





Subtitles To Transcripts

Subtitles To Transcripts Blog

 **Menu**

CS 6601 Artificial Intelligence

 Aarsh Talati  Uncategorized  September 12, 2018November 29, 2018  389 Minutes

1. GAME PLAYING

1 – Course Introduction – lang_en_vs51.srt (1. Game Playing)

.....

Hi, my name is Thad Starner. At Georgia Tech, I teach the AI course, the mobile ubiquitous computing course, and when I'm lucky, the computer science rapid prototyping hardware class. In this course, we're taking parts of Sebastian Thrun's and Peter Norwick's AI course on Udacity and mixing it with the best parts of my teaching at Georgia Tech. When Sebastian and Peter first presented the idea of this online courses, one hope was that we could mix the best parts of the best courses of top lecturers and make a class which is better than what anyone of us could do. Peter will be covering search and logical planning. Sebastian will be teaching the section on probability and base nets. And I'll be covering the rest of the content. Let's get started.

2 – Overview – lang_en_vs52.srt (1. Game Playing)

.....

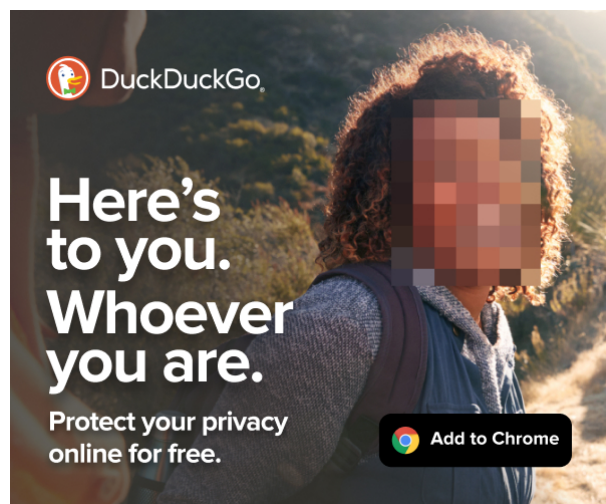
Your goal in this lesson is to design an AI that is smarter than yourself. At least in playing a real game. Now granted, a single game is a very limited context. But making a program that can play a game better than you can is an exciting introduction to AI. It's the way I got hooked. I'm hoping to share my enthusiasm. Along the way, you will learn how to use adversarial search to find optimal moves. In particular, we will talk about the minimax algorithm and alpha-beta pruning. We'll also talk about how to create evaluation functions for games. These ideas are enough to create surprisingly powerful game AIs. We will apply our knowledge to Isolation, a complex game with surprisingly simple rules. We will then examine multi-player and probabilistic games. As your assignment for this lesson, you will need to create a game player for Isolation. Note how long it takes you and how difficult you find it. That should give you a sense of the amount of time and effort that this course will require.

3 – Isolation – lang_en_vs52.srt (1. Game Playing)

.....

Let's dive in and start with something fun. Let's teach you the rules of isolation, the example game we'll be using. Shelly and I will play a game of isolation on a five by five board. I'll be O and move first. O gets to place his piece anywhere on the game board. I'll choose here. Shelly will be X. And she can place her piece anywhere remaining. Shelly, where do you want to go? I'll go here. From now on, our players move like queens in chess. For example, for my next move I can go any square that's horizontal or vertical or diagonal from me, except I can't move through Shelly's piece here. I only have this diagonal move. Okay, I get how players move. But do they attack each other like in chess? What's the objective of the game? The objective is to be the last player to move. The players cannot go outside the boundaries of the game board. Nor through a position that is currently or was previously occupied. The first player to get isolated, and by that I mean, unable to move on their turn, loses. For my next move I think I will move here. Wait a second. Why aren't the squares between the start and the end position filled in? You mean these squares here? You're thinking about the game, Snake. In Isolation, it's just where you land that becomes unobtainable for future moves. Okay, in that case I'll move here. Great so now we have a game going. So let's see here. I want to go here and limit your moves for the future. Okay, so I'll go here. Cool, I think that what I'll do is try to force an early end to the game. I'm going to do this and start dividing up the board. Okay, so there's more spots filled up on the right side of the board. So I think I'll go towards the left so I have more space to move in. Okay, that was a wise move. Let me actually go here and see if I can trap you in. Okay, so I'm going to keep going left and see if I can escape, maybe get back on to the right side. Okay, I want to prevent you from getting out, so I think I can do that. I think I can actually win at this point, so I'm going to go here and stop your diagonal move out. I look like I'm stuck [LAUGH] I guess I have to try to get out. We're getting to end game pretty quickly here because of that immediate move where I could partition the board. So I'll go up here and now you're stuck and now since you have no move, I win. Yehey, okay, so what we've shown here is basically, the rules of isolation and the game board. We've also shown some tricks that show about the difficulty of the game and how you can early on create a partition that the device the boards up on the one side of the other and hang them up then fight for their correct side. Strategy for isolation is really quite complex. You'll find out pretty quickly when you actually try it on your homework.

Advertisements



REPORT THIS AD

4 – Building a Game Tree – lang_en_vs52.srt (1. Game Playing)

Now let's use a simple two by three version of isolation to illustrate how to make a game tree that shows all the moves possible during a game. We will use this game tree to select moves which give the best guarantee of winning. Consider any field spot on the game board as unavailable. We will start with one field spot. We'll call the

beginning game level zero in our game tree. Assume that O is the computer player. O is the first to move, so the computer has five choices in the beginning. We draw out all the possible boards for level one of the game tree. X goes next and can put its piece anywhere on the game board that has not been taken by O. So no matter which position O occupied for its first move, X has four choices. We can quickly draw the nodes for level two of our tree. From then on, the number of choices for each player becomes smaller as the pieces are limited to moves like a queen in chess. O goes next. To the left most leaf, O can move straight down. [BLANK_AUDIO] Diagonally down to the right. But it can't move through X.

5 – Which of These Are Valid Moves? – lang_en_vs52.srt (1. Game Playing)

.....

Shelly, why don't we have a quick quiz on isolation? Sure. So for each of these board states, assuming it's O's turn, select the valid positions to which O can move

6 – Which of These Are Valid Moves? Solution – lang_en_vs52.srt (1. Game Playing)

.....

Okay, so remember that the pieces can move like a queen in chess, but can not pass through positions that have been previously played. Also, when a piece is moved it doesn't leave a trail like the game snake, but just the last position is marked as impassable. So here are those resulting child nodes. [BLANK_AUDIO]


7 – Building a Game Tree (Contd.) – lang_en_vs51.srt (1. Game Playing)


.....

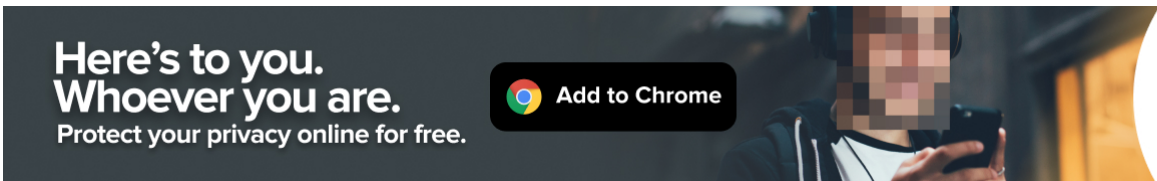
Let's keep expanding the nodes in order until we have fully expanded level three of our tree. As we see some nodes have two children. Others have three. Even so we see that the exponential growth of the children makes it an increasingly difficult to display the full tree on the screen at once. In level four, it is x's turn. Again, we expand the tree completely. Scrolling through the tree shows how increasingly dense it is getting. Level five gets a bit more interesting. In the left-most branch, o has one final move. However, in the next branch x has blocked o in. O has no move and the game ends. To indicate that our computer player has lost in this situation we will mark this node with a -1. Expanding level five completely we see that one branch looks particularly dire for a computer player. As it has many losing situations. Even worse, the opponent has control of the board at the level above. We have to find a way to warn our computer player from making a move that leads to this branch. By level six there are no moves left anywhere on the board. All games have ended. The ones where o wins, we will mark with a plus one. The ones where o lost on level five, we have already marked with a -1. Now that we have seen all the possible futures of the game. We will back propagate this knowledge to our computer player so it can make a wise first move.

Advertisements

**Here's to you.
Whoever you are.**
Protect your privacy online for free.



 DuckDuckGo



REPORT THIS AD

8 – How Do We Tell the Computer Not to Lose? – lang_en_vs52.srt (1. Game Playing)

.....

In most situations in this game tree our computer player O wins. But in a few situations, the game ends with our opponent X winning at level five. If our computer player makes a poor choice for its first move, X can win no matter what O does. Clearly we never want our computer player to make this bad first move. To avoid that situation we can keep a table of best first moves for this board configuration. This table is called an opening book, similar tables can be kept for end games. But how do we make discovering these bad moves automatic? Better yet how do we discover the best moves for a computer player? The answer is the mini max algorithm, a version of which we will see next.

9 – MIN and MAX Levels – lang_en_vs50.srt (1. Game Playing)

.....

With terminal node in our game tree. With a +1 if our computer player wins and a -1 if it loses. We've used squares to indicate the terminal nodes and put their scores inside them. We are going to assume that our opponent x is really smart and will always play to win. In other words, we'll assume that x will always minimize o's scores. While our computer player will always try to maximize its scores. To show that idea, we are going to mark the game tree with a triangle pointing up to indicate turns when our computer player is trying to maximize the score. The triangle pointing down when the component is trying to minimize the score. Since our computer player is going first, we start with the maximization level. In general, since we are only searching from our computer player's perspective, they can only affect the game when it is its turn, the top of a minimax tree is always going to be a max level.

10 – Propagating Values Up the Tree – lang_en_vs52.srt (1. Game Playing)

.....

In order to propagate wins and losses up the game tree, to the point where a computer player can make a decision, we need to start at the bottom of the tree. Let's start with the left side of the tree. In the left-most branch, O wins. In the next branch over, O never gets to use the last square on the game board and loses. Since our opponent x plays to win and minimizes O's score, it will never choose the left most branch. Instead it will choose this branch which is guaranteed to give the computer player a score of -1. However, our computer player can avoid this situation by making a wiser choice one level up. Both branches of this tree lead to victory, hence scores of +1. We can keep propagating these scores up the tree, alternating min in max levels until we arrive at the top. This process of computing values for each node bottom to top is known as the mini max algorithm. For each max node, pick the maximum value among its child nodes. If there's at least one plus one child, the computer can always pick that to win. Otherwise, it could never win from that point on, assuming a perfect opponent. Of course, through represent the opponent, at each min node, we pick the minimum value.

Advertisements



REPORT THIS AD

11 – Computing MIN MAX Values – lang_en_vs52.srt (1. Game Playing)

.....

Shelly, I think I've talked for too long. It seems time for a quiz. I agree. So Thad filled in the left side of this tree. Try to fill the rest of the values on the right side.

12 – Computing MIN MAX Values Solution – lang_en_vs52.srt (1. Game Playing)

.....

Here are the rest of the values for the tree. [BLANK_AUDIO]

13 – Choosing the Best Branch – lang_en_vs52.srt (1. Game Playing)

.....

Thanks, Shelly. Now we can see that our computer player loses in only two branches, assuming otherwise perfect play. All our computer player has to do is choose any of the other branches and it will win.

14 – Aside: Reading the Book – lang_en_vs1.srt (1. Game Playing)

.....

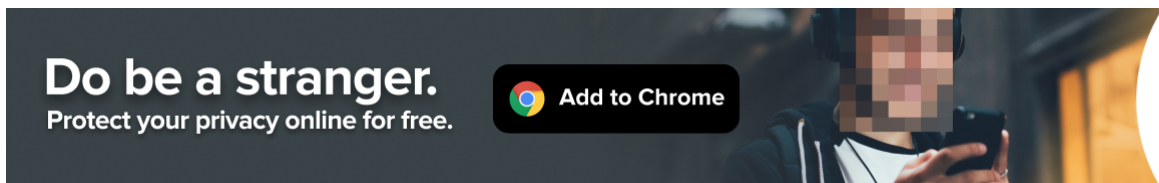
Y'all can think of Shelley and me as your guides on a whirlwind tour of AI. These lectures are designed to show you what we feel are the most important stops on the tour, and to give you some context and intuition. We also hope to peak your curiosity and get you excited about the material. However, real understanding will require reading the book and the papers we provide. As we go we'll point out the relevant readings.

15 – Max Number of Nodes Visited – lang_en_vs52.srt (1. Game Playing)


.....


So far we've worked with an isolation board with only five possible moves. It's searching to end game caused huge gain trees that were hard to show on the screen. How bad will it get with an interesting game board? Well let's pull up the five by five board Shelly and I were playing on. How many different leaf nodes can we expect? Well it's easy to put an upper limit on the number of nodes. How so? On the first move, O can put its piece in any of 25 places. Then on the next move, x can put its piece in any of 24 places. After that, there are 23 empty places followed by 22 and so on. Now all of these spots are always possible moves, so we know that there have to be less nodes than $25 \times 24 \times 23$, and so on. So less than 25 factorial nodes. 25 factorial, how big is that? About 10 to the 25th power. Okay, if we assume a moderate multi-core gigahertz processor which can do 10 to the 9th operations per second, it will take 10 to the 16th seconds to search the entire game. Okay, each hour is 36 hundred seconds and each day is 24 hours and each year has 365 days. That means over 300 million years. Actually 317,097,920 years to be precise. I don't think we have that long to wait. I just gave you a quick upper bound estimate of the number of nodes. It isn't really that bad. You're right. Let's see if we can do a better estimate. Okay, there's not much we can do about the first moves. There really are 25×24 potential first moves. But after that, each piece moves like a queen and there are less moves. What is the maximum number of moves possible in the third move of the game? That sounds like a great quiz question.

Advertisements



Do be a stranger.
Protect your privacy online for free.

 Add to Chrome

 DuckDuckGo.

[REPORT THIS AD](#)

16 – Max Moves – lang_en_vs52.srt (1. Game Playing)

.....

Select the position on the board with the maximum number of options to move next once the piece is limited to moving like a queen in chess. In other words, the third move in the game. Assume that the first move would not block you.

17 – Max Moves Solution – lang_en_vs52.srt (1. Game Playing)

.....

Assuming that the opponent is not in the way, the center position has a maximum potential moves with 16. Two vertically up. Two diagonally to the right. Two horizontally to the right. Two diagonally down to the right. Two down vertically. Two diagonally down to the left. Two to the left horizontally. And two diagonally up to the left.

18 – The Branching Factor – lang_en_vs51.srt (1. Game Playing)

.....

While the center position has 16 spaces in which it can move, most other positions have 12 or less. As the game plays, we will have less and less spaces our player can move. That means we can make a better estimate on how many nodes we have right? Yep, okay, we know we cannot have more than 25 moves in a game. That is the maximum depth our tree is going to be. And we already know how many moves can be done in the beginning,

which leaves 23 moves left. We know that each move after the first two are generally going to have 12 or fewer moves available. We'll call this the branching factor. In worse case, we can expect 25 times 24 times 12 to the 23rd power nodes in our game tree. Shelly, how many is that? More than 10 to the 27th power. Ouch, okay that really didn't help. Perhaps we can improve our estimate more. Well, we know at the end of the game there have to be progressively fewer moves available. There are zero moves in the end, one move before that, a maximum of two before that, three before that, and so on. You're right. So for the last few moves, assuming a branching factor of 12 is way too much. That would be around 10 to the 13th nodes. But the maximum it could be is 12 factorial, or about 5 times 10 to the 8th. I guess we could redo our estimate as 25 times 24 times 13 12s in a row, times 5 times 10 to the 8th. That's approximately equal to 3 times 10 to the 23rd. That is better than 10 to the 25th, or the 10 to the 27th, but still seems like a gross overestimate as most branches will have much less than the maximum branching factor. Let's see if the students have a better answer than we do.

Advertisements

REPORT THIS AD

19 – Number of Nodes in a Game Tree – lang_en_vs52.srt (1. Game Playing)

.....

How many nodes do you think the Minimax algorithm will need to visit in the game tree? Here's some choices where b is the average branching factor, and d is the average depth. So we have bd , d to the power of b , d squared, and b to the power of d . Pick what you think is the best answer.

20 – Number of Nodes in a Game Tree Solution – lang_en_vs52.srt (1. Game Playing)

.....

We can approximate the number of nodes MINIMAX will need to visit using the formula b to the power of d .

21 – The Branching Factor (Contd.) – lang_en_vs52.srt (1. Game Playing)

.....



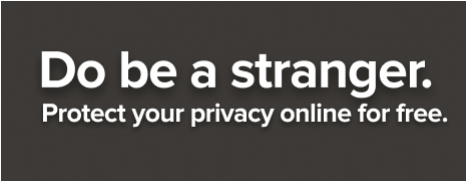
What if we do average branching factor instead? That sounds like a good idea but, how do we do that? Just try it out. Sure, why not? I always say do the simple thing first and only add intelligence when necessary. Let's write a quick computer game player that moves randomly and just test see what our branching factor is on average.
[BLANK_AUDIO]

22 – Max Number of Nodes – lang_en_vs50.srt (1. Game Playing)

.....

Okay, after playing with five by five isolation for a bit. The average branching factor seems to be about 8. So we now get an estimate of 8 to the 25th nodes. That's about 10 to the 22nd nodes. Great, that means we only have to wait for around 1.2 million years to get our answer. Yeah, you're not going to be alive by then. Them dang kids, no good Thanks. No problem. Okay. That means we need to be more clever about how we make a computer player for isolation. The exponential growth of the game tree means we can't brute force the problem and search for the end-game easily. In general, more interesting games will not be searchable til the end. Either the branching factor is too large, or the depth, or both. When the number of nodes, which we can estimate by branching factor to the depth power. Starts becoming comparable to the number of seconds remaining in the life of the universe. We know we are in trouble.

Advertisements



REPORT THIS AD

23 – Depth-Limited Search – lang_en_vs52.srt (1. Game Playing)

.....

Waiting for a computer game player to make a turn is no fun. After two seconds of waiting, a human player is likely to start thinking of other things. We really need a way to choose a move quickly. Well, we can't search to end game but how can we search? Let's assume again that we can search ten to the ninth nodes per second. So in two seconds we can actually search two times ten to the ninth nodes. So we need to solve the equation eight to the X is less than two times ten to the ninth. We can do this by taking log base eight on both sides. That log base eight is pretty annoying. Yup, time to bring out some old math skills. We know that log A of X is equal to the log B of X over the log B of A. So we can use that formula to solve our problem. We'll use log base ten here for convenience. So we end up with X is less than log base ten of two times ten to the ninth over log base ten of eight. And we end up with X being less than 10.3. So we should be okay if we only search ten levels deep? Provided that our estimate of a branching factor of eight on average is good. To be on the safe side let's only go to depth nine. This idea of only going as far as we think we can to safely meet our deadline is called a depth limited search.

24 – Evaluation Function Intro – lang_en_vs52.srt (1. Game Playing)

.....

Okay, great, but what do we do when we get to level nine in our mini-max tree? That's when things really get interesting. We want to evaluate the goodness of a node at level nine based on how much we expect it to lead to a win for our computer player. Can we create an evaluation function that takes in each game board generated level nine of our mini-max game tree, return a number that we can use to compare that node for all the other nodes at that level? Well, we know the only way to win is have our computer player have moves left at the end of the game. Maybe our computer player shouldn't maximize the number of moves it has? That sounds right. We want an evaluation function that returns a higher number depending how good the board is for our computer player. With an evaluation function like simply counting the number of moves our computer player has available at a given node,

the player will select branches in the mini-max tree that lead to spaces where a player has the most options. It seems like a really good idea. Let's call that evaluation function, number my moves for convenience. Why don't we test it out on our simple five-move game board and see how well it works? Let's do it.

25 – Testing the Evaluation Function – lang_en_vs52.srt (1. Game Playing)

.....

For this mini max sub-tree, fill in the values. Assuming an evaluation function of number of my moves or the number of moves possible for player O. Fill in the numbers starting at the bottom and then propagate them up to the top of the tree.

26 – Testing the Evaluation Function Solution – lang_en_vs52.srt (1. Game Playing)

.....

The bottom level is easy to compute. It's just the number of moves available to player O. Then the second level from the bottom is a max level. So we take the max at each branch here. Then the top level of this subtree is a min level. And we end up with a 2.

27 – Testing the Evaluation Function Part 2 – lang_en_vs50.srt (1. Game Playing)

.....

Now try filling out the values for the entire minimax tree. Again, using the number of my moves evaluation function. Propagate the numbers to the top of the minimax tree. Since we can't fit the entire level three tree here, we'll use this abbreviated version instead. A link to the full tree has been provided in the instructor notes. Open it up and compute all the values, starting at the bottom and propagate them up to the top. Then report back the values for a level one and the root node here.

28 – Testing the Evaluation Function Part 2 Solution – lang_en_vs52.srt (1. Game Playing)

.....

So, here are the values that I got. There's not much variation here, but there's a number of good choices. You can see three branches that have a value of 2. Our computer player will take the one on the left automatically, so we'll probably take this move to start with

29 – Testing Evaluation Functions – lang_en_vs51.srt (1. Game Playing)

.....

Advertisements



Need a website quickly – and on a budget?
Let us build it for you

Let's get started

REPORT THIS AD

Remember when we had our simple five-move isolation board? And we searched it to end-game? We found that player one would always win unless it moved to one of the center positions first. I see where you're going. The question is whether our evaluation function of `#my_moves` predicts which branches will lead to a win and which ones will lead to failure? Exactly. I always like to test my assumptions because they are often wrong. Okay, so here's the mini-max tree where I have loaded the the depth to three and applied the evaluation function. I've also propagated the values up the mini-max tree to the top. The result looks promising. The two center positions have scores of 1, while the rest of the branches, which we know when, have higher scores. Let's try another example. What happens if we try the same thing but assume we could only go two levels deep on our search tree?

30 – Testing the Evaluation Function Part 3 – lang_en_vs52.srt (1. Game Playing)

.....

Now, let's assume we can only search to level two of this minimax tree. Fill in the values of the bottom of the tree, again assuming the evaluation function of number of my moves. Propagate the numbers to the top of the minimax tree. As in the previous quiz, use the complete level two tree linked in the instruction notes, then fill out the top two levels here.

31 – Testing the Evaluation Function Part 3 Solution – lang_en_vs51.srt (1. Game Playing)

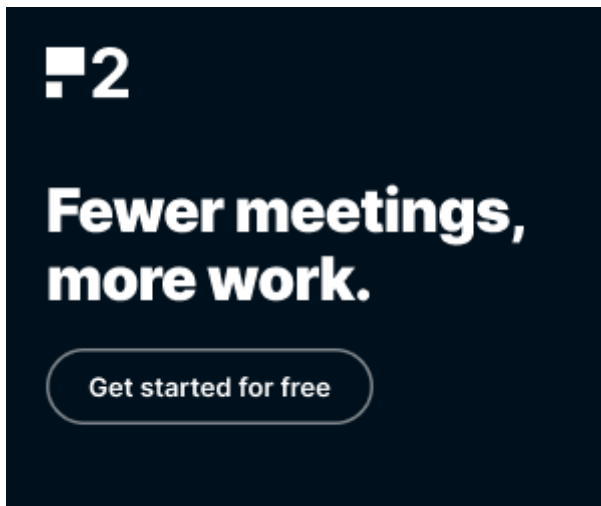
.....

Here are the values we get after searching to level two of our game tree. We can see that the values are significantly different from when we searched to level three. Previously, we would have chosen one of these three nodes, because they received the highest value in our evaluation function. But after only searching to level two, we see that it's actually the other nodes that get a higher value in our evaluation function. So, in this case, our computer player would choose one of these two moves. And if it takes the move on the left first, it would choose this move, which is a different choice than he would have made after surging to level three.

32 – Quiescent Search – lang_en_vs52.srt (1. Game Playing)

.....

Advertisements



[REPORT THIS AD](#)

With the depth limit of two we get a very different result than with the limit of three. Yep, at level two, the losing center moves. Both have scores of three, while the winning moves have smaller scores. Does that mean our evaluation function is bad? Not necessarily. It may mean that we're not searching deep enough to get good answers. How do we tell? One way to check is to see which branches the minimax algorithm recommends at each level of the search. If the results vary widely between certain levels, it may be because critical a decision is being made at those levels. Let's try it out on the example isolation game. We will continue to use the number of my moves evaluation function. Which can have a maximum score of five and a minimum score of zero. If we find a branch where our computer player loses, we'll give it a score of -1. If we find a branch where our computer player wins, we'll give it a score of 100. With this example isolation game at level one, the number of my moves evaluation function would recommend the two center moves and the top left move. Each of these have scores of four. At Level 2, the two center moves are considered the best with scores of 3. At Level 3, the center moves are considered worse. That continues for Level 4, 5 and 6. [BLANK_AUDIO] After level three, we've reach a state of quiescence. The recommended branches are not changing much. We know we are in good shape. [BLANK_AUDIO]

33 – A Problem – lang_en_vs52.srt (1. Game Playing)

.....

But that doesn't help us. Well, why not? Well, the whole point of limiting the depth of our search and using an evaluation function was to avoid the exponential explosion in the number of nodes we have to examine. With quiescence search we have to search the game three for two levels and maybe a whole lot more if the results are changing between levels. Well, we don't have to use quiescence search all the time. In fact, if we play isolation a lot and see that the evaluation function is giving us consistent results, then we might not bother at all. However, we might find that in our testing that certain times in the game creates quiescent search might give us some better results. Often times, that's at the beginning of the game, or at the end of the game. Also we have another trick we can use called iterative deepening, and that might help us too.

34 – Iterative Deepening – lang_en_vs52.srt (1. Game Playing)

.....

What's iterative deepening? Let's go back to our problem of making our computer respond to an opponent in two seconds or less. Before, we calculated the maximum depth we thought we could search in time safely. With iterative deepening, we are going to do something simpler. We are going to search the Level 1 and get an answer for what we think is the best move. We'll keep that answer in case we run out of time. But we'll start the process again and go to Level 2 this time. If we finish searching Level 2 before time runs out, we'll keep its best move and restart the search from going from Level 3. We'll continue doing this process until we run out of time and then return our best answer. So you're saying we should do the same thing we did to determine quiescence? Basically quiescence is a good side effect. But that still doesn't answer my question. Isn't this process inefficient? Surprisingly, iterative deepening doesn't waste that much time. Because of the exponential nature of the problem the amount of time is dominated by the last level searched.

35 – Understanding Exponential Time – lang_en_vs52.srt (1. Game Playing)

.....

Let's look at a simple case of Iterative deepening on a tree with a branching factor of 2. A level of 0, only one node is searched. At level 1, we touch this first node again and then look at the two child nodes. We've had to explore three nodes for level 1 plus one node from level 0. So, our total for iterative deepening nodes, is 4. At level 2, we add four nodes to the bottom of the tree. To add these nodes, we are looking at the 3 above it as well for a total of seven for this level. Adding level 2 to our running tally is 11 nodes for iterative deepening. At level 3, we had eight more nodes, so we examine 15 nodes for this level in total. Our iterative deepening total is now 26. Know that at each level, the sum total of nodes visited and revisited by iterative deepening is actually under two times the number of nodes expanded by that level. A depth-limited search to level 4 would examine 31 nodes, but the iterative deepening total was 57. Depth limited to level 5 would be 63, and iterative deepening would be 120. While researching the tree may seem like a waste in practice, it is a small multiplier compared to the exponential increase from searching each level deeper. With a branching factor of 2, iterative deepening expands less than two times the number of nodes a depth limited search would have done at the same level. When the branching factor is greater than 2, as in most games, then the redundancy caused by iterative deepening is even less. Perhaps a quick quiz would help.

36 – Exponential $b=3$ – lang_en_vs52.srt (1. Game Playing)

.....

Now let's consider a tree with a branching factor of three. How many nodes are visited at levels zero, one, two, three, and four? Tell us the number of tree nodes, and also the number of tree nodes visited by iterative deepening.

37 – Exponential $b=3$ Solution – lang_en_vs52.srt (1. Game Playing)

.....

Here's the answer. At level 0, we have 1 node in the tree and iterative deepening has only visited 1 node. At level 1 we now have 4 nodes in the tree. Iterative deepening will have visited 5, the 4 nodes in the tree now and one node from the previous level. At level 2, we now have 13 nodes in the tree and iterative deepening will have visited 18. 13 from this tree plus the 5 from the previous time. At level three we have 40 nodes in the tree plus 18 makes 58 nodes visited binder of deepening. Finally, at level 4 we have 121 nodes in the tree, iterative deepening we'll be visiting 179. Which is actually less than one and a half times the number of nodes in the tree. Remember that

before when the [INAUDIBLE] factor of 2, we had 2^{d+1-1} nodes in the tree. Now with the project factor of three, we have 3^{d+1-1} nodes in the tree. Let's go ahead and generalize this. For a project factor of k , the formulas k^{d+1-1} for the number of nodes in the tree.

Advertisements



[REPORT THIS AD](#)

38 – Varying the Branching Factor – lang_en_vs52.srt (1. Game Playing)

So basically you're saying that iterative deepening is almost free because of the exponential nature of the problem? Yeah. And I bet that iterative deepening really helps when you have a game like Isolation with the wildly varying branching factor. How so? Well when we calculated that we should only go nine levels deep in a five-by-five isolation tree to return within two seconds. We assumed a branching factor of eight. Yet toward the end game, Isolation players only have a few moves available. We might be able to search much more deeply at end game. And often, end game is precisely the time, that you want to search more deeply, so you can see what will happen. On the other hand at the beginning of the game the branching factor is quite large. Our computer player might not return within two seconds. Correct. Iterative deepening means that our computer player always has an answer ready in case it runs out of time and it can search as far as possible within its time constraints. We limited our time to two seconds per move. Yet in some games, like speed chess, we limit the total time a player can take for the entire game. In those situations, our computer player will want to search deeper in some parts of the game and shallower in other parts of the game. Often we can create a strategy for how deep we want to search. So something like having a book of standard initial moves for the beginning of the game, then search deeper in the middle and use less time towards the end. Relying on the reduction and branching factor and iterative deepening to still allow our computer player to search to end game or as deep as it can given the remaining time. Or we might want to have a conservative amount of time we dedicate per move and use iterative deepening and quiescent search to determine the few moves we wanted to spend extra time. All these are great strategies, but we can still get into trouble due to something called the horizon effect.

39 – Horizon Effect – lang_en_vs51.srt (1. Game Playing)

One remaining challenge in making our computer player are situations where it is obvious to a human player that the game will be decided in the next move, but that the computer player cannot search far enough into the future to figure out the problem. This situation is called the horizon effect. Take a look at this isolation board. It's o's turn, which move should o take? Well, as soon as o moves diagonally down and left one, x will only have six moves as compared to o's seven. Then o will win. But that is 13 moves in the future. How did you figure that out? X will be blocked on the wrong side of a partition and not be able to reach o. Both players will have to fill in the rest of the game board efficiently in order to have a chance of winning. That's a great observation, but our computer player may not see it. Remember, when we started talking about having only two seconds to make a move, we said that our computer player should only look ahead nine moves. It won't see the endgame coming. But with iterative deepening it might be able to go deeper. The branching factor towards the end of the game is much smaller. True, however, imagine that we have all ready reached the next to last depth of our search tree in the time we have. We have to evaluate the goodness of the children of this node. Which move would our my moves evaluation function pick? Well, certainly not the winning move. The evaluation function will return three for the winning move, but I can see a couple moves that would be preferred. Going the whole way to the right gives a five. And going diagonally left and down two spaces gives a six, yet both of these moves will result in a loss. Yep, going to the right will get o on the wrong side of the partition. Going to the left causes o to create another partition with even less moves.

40 – Horizon Effect (Contd.) – lang_en_vs53.srt (1. Game Playing)

So what do we do? Well, perhaps we should include a check in the evaluation function to see if a partition is formed by the next move. And if so, start counting the number of moves left to each player. That sounds like a good idea, but how much is it going to cost us? What do you mean? Our evaluation function goes from a very simple function to compute to something that may involve a complicated check followed by a lot of counting. Simply counting the number of contiguous squares a player can use is similar to searching more levels down in the search tree. We know that is very time consuming. More to the point, we just doubled the amount of time the evaluation function takes. That time becomes multiplied exponentially due to the branching factor as the search goes deeper. Ouch! Are complicated evaluation functions always not worth it? It really depends on the game. Changing the rules of isolation slightly can dramatically change whether it is better to use a simple evaluation function and search deeper. Or use a complicated evaluation function that catches these dangerous situations. I've tried isolation variants like having the players move a knight in chess. Or allowing a wrap around on the borders or even having each player control two pieces. And I'm often surprised at which strategy seems to win. It always pays to think carefully about the evaluation function. Whether it's capturing the desired behavior and how efficiently it can be implemented.

41 – Good Evaluation Functions – lang_en_vs52.srt (1. Game Playing)

Now let's think about some other potential evaluation functions for isolation. Here we have some options. Number of opponent_moves. So similar to the number of my_moves, this is the number of moves your opponent has available. Number of squares_remaining, which is the number of empty squares left on the board. Number of squares_remaining minus number of my_moves. And finally, we have number of my_moves minus number of opponent_moves. So go through and mark all the formulas you think that would make a good evaluation function for isolation.

42 – Good Evaluation Functions Solution – lang_en_vs52.srt (1. Game Playing)

.....

The number of opponent moves would actually do the opposite of what we want. It would label boards as good where our opponent has a large number of moves when we actually want to isolate them and prevent them from moving. Number of squares remaining doesn't reflect the goodness of the board. It's constantly decreasing with each move. Number of squares remaining minus number of my moves would penalize our computer player for boards with more potential moves, which is counter productive. Finally, number of my moves minus number of opponent moves is a good potential evaluation function for isolation. It continues to take into account boards where the current player can make a larger number of moves and also penalizes boards where the opponent can make a large number of moves.

43 – Evaluating Evaluation Functions – lang_en_vs52.srt (1. Game Playing)

.....

Let's talk about a different evaluation function. Let's use number of `#my_moves` minus the number of my `#opponents_moves`. I really like variants of this function for simple isolation games. The point of isolation is to illuminate the opponent's moves. `#my_moves - #opponents_moves` causes the computer player to seek moves with the most options while trying to get in the way of the opponent's moves. We can even weight the components of the formula to try to encourage more aggressive or less aggressive game play. For example, `#my_moves - 2 * #opponents_moves` will cause our computer player to chase after the opponent. That makes the examples you gave for the horizon effect much more interesting. How so? Well, the winning move now has the highest evaluation function result. Here is the winning move. And the evaluation function now returns a 1. For the move immediately to the right, results in a -2. The move to the far right returns a -1. And the far diagonal move returns a 0. Maybe that is the answer, maybe keeping your options close, but your enemies closer is the right strategy in isolation. I'm not so sure. I think the only way to really know is to try lots of variants of evaluation functions and see which ones are the best. You're right, but in addition to minimax and iterative deepening, there's one more trick we have to teach that can really affect the efficiency of game tree search before we spend time doing the evaluation function.

44 – Alpha-Beta Pruning – lang_en_vs52.srt (1. Game Playing)

.....

Alpha-beta is a pruning technique that allows us to ignore whole sections of the game tree, but still get the same answer as with minimax. So you're saying that alpha-beta never changes the answer, but is more efficient than minimax? Precisely, often it is much more efficient. Let's take another look at the minimax tree for level three of the five move Isolation board. We'll assume that our computer player is evaluating the game tree from left to right. We have five subtrees that we'll consider. Looking at the most left one, the first branch has only two nodes with values of 1 and 2 respectively. It is the max level, so we choose the two and propagate it up to the mean level. Since the opponent x will choose a branch that minimizes the value, we know this sub tree will have a value of 2 or less. So that means for any of the remaining branches, as soon as we see a 2 or more, we could ignore the rest of the nodes because they will never be selected. Precisely. And we have 2s in the leftmost node of these three branches. Yep. Which means we can ignore 6 of these 11 nodes. Wow, that's going to save some time. It gets even better. Let's look at the next subtree over. As soon as we get to this two on the left hand branch, we know that this whole

subtree is going to return a 2 or less. But we already know that we have a 2 from the first subtree. So at the highest node, we already know that we will get a 2 or more. That means we can ignore all the remaining branches of the second subtree. Going on to the middle subtree, we again get a 2 in the left most node. We don't have any constraints on the node above yet. So we keep exploring and get another 2. No other valid move is possible, so this max node returns a 2. Now, the node above is a minimizing node, so it must return a 2 or less. Thus, because we know that the top most level already has a branch with a 2 value, we can ignore all the rest of the nodes in the subtree. I see a pattern here. With the fourth subtree, we also get a 2 early on, which means we can ignore most of the nodes in this subtree as well. [BLANK_AUDIO] And the same thing happens in this last subtree. [BLANK_AUDIO] Hold it, this sub-tree is one of the bad moves. Its value is actually 1. Don't we want our computer player to know that? Why? As long as we only care about our computer player selecting the same good move as it would with the minimax app rhythm, we are okay. Here our computer selects the left most branch, which is the same as our minimax arrow. You're assuming that minimax is valuing left to right and takes the left most branch in case of a tie. Yep, we're assuming that our goal was to play the game, not keep a list of all the equally good moves on a given level. That's exciting. Looking at the entire tree, it seems we don't need to look at most of the nodes. We only need to look at 29 of the 78 nodes on the entire tree. That will save a lot of time. In fact, while our minimax algorithm runs in b to the d time, minimax with alpha-beta pruning can run b to the d over 2 time if the nodes are ordered optimally with the best moves being first. Even with random move ordering, alpha-beta pruning reduces the expected run time which is b to the three-quarters d .

Advertisements



[REPORT THIS AD](#)

45 – Minimax Quiz – lang_en_vs52.srt (1. Game Playing)

.....

Fill in the values of this minimax tree. Remember, the root node is the max level, then we alternate min levels and max levels.

46 – Minimax Quiz Solution – lang_en_vs52.srt (1. Game Playing)

.....

Here's the answer. We'll start at the bottom and propagate the bottoms up. The first level from the bottom is a max level, so we'll take the max of each of our leaf nodes. Next, we have a min level, so we'll take the min of each of the values we just found. Finally, the top is the max level.

47 – Alpha-Beta Pruning Quiz 1 – lang_en_vs51.srt (1. Game Playing)

Now that we've filled in the tree, which nodes can be pruned with Alpha-Beta? So go through and click any of the check boxes for nodes or branches that you think would be pruned by the Alpha-Beta algorithm.

48 – Alpha-Beta Pruning Quiz 1 Solution – lang_en_vs52.srt (1. Game Playing)

Here we can print this node right here. If we only knew the 11, this max level here would be 11 or greater. However, even if the value was greater than 11, the next node up would still take the five senses of mean level. That means that we actually don't need to check the four node because it's irrelevant. And thus we can prune it from the tree. We can't prune any other nodes or branches in this situation.

49 – Alpha-Beta Pruning Quiz 2 – lang_en_vs52.srt (1. Game Playing)

Let's consider the same tree as the previous quiz. Is there a way to re-order the leaf nodes of the tree in order to prune as many nodes as possible? You'll find a link to the original version in the instructor notes below this video. Refer to that tree and fill in the values here for your re-ordered tree. Then, go ahead and check the boxes to indicate which nodes or branches you can now prune from the tree. Remember, you can enter an exact value like five or a range like greater than or equal to five

50 – Alpha-Beta Pruning Quiz 2 Solution – lang_en_vs52.srt (1. Game Playing)

Here's the answer. We haven't made any changes to the left side of the tree. So, as before, the same four node will be pruned. On the right side, you can see that we've switched the right two branches. We evaluated this branch and got a 4. So, we know that in the mid-node above, our value will be less than or equal to 4. However, the final node is a max node and we already have a 5 to compare that to. So, we'll always end up choosing this 5 in the max node above. That means that the right most branches are relevant and thus could be pruned.

51 – Thad's Asides – lang_en_vs52.srt (1. Game Playing)

Many problems in artificial intelligence are exponential in time or space, or both. In fact, NP hard problems are so common in AI that researchers joke half seriously that AI is the study of finding clever hacks to exponential problems. Often when a clever hacker is finally found, or when computers finally get fast enough to address that particular problem successfully, the world no longer thinks of the problem as one belonging to artificial intelligence. How many people think about the system that helps consumers choose plane flights as an AI? Or the system that helps determine when to deploy an airbag on an SUV? Yet at one point, these types of problems were considered to be part of AI. That idea leads me to another joke definition of AI. Artificial intelligence consists of all the NP hard problems that have not yet been solved. What matters to me are the problems that the field of AI tackles. On the way here, I was practicing this lecture with my taxi driver, Dilber, when I realized he was using AI while we were talking. To get me to Udacity headquarters, the navigation program on his smartphone had plotted a path that was better than the one I would have told him. And was communicating with him in a way that he was getting the information as he needed it. That is what is exciting about AI. You get a chance to work on everyday problems that can improve people's lives.


52 – Solving 5×5 Isolation – lang_en_vs52.srt (1. Game Playing)

.....

Let me introduce a special guest student from a previous class, Malcolm Haines. Hi, Malcolm. Hi, Thad. When working on the 5×5 isolation game in class, you believed you managed to search the end game. That's right. But when we talked about branching factor earlier, we said it would be impossible to search to endgame in a reasonable amount of time. So how did you do it? First, I implemented the minimax algorithm with alpha beta pruning. As you know, this reduces the size of the search space from b to the d , to b to the d over two, thus reducing it from approximately 8 to the 25th to 8 to the 12th. Next, I realized that some moves are equivalent. For example, let's look at player one's first move. [BLANK_AUDIO] If player one moves to 0, 0 it's the equivalent of moving to 4, 0. You can simply rotate the board 90 degrees and you have the same board state. Therefore, if you know the game tree for our first move of 0, 0, you know the game tree for our first move of 4, 0, which is the same as 4, 4 but also 0, 4. This is useful especially at the beginning of the game when the branching factor is high. For example, while player one has 25 possible moves, in reality, there are only six unique moves, as you can use symmetry about the horizontal and vertical axis. And about the diagonal axis, and of course, there's the center move. We've just shown player's one six unique first moves and their equivalents. A similar analysis is possible for any board state which is defined as a series of ord moves. Let's run through an example. In this example, o moved to 2.2. Now it's x's turn. x is considering possible moves and first evaluates a move to 2.1. x now knows the value of the board state, (2,2), (2,1). Now, let x consider a move to 2,3. x will check to see if the outcome of this move is already known. To do this x checks to see if he knows the value of the board state (2,2), (2,3). He does not. Next, x rotates the board 90 degrees and checks to see if he knows the value of the board state, (2,2), (1,2). He does not. Next, x board takes the board 180 degrees, and checks to see if he knows the value of the board state, (2,2), (2,1). Hah, it does. x returns the value of the board state (2,2), (2,1) and does not need to expand the game tree further. If x hadn't found a solution, he would have checked the board states created by rotating the board 270 degrees and flipping the board along its diagonal axis. Only then would x have needed to expand the game tree. So you found that symmetry really cut down on the number of nodes you had to expand? Only until about level 3 of the search tree. After that I gave up because symmetry was rare and the amount of effort needed to check for symmetry just wasn't worth it. So you're telling me that just those tricks are sufficient to search the endgame? Not quite. While alpha beta pruning in equivalent boards tremendously reduced the number of possible game states, they still weren't enough. Luckily, while searching for a good evaluation function, I discovered that I didn't need to always search to the end of the game tree. It turns out that I know the outcome as soon as there's a partition. Why is that? Well, a partition separates two players completely. Therefore, the player with the longest path wins. For example, In this game, player o moves next. But the outcome of this game is already known, because player eight has eight possible moves

while player two only has six. Okay. So partitions, alpha-beta, and symmetry allows you to finish after you completed looking through all possible moves. Does someone always win if they play optimally? Yes Thad, it turns out that player 2 always wins. Well that's unexpected. But you mentioned you also found a good first move for player one, right? That's right. At first, in fact I thought player one was always going to win. This is because of reflection. Reflection? What do you mean? Reflection is just a 180-degree rotation of the opponent's move. In trying to solve the game, I discovered an interesting phenomenon. If player one's first move is to the center and player two's first move is one that player one can reflect, then player one can always win. All she has to do is reflect every move player two makes. Okay then, how does player two avoid losing? By moving to a location that player one can not reflect. There are eight such possible moves of the 24 available to player two. I leave it as an exercise for the student to determine those eight moves. We're going to have a competition where the students are going to try make the best isolation players they can. Do you have any advice for them? I sure do. If you're player one, move to the center square. Then if possible, reflect player two's moves and you will win. But, it's better to be player two. In this case, create a good book of opening moves and hints. We've discussed the best first move if player one occupies the center square, and it turns out that in most cases if player one does not occupy the center square then player two should occupy the center square. Anyway, once you have a good book of opening moves, use your understanding of equivalent moves and hash tables to load and search your order book efficiently. Remember, you only have a limited amount of time to complete your move. After exhausting your book of opening moves, implement minimax then add iterative deepening, and finally alpha beta pruning. Finally, focus on your evaluation function. Now a good evaluation function is, well, that's up to you to decide. Good luck. Thank you, Malcolm. [BLANK_AUDIO]

Advertisements



The advertisement features a background image of hands typing on a laptop. In the center, there is a circular logo with a 'W'. Below the logo, the text reads: "Need a website quickly – and on a budget? Let us build it for you". On the right side, there is a white button with the text "Let's get started".

