. If they are smart to give me 100 features well I still have to learn only 100 numbers. Still big deal.

See if you think about the number of configurations in the game, they will be just more than the number of atoms in the world. But all of that data, I am compressing in a few small features. So that is not bad. You know let human designer come up with a 1000 features 100,000 features. It is still much better than the large state space we have to deal with and humans can only come up with so many features.

Now disadvantage is that states share features, but they actually will be may have a very different value. So in fact, so therefore you cannot do too much with this approximation, but you know we have to live with that. So let us see how we used to do it in the previous world and then I will talk about how we do it in the modern world.

(Refer Slide Time: 10:03)


Approximate Q-Learning

$$Q(s,a) = w_1 f_1(s,a) + w_2 f_2(s,a) + \ldots + w_n f_n(s,a)$$

- Exact Q-Learning
  - $Q(s,a) \leftarrow Q(s,a) + \alpha \underbrace{(r + \gamma \max_{a'} Q(s',a') - Q(s,a))}_{\text{difference}}$
- Q-Learning with linear function approximation
  - $w_m \leftarrow w_m + \alpha(r + \gamma \max_{a'} Q(s',a') - Q(s,a)) f_m(s,a)$
- Move feature weights up/down based on difference and feature values

So in the previous world for exact Q-learning, this used to be my equation. Q s, a is equal to Q s, a plus alpha times the estimated value minus the new value. Now I am not maintaining Q s, a separately. In fact, let us say I have some w i’s that I have already estimated. That means I can compute Q s, a and Q s prime a prime by plugging in the actual feature values for s, a and multiplying with the appropriate current weights and computing some value Q i or Q s, a.

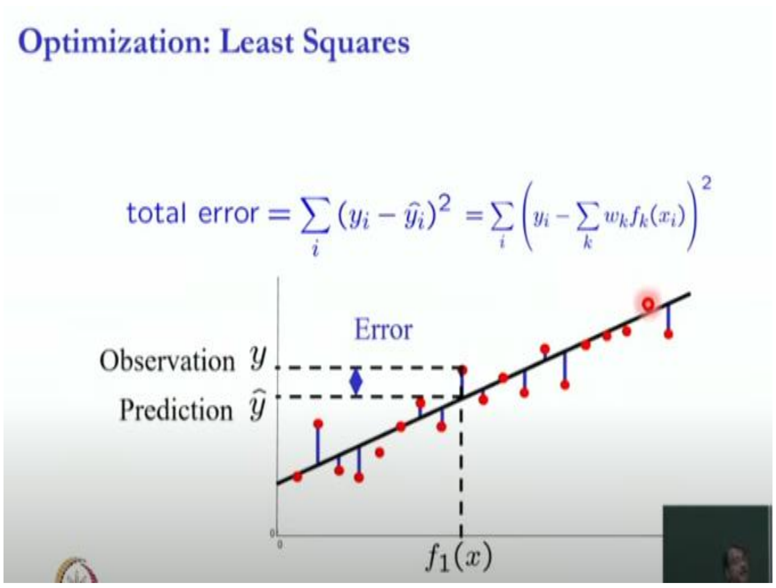
So again, this is computed, this is computed, right. This is the difference that we have to maintain. What we do is and what parameter do we have to change? We have to change the weight parameter because those are my free parameters. My machine learning has to figure out the weights. So what we do is that we said w m the mth weight for the mth feature is nothing but the previous value plus alpha times the difference multiplied by the feature value.

So this difference is the same as in the previous equation except now it is computed as opposed to freely estimated. But we multiplied with the feature function. Now why do we do this? Let us make sure that we agree with this. So what happens? Let us say my value needs to increase. Value of Q s, a needs to increase. Now let us say the value of Q s, a needs to increase then which weights need to increase?

The weights which were positive in the weights for which the features were positive in s, a. And which weights need to decrease? The weights or features which were negative in Q s, a. So by multiplying this term with f m I figure out the sign, the direction much better. Because if f m is positive this weight increases. If f m is negative this weight decreases, which is what we wanted to do.

But you can say why did we multiply with f m? Why not square root of f m? Why not square of f m. Why not log of f m? Why f m? We agree. You can say we agree that my sign is right. The direction is right, but how do we figure out the magnitude? And there we need to use a little bit of theory. So the theory is like this. It is a theory of optimization.

