# Artificial Intelligence

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

## Lecture - 91
## Deep Learning: Deep Reinforcement Learning

Good. So today we are going to wrap up the whole story, right. So if you think about it, we started some time back. We defined a Markov decision process. We said ah we can, if you are given transition and reward we can do value iteration. Great, life is good. Then we said but you know sometimes transitions and awards are not known. What do we do? We do Q learning, reinforcement learning, we do exploration exploitation trade off, we do all of that.

We still compute what is the best action and while we are taking actions in the environment, we learn reinforcement learning. Awesome, life is good. But then we said no, that does not work either. Why? Because we are maintaining a table of state action pairs for example, in Q learning. This is an s cross a table, your actions may be small, but your state space may be extremely large.

In fact, so large that you cannot even write it out forget even computing a Q table for it. Forget, you know visiting it infinitely often. So that is just not going to work. So we said okay, what can we do? Well, not every part of the state space is important, you know some features are not important some features are more important. Let us see if we can approximate the Q function as a function approximation over the features of the state as opposed to the state itself.

So we said okay, we can define a linear basis function representation, where the basis functions are given the features are given by the domain designer, we did this. But the problem was a linear is too not very expressive and b we required features from the domain designer. So what did we do? We said forget that. Let us do a better function approximation. So we learned deep learning.

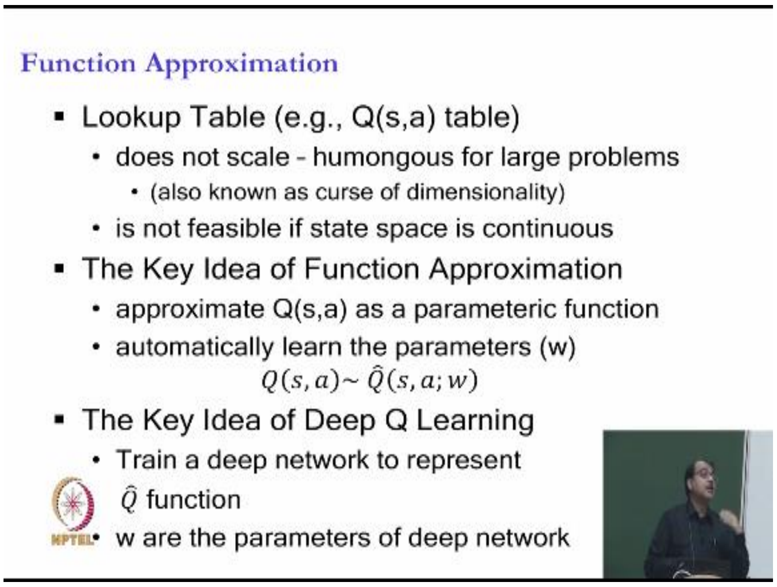
Deep learning says give me the raw input data, like a computer image, a vision image, sorry a digital image for example, a video or whatever. And the model will compute an approximation of the function to as closeness as possible, given the target and the parameters of the model. So great, we did that too, in the context of you know separate function approximation. Now all of this has to come together today.

What are we going to do? We are going to say look, now we can do Q learning with function approximation, where the approximation is a deep network. Exactly. Okay. And there is no way there is no magic going to happen today because it is a fairly advanced area of research. The first Q learning paper, deep Q learning paper came in 2015.

So think about 1977 Bellman's Markov decision process and now we have reached a point in 2015 in terms of our trajectory of AI. So we have come very close to the current state of the art, but that also means that it is fairly sophisticated. So I am going to touch upon it, I am going to just describe it at a very shallow level, I do not expect you to understand it deeply.

But you know later you can take an RL first and then you will have a better understanding of what is happening, okay.

**(Refer Slide Time: 03:25)**



Function Approximation

- Lookup Table (e.g., Q(s,a) table)
  - does not scale - humongous for large problems
    - (also known as curse of dimensionality)
  - is not feasible if state space is continuous
- The Key Idea of Function Approximation
  - approximate Q(s,a) as a parameteric function
  - automatically learn the parameters (w)

$$Q(s,a) \sim \hat{Q}(s,a;w)$$

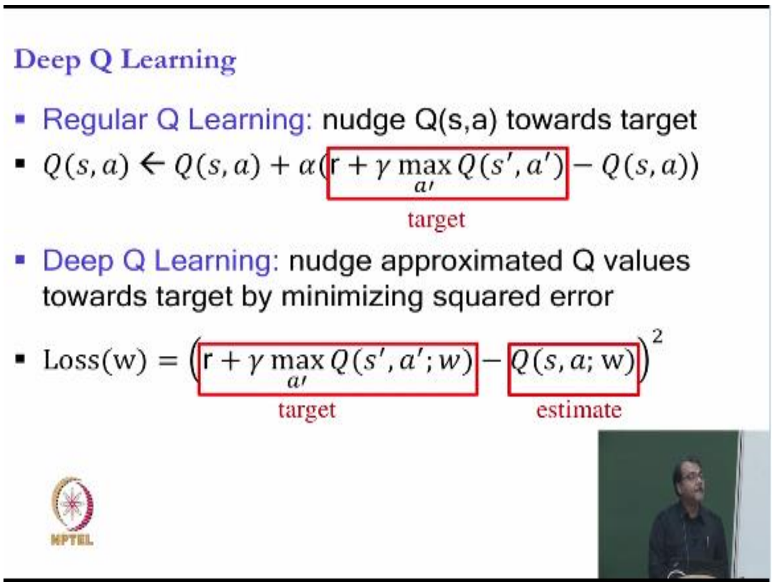
- The Key Idea of Deep Q Learning
  - Train a deep network to represent $\hat{Q}$ function
  - w are the parameters of deep network

So we have we are not going to do a lookup table of Q s, a. Instead, we are going to approximate Q s, a with the parameter w. So we have set of parameters think basis function representation. The idea is same, but we will think deep networks. So we have parameters w for the deep network w are the weight matrices and bias terms at different layers and so on so forth.

And essentially what is going to happen is that a state will go as input, a state action will go as input, and the deep network will do its magic and come up with a value, the Q value. It will not be the exact Q value, it would be the approximate Q value. That is why we call it Q, Q predicted or Q hat, right. And we are going to train this deep network. Of course, deep network needs training because the weight need training.

We are going to train this deep network such that it predicts the right Q values. Now how do we typically do Q learning and how do we typically do deep learning.