REPORT THIS AD

53 – 3-Player Games – lang_en_vs52.srt (1. Game Playing)

What about three player games like 3-player isolation? What's 3-player isolation? It's the same as normal isolation but with three players trying to be the last to move. So do the players form alliances against each other? They can, but there can only be one winner in the end. That could make the evaluation function difficult. Why don't we ignore that for now and just use #my_moves to make things simple? How would Minimax work? Well, for multiplayer games we don't use Minimax any more. Instead we evaluate the gameboard from the perspective of each player and propagate the values up the tree. How does that work? Let's imagine the 3-player isolation game tree where we search down to the level 3 of the tree. On the leftmost branch we evaluate the resulting game board from each of the player's perspectives. For player 1, the valuation function returns a 1. For player 2, it's a 2. And for player 3, the valuation is a 6. I guess we evaluate each of the board nodes at this level and then return triplets for each of them. Yep, and then we propagate the values of the tree. We first choose the max value at each of the level 2 branches from the player 3's perspective. The leftmost node, player 3 has a choice between 6 and 3. So of course, we choose 6. In the next branch from the right on that level, there is a choice between a 2 and a 1, so we choose 2. And the third branch from the left we choose between a 2 and a 1, so we choose the 2. And finally in the rightmost branch, we choose the 5. Okay, and for the next level up I guess we choose the maximum value from player 2's perspective. Yep. So on the left branch we choose the branch with the 2 and, on the right branch we chose the 5.

Precisely, and at the top level we choose the maximum value from player 1's perspective. In this example, both options are equally good. So we choose the one on the left by default. This version of a game tree, which we call MAX N, can work for multiplayer games with any numbers of players. Let's try a quiz on this concept.

54 – 3-Player Games Quiz – lang_en_vs52.srt (1. Game Playing)

.....

Fill in the values of this three-player game tree, which alternates levels between players. The values of each node are in the order, player one score, player two score, and player three score. Each agent tries to maximize their own score, so will propagate up the values based on which agent's move it is.

Advertisements



[REPORT THIS AD](#)

55 – 3-Player Games Quiz Solution – lang_en_vs52.srt (1. Game Playing)

.....

Each agent tries to maximize its own score, so we propagate up the values based on which agent's move it is. We start with agent 3's level. Here agent 3 has a choice between a 1 and a 2, so it'll choose the 2. Then it has a 3 and a 4, so it'll choose the 4. Then it chooses between the 5 and the 6, so it'll take the 6. And then a 7 and a 8, so it'll take the 8. Next is a player 2 level. Player 2 is choosing between a 7 and a 5, so it's going to take the 7. And then a 3 and a 1, so it'll take the 3. Finally, it's player 1's turn. And player 1 will choose the 7 when compared to a 1.

56 – 3-Player Alpha-Beta Pruning – lang_en_vs50.srt (1. Game Playing)

.....

The next question is whether alpha beta can work with multiplayer games. Well according to a paper by Korsh, pruning can work as long as some of the values of the players and valuation functions, has an upper bound and each player's value has a lower bound. Well for our number of my moves the valuation function for isolation, zero is a natural lower bound. But what will make a good upper bound for the sum? If an evaluation function could estimate the number of total moves remaining for each player, it would work. Because the sum of the number of total moves should not exceed the number of empty squares on the isolation board. For example with this board, the limit should be 22. Or around seven per player. That's fine in theory. But the upper bound would change with each level of the game tree. Can't we simplify things for illustration purposes? Okay, we'll say that the sum of player scores can exceed ten. And to make things even easier, we'll say that the sum has to equal exactly ten. And that's going to allowing pruning? Yep, and we'll show an example that allows both immediate pruning and shallow pruning. Does that mean we can't do deep pruning like before? Unfortunately that's correct. While some pruning is possible, deep pruning like what alpha beta allowed before isn't possible. Okay, so show me an example of immediate pruning. Let's look at the left most bottom branch of the search tree. Since we know the maximum value is ten, there's no point in evaluating the next two branches to the right. They can return at most ten and we already have that option.

We can safely ignore them and propagate the values up the tree. Great. And I guess we can put limits on what values will be at the top most node. However in this case it doesn't help. The value of player one could be anywhere from zero to ten. The other ones will be less than ten. But the middle branch would limit things a bit. And you will see an example of shallow pruning for the right branch. At the bottom level of the middle branch, player two will pick the left most option. Now we see player one will have a value greater than or equal to three. And since the sum has to be ten, each of the others are limited to values less than or equal to seven. But in the next branch over, player two will get a seven or better. Which means that player one will get a three or less. Since we already have a three, we can prune this last branch. Its values will not matter, because we're going to choose the option we already have. Now you see it. Let's have a quiz and work another example.

57 – Probabilistic Games – lang_en_vs52.srt (1. Game Playing)

.....

What if I want to make a computer player for a stochastic game, like Backgammon? Well, in Backgammon, your moves are limited each turn based on the roles of two dice. Since you can't know the result of the dice ahead of time, it would seem at first that you can't do a game-tree for it. In actuality, you can. And then use an algorithm called expect to max, to make best-choice decisions on your moves. To show how we handle probabilistic games, I've invented a variant of isolation I call sloppy isolation.

58 – Sloppy Isolation – lang_en_vs52.srt (1. Game Playing)

.....

To show how probabilistic games work, I've invented a version of isolation called Sloppy Isolation. In this game, players may not actually move where they intended. For example, if our player is trying to go to here, it only has an 80% chance of hitting the square intended. There's a 10% chance the player will undershoot its goal. And there's a 10% chance the computer player will overshoot its goal. But suppose the computer player is in a place where moving is very constrained. For example, here, if the computer player is moving to the right, it can't go beyond the border. In that case, there's 100% chance it'll land on the intended square. If the computer player is trying to move to the last square here, there's a 90% chance of it hitting its intended square and a 10% chance of it falling short. Similarly, if our computer player is just trying to move over by one, there's a 90% chance it will hit the square once and a 10% chance it will overshoot.

59 – Sloppy Isolation Expectimax – lang_en_vs52.srt (1. Game Playing)

.....

Let's use a simple six move isolation game board to illustrate the ideas. We'll add green circles to show probability nodes. It's O's turn and it has four possible moves. The first node we'll explore is the one where O tries to go to the left most top position, but if it tries, it'll only succeed 90% of the time. And 10% of the time it will end up at the position to its immediate left. Yep, its other moves are to try to go one move to the left, which it will do 90% of the time and 10% of the time, it'll overshoot. But if it attempts the other two positions it has no chance to overshoot or undershoot, so those will be easy. Yep, but let's concentrate on the left most branch first. When it's X's turn, it'll have three possible moves. It can try to go immediately right with 90% chance of being successful and 10% chance of overshooting. Similarly, if it tries to go the whole way to the right, it will make it 90% of the time and undershoot 10% of the time. And if goes diagonally up and right it will hit it 100% of the time. So now we array to

do some calculation. Using our standard number of my moves evaluation function. O has one move available to it in this game board. And two nodes available to it here. That means the expected value of this branch is $0.9 \times 1 + 0.1 \times 2$ or 1.1. Then for the next node over we have a value of 2 for this node times 0.9 plus the value of the right node, which is 1, times 0.1. But notice we don't have to evaluate the right node. Since this level is a min level, as soon as we saw the 0.9 times 2 we had a 1.8. That's already bigger than the 1.1 we had on the left side, so we know we're never going to use this node. We don't have to evaluate this node at all. That's okay in this case, because we know the valuation function can't go below zero. But what if our valuation function could go negative? Well then we couldn't prune. In general, with EXPECTIMAX, you can only prune when you have known bounds on the values that will be returned by the EXPECTIMAX function. Okay, so here we'll actually choose the next branch because its value is 1 and its probability is also 1. So, expected value for this node is 1. Let's try the next sub tree over. This one is easy, looking at the bottom nodes, everything evaluates to 2, therefore all branches will evaluate to 2 and this node's value is also 2. Great, let's try the next sub tree. Here again, all the lower nodes are 2s. So the nodes value will be 2. The next one over is a bit more interesting, the left most bottom node has only one move remaining for O. And that's multiplied by a 0.9. Added to that is the 2 times the 0.1 in the node over in case X overshoots. That leads to a sum of 1.1. Looking at the next branch, if X tries to go the whole way to the right, we get a 2 times 0.9 or 1.8. We could prune this other branch now. But let's calculate it out for illustration purposes. Okay. Well, the evaluation function number of my moves which is turn one here. So 2 times 0.9 plus 1 times 0.1 would equal 1.9. But the last option is the best. It returns a 1 with a probability of 100%. So the value of this node would be 1. The next option is pretty easy to calculate. Everything comes up 2s. The one after that is also easy as all the moves are 100% accurate. Since it's a mid node we'll choose a 1. Great, now that we have all the values calculated, we can choose the branch with the highest value at the top of the max node. Looks like this branch wins. Yep, though it won't help O in this example game, O is going to lose unless X makes a bad choice or is unlucky. But if we know our player O is going to lose, why shouldn't we try to choose a branch that would still have the possibility for X to be unlucky or choose poorly? That is a good point. In fact, if we use this algorithm and search the end game, the algorithm might choose a branch where X could be unlucky and lose. Unfortunately, our simple number of my moves evaluation function is not really capturing that possibility. Searching deeper to end game should solve the problem. It'd be a good exercise to do, but now that we've covered everything, I'm excited to get back to the challenge question we started at the beginning of this section.

60 – Expectimax Alpha-Beta Pruning – lang_en_vs52.srt (1. Game Playing)

At the beginning of this section, we gave a challenge question. Let's go over it carefully now. The leftmost branches values calculated by multiplying 0.1 x the minimum value which is $8 + 0.5 \times$ the minimum value of the next branch which is $5 + 0.4 \times 8$. The sum of that is equal to 6.5. The bottom left node of the middle subtree returns a value of 0. 0.5×0 is still 0. We don't even need to evaluate the right branch of the middle tree now. The maximum we could get is a 0.5×10 which = 5. Since 5 is less than 6.5, there's no possible way for the right branch of the subtree to matter so we can ignore it. Note that if our nodes were ordered better, we could have had a 0 on the leftmost branch. Then, we could have ignored everything else. As we know that the value of this subtree is going to be 5 or lower. In other words, we would have known that, 0.5×0 or less + 0.5 times 10 or less is still going to be less than or equal to 5. For the right subtree, we have 0.1×3 which = 0.3. We still to get a 10 in the next branch or on some of the next branches so we get a 0.9 times 10 out of the rest of the subtree. So we just still continue because we're going to actually get a 9.3 in the end. Note that there's a lesson here. We should probably evaluate these subtrees with the highest probabilities first and the highest expected values first. However in this case, we continue with 0.5×9 or 4.5. So we have $0.3 + 4.5$ so far which is = to 4.8. There's still a possibility that we can get a value better than 6.5 so we continue. The next branch yields a 2. At this point, we know that this

subtree is going to have a value that is less than or equal to $0.1 \times 3 + 0.5 \times 9 + 0.4 \times 2$, which = 5.6. Thus, we are done. The last branch does not matter. Our player is going to choose the leftmost branch. Let's go to Shelly and one final quiz on the section.

61 – Probabilistic Alpha-Beta Pruning – lang_en_vs51.srt (1. Game Playing)

.....

What nodes can be pruned in this probabilistic game tree? Check the boxes for nodes or even full branches that could be pruned. Then find expected values of each remaining node. You can enter an exact number or a range, like less than or equal to 4.

62 – Probabilistic Alpha-Beta Pruning Solution – lang_en_vs52.srt (1. Game Playing)

.....

Here's the answer. We can prune these nodes and these branches. The final answer is 5.2.

2. SEARCH

1 – Introducing Peter Norvig – lang_en_vs1.srt (2. Search)

.....

Today we have the privilege of meeting Peter Norvig, whose online course with Sebastian Thrun on AI was one of the first big MOOCs and led to Udacity and a lot of entrepreneurial interest in teaching online classes. He's the director of research at Google and previously directed Google's core search algorithms group. He worked as a top NASA computer scientist, taught at Berkeley, and wrote the most popular AI textbook ever, which hopefully you all are enjoying right now. Thanks for coming today Peter. Well thanks for having me here Thad. I'm glad to see so many students interested in the course and in the Masters program.

2 – Intro to Search – lang_en_vs52.srt (2. Search)

.....

Peter is going to teach us about search. While we already did a lot of searching in the game playing section, most search algorithms are not adversarial. Peter, why is search so important in AI? Well, to me, AI is all about figuring out what to do when you don't know what to do. Regular programming is writing instructions to make the computer do what you want when you do know what to do. And AI is for when you don't know. And search is one of the key areas where we can figure out what to do by planning a sequence of steps, even when we have no idea what the first step should be until we solve the search problem. When watching these lessons, please pay careful attention to the A* algorithm. It is one of the most famous AI algorithms, and a lot of job interviewers in the field will assume you are familiar with its concepts. Let's do a challenge question to give you a preview of what you'll be learning.

3 – Challenge 1 Tri city Search – lang_en_vs52.srt (2. Search)

.....

The Carter Center in Atlanta is famous for monitoring elections to make sure they are fair. This year, they decided to have their interns practice on local elections. Sally, a Georgia Tech computer scientist turned activist, needs to visit these three polling stations on election day. The night before, she plans to stay at a hotel close to one of the polling stations. After checking the voting machines at the first station, she will visit each of the other two in turn. Sally is on her bicycle, so she wants to minimize her travel distance, and she is willing to go the wrong way down one way streets to do it. Fortunately, Sally took the artificial intelligence course. So she knows how to write a program that tells her where to start and the optimal streets used to get to each polling station. Proud of being a nerd, Sally wrote her program such that it had minimal runtime and memory overhead. How did she do it? Choose the best answer. Remember, these challenge questions are designed to give a preview of the lesson to come. We do not expect you to know how to do the answer yet.

4 – Challenge 1 Tri city Search Solution – lang_en_vs52.srt (2. Search)

.....

There are many ways to solve what we'll call the tri-cities problem. A straightforward way is do a search, from Buckhead to Scottdale, then from Scottdale to Little Five Points, then from Little Five Points to Buckhead. However, this method would explore many more street intersections than necessary. The trick here is to start the search from each of the three locations simultaneously, using the A* algorithm, which we'll call the tridirectional A* search. So the correct answer is none of the above. We'll revisit this challenge question at the end of the lesson in more detail

5 – Challenge 2 Rubik's Cube – lang_en_vs52.srt (2. Search)

.....

Here's another question, but it's more of a research question. In the 1980s, everyone owned a Rubik's Cube puzzle. The goal was to have all the same colors on the same side. The levels of the cube could be twisted on the vertical or horizontal. How do we design a search algorithm for Rubik's Cube that guarantees the least number of moves required to finish from any starting state? We'll consider each quarter turn to be a move. Choose a search strategy most likely to be part of your algorithm.

6 – Challenge 2 Rubik's Cube Solution – lang_en_vs52.srt (2. Search)

.....

As it turns out, the correct answer is iterative deepening A. *This problem is much harder than it sounds, simply due to the size of the search space. This table shows the number of nodes in the search tree as a function of depth. Clearly the problem becomes intractable quickly. However, in 1997, Richard Korf published an approach using iterative deepening A search.* He had to search to a median depth of 18 for the random configurations he attempted. The question is, what admissible heuristic did Korf invent for his A* search? To find out, follow the link to his paper in the instructor notes. This problem continued to bother computer science researchers. In 2010, Thomas Rokiki and Morley Davidson proved that the maximum number of quarter turns needed to solve the cube is 26, using 29 CPU years of idle computer time at the Ohio Super Computing Center. Games have a long history of inspiring rapid to nasty for AI researchers. As Peter teaches us about search, keep in mind interesting problems like these, and look for clues as to how we might use search techniques to solve them.

7 – Introduction – lang_en_vs1.srt (2. Search)

.....

[PROBLEM SOLVING] In this unit we're going to talk about problem solving. The theory and technology of building agents that can plan ahead to solve problems. In particular, we're talking about problem solving where the complexity of the problem comes from the idea that there are many states. As in this problem here. A navigation problem where there are many choices to start with. And the complexity comes from picking the right choice now and picking the right choice at the next intersection and the intersection after that. Streaming together a sequence of actions. This is in contrast to the type of complexity shown in this picture, where the complexity comes from the partial observability that we can't see through the fog where the possible paths are. We can't see the results of our actions and even the actions themselves are not known. This type of complexity will be covered in a later unit. Here's an example of a problem. This is a route-finding problem where we're given a start city, in this case, Arad, and a destination, Bucharest, the capital of Romania, from which this is a corner of the map. And the problem then is to find a route from Arad to Bucharest. The actions that the agent can execute when driving from one city to the next along one of the roads shown on the map. The question is, is there a solution that the agent can come up with given the knowledge shown here to the problem of driving from Arad to Bucharest?

8 – What Is A Problem – lang_en_vs2.srt (2. Search)

.....

And the answer is no, there is no solution that the agent can come up with because Bucharest doesn't appear on the map and so the agent doesn't know any actions that can arrive there. So let's give the agent a better chance. Now we've given the agent the full map of Romania. The start is in Arad and the destination, or goal, is in Bucharest and the agent is given the problem of coming up with a sequence of actions that will arrive at the destination. Now is it possible for the agent to solve this problem? And the answer is yes. There are many routes or steps or sequences of actions that will arrive at the destination. Here's one of them. Starting out in Arad taking this step first, then this one. And then this one, then this one, and then this one, to arrive at the destination. So that would count as a solution to the problem. So, sequence of actions, chained together, that are guaranteed to get us to the goal. Now, let's formally define what a problem looks like. A problem can be broken down into a number of components. First, the initial state that the agent starts out with. In our route finding problem, the initial state was the agent being in the city of Arad. Next a function actions that takes a status input and returns a set of possible actions that the agent can execute when the agent is in the state. In some problems, the agent will have the same actions available in all states. And in other problems they'll have different actions dependent on the state. In the route finding problem, the actions are dependent on the state. When we're in one city we can take the routes to the neighboring cities, but we can't go to any other cities. Next, we have. A function called result which takes as input a state. And an action and delivers as its output, a new state. So for example, if the agent is in the city of Arad, and, that would be the state, and takes the action of driving along route E671 towards Timisoara, then the result of applying that action in that state would be the new state where the agent is in the city of Timisoara. Next, we need a function.

Called GoalTest which takes a state. And returns a boolean value true or false, telling us if this data is a goal or not. In a route finding problem, the only goal would be being in the destination city, the city of Bucharest, and all the other states would return false for the goal test. And finally, one more thing, which is a path cost function which takes a path, a sequence of state action transitions and returns a number which is the cost of that path. Now for most of the problems we'll deal with we'll make the path cost function be additive so that the cost of the path is just the sum of the individual steps. And so we'll implement this path cost function in terms of a step cost function. The step cost function takes a state, an action, and the resulting state from that action. And returns a number n which is the cost of that action. In the route finding example, the cost might be the number of miles traveled, or maybe the number of minutes it takes to get to that destination.

9 – Example Route Finding – lang_en_vs5.srt (2. Search)

.....

Now let's see how the definition of a problem maps onto the root founding domain. First, the initial state we're given. Let's say we start of in Arad. And the goal test, let's say that the state of being in Bucharest is the only state that counts as a goal. And all other states are not goals. Now the set of all the states here is known as the states space. And we navigate the states space by applying actions. The actions are specific to each city. So when we're in Arad, there are three possible actions, to follow this road, this one, or this one. And as we follow them we build paths or sequences of actions. So just being in Arad is the path of length zero. And now we could start exploring the space and add in this path of length one. This path of length one and this path of length one. We can add in another path, here of length two and another path here of length two. Here's another path of length two. Here's a path of length three, another path of length two, and so on. Now at every point, we want to separate the state out into three parts. First, the ends of the paths, the farthest paths that have been explored, we call the frontier. And so the frontier in this case consists of these states and are the furthest out we can explore. And then, to the left of that in this diagram, we have the explored part of the state. And then off to the right we have the unexplored. So let's write down those three components. We have the frontier, we have the unexplored region, and we have. Explored region. One more thing. In this diagram, we've labeled the step cost of each action along the route. So the step cost of going between Neamt and Iasi would be 87, corresponding to a distance of 87 kilometers. And then the path cost is just the sum of the step cost. So the cost of the path of going from Arad to Oradea would be 71 plus 75.

10 – Tree Search – lang_en_vs1.srt (2. Search)

.....

[Narrator] Now let's define a function for solving problems. It's called Tree Search because it superimposes a search tree over the state space. Here's how it works: It starts off by initializing the frontier to be the path consisting of only the initial states, and then it goes into a loop in which it first checks to see do we still have anything left in the frontier? If not we fail, there can be no solution. If we do have something, then we make a choice. Tree Search is really a family of functions not a single algorithm which depends on how we make that choice, and we'll see some of the options later. If we go ahead and make a choice of one of the paths on the frontier and remove that path from the frontier, we find the state which is at the end of the path, and if that state's a go then we're done. We found a path to the goal; otherwise, we do what's called expanding that path. We look at all the actions from that state, and we add to the path the actions and the result of that state; so we get a new path that has the old path, the action and the result of that action, and we stick all of those paths back onto the frontier. Now Tree Search represents a whole family of algorithms, and where you get the family resemblance is that they're all looking at the frontier, copying items off and and looking to see if their goal tests, but where you get the difference is right here, in the choice of how you're going to expand the next item on the frontier, which path do we look at first, and we'll go through different sets of algorithms that make different choices for which path to look at first. The first algorithm I want to consider is called Breadth-First Search. Now it could be called shortest-first search because what it does is always choose of the frontier one of the paths that hadn't been considered yet that's the shortest possible. So how

does it work? Well we start off with the path of length 0, starting in the start state, and that's the only path in the frontier so it's the shortest one so we pick it, and then we expand it, and we add in all the paths that result from applying all the possible actions. So now we've removed this path from the frontier, but we've added in 3 new paths. This one, this one, and this one. Now we're in a position where we have 3 paths on the frontier, and we have to pick the shortest one. Now in this case all 3 paths have the same length, length 1, so we break the tie at random or using some other technique, and let's suppose that in this case we choose this path from Arad to Sibiu. Now the question I want you to answer is once we remove that from the frontier, what paths are we going to add next? So show me by checking off the cities that ends the paths, which paths are going to be added to the frontier?

11 – Tree Search Continued – lang_en_vs1.srt (2. Search)

[Male narrator] The answer is that in Sibiu, the action function gives us 4 actions corresponding to traveling along these 4 roads, so we have to add in paths for each of those actions. One of those paths goes here, the other path continues from Arad and goes out here. The third path continues out here and then the fourth path goes from here— from Arad to Sibiu and then backtracks back to Arad. Now, it may seem silly and redundant to have a path that starts in Arad, goes to Sibiu and returns to Arad. How can that help us get to our destination in Bucharest? But we can see if we're dealing with a tree search, why it's natural to have this type of formulation and why the tree search doesn't even notice that it's backtracked. What the tree search does is superimpose on top of the state space a tree of searches, and the tree looks like this. We start off in state A, and in state A, there were 3 actions, so we gave those paths going to Z, S, and T. And from S, there were 4 actions, so that gave us paths going from O, F, R, and A, and then the tree would continue on from here. We'd take one of the next items and we'd move it and continue on, but notice that we returned to the A state in the state space, but in the tree, it's just another item in the tree. Now, here's another representation of the search space and what's happening is as we start to explore the state, we keep track of the frontier, which is the set of states that are at the end of the paths that we haven't explored yet, and behind that frontier is the set of explored states, and ahead of the frontier is the unexplored states. Now the reason we keep track of the explored states is that when we want to expand and we find a duplicate— so say when we expand from here, if we pointed back to state T, if we hadn't kept track of that, we would have to add in a new state for T down here. But because we've already seen it and we know that this is actually a regressive step into the already explored state, now, because we kept track of that, we don't need it anymore.

12 – Graph Search – lang_en_vs1.srt (2. Search)

Now we see how to modify the Tree Search Function to make it be a Graph Search Function to avoid those repeated paths. What we do, is we start off and initialize a set called the explored set of states that we have already explored. Then, when we consider a new path, we add the new state to the set of already explored states, and then when we are expanding the path and adding in new states to the end of it, we don't add that in if we have already seen that new state in either the frontier or the explored. Now back to Breadth First Search. Let's assume we are using the Graph Search so that we have eliminated the duplicate paths. Arad is crossed off the list. The path that goes from Arad to Sibiu and back to Arad is removed, and we are left with these one, two, three, four, five possible paths. Given these 5 paths, show me which ones are candidates to be expanded next by the Breadth First Search Algorithm.

13 – Breadth First Search 1 – lang_en_vs1.srt (2. Search)

[Male narrator] And the answer is that Breadth – First Search always considers the shortest paths first, and in this case, there's 2 paths of length 1, and 1, the paths from Arad to Zerind and Arad to Timisoara, so those would be the 2 paths that would be considered. Now, let's suppose that the tie is broken in some way and we chose this path from

Arad to Zerind. Now, we want to expand that node. We remove it from the frontier and put it in the explored list and now we say, “What paths are we going to add?” So check off the ends of the paths the cities that we’re going to add.

14 – Breadth First Search 2 – lang_en_vs1.srt (2. Search)

.....

[Male narrator] In this case, there’s nothing to add because of the 2 neighbors, 1 is in the explored list and 1 is in the frontier, and if we’re using graph search, then we won’t add either of those.

15 – Breadth First Search 3 – lang_en_vs1.srt (2. Search)

.....

[Male narrator] So we move on, we look for another shortest path. There’s one path left of length 1, so we look at that path, we expand it, add in this path, put that one on the explored list, and now we’ve got 3 paths of length 2. We choose 1 of them, and let’s say we choose this one. Now, my question is show me which states we add to the path and tell me whether we’re going to terminate the algorithm at this point because we’ve reached the goal or whether we’re going to continue.

16 – Breadth First Search 4 – lang_en_vs1.srt (2. Search)

.....

[Male narrator] The answer is that we add 1 more path, the path to Bucharest. We don’t add the path going back because it’s in the explored list, but we don’t terminate it yet. True, we have added a path that ends in Bucharest, but the goal test isn’t applied when we add a path to the frontier. Rather, it’s applied when we remove that path from the frontier, and we haven’t done that yet.

17 – Breadth First Search 5 – lang_en_vs1.srt (2. Search)

.....

[Male narrator] Now, why doesn’t the general tree search or graph search algorithm stop when it adds a goal node to the frontier? The reason is because it might not be the best path to the goal. Now, here we found a path of length 2 and we added a path of length 3 that reached the goal. The general graph search or tree search doesn’t know that there might be some other path that we could expand that would have a distance of say, $2\frac{1}{2}$, but there’s an optimization that could be made. If we know we’re doing Breadth – First Search and we know there’s no possibility of a path of length $2\frac{1}{2}$. Then we can change algorithm so that it checks states as soon as they’re added to the frontier rather than waiting until they’re expanded and in that case, we can write a specific Breadth – First Search routine that terminates early and gives us a result as soon as we add a goal state to the frontier. Breadth – First Search will find this path that ends up in Bucharest, and if we’re looking for the shortest path in terms of number of steps, Breadth – First Search is guaranteed to find it, But if we’re looking for the shortest path in terms of total cost by adding up the step costs, then it turns out that this path is shorter than the path found by Breadth – First Search. So let’s look at how we could find that path.

18 – Uniform Cost Search – lang_en_vs1.srt (2. Search)

.....

An algorithm that has traditionally been called uniform-cost search but could be called cheapest-first search, is guaranteed to find the path with the cheapest total cost. Let’s see how it works. We start out as before in the start state. And we pop that empty path off. Move it from the frontier to explored, and then add in the paths out of that state. As before, there will be 3 of those paths. And now, which path are we going to pick next in order to expand according to the rules of cheapest first?

19 – Uniform Cost Search 1 – lang_en_vs1.srt (2. Search)

.....

Cheapest first says that we pick the path with the lowest total cost. And that would be this path. It has a cost of 75 compared to the cost of 118 and 140 for the other paths. So we get here. We take that path off the frontier, put it on the explored list, add in its neighbors. Not going back to Arad, but adding in this new path. Summing up the total cost of that path, $71 + 75$ is 146 for this path. And now the question is, which path gets expanded next?

20 – Uniform Cost Search 2 – lang_en_vs1.srt (2. Search)

.....

Of the 3 paths on the frontier, we have ones with a cost of 146, 140, and 118. And that's the cheapest, so this one gets expanded. We take it off the frontier, move it to explored, add in its successors. In this case it's only 1. And that has a path total of 229. Which path do we expand next? Well, we've got 146, 140, and 229. So 140 is the lowest. Take it off the frontier. Put it on explored. Add in this path for a total cost of 220. And this path for a total cost of 239. And now the question is, which path do we expand next?

21 – Uniform Cost Search 3 – lang_en_vs1.srt (2. Search)

.....

The answer is this one, 146. Put it on explored. But there's nothing to add because both of its neighbors have already been explored. Which path do we look at next?

22 – Uniform Cost Search 4 – lang_en_vs1.srt (2. Search)

.....

The answer is this one. Two-twenty is less than 229 or 239. Take it off the frontier. Put it on explored. Add in 2 more paths and sum them up. So, 220 plus 146 is 366. And 220 plus 97 is 317. Okay, and now, notice that we're closing in on Bucharest. We've got 2 neighbors almost there, but neither of them is their turn yet. Instead, the cheapest path is this one over here, so move it to the explored list. Add 70 to the path cost so far, and we get 299. Now the cheapest node is 239 here, so we expand, finally, into Bucharest at a cost of 460. And now the question is are we done? Can we terminate the algorithm?

23 – Uniform Cost Search 5 – lang_en_vs2.srt (2. Search)

.....

And the answer is no, we're not done yet. We have reached a goal state. We put a path onto the frontier that reaches the goal of Bucharest, but we haven't popped that path off the frontier. And uniform cost search continues to search until we pop it off the frontier. We continue looking to see if there's a better path that also reaches the goal. So let's see. I forgot to say Thagoras is explored, so let's continue. Let's take the cheapest path on the frontier and expand that. The cheapest path is this, 146. We'll expand that, get another path into (Sibiyu). That's a worse path than we had before, so we'll drop it. Then let's see what's next. Looking on the frontier, the cheapest now is here at 299. We'll expand that. We get a path of cost 374. Put that on the frontier. Now let's go again. Now the cheapest path is over here at 317. We'll mark that as explored and add two more paths—one here that's a worse path, so it gets dropped. And one path that also reaches the goal, and that has a total cost of 418. So that just shows it's a good thing we waited, a good thing we didn't stop when we found the first path to the goal, because now this second path found is actually cheaper than the first path found. But we're not going to stop here because we still haven't popped off a path that reaches the goal. So we'll continue. What's next? Now the cheapest path on the frontier is here at 366. We expand that, and we get paths that are worse paths to points we've already seen before. So nothing new goes on the frontier. Next, the cheapest path on the frontier is at 374. Again, expanding that leads nothing useful.

Only worse paths than we've seen before. And now finally, the cheapest path on the frontier is this 418 path to Bucharest, so we pop that off, and now we reach the goal, and now we stop. So even though we found the 460 path first, we don't stop there because there might be another path that also reaches the goal that's cheaper. We keep on going until we popped a path off of the frontier that reaches the goal, and that's why uniform cost search is guaranteed to find the cheapest path to the goal.

24 – Search Comparison – lang_en_vs1.srt (2. Search)

.....

So, we've looked at 2 search algorithms. One, breadth-first search, in which we always expand first the shallowest paths, the shortest paths. Second, cheapest-first search, in which we always expand first the path with the lowest total cost. And I'm going to take this opportunity to introduce a third algorithm, depth-first search, which is in a way the opposite of breadth-first search. In depth-first search, we always expand first the longest path, the path with the most lengths in it. Now, what I want to ask you to do is for each of these nodes in each of the trees, tell us in what order they're expanded, first, second, third, fourth, fifth and so on by putting a number into the box. And if there are ties, put that number in and resolve the ties in left to right order. Then I want you to ask one more question or answer one more question which is are these searches optimal? That is, are they guaranteed to find the best solution? And for breadth-first search, optimal would mean finding the shortest path. If you think it's guaranteed to find the shortest path, check here. For cheapest first, it would mean finding the path with the lowest total path cost. Check here if you think it's guaranteed to do that. And we'll allow the assumption that all costs have to be positive. And in depth first, cheapest or optimal would mean, again, as in breadth first, finding the shortest possible path in terms of number of lengths. Check here if you think depth first will always find that.

25 – Search Comparison 1 – lang_en_vs1.srt (2. Search)

.....

Here are the answers. Breadth-first search, as the name implies, expands nodes in this order. One, 2, 3, 4, 5, 6, 7. So, it's going across a stripe at a time, breadth first. Is it optimal? Well, it's always expanding in the shortest paths first, and so wherever the goal is hiding, it's going to find it by examining no longer paths, so in fact, it is optimal. Cheapest first, first we expand the path of length zero, then the path of length 2. Now there's a path of length 4, path of length 5, path of length 6, a path of length 7, and finally, a path of length 8. And as we've seen, it's guaranteed to find the cheapest path of all, assuming that all the individual step costs are not negative. Depth-first search tries to go as deep as it can first, so it goes 1, 2, 3, then backs up, 4, then backs up, 5, 6, 7. And you can see that it doesn't necessarily find the shortest path of all. Let's say that there were goals in position 5 and in position 3. It would find the longer path to position 3 and find the goal there and would not find the goal in position 5. So, it is not optimal.

26 – Search Comparison 2 – lang_en_vs1.srt (2. Search)

.....

Given the non-optimality of depth-first search, why would anybody choose to use it? Well, the answer has to do with the storage requirements. Here I've illustrated a state space consisting of a very large or even infinite binary tree. As we go to levels 1, 2, 3, down to level n , the tree gets larger and larger. Now, let's consider the frontier for each of these search algorithms. For breadth-first search, we know a frontier looks like that, and so when we get down to level n , we'll require a storage space of 2^n to the n of pass in a breadth-first search. For cheapest first, the frontier is going to be more complicated. It's going to sort of work out this contour of cost, but it's going to have a similar total number of nodes. But for depth-first search, as we go down the tree, we start going down this branch, and then we back up, but at any point, our frontier is only going to have n nodes rather than 2^n nodes, so that's a substantial savings for depth-first search. Now, of course, if we're also keeping track of the explored set, then we don't get that much savings. But without the explored set, depth-first search has a huge advantage in terms

of space saved. One more property of the algorithms to consider is the property of completeness, meaning if there is a goal somewhere, will the algorithm find it? So, let's move from very large trees to infinite trees, and let's say that there's some goal hidden somewhere deep down in that tree. And the question is, are each of these algorithms complete? That is, are they guaranteed to find a path to the goal? Mark off the check boxes for the algorithms that you believe are complete in this sense.

27 – Search Comparison 3 – lang_en_vs1.srt (2. Search)

.....

The answer is that breadth-first search is complete, so even if the tree is infinite, if the goal is placed at any finite level, eventually, we're going to march down and find that goal. Same with cheapest first. No matter where the goal is, if it has a finite cost, eventually, we're going to go down and find it. But not so for depth-first search. If there's an infinite path, depth-first search will keep following that, so it will keep going down and down and down along this path and never get to the path that the goal consists of and never get to the path on which the goal sits. So, depth-first search is not complete.

28 – More On Uniform Cost – lang_en_vs1.srt (2. Search)

.....

Let's try to understand a little better how uniform cost search works. We start at a start state, and then we start expanding out from there looking at different paths, and what we end of doing is expanding in terms of contours like on a topological map, where first we span out to a certain distance, then to a farther distance, and then to a farther distance. Now at some point we meet up with a goal. Let's say the goal is here. Now we found a path from the start to the goal. But notice that the search really wasn't directed at any way towards the goal. It was expanding out everywhere in the space and depending on where the goal is, we should expect to have to explore half the space, on average, before we find the goal. If the space is small, that can be fine, but when spaces are large, that won't get us to the goal fast enough. Unfortunately, there is really nothing we can do, with what we know, to do better than that, and so if we want to improve, if we want to be able to find the goal faster, we're going to have to add more knowledge. The type of knowledge that is proven most useful in search is an estimate of the distance from the start state to the goal. So let's say we're dealing with a route-finding problem, and we can move in any direction—up or down, right or left— and we'll take as our estimate, the straight line distance between a state and a goal, and we'll try to use that estimate to find our way to the goal fastest. Now an algorithm called greedy best-first search does exactly that. It expands first the path that's closest to the goal according to the estimate. So what do the contours look like in this approach? Well, we start here, and then we look at all the neighboring states, and the ones that appear to be closest to the goal we would expand first. So we'd start expanding like this and like this and like this and like this and that would lead us directly to the goal. So now instead of exploring whole circles that go out everywhere with a certain space, our search is directed towards the goal. In this case it gets us immediately towards the goal, but that won't always be the case if there are obstacles along the way. Consider this search space. We have a start state and a goal, and there's an impassable barrier. Now greedy best-first search will start expanding out as before, trying to get towards the goal, and when it reaches the barrier, what will it do next? Well, it will try to increase along a path that's getting closer and closer to the goal. So it won't consider going back this way which is farther from the goal. Rather it will continue expanding out along these lines which always get closer and closer to the goal, and eventually it will find its way towards the goal. So it does find a path, and it does it by expanding a small number of nodes, but it's willing to accept a path which is longer than other paths. Now if we explored in the other direction, we could have found a much simpler path, a much shorter path, by just popping over the barrier, and then going directly to the goal. but greedy best-first search wouldn't have done that because that would have involved getting to this point, which is this distance to the goal, and then considering states which were farther

from the goal. What we would really like is an algorithm that combines the best parts of greedy search which explores a small number of nodes in many cases and uniform cost search which is guaranteed to find a shortest path. We'll show how to do that next using an algorithm called the A-star algorithm.

29 – A* Search – lang_en_vs1.srt (2. Search)

[Male narrator] A* Search works by always expanding the path that has a minimum value of the function f which is defined as a sum of the $g + h$ components. Now, the function g of a path is just the path cost, and the function h of a path is equal to the h value of the state, which is the final state of the path, which is equal to the estimated distance to the goal. Here's an example of how A* works. Suppose we found this path through the state's base to a state x and we're trying to give a measure to the value of this path. The measure f is a sum of g , the path cost so far, and h , which is the estimated distance that the path will take to complete its path to the goal. Now, minimizing g helps us keep the path short and minimizing h helps us keep focused on finding the goal and the result is a search strategy that is the best possible in the sense that it finds the shortest length path while expanding the minimum number of paths possible. It could be called "best estimated total path cost first," but the name A* is traditional. Now let's go back to Romania and apply the A* algorithm and we're going to use a heuristic, which is a straight line distance between a state and the goal. The goal, again, is Bucharest, and so the distance from Bucharest to Bucharest is, of course, 0. And for all the other states, I've written in red the straight line distance. For example, straight across like that. Now, I should say that all the roads here I've drawn as straight lines, but actually, roads are going to be curved to some degree, so the actual distance along the roads is going to be longer than the straight line distance. Now, we start out as usual—we'll start in Arad as a start state—and we'll expand out Arad and so we'll add 3 paths and the evaluation function, f , will be the sum of the path length, which is given in black, and the estimated distance, which is given in red. And so the path length from this path will be $140+253$ or 393 ; for this path, $75+374$, or 449 ; and for this path, $118+329$, or 447 . And now, the question is out of all the paths that are on the frontier, which path would we expand next under the A* algorithm?

30 – A* Search Solution – lang_en_vs1.srt (2. Search)

The answer is that we select this path first—the one from Arad to Sibiu—because it has the smallest value— 393 —of the sum $f=g+h$.

31 – A Search 1 – lang_en_vs1.srt (2. Search)

Let's go ahead and expand this node now. So we're going to add 3 paths. This one has a path cost of 291 and an estimated distance to the goal of 380 , for a total of 671 . This one has a path cost of 239 and an estimated distance of 176 , for a total of 415 . And the final one is $220+193=413$. And now the question is which state to we expand next?

32 – A Search 1 Solution – lang_en_vs1.srt (2. Search)

The answer is we expand this path next because its total, 413 , is less than all the other ones on the front tier—although only slightly less than the 415 for this path.

33 – A* Search 2 – lang_en_vs1.srt (2. Search)

So we expand this node, giving us 2 more paths—this one with an f -value of 417 , and this one with an f -value of 526 . The question again—which path are we going to expand next?

34 – A* Search 2 Solution – lang_en_vs1.srt (2. Search)

.....

And the answer is that we expand this path, Fagaras, next, because its f-total, 415, is less than all the other paths in the front tier.

35 – A* Search 3 – lang_en_vs1.srt (2. Search)

.....

Now we expand Fagaras and we get a path that reaches the goal and it has a path length of 450 and an estimated distance of 0 for a total f value of 450, and now the question is: What do we do next? Click here if you think we're at the end of the algorithm and we don't need to expand next or click on the node that you think we will expand next.

36 – A* Search 3 Solution – lang_en_vs1.srt (2. Search)

.....

The answer is that we're not done yet, because the algorithm works by doing the goal test, when we take a path off the front tier, not when we put a path on the front tier. Instead, we just continue in the normal way and choose the node on the front tier which has the lowest value. That would be this one—the path through Pitesti, with a total of 417.

37 – A* Search 4 – lang_en_vs1.srt (2. Search)

.....

So let's expand the node at Pitesti. We have to go down this direction, up, then we reach a path we've seen before, and we go in this direction. Now we reach Bucharest, which is the goal, and the h value is going to be 0 because we're at the goal, and the g value works out to 418. Again, we don't stop here just because we put a path onto the front tier, we put it there, we don't apply the goal test next, but, now we go back to the front tier, and it turns out that this 418 is the lowest-cost path on the front tier. So now we pull it off, do the goal test, and now we found our path to the goal, and it is, in fact, the shortest possible path. In this case, A-star was able to find the lowest-cost path. Now the question that you'll have to think about, because we haven't explained it yet, is whether A-star will always do this. Answer yes if you think A-star will always find the shortest cost path, or answer no if you think it depends on the particular problem given, or answer no if you think it depends on the particular heuristic estimate function, h.