(Refer Slide Time: 12:43)



Optimization: Least Squares

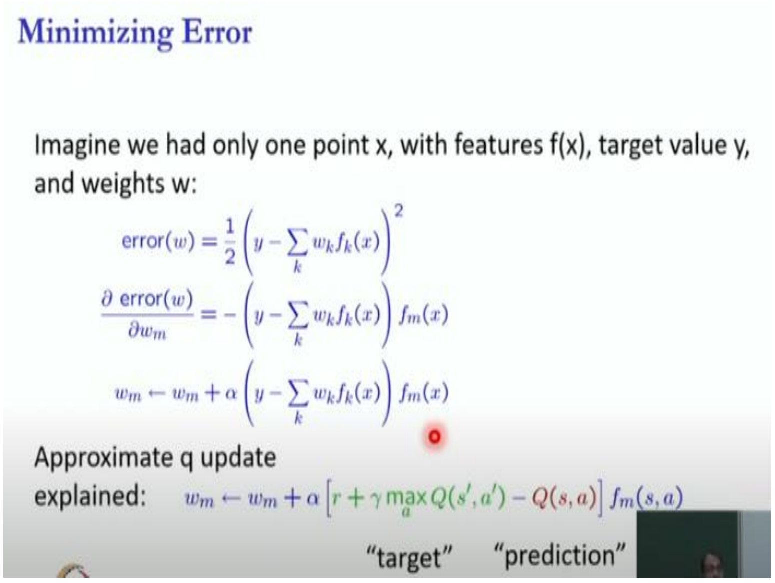
$$\text{total error} = \sum_i (y_i - \hat{y}_i)^2 = \sum_i \left(y_i - \sum_k w_k f_k(x_i)\right)^2$$

The optimization is and again, I am taking a using a different equation, different setting. Let us say I have x. I want to figure out the function f that finds y. Let us say I am given these observations in the red and let us say I hypothesize that y is equal to f x is a linear function. So y is equal to m x plus c. Now for a given set of m and c I will make some predictions. I will have a linear line and then I can figure out the error.

What is the error? The error is what is the y that is estimated? And what is the y that I should have predicted? That gives me the error. So y hat is what I predict, y i is what I should have predicted. This is my target value. And this is my predicted value; y i is target y hat is predicted. Now I say that I want to reduce this error irrespective of the direction positive, negative.

So I just square it and sum it. And I say that I want to minimize the sum of the squared errors. This you can write down as y i minus summation k w k f k, right. Because y hat is predicted by a sum of a linear function.

(Refer Slide Time: 14:00)



Minimizing Error

Imagine we had only one point x, with features f(x), target value y, and weights w:

$$\text{error}(w) = \frac{1}{2}\left(y - \sum_k w_k f_k(x)\right)^2$$

$$\frac{\partial\, \text{error}(w)}{\partial w_m} = -\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

$$w_m \leftarrow w_m + \alpha\left(y - \sum_k w_k f_k(x)\right) f_m(x)$$

Approximate q update explained: $w_m \leftarrow w_m + \alpha\left[r + \gamma \max_a Q(s', a') - Q(s, a)\right] f_m(s, a)$

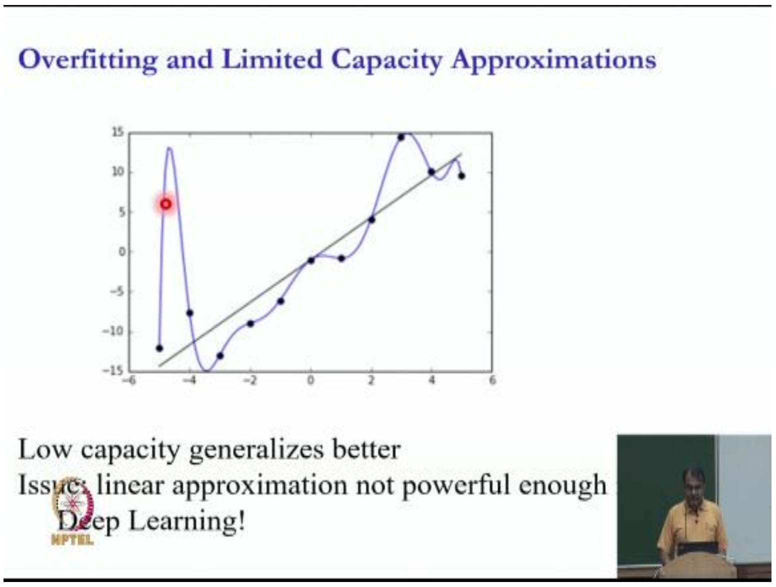
"target" "prediction"