**(Refer Slide Time: 04:32)**



## Deep Q Learning

- Regular Q Learning: nudge Q(s,a) towards target
- $Q(s,a) \leftarrow Q(s,a) + \alpha(\underbrace{\boxed{r + \gamma \max_{a'} Q(s',a')}}_{\text{target}} - Q(s,a))$
- Deep Q Learning: nudge approximated Q values towards target by minimizing squared error
- $\text{Loss}(w) = \left(\underbrace{\boxed{r + \gamma \max_{a'} Q(s',a';w)}}_{\text{target}} - \underbrace{\boxed{Q(s,a;w)}}_{\text{estimate}}\right)^2$

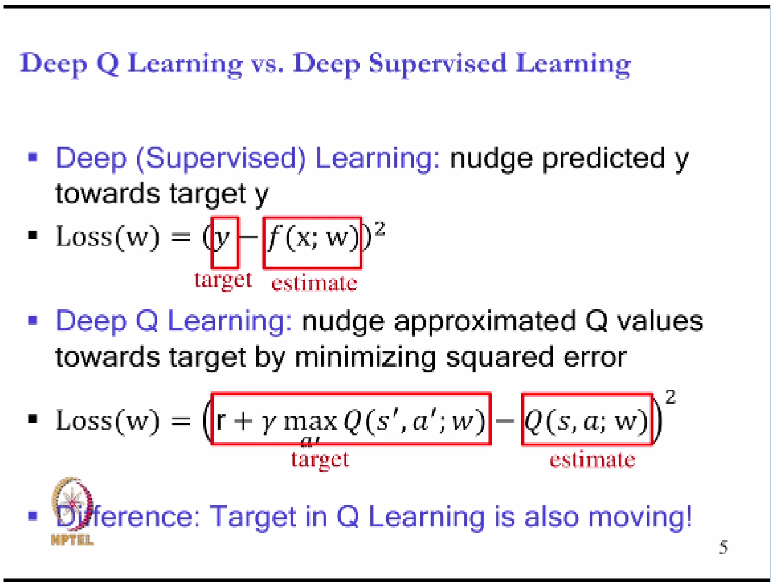
So if you remember, Q learning was nothing but give me a new target value. In this case by let us say one step look ahead. So the target value is typically the current reward plus max over a prime Q s, a prime a prime and this would be your current, the target value. And the old value was Q s, a and you want to nudge Q s, a towards the target. So you basically say old value plus alpha times the difference.

This is the standard temporal difference formulation for Q learning or TD learning or anything like that. On the other hand, you know that deep Q networks or sorry not deep networks basically are optimized for a given loss function. They make a prediction y hat, and they have the true value y the target, and they basically say, give me a loss function, let us say squared loss. So they say sum of squared of y minus y hat whole square.

Now let us reconcile them, it is not very hard. So basically, we now have the old value Q s, a but not the one that we were using in the lookup table, but the one that is computed by the deep network, so w parameters. And then we take some action a and then when we take an action a we get some immediate reward and we get we reach state s prime in the environment.

So our new target value becomes r plus gamma max a prime Q of s prime a prime like in typical Q learning except that again Q s, a prime a prime will be computed by the network. So now I have a new target value computed by the network while interacting in the environment. And the old value computed by the network and I want to nudge the parameters w such that the squared loss is minimized. This is called deep learning at the core, okay.

**(Refer Slide Time: 06:36)**



## Deep Q Learning vs. Deep Supervised Learning

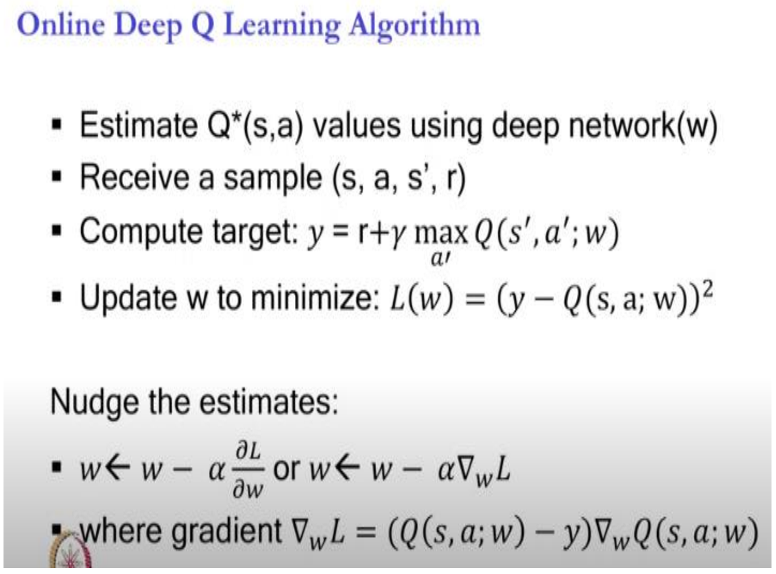
- Deep (Supervised) Learning: nudge predicted y towards target y
- $\text{Loss}(w) = (\underbrace{\boxed{y}}_{\text{target}} - \underbrace{\boxed{f(x; w)}}_{\text{estimate}})^2$
- Deep Q Learning: nudge approximated Q values towards target by minimizing squared error
- $\text{Loss}(w) = \left(\underbrace{\boxed{r + \gamma \max_{a'} Q(s', a'; w)}}_{\text{target}} - \underbrace{\boxed{Q(s, a; w)}}_{\text{estimate}}\right)^2$
- Difference: Target in Q Learning is also moving!

5

Now there are many problems to this. So but we will come to the problems. The main difference between a standard deep learning and deep Q learning or deep RL is that here my target is also moving. So think about it. Suppose I changed my weights, such that r plus gamma max a prime Q s prime a prime also became closer to Q s, a there is a problem. What is the problem? Now the weights have changed.

So the target itself has changed. See in typical deep learning the for an image the classification is given to you, the gold is given to you that is called training data. Here the problem is that the training data is generated by w and then that is being used to update w. There is a circularity here, but we will come to some fixes, right. So this is a major problem.