38 – A Search 5 – lang_en_vs1.srt (2. Search)

.....

The answer is that it depends on the h function. A-star will find the lowest-cost path if the h function for a state is less than the true cost of the path to the goal through that state. In other words, we want the h to never overestimate the distance to the goal. We also say that h is optimistic. Another way of stating that is that h is admissible, meaning is it admissible to use it to find the lowest-cost path. Think of all of these of being the same way of stating the conditions under which A-star finds the lowest-cost path.

39 – Optimistic Heuristic – lang_en_vs1.srt (2. Search)

.....

Here we give you an intuition as to why an optimistic heuristic function, h, finds the lowest-cost path. When A-star ends, it returns a path, p, with estimated cost, c. It turns out that c is also the actual cost, because at the goal the h component is 0, and so the path cost is the total cost as estimated by the function. Now, all the paths on the front

tier have an estimated cost that's greater than c , and we know that because the front tier is explored in cheapest-first order. If h is optimistic, then the estimated cost is less than the true cost, so the path p must have a cost that's less than the true cost of any of the paths on the front tier. Any paths that go beyond the front tier must have a cost that's greater than that because we agree that the step cost is always 0 or more. So that means that this path, p , must be the minimal cost path. Now, this argument, I should say, only goes through as is for tree search. For graph search the argument is slightly more complicated, but the general intuitions hold the same.

40 – State Spaces – lang_en_vs1.srt (2. Search)

.....

So far we've looked at the state space of cities in Romania– a 2-dimensional, physical space. But the technology for problem solving through search can deal with many types of state spaces, dealing with abstract properties, not just x-y position in a plane. Here I introduce another state space–the vacuum world. It's a very simple world in which there are only 2 positions as opposed to the many positions in the Romania state space. But there are additional properties to deal with as well. The robot vacuum cleaner can be in either of the 2 conditions, but as well as that each of the positions can either have dirt in it or not have dirt in it. Now the question is to represent this as a state space how many states do we need? The number of states can fill in this box here.

41 – State Spaces 1 – lang_en_vs1.srt (2. Search)

.....

And the answer is there are 8 states. There are 2 physical states that the robot vacuum cleaner can be in– either in state A or in state B. But in addition to that, there are states about how the world is as well as where the robot is in the world. So state A can be dirty or not. That's 2 possibilities. And B can be dirty or not. That's 2 more possibilities. We multiply those together. We get 8 possible states.

42 – State Spaces 2 – lang_en_vs1.srt (2. Search)

.....

Here is a diagram of the state space for the vacuum world. Note that there are 8 states, and we have the actions connecting the states just as we did in the Romania problem. Now let's look at a path through this state. Let's say we start out in this position, and then we apply the action of moving right. Then we end up in a position where the state of the world looks the same, except the robot has moved from position 'A' to position 'B'. Now if we turn on the sucking action, then we end up in a state where the robot is in the same position but that position is no longer dirty. Let's take this very simple vacuum world and make a slightly more complicated one. First, we'll say that the robot has a power switch, which can be in one of three conditions: on, off, or sleep. Next, we'll say that the robot has a dirt-sensing camera, and that camera can either be on or off. Third, this is the deluxe model of robot in which the brushes that clean up the dust can be set at 1 of 5 different heights to be appropriate for whatever level of carpeting you have. Finally, rather than just having the 2 positions, we'll extend that out and have 10 positions. Now the question is how many states are in this state space?

43 – State Spaces 3 – lang_en_vs1.srt (2. Search)

.....

The answer is that the number of states is the cross product of the numbers of all the variables, since they're each independent, and any combination can occur. For the power we have 3 possible positions. The camera has 2. The brush height has 5. The dirt has 2 for each of the 10 positions. That's 2^{10} or 1024. Then the robot's position can be any of those 10 positions as well. That works out to 307,200 states in the state space. Notice how a fairly trivial problem– we're only modeling a few variables and only 10 positions– works out to a large number of state spaces. That's why we need efficient algorithms for searching through states spaces.

44 – Sliding Blocks Puzzle – lang_en_vs1.srt (2. Search)

.....

I want to introduce one more problem that can be solved with search techniques. This is a sliding blocks puzzle, called a 15 puzzle. You may have seen something like this. So there are a bunch of little squares or blocks or tiles and you can slide them around. and the goal is to get into a certain configuration. So we'll say that this is the goal state, where the numbers 1-15 are in order left to right, top to bottom. The starting state would be some state where all the positions are messed up. Now the question is: Can we come up with a good heuristic for this? Let's examine that as a way of thinking about where heuristics come from. The first heuristic we're going to consider we'll call h_1 , and that is equal to the number of misplaced blocks. So here 10 and 11 are misplaced because they should be there and there, respectively, 12 is in the right place, 13 is in the right place, and 14 and 15 are misplaced. That's a total of 4 misplaced blocks. The 2nd heuristic, h_2 , is equal to the sum of the distances that each block would have to move to get to the right position. For this position, 10 would have to move 1 space to get to the right position, 11 would have to move 1, so that's a total of 2 so far, 13 is in the right place, 14 is 1 displaced, and 15 is 1 displaced, so that would also be a total of 4. Now, the question is: Which, if any, of these heuristics are admissible? Check the boxes next to the heuristics that you think are admissible.

45 – Sliding Blocks Puzzle 1 – lang_en_vs1.srt (2. Search)

.....

H_1 is admissible, because every tile that's in the wrong position must be moved at least once to get into the right position. So h_1 never overestimates. How about h_2 ? H_2 is also admissible, because every tile in the wrong position can be moved closer to the correct position no faster than 1 space per move. Therefore, both are admissible. But notice that h_2 is always greater than or equal to h_1 . That means that, with the exception of breaking ties, an A^* search using h_2 will always expand fewer paths than one using h_1

46 – Sliding Blocks Puzzle 2 – lang_en_vs1.srt (2. Search)

.....

Now, we're trying to build an artificial intelligence that can solve problems like this all on its own. You can see that the search algorithms do a great job of finding solutions to problems like this. But, you might complain that in order for the search algorithms to work, we had to provide it with a heuristic function. A heuristic function came from the outside. You might think that coming up with a good heuristic function is really where all the intelligence is. So, a problem solver that uses an heuristic function given to it really isn't intelligent at all. So let's think about where the intelligence could come from and can we automatically come up with good heuristic functions. I'm going to sketch a description of a program that can automatically come up with good heuristics given a description of a problem. Suppose this program is given a description of the sliding blocks puzzle where we say that a block can move from square A to square B if A is adjacent to B and B is blank. Now, imagine that we try to loosen this restriction. We cross out "B is blank," and then we get the rule "a block can move from A to B if A is adjacent to B," and that's equal to our heuristic h_2 because a block can move anywhere to an adjacent state. Now, we could also cross out the other part of the rule, and we now get "a block can move from any square A to any square B regardless of any condition. That gives us heuristic h_1 . So we see that both of our heuristics can be derived from a simple mechanical manipulation of the formal description of the problem. Once we've generated automatically these candidate heuristics, another way to come up with a good heuristic is to say that a new heuristic, h , is equal to the maximum of h_1 and h_2 , and that's guaranteed to be admissible as long as h_1 and h_2 are admissible because it still never overestimates, and it's guaranteed to be better because it's getting closer to the true value. The only problem with combining multiple heuristics like this is that there is some cause to compute the heuristic and it could take longer to compute even if we end up expanding pure paths. Crossing out parts of the rules like this is called "generating a relaxed problem." What we've done is we've taken the original problem, where it's hard to move squares around, and made it easier by relaxing one of the constraints. You can see that as adding new links in

the state space, so if we have a state space in which there are only particular links, by relaxing the problem it's as if we are adding new operators that traverse the state in new ways. So adding new operators only makes the problem easier, and thus never overestimates, and thus is admissible.

47 – Problems With Search – lang_en_vs1.srt (2. Search)

.....

We've seen what search can do for problem solving. It can find the lowest-cost path to a goal, and it can do that in a way in which we never generate more paths than we have to. We can find the optimal number of paths to generate, and we can do that with a heuristic function that we generate on our own by relaxing the existing problem definition. But let's be clear on what search can't do. All the solutions that we have found consist of a fixed sequence of actions. In other words, the agent Hirin Arad, thinks, comes up with a plan that it wants to execute and then essentially closes his eyes and starts driving, never considering along the way if something has gone wrong. That works fine for this type of problem, but it only works when we satisfy the following conditions. [Problem solving works when:] Problem-solving technology works when the following set of conditions is true: First, the domain must be fully observable. In other words, we must be able to see what initial state we start out with. Second, the domain must be known. That is, we have to know the set of available actions to us. Third, the domain must be discrete. There must be a finite number of actions to choose from. Fourth, the domain must be deterministic. We have to know the result of taking an action. Finally, the domain must be static. There must be nothing else in the world that can change the world except our own actions. If all these conditions are true, then we can search for a plan which solves the problem and is guaranteed to work. In later units, we will see what to do if any of these conditions fail to hold.

48 – A Note On Implementation – lang_en_vs1.srt (2. Search)

.....

Our description of the algorithm has talked about paths in the state space. I want to say a little bit now about how to implement that in terms of a computer algorithm. We talk about paths, but we want to implement that in some ways. In the implementation we talk about nodes. A node is a data structure, and it has four fields. The state field indicates the state at the end of the path. The action was the action it took to get there. The cost is the total cost, and the parent is a pointer to another node. In this case, the node that has state "S", and it will have a parent which points to the node that has state "A", and that will have a parent pointer that's null. So we have a linked list of nodes representing the path. We'll use the word "path" for the abstract idea, and the word "node" for the representation in the computer memory. But otherwise, you can think of those two terms as being synonyms, because they're in a one-to-one correspondence. Now there are two main data structures that deal with nodes. We have the "frontier" and we have the "explored" list. Let's talk about how to implement them. In the frontier the operations we have to deal with are removing the best item from the frontier and adding in new ones. And that suggests we should implement it as a priority queue, which knows how to keep track of the best items in proper order. But we also need to have an additional operation of a membership test as a new item in the frontier. And that suggests representing it as a set, which can be built from a hash table or a tree. So the most efficient implementations of search actually have both representations. The explored set, on the other hand, is easier. All we have to do there is be able to add new members and check for membership. So we represent that as a single set, which again can be done with either a hash table or tree.

49 – Challenge Question Revisited – lang_en_vs52.srt (2. Search)

.....

Searching is one of the most fundamental methods of solving problems in AI. We've now seen depth-first search, breadth-first search, uniform cost search in a star. In the book, we've read about iterative deepening and bidirectional search and the space and time advantages of these different search methods. Remember Sally from the

Carter Center? Let's use the knowledge from the section to look at the problem more thoroughly. Depth-first search is non-viable here. If we were unlucky, we might cross a country several times in just our first branch. Breadth-first search, or our uniform cost search would work, but we'd have to do three searches, one from Little 5 Points to Scottdale, one from Scottdale to Buckhead, and one from Buckhead to Little 5 Points. We would then determine the shortest two of the three paths. And so, the pulling station that is common between those two paths is a second pulling station we would visit. We could then start at either of the other two cities. The problem is that this method revisits some nodes repeatedly in the three searches, and the search would range much farther than is necessary. A bi-directional search would help considerably. Instead of each of the three searches requiring B to the D nodes, it would be done in B to the D over two nodes. But we would still revisit a lot of the nodes in the triangle between the three cities. Instead, what if we start up the search at all the locations simultaneously? Growing them out until the first two connect, and then continuing all until the third location connects somewhere with the first two. That would avoid the repeated node visits. The question is, can we modify this uninformed tridirectional search into a tridirectional a star search? The key is figuring out an admissible heuristic that takes into account two potential goals at the same time. Give it some thought, see if you can come up with one

50 – Peter's take on AI – lang_en_vs52.srt (2. Search)

.....

I've been asking this question of all the AI researchers I interview. What is your definition of artificial intelligence? To me, AI is programming a computer to do the right thing when you don't know for sure what the right thing is. So Peter, you've written the most widely used AI textbook in history. Can you tell us about that process and what led to its success? Well, around 1990, I remember the AI faculty at Berkeley, we all went to lunch together. And we were lamenting that none of the textbooks available at that time were quite what we wanted. And we should write a new text, we said, but it didn't happen. I ended up leaving Berkeley, and I went into industry, first with Sun Microsystems. About a year later, I saw Stuart Russell at a conference. And I said, well, you guys must be almost done with that book you were always talking about. And he said, no, they hadn't done anything. So I said, well, the two of us should just go ahead and do it, and we did. I think the book was successful because of a fortuitous accident of timing. We happened to come along just as AI was making this transition from Boolean logic to probability, and from handcrafted knowledge to machine learning. And so we were able to capture that in our book in a way that wasn't in previous books. Your intro to AI course has had over 160,000 students, and led to the founding of Udacity. Why did you and Sebastian decide to do the course in the first place? Well, we taught the intro to AI course at Stanford in 2010, a regular course in the classroom. And then in winter 2011, I guess they couldn't come up with anyone else to teach the course, so they asked us to do it again, and we agreed. But since we had just done it, we decided we wanted to do something different this time. And Sebastian suggested opening the course up to the world. I thought, that's a great idea. Maybe we might get 1,000 students to sign up. Well, I was off by two orders of magnitude. Wow. What lessons did you learn from being the head of search quality for Google in the early days? I think one important lesson is, pay attention to the data. Understand what's out there, what you're doing, and how well you're doing. Keep experimenting and measuring how well you're doing, and that way, you can improve. And two, pay attention to people. I think we did a good job at building a high-quality search engine. But it all comes down to having great people and keeping them happy and productive. So Peter, we both love Lisp as a programming language. What do you think that makes it special? I would say that today, Lisp is no longer really special. But to me, that means that Lisp won. It influenced all these other languages into adopting the key features. What things are they? Well, a rich collection of data types, a read-eval-print loop, the notion of debugging a running program and being able to easily inspect the data. Preference for lots of functions, many without side effects, object orientation and inheritance if you want it. All these things were unique to Lisp, but now they're in most languages. I think the one thing that hasn't caught on in other languages was the ability to define rich macros. Some languages have a limited macro capability. Lisp really went much farther. But that hasn't moved into current date languages yet. So when we first talked about my preparing this course, you convinced me to use Python for the assignments, a language I didn't know at the time. Why do you think Python is good for teaching this course?

Well, when Stuart and I wrote the book, we didn't want to impose a programming language choice on the reader or the instructor. And so we just used a pseudo-code in the algorithms in the book. We made up a format, and then we provided Lisp implementations for the code online. Over time, fewer and fewer students knew Lisp, so I decided I would have to rewrite the code in a language. And I wanted one that would be as close to the pseudo-code as possible, popular enough that many students would know it. And easy enough so that if a student didn't know it, they'd still be able to understand it and pick it up quickly. And so I surveyed all the languages, and Python seemed to be by far the best choice. Somehow, Guido and we were in sync. And we thought of the same pseudo-code, more or less, as he thought of as a language. And so that worked out well. Cool, thank you.

3. SIMULATED ANNEALING

1 – Travelling Salesman Problem – lang_en_vs52.srt (3. Simulated Annealing)

.....

You've heard me say, do the stupid thing first and only add intelligence when necessary. It turns out that there are a whole class of problems where just adding a little bit of intelligence and iteratively improving the solution gets you very close to an optimal solution. Like what? One of the classic ones is the traveling salesman problem. Imagine you're a salesman. That might be nice. I'd probably make more money than what you're paying me. [SOUND] As I was saying, imagine you're a salesman and you have five cities you want to visit. You can start in any city, but you have to fly to all of them before your tour is over. And you have to come back to the starting city in the end. What is the most efficient order of flights to minimize overall distance flown. This problem sounds NP hard. What do you mean? Well, NP actually means non-deterministic polynomial time. But people commonly talk about non-polynomial algorithm as all being about as hard as each other or NP hard. This problem seems exponentially difficult in the number of cities considered and so it seems just as bad as our game playing or search problems. Precisely. But like with our other problems, we're going to figure out tricks to help solve the problem efficiently. First we're going to connect the cities randomly. How about like this? Great. Next we are going to look at any place where the paths cross. We'll show them in red here. I get it. We need to revise the figure to uncross each situation. That should reduce the distance traveled. Yep, and for large problems we can do the process iteratively until there are no crossed paths. You can do this simple process with thousands of cities and get a result that is within 1% of the optimal solution. But that seems like a hack. How does this technique generalize? That is the subject of this section, it will improve in algorithms. Let's start out with our challenge question.

2 – 4-Queens – lang_en_vs51.srt (3. Simulated Annealing)

.....

Can we get another example of an iterative improvement problem? Sure. Here's one we'll use throughout this lesson. It's called N Queens. Basically, the puzzle is to place n queens so they cannot attack each other on an n by n chess board. In other words, no queen can be on the same horizontal row, vertical column or diagonal. Okay, so here's an example board of 4 queens. I just put down the queens randomly and there are five ways that the queens could attack each other. So our goal is to get the number of attacks down to 0? Yep, and we have four dimensions in which to move. Specifically, we can move queen in its column to minimize the number of attacks. Well, there are two queens that have three attacks. It seems like a good idea to move one of those first. If we move the second queen to the top row, we could eliminate three of the five attacks. And that leaves only two attacks left. It seems like we should always start from the queen with the most attacks. This reminds me of the most constrained variable heuristic in constraint satisfaction problems, but we'll cover that topic in the next lesson. Yep, and taking that

approach, the next queen we should work on is the third one which is involved in two attacks. If we move it to the bottom row, then we will have no more attacks and we've solved the problem. That's surprisingly effective. We've solved the problem in just two moves, whereas with simple search, we would have had many more steps. Exactly. It is an example of trying the stupid thing first and then adding intelligence until we solve the problem. However, 4 queens seemed too easy. Let's try 5 queens and see if it gets harder.

3 – 5-Queens Quiz – lang_en_vs52.srt (3. Simulated Annealing)

.....

Can we solve the 5-queens problem? Check the boxes for spaces that will contain a queen such that no two are attacking each other. There are multiple solutions here, so just put down one.

4 – 5-Queens Quiz Solution – lang_en_vs52.srt (3. Simulated Annealing)

.....

Here is one possible solution. [BLANK_AUDIO]

5 – n-Queens Heuristic Function – lang_en_vs52.srt (3. Simulated Annealing)

.....

Let's be a little more formal about the n-Queens problems. Given a current board we want to make the one move that can most greatly improve the situation. In fact to simplify the discussion let's constrain ourselves to moving a single queen up or down within its column. Then we keep iterating until we reach our goal of having zero attacks. Given this board which currently has 17 attacking pairs. The best we can do with a single move is to get the number of attacks down to 12. For example, if you move this queen here, you reduce the number of conflicts to 12. That's right. But, how do we figure out which one to take? Do we just select it randomly? Seemed as good an idea as any. Okay so we can continue to iterate, hopefully reducing the number of attacks with each move. But what if we get into a situation where no move decreases our number of attacks?

6 – n-Queens Local Minima – lang_en_vs52.srt (3. Simulated Annealing)

.....

You mean like this board? Hold on, I have to think about this one for a second. Okay, I see that there's only one current attack on this diagonal. So when I move one of these queens I should be done. Sure, why not try it out? Okay, well let's work with the queen in the fourth column first. I can't put it in the topmost row, because it gets attacked by the same queen as well as its diagonal neighbor. With the next row down, it's attacked both here and here. The next row down is attacked both here and here. Hold it, there is no place on this board I can put it without being attacked horizontally. Except where I had it originally, in fact, every possible spot makes the problem worse. We'll cause more attacks. Is there anything else we could do? We'll have the same problem with the queen in the second to the last column. Moving it down one row leads to two attacks, here and here. Going to the next row causes two attacks, as well. Every single move with this queen gets more attacks, as well. And moving any of the other six queens will also cause an increase of attacks because they will be attacked horizontally, plus we will still have the original attack. For example, look at this left most queen, every row, except for the one it's in, currently will have another attack. Does that mean the problem is unsolvable? No, I know the problem is solvable, but I seem to be stuck. No one move will improve my situation, this puzzle hurts my brain. Maybe we can use a simpler problem to figure out a solution and then come back to this one later. That's a great idea.

7 – Hill Climbing – lang_en_vs51.srt (3. Simulated Annealing)

.....

The n queens problem was multidimensional. And we were trying to minimize the number of attacks. Here for convenience we'll make our goal to maximize this value. And we can only move left or right along the x-axis instead of having so many different pieces to move.

8 – Local Maximum – lang_en_vs52.srt (3. Simulated Annealing)

.....

Suppose I start here. I look to the left and it goes down. I look to the right and it's going up. So I'll take a step in that direction. And what do you do when you reach the top? Well we will get stuck, just like we did with the end queens problem. There's no immediate step that I can do that has a positive gradient. Everything is negative, that's a bummer. Is that a problem, didn't we want to get to a maximum? We actually wanted to get to the global maximum. Looking at the graph, we can see that there's actually a taller peak over here. I see, but our computer agent doesn't know that. It can only see what's one step away from it, right? So how are you going to get unstuck? But what if I just start again somewhere else?

9 – Random Restart – lang_en_vs52.srt (3. Simulated Annealing)

.....

This time I start here and I go to the left because that is the positive gradient. And I'm going to get to the top of the global maximum. But what's the chance that you're going to get to global maximum on your second attempt? Better than just with my first attempt, but I see what you mean. We're computer scientists and we have computer power to burn. We can just do lots of random restarts, hill climb until we reach the local peak, and then take the maximum of all the iterations. How do you know you've done enough examples? Well, let's just do lots. It's inefficient, but as our number of samples goes to infinity it should work. Well, you could be more efficient and keep track of the places on the graph you've been before, and restart the sample whenever you see you've gotten to the same place. That trick is called a taboo search, as in it's taboo to go somewhere you've already been. Well I could also just keep a list of all the local maxima and try to get an idea of the general shape of the problem, and try to predict where a new maximum might be, given the areas I haven't explored yet. That sounds like a variant of the stage algorithm. So what you're saying is that random restart seems like a good idea in general, given that there are all these improvements that have been made. Yup, let's try a quiz on what we've learned so far.

10 – Hill Climbing Quiz – lang_en_vs51.srt (3. Simulated Annealing)

.....

We are going to consider multiple starting points for the hill climbing algorithm in this quiz. By convention, we'll call these particles. For each of these particles, tell us their value assuming your algorithm has a step size of one and that it stops when no positive gradient is found.

11 – Hill Climbing Quiz Solution – lang_en_vs52.srt (3. Simulated Annealing)

.....

Particle 1 starts at $x=1$ and follows a positive gradient up, but then it gets stuck at the shoulder where $y=2$. Particle 2 starts at $x=8$ and follows a positive gradient up, but it gets stuck at this local maximum at $y=6$. Particle 3 starts at $x=9$ and follows the positive gradient in the right direction, but again it gets stuck at a local maximum, at $y=8$. Particle 4 starts at $x=14$ and also goes to the same local maximum. Particle 5 starts at $x=20$ and follows the positive gradient up towards the global maximum at $y=12$.

12 – Step Size Too Small – lang_en_vs52.srt (3. Simulated Annealing)

.....

Does hill climbing have any other problems we should worry about? Suppose we start at the left edge of the graph, using small steps we gradually climb higher until we get to the shoulder here. However, if we stop when we no longer see a way to improve we can get stuck at this shoulder. So you mean we get to this point and if the step size is small enough we never see the sudden increase in gradient here? Yup. For flat areas how do we know which way to travel? If we just choose a direction randomly we can wander on this plateau for a while. The code needs to return a result at some point. So how many times in a row should we allow a zero improvement in score before we stop? We could just keep going in the direction of the last positive gradient. But with a small enough step size, we can easily think we are not improving and the algorithm should stop.

13 – Step Size Too Large – lang_en_vs52.srt (3. Simulated Annealing)

.....

Well, then why not just keep the step size large? A couple of reasons. First, with a really large step size, we can miss sharp hills completely. Going back to our first starting place if my step size is this big, I would skip the hill I intended to take and would start going up the next hill instead. Well, that seems obvious. But you said there were a couple of reasons, what's the other one? In certain situations, the algorithm could get into an infinite loop and never terminate. For example, look at the hill with the global maximum. If I start to the left of it, above the shoulder, and have a step size this big, I'll skip to the other side of the hill. Now the gradient is going to go back the other way. So I step back in the direction in which I came. End up close to where I started. The algorithm can oscillate and not converge on the answer. But it should be easy to check for that in our code. Maybe if we see oscillation we could just do smaller steps. You've hit upon a key idea. We can start with a large step size and decrease it over time to better ensure that we reach the global maximum. However, we'll get the same result with something called simulated annealing which is one of the big concepts in this course. Before we continue I'd like to experiment a bit with these hill climbing ideas.

14 – Hill Climbing Quiz 2 – lang_en_vs52.srt (3. Simulated Annealing)

.....

Now assume your algorithm has a step size of 2. It still stops when no positive gradient is found. What value would these particles return? Particle 1 starts at x equal to 7. Particle 2 starts at x equal to 8, and particle 3 starts at x equal to 10. Select their appropriate answer for each particle from the choices given here.

15 – Hill Climbing Quiz 2 Solution – lang_en_vs52.srt (3. Simulated Annealing)

.....

Particle 1 starts at 7, and with a step size of two, it reaches the peak at 10 with one step. Particle 2 starts at 8, it goes towards the peak, but then it oscillates around the peak because the step size is too large. Particle 3 starts at 10, it follows this upward gradient, but then it gets stuck at this plateau.

16 – Annealing – lang_en_vs52.srt (3. Simulated Annealing)

.....

We've been trying to emphasise algorithms that are of a particular importance. A star and alpha-beta pruning on the minimax algorithm are some examples. Now we're going to introduce another key concept, simulated annealing. Hold on. Simulated annealing? That implies that there's real annealing too. That's right. We're going to steal some ideas from the physicists for this section. First let's talk about energy minimization. When external conditions allow molecules to be mobile, and then mobility of the molecules slowly reduces, the molecules then arrange themselves into the lowest energy configuration. Often these conditions result in regular patterns. You mean like mud cracks. Yep, that's a common example where the decreasing amount of water in the mud reduces the molecules and mobility over time. And the mud cracks into regular patterns. And we see similar structures in honeycombs.

Honeybees try to optimize their storage space and minimize the building materials for the structure that their building. That's an example of minimization done by design. But the same effect can happen in the earth as erupted lava slowly cools and rocks form. Here's an example of columnar basalt from the Devils Postpile in California. As the rock cools it shrinks and cracks into hexagonal lattice which is a minimal energy configuration. Iron molecules have similar low energy states into which they can pack. They can form a lattice with other molecules like carbon. Some of these lattices are harder or softer than others. For certain applications like sword making a mixture of hardness and ductility is needed. Sword makers realize the need to heat and cool iron repeatedly to get iron molecules to align in the desired lattice structures to make swords hard but not brittle. They heated the iron above the temperature where the atoms could move about and form new structures with carbon and other atoms. Then they cooled the iron to preserve the types of lattice structures they wanted. This annealing process was repeated according to closely guarded formulas and times to get the desired properties. The best steel-makers were treasured. Looking at the edge of the blade of a finely made sword or cooking knife shows a process of heating and recrystallization. But how does annealing help our problem? We are going to use the idea of heating and cooling to help us get out of the local minima on the way to finding global minimum. For us, high temperature equates to more randomness. And gradual cooling will decrease the randomness until we converge on a solution.

17 – Simulated Annealing – lang_en_vs51.srt (3. Simulated Annealing)

.....

Here is the version of the algorithm we're going to use. Just like hill climbing, we're going to iterate with simulated annealing, looking for points close to our current position that might have an improved value. However, we're going to select our next position randomly from the points in the region near us. If a new position is better than our current position, we're going to take it. However, if it isn't better, we're still going to take it with a probability of $e^{-\frac{\Delta E}{T}}$, where T is our temperature. So we start with a high T , do we change it later on? Yes, in the algorithm we have a schedule for what the temperature should be. And we'll start with T being very high. And when T is high, say close to infinity, ΔE over T goes to 0. No matter what the ΔE is, even if it's negative. So e^{-0} is 1. So in the beginning, we have a lot of random motion as we take all the random positions offered to us. No matter how bad the new position is. I see, just like in real annealing, when the temperature is very high, the particle's jumping around a lot in the beginning. So that if the particle gets stuck in a local maximum instead, it has the ability to leave that peak, ignoring that it is going the wrong way and instead end up at a different part of the graph. Because the temperature is high, the next point that's fixed randomly can make it move down the slope. Given enough time the randomness will ensure we get off this particular hill and hit the hill with the global maximum instead. Precisely. But how does it converge? Won't we continue doing this random walk forever? Well, let's first take a look at the opposite situation where T is near 0. We never want T to actually be 0 because that will give us an undefined answer. So for illustration purposes let's say T is equal to 0.01. We already said that if the purported new random position improves our score, we're going to take it. If it stays the same ΔE is 0. Or if it gets worse, ΔE is negative. Let's say ΔE here is -1. Now we have $e^{-\frac{-1}{0.01}}$. Which is the same as e^{-100} . Which is a very small number. So we have almost no chance of taking that suggested new random position. Instead, we'll keep generating new random positions until we get one that improves e . Then, when T is small, we basically have normal hill climbing. Correct. So we'll just slowly change T from very large, where we're going to move over the graph randomly, to very small, where we climb to the nearest peak. Yep. And if we happen to get stuck at a plateau along the way, like here, ΔE is 0. But that makes the equation e^{-0} again. So that the algorithm will take the new random position instead. That will happen no matter what T is. And eventually, we'll random walk off the plateau, back to someplace where there's a positive gradient and continue going up to the maximum. Exactly, the great thing about simulated annealing is that it's guaranteed to converge to the global maximum if we start T high and decrease it slowly enough.

18 – Simulated Simulated Annealing – lang_en_vs52.srt (3. Simulated Annealing)

.....

One of my undergraduate students, Juliet, has created a nice simulation of simulated kneeling using Raptor prototyping. What does it do? Well we put in a ball that represents our particle, and we shake the box really hard. [SOUND] And as you slowly decrease the amount of shaking, the ball falls into the deepest well and stays there. But I have to be careful, to slowly decrease the temperature, or else it will not work. You all seem to work very hard for few seconds of demonstration. There's definitely a theme in my research.

19 – Local Beam Search – lang_en_vs51.srt (3. Simulated Annealing)

.....

While we're on this topic, I'd like to talk about local beam search, because we'll use it later in the course. Okay, it's your show after all. The local beam search, instead of just using one position, which I'll call a particle, just because that's how I think of it, will keep track of k particles. At each time frame, we'll look at the randomly generated neighbors of each of these particles and keep the k best ones for the next iteration. If ever any of these particles have reached a goal, we terminate. Now you'd think that this algorithm is just like random restart. But it's not, because we are comparing all the neighbors of all the particles to each other, there is information being passed between each position, right? Normal random restart doesn't share any of this information between iterations. That's correct. Okay, that seems fine. But I've heard you talk about stochastic beam search before as being more useful. How is that different? Stochastic beam search is the same thing. But the successors are chosen not just based on their fitness, but some randomness. Which helps ensure we don't get stuck in a local maximum. This idea has some similarity to simulated annealing. As in the class of algorithms to which generic algorithms belongs, which we'll tackle next.

20 – Representing n-Queens – lang_en_vs52.srt (3. Simulated Annealing)

.....

We're going to use the n-Queens problem again to talk about genetic algorithms, but before we do that, we need to create a convention to represent a given board. We'll use the one in the book. Since there can only be one queen in each column, we can encode a board with the number per column that indicates where the queen is. So for this board, the encoding would be 86427531.

21 – 8-Queens Representation – lang_en_vs51.srt (3. Simulated Annealing)

.....

Here's another example board for 8-Queens. Give us a string that represents the position of each piece on this board.

22 – 8-Queens Representation Solution – lang_en_vs52.srt (3. Simulated Annealing)

.....

Here is the answer. [BLANK_AUDIO]

23 – Genetic Algorithms – lang_en_vs52.srt (3. Simulated Annealing)

.....

Another specific we need to know before doing our genetic algorithms example is that there are 28 plausible pairs of attacking queens on this eight by eight board. How did you figure that out? Well, we have eight queens and we want to examine every possible pair that could attack each other. So that it has 8 choose 2 or 8 factorial over 2 factorial times 2 factorial. That's good to know since our goal with n queens is to reduce the number of attacking pairs of queens to 0, we're going to find a fitness function for a board to be the maximum number of attacking pairs of queens, which is 28, minus the number of attacking queens for a given board. In the case of eight queens, when the fitness function reaches 28, we'll know we've won. Genetic algorithms is an analogy to natural selection

in biology. It uses breeding and mutation to find the optimal answer to a problem. I think it's best to explain genetic algorithms with an example. Okay, then let's work on the eight queens problem. To get things started, let's choose four random boards. These four boards represent our gene pool, and we're going to try and breed better boards by combining them in different patterns. How do we choose which ones to use? Easy, survival of the fittest. We'll evaluate each board according to the fitness function we just described. The top board only had 4 attacking pairs, which means it has a score of 28 minus 4, or 24. The next one had 5 attacking pairs, so its score is 23. The remaining two have scores of 20 and 11 respectively. Poor number 4. He will probably never get to have kids. Evolution can be cruel. Anyway, from these fitness scores, we will create proportional probabilities for how likely each one is to breed. So basically, we're going to add the four scores and normalize each one into a percentage. Yep. Okay, so 24 plus 23 plus 20 plus 11 is equal to 78. And $24/78 = 31\%$. So our most fit board has a 31% chance to be chosen as a parent. That's right, we'll continue to calculate these percentages for each board. And we get 29%, 26%, and 14%. Poor number four. Would you stop it? [SOUND] Okay, now we're going to select four parents to create four new children. To do that, we basically roll a hundred sided die to select the first parent. If it is 1 to 31, we select the first board, and if 32 to 60, the second, and so on. Say we roll a 55. So for the first parent, we get the second board. And we roll the 100 sided die again. And for the second parent, I get the first board. And we continue this process. The third parent will be the second board. Lucky board. And for the last parent, we get the third board. Poor- Don't say it! Okay. So for these lucky boards, they are going to produce offspring. How many children will we produce from each set of parents? Two. Is there any reason for that? Well, with genetic algorithms, there are many parameters we can change to try to optimize how quickly we converge to a good result. Let's stick with two here for convenience. Okay, so now that we know which parents we have and how many children we will produce, how do we make children? That's a process we call crossover. You see, we've selected a point along each pair of parents. We pick this point randomly, and then we create the first child by taking the first part of the first parent and then tack that on to the last part of the second parent. This process sounds more like assembling Frankenstein's monster from body parts than breeding, but okay. [SOUND] Then for the second child we take the first part of the second parent and add it to the last part of the first parent. We do the same process with the second set of parents here. It's alive! [LAUGH] ` So hopefully, at least one of the children will get the best attributes of the parents. It kind of reminds me of the movie Twins with Arnold Schwarzenegger and Danny DeVito. You're showing your age. I wasn't born yet when that movie came out. Still, it was a good movie. But I actually had a point. What if, like in the supposition in the movie, one child gets all the good aspects of the parents and the other one gets all the bad aspects? Well, then in the next generation, the one with less attacking queens will have a higher fitness function and have a higher chance to create children. And the one with more attacking queens will have a lower fitness function and less chance to create children. And we hope that by having enough generations, we will eventually evolve an eight queens game board that solves the problem? There's one more critical step. But first, let's review crossover more carefully.

24 – GA Crossover – lang_en_vs52.srt (3. Simulated Annealing)

.....

Okay, let me handle this one. Given a parent, 32752411, which corresponds to a board that looks like this, we've randomly selected the spot between the third and fourth column to be the crossover point. For the first child, we'll take the first part of the first parent, which we'll mark in blue. Looking at the second parent, which is 24748552, we're going to take the last part of it. Let's mark that in yellow. And now, we'll add them together to get a child that is 32748552. That makes more sense to me now that I see it done with the actual game boards.

25 – GA Mutation – lang_en_vs51.srt (3. Simulated Annealing)

.....

But there is one problem. What if there is a critical part of the solution that is in none of the pairings? That could be an issue. I can also imagine that a critical piece might be in a board that never gets selected as a parent. Like poor number four here. And that critical piece gets bred out in the early stages and never comes back. How do we handle it? More randomness. How so? Just like we have mutations in biology, we're going to use mutations in genetic algorithms. For each digit I'm going to have a small but significant chance that the digit will mutate into some other digit. In the first child the 5 changes into a 1. The second child does not have any mutations, and the third and fourth children each have one mutation. So this mutation step is like occasionally choosing a random direction and simulated annealing. Where the randomness is stochastic beam search. Exactly. And given enough generations and mutations we expect to get the desired answer. Yup. And one of the things often reported in genetic algorithms papers is how many generations are needed until we get a good answer. Okay, now that we've seen all the components of genetic algorithms it seems like a good time for a quiz.

26 – GA Crossover Quiz – lang_en_vs52.srt (3. Simulated Annealing)

.....

Try to apply what you've learned so far to complete this iteration of the genetic algorithm. First, fill in the fitness value or the number of non attacking pairs of queens for each board state in the initial population. Then, choose parents according to their fitness score and fill them in this column. Order them by fitness value where the best is on top. Now cross over the top two selected parents, making sure that they're different individuals. Then produce two children using a four, four split. Finally, fill in the fitness values for the resulting children.

27 – GA Crossover Quiz Solution – lang_en_vs52.srt (3. Simulated Annealing)

.....

Here's the answer. Note that the children are worse than some of the parents. In progressive generations, we might continue to see this. But with mutation step, we might get closer to the goal. Without the mutation step, we run the risk of never actually reaching the goal.

28 – Method Similarities – lang_en_vs52.srt (3. Simulated Annealing)

.....

Shelly, is there any idea of reducing the randomness of a time in generic algorithm like say in simulated? Well, not limitations that we have shown so far. But the fitness function naturally reduces the randomness as each generation gets closer to the solution. However, we could argue that the method could do better. In fact, there are a lot of publications that talk about tuning, cross over, mutations, number of parents and all the other parameters to optimize genetic algorithms, to converge as quickly as possible. In some sense, genetic algorithms are a fancy version of beam search that happens to make a nice analogy with biology. Yep, some people call genetic algorithms the second best solution to any problem. But analogies are nice things when learning and using different methods. We used a lot throughout this section. Hill climbing, a kneeling and genetic algorithms are all analogies to some physical or biological process.

(1. Game Playing)

29 – Challenge Question Revisited – lang_en_vs52.srt (3. Simulated Annealing)

.....

Let's revisit the challenge question from the beginning of this section. We asked about which methods enable us to find the maximum point in this graph. We show that we can have problems getting stuck in local maxima or on plateaus where there is not enough local information to determine which way to go to improve the answer, or determine if we have reached the maximum. We used randomness in three methods to get out of this trap. They included random restart, where we start many particles at random positions on the graph, hill climb, and take the

best results. Another solution was simulated annealing, where we make an analogy to the process of gradually cooling the temperature of our particle to make it move in progressively less random ways while hill climbing. Until we converge on the global maximum. Finally, we said genetic algorithms could help us. This is where we select positions based on their fitness to breed children that mutate, but eventually converge on the solution. We talked about how a random restart, simulated annealing, and genetic algorithms are combined into the ideas to cast a theme search. Where multiple particles are released on the graph, and their children generate progressively less randomly, eventually converge to the maximum value. We'll see these ideas repeatedly in AI, in techniques like particle filters, pattern recognition with hidden Markov models, and Monte Carlo Markov chains. Keep these ideas in mind whenever you're creating your own algorithms or attacking new problems. Sometimes letting go and embracing randomness is the key to the solution.

4. CONSTRAINT SATISFACTION

1 – Introduction – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Here at Atlanta we are homes the world's busiest airport. They have five runways and 2,500 arrivals and departures per day. Have you ever wondered how they schedule all those flights? Those runways are putting out a plane every 30 seconds. Wow, it turns out this is an example of a classic AI problem called constraint satisfaction. Let's start out with a simple example as a challenge question.

2 – Map Coloring – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Map coloring is another classic example of constraint satisfaction. We are trying to color this map of Australia, using the minimum number of colors as possible. But making sure that no neighboring regions have the same color. I guess if they had the same color, it would be hard to tell where one territory stopped and the next began. Exactly, we can set up the problem more formally by first listing all the variables, in this case the territories. And the possible colors of the territories are listed as the domains of the problem. Finally, the rules of the problem, that no territory can be assigned the same color as a neighboring territory, are listed as the constraints. We can do that explicitly by listing all the possible pairwise constraints. Like how Western Australia's color can't be the same as the Northern Territory. Or we could list all the allowable assignments for each pair of territories. A solution is an assignment of colors to each of the territories that satisfies all the constraints. I always wanted to paint Australia red.

3 – Constraint Graph – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Anyway, unary constraints restrict the value of a given variable, like saying Tasmania can't be purple. Binary constraints relate, at most, two variables. We can represent those with a constraint graph. Here's a constraint graph for our Australia coloring problem. Nodes are the variables, like Western Australia, South Australia, Queensland, etc. Arcs show the constraints between the variables. We can apply general-purpose constraint satisfaction problem algorithms to the graph structure to speed up our search for an answer. For example, Tasmania is independent from the rest of the problem.

4 – Map Coloring Quiz – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Now try this map coloring problem. Color each region by typing in a number to represent your color. For example type the number one to represent color number one, two for color number two, and so on. What is the minimum number of colors needed for this map?

5 – Map Coloring Quiz Solution – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Here is one possible set of answers. Answers could vary depending on where you started coloring in the map. We had four colors, at minimum, to color in this map because of all the adjacent regions in the center.

6 – CSP Examples – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Besides unary and binary constraints, we could have constraints that involve 3 or more variables or soft constraints like the fact that I prefer to teach on a Tuesday/Thursday schedule. Problems with preference constraints are called constraint optimization problems and are with solving linear programming. There are lots of examples of constraint satisfaction problems. Sudoku is a classic game which can be solved with any of the methods we will be teaching. Other examples include car assembly, job scheduling in factories, class scheduling, spreadsheets, transportation scheduling or floor planning. Many of these require real-valued variables.

7 – Constraint Hypergraph – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Let's look at an example of cryptarithmic, the classic two + two = four problem. Here, we have unary constraints, which we'll represent by square boxes on a constraint hypergraph. For example, there is a global constraint, that none of the letters can represent the same digit. And another is, that $O+O$ has to equal R except in the case that there is a carry to the next column, which we'll represent as a new variable, x_1 , which could be 0 or 1. We can write that as the constraint, $O+O=R+10 \times x_1$ and show it on our constrain hypergraph, like this. We'll revisit this problem again later.

8 – Backtracking Search – lang_en_vs51.srt (4. Constraint Satisfaction Subtitles)

.....

So which algorithm are we going to show first to solve constraint satisfaction problems? Well the stupid one first of course. I had to ask. Then we'll add intelligence as we need it. Okay, we'll use simple search. States are defined by the values assigned so far. The initial state is the empty assignment. First we'll check to see if the current assignment is the goal. Then we will assign a value to unassigned variable that does not conflict with the current assignment. We call it a dead end if there are no legal assignments. In that case, we will back up to the previous date and try another assignment. And we will recurse until you either find the answer or try all the possible assignments and report failure. Let's make the algorithm concrete by doing an example. Okay, here's our map of Australia. We will work with Western Australia first. You can have one of three colors, orange, green, or blue. I'll choose orange. Okay, what section do you want us to look at next? How about the Northern Territory. Great, since we chose orange for Western Australia, the Northern Territory can have two colors, green or blue. Let's choose green. Okay, now let's work with Queensland next. It can be orange or blue. Let's choose blue. But now we are in trouble, there are no colors left for South Australia. Okay well, that's a dead end. Let's back up and try the other branch where we only used orange and green up to this point. And eventually we'll find the answer this way, but we can improve the efficiency of the algorithm significantly.