So therefore when I minimize this error and let us say this is my error function y minus summation w k f k, I will take the derivative with respect to w because those are my weights. So I take the derivative with respect to w and I want to set it to zero. But what is the derivative? The derivative is nothing but y minus summation w k f k times f m x. Because weight k is derivative with respect to weight j is 0 for j not equal to k, okay.

I am doing this algebra very fast, but I will do it slightly slower next week. So therefore what I find is that when I do w m plus alpha times the derivative this becomes exactly the equation that I was interested in. This is the difference of the prediction. This is the current error multiplied by the feature value. So in case of approximate Q update, this is y which is r plus gamma max a Q s prime a prime the target value.

And this is the predicted value minus Q s, a and this is the difference and it gets multiplied by f m of x. So therefore if I have a squared error as my objective function and I want to find the right weights, then my derivative is such that it multiplies the feature weight to give me the magnitude of the change, okay. So this is the underlying theory for why we have this particular equation here, okay.

(Refer Slide Time: 15:43)



Overfitting and Limited Capacity Approximations

Low capacity generalizes better

Issue: linear approximation not powerful enough

Deep Learning!

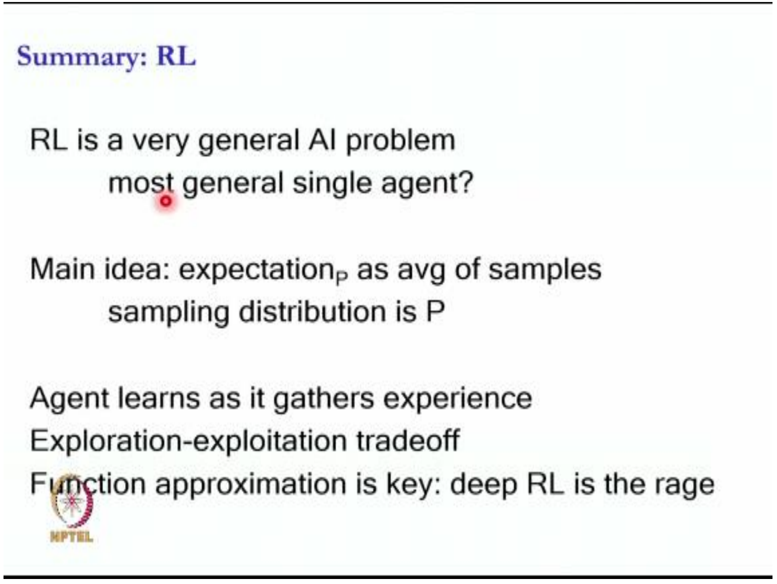
So last but not the least, linear function has some advantages and some disadvantages. It has some advantages in the sense that suppose my data is very noisy then a linear function sort of removes the noise and gives me the best fit line. So it will make a good prediction. Suppose my data was in fact linear and I just had experimental error, then I will compute the value of this point as somewhat reasonable.

On the other hand if I allowed it to fit a very general curve then it may over fit. Over fit means that it will fit perfectly but it will not generalize very well. Because now for this point, it may say it should be very high this value, but there is no evidence that it should be so high. So there is some benefit for having low capacity classifiers because they generalize better, but then they have low capacity so they cannot represent very much.

Of course linear approximation is not powerful enough in practice. You cannot argue. Their all features are affecting linearly my final Q value system. So what do we do? And in the most modern world we use deep learning. So this is where deep learning significance start to come. You have feature values and you want to fit a new function y or Q s, a or whatever and you do not want this function to be linear.

You do not want this function to be low capacity. You are letting it be high capacity. You are allowing yourself to have lots of parameters that is where deep learning comes into picture.

(Refer Slide Time: 17:22)



Summary: RL

RL is a very general AI problem
most general single agent?