**(Refer Slide Time: 07:35)**



## Online Deep Q Learning Algorithm

- Estimate Q*(s,a) values using deep network(w)
- Receive a sample (s, a, s', r)
- Compute target: $y = r + \gamma \max_{a'} Q(s', a'; w)$
- Update w to minimize: $L(w) = (y - Q(s, a; w))^2$

Nudge the estimates:

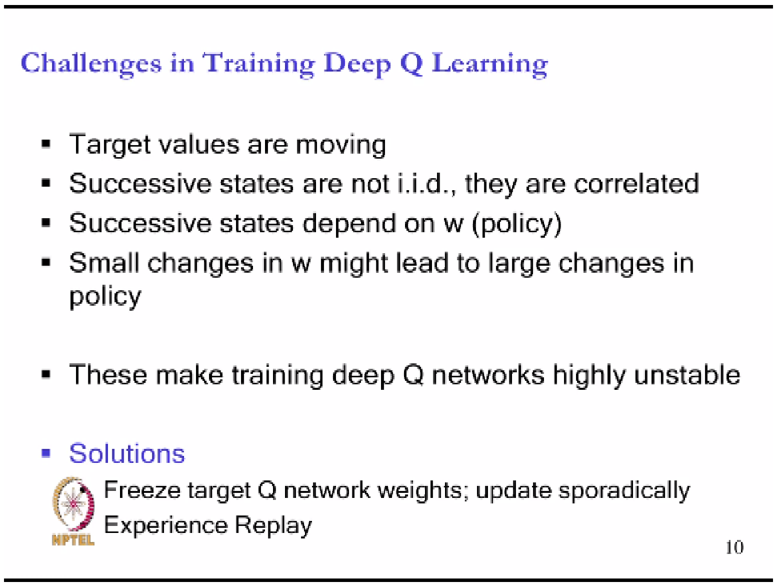
- $w \leftarrow w - \alpha \frac{\partial L}{\partial w}$ or $w \leftarrow w - \alpha \nabla_w L$
- where gradient $\nabla_w L = (Q(s, a; w) - y) \nabla_w Q(s, a; w)$

So first what is the final algorithm? The algorithm is you are in a state s you take an action a you get to state s prime you get some immediate reward r. Then you compute the new target. The new target would be r plus gamma max a prime Q s prime a prime computed by w. Then you update weights to reduce the error. The error would be this new target minus the old value whole square.

For that you would have to compute gradients with respect to w. And if you compute gradients with respect to w it will come out to be Q s, a w minus y times the gradient with respect to Q s, a w right and you will have a weight update which basically says take the w opposite to the direction of the gradient because this is a minimization option, right.

And because we are doing it one state at a time or a few states at a time, this would be a stochastic gradient descent although we can come we will come up with the mini batch version of it, okay. Did I talk about stochastic gradient descent in the when we did, okay I did. Just check.

**(Refer Slide Time: 08:45)**



## Challenges in Training Deep Q Learning

- Target values are moving
- Successive states are not i.i.d., they are correlated
- Successive states depend on w (policy)
- Small changes in w might lead to large changes in policy

- These make training deep Q networks highly unstable

- Solutions

Freeze target Q network weights; update sporadically

Experience Replay

10

Now come the issues right. So what are the issues? There are many issues. This is not an easy problem, right? What are the issues? Target values are not fixed. They are moving. Moreover, when we take a mini batch or when we take the next data point and take the gradient with respect to that data point, we assume that the data points are i, i, d independent and identically distributed.

Unfortunately, when you take an action a in state s you reach state s prime. Now the state s prime is highly correlated with s, because it is that state which you reached by taking a certain action in state s. So therefore, these data points are not independent. This is a highly biased data points, highly correlated data points and that also makes the problem harder and moreover, they depend on w itself.

They depend on the policy, which depends on the parameters. And moreover, if you have small changes in the parameters you do not know because it is a highly nonlinear function how the policy is going to change. It is possible that policy changes completely crazily, in which case training becomes highly unstable, okay. And this is 36,000 feet view. So there are two ways in which we deal with this.

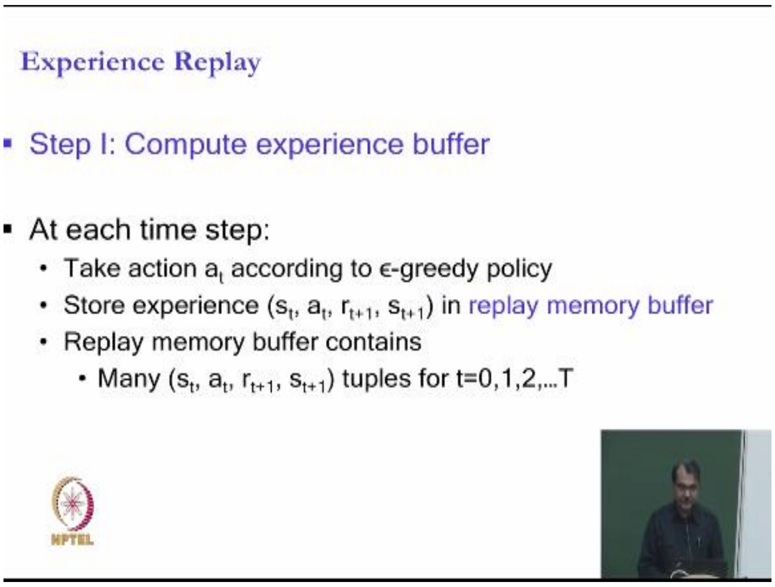
So problem number one, target values keep moving. Problem number two successive samples are correlated. And there are two different ways to solve this problem. These are not very unusual. If you think about it, how we did Minimax that we fixed the opponent, and then we do the best player and then we make the player the opponent and then we make the best player, right. So this is how Minimax used to work.

And the same idea is going to work here. So we will fix the target Q network by previous set of weights. It will say target does not move. Then we will come up with new set of weights, which are very similar to the Q learning loss which we minimize the Q learning loss. Then we will give this to the target and repeat the process. So that would be one solution.

And the other solution is because my successive samples are identically, are correlated, I am going to use a term called experience replay. It is like I have had some experience, I will go into my memory and I will bring out the experience and based on that I will learn. It is like that kind of a person who never loses memory.

So they remember exactly what they did at every point in time and now they know more about the world, they will go back to that memory and improve their parameter. Then go back to their memory and improve their understanding of the world. So that would be the two ideas. Let us quickly look at them.