9 – Improving Backtracking Efficiency – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

I guess we've already seen one optimization. How so? If I choose the least concerning value for Queensland, we would not have had to backup. Let me show you. We started with Western Australia being orange, then the Northern Territory was green. Since Queensland is not next to Western Australia, I can use orange again. That will leave blue unused. So orange was the value that least constrained our future choices. Now you get it. Another optimization is to choose to assign the variable with the minimum remaining values next. If we start with assigning Western Australia and then the Northern Territory colors, the variable with the least number of values remaining is South Australia, so we should choose one that next to assign. What if I have a tie? What do you mean? In this example, we have assigned Southern Australia and Northern Territory. There are two regions that can only be assigned orange, Queensland and Western Australia. Which do we choose? Choose the one with the most constraints. You mean Queensland in this case because it borders New South Wales as well as the Northern Territory and Southern Australia. Western Australia only borders Northern Territory and Southern Australia, so it's not as complex. By choosing the one that's more constrained in the future, we'll be able to run into problems sooner rather than delaying them. Correct. Okay, how much do these improvements to backtracking help? Remember our End Queens problem? Yeah. It is another example of a constraint satisfaction problem. Okay. Using the least constraining value and minimum remaining values heuristics to improve backtracking search will allow us to do the 1,000 Queens problem. I don't expect I could ever do that by hand. That's impressive. You haven't seen anything yet.

10 – Backtracking Optimization Quiz – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Let's consider the optimizations we just discussed. For this half colored map, which region should we fill in next in order to minimize backtracking? There might be multiple answers here. But choose one of these optimizations and choose the region that matches with that.

11 – Backtracking Optimization Quiz Solution – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Here are two possible answers. This region is holding a green region and an orange region, so it only has two options for colors. Compared to all other regions, this is the minimum remaining values of those regions. For this region, we can pick the least constraining value which is green in this case. Because its bordering region can only take light blue or dark blue. So avoiding those two choices gives us more options later on.

12 – Forward Checking – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Our next optimization is forward checking. Basically, for each unassigned variable, we're going to keep track of the remaining legal values. I get it. So if ever make a decision that causes an unassigned variable to not be able to have a value, we stop our search and backup. Right. Let's try that with the Australia example. We start with nothing colored and all the regions could have any of the three colors. In our first step, we assign orange to Western Australia. That removes orange from the colors available to the Northern Territory and to Southern Australia. Suppose we assign green to Queensland. Now both the Northern Territory and Southern Australia can only have blue, and New South Wales can have orange or blue. If we assign blue to Victoria. Then we see that New South Wales can only have orange. The Northern Territory can only have blue and South Australia can't have anything. So we figured out that we are stuck and can backtrack earlier than we would have if we tried to continue assigning variables. Exactly, forward checking is an early warning system that a surge is going down the wrong branch.

13 – Constraint Propagation and Arc Consistency – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

However, I noticed we could've stop our search earlier. How? At this step we said the Northern Territory in South Australia both had to be blue. So? But this two regions are adjacent. We know they can't both be blue. I see, for checking propagates information from assigned to unassigned variables but doesn't provide early detection from all failures. Instead, we can use constraint propagation repeatedly to enforce all the local constraints. We can use Arc Consistency for a simple version of constraint propagation. A variable, and a constraint satisfaction problem is arc consistent, with respect to another variable, if there is some value still available for the second variable, after we assign a value to the first variable. If all variables in a graph satisfies this condition, then the network is arc consistent. Let's look at the practical example. Once we get to the stage of signing green to Queensland we look at of its neighbors that are unassigned and see if the assignment of green reduces the number colors available to them. If so, we remove from a neighboring region. And then look at its neighboring regions to see if we need to appropriate the change onward. We keep doing this process until it terminates or we determine that one of the regions does not have any colors left. So here, we have removed green from South Australia's list and we're now looking at its neighbors. New South Wales is a neighbor, and in order to be arc consistent, we will have to remove blue from its available colors. It can only be orange. Victoria is a neighbor of New South Wales so we look to see if the change affects it. In fact it does. We remove orange from Victoria's list of available colors. Our choice in green for Queensland has caused a chain of changes in possible values for our variables. Why don't we remove blue, while we're at it so Victoria doesn't conflict with South Australia? Well, we haven't gotten that far in our queue of things to check yet. Next, we're going to look at the Northern Territory because it has another neighbor that might be affected by South Australia's change. In fact, we see South Australia and the Northern Territory are both restricted to blue which is not allowed. So we return a failure and force the search to try another option for Queensland. Wow, that was a lot of work. But it can save a lot of unnecessary deep searching for more complicated problems.

14 – Constraint Propagation Quiz – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Here's a map where we've colored in two regions for you. Given this state in the map coloring, propagate through the constraints to make each variable arc consistent. Check the boxes to indicate which color is possible in each region. Finally, is the entire network arc consistent?

15 – Constraint Propagation Quiz Solution – lang_en_vs51.srt (4. Constraint Satisfaction Subtitles)

.....

Here's the answer. We have several islands on this map, so they can have any of the three colors. However, K2 is constrained to only one color, green, because of the two bordering regions. Finally, the entire network is arc consistent, because we don't have any regions without a possible color. [BLANK_AUDIO]

16 – Structured CSPs – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

.....

Speaking about saving time through deep searching another powerful trick is to look at the structure of the problem and see if we can decompose it to several independent problems. For example, Tasmania is independent of the rest of the Australia problem. We can solve it separately. Which takes a level off our search tree. In general, suppose we have a problem with 80 binary variables. We can divide it into 4 problems of 20 variables. We go from a search space of 2 to the 80th to 4 times 2 to the 20th. Even better, if we have a CSP where there are no loops, we can solve the problem in $O(N^2)$ squared time instead of $O(N^N)$ to the N. How? We pick any variable to be the root of the tree and choose an ordering of the variable such that each variable appeared after its parent in the tree. We then start at the

end and make each parent arc consistent, going up the tree until we get to the top. What if that is not possible? Then we report failure of the problem. Okay. What's next? It's simple. We start at the top of the tree and pick any assignment available. Going back down the tree until we get to the end. Since the trees aren't consistent any of the available assignments will solve the problem. How fast is this process? $O(N^2D)$ where N is the number of variables again and D is the size of the domain. So this trick really helps out with the speed of finding a solution. But what if the constrain graph really isn't a tree? Well, sometimes you condition a problem by assigning value to some variables and change the constrain graph into a tree. Like with our Australia coloring example. If we assign a value to South Australia first and remove that value from the possibilities of its neighbors, then the problem becomes a tree and we can use our fast method to solve the problem. Precisely.

17 – Iterative Algorithms – lang_en_vs52.srt (4. Constraint Satisfaction Subtitles)

Hold on, we said that N queen was a constraints satisfaction problem. When we cover that in the last lesson using generic algorithms or hill climbing algorithms to solve the problem. That's right, we can randomly assign values to the variable and iteratively improve the assignments until we come to the solution. Like our previous example with four queens, while there were four to the four, which equals 256 possible states, we can quickly find the solution by minimizing the number of attacks with each improvement. In fact, this minimizing conflict method can solve N queens with N equals 10 million. In the million queens case, this min conflicts algorithm can solve the problem in an average of 50 moves. Wow, I guess that's because there's a large number of solutions to the problem. Yep, iterative improvement algorithms work well when there are many solutions and when there are very few. However, there are problems with the number constraints versus the number of variables hit a certain ratio. And the problem becomes very hard to solve. Well, then let's stick to the easy ones.

18 – Challenge Question Revisited – lang_en_vs51.srt (4. Constraint Satisfaction Subtitles)

Now that we've been learning a few tricks, let's reexamine the challenge question. We'll use our constraint hypergraph for the problem where the top square represents the global constraint that all the letters have to be different. In addition, we know that $O + O$ has to be equal to $R + 10 \text{ times } X_1$, where X_1 represents the carry to the tens place and it can be 0 or 1. Similarly, $x_1 + W + W$ has to equal $U + 10 \text{ times } x_2$, where x_2 represents the carry to the hundredth place. Also, $x_2 + T + T$ has got to equal $0 + 10 \text{ times } x_3$, where x_3 is the carry to the thousandth place. And F has to equal X_3 . We also know that F can not be 0, because the problem does not allow leading 0's. So now we're going to use the strategy we learned of back tracking with forward checking and the MRV and least constraining value heuristics. There are a couple of ways to follow this procedure. And we'll show one here for illustration purposes. First, let's choose the X_3 variable. If the main is 0 and 1, choose the value 1 for X_3 . We can't choose 0, it wouldn't survive forward checking. Because it would cause F to be 0. The leading digit of the sum must be non 0. Choose F , because it has only one remaining value. Choose the value 1 for F . Now X_2 and X_1 are tied for minimum remaining values at 2. Let's choose X_2 , either values survives forward checking. Let's choose 0 for X_2 . Now X_1 has the minimum remaining values. Again, arbitrary choose 0 for the value for X_1 . The variable O must be an even number, because it is the sum of $T + T$ less than 5. Because $O + O = R + 10 \text{ times } 0$. That makes it most constrained. Arbitrarily choose 4 as the value for O . Now R has only one remaining value. Choose the value 8 for R . T now has only one remaining value. Choose a value, 7, for T . U must be an even number less than 9. Choose U , the only value for U that survives forward checking is 6. The only variable left is W . The only value left for W is 3. This is the solution. This puzzle is rather easy because it is under-constrained. It is not surprising that we arrive at a solution with no backtracking given that we are allowed to use forward checking. We could have used iterative improvement for this problem and probably would have converged quickly as it's under-constrained. But I find it difficult to keep all those numbers and letters in my head.

5. PROBABILITY

1 – Introducing Sebastian Thrun – lang_en_vs51.srt (5. Probability Subtitles)

.....

It is my great pleasure today to introduce Sebastian Thrun. For AI researchers, Sebastian is well-known for probabilistic robotics in autonomous vehicles. His group is the first to win the DARPA Grand Challenge on this topic. And he got even more fame from leading Google's self-driving cars. He's also a founder of Udacity and of Google X. Today, Sebastian's going to teach us about the fundamentals of probability. So Sebastian, why is probability important to AI researchers? It's one of the most fundamental concepts, and it has to do with the fact that we often don't know. So early artificial intelligence assumed you know everything, and they put all these words together. And these machines were pretty okay, but they couldn't learn anything. And it was only when people realized that there's an ability to learn that makes people so different that the machines became really, really powerful. And the fundamental basis for ignorance, for lack of knowledge, for uncertainty is probability. That's why probability matters for everything in artificial intelligence. So one of the things I really like about Sebastian's presentation of probability is how he goes from basics to dependency to Bayes' rule, and from there to Bayes' nets. It makes AI seem like an extension of probability. So as you're going through this section, make sure to really understand Bayes' rule because we'll be continuing to use it throughout the course.

3 – Intro To Probability And Bayes Nets.srt (5. Probability Subtitles)

.....

So the next units will be concerned with probabilities. Particularly with restructured probabilities using Bayes Networks. This is some of the most involved material in this class. And, since this is a Stanford level class, you'll find out that some of the quizzes are actually really hard. So, as you go through the material, I hope the hardness of the quizzes won't discourage you. It'll really entice you to take a piece of paper and a pen and work them out. Let me give you a flavor of a Bayes Network, using an example. Suppose you find in the morning that your car won't start. Well, there's many causes why your car might not start. One is that your battery is flat. Even for a flat battery, there's multiple causes. One, it's just plain dead. And one is that the battery is okay, but it's not charging. The reason why a battery might not charge is that the alternator may be broken. Or the fan belt might be broken. If you look at this influence diagram, also called a Bayes network, you'll find there's many different ways to explain that the car won't start. And a natural question you might have is, can we diagnose the problem? One diagnostic tool is a battery meter. Which may increase or decrease your belief that the battery may cause your car failure. You might also know your battery age. Older batteries tend to go dead more often. And there's many other ways to look at reasons why the car might not start. You might inspect the lights, the oil light, the gas gauge. You might even dip into the engine to see what the oil level is with a dipstick. And all of those relate to alternative reasons to why the car might not be starting. Like no oil, no gas, the fuel line might be blocked, or the starter may be broken. And all of these can influence your measurements, like the oil light or the gas gauge, in different ways. For example, the battery flat would have an effect on the lights, it might in fact have an effect on the oil light and on the gas gauge. But it won't really affect the oil you measure with the dipstick, that is affected by the actual oil level, which also affects the oil light. Gas will affect the gas gauge. And of course, without gas the car doesn't start. So this is a complicated structure that really describes one way to understand how a car doesn't start. A car is a complex system, there's lots of variables you can't really measure immediately. And it has sensors, which allow you to understand a little bit about the state of the car. What the Bayes Network does, it really assists you in reasoning from observable variables like the car won't start and the value of the dip stick, to hidden causes like is the fan belt

broken or is the battery dead? What you have here is a Bayes Network. A Bayes Network is composed of nodes, these nodes correspond to events that you might or might not know. They're typically called random variables. These nodes are linked by arcs, and the arcs suggest that a child of an arc is influenced by its parent. But not in a deterministic way, it might be influenced in a probabilistic way. Which means in order a battery, for example, has a higher chance of causing the battery to be dead. But it's not clear that every old battery is dead. There's a total of 16 variables in this Bayes Network. What the graph structure and associated probabilities specify is a huge probability distribution in the space of all of these 16 variables. If they are binary, which we will assume throughout this unit, they can take 2 to the 16 different values, which is a lot. The Bayes Network, as we find out, is a complex representation of a distribution over this very very large joint probability distribution of all these variables. Further, once we specify the Bayes Network, we can observe for example, the car won't start. You can observe things like the oil light, and the lights, and the battery meter. And then compute probabilities of hypothesis like the alternator is broken, or the fan belt is broken, or the battery is dead. So in this class, you're going to talk about how to construct this Bayes Network, what the semantics are. And how to reason in this Bayes Network to find out about variables you can't observe, like whether the fan belt is broken or not. That's an overview. Throughout this unit, I'm going to assume that every event is discrete, expect it's binary. We'll start with some consideration of basic probability. We work our way into some simple Bayes Networks. We'll talk about concepts like conditional independence. And then, define Bayes Networks more generally. Move into concepts like D-separation and start doing parameter counts. Later on, Peter will tell you about inference of Bayes Networks, so we won't do this in this class. I can't over emphasize how important this class is. Bayes Networks are used extensively in almost all fields of smart computer systems, in diagnostics, for prediction, for machine learning, in a field like finance, inside Google, in robotics. Bayes Networks are also the building blocks of more advanced AI techniques, such as particle filters, hidden Markov models, MDP's and POMDP's, Kalman filters, and many others. These are words that don't sound familiar quite yet, but as you go through the class, I can promise you you will get to know what they mean. So let's start now at the very, very basics.

Coin Flip – lang_en_vs1.srt (2 – Probability)

[Thrun] So let's talk about probabilities. Probabilities are the cornerstone of artificial intelligence. They are used to express uncertainty, and the management of uncertainty is really key to many, many things in AI such as machine learning and Bayes network inference and filtering and robotics and computer vision and so on. So I'm going to start with some very basic questions, and we're going to work our way up from there. Here is a coin. The coin can come up heads or tails, and my question is the following: Suppose the probability for heads is 0.5. What's the probability for it coming up tails?

4 – Coin Flip 2 – lang_en_vs1.srt (5. Probability Subtitles)

[Thrun] Let me ask my next quiz. Suppose the probability of heads is a quarter, 0.25. What's the probability of tail?

5 – Coin Flip 2 Solution – lang_en_vs1.srt (5. Probability Subtitles)

[Thrun] And the answer is 3/4. It's a loaded coin, and the reason is, well, each of them come up with a certain probability. The total of those is 1. The quarter is claimed by heads. Therefore, 3/4 remain for tail, which is the answer over here.

6 – Coin Flip 3 – lang_en_vs1.srt (5. Probability Subtitles)

[Thrun] Here's another quiz. What's the probability that the coin comes up heads, heads, heads, three times in a row, assuming that each one of those has a probability of a half and that these coin flips are independent?

7 – Coin Flip 3 Solution – lang_en_vs1.srt (5. Probability Subtitles)

.....

[Thrun] And the answer is 0.125. Each head has a probability of a half. We can multiply those probabilities because they are independent events, and that gives us 1 over 8 or 0.125.

8 – Coin Flip 4 – lang_en_vs1.srt (5. Probability Subtitles)

.....

[Thrun] Now let's flip the coin 4 times, and let's call X_i the result of the i -th coin flip. So each X_i is going to be drawn from heads or tail. What's the probability that all 4 of those flips give us the same result, no matter what it is, assuming that each one of those has identically an equally distributed probability of coming up heads of the half?

9 – Coin Flip 4 Solution – lang_en_vs1.srt (5. Probability Subtitles)

.....

[Thrun] And the answer is, well, there's 2 ways that we can achieve this. One is the all heads and one is all tails. You already know that 4 times heads is $1/16$, and we know that 4 times tail is also $1/16$. These are completely independent events. The probability of either one occurring is $1/16$ plus $1/16$, which is $1/8$, which is 0.125.

10 – Coin Flip 5 – lang_en_vs1.srt (5. Probability Subtitles)

.....

[Thrun] So here's another one. What's the probability that within the set of X_1 , X_2 , X_3 , and X_4 there are at least three heads?

11 – Coin Flip 5 Solution – lang_en_vs1.srt (5. Probability Subtitles)

.....

[Thrun] And the solution is let's look at different sequences in which head occurs at least 3 times. It could be head, head, head, head, in which it comes 4 times. It could be head, head, head, tail and so on, all the way to tail, head, head, head. There's 1, 2, 3, 4, 5 of those outcomes. Each of them has a 16th for probability, so it's 5 times a 16th, which is 0.3125.

12 – Dependence – lang_en_vs1.srt (5. Probability Subtitles)

.....

[Thrun] So let me ask you about dependence. Suppose we flip 2 coins. Our first coin is a fair coin, and we're going to denote the outcome by X_1 . So the chance of X_1 coming up heads is half. But now we branch into picking a coin based on the first outcome. So if the first outcome was heads, you pick a coin whose probability of coming up heads is going to be 0.9. The way I word this is by conditional probability, probability of the second coin flip coming up heads provided that or given that X_1 , the first coin flip, was heads, is 0.9. The first coin flip might also come up tails, in which case I pick a very different coin. In this case I pick a coin which with 0.8 probability will once again give me tails, conditioned on the first coin flip coming up tails. So my question for you is, what's the probability of the second coin flip coming up heads?

13 – Dependence Solution – lang_en_vs1.srt (5. Probability Subtitles)

.....

[Thrun] The answer is 0.55. The way to compute this is by the theorem of total probability. Probability of X_2 equals heads. There's 2 ways I can get to this outcome. One is via this path over here, and one is via this path over here. Let me just write both of them down. So first of all, it could be the probability of X_2 equals heads given that and I will assume X_1 was head already. Now I have to add the complementary event. Suppose X_1 came up tails. Then I can ask the question, what is the probability that X_2 comes up heads regardless, even though X_1 was tails? Plugging in the numbers gives us the following. This one over here is 0.9 times a half. The probability of tails is 0.8, thereby my head probability becomes 1 minus 0.8, which is 0.2. Adding all of this together gives me 0.45 plus 0.1, which is exactly 0.55.

14 – What We Learned – lang_en_vs1.srt (5. Probability Subtitles)

.....

So, we actually just learned some interesting lessons. The probability of any random variable Y can be written as probability of Y given that some other random variable X assumes value i times probability of X equals i , sums over all possible outcomes i for the (inaudible) variable X . This is called total probability. The second thing we learned has to do with negation of probabilities. We found that probability of not X given Y is 1 minus probability of X given Y . Now, you might be tempted to say “What about the probability of X given not Y ?” “Is this the same as 1 minus probability of X given Y ?” And the answer is absolutely no. That's not the case. If you condition on something that has a certain probability value, you can take the event you're looking at and negate this, but you can never negate your conditional variable and assume these values add up to 1.

15 – Weather – lang_en_vs1.srt (5. Probability Subtitles)

.....

We assume there is sometimes sunny days and sometimes rainy days, and on day 1, which we're going to call D_1 , the probability of sunny is 0.9. And then let's assume that a sunny day follows a sunny day with 0.8 chance, and a rainy day follows a sunny day with—well—

16 – Weather Solution – lang_en_vs3.srt (5. Probability Subtitles)

.....

Well, the correct answer is 0.2, which is a negation of this event over here.

17 – Weather 2 – lang_en_vs1.srt (5. Probability Subtitles)

.....

A sunny day follows a rainy day with 0.6 chance, and a rainy day follows a rainy day— please give me your number.

18 – Weather 2 Solution – lang_en_vs1.srt (5. Probability Subtitles)

.....

0.4

19 – Weather 3 – lang_en_vs1.srt (5. Probability Subtitles)

.....

So, what are the chances that D_2 is sunny? Suppose the same dynamics apply from D_2 to D_3 , so just replace D_3 over here with D_2 s over there. That means the transition probabilities from one day to the next remain the same. Tell me, what's the probability that D_3 is sunny?

20 – Weather 3 Solution – lang_en_vs1.srt (5. Probability Subtitles)

.....

So, the correct answer over here is 0.78, and over here it's 0.756. To get there, let's complete this one first. The probability of D2 = sunny. Well, we know there's a 0.9 chance it's sunny on D1, and then if it is sunny, we know it stays sunny with a 0.8 chance. So, we multiply these 2 things together, and we get 0.72. We know there's a 0.1 chance of it being rainy on day 1, which is the complement, but if it's rainy, we know it switches to sunny with 0.6 chance, so you multiply these 2 things, and you get 0.06. Adding those two up equals 0.78. Now, for the next day, we know our prior for sunny is 0.78. If it is sunny, it stays sunny with 0.8 probability. Multiplying these 2 things gives us 0.624. We know it's rainy with 0.2 chance, which is the complement of 0.78, but a 0.6 chance if it was (inaudible) sunny. But if you multiply those, 0.132. Adding those 2 things up gives us 0.756. So, to some extents, it's tedious to compute these values, but they can be perfectly computed, as shown here.

21 – Cancer – lang_en_vs1.srt (5. Probability Subtitles)

.....

Next example is a cancer example. Suppose there's a specific type of cancer which exists for 1% of the population. I'm going to write this as follows. You can probably tell me now what the probability of not having this cancer is.

22 – Cancer 2 – lang_en_vs1.srt (5. Probability Subtitles)

.....

And yes, the answer is 0.99. Let's assume there's a test for this cancer, which gives us probabilistically an answer whether we have this cancer or not. So, let's say the probability of a test being positive, as indicated by this + sign, given that we have cancer, is 0.9. The probability of the test coming out negative if we have the cancer is—you name it.

23 – Cancer 3 – lang_en_vs1.srt (5. Probability Subtitles)

.....

0.1, which is the difference between 1 and 0.9. Let's assume the probability of the test coming out positive given that we don't have this cancer is 0.2. In other words, the probability of the test correctly saying we don't have the cancer if we're cancer free is 0.8. Now, ultimately, I'd like to know what's the probability they have this cancer given they just received a single, positive test? Before I do this, please help me filling out some other probabilities that are actually important. Specifically, the joint probabilities. The probability of a positive test and having cancer. The probability of a negative test and having cancer, and this is not conditional anymore. It's now a joint probability. So, please give me those 4 values over here.

24 – Cancer 3 Solution – lang_en_vs1.srt (5. Probability Subtitles)

.....

And here the correct answer is 0.009, which is the product of your prior, 0.01, times the conditional, 0.9. Over here we get 0.001, the probability of our prior cancer times 0.1. Over here we get 0.198, the probability of not having cancer is 0.99 times still getting a positive reading, which is 0.2. And finally, we get 0.792, which is the probability of this guy over here, and this guy over here.

25 – Cancer 4 – lang_en_vs1.srt (5. Probability Subtitles)

.....

Now, our next quiz, I want you to fill in the probability of the cancer given that we just received a positive test.

26 – Cancer 4 Solution – lang_en_vs1.srt (5. Probability Subtitles)

.....

And the correct answer is 0.043. So, even though I received a positive test, my probability of having cancer is just 4.3%, which is not very much given that the test itself is quite sensitive. It really gives me a 0.8 chance of getting a negative result if I don't have cancer. It gives me a 0.9 chance of detecting cancer given that I have cancer. Now, what comes (inaudible) small? Well, let's just put all the cases together. You already know that we received a positive test. Therefore, this entry over here, and this entry over here are relevant. Now, the chance of having a positive test and having cancer is 0.009. Well, I might—when I receive a positive test—have cancer or not cancer, so we will just normalize by these 2 possible causes for the positive test, which is $0.009 + 0.198$. We know both these 2 things together gets 0.009 over 0.207, which is approximately 0.043. Now, the interesting thing in this equation is that the chances of having seen a positive test result in the absence of cancers are still much, much higher than the chance of seeing a positive result in the presence of cancer, and that's because our prior for cancer is so small in the population that it's just very unlikely to have cancer. So, the additional information of a positive test only erased my posterior probability to 0.043.

27 – Bayes Rule – lang_en_vs1.srt (5. Probability Subtitles)

.....

So, we've just learned about what's probably the most important piece of math for this class in statistics called Bayes Rule. It was invented by Reverend Thomas Bayes, who was a British mathematician and a Presbyterian minister in the 18th century. Bayes Rule is usually stated as follows: $P(A \text{ given } B)$ where B is the evidence and A is the variable we care about is $P(B \text{ given } A) \text{ times } P(A) \text{ over } P(B)$. This expression is called the likelihood. This is called the prior, and this is called marginal likelihood. The expression over here is called the posterior. The interesting thing here is the way the probabilities are reworded. Say we have evidence B . We know about B , but we really care about the variable A . So, for example, B is a test result. We don't care about the test result as much as we care about the fact whether we have cancer or not. This diagnostic reasoning—which is from evidence to its causes—is turned upside down by Bayes Rule into a causal reasoning, which is given—hypothetically, if we knew the cause, what would be the probability of the evidence we just observed. But to correct for this inversion, we have to multiply by the prior of the cause to be the case in the first place, in this case, having cancer or not, and divide it by the probability of the evidence, $P(B)$, which often is expanded using the theorem of total probability as follows. The probability of B is a sum over all probabilities of B conditional on A , lower caps a , times the probability of A equals lower caps a . This is total probability as we already encountered it. So, let's apply this to the cancer case and say we really care about whether you have cancer, which is our cause, conditioned on the evidence that is the result of this hidden cause, in this case, a positive test result. Let's just plug in the numbers. Our likelihood is the probability of seeing a positive test result given that you have cancer multiplied by the prior probability of having cancer over the probability of the positive test result, and that is—according to the tables we looked at before—0.9 times a prior of 0.01 over—now we're going to expand this right over here according to total probability which gives us 0.9 times 0.01. That's the probability of + given that we do have cancer. So, the probability of + given that we don't have cancer is 0.2, but the prior here is 0.99. So, if we plug in the numbers we know about, we get 0.009 over $0.009 + 0.198$. That is approximately 0.0434, which is the number we saw before.

6. BAYES NETS

1 – Introduction – lang_en_vs51.srt (6. Bayes Nets Subtitles)

.....

Sebastian, why are Bayes Nets important? They're one of the most amazing accomplishments of recent artificial intelligence. They take the idea of uncertainty and probability and marry it with efficient structures. So you don't have a big mumble jumble but you can really understand, like what uncertain variable influences other uncertain variable. A theory of the world using just Bayes Networks is really an amazing. Well, I've been teaching AI for many years, I've found that listening to Sebastian's lessons has really given me a better intuition for Bayes Nets. As you go through these lessons, make sure to understand how to construct a Bayes Nets and how to do inference with them. One of my favorite algorithms is Monte Carlo Markov Chain. Have fun with the lesson.

2 – Exercise: Bayes Network – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

So, if you want to draw Bayes rule graphically, we have a situation where we have an internal variable A, like whether I'm going to die of cancer, but we can't sense A. Instead, we have a second variable, called B, which is our test, and B is observable, but A isn't. This is a classical example of a Bayes network. The Bayes network is composed of 2 variables, A and B. We know the prior probability for A, and we know the conditional. A causes B—whether or not we have cancer, causes the test result to be positive or not, although there was some randomness involved. So, we know what the probability of B given the different values for A, and what we care about in this specific instance is called diagnostic reasoning, which is the inverse of the causal reasoning, the probability of A given B or similarly, probability of A given not B. This is our very first Bayes network, and the graphical representation of drawing 2 variables, A and B, connected with an arc that goes from A to B is the graphical representation of a distribution of 2 variables that are specified in the structure over here, which has a prior probability and has a conditional probability as shown over here. Now, I do have a quick quiz for you. How many parameters does it take to specify the entire joint probability within A and B, or differently, the entire Bayes network? I'm not looking for structural parameters that relate to the graph over here. I'm just looking for the numerical parameters of the underlying probabilities.

3 – Exercise: Bayes Network Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

And the answer is 3. It takes 1 parameter to specify P of A from which we can derive P of not A. It takes 2 parameters to specify P of B given A and P given not A, from which we can derive P not B given A and P of not B given not A. So, it's a total of 3 parameters for this Bayes network.

4 – Computing Bayes Rule – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

So, we just encountered our very first Bayes network and did a number of interesting calculations. Let's now talk about Bayes Rule and look into more complex Bayes networks. I will look at Bayes Rule again and make an observation that is really non-trivial. Here is Bayes Rule, and in practice, what we find is this term here is relatively easy to compute. It's just a product, whereas this term is really hard to compute. However, this term over here does not depend on what we assume for variable A. It's just the function of B. So, suppose for a moment we also care about the complementary event of not A given B, for which Bayes Rule unfolds as follows. Then we find that the normalizer, $P(B)$, is identical, whether we assume A on the left side or not A on the left side. We also know from prior work that $P(A)$ given B plus P of not A given B must be one because these are 2 complementary events. That allows us to compute Bayes Rule very differently by basically ignoring the normalizer, so here's how it goes. We compute $P(A)$ given B—and I want to call this prime, because it's not a real probability—to be just $P(B)$ given A times $P(A)$, which is the normalizer, so the denominator of the expression over here. We do the same thing with not

A. So, in both cases, we compute the posterior probability non-normalized by omitting the normalizer B. And then we can recover the original probabilities by normalizing based on those values over here, so the probability of A given B, the actual probability, is a normalizer, eta, times this non-normalized form over here. The same is true for the negation of A over here. And eta is just the normalizer that results by adding these 2 values over here together as shown over here, and dividing them for one. So, take a look at this for a moment. What we've done is we deferred the calculation of the normalizer over here by computing pseudo probabilities that are non-normalized. This made the calculation much easier, and when we were done with everything, we just folded it back into the normalizer based on the resulting pseudo probabilities and got the correct answer.

5 – Two Test Cancer – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

The reason why I gave you all this is because I want you to apply it now to a slightly more complicated problem, which is the 2-test cancer example. In this example, we again might have our unobservable cancer C, but now we're running 2 tests, test 1 and test 2. As before, the prior probability of cancer is 0.01. The probability of receiving a positive test result for either test is 0.9. The probability of getting a negative result given they're cancer free is 0.8. And from those, we were able to compute all the other probabilities, and we're just going to write them down over here. So, take a moment to just verify those. Now, let's assume both of my tests come back positive, so $T1 = +$ and $T2 = +$. What's the probability of cancer now written in short form probability of C given ++? I want you to tell me what that is, and this is a non-trivial question.

6 – Two Test Cancer Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

So, the correct answer is 0.1698 approximately, and to compute this, I used the trick I've shown you before. Let me write down the running count for cancer and for not cancer as I integrate the various multiplications in Bayes Rule. My prior for cancer was 0.01 and for non-cancer was 0.99. Then I get my first +, and the probability of a + given they have cancer is 0.9, and the same for non-cancer is 0.2. So, according to the non-normalized Bayes Rule, I now multiply these 2 things together to get my non-normalized probability of having cancer given the plus. Since multiplication is commutative, I can do the same thing again with my 2nd test result, 0.9 and 0.2, and I multiply all of these 3 things together to get my non-normalized probability P prime to be the following: 0.0081, if you multiply those things together, and 0.0396 if you multiply these facts together. And these are not a probability. If we add those for the 2 complementary of cancer/non-cancer, I get 0.0477. However, if I now divide, that is, I normalize those non-normalized probabilities over here by this factor over here, I actually get the correct posterior probability P of cancer given ++. And they look as follows: approximately 0.1698 and approximately 0.8301.

7 – Two Test Cancer 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Calculate for me the probability of cancer given that I received one positive and one negative test result. Please write your number into this box.

8 – Two Test Cancer 2 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

We apply the same trick as before where we use the exact same prior of 0.01. Our first + gives us the following factors: 0.9 and 0.2. And our minus gives us the probability 0.1 for a negative first test result given that we have cancer, and a 0.8 for the inverse of a negative result of not having cancer. We multiply those together. We get our non-normalized probability. And if we now normalize by the sum of those two things to turn this back into a

probability, we get 0.009 over the sum of those two things over here, and this is 0.0056 for the chance of having cancer and 0.9943 for the chance of being cancer free. And this adds up approximately to 1, and therefore, is a probability distribution.

9 – Conditional Independence – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

I want to use a few words of terminology. This, again, is a Bayes network, of which the hidden variable C causes the still stochastic test outcomes T1 and T2. And what is really important is that we assume not just that T1 and T2 are identically distributed. We use the same 0.9 for test 1 as we use for test 2, but we also assume that they are conditionally independent. We assumed that if God told us whether we actually had cancer or not, if we knew with absolute certainty the value of the variable C, that knowing anything about T1 would not help us make a statement about T2. Put differently, we assumed that the probability of T2 given C and T1 is the same as the probability of T2 given C. This is called conditional independence, which is given the value of the cancer variable C. If you knew this for a fact, then T2 would be independent of T1. It's conditionally independent because the independence only holds true if we actually know C, and it comes out of this diagram over here. If we look at this diagram, if you knew the variable C over here, then C separately causes T1 and T2. So, as a result, if you know C, whatever counted over here is kind of cut off causally from what happens over here. That causes these 2 variables to be conditionally independent. So, conditional independence is a really big thing in Bayes networks. Here's a Bayes network where A causes B and C, and for a Bayes network of this structure, we know that given A, B and C are independent. It's written as B conditionally independent of C given A. So, here's a question. Suppose we have conditional independence between B and C given A. Would that imply—and there's my question—that B and C are independent? So, suppose we don't know A. We don't know whether we have cancer, for example. What that means is that the test results individually are still independent of each other even if we don't know about the cancer situation. Please answer yes or no.

10 – Conditional Independence Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

And the correct answer is No Intuitively, getting a positive test result about cancer gives us information about whether you have cancer or not. So if you get a positive test result you're going to raise the probability of having cancer relative to the prior probability. With that increased probability we will predict that another test will with a higher likelihood give us a positive response than if we hadn't taken the previous test. That's really important to understand So that we understand it let me make you calculate those probabilities

11 – Conditional Independence 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Let me draw the cancer example again with two tests. Here's my cancer variable and then there's two conditionally independent tests T1 and T2. And as before let me assume that the prior probability of cancer is 0.01 What I want you to compute for me is the probability of the second test to be positive if we know that the first test was positive. So write this into the following box.

11 – Conditional Independence 2 – lang_en_vs52.srt (6. Bayes Nets Subtitles)

.....

So let me draw the cancer example again with two tests. Here's my cancer variable. And then there's two conditionally independent tests, T1 and T2. And as before, let me assume that the probability of cancer is 0.01. What I want you to compute for me is the probability of the second test to be positive if we know that the first test was positive. So write this into the following box.

12 – Conditional Independence 2 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

So, for this one, we want to apply total probability. This thing over here is the same as probability of test 2 to be positive, which I'm going to abbreviate with a +2 over here, conditioned on test 1 being positive and me having cancer times the probability of me having cancer given test 1 was positive plus the probability of test 2 being positive conditioned on test 1 being positive and me not having cancer times the probability of me not having cancer given that test 1 is positive. That's the same as the theorem of total probability, but now everything is conditioned on +1. Take a moment to verify this. Now, here I can plug in the numbers. You already calculated this one before, which is approximately 0.043, and this one over here is 1 minus that, which is 0.957 approximately. And this term over here now exploits conditional independence, which is given that I know C, knowledge of the first test gives me no more information about the second test. It only gives me information if C was unknown, as was the case over here. So, I can rewrite this thing over here as follows: P of +2 given that I have cancer. I can drop the +1, and the same is true over here. This is exploiting my conditional independence. I knew that P of +1 or +2 conditioned on C is the same as P of +2 conditioned on C and +1. I can now read those off my table over here, which is 0.9 times 0.043 plus 0.2, which is 1 minus 0.8 over here times 0.957, which gives me approximately 0.2301. So, that says if my first test comes in positive, I expect my second test to be positive with probably 0.2301. That's an increased probability to the default probability, which we calculated before, which is the probability of any test, test 2 come in as positive before was the normalizer of Bayes rule which was 0.207. So, my first test has a 20% chance of coming in positive. My second test, after seeing a positive test, has now an increased probability of about 23% of coming in positive.

13 – Absolute And Conditional – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

So, now we've learned about independence, and the corresponding Bayes network has 2 nodes. They're just not connected at all. And we learned about conditional independence, in which case we have a Bayes network that looks like this. Now I would like to know whether absolute independence implies conditional independence. True or false? And I'd also like to know whether conditional independence implies absolute independence. Again, true or false?

14 – Absolute And Conditional Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

And the answer is both of them are false. We already saw that conditional independence, as shown over here, doesn't give us absolute independence. So, for example, this is test #1 and test #2. You might or might not have cancer. Our first test gives us information about whether you have cancer or not. As a result, we've changed our prior probability for the second test to come in positive. That means that conditional independence does not imply absolute independence, which means this assumption here falls, and it also turns out that if you have absolute independence, things might not be conditionally independent for reasons that I can't quite explain so far, but that we will learn about next.

15 – Confounding Cause – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] For my next example, I will study a different type of a Bayes network. Before, we've seen networks of the following type, where a single hidden cause caused 2 different measurements. I now want to study a network that looks just like the opposite. We have 2 independent hidden causes, but they get confounded within a single observational variable. I would like to use the example of happiness. Suppose I can be happy or unhappy. What makes me happy is when the weather is sunny or if I get a raise in my job, which means I make more money. So

let's call this sunny, let's call this a raise, and call this happiness. Perhaps the probability of it being sunny is 0.7, probability of a raise is 0.01. And I will tell you that the probability of being happy is governed as follows. The probability of being happy given that both of these things occur– I got a raise and it is sunny—is 1. The probability of being happy given that it is not sunny and I still got a raise is 0.9. The probability of being happy given that it's sunny but I didn't get a raise is 0.7. And the probability of being happy given that it is neither sunny nor did I get a raise is 0.1. This is a perfectly fine specification of a probability distribution where 2 causes affect the variable down here, the happiness. So I'd like you to calculate for me the following questions. Probability of a raise given that it is sunny, according to this model. Please enter your answer over here.

16 – Independence – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] The answer is surprisingly simple. It is 0.01. How do I know this so fast? Well, if you look at this Bayes network, both the sunniness and the question whether I got a raise impact my happiness. But since I don't know anything about the happiness, there is no way that just the weather might implicate or impact whether I get a raise or not. In fact, it might be independently sunny, and I might independently get a raise at work. There is no mechanism of which these 2 things would co-occur. Therefore, the probability of a raise given that it's sunny is just the same as the probability of a raise given any weather, which is 0.01.

16 – Independence – lang_en_vs52.srt (6. Bayes Nets Subtitles)

.....

The answer is surprisingly simple. It is 0.01, but how do I know this so fast? Well, if we look at this Bayes network, both the sunniness and the question whether I got a raise, impact my happiness. But, since I don't know anything about the happiness, there's no way that just the weather might implicate or impact whether I get a raise or not. In fact, it might be independently sunny and might independently get a raise at work. There is no mechanism of which this two things would co-occur. Therefore, the probability of a raise given it a sunny is just the same as probability of a raise given any weather which is 0.01.

17 – Explaining Away – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] Let me talk about a really interesting special instance of Bayes net reasoning which is called explaining away. And I'll first give you the intuitive answer, then I'll wish you to compute probabilities for me that manifest the explain away effect in a Bayes network of this type. Explaining away means that if we know that we are happy, then sunny weather can explain away the cause of happiness. If I then also know that it's sunny, it becomes less likely that I received a raise. Let me put this differently. Suppose I'm a happy guy on a specific day and my wife asks me, "Sebastian, why are you so happy?" "Is it sunny, or did you get a raise?" If she then looks outside and sees it is sunny, then she might explain to herself, "Well, Sebastian is happy because it is sunny." "That makes it effectively less likely that he got a raise "because I could already explain his happiness by it being sunny." If she looks outside and it is rainy, that makes it more likely I got a raise, because the weather can't really explain my happiness. In other words, if we see a certain effect that could be caused by multiple causes, seeing one of those causes can explain away any other potential cause of this effect over here. So let me put this in numbers and ask you the challenging question of what's the probability of a raise given that I'm happy and it's sunny?

18 – Explaining Away Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] The answer is approximately 0.0142, and it is an exercise in expanding this term using Bayes' rule, using total probability, which I'll just do for you. Using Bayes' rule, you can transform this into $P(H|R,S)$ times $P(R|S)$ over $P(H|S)$. We observe the conditional independence of R and S to simplify this to just $P(R)$, and the denominator is expanded by folding in R and not R , $P(H|R,S)$ times $P(R|S)$ plus $P(H|\neg R,S)$ times $P(\neg R|S)$, which is total probability. We can now read off the numbers from the tables over here, which gives us 1×0.01 divided by this expression that is the same as the expression over here, so 0.01 plus this thing over here, which you can find over here to be 0.7 , times this guy over here, which is 1 minus the value over here, 0.99 , which gives us approximately 0.0142.

19 – Explaining Away 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] Now, to understand the explain away effect, you have to compare this to the probability of a raise given that we're just happy and we don't know anything about the weather. So let's do that exercise next. So my next quiz is, what's the probability of a raise given that all I know is that I'm happy and I don't know about the weather? This happens to be once again a pretty complicated question, so take your time.

20 – Explaining Away 2 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] So this is a difficult question. Let me compute an auxiliary variable, which is $P(H)$. That one is expanded by looking at the different conditions that can make us happy. $P(H|S,R)$ times $P(S,R)$, which is of course the product of those 2 because they are independent, plus $P(H|\neg S,R)$, probability of not as R plus $P(H|S,\neg R)$ times the probability of $P(S,\neg R)$ plus the last case, $P(H|\neg S,\neg R)$. So this just looks at the happiness under all 4 combinations of the variables that can lead to happiness. And you can plug those straight in. This one over here is 1, and this one over here is the product of S and R , which is 0.7×0.01 . And as you plug all of those in, you get as a result 0.5245. That's $P(H)$. Just take some time and do the math by going through these different cases using total probability, and you get this result. Armed with this number, the rest now becomes easy, which is we can use Bayes' rule to turn this around. $P(H|R)$ times $P(R)$ over $P(H)$. $P(R)$ we know from over here, the probability of a raise is 0.01. So the only thing we need to compute now is $P(H|R)$. And again, we apply total probability. Let me just do this over here. We can factor $P(H|R)$ as $P(H|R,S)$, sunny, times probability of sunny plus $P(H|R,\neg S)$ times the probability of not sunny. And if you plug in the numbers with this, you get 1×0.7 plus 0.9×0.3 . That happens to be 0.97. So if we now plug this all back into this equation over here, we get 0.97×0.01 over 0.5245. This gives us approximately as the correct answer 0.0185.