Main idea: expectation$_P$ as avg of samples
sampling distribution is P

Agent learns as it gathers experience
Exploration-exploitation tradeoff
Function approximation is key: deep RL is the rage

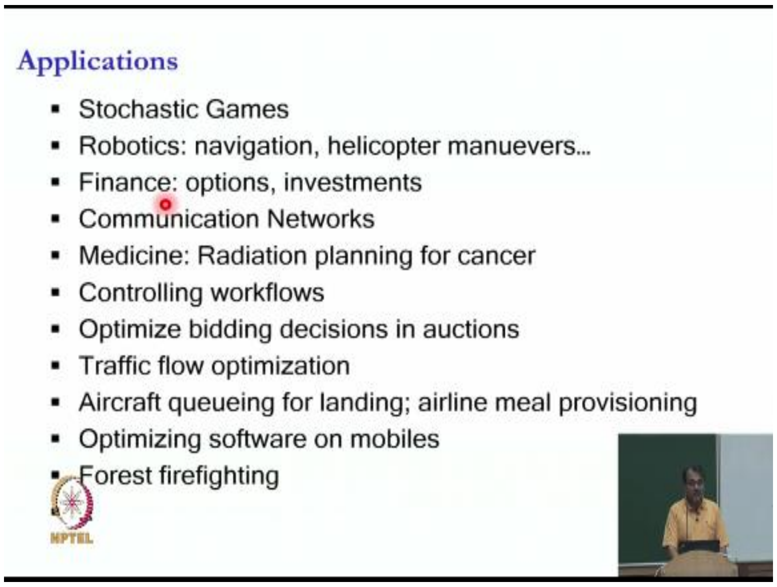
So let me summarize our topic for today, which was RL. RL is a very general problem, right? It is the problem where I have to take long-term decisions in an environment where there is uncertainty and I do not even know transition and reward function. So it is sort of most general single agent as long as it is not playing any adversarial game. The main idea for all the other algorithms is expectation is average of samples.

So this is the technical, core technical idea that you should keep at the back of your mind always in life. And then there were many interesting ideas like how we connect with people in terms of temporal difference and their happiness. How we connect with exploration exploitation trade-offs and so on so forth, right. So agent learns as it gathers experience. Agent learns as it optimizes its reward.

It does some exploration exploitation trade-off, but in actually making it work you have to do some kind of function approximation where deep learning will come into the picture. And therefore if you do deep learning for reinforcement learning that field of AI is called deep reinforcement learning and this is the rage today. RL was a very important community theoretically 5 years ago.

But not a very important community practically. But in the last 5 years after using deep learning after all this AlphaGo success and so on so forth deep reinforcement learning and interest in it has skyrocketed. Lots of people are interested in it and lots of very exciting ideas are coming out of that particular community today.

(Refer Slide Time: 18:51)



Applications

- Stochastic Games
- Robotics: navigation, helicopter manuevers...
- Finance: options, investments
- Communication Networks
- Medicine: Radiation planning for cancer
- Controlling workflows
- Optimize bidding decisions in auctions
- Traffic flow optimization
- Aircraft queueing for landing; airline meal provisioning
- Optimizing software on mobiles
- Forest firefighting

I should also point out that in the last set of lectures when we did Markov decision processes, I said that MDP's are used in lot of applications. But now I want to say that yes MDP's are used but not value iteration or policy iteration. Not planning style MDP is you never really know the you really know transition function or reward function.

For example, if you are trying to control cancer through radiation, you do not know the exact transition function of how much cancer tissue is going to burn and how much real good tissue is going to burn, how your body is going to react. You do not know the transition function. You can estimate it using reinforcement learning, but you do not know it explicitly.

If you are doing forest fighting you do not know exactly how you put how much water on the certain tree how much fire will get extinguished. You when you are working in a real setting you almost never know the transition function or the reward function. Sometimes you use a training step, learn these and then use VIPI which you can call model-based RL.

And sometimes you have it approximate and you let the model figure it out as it you know does better and better actions and that is where you do more interesting reinforcement learning. When you play real games this becomes even more important. You do not know with what probability Pac-Man will go left. You do not know with what probability Pac-Man will go, right?

Whatever interesting game you are playing you do not know when the enemy is going to you know come and kill you from where. You do not have all that probability transition function. So you learn it as you are playing the game. So RL is the most important, one of the most important AI problems. And we have almost completely done this except that now we will sub goal on deep learning.

Next three lectures we will talk of deep learning and then we will talk about how it fits in the big picture of deep reinforcement learning. Okay, so we will stop here.