**(Refer Slide Time: 11:32)**



Experience Replay

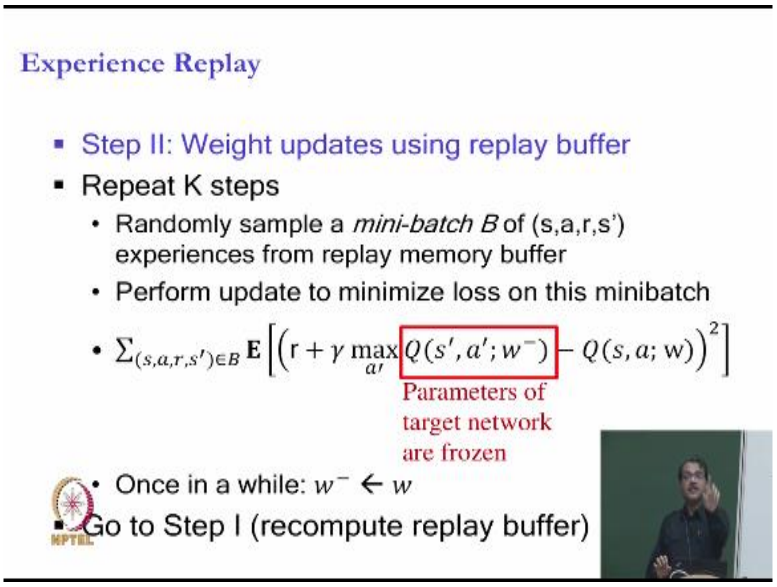
- Step I: Compute experience buffer
- At each time step:
  - Take action $a_t$ according to $\epsilon$-greedy policy
  - Store experience $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory buffer
  - Replay memory buffer contains
    - Many $(s_t, a_t, r_{t+1}, s_{t+1})$ tuples for t=0,1,2,...T

So in experience replay, we first create experience. So let us say we have a current set of weight parameters, because we have a current set of weight parameters. We have an epsilon greedy policy. We can use that epsilon greedy policy to all whatever exploration exploitation trade off policy to take the action a t in the state s t. When we do that, we get to state s t + 1 and get some reward in the environment.

We memorize it, we keep it in my experience buffer. So now I have lots of s i, a i, r i + 1, s i + 1 tuples in my experience. Now these are successively correlated. So s i and s i + 1 is correlated; s i + 1and s i + 2 are correlated. But, if I randomly pick the first and the 750th and the 973rd and so on tuples from this and create a batch then they may not be exact, I mean the correlation has become much less.

Because you have come a long way from where you started from. So that is the intuition that people use.

**(Refer Slide Time: 12:52)**



## Experience Replay

- Step II: Weight updates using replay buffer
- Repeat K steps
  - Randomly sample a *mini-batch B* of (s,a,r,s') experiences from replay memory buffer
  - Perform update to minimize loss on this minibatch
  - $\sum_{(s,a,r,s')\in B} \mathbf{E}\left[\left(r + \gamma \max_{a'} \boxed{Q(s', a'; w^-)} - Q(s, a; w)\right)^2\right]$

Parameters of target network are frozen

  - Once in a while: $w^- \leftarrow w$
- Go to Step I (recompute replay buffer)

So essentially they say that, I will randomly pick some points from this experience and I will call that my mini-batch. So my mini-batch is a set of s, a, r, s prime tuples. And on each of those, I can compute my Q learning loss based on the target minus the predicted except I do not want the target to move. So I will use my w minus which is the previous generation of the weight matrix for the Q network there.

I will keep it frozen. I will not move w minus, I will only move w. So basically what am I saying? I am saying find me those parameters w such that the one step look ahead Q function as computed by old parameters computes the same value as I compute with my current parameters w. Once I have found it I have got some internal consistency.

It basically says that the previous set of deep networks one step look ahead I have already encapsulated in my current w parameters. Now I will give this w to w minus and repeat. And so basically, you can think of it as every time I do this, I am going one more step in terms of my look ahead. So over time, I will increase my look ahead span. And I will do better at convergence to the optimal or convergence to near optimal.

Or we might even not be able to say whether it is optimal or not, because we may not even be able to express the function which is the true Q function. So it is always going to be an approximation, but in practice, it will have generalization. Now there are two questions. So let us do one at a time. Divyanshu. Right. So Divyanshu says my original problem was I cannot enumerate all states, but now my states are in my experience.

Do I need to enumerate all states in my experience? And the answer is, well, I am not giving a theoretical answer. But in practice no. The whole point was that I will be able to generalize about state features. So by giving these states and by learning this function approximation, I am hoping that the function approximation will try to capture the salient parts of a state.

And so if I have, if I had the test time I get a state I have never seen before, it will be still like some other states will have properties shared by some other states that were seen in the experience in the past. And based on that my function approximation would be able to give a good enough prediction to the value.

So hopefully, if we train this right and if the Q function is expressive enough, and if we have done all the bells and whistles so the training stabilizes, etc., I would have a much better value function than what I would have done if I did the basic Q level path.  Right. So good question. So Parth says how do you initialize these things, right very good. So what happens in this is you start always with a random initialization.

So when you do this, initially you have a random initialization, you give that to w minus. You freeze it. Then you do a different random initialization for w. You do this internal consistency, you have a new trained w now. You give this trained w to w minus, and then you randomly initialize w again and then repeat. Or you can choose not to randomly initialize.

You can do small perturbations in the original w and then you start from there. So basically you need a new starting point, think of it as local search random restart. So you have to come up with a good starting point, you can say that the w minus parameters are good, but I will just do some perturbation so that I can move away from the slight local optima that I may have reached, things like that.

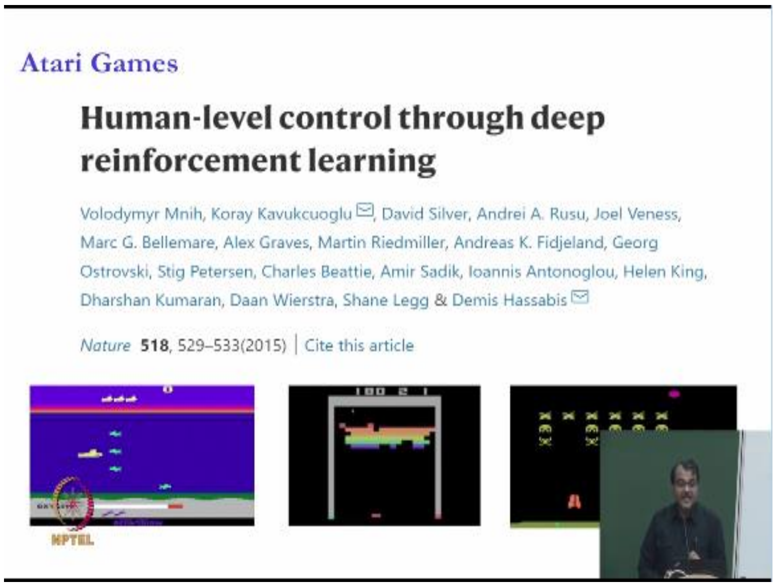
And then there are many things that people do in order to make these actually work. So if you really care for deep Q learning or deep, sorry deep reinforcement learning, deep Q networks are not the state of the art. In fact, I do not know exactly what is the state of the art. But one algorithm that has become extremely popular and famous is an algorithm called A3C. It stands for asynchronous advantage actor critic algorithms.