21 – Explaining Away 3 – lang_en_vs3.srt (6. Bayes Nets Subtitles)

.....

And if you got this right, I will be deeply impressed by the fact that you got this right. My happiness is well explained by the fact it's sunny. So if someone observes me to be happy and asks the question, "Is this because Sebastian got a raise at work?" Well, if you know it's sunny and this is a fairly good explanation for me being happy, you don't have to assume I got a raise. If you don't know about the weather, then obviously the chances are higher that the raise caused my happiness, and therefore this never goes up from .014 to .018. Now, let me ask you one final question in this next quiz, which is the probability of a raise, given that I look happy, and it's not sunny. This is the most extreme case for making a raise likely, because I am a happy guy, and it's definitely not caused by the weather. So it could just be random, or it could be caused by the raise. So please calculate this number for me and enter it into this box.

22 – Explaining Away 3 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] Well, the answer follows the exact same scheme as before, with S being replaced by not S. So this should be an easier question for you to answer. P of R given H and not S can be inverted by Bayes' rule to be as follows. Once we apply Bayes' rule, as indicated over here where we swapped H to the left side and R to the right side, you can observe that this value over here can be readily found in the table. It's actually the 0.9 over there. This value over here, the raise is independent of the weather by virtue of our Bayes network, so it's just 0.01. And as before, we apply total probability to the expression over here, and we obtain off this quotient over here that these 2 expressions are the same. P of H given not S, not R is the value over here, and the 0.99 is the complement of probability of R taken from over here, and that ends up to be 0.0833. This would have been the right answer.

23 – Conditional Dependence – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] It's really interesting to compare this to the situation over here. In both cases I'm happy, as shown over here, and I ask the same question, which is whether I got a raise at work, as R over here. But in one case I observe that the weather is sunny; in the other one it isn't. And look what it does to my probability of having received a raise. The sunniness perfectly well explains my happiness, and my probability of having received a raise ends up to be a mere 1.4%, or 0.014. However, if my wife observes it to be non-sunny, then it is much more likely that the cause of my happiness is related to a raise at work, and now the probability is 8.3%, which is significantly higher than the 1.4% before. This is a Bayes network of which S and R are independent but H adds a dependence between S and R. Let me talk about this in a little bit more detail on the next paper. So here is our Bayes network again. In our previous exercises, we computed for this network that the probability of a raise of R given any of these variables shown here was as follows. The really interesting thing is that in the absence of information about H, which is the middle case over here, the probability of R is unaffected by knowledge of S— that is, R and S are independent. This is the same as probability of R, and R and S are independent. However, if I know something about the variable H, then S and R become dependent— that is, knowing about my happiness over here renders S and R dependent. This is not the same as probability of just R given H. Obviously, it isn't because if I now vary S from S to not S, it affects my probability for the variable R. That is a really unusual situation where we have R and S are independent but given the variable H, R and S are not independent anymore. So knowledge of H makes 2 variables that previously were independent non-independent. Offered differently, 2 variables that are independent may not be in certain cases conditionally independent. Independence does not imply conditional independence.

24 – General Bayes Net – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] So we're now ready to define Bayes networks in a more general way. Bayes networks define probability distributions over graphs or random variables. Here is an example graph of 5 variables, and this Bayes network defines the distribution over those 5 random variables. Instead of enumerating all possibilities of combinations of these 5 random variables, the Bayes network is defined by probability distributions that are inherent to each individual node. For node A and B, we just have a distribution P of A and P of B because A and B have no incoming arcs. C is a conditional distribution conditioned on A and B. D and E are conditioned on C. The joint probability represented by a Bayes network is the product of various Bayes network probabilities that are defined over individual nodes where each node's probability is only conditioned on the incoming arcs. So A has no incoming arc; therefore, we just want it P of A. C has 2 incoming arcs, so we define the probability of C conditioned on A and B. And D and E have 1 incoming arc that's shown over here. The definition of this joint distribution by using the following factors has one really big advantage. Whereas the joint distribution over any 5 variables requires 2^5 to the 5 minus 1, which is 31 probability values, the Bayes network over here only requires 10 such values. P of A is one value, for which we can derive P of not A. Same for P of B. P of C given A B is derived

by a distribution over C conditioned on any combination of A and B , of which there are 4 of A and B as binary. P of D given C is 2 parameters for P of D given C and P of D given not C . And the same is true for P of E given C . So if you add those up, you get 10 parameters in total. So the compactness of the Bayes network leads to a representation that scales significantly better to large networks than the common natorial approach which goes through all combinations of variable values. That is a key advantage of Bayes networks, and that is the reason why Bayes networks are being used so extensively for all kinds of problems. So here is a quiz. How many probability values are required to specify this Bayes network? Please put your answer in the following box.

25 – General Bayes Net Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] And the answer is 13. One over here, 2 over here, and 4 over here. Simplifiably speaking, any variable that has K inputs requires 2 to the K such variables. So in total we have 1, 9, 13.

26 – General Bayes Net 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] Here's another quiz. How many parameters do we need to specify the joint distribution for this Bayes network over here where A , B , and C point into D , D points into E , F , and G and C also points into G ? Please write your answer into this box.

27 – General Bayes Net 2 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] And the answer is 19. So 1 here, 1 here, 1 here, 2 here, 2 here, 2 arcs point into G , which makes for 4, and 3 arcs point into D . Two to the 3 is 8. So we get 1, 2, 3, 8, 2, 2, 4. If you add those up, it's 19.

28 – General Bayes Net 3 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] And here is our car network which we discussed at the very beginning of this unit. How many parameters do we need to specify this network? Remember, there are 16 total variables, and the naive joint over the 16 will be 2 to the 16th minus 1, which is 65,535. Please write your answer into this box over here.

28 – General Bayes Net 3 – lang_en_vs52.srt (6. Bayes Nets Subtitles)

.....

And here is our current rope, which we discussed at the very beginning of this unit. How many parameters do we need to specify this net worth? Remember there are 16 total variables and the naive joint over the 16 will be 2 to 16th minus 1, which is 65,535. Please write your answer into this box over here.

29 – General Bayes Net 3 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] To answer this question, let us add up these numbers. Battery age is 1, 1, 1. This has 1 incoming arc, so it's 2. Two incoming arcs makes 4. One incoming arc is 2, 2 equals 4. Four incoming arcs makes 16. If we add all the right numbers, we get 47.

30 – Value Of A Network – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Thrun] So it takes 47 numerical probabilities to specify the joint compared to 65,000 if you didn't have the graph-like structure. I think this example really illustrates the advantage of compact Bayes network representations over unstructured joint representations.

31 – D Separation – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Thrun] The next concept I'd like to teach you is called D-separation. And let me start the discussion of this concept by a quiz. We have here a Bayes network, and I'm going to ask you a conditional independence question. Is C independent of A? Please tell me yes or no. Is C independent of A given B? Is C independent of D? Is C independent of D given A? And is E independent of C given D?

32 – D Separation Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Thrun] So C is not independent of A. In fact, A influences C by virtue of B. But if you know B, then A becomes independent of C, which means the only determinate into C is B. If you know B for sure, then knowledge of A won't really tell you anything about C. C is also not independent of D, just the same way C is not independent of A. If I learn something about D, I can infer more about C. But if I do know A, then it's hard to imagine how knowledge of D would help me with C because I can't learn anything more about A than knowing A already. Therefore, given A, C and D are independent. The same is true for E and C. If we know D, then E and C become independent.

33 – D Separation 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Thrun] In this specific example, the rule that we could apply is very, very simple. Any 2 variables are independent if they're not linked by just unknown variables. So for example, if we know B, then everything downstream of B becomes independent of anything upstream of B. E is now independent of C, conditioned on B. However, knowledge of B does not render A and E independent. In this graph over here, A and B connect to C and C connects to D and to E. So let me ask you, is A independent of E, A independent of E given B, A independent of E given C, A independent of B, and A independent of B given C?

34 – D Separation 2 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Thrun] And the answer for this one is really interesting. A is clearly not independent of E because through C we can see an influence of A to E. Given B, that doesn't change. A still influences C, despite the fact we know B. However, if we know C, the influence is cut off. There is no way A can influence E if we know C. A is clearly independent of B. They are different entry variables. They have no incoming arcs. But here is the caveat. Given C, A and B become dependent. So whereas initially A and B were independent, if you give C, they become dependent. And the reason why they become dependent we've studied before. This is the explain away effect. If you know, for example, C to be true, then knowledge of A will substantially affect what we believe about B. If there's 2 joint causes for C and we happen to know A is true, we will discredit cause B. If we happen to know A is false, we will increase our belief for the cause B. That was an effect we studied extensively in the happiness example I gave you before. The interesting thing here is we are facing a situation where knowledge of variable C renders previously independent variables dependent.

35 – D Separation 3 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Thrun] This leads me to the general study of conditional independence in Bayes networks, often called D-separation or reachability. D-separation is best studied by so-called active triplets and inactive triplets where active triplets render variables dependent and inactive triplets render them independent. Any chain of 3 variables like this makes the initial and final variable dependent if all variables are unknown. However, if the center variable is known— that is, it's behind the conditioning bar— then this variable and this variable become independent. So if we have a structure like this and it's quote-unquote cut off by a known variable in the middle, that separates or deseparates the left variable from the right variable, and they become independent. Similarly, any structure like this renders the left variable and the right variable dependent unless the center variable is known, in which case the left and right variable become independent. Another active triplet now requires knowledge of a variable. This is the explain away case. If this variable is known for a Bayes network that converges into a single variable, then this variable and this variable over here become dependent. Contrast this with a case where all variables are unknown. A situation like this means that this variable on the left or on the right are actually independent. In a single final example, we also get dependence if we have the following situation: a direct successor of a conversion variable is known. So it is sufficient if a successor of this variable is known. The variable itself does not have to be known, and the reason is if you know this guy over here, we get knowledge about this guy over here. And by virtue of that, the case over here essentially applies. If you look at those rules, those rules allow you to determine for any Bayes network whether variables are dependent or not dependent given the evidence you have. If you color the nodes dark for which you do have evidence, then you can use these rules to understand whether any 2 variables are conditionally independent or not. So let me ask you for this relatively complicated Bayes network the following questions. Is F independent of A? Is F independent of A given D? Is F independent of A given G? And is F independent of A given H? Please mark your answers as you see fit.

35 – D Separation 3 – lang_en_vs52.srt (6. Bayes Nets Subtitles)

.....

This leads me to the general study of position independence in base networks. Often called D-separation, or reachability. D-separation is best studied by so-called active triplets and inactive triplets. Active triplets render variables dependent. And inactive triplets render them independent. Any chain of three variables like this makes the initial and final variable dependent if all variables are known. However, if the center variable is known. That is, it's behind the conditioning bar. Then this variable and this variable become independent. So if you have a structure like this. And it's, quote unquote, cut off by a known variable in the middle. That separates or de-separates the left variable from the right variable, and they become independent. Similarly, any structure like this renders the left variable and the right variable dependent on this. The center variable is no. In which case, the left and right variable become independent. Another active triplet now requires knowledge of a variable. This is the explain away case. If this variable is known for a base network that converges into a single variable. Then this variable and this variable over here become dependent. Contrast this to the case where all variables are unknown. A situation like this means that this variable on the left or on the right are actually independent. In a single final example. We also get dependents if we have the following situation. A direct successor of a conversion variable is known. So it is sufficient if a successor of this variable is known. The variable, itself, does not have to be known. And the reason is, if you know this guy over here. We need to get knowledge about this guy over here. And by virtue of it, the case over here essentially applies. If you look at those rules. Those rules allow you to determine, for any base network, where the variables are dependent or not dependent. Given the evidence you have. If you color the notes dark, for which you do have evidence. Then you can use these rules to understand whether any two variables are conditionally dependent or not. So let me ask you for this relatively complicated base network, the following questions. Is F independent of A? Is F independent of A, given D? Is F independent of A given G? And is F independent of A, given H? Please mark your answers as you see fit.

36 – D Separation 3 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Thrun] And the answer is yes, F is independent of A. What we find for our rules of D-separation is that F is dependent on D and A is dependent on D. But if you don't know D, you can't govern any dependence between A and F at all. If you do know D, then F and A become dependent. And the reason is B and E are dependent given D, and we can transform this back into dependence of A and F because B and A are dependent and E and F are dependent. There is an active path between A and F which goes across here and here because D is known. If we know G, the same thing is true because G gives us knowledge about D, and D can be applied back to this path over here. However, if you know H, that's not the case. So H might tell us something about G, but it doesn't tell us anything about D, and therefore, we have no reason to close the path between A and F. The path between A and F is still passive, even though we have knowledge of H.

37 – Congratulations! – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Thrun] So congratulations. You learned a lot about Bayes networks. You learned about the graph structure of Bayes networks, you understood how this is a compact representation, you learned about conditional independence, and we talked a little bit about application of Bayes network to interesting reasoning problems. But by all means this was a mostly theoretical unit of this class, and in future classes we will talk more about applications. The instrument of Bayes networks is really essential to a number of problems. It really characterizes the sparse dependence that exists in many readable problems like in robotics and computer vision and filtering and diagnostics and so on. I really hope you enjoyed this class, and I really hope you understood in depth how Bayes networks work.

38 – Overview And Example – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Probabilistic Interference] [Male] Welcome back. In the previous unit, we went over the basics of probability theory and saw how a Bayes network could concisely represent a joint probability distribution, including the representation of independence between the variables. In this unit, we will see how to do probabilistic inference. That is, how to answer probability questions using Bayes nets. Let's put up a simple Bayes net. We'll use the familiar example of the earthquake where we can have a burglary or an earthquake setting off an alarm, and if the alarm goes off, either John or Mary might call. Now, what kinds of questions can we ask to do inference about? The simplest type of question is the same question we ask with an ordinary subroutine or function in a programming language. Namely, given some inputs, what are the outputs? So, in this case, we could say given the inputs of B and E, what are the outputs, J and M? Rather than call them input and output variables, in probabilistic inference, we'll call them evidence and query variables. That is, the variables that we know the values of are the evidence, and the ones that we want to find out the values of are the query variables. Anything that is neither evidence nor query is known as a hidden variable. That is, we won't tell you what its value is. We won't figure out what its value is and report it, but we'll have to compute with it internally. And now furthermore, in probabilistic inference, the output is not a single number for each of the query variables, but rather, it's a probability distribution. So, the answer is going to be a complete, joint probability distribution over the query variables. We call this the posterior distribution, given the evidence, and we can write it like this. It's the probability distribution of one or more query variables given the values of the evidence variables. And there can be zero or more evidence variables, and each of them are given an exact value. And that's the computation we want to come up with. There's another question we can ask. Which is the most likely explanation? That is, out of all the possible values for all the query variables, which combination of values has the highest probability? We write the formula like this, asking which Q values are maxable given the evidence values. Now, in an ordinary programming language, each function goes only one way. It has input variables, does some computation, and comes up with a result variable or result variables. One great thing about Bayes nets is that we're not restricted to going only in one direction. We could go in the causal direction, giving as evidence the route nodes of the tree and asking as query values the nodes at the bottom. Or, we

could reverse that causal flow. For example, we could have J and M be the evidence variables and B and E be the query variables, or we could have any other combination. For example, we could have M be the evidence variable and J and B be the query variables. Here's a question for you. Imagine the situation where Mary has called to report that the alarm is going off, and we want to know whether or not there has been a burglary. For each of the nodes, click on the circle to tell us if the node is an evidence node, a hidden node, or a query node.

39 – Overview And Example Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

The answer is that Mary calling is the evidence node. The burglary is the query node, and all the others are hidden variables in this case.

39 – Overview And Example Solution – lang_en_vs52.srt (6. Bayes Nets Subtitles)

.....

The answer is that Mary calling is the evidence node. The burglary is the query node. Then all the others are hidden variables in this case.

40 – Enumeration – lang_en_vs2.srt (6. Bayes Nets Subtitles)

.....

Now we're going to talk about how to do inference on Bays Net. We'll start with our familiar network, and we'll talk about a method called enumeration, which goes through all the possibilities, adds them up, and comes up with an answer. So what we do is start by stating the problem. We're going to ask the question of, "What is the probability that the burglar alarm occurred, given that John called and Mary called?" We'll use the definition of conditional probability to answer this. So this query is equal to the joint probability distribution of all three variables, divided by the conditionalized variables. Now, note I'm using a notation here where, instead of writing out the probability of some variable equals true, I'm just using the notation, "Plus," and then the variable name in lowercase. And if I wanted the negation, I would use negation sign. Notice there's a different notation where, instead of writing out the plus and negation signs, we just use the variable name itself, P of E, to indicate E is true. That notation works well, but it can get confusing between, "Does P of E mean E is true, "or does it mean E is a variable?" And so we're going to stick to the notation where we explicitly have the pluses and negation signs. To do inference by enumeration, we first take a conditional probability and rewrite it as unconditional probabilities. Now we enumerate all the atomic probabilities and calculate the sum of products. Let's look at just the complex term on the numerator first. The procedure for figuring out the denominator would be similar, and we'll skip that. So, the probability of these three terms together can be determined by enumerating all possible values of the hidden variables. In this case there are two—E and A. So we'll sum over those variables for all values of E, and for all values of A— In this case they're Boolean, so there's only two values of each. We ask, "What's the probability of this unconditional term?" And that we get by summing out over all possibilities, E and A being true or false. Now, to get the values of these atomic events, we'll have to rewrite this equation in a form that corresponds to the conditional probability tables that we have associated with the Bays net. So we'll take this whole expression and rewrite it. It's still a sum over the hidden variables, E and A. But now I'll rewrite this expression in terms of the parents of each of the nodes in the network. So that gives us the product of these five terms, which we then have to sum over all values of E and A. If we call this product F of EA, then the whole answer is the sum of f for all values of E and A. So it's the sum of four terms, where each of the terms is a product of five numbers. Where do we get the numbers to fill in this equation? From the conditional probability tables from our model. So let's put the equation back up, and we'll ask you for the case where both E and A are positive, two look up in the conditional probability tables, and fill in the numbers for each of these five terms, and then multiply them together and fill in the product.

41 – Enumeration Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

We get the answer by reading numbers off the conditional probability tables, so probability of B being positive is 0.001. Of E being positive, because we're dealing with the positive case now for the variable E, is 0.002. The probability of A being positive, because we're dealing with that case, given that B is positive and the case for an E is positive, that we can read off here as 0.95. The probability that J is positive given that A is positive is 0.9. And finally, the probability that M is positive given that A is positive we read off here as 0.7. We multiple all those together, it's going to be a small number because we've got the .001 and the .002 here. Can't quite fit it in the box, but it works out to .000001197. That seems like a really small number, but remember, we have to normalize by the $P(+j,+m)$ term, and this is only 1 of the 4 possibilities. We have to enumerate over all 4 possibilities for E and A, and in the end, it works out that the probability of the burglar alarm being true given that John and Mary calls, is 0.284. And we get that number because intuitively, it seems that the alarm is fairly reliable. John and Mary calling are very reliable, but the prior probability of burglary is low. And those 2 terms combine together to give us the 0.284 value when we sum up each of the 4 terms of these products.

42 – Speeding Up Enumeration – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Norvig] We've seen how to do enumeration to solve the inference problem on belief networks. For a simple network like the alarm network, that's all we need to know. There's only 5 variables, so even if all 5 of them were hidden, there would only be 32 rows in the table to sum up. From a theoretical point of view, we're done. But from a practical point of view, other networks could give us trouble. Consider this network, which is one for determining insurance for car owners. There are 27 different variables. If each of the variables were boolean, that would give us over 100 million rows to sum out. But in fact, some of the variables are non-boolean, they have multiple values, and it turns out that representing this entire network and doing enumeration we'd have to sum over a quadrillion rows. That's just not practical, so we're going to have to come up with methods that are faster than enumerating everything. The first technique we can use to get a speed-up in doing inference on Bayes nets is to pull out terms from the enumeration. For example, here the probability of b is going to be the same for all values of E and a. So we can take that term and move it out of the summation, and now we have a little bit less work to do. We can multiply by that term once rather than having it in each row of the table. We can also move this term, the P of e, to the left of the summation over a, because it doesn't depend on a. By doing this, we're doing less work. The inner loop of the summation now has only 3 terms rather than 5 terms. So we've reduced the cost of doing each row of the table. But we still have the same number of rows in the table, so we're going to have to do better than that. The next technique for efficient inference is to maximize independence of variables. The structure of a Bayes net determines how efficient it is to do inference on it. For example, a network that's a linear string of variables, X_1 through X_n , can have inference done in time proportional to the number n, whereas a network that's a complete network where every node points to every other node and so on could take time 2^n to the n if all n variables are boolean variables. In the alarm network we saw previously, we took care to make sure that we had all the independence relations represented in the structure of the network. But if we put the nodes together in a different order, we would end up with a different structure. Let's start by ordering the node John calls first and then adding in the node Mary calls. The question is, given just these 2 nodes and looking at the node for Mary calls, is that node dependent or independent of the node for John calls?

43 – Speeding Up Enumeration Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

[Norvig] The answer is that the node for Mary calls in this network is dependent on John calls. In the previous network, they were independent given that we knew that the alarm had occurred. But here we don't know that the alarm had occurred, and so the nodes are dependent because having information about one will affect the

information about the other.

44 – Speeding Up Enumeration 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Norvig] Now we'll continue and we'll add the node A for alarm to the network. And what I want you to do is click on all the other variables that A is dependent on in this network.

45 – Speeding Up Enumeration 2 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Norvig] The answer is that alarm is dependent on both John and Mary. And so we can draw both nodes in, both arrows in. Intuitively that makes sense because if John calls, then it's more likely that the alarm has occurred, likely as if Mary calls, and if both called, it's really likely. So you can figure out the answer by intuitive reasoning, or you can figure it out by going to the conditional probability tables and seeing according to the definition of conditional probability whether the numbers work out.

45 – Speeding Up Enumeration 2 Solution – lang_en_vs52.srt (6. Bayes Nets Subtitles)

And the answer is that alarm is dependent on both John and Mary. And so we can draw both nodes in, both arrows in. Intuitively that make sense. Because if John calls, then it's more likely that the alarm has occurred, likewise as if Mary calls and if both called, it's really likely. So you can figure out the answer by intuitive reasoning. Or you can figure it out by going to the conditional probability tables, and seeing, according to the definition of conditional probability, whether the numbers work out. [BLANK_AUDIO]

46 – Speeding Up Enumeration 3 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Norvig] Now we'll continue and we'll add the node B for burglary and ask again, click on all the variables that B is dependent on.

47 – Speeding Up Enumeration 3 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Norvig] The answer is that B is dependent only on A. In other words, B is independent of J and M given A.

48 – Speeding Up Enumeration 4 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Norvig] And finally, we'll add the last node, E, and ask you to click on all the nodes that E is dependent on.

49 – Speeding Up Enumeration 4 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Norvig] And the answer is that E is dependent on A. That much is fairly obvious. But it's also dependent on B. Now, why is that? E is dependent on A because if the earthquake did occur, then it's more likely that the alarm would go off. On the other hand, E is also dependent on B because if a burglary occurred, then that would explain why the alarm is going off, and it would mean that the earthquake is less likely.

50 – Causal Direction – lang_en_vs1.srt (6. Bayes Nets Subtitles)

[Norvig] The moral is that Bayes nets tend to be the most compact and thus the easier to do inference on when they're written in the causal direction– that is, when the networks flow from causes to effects.

51 – Variable Elimination – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Let's return to this equation, which we use to show how to do inference by enumeration. In this equation, we join up the whole joint distribution before we sum out over the hidden variables. That's slow, because we end up repeating a lot of work. Now we're going to show a new technique called variable elimination, which in many networks operates much faster. It's still a difficult computation, an NP-hard computation, to do inference over Bayes nets in general. Variable elimination works faster than inference by enumeration in most practical cases. It requires an algebra for manipulating factors, which are just names for multidimensional arrays that come out of these probabilistic terms. We'll use another example to show how variable elimination works. We'll start off with a network that has 3 boolean variables. R indicates whether or not it's raining. T indicates whether or not there's traffic, and T is dependent on whether it's raining. And finally, L indicates whether or not I'll be late for my next appointment, and that depends on whether or not there's traffic. Now we'll put up the conditional probability tables for each of these 3 variables. And then we can use inference to figure out the answer to questions like am I going to be late? And we know by definition that we could do that through enumeration by going through all the possible values for R and T and summing up the product of these 3 nodes. Now, in a simple network like this, straight enumeration would work fine, but in a more complex network, what variable elimination does is give us a way to combine together parts of the network into smaller parts and then enumerate over those smaller parts and then continue combining. So, we start with a big network. We eliminate some of the variables. We compute by marginalizing out, and then we have a smaller network to deal with, and we'll show you how those 2 steps work. The first operation in variable elimination is called joining factors. A factor, again, is one of these tables. It's a multidimensional matrix, and what we do is choose 2 of the factors, 2 or more of the factors. In this case, we'll choose these 2, and we'll combine them together to form a new factor which represents the joint probability of all the variables in that factor. In this case, R and T. Now we'll draw out that table. In each case, we just look up in the corresponding table, figure out the numbers, and multiply them together. For example, in this row we have a +r and a +t, so the +r is 0.1, and the entry for +r and +t is 0.8, so multiply them together and you get 0.08. Go all the way down. For example, in the last row we have a -r and a -t. -r is 0.9. The entry for -r and -t is also 0.9. Multiply those together and you get 0.81. So, what have we done? We used the operation of joining factors on these 2 factors, getting us a new factor which is part of the existing network. Now we want to apply a second operation called elimination, also called summing out or marginalization, to take this table and reduce it. Right now, the tables we have look like this. We could sum out or marginalize over the variable R to give us a table that just operates on T. So, the question is to fill in this table for P(T)– there will be 2 entries in this table, the +t entry, formed by summing out all the entries here for all values of r for which t is positive, and the -t entry, formed the same way, by looking in this table and summing up all the rows over all values of r where t is negative. Put your answers in these boxes.

52 – Variable Elimination Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

The answer is that for +t we look up the 2 possible values for r, and we get 0.08 or 0.09. Sum those up, get 0.17, and then we look at the 2 possible values of R for -t, and we get 0.02 and 0.81. Add those up, and we get 0.83.

53 – Variable Elimination 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

So, we took our network with RT and L. We summed out over R. That gives us a new network with T and L with these conditional probability tables. And now we want to do a join over T and L and give us a new table with the joint probability of P(T, L). And that table is going to look like this.

54 – Variable Elimination 2 Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

The answer, again, for joining variables is determined by pointwise multiplication, so we have 0.17 times 0.3 is 0.051, +t and +l, 0.17 times 0.7 is 0.119. Then we go to the minuses. Minus 0.83 times 0.1 is 0.083. And finally, 0.83 times 0.9 is 0.747.

55 – Variable Elimination 3 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Now we're down to a network with a single node, T, L, with this joint probability table, and the only operation we have left to do is to sum out to give us a node with just L in it. So, the question is to compute $P(L)$ for both values of L, +l and -l.

56 – Variable Elimination 3 Solution – lang_en_vs2.srt (6. Bayes Nets Subtitles)

.....

The answer is that the positive L values—0.051 plus .083 equals .134. And the negative values, .119 plus .747 equals .866.

57 – Variable Elimination 4 – lang_en_vs2.srt (6. Bayes Nets Subtitles)

.....

So that's how variable elimination works. It's a continued process of joining together factors to form a larger factor, and then eliminating variables by summing out. If we make a good choice of the order in which we apply these operations, then variable elimination can be much more efficient than just doing the whole enumeration.

58 – Approximate Inference – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Now I want to talk about approximate inference by means of sampling. What do I mean by that? Say we want to deal with a joint probability distribution, say the distribution of heads and tails over these 2 coins. We can build a table and then start counting by sampling. Here we have our first sample. We flip the coins and the one-cent piece came up heads, and the five-cent piece came up tails, so we would mark down one count. Then we'd toss them again. This time the five cents is heads, and the one cent is tails, so we put down a count there, and we'd repeat that process and keep repeating it until we got enough counts that we could estimate the joint probability distribution by looking at the counts. Now, if we do a small number of samples, the counts might not be very accurate. There may be some random variation that causes them not to converge to their true values, but as we add more counts, the counts—as we add more samples, the counts we get will come closer to the true distribution. Thus, sampling has an advantage over inference in that we know a procedure for coming up with at least an approximate value for the joint probability distribution, as opposed to exact inference, where the computation may be very complex. There's another advantage to sampling, which is if we don't know what the conditional probability tables are, as we did in our other models, if we don't know these numeric values, but we can simulate the process, we can still proceed with sampling, whereas we couldn't with exact inference.

58 – Approximate Inference – lang_en_vs51.srt (6. Bayes Nets Subtitles)

.....

Now I want to talk about [BLANK_AUDIO] Approximate inference by means of sampling. What do I mean by that? Say we want to deal with a joint probability distribution. Say the distribution of heads and tails over these two coins. We can build table, and then start counting by sampling. Here we have our first sample. We flip the coins and

the one cent piece came up heads. And the five cent piece came up tails. So we mark down one count [BLANK_AUDIO] Then we toss them again. This time the five cent is heads, and the one cent is tails. So we put down a count there, and we repeat that process. [BLANK_AUDIO] And keep repeating it. [BLANK_AUDIO] Until we got enough counts that we could estimate the joint probability distribution by looking at the counts. Now if we do a small number of samples, the counts might not be very accurate. There may be some random variation that causes them not to converge to the true values. But as we add more counts, as we add more samples, the counts we get will come closer to the true distribution. Thus, sampling has an advantage over inference in that we know a procedure for coming up with at least an approximate value for the joint probability distribution. As opposed to exact inference, where the computation may be very complex. There's another advantage to sampling, which is, if we don't know what the conditional probability tables are, as we did in our other models, if we don't know these numeric values, but we can simulate the process, we can still proceed with sampling, whereas we couldn't with exact inference.

59 – Sampling Example – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Here's a new network that we'll use to investigate how sampling can be used to do inference. In this network, we have 4 variables. They're all boolean. Cloudy tells us if it's cloudy or not outside, and that can have an effect on whether the sprinklers are turned on, and whether it's raining. And those 2 variables in turn have an effect on whether the grass gets wet. Now, to do inference over this network using sampling, we start off with a variable where all the parents are defined. In this case, there's only one such variable, Cloudy. And it's conditional probability table tells us that the probability is 50% for Cloudy, 50% for not Cloudy, and so we sample from that. We generate a random number, and let's say it comes up with positive for Cloudy. Now that variable is defined, we can choose another variable. In this case, let's choose Sprinkler, and we look at the rows in the table for which Cloudy, the parent, is positive, and we see we should sample with probability 10% to +s and 90% a -s. And so let's say we do that sampling with a random number generator, and it comes up negative for Sprinkler. Now let's jump over here. Look at the Rain variable. Again, the parent, Cloudy, is positive, so we're looking at this part of the table. We get a 0.8 probability for Rain being positive, and a 0.2 probability for Rain being negative. Let's say we sample that randomly, and it comes up Rain is positive. And now we're ready to sample the final variable, and what I want you to do is tell me which of the rows of this table should we be considering and tell me what's more likely. Is it more likely that we have a +w or a -w?

60 – Sampling Example Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

The answer to the question is that we look at the parents. We find that the Sprinkler variable is negative, so we're looking at this part of the table. And the Rain variable is positive, so we're looking at this part. So, it would be these 2 rows that we would consider, and thus, we'd find there's a 0.9 probability for w, the grass being wet, and only 0.1 for it being negative, so the positive is more likely. And once we've done that, then we generated a complete sample, and we can write down the sample here. We had +c, -s, +r. And assuming we got a probability of 0.9 came out in favor of the +w, that would be the end of the sample. Then we could throw all this information out and start over again by having another 50/50 choice for cloudy and then working our way through the network.

61 – Approximate Inference 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Now, the probability of sampling a particular variable, choosing a +w or a -w, depends on the values of the parents. But those are chosen according to the conditional probability tables, so in the limit, the count of each sampled variable will approach the true probability. That is, with an infinite number of samples, this procedure computes the true joint probability distribution. We say that the sampling method is consistent. We can use this kind of sampling

to compute the complete joint probability distribution, or we can use it to compute a value for an individual variable. But what if we wanted to compute a conditional probability? Say we wanted to compute the probability of wet grass given that it's not cloudy. To do that, the sample that we generated here wouldn't be helpful at all because it has to do with being cloudy, not with being not cloudy. So, we would cross this sample off the list. We would say that we reject the sample, and this technique is called rejection sampling. We go through ignoring any samples that don't match the conditional probabilities that we're interested in and keeping samples that do, say the sample -c, +s, +r, -w. We would just continue going through generating samples, crossing off the ones that don't match, keeping the ones that do. And this procedure would also be consistent. We call this procedure rejection sampling.

62 – Rejection Sampling – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

But there's a problem with rejection sampling. If the evidence is unlikely, you end up rejecting a lot of the samples. Let's go back to the alarm network where we had variables for burglary and for an alarm and say when arrested, in computing the probability of a burglary, given that the alarm goes off. The problem is that burglaries are very infrequent, so most of the samples we would get would end up being— we start with generating a B, and we get a -b and then a -a. We go back and say does this match? No, we have to reject this sample, so we generate another sample, and we get another -b, -a. We reject that. We get another -b, -a. And we keep rejecting, and eventually we get a +b, but we'd end up spending a lot of time rejecting samples. So, we're going to introduce a new method called likelihood weighting that generates samples so that we can keep every one. With likelihood weighting, we fix the evidence variables. That is, we say that A will always be positive, and then we sample the rest of the variables, so then we get samples that we want. We would get a list like -b, +a, -b, +a, +b, +a. We get to keep every sample, but we have a problem. The resulting set of samples is inconsistent. We can fix that, however, by assigning a probability to each sample and weighing them correctly.

63 – Likelihood Weighting – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

In likelihood weighting, we're going to be collecting samples just like before, but we're going to add a probabilistic weight to each sample. Now, let's say we want to compute the probability of rain given that the sprinklers are on, and the grass is wet. We start as before. We make a choice for Cloudy, and let's say that, again, we choose Cloudy being positive. Now we want to make a choice for Sprinkler, but we're constrained to always choose Sprinkler being positive, so we'll make that choice. And we know we were dealing with Cloudy being positive, so we're in this row, and we're forced to make the choice of Sprinkler being positive, and that has a probability of only 0.1, so we'll put that 0.1 into the weight. Next, we'll look at the Rain variable, and here we're not constrained in any way, so we make a choice according to the probability tables with Cloudy being positive. And let's say that we choose the more popular choice, and Rain gets the positive value. Now, we look at Wet Grass. We're constrained to choose positive, and we know that the parents are also positive, so we're dealing with this row here. Since it's a constrained choice, we're going to add in or multiply in an additional weight, and I want you to tell me what that weight should be.

64 – Likelihood Weighting Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

The answer is we're looking for the probability of having a +w given a +s and a +r, so that's in this row, so it's 0.99. So, we take our old weight and multiply it by 0.99, gives us a final weight of 0.099 for a sample of +c, +s, +r and +w.

65 – Likelihood Weighting 1 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

When we include the weights, counting this sample that was forced to have a +s and a +w with a weight of 0.099, instead of counting it as a full one sample, we find that likelihood weighting is also consistent.

66 – Likelihood Weighting 2 – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Likelihood weighting is a great technique, but it doesn't solve all our problems. Suppose we wanted to compute the probability of C given +s and +r. In other words, we're constraining Sprinkler and Rain to always be positive. Since we use the evidence when we generate a node that has that evidence as parents, the Wet Grass node will always get good values based on that evidence. But the Cloudy node won't, and so it will be generating values at random without looking at these values, and most of the time, or some of the time, it will be generating values that don't go well with the evidence. Now, we won't have to reject them like we do in rejection sampling, but they'll have a low probability associated with them.

67 – Gibbs Sampling – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

A technique called Gibbs sampling, named after the physicist Josiah Gibbs, takes all the evidence into account and not just the upstream evidence. It uses a method called Markov Chain Monte Carlo, or MCMC. The idea is that we resample just one variable at a time conditioned on all the others. That is, we have a set of variables, and we initialize them to random variables, keeping the evidence values fixed. Maybe we have values like this, and that constitutes one sample, and now, at each iteration through the loop, we select just one non-evidence variable and resample it based on all the other variables. And that will give us another sample, and repeat that again. Choose another variable. Resample that variable and repeat. We end up walking around in this space of assignments of variables randomly. Now, in rejection and likelihood sampling, each sample was independent of the other samples. In MCMC, that's not true. The samples are dependent on each other, and in fact, adjacent samples are very similar. They only vary or differ in one place. However, the technique is still consistent. We won't show the proof for that.

68 – Monty Hall Problem – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Now, just one more thing. I can't help but describe what is probably the most famous probability problem of all. It's called the Monty Hall Problem after the game show host. And the idea is that you're on a game show, and there's 3 doors: door #1, door #2, and door #3. And behind each door is a prize, and you know that one of the doors contains an expensive sports car, which you would find desirable, and the other 2 doors contain a goat, which you would find less desirable. Now, say you're given a choice, and let's say you choose door #1. But according to the conventions of the game, the host, Monty Hall, will now open one of the doors, knowing that the door that he opens contains a goat, and he shows you door #3. And he now gives you the opportunity to stick with your choice or to switch to the other door. What I want you to tell me is, what is your probability of winning if you stick to door #1, and what is the probability of winning if you switched to door #2?

69 – Monty Hall Problem Solution – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

The answer is that you have a 1/3 chance of winning if you stick with door #1 and a 2/3 chance if you switch to door #2. How do we explain that, and why isn't it 50/50? Well, it's true that there's 2 possibilities, but we've learned from probability that just because there are 2 options doesn't mean that both options are equally likely. It's easier to explain why the first door has a 1/3 probability because when you started, the car could be in any one of 3 places. You chose one of them. That probability was 1/3. And that probability hasn't been changed by the revealing of one of the other doors. Why is door #2 two-thirds? Well, one way to explain it is that the probability has to sum

to 1, and if $1/3$ is here, the $2/3$ has to be here. But why doesn't the same argument that you use for 1 hold for 2? Why can't we say the probability of 2 holding the car was $1/3$ before this door was revealed? Why has that changed 2 and has not changed 1? And the reason is because we've learned something about door #2. We've learned that it wasn't the door that was flipped over by the host, and so that additional information has updated the probability, whereas we haven't learned anything additional about door #1 because it was never an option that the host might switch door #1. And in fact, in this case, if we reveal the door, we find that's where the car actually is. So you see, learning probability may end up winning you something.

70 – Monty Hall Letter – lang_en_vs1.srt (6. Bayes Nets Subtitles)

.....

Now, as a final epilogue, I have here a copy of a letter written by Monty Hall himself in 1990 to Professor Lawrence Denenberg of Harvard who, with Harry Lewis, wrote a statistics book in which they used the Monty Hall Problem as an example, and they wrote to Monty asking him for permission to use his name. Monty kindly granted the permission, but in his letter, he writes, "As I see it, it wouldn't make any difference after the player has selected Door A, and having been shown Door C– why should he then attempt to switch to Door B? So, we see Monty Hall himself did not understand the Monty Hall Problem.

7. MACHINE LEARNING

1 – Introduction to Machine Learning – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Thad, what's the difference between pattern recognition and machine learning? In practice, not much. Often the terms are used interchangeably. In theory, though, pattern recognition is not necessarily imply learning. Okay, how about data mining Again, the difference can be subtle. Often data mining focuses on exploring a large data set and trying to explain or better utilize portions of it. Often the same techniques are used in pan recognition, machine learning, and data mining. We want to deduce these sorts of techniques in this section and try to provide an intuition for when to use them. What sorts of problems can these algorithms address? Typical tasks include optical character recognition, face recognition, text retrieval, handwriting recognition, speech recognition, activity recognition, web search, and spam filtering. But toolkits like and HTK have allowed more and more people to try machine learning on a large variety of problems. Most of the problems you mentioned are for supervising where we have labeled examples from which to learn. What about unsupervised learning where we don't have labels? We'll cover some of those algorithms as well in this lesson. In fact, much of my group's current research is on creating new unsupervised algorithms which help researchers discover repeated patterns in large databases. For example, my group is working with the wild dolphin project to try to discover the fundamental units of vocalization used by Atlantic spotted dolphins. We'll mention some of that in the next lesson called parent recognition through time. In this lesson, we'll focus more on situations where the data represents a static image, or discrete moment in time. To orient everyone, let's start with a challenge question.

2 – Challenge Question – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Given the following data about the on campus AI class from last semester, construct a compact decision tree using Greedy's search and information gain that predicts student's grades this semester. Pick the most appropriate decision tree from these choices.

3 – Challenge Question Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

The answer is C. With the first attribute of Does Projects, the tree classifies 50% of its training examples, Attends Class, then separates the other two cases into B and C. Tree D works, but it's not nearly as compact. B is compact, but actually classifies examples from its training set incorrectly. After the two A grades are separated by the Attends Class attribute, the tree is trying to differentiate the last two examples using Reads Book. Both remaining training examples are classified as B grades as the attribute Reads Book does not differentiate them. A is pretty similar to C in that it's compact, but construction was not sufficiently greedy. The first attribute only classifies one training example definitively whereas the tree in C classified two of the training examples on the first question. Decision trees are an understandable and easy to learn method of classification, however they have trouble approximating certain functions like parody. Machine learning requires understanding a variety of methods and choosing ones that will most likely solve a given classification problem. In the following lesson, we hope to provide some intuition to some of the more popular machine learning methods.

4 – k-Nearest Neighbors – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Let's start a discussion on machine learning with K nearest neighbors. That sounds like a good idea. It is one of the simplest techniques to understand, and shows the challenges of choosing the right parameters for an algorithm. Right. Suppose we have some two dimensional training data, with two classes. Plus, and O. What do the pluses and O's represent? It could be anything. You're the one who tells stories in class all the time. Okay. Let's pretend I am trying to model two types of sharks by the length and width of their bite marks. Hold on. This is really specific. Where'd it come from? I was on vacation in Hawaii when I wrote this section and they were having a record number of shark attacks at the time. I know you make up these examples on the fly during class, but this one is strange even for you. Just go with it. The x-axis is the width of the wound. The y-axis is the length. Okay. And we are trying to determine is from the cookiecutter shark, which is named for the roundish bite marks it leaves in its prey. Now you're making things up again. No, no, really. Go look it up. So the pluses represent wounds we know are made from the cookiecutter sharks. And O's are from great whites? No, they're too big. But for the sake of illustration, let's let the O's be all the other bite marks we observed. Are we talking bites like on humans on a beach? No, bites on cetaceans like whales. Somehow that did not make me feel better about this example. We're trying to figure out if there has been a surge in the cookie cutter shark population, by looking at the number and types of wounds on whales because the cookie cutter sharks are rare to see, and only come out to feed at night. That actually sounds like a real problem. Only somewhat. I needed a good story to match an old data plot I had, so I did some research and came up with this one. Your mind is an interesting place to live in. I'll take that as a compliment. At the very least, this example shows the incredible variety of problems one might use machine learning for. Okay. We'll run it. Suppose I have a new bite mark I've observed on a whale and it fits into the data here. Is it a cookie cutter bite? I'd say no. It looks like a normal bite because all the other examples in the region around it are Os, which represent normal bites. Now how about this one? Looks like a cookie cutter bite to me because all the other bites around it are pluses. Care to make an algorithm around that idea? Sure. Find the nearest example in the training data set to the unknown example, and use whatever label the training example has. Okay, that algorithm will be called one nearest neighbor. But isn't the algorithm called K nearest neighbor? What's the K for? Well, sometimes you use K closest training set examples to classify your unknown example. Why would you do that? Take a look at this example. Is it a cookie cutter bite or a normal shark bite. Well the nearest neighbor to it is cookie cutter, but the next nearest are all normal shark bites. So which is it? I'd go with normal. How many nearest neighbors have you used here? Well in this case, I used K equals three and just label the unknown example with whatever the majority label was from the training examples. But how will I figure out what k is in general? Knowing you, you'd just try lots of values of k and pick the best. Precisely, but let's make that process a bit more formal.