And you will have to read up a little bit on after Q networks, you will have to read up on policy gradient after policy gradient you will have to read up on actor critic and then you have will have to talk about what is asynchronous and what is advantage. So there are multiple steps to go there. Again, that would be best covered in a Q sorry in a RL course. Yes. How are we updating our experience? Well very simple.

After you have done one step you can recompute to experience working in the environment, right? So you have now a new set of parameters w minus. Those w minus define a new epsilon greedy policy. You recompute the experience by a epsilon greedy policy we have, right. So that kind of stuff. And you do not have to throw away the old experience. I apologize.

You do have to throw away the old experience, right. Sorry. Yeah. Because the distribution is different.

**(Refer Slide Time: 18:27)**


Atari Games

**Human-level control through deep reinforcement learning**

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg & Demis Hassabis

*Nature* **518**, 529–533(2015) | Cite this article

Okay. So this led to a nature paper and until 2014, you rarely saw a nature paper coming out of the AI community. They may have been exceptions again, I have limited time horizon. I started working in this field only in 2002. So but I did not see many nature papers in the field of AI in the last 15, 20 years.

Except in the last 4, 5 years where these catastrophic, amazing, exponential changes have been happening in the field. So this is work that came out of DeepMind. DeepMind was a startup company at the time. I remember that it was sold to Google's parent company around 2014, 15. I think it was 2014. Because I remember teaching my first course in IIT at that time.

And that is the time we had heard that Google has acquired these 20 researchers sitting in London, who do some work, which nobody knows what. It was not disclosed, who do some undisclosed work for an undisclosed amount. And in hindsight, Google made a very valuable buy, right purchase by acquiring DeepMind.

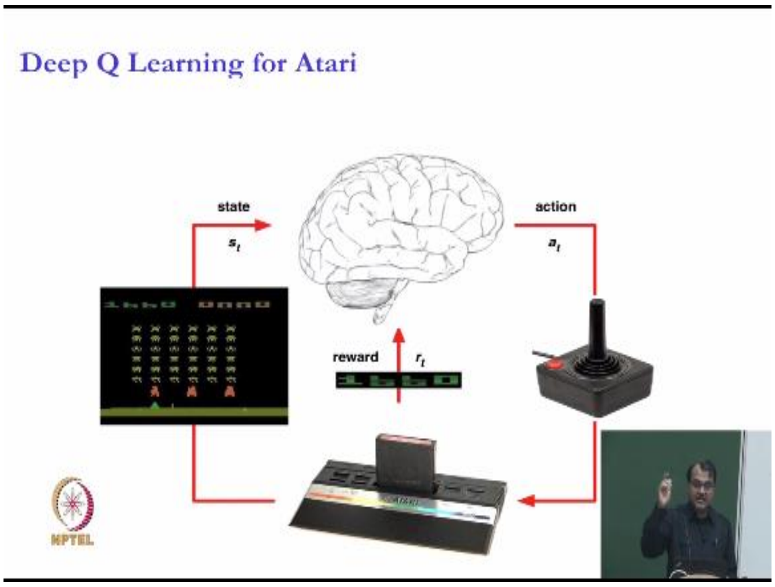
Because today if there is one company, which is at the forefront of deep learning and reinforcement learning, that would be DeepMind. Their nature papers was very cool. They created an Atari simulator, which I think existed even earlier. I do not know if you know Atari games, but when we were kids, there was a time we would play these, you know simple games where there were very simple joystick controls.

And you know there were not very sophisticated graphics and very basic games, but they were fun. And many of those basic games are now available as apps in your phones and so on so forth. What did they say? They said, let me take the original video sequence of the Atari game, not just one but all of them. And let me train a network which learns to play each Atari game.

A single network which learns to play each Atari game using deep 2 networks. It was unfathomable that we can do this. By that time deep learning had just started to come out. It was not very famous. And the fact that you can train this network to play games was amazing in and of, of itself. But by taking the original video input and not even taking anything about the game, not even saying this is the object, this is shooting, this is that.

It kills the nothing. Let the model figure it out completely from video. They just did one small thing.

**(Refer Slide Time: 21:43)**

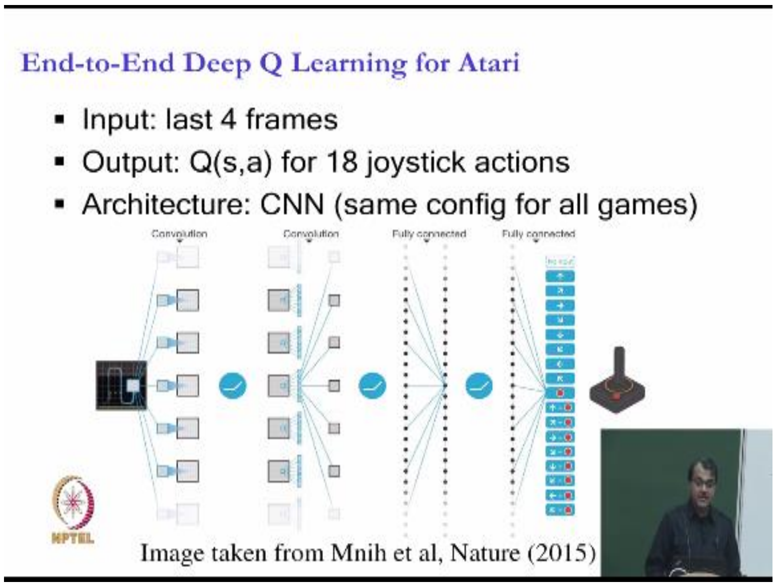


## Deep Q Learning for Atari

They said that this is my Atari simulator. It will send an a sequence of images to the deep network. The deep network will send an action. The action will go to the simulator and this will repeat. The only thing they did, only thing was that t the top where there is score, 1, 2, 5 written in characters, vision character, I mean pixels. They read that and explicitly fed that. That is it.

So the reward signal was a scalar. It was not an image. Otherwise humans read the rewards using image also but that is the only bit they did not do. They used the original sequence of videos and the reward signal and said, let us train this network. And of course, because the original input is an image or a sequence of images, you have to use a convolution neural network.

**(Refer Slide Time: 22:37)**



## End-to-End Deep Q Learning for Atari

- Input: last 4 frames
- Output: Q(s,a) for 18 joystick actions
- Architecture: CNN (same config for all games)

Convolution
Convolution
Fully connected
Fully connected
NPTEL

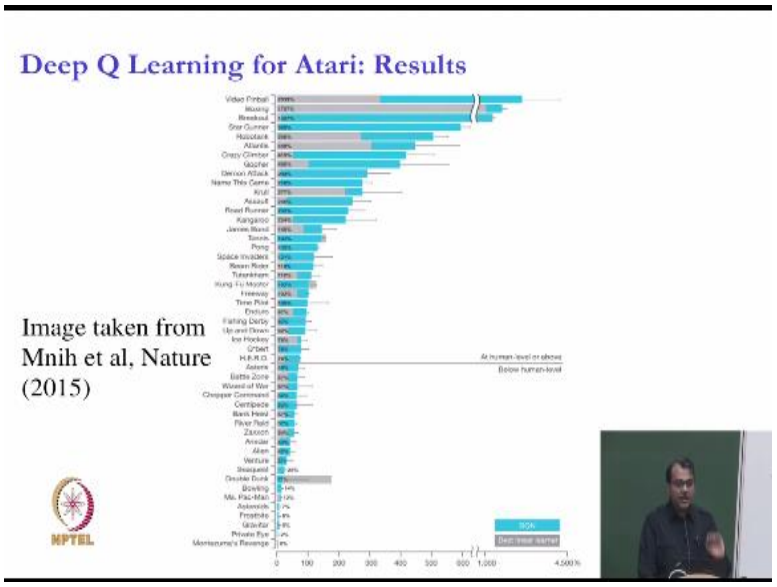
Image taken from Mnih et al, Nature (2015)

And they also did a little bit of thing like for example, you have to make sure somebody is moving in the right direction, right or left direction. So just the image will not tell you that. So they use the last four images basically. The last four images will tell them what is the sequence, how things are moving, and that is it. And then everything else they did the model to figure out completely automatically.

So it was end-to-end learning of Q s, a. The word end-to-end comes from I am giving raw input and nothing else. I am not giving additional features. I am not curating the model myself. It is completely end-to-end just the data driven. And output would be Q s, a with for 18 joystick button configurations. And I do not know all the 18 but it was up, down left, right, you know some button with something whatever, right.

So 18 Q s, a values they will output. And as I said, input state is a stack of raw pixels from the last 4 frames. And the network architecture is nothing but a CNN.

**(Refer Slide Time: 23:40)**



## Deep Q Learning for Atari: Results

Image taken from Mnih et al, Nature (2015)

And this is what they found. Given enough resources after enough training, they were able to show that for most games, a deep network was able to surpass human performance, human performance. Now this human performance may not be the human champion performance, average human performance, average performance of a human who plays this game for a reasonable amount of time and get somewhere.

But whatever and sometimes, the improvement over human performance is 2,500%. This is the line that separates machine better than humans versus human better than machines. These are all the Atari games on the x axis. And of course, there was one game, which was the hardest for them. And that was do you know do you guys know? It is called the Montezuma's revenge.

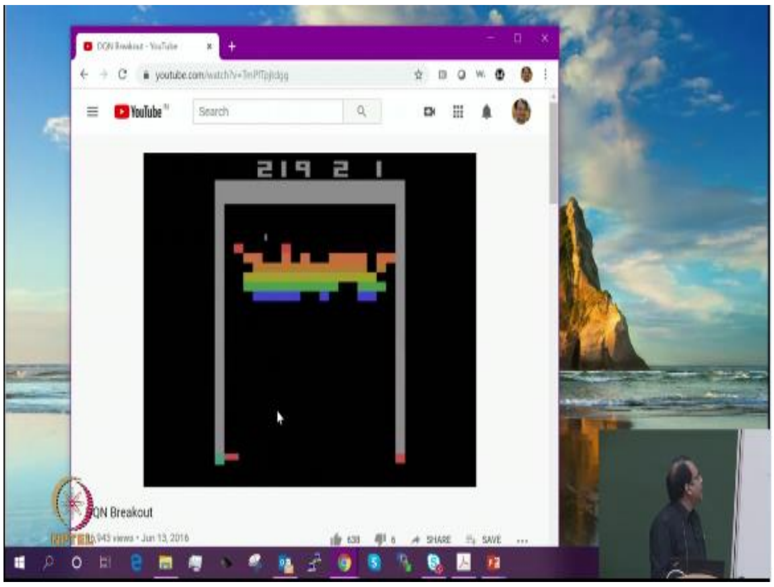
So later, people started saying, okay why is Montezuma's revenge so hard? And they figured out that it required lot of long-term planning and there are no intermediate reward signals. Like if you think about a car racing thing, then at every moment you are getting some reward. As you move forward, your reward is increasing. But as you crash, your reward goes down or whatever it is.

But in Montezuma's revenge, you had to solve a set of complicated steps and puzzle to reach the goal, only then you get the reward. So when the reward signal was so sparse, and problem so requiring so high level reasoning, that is when it could not do this. So even today, there are a lot of people who work on reasoning problems, because deep networks are not the best for reasoning, but they are amazing for perception.

Perception means just looking at the image and understanding what is going on, looking at the speech signal and understanding what is being said, looking at whatever, right. So just understanding, basic shallow level understanding of what is happening in an image, or audio sequence or a text, that they are very good at shallow patterns.

But they are not very good at taking those patterns and piecing them in very complicated ways to do these reasoning for a long-term task, okay. That is what a lot of people have been, in fact thinking about how to make deep networks do it, how do deep networks combine with other reasoning methods together, etc. Let me show you one quick demo.