5 – Cross Validation – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Cross validation is a testing technique that can help us tune the parameters of whichever technique we are using. It can also help us estimate what our performance is going to be on a problem, give us an idea of how hard a problem is and help to avoid a problem called over fitting. Okay, let's get started. What should we do first? First let's suppose we have 100 examples of training. We're going to make an important assumption here. But the data we've collected for training is a good representation of the problem we're trying to solve. And also, that the data spans the space of the unknown examples we might encounter. Why wouldn't that be the case? For many, many reasons. Let's take our shark bite example. Perhaps we don't have enough examples to really capture all the types of cookie cutter shark bites. Perhaps we only looked at bite marks on whales for training, but then we later decide to look at bite marks on spinner dolphins. Maybe only smaller sharks attack dolphins, or maybe we only see the whales during the winter, when they are migrating through Hawaii and the most mature sharks attack. Whereas spinner dolphin attacks happen year round and are smaller and more oval. I'm making up these specific situations, but these types of problems are real. Often a researcher spend a lot of effort making a recognizer, only to discover the data does not really represent the problem. Or that they're rare, but important examples that don't happen to be in the training set. Okay, let's assume the data we have does represent the problem adequately. Okay, let's represent those 100 examples with this gray box. We're going to choose 10% of those examples randomly and reserve them for the future. We're going to take the rest of the data and divide that into a training and test set. Okay, so I'll suggest that we take our remaining 90 examples and use 20% for testing. That would be 18 of the remaining 90 examples. And then the rest for training. That's a fine idea but it's important to choose that 80% and 20% randomly and not to mix the two. Why is that? Well, having some of our training set and our testing set will give the algorithm an unfair advantage. Okay, I see. So you use all the day for training and then all of it for testing. With Kenny's nearest neighbors and $k=1$, I can get 100% accuracy. That's exactly right. That situation is a classic example of over fitting. With $k=1$ and training and testing being the same data, it's easy to get a high recognition rate. Basically, for each test datum, the training datum that is closest is itself. But then, when we get data that was not in the training set, result might not generalize well to that new data. This tension between over fitting and generalization is one of the main challenges in machine learning. Okay, so let's go back to my suggestion. We'll use 80% for training. And 20% for an independent test set. An independent test set? It's just a buzz word that none of the trained data is in the test data. Machine learning researchers look for those magic words when reviewing papers. When writing up a result, it's always good to explicitly say you chose a randomly chosen independent test set. Hold on, why random? To avoid bias. Imagine that our bite marks on our whales were tabulated over the winter season. During that time, the cookie cutter sharks were growing. If we just chose the first 80% of the data for training, and the last 20% for tests, we might bias our training set against ourselves. Because the biggest circular bite marks are not included in the training set. That makes sense. And I guess there are lots of potential biases in collecting real data. Correct, there are trends in almost every real data set, and to combat that, we use randomness. Okay, so we selected our 80% 20% randomly. Next, for each example in the 20% independent test set, we compare it to the training data. And since we're using $k=3$, we look for the three closest neighbors. We label each test example with the majority label. And then compare it to the actual known value. We then calculate the percentage correct and report that as our accuracy. But we're not done here. We're not? No, suppose I got a result of 97%. I could have just gotten lucky in my selection of test cases. Remember, we started out with 100 examples. We reserved ten for future use, and choose 72 for training and 18 for tests. Those 18 could have happened to be the easy cases. Don't tell me, we're going to solve this problem with more iterations? Yep, we're going to divide the 90 examples randomly again into 80% training and 20% test. And we're going to calculate our accuracy on that set. Then we'll do it again and again. And average the results. How many times should we do it? Well, whatever makes sense for the amount of computer time it takes and the amount of data available. In this case, since we have 90 choose 18 possibilities. Which is over ten to the 18. The chance that we get a lot of repeated divisions of the database is small. So I would use something like 100 iterations and then average the result. You would then report we had an average 93% accuracy using 100 for cross validation. Using randomly chosen independent test subs.

That's right, you must be looking ahead in the script. No, no, that's my line. See it says it right there. Okay, yeah you're right. Let's talk about that reserve 10% of the data. Why did we do that? Okay, suppose we have a situation where we didn't know what value for k worked well for our problem. Well, we could just start with k equals one, do 100 fold cross validation, then k equals two, do cross validation again. And keep going until we get to the largest value of k , which is reasonable to test, and then choose the best. Yep, but then we've over fit our algorithm's parameters to the data. What do you mean? Tweaking an algorithm's parameters excessively can lead to overfitting, to that particular set of data. How we combat this problem, is to use our 90 examples, treating our parameters and algorithms, until we get the best results we think are reasonable. And then doing one more test, the test on that reserve 10% of the data. So we only do that test at the end, and then that result is what we hope will be representative of how our will work in real life. That's the idea. And if the data set is big enough, and represents a problem well enough, and we don't get unlucky with randomization, it should be okay. Is this sort of testing how those machines in competitions work? Yep, when DARPA was funding face recognition algorithm research, they reflect a large data set. Give some of it to the researchers to tune their algorithms. Then, use the reserve testing set to see how well the algorithms actually worked. These competitions often led to continued funding in the millions of dollars. Or not. Or not. The procedures were taken very seriously. Today, we see similar competitions at conferences and on websites. Some are for fun, others are for real money. Going to caggle.com and chlearn.org will show some examples of current competitions. Probably one of the most famous, though, is the Netflix prize. I remember that one. They paid \$1 million dollars for the algorithm that could improve the predictions of whether or not their users would like a movie by 10%. And they used a strategy similar to what we are talking about to do the testing. But Netflix has millions of examples in their data set. What if you only have a few? What, like 10 examples? Sure. Well, then you do a leave one out strategy. You mean train on nine and test on the tenth for all ten combinations of the possible training testing sets? Yep, in papers you'll sometimes see this abbreviated as LOOCV, or Leave One Out, Cross Validation. I suppose another way to do it is to leave a couple out. Yep, training on eight and testing on two allows ten choose two possibilities. Which equals 45. How did you? The script, remember? Yeah, hm. Does that sort of testing lead to publishable results? It does, if the data are rare or each example's very complicated. Okay, what about the amount of variation between folds of cross validation? Does that tell us anything? Yep, it can tell us how sensitive the system is to small changes. I don't like to depend on a recognizer that shows a lot of variation in results between folds. It could mean the algorithm is too sensitive to outliers in the data, or that I don't understand all the variables that are affecting the data. It seems to me that we can also use cross validation to determine how sensitive an algorithm is to its parameters. Yep, that's a good point. For example, if we are using k nearest neighbors, and we see a huge change in average accuracy between k equals ten and k equals 11, there is probably something fishy going on. The same can be said for any algorithm. We have to choose a parameter for the algorithm to work. We want to know how bad things can get if we are a little off. Stability to initial parameters and initial conditions is something we like to see in any algorithm we are using. So we've talked about keeping the training and testing data separate. Is there any time when training on testing data is allowed? Personally, I like to do it to see how hard a problem is, how prone to over fitting my algorithm is. I'll compare my results on testing on training to a fair test and see how much difference there is between them. If there is a lot, I know that the algorithm is over fitting or that I don't have enough training data yet. Or if I cant even get good recognition results, even on the training data, I know the problem is hard or I am using the wrong algorithm. Wow, we spent a lot of time on cross validation. Perhaps it's time for a quiz. That sounds like a good idea.

6 – Cross Validation Quiz – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Suppose you are given a top secret project by the government. Your mission, should you choose to accept it, is to a design system to identify when something important is happening at the Kremlin by observing the automobile traffic outside. Once finished, your system's accuracy will be compared against other means as new satellite data

comes in. However, to get you started the government gives you 100 days worth of observations, which are annotated as important or unimportant. Given these choices, which seems most reasonable to create the best system possible?

7 – Cross Validation Quiz Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

The last two answers train the system on all 100 days of data. They are both going to result in overfitting, even though the last answer does have a more involved test strategy. The first answer is better in that it has an independent test set. However, we're not repeating the process, so it's possible to just get lucky or unlucky in the split of the data. Therefore, this is the answer. We randomly choose our training set and our test set. We also withheld a final test set, which will represent the final performance of the system. And we've repeated the process until we have good results.

8 – The Gaussian Distribution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

We're going to be using the Gaussian distribution for a lot of our discussions on machine learning. What does it look like? It's the bell curve we often talk about in grading. You mean the one we use when we calculate the mean and standard deviation of grades in class? Yep. Great, what's its formula? It's an exponential. μ is the mean and σ^2 is the variance. And the standard deviation is just the square root of the variance, so there's just σ here. And the standard deviation tells us how fat the Gaussian is, right? Right, and there are some other nice properties of Gaussians. Like what? Well 68% of the probability mass is within 1 standard deviation of the mean. And 95% is within 2 standard deviations. 99.7% is within 3 standard deviations. Now I get why teachers grade on the curve. If a student is within 1 standard deviation of average, they get a C, which should account for most of the class. 68% to be precise. They account for some variance in randomness in performance in grading above and below the mean. If the student achieves between 1 and 2 sigmas above the mean, they get a B. And above 2 sigmas they get an A. But that would mean that only 2.5% of students would get an A. And 13.5% would get a D. Hm, that seems a bit harsh. Right, many grad courses now grade so that the A B line is at the mean. And to get a C means you had to be 2 standard deviations below it. Hm, so how are we grading this class again? I think that's a topic for later. The main point we're trying to get acquainted with here is the properties of the distribution. But why do we use the Gaussian for grading anyways?

9 – Central Limit Theorem – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Because of something called the central limit theorem. What's that? Well, very loosely, it says that if you have enough independent random variables, the sum of them will form the normal distribution which is another name for the Gaussian. Okay, let me see if I get this right. If we have enough factors influencing the student's grade like how smart they are, how good the lessons are, if they got enough sleep, how all the assignments are written, etc, the grades will approach the shape of a Gaussian? Well in theory, if there are enough factors and we have enough students. In practice those assumptions don't always hold for grading, but for many other real-world situations it's a good approximation.

10 – Grasshoppers Vs Katydid – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Okay let's do a pattern recognition example that uses the Gaussian distribution. I found this one from my colleague, Evan Keyo's work on computational entomology. What? Studying insects using computer science. Evan likes to apply data mining techniques to interesting domains and he creates some good examples. That's something that I

never would have thought to do but I'm not really a fan of bugs. Here we are trying to classify insects as grasshoppers or katydids by the abdomen length or antenna length. That's not so bad. Blue represents the grasshoppers and red represents the katydids. Let's put up a lot of data and show the plot histograms of the antenna lengths by projecting the data on the y axis. And that way we can see that the data is forming a Goshen. Correct, instead of using histograms. We're going to fit a normal distribution to the data. So you mean you're going to calculate the mean and standard deviation of the grasshopper data and use that to plot a Gaussian with the same median and standard deviation. And we will do the same thing for the katydid data as well. Now suppose I have an insect with an antenna length of three units. Is it more likely a grasshopper or a katydid? Probably a grasshopper. Correct. Because the height of the blue grasshopper curve at three is much higher than the red katydid curve. Can we put some probabilities on our guess? Yep pretty easily. What we are trying to do is calculate the probability that we are seeing a specific case c sub j given that we have seen data d . Hold on, all you need to do is measure the heights of the curves at x equal to 3 and normalize them by the sum? Yep, and since we already have the equation of the Gaussian from simply calculating the mean and standard deviation of the data- It becomes pretty easy. Just plug in the x value, get out the y value and do some simple division. Wow, that's easy. Because the central limit theorem, a lot of data tends towards being Gaussian, so this trick works for a surprising number of problems.

11 – Gaussians Quiz – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Here we have the distributions for grasshopper in blue and katydid in red. Take a look at the attached Excel sheet. Use that to calculate the mean and standard deviation of each distribution. Then, can use the equation for the Gaussian to tell us the probability that we have an antenna length of 7.

12 – Gaussians Quiz Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Here's the answer. As you can see for an antenna length of 7, it's much more likely that our insect is a katydid.

13 – Decision Boundaries – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

I guess the break-even point is where the curves cross. What do you mean? Well, at an antenna length of 5, there's equal probability that an insect is a grasshopper or a katydid. Right. That point is called a decision boundary. Meaning that everything to the right of the decision boundary is going to be guessed to be a grasshopper, and everything to the left of it is going to be predicted to be a katydid. Hey, I just noticed something. Our variances are the same in both classes. What if they are different? You can still figure the decision boundary just by looking at where the Gaussians cross. Here's an example of Gaussians with different variances. That's cool. One thing to notice with two classes of different variances is that we can get decision boundaries where we have two thresholds instead of one. How can that happen? Here's an example. Imagine that both our classes have the same mean, but different variances. The one with the lower standard deviation is skinnier and rises above the fatter one in the middle, but on either side of the middle, the fatter one wins. Neat, but what if one class is more probable than the other? You mean like if there are two times as many grasshoppers as katydids? Sure. Well, so far we've assumed equal probability of the classes, and the sum of the area under each Gaussian had to be 1. But we could change that so that the area under the Gaussian for the grasshoppers is 2. In this case, the boundaries that indicate if an insect is a katydid get pulled in towards the mean a bit more. I'm glad we talked about this issue. We've mostly been assuming the prior probabilities of the class have been equal, but it can have a big effect on creating a classifier when once class is much more likely than the other. Perhaps we should have a quiz on this idea.

14 – Recognition Quiz – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Let's say I have a set of data consisting of 10% positive examples and 90% negative examples. A student provides me with a recognizer that works 90% of the time. Should I believe that they have made a working recognizer?

15 – Recognition Quiz Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

The answer is no. If the student's classifier just returned negative 100% of the time, it would get 90% accuracy. We hope that the recognizer could do much better than that. When evaluating a recognizer, we must ask, what percentage of the database is a large class, and then compare the accuracy of our recognizer to that. If we expect 50% positive and 50% negative examples then 90% accuracy is actually impressive. But if we expect to see negatives 97% of the time then a 90% accuracy recognizer is doing very poorly.

16 – Decision Boundaries in Higher Dimensions – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Okay, I'm going to keep asking questions because I like these pretty graphics. Sure, go ahead. What if we have 2D data like how we started? Having additional dimensions might help separate the different classes of data better. Well assuming that we are modelling the distributions with 2D Gaussians, then the decision boundaries will be a conic section. You mean like lines, parabolas, hyperbolas, ellipses, or circles? Yep, and when using three dimensional Gaussian distributions, we get Bayes decision boundaries that are two dimensional hyperquadrics. Those are quite pretty.

17 – Error – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

We can also use our graphs to understand how much error we are going to have as a result of setting a decision boundary at a certain point. How so? Let's use an example from Keogh again. He is using backscatter from a laser to determine the frequency of a mosquito's wing's beating. Based on that frequency, he determined which types of mosquitos are in that area. Wait, really? Actually, one of the first uses of the principles of passive RFID was for this sort of problem. Back when the Mediterranean fruit fly was embedding California, scientists coated some fruit flies' wings with silver nitride, sterilized them and released them to see which orchards they would migrate to and how fast they'd moved. Using a radar gun, they could capture the reflected radiation. And since the fruit fly's wings were the only things oscillating around the frequency of their beating wings, they could track them really well. Given Keogh's work, it seems they can do similar things with lasers now. Without coating everything in silver? Presumably. So back to the problem. We know that our decision boundary between these two species of mosquitoes is 517 hertz. What is our expected error? Well, to the left of the decision boundary we are going to have some insights from the right class, mis-classified as left class. We can calculate that just by integrating the right Gaussian from negative infinity to 517. Or we could use tables to look up the area under the curve for a Gaussian up to that number of standard deviations to the left of the mean. Statistics books often have these sorts of tables. In this case, the area under the curve up that point is 12.2% of the whole. So we can do the same thing to the other side as well. Yep. Given our decision boundary, everything under the red curve is going to be misclassified to the right of 517 hertz. That area is 8.02%. So that means our classifier is going to have an error of 20.22%? Yep and that amount of error is the best we can do given this feature and Gaussian data. But what if misclassifying one type of mosquito is much worse than misclassifying the other? When would that happen? Well as I was working on this lesson in Hawaii they were having an outbreak of Dengue Fever. They were spraying to reduce the mosquito population. However, only a few species of mosquito carry dengue. Suppose the Gaussian on the left represents the species that carries dengue, but the one on the right is a harmless mosquito. We live in the south, there's no such thing as a harmless mosquito. Point taken, so to speak. Let's just say that the right distribution is a mosquito that doesn't carry dengue. Okay. So if we are using our lasers to determine if a particular garden needs spraying, we want to err on the side of caution. So we can move our decision boundary to the right, so there's a minimum chance that we will

miss a dengue carrying mosquito. Correct, and now the question is, how much do we move the boundary? Let's say it is ten times worse to misclassify a mosquito from the left dengue carrying class, than from the right, less dangerous class. So we move the boundary to the right. So that the amount of error contributed by the area under the right curve is 10x more than the error contributed by the area under the left curve. So basically we can weight our decision based on whatever areas we want to avoid. We might cause more error, but those errors are less serious. Correct.

18 – Bayes Classifier – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Perhaps we should be more formal about the Bayesian Classifiers as well? If we must. I like to learn the intuition first, and then stare at the math in the book until I really get it. This stuff isn't hard, it's just Bayes Rule again. Let's do a quick review of Bayes Rule. Take a look at the formula. We are calculating the probability of class c_j , given that we have data point d . Which is equal to $p(D/C_j)$. Which is the probability of generating instance of d being in class c_j , Times $p(C_j)$ Which is the probability of the occurrence of class c_j . Divided by P of d . Which is the probability of instance d occurring. The hard part seems to be that p of d given c_j . How so Well, if I have a piece of data and I'm trying to find out if it belongs to class one or two Or classes three, four, five, etc. Good point, a number of classes. The divisor p of d is going to be the same across all the equations. We probably know the prior probability of each of the classes p of c_j simply by counting the number of each in our training data. If we're lucky they're all equal. And they affect each equation the same. Which means the important part remains is the p of d given c_j . Which we are modeling with the curves above. Or we can just use simple counting to determine. Actually, that's not a bad idea. Let's do a simple one d example where we estimate our distributions with simple counting to reinforce the the basic concept of what is going on before we move on.

19 – Bayes Rule Quiz – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Assume we have two classes, male and female. We have a person whose gender we do not know named Drew. Given the data base of names and genders, can you tell us whether it is more likely for Drew to be male or female?

20 – Bayes Rule Quiz Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

The answer is Female. Let's go through the math. We need to find out which is greater, the probability of male, given Drew, or the probability of female, given Drew. We can use a standard Bayes rule formula. For example, for male it would be probability of Drew, given male, times probability of male all over the probability of Drew. Let's do male first. There are three males in our database and only one is named Drew, so we have a 1/3 to start. Then for the probability of male, we have 3 males out of the 8 people in our database. The in the denominator we have the probability of Drew. We have three Drews in our database, so that's, again, 3/8. Next is female. We have 5 females in the database, and 2 of them are named Drew. That gives us our 2/5th here. And then 5 females out of our 8 person database. The denominator is the same. We have 3 Drews in our 8 person database. So we really only need to compare the numerators here, and we can see that female has a greater probability.

21 – Naive Bayes – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

When using Bayes rule, one of our problems is calculating the probability of a given data point, given a class. You mean p of d given c_j like we were talking about before. And so far in our examples we used 1 d data. What happens when we use multiple features? Then things could get pretty complicated. But if we assume that each feature is independent of the others, then it becomes easy again. That independence assumption leads to the naive

Bayes technique which is surprisingly powerful. Can we do an example? Sure. Let's assume we want to determine whether a person named Drew is male or female. And we know other features about the person. Like whether they are over 170 centimeters tall, or what their eye color is, or their hair length. And we have a database of such data, from which to trade our male versus female classifier. Okay. Hold it. We already know how to do this problem. We've seen it before. What do you mean? Well I got inspired when you said something about assuming all the features are independent of each other. If that is the case, we can represent the problem as a base net. Go on. The features are really conditionally independent, based on that class. We are assuming that each class has different distributions for the features. Thus, we can model the net, such that the class c_j is at the top, with arrows going to each of the features. Because the class is the cause and the feature distributions are the effects. Right. Once the class is established, each node in the net is independent. In other words, conditional independence based on the class. Yep, and we can represent any naive Bayes classifier in this sort of general framework with the class pointing to the features. In addition, our calculations become easier. How so?

22 – Maximum Likelihood – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

the probability of class C_j generating the data d simply the product of the the probabilities of the class generating each feature. In other words, class c_j generating feature d_1 times the probability c_j featured d_2 times all the rest of the probabilities until we get to the last feature, d_n . Yep the more efficient way to write that equation is that $p(d \text{ given } C_j)$ is equal to the product of $P(d_i \text{ given } c_j)$ for i from one to n . Assuming that all classes are equally likely the one that maximizes this equation is the one to which we assign the data. Hold it. You slipped in that all classes are equally likely. True. Technically this way of determining to which class the data belongs is called maximum likelihood. But why would we assume all classes are equally likely? Perhaps we don't trust our priors, given a new situation, or we have too little training data. Or our priors really are approximately equal. Or we're too lazy to do maximum a posteriori learning. What? It's the version of Bayes learning that weighs the hypothesis by the priors. Maximum likelihood can be thought of as a special case of maximum a posteriori. Okay. But let's continue on with our example. We're trying to give the intuition here. Yeah, good idea. Like I said, I like to get the intuition first, and then stare at the book until I get the math. So for our specific case here, we get an equation that looks like this. Correct, and the naive bayse independent assumption gives us quite a few nice benefits. Like what? First, it is space efficient. We only have to store the probability tables for each feature, not all combinations of features. And classifying with Naive Bayes is pretty fast. Just a series of table lookups and multiplication. All good things. Also, we could always switch back to the full Bayes network way of doing inference if we find that the features really are not independent and we have to start including arts with different feature nodes. So basically you're saying the same Bayes net inference methods will work for Naive Bayes and will run pretty fast. But we always have more representation power if we need it. Yep, and what I find useful but naive Bayes is that it is not sensitive to irrelevant features. Okay, I was following you up until now. But we'll have to spend some more time on this one. What do you mean? Suppose we were trying to classify a person's sex based on the features we described before, like eye color. But eye color and gender are not linked. You know that and I know that, but the technique doesn't. Suppose we just blindly use all the features we have. What happens? Well, let's look at the case of a person named Jessica. And we are going to use features like eye color and whether he or she wears a dress to determine gender. I feel like we're going to get in trouble for this. Anyway I have database of 20,000 examples. So we're data driven. Yep. In that database, we have 9,000 females who have brown eyes and 9,001 males. So this irrelevant feature basically cancels out. Yep, and the features that are more discriminating, like whether Jessica is known to wear dresses have a strong effect. Precisely. That's pretty neat. It is, and something I didn't fully appreciate until now. I mean, Bayes really is pretty spiffy. Let's do another example of naive Bayes as a quiz.

23 – Naive Bayes Quiz – lang_en_vs1.srt (7. Machine Learning Subtitles)

Here's the Bayes network representing a spam detector where the class is whether a message is spam or not, and the features of a message being spam are certain keywords in the message. Here we have Piazza, Bank, and Diplomat. Piazza might be the least likely of the use to be seen in a spam message, while Bank and Diplomat are more likely, but still could appear in regular email as well. Find the probability that a particular email is spam given that it has the word bank, but not piazza and not diplomat.

24 – Naive Bayes Quiz Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

To find the answer, we're first going to apply Bayes rule. So the probability of span given not Piazza, bank, and not diplomat, turns into the probability of not piazza, bank, and not diplomat given span, times the probability of spam all over the probability of not piazza, bank, and not diplomat. We can break the numerator up further into the probability of not piazza, given spam times the probability of bank, given spam times the probability of not diplomat given spam, all times the probability of spam. And then this will all be divided by the probability of not piazza times the probability of bank times the probability of not diplomat. Now we can go ahead and substitute in some values from the ones you were given before.

25 – No Free Lunch – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Okay, we need to keep moving. We have a lot to cover yet. But we have two good pattern recognition pattern algorithms already, KNN and naive Bayes. Why do we need anything else? Because there is no free lunch. What? The no free lunch theorem. It basically states that no one particular algorithm is optimal for all problems. Specifically, Wolpert and Macready state for any algorithm, any elevated performance over one class of problems is offset by performance over another class So you mean that naive Bayes won't work for some problems? It might not work as well as another algorithm. Can we show that? Sure.

26 – Naive Bayes vs kNN – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Let's look at our original problem. The shark bites again? Well, you asked. We can think about any of our classifiers as making a decision boundary. Even kNN? Yep, let's draw the decision boundary for one nearest neighbor. Wow, that's pretty complex. Yeah, it doesn't look anything we saw with maximum likelihood. But let's try it anyway. What do you think the decision boundary would look like if we assume both classes were equally probable, we mulled each class with a Gaussian? Ouch, that's pretty hard to visualize. Yeah, but give it a try. How about this? That will do. So the blue ellipse is the one standard deviation for iso line for the plus class. And the red ellipse is the one standard deviation contour for the O class. What do you think the decision boundary will look like? Why do you give me the hard problems? because I'm lazy. Fine. Maybe a parabola that looks like this. Actually the decision boundary is going to work better than I thought it would at first. Yeah, but look at all the border cases it won't get right. It's probably not as good as the kNN algorithm. It depends. On what? On how noisy the data is. The underlying process that generates the data really is Gaussian, Then this representation might be better in the long run, as we classify more and more data. But if the true decision boundary is more complex. Then all the cavities and complexities of the kNN boundary are going to be needed. Hey, can't we use multiple Gaussians to better this data? Sure.

27 – Using a Mixture of Gaussians – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

How about we use two Gaussian for each class? Wow that creates a much more complex decision boundary. So basically we can get any arbitrary decision boundary by adding more Gaussian? Yep it's a trick called a mixture of Gaussian which we'll come back to later. How do we keep from over fitting? What do you mean? Well in limit I would make each data point it's own Gaussian, and it would basically be the same as KNN hip and in-between the two extremes we can use fewer Gaussians which would cause the cision boundary to be smooth. But still give a good continuous approximation of the shape and the density of each class. This trick is called kernel density estimation where I have learned it as parsum window density estimation. In literature, the Gaussian kernel is often referred to as a radio basis function. So, I noticed you didn't answer my question about overfitting. Well, we can use cross validation to try to pick the number of Gaussians that give the best results. Just like we use it to figure out what K should be for KNN. Yep. And we can even use it to compare different methods, right Yep we can, but again we have a danger of accidentally overtuning both the method and its parameters to a particular training set. Which is why that final independent test set is very important. That's correct.

28 – Generalization – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

We've shown when K and N might provide better decision boundaries, than naive base, and Gaussian mixture models. And warn of the dangers of over fitting. But when does having a smoother boundary help us? Well let's look at the case where we don't have enough training data. Where would we put the decision boundary, if we're using one nearest neighbors for this problem? Here. And if we were using maximum likelihood, modeling the classes with Gaussians? Probably around here. But suppose we get one more piece of training data that looks like this? Okay, now that would cause the one year's neighbor boundary to split here, here and here. But the data is really starting to look like it's being generated by Gaussian processes. So I bet that this boundary is going to be better for us long term. Why not use three nearest neighbor? We could, that would lead to one threshold for the boundary again, but we're still sensitive to individual trading points on the boundary possibly messing us up. Actually let's use an extreme example to show the problems of over generalization with K nearest neighbor. Okay. What happens as we make K big? Well, we already said it smooths the decision boundary. You mean like using all the data points in the training set? Yep. In this case, everything would be classified as the left red class because it has six examples and the blue class only has five. I guess that's why there's no free lunch. We need to find a balance between the method which classifies data with the highest accuracy, doesn't over fit and generalizes well for trading data to our unseen data without losing its discrimination power. And that is why there are so many different methods in machine learning. How do we choose one?

29 – Visualization – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

One of the first things to do is visualize the data to get a sense of it. If most of the classes from balls of data without many concavities, then modeling it with Gaussians will probably work well. If there are situations where classes interpenetrate, but still have distinct boundaries, then k-nearest neighbors or one of the kernel methods will probably work. Sometimes the data is so highly dimensional that it's hard to visualize. For those situations, we can also use methods like decision trees and boosting to help you understand which features are most important. We should probably talk about those techniques next.

30 – Decision Trees with Discrete Information – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Decision trees are another machine learning technique that is easy to understand and can often reveal features about a dataset that might not be immediately obvious. Why don't we get started with an example? Okay, suppose we like to play tennis. I'm more of a table tennis guy, but okay, I enjoy hitting the ball around on the court. Since we live in Atlanta we don't like to play when it's sunny and humid because we'll get heat stroke. But since it's often rainy in

Atlanta, we'll still play as long as it isn't too windy. Because then you could just get too cold. The right sort of day is an overcast one. We can capture all that in a decision tree. I see. First, we look to see what the forecast is. If it is going to be overcast, we'll reserve a tennis court. But if it's going to be sunny, then we have to see if it's also going to be humid, in which case we won't play tennis. But if the humidity is going to be normal, we'll reserve the court. And if it is predicted to be rainy, then you have to look at another feature, the windiness, to see if we are going to reserve the court or not. Decision trees look pretty simple to use. They are, and fast, too. Here's a table of a few days' worth of data. Across the top is humidity, wind, outlook and whether we play or not. So we look outlook first each time? Yes, because it's at the top of the decision tree. For this example, since it's sunny, we're going to look at humidity next to see if we're going to play. But in this example all we need to see is that the outlook is overcast, and we have our decision.

31 – Decision Tree Quiz 1 – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Now let's take a look at the last combination of conditions. What is the decision in this case, will we play tennis?

32 – Decision Tree Quiz 1 Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

The answer is yes. First, we check outlook and we see that it's raining. Then we check wind and we see that it's forecasted to be weak. So our answer is yes, we will play tennis.

33 – DTs with Continuous Information – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

What happens if we have continuous values instead of a discrete values? Then we just create a threshold for the attribute. For example, what is the tree to discriminate between these three classes? Well that's pretty easy. If x is less than threshold 1, then it's the orange class. Otherwise we need to look at the y value. If the y value is above threshold 2, then it's the red class. Otherwise its the blue class.

34 – Minimum Description Length – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Let's talk about more complex decision trees. Okay. Here's a decision tree from the Russell and Norvig book which the authors claim is how they decide if they're going to eat at a restaurant. If there are no customers the restaurant must be bad, so we don't want to eat there. If there are some customers then it must be okay, so we'll have dinner. But if it is full then it gets complex. Right, if the wait is short then they'll wait for a table. If it is over 60 minutes, then they'll leave. But between ten minutes and 60 minutes wait, the algorithm gets more complicated and looks to see if there is things like a bar, if there's an alternative nearby, if there's a reservation, and so forth. Here's data from 12 outings that we can learn a decision tree from instead of trusting what Russell and Norveig claim in their algorithm. Okay, but you don't trust Russell and Norveig? Well, they are AI researchers after all. Hey, I resent that remark. Anyway, if we learn the decision tree, we'll get something like this. Hold it. That seems a lot simpler than the original. What's the difference if it gets the same results? Well maybe our training examples do not cover all the possible situations. Suppose they did? Well, simpler is better. There's a concept of minimal description length in compression and machine learning. You want the sum of the number of bits needed to describe your compression algorithm. And the bits needed to then express the data you want to transmit, using that algorithm, to be as small as possible. Well, that gets into what we're going to discuss next, information theory. To create a compact decision tree, we want to ask a question whose answer provides the most information towards the problem. Here's the idea. We want an attribute that will split our training examples into all positive and all negative examples. But there is no one

attribute that does that. Otherwise, the problem would be trivial. Well, which attribute goes the farthest towards that goal? I guess patrons. It conclusively classifies six of the 12 examples. Okay. Which attribute is the worst? Well, I guess it would have to be type, as it doesn't provide an answer for any of the training examples.

35 – Entropy – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Before we go on, we should be more formal about what we mean by information. Here's a definition of entropy which is a measure of uncertainty and unpredictability in a random variable. We can also use it to quantify the amount of information in the message. We're going to use entropy to determine how many bits of information we need to solve this restaurant problem. Okay. Here we have a binary decision. To stay or not to stay at the restaurant. So it goes from one to two in this case. And since our goal is to separate out 6 positive and 6 negative examples for a total of 12 examples, we get, minus 6 over 12 times a log base 2 of 6 over 12 minus the quantity, minus 6 over 12 times log 2 of 6 over 12. Which actually equals 1 bit. Right. So we need 1 bit of information to solve this problem. As we go forward, we're going to use a simpler form of the equation for binary cases. B of q is equal to negative the quantity q times log base 2 of q plus 1 minus q times log base 2 of 1 minus q . And if we have p positive examples and n negative examples, we will represent it as B of p over p plus n .

36 – Information Gain – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Okay, so now we need to figure out which attribute we should use first in our decision tree. We will do that using information gain. A test on a single attribute A will probably give us only part of the one bit we need. We can measure exactly how much by looking at the entropy remaining after the attribute test. An attribute A , with d distinct values, divide the training set E into subsets E_1 through E_d . Each subset $E_{sub\ k}$ has $p_{sub\ k}$ positive examples and $n_{sub\ k}$ negative examples. So if we go along that branch we will need an additional B of $p_{sub\ k} / p_{sub\ k} + n_{sub\ k}$ bits of information to answer the question. A randomly chosen example from the training set has a k th value for the attribute. But probably $p_{sub\ k} + n_{sub\ k} / p + n$. So the expected entropy remaining after testing attribute A is the sum of $k = 1$ to d of $p_{sub\ k} + n_{sub\ k} / p + n$ times $B(p_{sub\ k} / p_{sub\ k} + n_{sub\ k})$. The information gain from the attribute test on A is the expected reduction in entropy. $B(p / p + n) - \text{Remainder}(A)$. Now we can figure out which attribute is the most important one to use first. For patrons, the gain of patrons is equal to $1 - [2/12, \text{ because we have 2 examples where we go because there's no patrons in the restaurant, times } B(0/2) + 4/12 B(4/4), \text{ which is to represent the case where we have some patrons in the restaurant and we actually stay there, } + 6/12 B(2/6)],$ which represents the situation where it's full, and sometimes we stay and sometimes we leave. That gives us approximately 0.541 bits. The gain for type is equal to 1 again, remember, we're trying to get one bit of information, $- 2/12 B(1/2),$ which represents when the type is French. $+ 2/12 B(1/2),$ which represents Italian. $4/12 B(2/4),$ which is Thai. And finally, the Burger joint, which is $4/12 B(2/4)$. But because all of these numbers end up canceling each other out, we get 0 bits. So now we know that we should use patrons for our first attribute at the top of the tree. What do we do next? We take the remaining examples that patrons does not decide and iterate on the process until we have no more examples to explain. That seems easy enough. Here's something interesting. Remember that Type did not provide us much information for the top level? Yeah? Well, two levels down it will actually provide the most information gain. That's cool. Is it possible we will use the same attribute repeatedly in the same decision tree? Yes, that does happen sometimes. Interesting, let's try another example.

37 – Decision Tree Quiz 2 – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Here's a set of decisions. Given this database, calculate the information gain for each attribute. Outlook, Temp, Humidity, and Wind.

38 – Decision Tree Quiz 2 Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Let's go through how to find the answer here. Recall these formulas from our last lecture. To find the information gain, we have the total entropy for the situation. Minus the remainder after that attribute is taken care of. So first we need to calculate total entropy of our situation. We have nine positive examples out of 14 total examples in this situation. So you can go through the formula for B. You have negative 9 over 14, which is $.643 * \log_2$ of that number, minus 1 minus $.643 * \log_2$ of that number, which is .940 bits. Next, we need to calculate the remainder for each of our situations. Let's start with Outlook. We have three values for Outlook, sunny, overcast, and rainy. We have five examples of sunny, out of our 14 examples in our database. Two of those resulted in a positive decision, so we need the entropy of two out of five. Next we had four examples of overcast and all four of those were positive. Finally for rainy we had five examples and three of those were positive. Going through the calculation gives us 0.246. We also had three values for temperature. Hot, mild, and cool. Again we follow the same procedure to get our value of .028. Humidity and wind only have two values each. You can have a high humidity or a normal humidity. And for wind, we can have a weak wind or a strong wind. Our gain for humidity is 0.151 and our gain for wind is 0.047.

39 – Random Forests – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Have you heard of random forests? You mean forests with strange names like Dark Entry Forest or Crooked Forest? No, we're talking machine learning here, remember? You mean the ensemble learning technique where you train several decision trees and have them vote on the answer. Right. Random forests often work quite well for machine learning tasks, but how do they work? It is a bootstrap aggregation technique. I prefer the term bagging, especially since the name is so apt for the algorithm. Sure. It's a bagging technique where you use random sampling of the training data, a random selection from the attributes in that data, to create multiple decision trees. That's it. Well, when given unknown data, those decision trees vote on the result. The idea of having multiple techniques voting on the outcome is often called a mixture of experts, but here is just a mixture of decision trees. Yep. So why does this technique do better than a single decision tree? The random sampling seems to help avoid over-fitting, which is often a problem with a single decision tree. Can't see the forest for the tree? Sure, let's go with that.

40 – Boosting – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

One nice thing about decision trees is that they can tell you which features are the most useful automatically. How so? In my research on activity recognition, sometimes I don't know what data to collect. Is a tennis stroke most easily detected by an accelerometer on the wrist or a gyroscope? Does a sensor on the waist or ankle help? We can use all the sensors we can imagine and derive all the features we want and then let the decision tree learning process help us determine which features we should investigate more. It can help us with the feature selection problem in machine learning. Doesn't that lead to large trees that are hard to read? Most decision tree tools, like the one in weka or rapidminer, allow you to specify the number of decision tree leaves it is allowed to use. By setting it to a small number, we get trees that are easy for us to examine. And by doing lots of trees or by doing a random forest we can see how stable those features are. Yep. And we can even begin to quantify the percent contribution each test makes in the tree and determine which features are most complementary. So you can use the minimum number of features to get the accuracy you need? Why does that matter? Sometimes features take a lot of processing power to compute or the sensors that generate them require too much battery power. When using the sitting trees, you can optimize your classifier based on speed, power, space and many other criteria. Sometimes though, the only way to get high accuracy is to use lots of different features. That reminds me of boosting. What? Talking about lots of different features? Yeah. The idea behind boosting is that you can combine many weak

classifiers to form an ensemble that can do the classifying task. Want to explain that concept? Sure. Imagine we have ten items in our training set. Half of them are pluses and half are minuses. Okay. Now we are going to limit ourself to simple horizontal or vertical decision boundaries. Any reason not to include diagonals, or even the quadratic decision boundaries we saw with the Gaussians? Not really, but it's just easier to visualize to start with. Okay, well what's next? Let's start by picking a weak classifier that does the best job it can given the limitations. Well there are a couple I can see that would carve off two examples cleanly. Both here and here. Okay, pick one. Okay, I'll pick this one. That classifies all of the minuses correctly but only gets two of the pluses. So that gives us an error of 30 percent. Right. And we're going to plug that error into this equation to get this alpha value. And what does the alpha value do for you? Two things. It's going to give us a voting weight for this weak classifier. But don't worry about that for now. Okay. And it also gives us weights by which we increase the train examples we got wrong and decrease important examples we got right for the next generation. If the example is classified wrong we multiply it by e to the alpha sub t . If the example is classified correctly, we multiply it by e to the negative sub alpha t . In this case, the three pluses that we got wrong are now magnified for their next level, and everything else is smaller than when we started. Given this weighting, we now choose another weak classifier that best divides the examples. Okay, it looks like we'll use another vertical boundary. And we'll get a smaller error this time. But a bigger alpha value. Which means we'll weight the three minuses we got wrong even more strongly. Now what? We choose another weak classifier. Okay this time a horizontal boundary gives us the best result. And our error value is smaller and our alpha bigger. And I assume we continue iterating. Actually we are done. Well we can now classify all the examples in the training set with the weighted [INAUDIBLE] of all the weak classifiers. Where do the weights come from? They are the alphas we calculated along the way. I don't think I believe you. Try it out. Okay. Let me try this top right negative example in the training set. Well, the first weak classifier labels it a minus, so that gives us 0.42 times negative 1 so far. And a second weak classifier says it's a minus. So we have $-0.65 + -0.42$ which equals -1.07 . But the last classifier gets it wrong and adds 0.92 to the score. But the overall score is now -0.15 , which is negative, so the ensemble labels it a minus, which is correct. So we just keep monitoring the error during boosting and stop when it converges? Exactly. So I guess, just like decision trees, boosting can help us figure out which features are most interesting for recognition. Correct. That's cool but I'm not sure I trust this yet. Let's verify some more examples.

41 – Boosting Quiz – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Here's a new training set of positive and negative examples. Calculate the alpha value corresponding to each boundary choice here. Then, pick the boundary that best classifies these samples on its own.

42 – Boosting Quiz Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Here are the alpha values for each boundary. Recall that this is the formula for alpha sub T , where epsilon sub T is the number of misclassified examples. If we were only choosing one boundary, we would choose this one, since it only misclassifies these three negative examples. This gives it the highest alpha value of 0.88.

43 – Neural Nets – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

One of the oldest techniques in machine learning is neural nets. It is becoming popular again now. Why do we call it neural nets? Because it was inspired by the neurons in the brain. The brain has about 10 to 11 neurons of over 20 types. There are lots of connections between neurons, and these connections are called synapses. There are approximately 10 to the 14th synapses in the brain and their cycle time is between one to ten milliseconds. The signals are noisy spike trains of electrical potential. Okay, so what does that have to do with us? Well, in the 1950s, two researchers by the name of McCulloch and Pitts decided they would try to model neurons using inputs of

biased weight. A non-linear function that represent the neuron's cell body and outputs. What types of non-linear functions are we talking about? One was a simple step function, while the other was a probit, which is basically the Gaussian integrated. Which is better? Like everything else in machine learning, it depends on your problem, but a lot of people prefer the more smooth nature of the probit activation function. But it's pretty easy to just do the basics of computing using the step function. Just by changing the input weights and the bias weight I can do an AND and a NOT gate. That's all I need for logic and to do computing in general. And that was one the reasons why early researchers were so excited. They had shown that their simple model could do general computing. At least, in theory. Now, let's pause for quiz.

44 – Neural Nets Quiz – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Fill in this truth table to create neural net for the nor function, given that i_1 and i_2 are the inputs and a is the activation output. What are the bias weights and the input weights in this case? And what type of activation function would you use here?

45 – Neural Nets Quiz Solution – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Here's the answer. We'll use a step function here, because this is a binary function. Here is one option for the weights. You can see that when both our inputs are 0, we'll have a positive value for our bias. And as long as we set our threshold appropriately, we'll have a 1 at the output. For all other combinations, we'll get a negative value, which we can output as a 0. There are other possible combinations of these weights, but as long as they follow the truth table and also weight the inputs equally, they can still work.