**(Refer Slide Time: 26:07)**

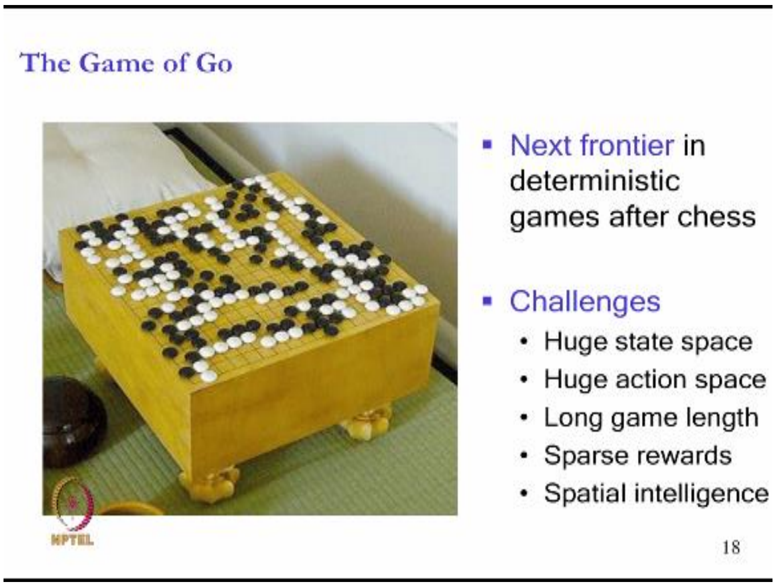


So this is a playing breakout. And I am sure this is after 100 iterations notice that it is missing the ball completely. So it has not been able to learn this very well. After more training episodes, you know it starts to do slightly better. It is sort of chasing the ball much better. But there is a trick in this game. What is the trick? Do you know the trick of the game?

You create a tunnel on the left or on the right and let the ball go up and then let the ball stay. So after one step if you do this right, the ball will remain there and then it will you know give you lot of scores. So after many training episodes by chance at some point it hits on this by chance, because this is exploration, and it will hit on this by exploration. And once it hits it, it realizes, wow, this is a great way to do this.

Let me so now you see how it has learnt how to play. It is constantly, it has learnt exactly how to send the ball on the oblique at an angle, so that it gives you lot of rewards with one particular action. And these kinds of videos are not just for one game, but for all games at the same time. So it actually blew us away. I am not kidding you. I mean, I would say that this purchase, howsoever hundreds of millions of dollars they spent for these 20 researchers is very, very well justified.

**(Refer Slide Time: 27:49)**



The Game of Go

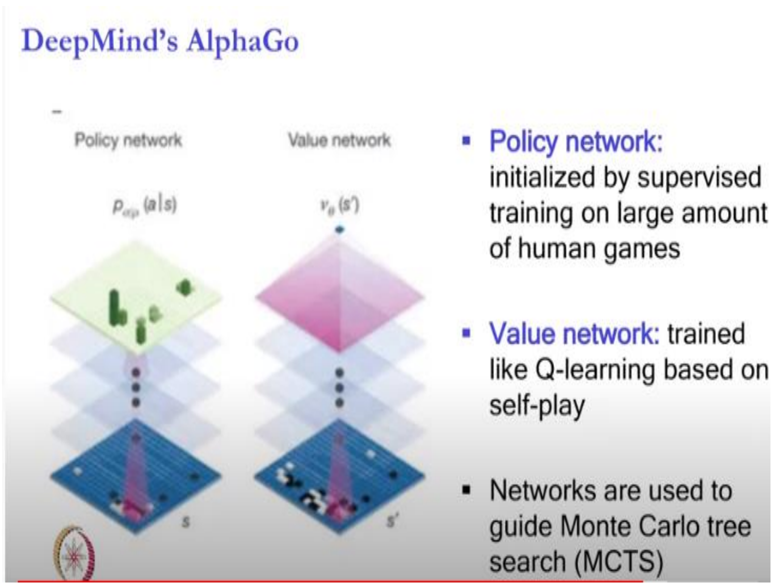
- Next frontier in deterministic games after chess
- Challenges
  - Huge state space
  - Huge action space
  - Long game length
  - Sparse rewards
  - Spatial intelligence

18

Because you know and then they did not stop. They went on from this MDP kind of an environment to deterministic MDP kind of environment to Go which is a adversarial game and I think you all know what happened. It is a deterministic environment. It is a known environment. But so therefore Go is equal to just you known  search actually Minimax search by its huge state space, huge action space, long sequences, fast rewards.

And moreover, there are some lot of vision things that if you create a wall of a certain kind, then you do better, and so on so forth.

**(Refer Slide Time: 28:18)**



## DeepMind's AlphaGo

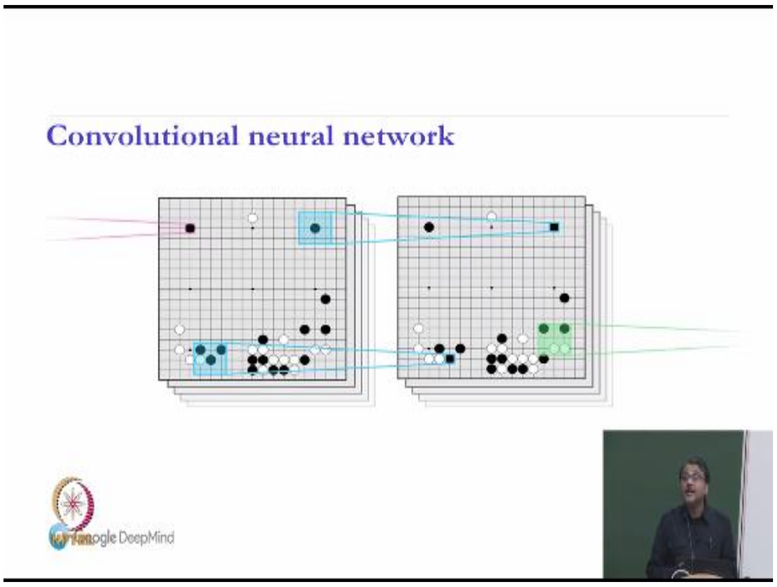
- Policy network: initialized by supervised training on large amount of human games
- Value network: trained like Q-learning based on self-play
- Networks are used to guide Monte Carlo tree search (MCTS)

And there they came up with the algorithm called AlphaGo, which was the next logical step from deep 2 networks, it had deep networks for value function like Q value or V value. And it also had what is called the Policy Network, which computed the policy and they were all parameterised by some weights w. And Policy Network was trained by supervised training on large amount of human games.

So they had a huge record of human games. And then they use that to train a policy network. And then they did self-play to train the value network. And then networks were not just deep learning, but also had some search in it. And that is not equal search, like in depth first search or breadth first search you are going in all directions.

It was unequal search. Some moves were highly evaluated till the very bottom and some moves were not evaluated because they did not look good.

**(Refer Slide Time: 29:10)**



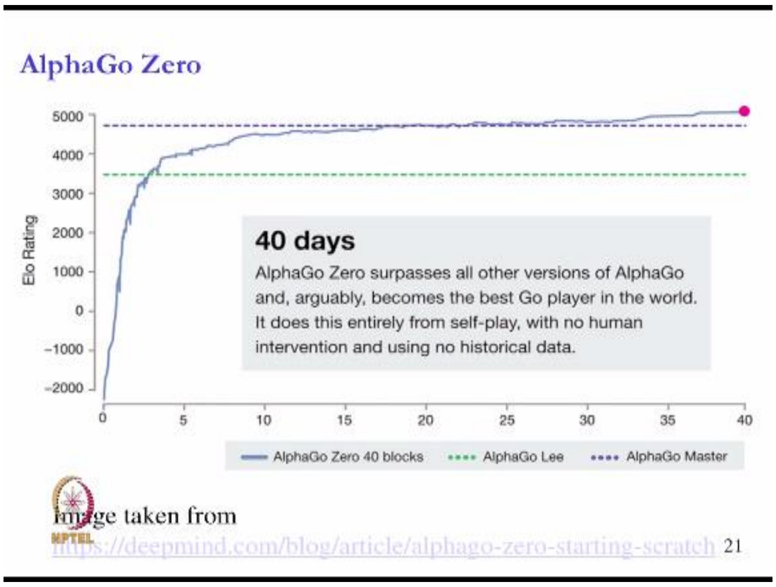
## Convolutional neural network

Of course, it was a the visual part of it was handled by a convolutional neural network. And of course, we know what happened. Lee Sedol was defeated by AlphaGo in this very seminal game in 2016, I believe, right. And he was defeated 4, 1. And in fact, there was a amazing game where everybody thought that AlphaGo has made a mistake. And Lee Sedol had thought that AlphaGo had made a mistake.

It was not a move that anybody had ever seen in the past. And Lee Sedol played very well. I am going to win, I am going to win. And then suddenly, the value function started to show that oh, he is losing because it was an unusual move. It was a great move, actually, right. And later down the line AlphaGo also defeated the world champ. So Lee Sedol is like Roger Federer but not like Novak Djokovic.

So the world champion was defeated. I think he was a Chinese person I forgot his name, the next year, I think in 2017. That did not make that much press because by then, it was clear that this can happen.

**(Refer Slide Time: 30:20)**

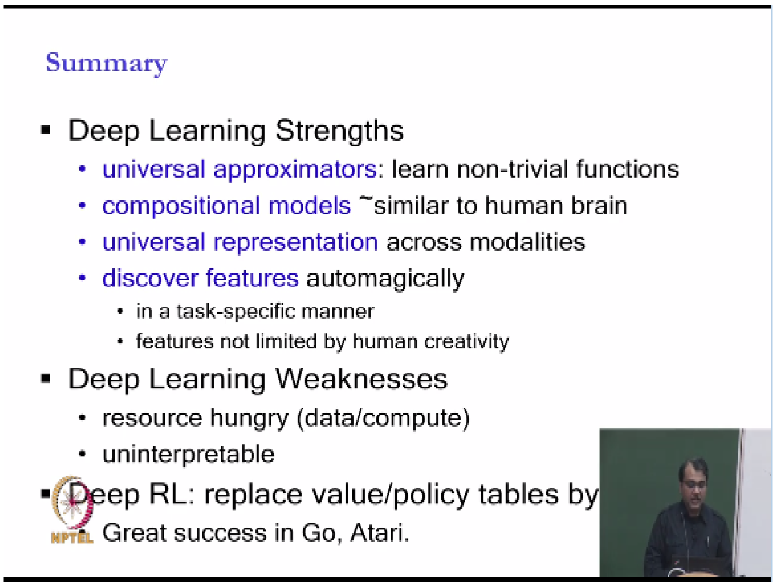


AlphaGo Zero

Image taken from https://deepmind.com/blog/article/alphago-zero-starting-scratch 21

They also later released AlphaGo Zero. So one of the criticism of this work was that it was initialized by supervised learning on a huge number of human games. So they released a patch called AlphaGo Zero an algorithm, which were not a patch, but an algorithm which would train without any prior knowledge.

**(Refer Slide Time: 30:38)**



Summary

- Deep Learning Strengths
  - universal approximators: learn non-trivial functions
  - compositional models ~similar to human brain
  - universal representation across modalities
  - discover features automagically
    - in a task-specific manner
    - features not limited by human creativity
- Deep Learning Weaknesses
  - resource hungry (data/compute)
  - uninterpretable
- Deep RL: replace value/policy tables by
  - Great success in Go, Atari.

Okay, so let me summarize. So we have now completed the full story, at least at the first level of first cut. We have talked about deep learning where the strengths are that it is a universal approximator. It can learn non-trivial functions, nonlinear functions, very highly nonlinear function. They also had layers one after the other. And that model is similar to human brain because we also think compositionally, and our brain also works somewhat in many layers.

It is also being considered as a universal representation across modalities. Because now audio gets done by recognition gets done by deep networks. Natural Language Processing gets done by deep networks. Computer vision gets done by deep networks. And now people are even starting to come up with logical inference and shortest path algorithms which can be done in deep networks. But they have not been super successful.

But you never know that 5 years or 10 years down the line everything is done by the deep networks and nothing is done by symbolic. There is also a set of people who believe that, that is not a good idea and they said look, symbolic algorithms are good at something, they are good at reasoning, they are good at manipulating variables in a human interpretable way.

And deep networks are really good at you know final accuracy but not interpretability and not long-term reasoning. And so how do we combine these two seamlessly. So there is a lot of work that is going on in that space also. One of the strengths in my opinion of deep networks is that it discovers features automatically in a task specific manner. And therefore features are not limited by human creativity.

So even if you think about this particular deep Q learning earlier, you would come up with some features you will say okay, a king with a checkmate, sorry with a check has some, is a feature. A king, which can get a checkmate in one step is a feature. If I have a queen, it is a feature. If they have a queen, it is a feature, etc. And those features come up from human pockets, they come from human insight.

And they can be great and they may not be great depending upon how good the human designer is. So therefore, AI gets reduced to human intelligence in some ways. Whereas now the model itself comes up with features and those features are trained in ways that we cannot understand, but they are these vectors of numbers, but they are trained so that you do really well on the task, okay.

At the same time, deep learning has weaknesses. It is highly resource hungry, it needs a huge amount of data. Typically, it also needs huge amount of compute power. So people find it really hard. I have been told that some of these large organizations sometimes burn a million dollars per month on just computational power, just. That is too much. So academicians are finding it harder and harder to catch up with some of the research that is coming out of you know larger organizations.

And also these are uninterpretable. So we do not know why the model is doing what it is doing. We will talk more about that in the ethics topic. And deep RL is, the basic idea is replace the value in policy tables by deep networks and have had great success in Go, Atari and many other areas. Okay, so that completes that chapter.