46 – Multilayer Nets – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

So I guess the next step was to combine these neurons in a massive way like in the brain. Well that was the hope, but the electronics and the computers at the time were much slower than now and large nets were difficult to simulate. However, the researchers knew they were on to something. Let's take a look at a simple feed forward network. Hold it. Feed forward network? That means it has no internal state. Feed forward networks basically implement functions on their inputs. But that implies that there is a type of network that does have internal state. Those are recurrent networks. They have directed cycles with delays, like flip flops in electronics. Okay. I could see how we can do that. Let's go back to this example. So this feed forward network has two inputs, labeled as these squares here. Right. Now what? Well, let's go in reverse. The output of this network a_5 is basically the output of its activation function g , with its input being the weight $W_{35} \times a_3$, which is the output node 3. Plus W_{45} times the output of node 4, which is a_4 . But we can break that down even more because we know the output of the middle nodes is simply the activation function run on the output of the previous layer times the weights. Which means we can write the output of the speed forward network like this. Hey, I have a question. What's that? What if the output function is linear? Then the entire network can be reduced to a linear combination of the weights, but then you lose the power of the network. I see. The nonlinearity of the activation function allows each neuron to make its individual contribution to the decision boundary. Yep, and by changing the weight slowly through training, we can iteratively improve the decision boundary.

47 – Perceptron Learning – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

To show learning in neural nets, we're going to start with a single layer perceptron. That looks like two layers to me. We generally don't count the input layer. And all the units operate separately as there are no shared weights. Let's look at this middle unit. Assume we are using the sigmoid as the activation function. Suppose the white squares have zero output so that only the two black inputs are contributing. The output of this middle unit is going to be a two dimensional function of those inputs. We can visualize it using this graph. Either one or the other input has to be pretty high, or they both have to be firing somewhat for the output of this unit to go high. Hey, I remember a story about one of the first uses of perceptrons. What's that? The goal was to make a piece of hardware to do optical character recognition on simple letters by using a grid of squares. Each square had a photo cell above it so it could tell if the square was white or black. On the other side of the box was a set of lights that correspond to the letters learned. Like H, I, and T. They made a real piece of hardware to do neural nets? Yep, it was before there were interactive computers, so they implemented everything in analog hardware. That's pretty cool. Yeah, I find thinking about the problem in this context really helps make it concrete for me. Okay we'll use it for demonstrating how to learn the ways for a perceptron. Sounds great. First we have to figure out an error metric. We are going to use squared error. Which, in the case where our input is x , is simply the square of the value of the output we want, y . Minus the result of the neural net, hW , where it is applied to input x . I get it. So let's look at the OCR hardware system I was talking about. Let's input vector x as a three by three grid of pixels and we just want to recognize a few letters like H, I, and T. Each of the units represent a letter. Like the first one could be an H, the middle one could be I, and the last one could be T. In that case, we'd show the I on the input side and we'd see the output units in the display. It would start out random, but we would want it to have the middle unit activated and the other two with no activation. So we'd subtract the current perceptron's output, which we'll think of as a vector, the desired output vector of 010, and square it. Yep, and now we are ready to use gradient descent to optimize a result. In this graph, the middle unit has nine weights to optimize. We're going to calculate the direction of the gradient to minimize the error for the first weight, W_0 . I get it. Since this space is nine-dimensional, we're going to calculate the gradient for each component, W_0 to W_8 separately, and then adjust the weights to minimize the error. Not quite. Remember that we have multiple training examples for which we need to optimize the weights. If we optimize for one example and then the next- We might end up thrashing between weight values. Okay, that's a problem. I can see that we might present the letter I to the perceptron, have it optimize the weights so that the I middle unit is maximally activated for this situation. But then, when we show the perceptron T, the I middle unit is lit quite a bit when we want it to be off. Similarly, train T will have the same problem. We need a couple of the weights to really inhibit I, based on the difference between it and T. But it's not obviously without having knowledge of both I and T at the same time. What's the solution? We're going to have a learning weight, α , that we're going to use to change the weight slowly. That way each example gets a chance to contribute to the weights over time. And we iterate, showing the perceptron the examples in the training set, over and over again. Slowly adjusting the weights, and converging on the set that provides least error. That makes a lot of sense. It is similar to the iterative improvement algorithms we talked about back with some related genetic algorithms and the stochastic beam search. Exactly. These models seem pretty simple. What sort of problems can they solve?

48 – Expressiveness of Perceptron – lang_en_vs1.srt (7. Machine Learning Subtitles)

Well a single layer perceptron can only do linear decision boundaries. That seems limited but still useful. With a linear boundary we can do and and or. But not xor and that's a problem. However, a perceptron can learn a function like majority quickly, whereas a decision tree has a lot of problems. Here's the performance of a decision tree and a perceptron being trained on the majority function with 11 inputs. I guess that makes sense. For decision tree, it would need to make a full tree with 11 levels in order to do a good job. It would really need to see 2 to the 11th or 2,048 unique examples to really get it. Yet the perceptron does pretty horribly with the restaurant example we did earlier. The decision tree does much better with a lot less examples. I guess there is no free lunch. True. Can we make perceptrons better? Sure.

49 – Multilayer Perceptrons – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

The trick is to allow hidden units in your neural net, like in this diagram. How does that help? Well, remember when we showed that cliff like function with the perceptron? Yeah. What if we combine two of those in the next layer of the neural net? Then we could create a ridge like this, and be able to model any continuous function. And if we went up to three layers of units, we could combine those ridges to create bumps like this. That would allow us to model any function. Wow! It's almost like our Gaussian mixture models before in kernels. Yep, these ideas are all interrelated in machine learning. Many of these techniques can work for the same problem but each technique has its own advantages and disadvantages. Well, you've got me interest in neural nets, how do we train one within layers?

50 – Back-Propagation – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

With a technique called back-propagation. It's actually pretty simple. Well, I see we start out the same way as with the perceptron for the output layer. Yep, we calculated our area looking at what the output should be and what it is for our training example. To update the connections between the input units and the hidden units, we need to define a quantity analogous to the error term for output nodes. In other words, we need to propagate the error back through the hidden nodes towards the input nodes. The idea is that a hidden node, J , is responsible for some fraction of the error, δ_i , in each of the output nodes to which it connects. Thus, the δ_i values are divided according to the strength of the connection between the hidden node and the output node. And are propagated back to provide the δ_j values for the hidden layer. The propagation rule for the delta values is δ_j is equal to g' of in_j , times the sum over all i of W_{ji} , times δ_i . Okay, and now that we have the back-propagated error, we can create the weight update rule for the weights between the inputs and the hidden layer, which is essentially the same as the update rule for the output layer. In other words, W_{kj} is equal to W_{kj} , plus α , times a_k , times δ_j . So basically the algorithm is computing the delta values for the output units using the observed error. Then, starting with the output layer, we propagate the delta values back to the previous layer and update the weights between the two layers. Then iterate until the earliest hidden layer is reached. In neural nets, this process of updating the weights and then summing the gradient updates for all the training examples is called an epoch. And machine learning researchers often can tell how hard a problem is by how long it takes for all the weights to converge to get a minimum error on the training set. Remember the restaurant example which the decision tree could handle but the perceptron couldn't? Here is a training curve showing their error converging on a training set of 100 examples, for a neural net with four hidden units that can find an exact fit. And here is a graph showing performance on the restaurant problem using a test set and different-sized training sets. Notice the decision tree gets better results faster, but the multi-layer perceptron eventually gets there.

51 – Deep Learning – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

It seems like neural nets can eventually get good performance on hard data sets. Yep and there has been a resurgence of interest in neural nets with multiple layers with what is called deep learning. Deep learning techniques use hierarchical structures to solve complex problems. So why not use neural nets on everything? Well neural nets can require a lot of computation and examples to train. But, one of the real problems is making the results human understandable. When a decision tree classifies a piece of data in an odd way the user can at least in theory follow the tree and gain an understanding of how the system made it's decision. With neural nets it gets more complex to determine what the system is doing than the limits of it's capability. Yet neural nets can discover new features in the data that can improve performance. And we have seen recent improvements in speech and image recognition this way. With something visual like hand writing recognition we can even visualize the outputs of different layers of the the network and gain intuition of how well it is doing. It might still be hard to use the

system to gain a better understanding of the problem domain. But for many problems we care more about performance. And when I listen to talks given by NeuroNet researchers I think we are close to making systems that can pull out meaningful structure at multiple levels. With the amount of data and processing power available, these systems are requiring less supervision while make significant advances.

52 – Unsupervised Learning – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

Speaking about supervision, we should talk about unsupervised learning, too. This is one of my favorite topics. Unsupervised learning means that the algorithm is given a set of data without labels, and it attempts to determine what classes are in the data and what data belongs to which class. These algorithms are especially useful when we have large databases that are hard to label, and we need a first pass at determining if there's any structure to the data. I'm currently working with marine mammal researchers to use unsupervised algorithms to hunt for structure in dolphin vocalizations. We have already used these algorithms to uncover structure in sign language, speech, exercise data, and optical character recognition, where we know the classes. But if we can discover some new information about animals from their vocalizations, we will have shown that our algorithms are useful. Why don't we introduce unsupervised learning using K-means. That's a great idea.

53 – k-Means and EM – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

With K-Means we specify the number of classes we expect to find in a dataset. Here we expect to find two. We will start by putting the two means randomly in the data. This method is sounding like iterative improvement already. Yep, that is where we are going. We'll use blue and red for convenience for denoting our classes. We will take every point in the database, and assign it to the nearest mean. This step is called expectation. Then we'll recalculate the means based on the assignment of the points to each of the clusters. This step is called maximization. That was quick. The means are already converging to the clusters in the data. Yep, and now we'll do another expectation step and classify every data point as red or blue, depending on how close it is to the respective mean. So we are basically getting a preliminary decision boundary. In a sense. And we'll reestimate the means again by averaging the coordinates of all the data points in their clusters. And then use those new means to create a new decision boundary and reclassify all the data. And estimate the means again. And then assign points to each cluster again. Until the means and the assigns to the points to the clusters don't change. This graph shows the sum of the squares of distances of each data point to its assigned mean. The blue circles show the maximization steps and the red circles show the expectation steps. For such a clean example as this one, we quickly converge on the correct two clusters. But, what if we get stuck and it does not converge? Well, we can try random restart like we did with hill climbing, and see if it converges better with different initial conditions. In fact, random restart is not a bad policy in general to see how stable the result is. I suspect that visualizing the data helps determine whether these unsupervised techniques will work well. Yep, it does. For good clustering, we want to see data with high intercluster variance, but low intracluster variance. In other words, we want to see well separated blobs. You got it. And in my research on human activity recognition I found that many activities had this property. Human activity tends to separate well into clusters in both time and space. In the past, my students and I have used month long recordings of GPS positions and unsupervised algorithms to discover locations of significance to a user, like home, work, and grocery store. We could then use this data to predict the user's movement between these locations. This model allows a computerized assistance to provide timely reminders of what to take to work, before even getting in the car.

54 – EM and Mixture of Gaussians – lang_en_vs1.srt (7. Machine Learning Subtitles)

.....

It occurs to me that we can use the same trick, to fit a mixture of Gaussians to the data. Yep, we talked about using mixtures of Gaussians before, but we did not say how to fit them. We can use the same expectation mechanization approach to find each Gaussian's mean and variance. But before we start doing We need to know how many Gaussians we want to use. Here we'll pick two and start with the identity as the covariance matrix. Meaning that they are circles. Right. Just like with K means, we assign each point of data to the nearest cluster. We'll then reestimate the means and variances of the Gaussians. And we'll continue iterating. Effectively adjusting the position and the shape of the clusters. Until it eventually converges. Note that because we had more parameters to estimate, it took 20 Cycles to converge. Where as it took about four before with K Means. Increasing the number of dimensions to estimate almost always requires more time and more data to complete. Expectation maximization is a powerful idea and we see variance of it repeatedly in machine learning, perception, and robotics. Please take some time to go through the readings and understand the details of this technique. The next section we'll use For techniques that will allow us to recognize speech, handwriting, sign language, and many other time series.

8. PATTERN RECOGNITION THROUGH TIME

1 – Pattern Recognition through Time Intro – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Thad, we've covered a lot of different machine learning algorithms. But what about situations where we have time series? You mean like speech recognition? Yeah, I know you've worked in sign language recognition and handwriting recognition. They seem to be similar problems. Yep, they are. Pattern recognition through time is one of my specialties. And it's where the AI community has made a lot of progress. When you think about it, pattern recognition through time should be easier than situations where you have just one measurement. You mean like how face recognition is easier when you have a video of a person, as opposed to a single photo? If one particular frame of the video is badly lit or the person is looking away the next frame might be better. Precisely. But in this section, we are going to focus on situations where there's a language-like structure to the problem. In other words, there's some fundamental unit, like a phoning in speech, or letter in handwriting, that's combined with other units to make bigger constructs like words. Which are then combined in even larger structures, like sentences. So beside speech, sign language and handwriting, what other problems have the same sort of structure? Well, in my opinion, most human activities through time have a language-like structure. Playing basketball, driving a car, or even just vacuuming the floor, all seem to have familiar units of movement, combinations of those movements and source statistical grammar. I believe this area of AI is one of the most promising for future research. I'm excited to share my knowledge of the subject. And then we're going to cover dynamic time warp and hidden Markov models in this section. Where should we start? Well, let's start with a challenge question on HMMs that gives a preview of what we're going to be learning.

2 – Dolphin Whistles – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

Let's start with a problem I'm currently exploring, dolphin communication. Here's a spectrogram of a dolphin whistle. In actuality, dolphins have several types of vocalizations, including burst pulses and echolocation, but whistles are the easiest to see in a spectrogram. First, let's talk about what a spectrogram is. On this spectrogram, the x-axis is time, and the y-axis is frequency. The brightness of the pixels indicate the amount of power in that frequency band. Since we are recording in the ocean there's a lot of noise in the lower frequencies. 5kHz and lower tends to have a lot of boat and wave noise. [SOUND] However, the whistles of the Atlantic spotted dolphin tend to

range from 5kHz-17kHz. [SOUND] These whistles can be heard up to 3 miles away under water. Why do they whistle? Probably for a lot of reasons. But one thing we do know is that dolphins have signature whistles that act much like our names. If a dolphin enters a new area and feels like company, It whistles his signature whistle, which helps its friends know where it is and how to find it. So what are we seeing in this spectrogram? To me it looks like the same whistle repeated twice. The whistle has two parts. Our task is to be able to recognize classes of whistles, so that our marine mammal researchers can automatically annotate their database of recordings. Well, I see one problem already. The second time the whistle is repeated, it seems a little more drawn out.

3 – Problems Matching Dolphin Whistles – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

That problem is going to be the focus for this lesson. We need to be able to handle classes of signals, where each example may be warped in time a bit differently. Okay, well, for features, can we just use the whistle frequencies through time? Normally, that would be a good idea, but in practice, the dolphin's often raise or lower the basic frequency of the whistle. What seems to matter is the pattern of relative rises and falls of the whistle through time. In that case, let's use delta frequency. Here, instead of 5, 14, 10, 7, 10, and 14 kilohertz, let's use 9, -4, -3, 3, and 4.

4 – Warping Time – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

That will help our recognizer handle the whistles no matter what frequency they start at. Next let's work on the time warping. Just to be clear you're talking about the problem where I could draw out your name saying Fad or say your name quickly like Fad. Yep, that's precisely the problem we have. But with dolphin whistles, the problem is easy to see in the spectrogram. Basically we need to match the features that make the whistle distinct no matter if they are drawn out or produced quickly.

5 – Euclidean Distance Not Sufficient – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

You always say do the simple thing first and add intelligence only if necessary. What happens if we just do Euclidean distance here? Let's try it and find out. Here are some reasonable delta frequency numbers for the top and bottom graphs. The top time series has 21 samples, but the bottom one only has 12. How do we do a Euclidean distance? A simple thing to do is pad the bottom one with 0s. That's reasonable. Other options would be to fill in with the average value or the last value, but any of these options will work for our example. Now we can figure out the Euclidean distance by simply doing the square root of the squares of the differences. Let's see, 0 minus 0 squared is 0, 0 minus 5 squared is 25. And 2 minus 5 squared is 9. And 3 minus 5 squared is 4. And so on. Doing the math, we get the square root of 170, which is approximately 13. Yep, now let's try using dynamic time warping through the same problem.

6 – Dynamic Time Warping – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Okay, so all dynamic time warping is doing is trying to align the samples between two whistles we're comparing so that they best match up. Yep, let's line up the two signals we are trying to compare on the x and y-axes. That will allow us to more easily see how we are matching the samples. Okay, let's put the shorter one on the y-axis for convenience. A match without any time warping would be a straight diagonal going from the lower left-hand corner to the upper right. But for the types of signals we're comparing, that would be rare. We know we have to start with the first sample in each signal. In this case, they're both 0. For the next sample in the x-axis, we have another 0. Is it better to stay matched to the 0 in the shorter sequence or transition to the 5 in the next sample? Well, there will be less difference if we stay with the 0. So we'll draw a horizontal line here indicating that we're

matching the first two 0's on the x-axis with the first 0 on the y-axis. For the next sample we have a 2. That still matches with the 0 better than the 5. So we continue the horizontal line. But then we go to 3 on the x-axis which matches the 5 better. So we transition up one on the y-axis to indicate that. We're going to have several 3's in a row next on the x-axis. How do we match them to the 5's on the y-axis? Well, we're going to need a transition to the 2 and 0 on the y-axis soon. So let's make our transitions so as to keep us close to the diagonal line as possible. I get it. We're trying to match the values as closely as possible to minimize the error. And if the values otherwise tie, we try to keep to the diagonal. Correct, the 3's will match the 5's pretty well. And by the time we get to the 2, the 3 matches that even better. When we get to the 2 and the 1 on the x-axis, it'll match very well with the 2 and 0 up here. And the 0 and -1 match the 0 pretty well. Then we can match the -2, -3, -3, and -1 sequence on the x-axis to the 3, -3 on the y-axis. And the rest of the long sequence, 0, 1, 1, 1, 1, matches the matches the 1, 1, and 2 on that y-axis. The last 1 has to match the last 2 to finish the process. Now we can calculate the euclidean distance as before, but this time we had better matches. Yep, writing it out, we have the distance being the square root 0 minus 0 squared which is 0. Plus 0 minus 0 squared again, which is again 0. 2 minus 0 squared is 4. 3 minus 5 squared which is 4. I think we get the idea. Sorry, I always have to see these things written out to really understand them. After a lot of scribbling, we get a distance of the square root of 34, which is approximately equal to 6. That is less than half of the distances we calculated before when we just padded the shorter sequence with 0's. Which shows the power of dynamic time warping. For situations where we know the signals we are comparing, we'll have some sections that are faster and some sections that are slower compared to each other. DTW is a very useful tool. I see a problem, though. What's that?

7 – Sakoe Chiba Bounds – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

Let's suppose we have two signals that really are not that similar. Dynamic time warping could allow them to match much better than they should. Look at this example where we use a different shorter signal along with the same long one as in the last example. Okay. While I have to start by matching the 0 and the 3, I can match the first half of the x-axis sequence to the 3 so that I can get to the negative section of the sequence to get a better match. Then I'm in a positive section of both sequences again. Yep, that can be a problem. In this case, the distance is poor because of all those numbers the first part of the sequence, matching the 3. But the severe amount of warping we're allowing means that the distance is probably smaller than we'd like. One way we could force a more reasonable matching is to bound how much we're allowed to deviate from the diagonal. Yep, we can use Sakoe Chiba Bounds to force more reasonable matches. Sakoe Chiba, now you're just making things up. No, no, really. It says we won't allow matches outside the limits placed by these diagonal lines. That will cause the matches to be worse leading to a bigger distance. Which is what we want. How do we calculate these bounds? Often empirically, we set different bounds and use cross validation to make sure they are reasonable. Well, what if some sections of the signal should have different bounds than others? Like on our example, this section here might have a lot of variance being shorter along, but the hump may have less variability and shape. And perhaps the last section can then have more variability again. Basically there are three sections of the signal and each can have a different amount of allowed warping. We could try to have different Sakoe Chiba Bounds for each section of the signal. True, but that seems complicated to train. And I know you prefer hidden markup models for such problems. Why don't you introduce them? Okay, you know they're my favorite technique.

8 – Hidden Markov Models – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

Hidden Markov models, or HMMs as they're known to their friends, are a useful tool for looking at pattern recognition through time. They're similar to other Markov models, in that they have states that represent a set of observed phenomena, and a set of transitions that describe how we can move from one state to another. We'll be considering first order Markov models which only depend on the state immediately preceding them and not a

history of states. Hidden Markov models are a variant that are used for recognition of many types of signals that have a language-like structure. What's hidden about them? With HMM, we don't necessarily know which state matches which physical event. Instead, each state can yield certain outputs. We observe the output over time and determine a sequence of states, based on how likely they were to produce that output. I see, since HMMs and codes sequences over time, we could use them for recognizing things like speech, handwriting or gesture. Exactly, and HMM researchers have spent decades figuring out tricks like state tiling, context, stochastic grammars and boosting to improve performance. In this lecture, we will start with how to decode an HMM, where we calculate which model best fits the sequence of data.

9 – HMM Representation – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

We should probably go over how to represent an HMM. Right, the Russell and Norvig book draws them like a Markov chain and adds an output node for each state. In this representation, which is common in the machine learning community, each X_i represents a frame of data. X_0 is the beginning state which is useful for keeping data. X_1 represents the first time frame t equals 1. E_1 represents the output at that time. X_2 is the next time frame and E_2 is the output at that time. And so on, until we get to the end of the sequence. The HMM states are implicit in this representation. However, I find it easier to think of HMMs in terms of their states. So I'm going to deviate from the book and use a representation that is more specific to HMMs for this discussion. To get things started let's imagine we have a signal through time that looks like this graph. At t equals 0, its value is -2. By t equals 10 it is at -1, by t equals 15, it's at 0. And at t equals 35 it's at 1, at t equals 38 it's at 2. So it looks like the graph is made up of 4 different parts. That's right, we're going to use a 4 state HMM to represent it. We are trying to design a model that could have generated this data. In this case we are going to use a left-to-right HMM, meaning that we never transition back to a previous state once we've left it. These loops are called self transitions, which indicates that the model can stay in the same state for several timeframes. Next we need to figure out the emission probabilities. I like to call them alpha probabilities, but it's just a different name for them. All it means is which values that are allowable while we are in a given state. It's a little bit more subtle than that, right? Since the output distributions are densities, these are not really probabilities. But it's a convenient fiction for our discussion, so I'm going to stay with it. In this case, creating the output probabilities is easy. The first part of the graph is from -2 to -1. And all values are equally represented, so we can just use a box car distribution. Great, for state two the upper distribution is going to be a box car from -1 to 0. In state 3, we have a box starting from 0 to 1, and state 4 is from 1 to 2. Now we need to figure out the values for the transition probabilities. Let's look at the first part of the graph again. We spend about 10 time frames in the first part of the graph before we transition to the second part of the graph. So we want to assign an appropriate probability here where we escape state 1. Well, that seems of one off, let's assign a probability of 0.1. That way on average we expect to stay in that state for 10 frames. Since all the probabilities out of a state have to sum to 1, and we only have the self-transition probability left, we know that it has to be 0.9. And here's a little trick, if we want to know the number of time frames we expect to stay in a given state. We can just use the formula, $1 / (1 - \text{self-transition probability})$, to figure it out. Thanks, for some reason, I keep forgetting to use that trick when I'm working on more complex HMMs where there are a lot of transitions out of the state. We can continue this process for each of the states, state 2 has 5 frames. So we'll make this output probability one-fifth or 0.2 and this one four-fifth or 0.8. State 3 has 20 frames, so this arrow gets a one-twentieth, or a 0.05 and this is 0.95. And the last transition out of the model is one-third, or 0.33, with the cell's transition being 0.67. Hold it, I just realized something, you said I could figure out how many frames I expect to be in the state by using the formula, $1 / (1 - \text{self-transition probability})$. But if I set the probability to 0, I get $1 / (1 - 0)$, which equals 1. That means I'll get at least 1 output from that state. But that's fine, we output as soon as enter the state from the previous state, and then we transition to the next state. Hm, okay, that explains something else to me. A lot of manuals and toolkits post dummy state at the beginning of each model. I guess to explicitly represent entering the first state. Actually that leads me to another point. When I'm being more formal, I just put the arrow at the first state. If we can enter the model at several different positions with equal probability, I put these arrows at each

potential point of entry. If the entry points have different probabilities, I'd write them at these arrows. There are more details we will cover later, but we have shown what we need to create an HMM by inspection to represent a given signal. In practice, we expect to have lots of examples of a signal we want to model. And we'll have to create a model that can accommodate all of the different examples, balancing both generalization and overfitting. But that will come later, for now, we will assume that we can create an HMM by inspection. In fact, the technique is robust enough that inspection will work for a first pass at a problem. How about we move on to a real problem? Sounds like a good idea.

10 – Sign Language Recognition – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

We will use sign language recognition as our first application of HMMs. For example, let's consider the signs I and we and create HMMs for each of them. Here's I. [BLANK_AUDIO] We is a little different. [BLANK_AUDIO] Let's focus on the I gesture. We'll use Δy as our first feature here. Wait a second. Why don't we use Δx ? It looks like it would be easier to differentiate the two words that way. While Δy is pretty similar for the both of them. That's right. But I want to show exactly how powerful HMMs can be. So I have purposefully chosen a bad feature. We'll actually still be able to tell the difference between the two words by the difference in timing. Let's go through the process and see how that works. Okay, but sometimes it's hard to visualize the derivative of a signal. Let's have a quiz first to make sure we understand what you're talking about.

11 – Delta-y Quiz – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Here are several plots of y versus t . Given these plots, match each of the y versus t plots with their derivative plots, Δy versus t .

12 – Delta-y Quiz Solution – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Here's the answer. [BLANK_AUDIO]

13 – HMM: "I" – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

Okay, I've made an HMM for sign language word, I. Great, how did you pick those states? Well, the gesture seemed like it had three separate motions. So, I made each of those their own state and chose the transition probabilities based on the timing. We can take a look at the observed Δy here to check to see if the model is reasonable. First, the arm comes up towards the chest, and then pauses for a moment and then goes back down again. We can see that we spend a lot of time in states one and three, so there's a high probability on their self loops. State two is short so we've given it a smaller probability. Right, then we can fill in the output probability distribution similarly. For example in state two, we have a higher probability of getting a Δy of zero. We use Gaussians to model the upper probabilities here since they have nice properties.

14 – HMM: "We" – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Great, now here's the HMM I created for the gesture, we. [BLANK_AUDIO] Hold on. I would have used four states here. Why did you only use three? Well it was mostly to simplify the problem for our purposes. Note that the middle section varies a little bit more in Δy than with the middle state of I. And we spend more time in the middle state as well. I see, it looks like the transition probabilities have changed to reflect this, right? Correct, we

spend longer in that state, so we have a higher probability in the self loop. Also, we have more variation in the delta y in this gesture, which leads to a wider, and shorter, output distribution. This seems like a good time for a short quiz.

15 – I vs We Quiz – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

What property of the observed sequences of δy s can help tell the difference between the two gestures? Probability distributions in respect to starting states, probability distributions in middle states, likely time spent in middle states, or none of the above? Select all answers that could apply.

16 – I vs We Quiz Solution – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

Here's the answer. It could be probability distributions in middle states, as well as likely time spent in middle states

17 – Viterbi Trellis: "I" – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

So how do we actually go about performing recognition with the models we just created? Suppose we have a set of observations, O , that represent the samples in an example we want to recognize. We'll create something called the Viterbi Trellis to see how likely each model generated the samples in O . The one that gives us the highest probability will be considered the proper match. However, we don't know the exact sequence of states that created the output. Those are hidden, so we'll need to go through all the possibilities. That sounds like it would be long, but maybe we can simplify it as we go. Okay, here's the sequence of δy s we collected in the example we want to recognize as I or we. So our overall goal is to find P of O given λ I or the probability of our observation sequence given our model of I. Correct. Here's how we start the trellis. We have one row for each state in the model and we want to look at which state we're in at each time step. Well at the beginning we have to start in state 1, correct? That's right. Okay, then at t equal to 2, we can stay in state 1 or transition to state 2, but we can't reach state 3 yet. Looks like we've already eliminated some of this trellis. Yes, the transition probabilities in these models really limit where you can be at any given time. Perhaps we can have a quiz on that idea.

18 – "I" Transitions Quiz – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

What state could we be in at t equals to 7? What about t equals 6? Check the boxes on each possible transition.

19 – "I" Transitions Quiz Solution – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Here's the answer. We know we need to end at state 3. Before that, we could only have been in state 2 or state 3.

20 – Viterbi Trellis: "I" (continued) – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

In the middle of the trellis it looks like we have many more options. We could really be in any of the three states. That's true. The next thing we need to do is to add transition probabilities. We can pull those directly from our model, λ I. Okay, I have added the transition probabilities, that seems simple enough. But how do we go from this to an overall probability? Here's where our observation sequence comes in. Let's look at time step 1. We saw a δy of 3, but we know we can only be in state 1 at that time. By looking at the output distribution at state 1, we can see how probable it is to get a 3 from it. Normally, we would have a real distribution for these outputs. And we

could find the actual probability for the output. But for illustration purposes, I'll pick some reasonable numbers. As long as we're consistent the example should work. I'll call the probability that the Gaussian in the first state generated a delta y of 3 to be 0.5. So let's write that into this node. I see. So then for time 2 we have an output of 7. 3 was two away from state 1's mean of 5. And 7 is also two away from 5. So it will be a probability of 0.5 here as well. That works. Now we also need to consider state 2. But the probability of getting a seven there is very small, nearly zero, but not quite. So let's say, 10 to the -7. Why don't we pause for a quiz to make sure this process makes sense?

21 – Nodes for “I” – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

For each of the nodes that t equals to 5, select the answer from these choices that is closest to the output probability. Note that we have filled in the probabilities up to this point to make it easier for you to see what's going on.

22 – Nodes for “I” Solution – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

Here's the answer for t equal to 5. We've also gone ahead and filled in the rest of the trellis.

23 – Viterbi Path – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

We can follow this process to fill out all the nodes in our charts. Now, we need to look for the most likely path. Each time frame will modify the transition probability times the output probability. Note, to the most likely path may not be the greedy path, in other words. The highest expected value with each transition, may not necessarily lead to the highest valued overall path. In this case the maximum path since, be this one. Okay, so let's consider the transition from time one to time two. We can stay in state 1 or move to state 2. What should we use to compare the two options? Expected value of staying in state 1 is 0.8 times 0.5, yet going from state 1 to state 2 is 0.2 times 10 to the -7. The greedy algorithm would expect to stay in state 1 since that value is bigger. Okay, I see, how does Viterbi account for the overall path? We need to keep track of the probability of each possible sequence of the trellis. There are only two nodes in each sequence so far, so the two probabilities are $1 \times 0.5 \times 0.8 \times 0.5$, which is equal to 0.2. Or we could've gone down $1 \times 0.5 \times 0.2 \times 10$ to the negative 7th which is equal to times to negative 8. It looks like these numbers could get really small. Yes, in practice we should use log space to calculate these probabilities. Otherwise we run out of precision in the way the OS represents numbers. To keep things simple, let's keep going. Okay, next at t equal to 3, our expected values from state 1 are .8 times .6 and .2 times 10 to the negative 5. It looks like we should multiply the previous result, .2 by our new expected values. Right We also have to account for the path that went through state two earlier. That one will become ten to the negative eight times zero point five time ten to the negative five or ten time zero point five time ten to the four. Free state, we going to keep the path with maximum value, for state one is going to be this number. For state two, it's going to be this number, for state three, it'll be this number. We'll continue this process keeping track of the maximum path to get to each state at each time through the trailers. At the end, we choose the most likely path. For this trellis, it turns out to be this one. The final answer for the probability of the observation given our model is about 0.00035. We can then compare this probability to the corresponding result from the trellis for we.

24 – “We”: Transitions Quiz – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

Now it's your turn. Can you tell us the most probable path through the trellis for the model of we? We'll start small. Check the boxes between states to indicate which state transitions can occur. For example, check this box to show that state two can transition to state three.

25 – “We”: Transitions Quiz Solution – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

Here is the answer, the transitions though the trellis looks like this. Note that this actually the same as the transitions for the model of I.

26 – “We”: Transition Probabilities Quiz – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Now fill out the transition probabilities for each arrow shown on the trellis based on the model we created earlier.

27 – “We”: Transition Probabilities Quiz Solution – lang_en_vs50.srt (8. Pattern Recognition through Time Subtitles)

.....

Here is the answer. Note that the main difference between I and We is the transitions for state two.

28 – “We”: Output Probabilities Quiz – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

In the last quiz, you looked at the transition probabilities. Now, let’s consider the output probabilities. We filled out some of the probabilities to get you started. Choose from these answers and fill out the remaining nodes in the trellis.

29 – “We”: Output Probabilities Quiz Solution – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Here’s the answer. [BLANK_AUDIO]

30 – “We”: Viterbi Path – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Finally, we need to determine the most likely sequence through the trellis. Check the boxes to indicate the best path and then fill out the probability of that path here.

31 – “We”: Viterbi Path Solution – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Here’s the most likely path through the trellis. Notice that it’s very similar to the path for i, but the probability is much smaller.

32 – Which Gesture is Recognized? – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

So it looks like it’s a lot more probable that the model for I generated this data. Yep. The main difference between the values for the models producing this observation sequence has to do with the middle state. Remember that we used delta y even though it is a relatively bad feature for distinguishing I from we. The output probability Gaussian for the middle state of both models have a mean value of zero. However, with I, the expected probability of getting an actual zero at the middle state is much higher than with we, a 0.9 versus a 0.7. Also with the gesture I, we spend much less time in the middle state than with We. The transition probabilities for the middle states reflect this difference. With I, our transition from the middle state to the last state is 0.5 but with We it was 0.3. This example

shows how well HMMs can distinguish between two gestures, even with relatively poor features. What is great about HMMs, is that the difference in values for even relatively weak features, accumulates through time. The longer the gesture is, the easier it is to recognize. Perhaps we can experiment with that idea in a quiz.

33 – New Observation Sequence for “I” – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Let’s look at a new observation sequence. We’ve replaced the middle 0 observation with a new sequence -1 0 and 1. Given these probabilities, can you tell us the probability of this observation sequence, given the model for I?

34 – New Observation Sequence for “I” Solution – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

Here’s the answer. By multiplying all the transition and output probabilities, along with the curvy path and the new trellis, we get the resulting probability for I 1.42×10^{-5} .

35 – New Observation Sequence for “We” – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Now lets do the same thing for We. We have the same observation sequence as the previous quiz, where the middle zero was replaced with the sequence negative one, zero, and one. We’ve given you new probabilities for We. So go ahead and tell us the probability of this observation sequence given our model for We.

36 – New Observation Sequence for “We” Solution – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

Here’s the resulting probability for We, 2.91×10^{-5} . Note that this answer is higher than what we got for the model of i. Indicating that this observation sequence probably came from a We gesture. This is a different result from what we saw previously. Showing how the additional time spent in the middle state maps better to our model for We.

37 – HMM Training – lang_en_vs51.srt (8. Pattern Recognition through Time Subtitles)

.....

When we started this lesson, we create our models by inspection, however, most of the time we want to train using the data itself. When using HMMs for gesture recognition, I like to have at least 12 examples for each gesture I’m trying to recognize, five examples at a minimum. For illustration purposes let’s just use three examples of data for gesture eye. Even though it’s not the best feature we are going to continue using delta Y for training, the first example is the longest, it has 16 data points. And I see the next one is really short was only five data points. The last one has 14 timeframes. Let’s use a three state left to right topology again for Himercap model. For each of this examples, how are we going to figure out which data goes with which state? Let’s assume all three states have about an equal number of frames. I’ve divided the examples roughly in the thirds and drawn a boundary between each third. On average are example 12x frames long. Step with main about 4 data points first date, for example. Yup, which means a transition probability out of this stage is going to be $1/4$. And since the transitions probability have to sum to one for each date. The self transitions are $3/4$. Now that we’ve fixed the transition probabilities, let’s calculate the output probabilities. For simplicity, let’s use a single galceon again. For the first eight, we assign the 1 3 7 9 7 from the first example and the 2 and 10 from the second example and the 1 3 7 8 7 to this first state. The average of that is approximately 5.4. The standard deviation is, hold on there, let me get out my copy of Octave, about 3.1. Let me draw this under the first state as its output probability. Since you’re being handy, can you do the

other states too? We can calculate the mean value of the second state. It's around negative 0.6 with a standard deviation of 3.1. The last state has a mean of minus 5, standard deviation of about 2.6. Let's try those out probabilities as well. Now we're going to iterate in the process very similar to expectation maximization. So basically we reset the transition probabilities and calculated the output probabilities based on the assumption that the transition probabilities are correct. Then we're going to assume that the output probabilities are correct and go back and adjust the transition probabilities and so on until we converge. So basically our next step is to see if moving these boundaries on these examples between the states will lead to a better explanation of the data. Basically, we are looking to see if we get Gaussian of less variance. Well, that could be interesting, let's look at the first boundary in the first example. Clearly the 5 is closer to the mean of 5.4 so the boundary should move to the right. But the two might be better than a second state. How do you display that? Well two is 3.4 the mean of the first state. Which is 3.4 over 3.1 standard deviations away. In other words two is a little more than one standard deviation away from the first state. Okay. But the mean for the second state is -0.6 which make the two only 2.6 units away from the mean of the second state. Or 2.6 over 3.1 standard deviations away. Which is less than one standard deviation away. So two should stay in the second state. For now. What do you mean, for now? By the time we finish this iteration, the mean and standard deviations for the states will change and two might move in the first state in the next round. Okay, but how many of the boundaries change in this iteration? Let's look. I don't think the -1 is going to move from the second stage. But in the second example does -7 much closer to the main third stage -5. Yup, lets move the boundary. Shouldn't we update the main in variant. Its better to face looking at rest of the boundaries first. Last example looks like we have a lot of things we can move. Right, the three should probably move from the second state to the first. And certainly the minus five should go from the second to the third. How about the negative three in the second state. I have to calculate that one out quick. It's $2.4/3.1$ standard deviations from the second state. And two over 2.6 standard deviations from the mean of the third state. It is slightly closer to the mean of the third state, though not by much. Great, let's update the boundaries again. Wow, that's going to cause a big change in our numbers. Yep, let's recalculate everything. On average, we now have one, two, three, four, five, six, seven, eight, nine, ten, 11, 12, 13, 14. 14 over three time frames we expect in the first date. So our transition probability here is going to be one over 4.7. Or approximately 0.21 making the self-loop 0.79. The second state has 7 over 3 time frames on average. Transition probably there will be .43 and .57. And the last state has 14 over 3 time frames on average or again approximate .21 transition problem here and .79 problem here. Okay, so now we need to update the output probability numbers. I was afraid you are going to say that. Okay, clinging at octave again, I get a mean of 5.2 and a standard deviation of 3.0 for the output probability for the first state. Come on, you're the one with the computer. What's the rest? Fine, state two is easy. It has a mean of 0 and a standard deviation of 1. Do you do that on purpose? What? Make the example so the numbers would come out nice? I wish I was that smart. I'm so disillusioned. Okay, what are the values for state three? State three's mean is -5 and standard deviation about 2.5. Great, now we can do our next iteration. You mean, we gotta do it again? Don't be a cry baby, it's almost done. Okay, well, as I predicted, this 2 is now closer to the first date on this iteration in the second state Well, I guess you win that one. It's just a little over one standard deviation from the mean of the first state. And two standard deviations from the mean of the second state. But the -1 still stays in the second state. Yup, in both places. So let's calculate again. Now state one's mean goes to five. State two's mean goes to slightly negative. And state three's mean stays the same. I think we've converged. Let me take a look. Yeah, in the next iteration none of the boundaries are going to move nor are the ALPA probably going to change. Does that mean we've successfully trained HMM for I? Not yet. Technically speaking, what we've done is use Viterbi alignment to initialize the values for each state of our HMM. In other words, for each time frame in each of our examples we've assigned a state for that time frame and calculated the resulting averages of all values assigned to each state. Along with the average amount of time we expect to stay in each state. Correct.

38 – Baum Welch – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

So what's next? A process called Baum Welch re-estimation. That's like Expectation-maximization again, right? Correct. But how does it differ from what we just did? It's very similar, but with Baum Welch, every sample of the data contributes to every state proportionally to the probability of that frame of data being in that state. So you're saying that even though this 9 here is most likely in state one, its values still effects state two and three. But only a little bit because it's likelihood of belonging to states two and three is pretty low. But for data like this 2 here, it has a high likelihood of being in state one or state two. And thus, has a bigger influence on the final output probabilities for those states than for state three. Okay, that doesn't sound so bad in theory. But it sounds hard to calculate because you have to weight everything appropriately. It's actually not that bad. We can use the forward-backward algorithm to help keep track of the calculations. And it's worth the effort, because the final model often gives better results than our initial estimate. Now that we've given the intuition for what is going on, there are several good writeup's on the details of Baum Welch. Tutorial is a classic reference, and Thad's master thesis has an explanation that continues with the sign language example. Or just looking at the code to a popular HMM toolkit like HTK can be useful, too. In any case, using one of these toolkits and watching the means and variants change as the re-estimation process iterates will help build further intuition, and we highly suggest it.

39 – Multidimensional Output Probabilities – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Now that we've shown how HMMs work, let's provide some more tips on how to improve them. Okay, in our example of using HMMs to distinguish between the signs I versus we, we used delta y as a feature. But in reality delta x would be a better feature. That's true, but for other signs delta y would be a good feature. Another good feature is the size of the hand which helps us get some information as to if the hand is coming towards the camera or away. Another good feature might be the angle the hand makes to the horizontal. So sign language is two handed, we should have these features both to the right and left hands. Now that we have eight features, we are tracking for time frame, how do we integrate it into our models? Actually, it's pretty easy. We just add more dimensions for the output probabilities. All our training and recognition work like before. We just have to calculate multi-dimensional distances instead of using just one dimension. You're right. It gets hard to show on our graphs here but it's easy enough to code.

40 – Using a Mixture of Gaussians – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

What if our output probabilities aren't Gaussian? Well according to the central limit theorem, we should get Gaussians if enough factors are affecting the data. But in practice sometimes the output probabilities really are not Gaussian. It is not hard for them to be bimodal. You mean like this. Yep. Well then we can use two Gaussians to model it. Basically using the mixture of Gaussians technique. Yep, in theory we can model anything with enough Gaussians. Look at this box card distribution. The more Gaussians we use the closer we will model it, but how may should we use? That really depends on your data. Visualization really helps here. In practice I tend to limit my mixtures to two or three Gaussians, otherwise it tends to over fit.

41 – HMM Topologies – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Next, let's talk about increasing the size of our vocabulary. Okay, I've selected some signs we can use to start making phrases. But we're going to have to choose topologies for each of them. Well, we've already chosen topologies for I and we. What sign is next on your list? Well, let's add, want [BLANK_AUDIO]. How many states does that look like to you? Well it has the onset where you bring your hands up, the actual motion for the sign itself. And then you're dropping your hands again. I would say three states. Well how about for, need? [BLANK_AUDIO] That looks similar, so I'd also use three states. Okay here's a trickier one. This is an example for the sign for table. [BLANK_AUDIO] But, hold on, that was actually a poor example. There's a better example

for table. So your hands go up and down twice? That seems like four states in this case. Yep, but to make thing even more interesting, here's another example. [BLANK_AUDIO] Hold on, that time you went up and down three times. How many times is allowed? Well, generally you'll see the motion repeated twice. But if somebody's thinking hard about what to say, they might repeat the motions, stalling for time. It's sort of like an English speaker saying or. Okay, so how can we deal with that in an HMM? Well, I didn't say we always had to go from left to right with an HMM Okay. So we can actually make a loop in the HMM, say from the third to the second state, so that the motions can be repeated any number of times. Yep, you got the idea. In practice, we can try lots of different typologies, use cross-validation with randomly selected, training-independent test sets to settle on the best one. Another example of your, try the simple thing first, strategy? Exactly, but in this case, lets try the simple typology first and only add states when necessary. Let's look at another issue. Here's an example for cat, [BLANK_AUDIO] And here's another example for cat [BLANK_AUDIO]. One is one-handed, while the other is two-handed. I don't think we're going to solve that with a simple typology trick. We could try just a normal three-state HMM, then use a mixture of Gaussians to help handle the probabilities we get. But I think it makes more sense just to recognize these as two different signs. Well, how so? Well we could train a model for the first variation of the sign. We'll call it cat one. And we'll train a separate model for the second variation and call it cat two. And then we recognize either cat one or cat two, we'll just have our recognizer output, cat. Yep, that should solve the problem. While this approach adds more gestures to our vocabulary, it is often the approach that results in highest accuracy, provided we have enough examples to train all the variations.

42 – Phrase Level Recognition – lang_en_vs50.srt (8. Pattern Recognition through Time Subtitles)

Now that we have topologies for our six signs, let's talk about phrase level sign language recognition. We have eight phrases we want to recognize. Actually, you mean 7 signs and 12 phrases. Since we have two variants of cat we are recognizing, expanding all the possibilities leads to 12 phrases. Good point. To keep things simple, let's use a strict grammar. We will always have a pronoun followed by a verb followed by a noun. And we'll only recognize one phrase at a time. So let's expand that out again to be explicit. Great. Now comes the messy part. Let's assume we have data from thad's signing that we want to recognize. Here he is actually signing I need cab. Really, that's the phrase you came up with for us to use? Everyone should have a cat. Meow. Okay, anyway suppose looking at the delta y feature through time gives us this string of observations. Which is about 20 samples through time. We're going to do recognition the same way as before, but with a much bigger trellis. We have 22 states to worry about. Three for each of I, WE, NEED, WANT, CAT1 and CAT2 and four for TABLE. So where can we be when t is equal to one? Well, the grammar says we have to start with I or WE. So we have to be in the first state, of either these two pronouns. Where can we be when t equals to two? Well, we could stand the current state. We could transition into the second state of I or we. How about t equal to three? It's still easy. We can be in states one through three for I or for we. We haven't had a chance to transition out the first sign yet. But what happens with t equal to four? Once we get to t equals four, we could, at least in theory, have transitioned to either the verbs want or need. We are beginning to see the complexity, that will soon overtake us. That's right, but it's not too bad yet. At t equal to five, we have more opportunities where we can transition to need or want. But we still can't get to the nouns yet. At t equals six, we're still with the pronouns and verbs. But, at t equals seven, we could, possibly, transition to TABLE, or CAT1, or CAT2. At t equals eight, we could get into the second state of the noun signs, and a t equals nine, we can actually transition into the third state of the noun signs At t equals a ten we have the first time we can transition back from state three to state two in table. This loop back complicates our trellis a little bit. But it's not too bad. We just now have this back and forth structure. And here's the full trails for after we taking care of all the data from the example phrase. Wow. And this is the simplest example we could come up with. But note here, we couldn't actually reach all this states in the end. We have to get back to the bottom states of one of the three nouns. Yup, this trail also shows how HMM is going to take up a lot of memory. Imagine if we had a vocabulary that included all of the six thousand signs in the American Sign Language, or even worse yet, all the variations of those signs that give the ASL it's expressiveness and speed. Not to mention what would happen if we were using a real

grammar? Which brings up an interesting point. Almost all of language recognizers will have problems keeping the full trellis and memory. And keeping all the paths updated will also be a problem. How will we deal with this problem? We'll use the caustic beam search.

43 – Stochastic Beam Search – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Stochastic beam search, did we see that before? Not in detail. So far we've been doing something like a breadth first search, expanding each possible path in each time step. But now we want to prune some of those paths. Well some of those paths are going to get a low probability pretty quickly. For example, staying in the first state of the model for t until t equals 10 seems improbable. We could just drop some of those low probability paths. Yup, that's the idea of the beam search. But we don't want to get rid of all the low-probability paths. It is possible that there is a bad match in the beginning of the phrase that becomes a good match later on. For example, the signer might hesitate or accidentally start with the wrong sign before changing it. Like someone stuttering or having a false start in a spoken language. Precisely. In that case, let's keep the paths randomly in proportion of their probability. Here are some examples of high probability paths through the trellis marked in red. And some low probability paths through the trellis marked in blue. In that case, let's keep the paths randomly in proportion with their probability. Yep, the idea has some similarity to the fitness function we talk about with genetic algorithms. Or to the randomness in simulated knee length. In practice, it works very well. Randomness seems to be useful in a lot of AI. It's a principle I practice in my daily life. What? Precisely. Okay, let's get back to the real topic.

44 – Context Training – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

OK. Now let's talk about another trick. When we moved from recognizing isolated signs to recognizing phrases of signs, the combination of movements looks very different. For example, when Thad signed NEED in isolation, his hands started from a rest position and finished in the rest position. When he signs NEED in the context of I NEED CAT, the first part of NEED runs into the last part of I, and the last part of NEED runs into the first part of CAT. His hands are no longer moving so much. That's right, the signs before and after a given sign can significantly affect how it looks. So instead of recognizing the three state model for need, we're going to concatenate a combined six state model of I NEED, and NEED CAT. In practice, we often have lots of examples of phrases of sign and not individual signs to train on. In my original paper on recognizing phrases of ASL, I had 40 signs in the vocabulary, but the only training data that I had was 500 phrases, that each contained five signs. In this case, let's suppose we have lots of examples of our three sign phrase. In our original example on training in HMM in data, we assume that the data for an isolated version of the sign for I was evenly divided among its three states. We then calculated the output probabilities given that assumption, adjusted the boundaries and transition probabilities, and iterated until convergence. We're going to do the same thing for our first step here, but this time we will assume that the data is evenly divided between each sign, and then we'll divide the data for each state in each sign. And we iterate the same way we did before, adjusting the boundaries on each state and each sign until we converge. Now's when things get interesting. After we've converged everything for each sign, we're going to go back and find every place where I NEED occurs. Notice that there will be a lot fewer of those than there are examples of NEED. But let's assume that there are enough examples. Okay. Well we are going to cut out the data we think belongs to I NEED, and train the combined six state model on it. How does that help? Well the output probabilities at the boundary between I and NEED, here and here, as well as the transition probabilities in that region, will be tuned to better represent I NEED than the general case which would include we need. In speech the effect of one phoneme affecting the adjacent phoneme is called coarticulation, and this method of modeling is called context training. So I see we're going to do the same thing for NEED CAT1. Yep, and for every other two sign combination. NEED-CAT2, WANT-CAT1, WANT-CAT2, I-WANT, WE-NEED, and WE-WANT. We are going to iterate using Baum-Welch, using these larger contexts from embedded training, until we converge again. Why not use three sign

contexts, or even more when the phrases are complex enough? If we have enough data that's not a bad idea, because the benefits are actually pretty large. For recognition tasks, where there's a language structure, we expect context training to divide our error rate in half.

45 – Statistical Grammar – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Statistical grammars can help us even more. In our example up to this point, we used a simple grammar, a pronoun, verb and noun. And that placed a strong limit of where we started and ended in our Viterbi trellis. But in real life, language is not so well segmented. Instead, we can record large amounts of language and affirm in the fraction of times need follows I, versus want following I, or want following we. We can then use these probabilities to help bias our recognition based on the expected distribution of the co-occurrence of these signs. In practice, using a statistical grammar divides the error rate by another factor of 4. We started with a fundamental error rate of E . When we used context training, we divide in half. And now with statistical grammars, we're dividing it by a factor of 4 again. So, basically, we end up with our error rate divided by 8. This trick is one of the secret weapons in my research. I look for problems that might have a language-like structure and then apply these techniques to get my error rate down to something reasonable.

46 – State Tying – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

Another trick is State Tying. You mean combining training for states with the states within the model are closed? Yup. Let's look at our models for I and we again. In the case where we are recognizing isolated signs, the initial movement of the right hand going to the chest is very similar in both models. Instead of having an I state one and a we state one, we're just going to have an initial state and define the I and we models such that they both include it. That way when we entrain the HMM we have twice as much data to entrain the initial state. And we can do the same thing with a final state, since the hand going back down to rest looks much the same in the isolated models for I and we. We have to be careful with this trick, because State Tying gets more complicated when we start to worry about context training. For our new table and we want CAT2, the only states that should be tied are the first state for I and we and the last state for table and cat. And maybe not even the last states for table and CAT2, if we're using more features than just delta Y, the end of table and CAT2 could be very different. Good point, in practice I often just look for states that seem to have close means and variances during training and then determine if tying them looks logical given the motion I expect. Again it's a situation where some visualization of the data and iteration can help us improve our results.

47 – Segmentally Boosted HMMs – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

In your past work on gesture recognition, how many dimensions have you used for your output probabilities? Up to hundreds. At one point, we are creating appearance models of the hand, using a similarity metric of how closely the current hand looked like different visual models of the hand as features for the HMM. However, the problem was there was a lot of noise in that the models that didn't match well were giving relatively random results. Causing many of the dimensions to be meaningless unless the hand at that particular time matched well. So all those dimensions were actually hurting you because they were all very noisy most of the time? Correct, however we can use boosting to help us weight the feature vector to combat this problem. This technique is called segmentally boosted HMMs. One of my students, did his PhD dissertation on the idea. And his results were often 20% better than normal HMMs. For some datasets, we got up to 70% improvement. How does it work? First, we align and train the HMMs as normal. Next, we use that training to align the data that belongs to each date as best we can. We examine each state in each model iteratively. We boost by asking which features help us most to differentiate the data for our chosen state versus the rest of the states. We then weight the dimensions appropriately in that HMM.

This trick combines some of the advantages of the discriminative models with generative methods. Is it in a toolkit somewhere? Not yet. But I'm thinking about adding it to HTK and our Georgia Tech Gesture Toolkit. Personally, I think it can be pretty powerful but it's not widely known yet.

48 – Using HMMs to Generate Data – lang_en_vs52.srt (8. Pattern Recognition through Time Subtitles)

.....

One last thing I'd like to cover is why the normal HMM formulation is not good for generating data. In the early days of speech recognition, there was the hope that we could use the same HMMs we use to recognize speech, to also generate it. It turned out not to be such a good idea. Even though later advancements led to good results. Maybe an example is the best way to show the issue. You're right. Let's take this same HMM we had at the beginning of the lesson. The one which modeled the stray lines? Yep, now with just looking at the HMM, not the example data, give me ten numbers that could be generated by the first state. Okay, -1, -1, -2, -1.5, -1, -1.25, -1.75, and -2. Now some numbers for the second state. 0, -0.5, -0.25, -0.75 and -1. Same thing for the third state. 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0. And finally the last state. 1.5, 1.75 and 1. Plotting these numbers looks nothing like the original example because the output distributions have no idea of continuity. How would we fix that? We could use a lot of states to try to force a better ordering of the output, but that would lead to over-fitting. A better idea is to model the state transitions with an HMM. But use a different process while in each state that is more aware of the context to generate the final data. There are several methods that are possible. But the most sophisticated work I know of is by speech synthesis researchers. For example, Google has some good documents of how they are combining HMMs with deep belief networks to get better results.

9. LOGIC AND PLANNING

1 – Introduction – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Narrator] Hi, and welcome back. This unit is about planning. We defined AI to be the study and process of finding appropriate actions for an agent. So in some sense planning is really the core of all of AI. The technique we looked at so far was problem solving search over a state space using techniques like A star. Given a state space and a problem description, we can find a solution, a path to the goal. Those approaches are great for a variety of environments, but they only work when the environment is deterministic and fully observable. In this unit, we will see how to relax those constraints.

2 – Intro to Logic and Planning – lang_en_vs52.srt (9. Logic and Planning Subtitles)

.....

In this next section, Peter will teach us about logic and planning. Peter, what important concepts should our students look out for? A logic is a way of describing the world. When we make logical statements, we capture part of what we want to know about the world and ignore other parts. Any description is only partial. The beauty of logic is that once we've made some statements about the world we can then reason completely within the logic without having to go back to reference the world to come to some conclusions. And we're guaranteed that, to the extent the original statements correctly represent the world, so too will the conclusions be correct. On the other hand, if you start with something false then logic will lead you to conclude more false things.

3 – Situation Calculus 3 – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Norvig] So I've talked about the possibility axioms and the successor state axioms. That's most of what's in situation calculus, and that's used to describe an entire domain like the airport cargo domain. And now we describe a particular problem within that domain by describing the initial state. Typically we call that S0, the initial situation. And in S0 we can make various types of assertions of different types of predicates. So we could say that plane P1 is at airport JFK in S0, so just a simple predicate. And we could also make larger sentences, so we could say for all C, if C is cargo, then that C is at JFK in situation S0. So we have much more flexibility in situation calculus to say almost anything we want. Anything that's a valid sentence in first order logic can be asserted about the initial state. The goal state is similar. We could have a goal of saying there exists some goal state S such that for all C, if C is cargo, then we want that cargo to be at SFO in state S. So this initial state and this goal says move all the cargo—I don't care how much there is—from JFK to SFO. The great thing about situation calculus is that once we've described this in the ordinary language of first order logic, we don't need any special programs to manipulate it and come up with the solution because we already have theorem provers for first order logic and we can just state this as a problem, apply the normal theorem prover that we already had for other uses, and it can come up with an answer of a path that satisfies this goal, a situation which corresponds to a path which satisfies this given the initial state and given the descriptions of the actions. So the advantage of situation calculus is that we have the full power of first order logic. We can represent anything we want. Much more flexibility than in problem solving or classical planning. So all together now, we've seen several ways of dealing with planning. We started in deterministic, fully observable environments and we moved into stochastic and partially observable environments. We were able to distinguish between plans that can or cannot solve a problem, but we had 1 weakness in all these different approaches. It is that we weren't able to distinguish between probable and improbable solutions. And that will be the subject of the next unit.

4 – Background and Expert Systems – lang_en_vs51.srt (9. Logic and Planning Subtitles)

.....

You know, in the early days of AI, there was a lot of excitement and work on logic. Can you give us a bit of historical perspective on the topic? AI started in the 50s through the 70s with the idea that what we needed was better languages and systems for writing down logical statements. And once we had written enough of them, we should be able to do anything. And some things work great. We were able to make a program be the best chess player in the world. But other projects failed. And I think there were two main issues. One, it was a mistake to try to make statements only in Boolean logic, where everything is true or false. And that's because the world is uncertain, so the logic of probability is a better fit to the world. And we didn't get good algorithms for dealing with probability until the mid-1980s. And two, we realized it was too laborious to try to write down everything by hand. In the 1990s to the current day, more and more data was becoming available online. And now we think first of learning about the world from data, rather than writing rules by hand. And still logic has its place. Well, how is logic used today? Well there's many domains where we can eliminate much of the uncertainty of the real world and solve significant problems using logic. One example would be a company like FedEx planning all the deliveries of its packages using its huge fleet of vehicles. We can describe this as a logic problem and find a solution that minimizes time and expense. Peter will also be talking about algorithms that are useful for understanding how expert systems work. Expert systems are used throughout the world and are so common that most people don't even think about them as AI anymore. One famous example is that during the first Persian Gulf War, one superhuman decision made by one military expert system about where to place a supply base saved the United States \$2 billion. Which was more money than DARPA ever invested in artificial intelligence research. One of the things I like about this section is how you and Sebastian move from logical representations to planning algorithms to planning algorithms for when actions or even the world is uncertain. What directions do you think we will be seeing for planning algorithms in the future? Well, learning from examples would be an important area. Transfer

learning across domains, where we learn to plan one area and then execute that in another area. Interactive planning, where a human-machine team solves problems together. And a lot of the issues are, what's the best interface. Rather than just getting the right answer, being able to work together as a team. And being able to explain things in terms that humans can understand. So that the humans can add things that might have been missed in the original statement of the problem. In these videos, be on the lookout for the resolution algorithm. It is an elegant way of inferring new knowledge from a knowledge base, and is one of my favorite in this section. Graphplan, which is described in the book reading, was created by Georgia Tech's own Merrick Furst back in 1995, and made planning practical for a whole new range of problems. Finally, value iteration is a key concept for Markov decision processing. Understanding these algorithms is a good goal for this section, because they require a firm foundation in the basics that are being taught.

5 – Introduction – lang_en_vsl.srt (9. Logic and Planning Subtitles)

.....

Welcome back. So far we've talked about AI as managing complexity and uncertainty. We've seen how a search can discover sequences of actions to solve problems. We've seen how probability theory can represent in reason with uncertainty. And we've seen how machine learning can be used to learn and improve. AI is a big and dynamic field because we are pushing against complexity in at least 3 directions. First, in terms of agent design, we start with a simple reflex-based agent and move into goal-based and utility-based agents. Secondly, in terms of the complexity of the environment, we start with simple environments and then start looking at partial observability, stochastic actions at multiple agents, and so on. And finally, in terms of representation, the agents model of the world becomes increasingly complex. And this unit will concentrate on that third aspect of representation, showing how the tools of logic can be used by an agent to better model the world.

6 – Propositional Logic – lang_en_vsl.srt (9. Logic and Planning Subtitles)

.....

The first logic we will consider is called propositional logic. Let's jump right into an example, recasting the alarm problem in propositional logic. We have propositional symbols B, E, A, M, and J corresponding to the events of a burglary occurring, of the earthquake occurring, of the alarm going off, of Mary calling, and of John calling. And just as in the probabilistic models, these can be either true or false, but unlike probability, our degree of belief in propositional logic is not a number. Rather, our belief is that each of these is either true or false or unknown. Now, we can make logical sentences using these symbols and also using the logical constants true and false by combining them together using logical operators. For example, we can say that the alarm is true whenever the earthquake or burglary is true with this sentence. $(E \vee B) \rightarrow A$ or B implies A. So that says whenever the earthquake or the burglary is true, then the alarm will be true. We use this \vee symbol to mean or and a right arrow to mean implies. We could also say that it would be true that both John and Mary call when the alarm is true. We write that as $A \rightarrow (J \wedge M)$ and we use this symbol \wedge to indicate an and, so that this upward-facing wedge looks kind of like an A with the crossbar missing, and so you can remember A is for "and" where with this downward-facing V symbol is the opposite of and, so that's the symbol for or. Now, there's 2 more connectors we haven't seen yet. There's a double arrow for equivalent, also known as a biconditional, and a not sign for negation, so we could say if we wanted to that John calls if and only if Mary calls. We would write that as $J \leftrightarrow M$. John is equivalent to Mary—when one is true, the other is true; when one is false, the other is false. Or we could say that when John calls, Mary doesn't, and vice versa. We could write that as John is equivalent to not Mary, and this is the not sign. Now, how do we know what the sentences mean? A propositional logic sentence is either true or false with respect to a model of the world. Now, a model is just a set of true/false values for all the propositional symbols, so a model might be the set B is true, E is false, and so on. We can define the truth of the sentence in terms of the truth of the symbols with respect to the models using truth tables.

7 – Truth Tables – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Male narrator] Here are the truth tables for all the logical connectives. What a truth table does is list all the possibilities for the propositional symbols, so P and Q can be false and false, false and true, true and false, or true and true. Those are the only 4 possibilities, and then for each of those possibilities, the truth table lists the truth value of the compound sentence. So the sentence not P is true when P is false and false when P is true. The sentence P and Q is true only when both P and Q are true and false otherwise. The sentence P or Q is true when either P or Q is true and false when both are false. Now, so far, those mostly correspond to the English meaning of those sentences with one exception, which is that in English, the word “or” is somewhat ambiguous between the inclusive and exclusive or, and this “or” means either or both. We translate this mark into English P implies Q; or as if P, then Q, but the meaning in logic is not quite the same as the meaning in ordinary English. The meaning in logic is defined explicitly by this truth table and by nothing else, but let’s look at some examples in ordinary English. If we have the proposition O and have that mean 5 is an odd number and P meaning Paris is the capital of France, then under the ordinary model of the truth in the real world, what could we say about the sentence O implies P? That is, 5 is an odd number implies Paris is the capital of France. Would that be true or false? And let’s look at one more example. If E is the proposition that 5 is an even number and M is the proposition that Moscow is the capital of France, what about E implies M? 5 is an even number implies Moscow is the capital of France. Is that true or false?

8 – Truth Tables Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Male narrator] The answers are first, the sentence if 5 is an odd number, then Paris is the capital of France, is true in propositional logic. It may sound odd in ordinary English, but in propositional logic, this is the same as true implies true and if we look on this line—the final line for P and Q, P implies Q is true. The second sentence, 5 is an even number, implies Moscow is the capital of France. That’s the same as false implies false, and false implies false according to the definition is also true.

9 – Truth Table Question – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Male narrator] Here’s a quiz. Use truth tables or whatever other method you want to fill in the values of these tables. For each of the values of P and Q—false/false, false/true, true/false, or true/true— look at each of these boxes and click on just the boxes in which the formula for that column will be true. So which of these 4 boxes, if any, will this formula be true, and this formula and this formula?

10 – Truth Table Question Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Male narrator] Here are the answers. For P and P implies Q, we know that P is true in these bottom 2 cases, and P implies Q, we saw the truth table for P implies Q is true in the first, second, and fourth case. So the only case that’s true for both P and P implies Q is the fourth case. Now, this formula, not the quantity, not P or not Q, can work that out to be the same as P and Q, and we know that P and Q is true only when both are true, so that would be true only in the fourth case and none of the other cases. And now, we’re asking for an equivalent or biconditional between these 2 cases. Is this one the same as this one? And we see that it is the same because they match up in all 4 cases. They’re false for each of the first 3 and true in the fourth one, so that means that this is going to be true no matter what. They’re always equivalent, either both false or both true, and so we should check all 4 boxes.

11 – Propositional Logic Question – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Male narrator] Here's one more example of reasoning in propositional logic. In a particular model of the world, we know the following 3 sentences are true. E or B implies A, A implies J and M, and B. We know those 3 sentences to be true, and that's all we know. Now, I want you to tell me for each of the 5 propositional symbols, is that symbol true or false, or unknown in this model, and tell me for the symbols E, B, A, J, and M.

12 – Propositional Logic Question Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

The answer is that B is true. And we know that because it was one of the 3 sentences that was given to us. And now, according to the first sentence, says that if E or B is true then A is true. So now we know that A is true. And the second sentence says if A is true then J and M are true. What about E? That wasn't mentioned. Does that mean E is false? No. It means that it is unknown that a model where E is true and a model where E is false would both satisfy these 3 sentences. So we mark E as unknown.

13 – Terminology – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

Now for a little more terminology. We say that a valid sentence is one that is true in every possible model, for every combination of values of the propositional symbols. And a satisfiable sentence is one that is true in some models, but not necessarily in all the models. So what I want you to do is tell me for each of these sentences, whether it is valid, satisfiable but not valid, or unsatisfiable, in other words, false for all models. And the sentences are P or not P, P and not P, P or Q or P is equivalent to Q, P implies Q or Q implies P. And finally, Food implies Party or Drinks implies party implies Food and Drinks implies Party.

14 – Terminology Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

The answers are P and not P is valid. That is, it's true when P is true because of this, and it's true when P is false because of this clause. P and not P is unsatisfiable. A symbol can't be both true and false at the same time. P or Q or P is equivalent to Q is valid. So we know that it's true when either P or Q is true, so that's 3 out of the 4 cases. In the fourth case, both P and Q are false, and that means P is equivalent to Q. And therefore, in all 4 cases, it's true. P implies Q or Q implies P, that's also valid. Now in ordinary English that wouldn't be valid. If the 2 clauses or the 2 symbols P and Q were irrelevant to each other we wouldn't say that either one of those was true. But in logic, one or the other must be true, according to the definitions of the truth tables. And finally, this one's more complicated, if Food then Party or if Drinks then Party implies if Food and Drinks then Party. You can work it all out and both sides of the main implication work out to be equivalent to Not Food or Not Drinks or Party. So that's the same as saying P implies P, saying one side is equivalent to the other side. And if they're equivalent, then the implication relation holds.

15 – Propositional Logic Limitations – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

Propositional logic. It's a powerful language for what it does. And there are very efficient inference mechanisms for determining validity and satisfiability, although we haven't discussed them. But propositional logic has a few limitations. First, it can only handle true and false values. No capability to handle uncertainty like we did in probability theory. And second, we can only talk about events that are true or false in the world. We can't talk about objects that have properties, such as size, weight, color, and so on. Nor can we talk about the relations between

objects. And third, there are no shortcuts to succinctly talk about a lot of different things happening. Say if we had a vacuum world with a thousand locations, and we wanted to say that every location is free of dirt. We would need a conjunction of a thousand propositions. There's no way to have a single sentence saying that all the locations are clean all at once. So, we will next cover first-order logic which addresses these two limitations.

16 – First Order Logic – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Norvig] I'm going to talk about first order logic and its relation to the other logics we've seen so far— namely, propositional logic and probability theory. We're going to talk about them in terms of what they say about the world, which we call the ontological commitment of these logics, and what types of beliefs agents can have using these logics, which we call the epistemological commitments. So in first order logic we have relations about things in the world, objects, and functions on those objects. And what we can believe about those relations is that they're true or false or unknown. So this is an extension of propositional logic in which all we had was facts about the world and we could believe that those facts were true or false or unknown. In probability theory we had the same types of facts as in propositional logic— the symbols or variables—but the beliefs could be a real number in the range 0 to 1. So logics vary both in what you can say about the world and what you can believe about what's been said about the world. Another way to look at representation is to break the world up into representations that are atomic, meaning that a representation of the state is just an individual state with no pieces inside of it. And that's what we used for search and problem solving. We had a state, like state A, and then we transitioned to another state, like state B, and all we could say about those states was are they identical to each other or not and maybe is one of them a goal state or not. But there wasn't any internal structure to those states. In propositional logic, as well as in probability theory, we break up the world into a set of facts that are true or false, so we call this a factored representation— that is, the representation of an individual state of the world is factored into several variables—the B and E and A and M and J, for example— and those could be Boolean variables or in some types of representations— not in propositional logic—they can be other types of variables besides Boolean. Then the third type—the most complex type of representation—we call structured. And in a structured representation, an individual state is not just a set of values for variables, but it can include relationships between objects, a branching structure, and complex representations and relations between one object and another. And that's what we see in traditional programming languages, it's what we see in databases—they're called structured databases, and we have structured query languages over those databases— and that's a more powerful representation, and that's what we get in first order logic.

17 – Models – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Norvig] How does first order logic work? What does it do? Like propositional logic, we start with a model. In propositional logic a model was a value for each propositional symbol. So we might say that the symbol P was true and the symbol Q was false, and that would be a model that corresponds to what's going on in a possible world. In first order logic the models are more complex. We start off with a set of objects. Here I've shown 4 objects, these 4 tiles, but we could have more objects than that. We could say, for example, that the numbers 1, 2, and 3 were also objects in our model. So we have a set of objects. We can also have a set of constants that refer to those objects. So I could use the constant names A, B, C, D, 1, 2, 3, but I don't have to have a one-to-one correspondence between constants and objects. I could have 2 different constant names that refer to the same object. I could also have, say, the name C that refers to this object, or I could have some of the objects that don't have any names at all. But I've got a set of constants, and I also have a set of functions. A function is defined as a mapping from objects to objects. And so, for example, I might have the Number Of function that maps from a tile to the number on that tile, and that function then would be defined by the mapping from A to 1 and B to 3 and C to 3 and D to 2, and I could have other functions as well. In addition to functions, I can have relations. For example, I could have the Above relation,

and I could say in this model of the world the Above relation is a set of tuples. Say A is above B and C is above D. So that was a binary relation holding between 2 objects. Say 1 block is above another block. We can have other types of relations. For example, here is a unary relation–vowel– and if we want to say the relation Vowel is true only of the object that we call A, then that’s a set of tuples of length 1 that contains just A. We can even have relations over no objects. Say we wanted to have the relation Rainy, which doesn’t refer to any objects at all but just refers to the current situation. Then since it’s not rainy today, we would represent that as the empty set. There’s no tuples corresponding to that relation. Or, if it was rainy, we could say that it’s represented by a singleton set, and since the arity of Rainy is 0, there would be 0 elements in each one of those tuples. So that’s what a model in first order logic looks like.

18 – Syntax – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Man] Now let’s talk about the syntax of first order logic, and like in propositional logic, we have sentences which describe facts that are true or false. But unlike propositional logic, we also have terms which describe objects. Now, the atomic sentences are predicates corresponding to relations, so we can say vowel (A) is an atomic sentence or above (A, B). And we also have a distinguished relation–the equality relation. We can say $2 = 2$ and the equality relation is always in every model, and sentences can be combined with all the operators from propositional logic so that’s and, or, not, implies, equivalent, and parentheses. Now, terms, which refer to objects, can be constants, like A, B, and 2. They can be variables. We normally use lowercase, like x and y. And they can be functions, like number of A, which is just another name or another expression that refers to the same object as 1, at least in the model that we showed previously. And then, there’s 1 more type of complex sentence besides the sentences we get by combining operators, that makes first order logic unique, and these are the quantifiers. And there are two quantifiers for all, which we write with an upside-down A followed by a variable that it introduces and there exists, which we write with an upside-down E followed by the variable that it introduces. So for example, we could say for all x, if x is a vowel, then the number of (x) is equal to 1, and that’s the valid sentence in first order logic. Or we could say there exists in x such that the number of (x) is equal to 2, and this is saying that there’s some object in the domain to which the number of function applies and has a value of 2, but we’re not saying what that object is. Now, another note is that sometimes as an abbreviation, we’ll omit the quantifier, and when we do that, you can just assume that it means for all; that’s left out just as a shortcut. And I should say that these forms, or these sentences are typical, and you’ll see these form over and over again, so typically, whenever we have a “for all” quantifier introduced, it tends to go with a conditional like vowel of (x) implies number of (x) = 1, and the reason is because we usually don’t want to say something about every object in the domain, since the objects can be so different, but rather, we want to say something about a particular type of object, say, in this case, vowels. And also, typically, when we have an exists an x, or an exists any variable, that typically goes with just a form like this, and not with a conditional, because we’re talking about just 1 object that we want to describe.

19 – Vacuum World – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[man]

Now let’s go back to the 2-location vacuum world and represent it in first order logic. So first of all, we can have locations. We can call the left location A and the right location B and the vacuum V, and the dirt–say, D1 and D2. Then, we can have relations. The relation loc, which is true of any location; vacuum, which is true of the vacuum; dirt, which is true of dirt; and at, which is true of an object and a location. And so if we wanted to say the vacuum is at location A, we just say at (V, A). If we want to say there’s no dirt in any location, it’s a little bit more complicated. We can say for all dirt and for all locations, if D is a dirt, and L is a location, then D is not at L. So that says there’s no dirt in any location. Now, note if there were thousands of locations instead of just 2, this sentence would still hold, and that’s really the power of first order logic. Let’s keep going and try some more

examples. If I want to say the vacuum is in a location with dirt without specifying what location it's in, I can do that. I can say there exists an L and there exists a D such that D is a dirt and L is a location and the vacuum is at the location and the dirt is at that same location. and that's the power of first order logic. Now one final thing. You might ask what "first order" means. It means that the relations are on objects, but not on relations, and that would be called "higher order." In higher order logic, we could, say, define the notion of a transitive relation talking about relations itself, and so we could say for all R, transitive of R is equivalent to for all A, B, and C; R of (A, B) and R of (B, C) implies R (A, C). So that would be a valid statement in higher order logic that would define the notion of a transitive relation, but this would be invalid in first order logic.

20 – FOL Question – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Man] Now let's get some practice in first order logic. I'm going to give you some sentences, and for each one, I want you to tell me if it is valid—that is, O is true—satisfiable, but not valid; that is, there's some models for which it is true; or unsatisfiable, meaning there are no models for which it is true. And the first sentence is there exists an x and a y such that $x = y$. Second sentence: there exists an x such that $x = x$, implies for all y there exists a z such that $y = z$. Third sentence: for all x, p of x or not p of x. And fourth: there exists an x, P of x.

21 – FOL Question Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Man] The answers are the first sentence is valid. It's always true. Why is that? Because every model has to have at least 1 object and we can have both x and y refer to that same object, and so that object must be equal to itself. Second, let's see. The left-hand side of this implication has to be true. X is always equal to x, and the right-hand side says for every y, does there exist a z such that y equals z? And we can say yes, there is. We can always choose y itself for the value of z, and then $y = y$, so true implies true. That's always true. Valid. Third sentence: for all x, P of x or not P of x, and that's always true because everything has to be either in the relation for P or out of the relation for P, so that's valid. And the fourth: there exists an x, P of x, and that's true for the models in which there is some x that is a member of P, but it doesn't necessarily have to be any at all. P might be an empty relation, so this is satisfiable. True in some models, but not true in all models.

22 – FOL Question 2 – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Man] Now I'm going to give you some sentences or axioms in first order logic, and I want you to tell me if they correctly or incorrectly represent the English that I'm asking about. So tell me yes or no, are these good representations? And the first, I want to represent the English sentence "Sam has 2 jobs," and the first order logic sentence is there exists an x and y such that job of Sam x and job of Sam y and not $x = y$. And so tell me yes, that correctly represents Sam has 2 jobs, or no, there's a problem. And secondly, I want to represent the idea of set membership. Now, assume I've already defined the notion of adding an element to a set. Can I define set membership with these 2 axioms? For all x and s, x is a member of the result of adding x to any set s, and for all x and s, x is a member of s implies that for all y, x is a member of the set that you get when you add y to s. And third, I'm going to try to define the notion of adjacent squares on, say, a checkerboard, where the squares are numbered with x and y coordinates and we want to just talk about adjacency in the horizontal and vertical direction. Can I define that as follows? For all x and y, the square x, y is adjacent to the square $+(x, 1)$, y, and the square (x, y) is adjacent to the square (x, $+(y, 1)$) and assume that we've defined the notion of + somewhere and that the character set allows + to occur as the character for a function. Tell me yes or no, is that a good representation of the notion of adjacency?

23 – FOL Question 2 Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Man] The first answer is yes, this is a good representation of the sentence “Sam has 2 jobs.” It says there exists an x and y , and one of them is a job of Sam. The other one is a job of Sam, and crucially, we have to say that x is not equal to y . Otherwise, this would be satisfied and we could have the same job represented by the variables x and y . Is this a good representation of the member function? No. It does do a good job of telling you what is a member, so if x is a member of a set because it’s one member and then we can always add other members and it’s still a member of that set, but it doesn’t tell you anything about what x is not a member of. So for example, we want to know that 3 is not a member of the empty set, but we can’t prove that with what we have here. And we have a similar problem down here. This is not a good representation of adjacent relation. So it will tell you, for example, that square (1,1) is adjacent to square (2,1) and also to square (1,2). So it’s doing something right, but one problem is that it doesn’t tell you in the other direction. It doesn’t tell you that (2,1) is adjacent to (1,1) and another problem is that it doesn’t tell you that (1,1) is not adjacent to (8,9) because again, there’s no way to prove the negative. And the moral is that when you’re trying to do a definition, like adjacent or member, what you usually want to do is have a sentence with the equivalent or the biconditional sign to say this is true if and only if rather than to just have an assertion or to have an implication in one direction.

24 – Problem Solving Vs Planning – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Narrator] You remember our problem-solving work? We have a state space like this, and we’re given a start space and a goal to reach, and then we’d search for a path to find that goal, and maybe we find this path. Now the way a problem-solving agent would work is first it does all the work to figure out the path to the goal just doing by thinking, and then it starts to execute that path to drive or walk, however you want to get there, from the start state to the end state, but think about what would happen if you did that in real life; if you did all your planning ahead of time, you had the complete goal, and then without interacting with the world, without sensing it at all, you started to execute that path. Well this has, in fact, been studied. People have gone out and blindfolded walkers, put them in a field and told them to walk in a straight line, and the results are not pretty. Here are the GPS tracks to prove it. So we take a hiker, we put him at a start location, say here, and we blindfold him so that he can’t see anything in the horizon, but just has enough to see his or her feet so that they won’t stumble over something, and tell them execute the plan of going forward. Put one foot in front of each other and walk forward in a straight line, and these are the typical paths we see. They start out going straight for awhile but then go in loop de loops and end up not at a straight path at all. These ones over here, starting in this location, are even more convoluted. They get going straight for a little bit and then go in very tight loops. So people are incapable of walking a straight line without any feedback from the environment. Now here on this yellow path, this one did much better, and why was that? Well it’s because these paths were on overcast days, and so there was no input to make sense of. Whereas on this path was on a very sunny day, and so even though the hiker couldn’t see farther than a few feet in front of him, he could see shadows and say, “As long as I keep the shadows pointing in the right direction then I can go in a relatively straight line.” So the moral is we need some feedback from the environment. We can’t just plan ahead and come up with a whole plan. We’ve got to interleave planning and executing.

25 – Planning Vs Execution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Narrator] Now why do we have to interleave planning and execution? Mostly because of properties of the environment that make it difficult to deal with. The most important one is if the environment is stochastic. That is if we don’t know for sure what an action is going to do. If we know what everything is going to do, we can plan it out right from the start, but if we don’t, we have to be able to deal with contingencies of say I tried to move forward, and the wheels slipped, and I went someplace else, or the brakes might skid, or if we’re walking our feet don’t go

100% straight, or consider the problem of traffic lights. If the traffic light is red, then the result of the action of go forward through the intersection is bound to be different than if the traffic light is green. Another difficulty we have to deal with is multi-agent environments. If there are other cars and people that can get in our way, we have to plan about what they're going to do, and we have to react when they do something unexpected, and we can only know that at execution time, not at planning time. The other big problem is with partial observability. Suppose we've come up with a plan to go from A to S to F to B. That plan looks like it will work, but we know that at S, the road to F is sometimes closed, and there will be a sign there telling us whether it's closed or not, but when we start off, we can't read that sign. So that's partial observability. Another way to look at it is when we start off we don't know what state we're in. We know we're in A, but we don't know if we're in A in the state where the road is closed or if we're in A in the state where the road is open, and it's not until we get to S that we discover what state we're actually in, and then we know if we can continue along that route or if we have to take a detour south. Now in addition to these properties of the environment, we can also have difficulty because of lack of knowledge on our own part. So if some model of the world is unknown, that is, for example, we have map or GPS software that's inaccurate or incomplete, then we won't be able to execute a straight-line plan, and, similarly, often we want to deal with a case where the plans have to be hierarchical. And, certainly, a plan like this is at a very high level. We can't really execute the action of going from A to S when we're in a car. All the actions that we can actually execute are things like turn the steering wheel a little bit to the right, press on the pedal a little bit more. So those are the low-level steps of the plan, but those aren't sketched out in detail when we start, when we only have the high-level parts of the plan, and then it's during execution that we schedule the rest of the low-level parts of the plan. Now most of these difficulties can be addressed by changing our point of view. Instead of planning in the space of world states, we plan in the space of belief states. To understand that let's look at a state.

26 – Vacuum Cleaner Example – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Narrator] Here's a state space diagram for a simple problem. It involves a room with 2 locations. The left we call A, and the right we call B, and in that environment there's a vacuum cleaner, and there may or may not be dirt in either of the 2 locations, and so that gives us 8 total states. Dirt is here or not, here or not, and the vacuum cleaner is here or here. So that's 2 times 2 times 2 is 8 possible states, and I've drawn here the states based diagram with all the transitions for the 3 possible actions, and the actions are moving right. So we'd go from this state to this state. Moving left, we'd go from this state to this state, and sucking up dirt, we'd go from this state to this state for example, and in this state space diagram, if we have a fully deterministic, fully observable world, it's easy to plan. Say we start in this state, and we want to be— end up in a goal state where both sides are clean. We can execute the suck-dirt action and get here and then move right, and then suck dirt again, and now we end up in a goal state where everything is clean. Now suppose our robot vacuum cleaner's sensors break down, and so the robot can no longer perceive either which location it's in or whether there's any dirt. So we now have an unobservable or sensorless world rather than a fully observable one, and how does the agent then represent the state of the world? Well it could be in any one of these 8 states, and so all we can do to represent the current state is draw a big circle or box around everything, and say, "I know I'm somewhere inside here." Now that doesn't seem like it helps very much. What good is it to know that we don't really know anything at all? But the point is that we can search in the state space of the least states rather than in the state space of actual spaces. So we believe that we're in 1 of these 8 states, and now when we execute an action, we're going to get to another belief state. Let's take a look at how that works.

27 – Sensorless Vacuum Cleaner Problem – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Narrator] This is the belief state space for the sensor-less vacuum problem. So we started off here. We drew the circle around this belief state. So we don't anything about where we are, but the amazing thing is, if we execute actions, we can gain knowledge about the world even without sensing. So let's say we move right, then we'll know we're in the right-hand location. Either we were in the left, and we moved right and arrived there, or we were in the right to begin with, and we bumped against the wall and stayed there. So now we end up in this state. We now know more about the world. We're down to 4 possibilities rather than 8, even though we haven't observed anything, and now note something interesting, that in the real world, the operations of going left and going right are inverses of each other, but in the belief state world going right and going left are not inverses. If we go right, and then we go left, we don't end up back where we were in a state of total uncertainty, rather going left takes us over here where we still know we're in 1 of 4 states rather than in 1 of 8 states. Note that it's possible to form a plan that reaches a goal without ever observing the world. Plans like that are called conform-it plans. For example, if the goal is to be in a clean location all we have to do is suck. So we go from one of these 8 states to one of these 4 states and, every one of those 4, we're in a clean location. We don't know which of the 4 we're in, but we know we've achieved the goal. It's also possible to arrive at a completely known state. For example, if we start here, we go left; we suck up the dirt there. If we go right and suck up the dirt, now we're down to a belief state consisting of 1 single state that is we know exactly where we are. Here's a question for you: How do I get from the state where I know my current square is clean, but know nothing else, to the belief state where I know that I'm in the right-hand side location and that that location is clean? What I want you to do is click on the sequence of actions, left, right, or suck that will take us from that start to that goal.

28 – Sensorless Vacuum Cleaner Problem Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Narrator] And the answer is that the state of knowing that you're current square is clean corresponds to this state. This belief state with 4 possible world states, and if I then execute the right action, followed by the suck action, then I end up in this belief state, and that satisfies the goal. I know I'm in the right-hand-side location and I know that location is clean.

29 – Partially Observable Vacuum Cleaner Example – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Narrator] We've been considering sensor-less planning in a deterministic world. Now I want to turn our attention to partially observable planning but still in a deterministic world. Suppose we have what's called local sensing, that is our vacuum can see what location it is in and it can see what's going on in the current location, that is whether there's dirt in the current location or not, but it can't see anything about whether there's dirt in any other location. So here's a partial diagram of the— part of the belief state from that world, and I want it to show how the belief state unfolds as 2 things happen. First, as we take action, so we start in this state, and we take the action of going right, and in this case we still go from 2 world states in our belief state to 2 new ones, but then, after we do an action, we do an observation, and we have the act precept cycle, and now, once we get the observation, we can split that world, we can split our belief state to say, "If we observe that we're in location B and it's dirty, then we know we're in this belief state here, which happens to have exactly 1 world state in it, and if we observe that we're clean then we know that we're in this state, which also has exactly 1 in it. Now what is the act-observe cycle do to the sizes of the belief states? Well in a deterministic world, each of the individual world states within a belief state maps into exactly 1 other one. That's what we mean by deterministic, and so that means the size of the belief state will either stay the same or it might decrease if 2 of the actions sort of accidentally end up bringing you to the same place. On the other hand, the observation works in kind of the opposite way. When we observe the world, what we're doing is we're taking the current belief state and partitioning it up into pieces. Observations alone can't introduce a new state—a new world state into the belief state. All they can do is say, "Some of them go here and some of them go

here.” Now maybe that for some observation all the belief states go into 1 bin, and so we make an observation that we don’t learn anything new, but at least the observation can’t make us more confused than we were before the observation.

30 – Stochastic Environment Problem – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Norvig] Now let’s move on to stochastic environments. Let’s consider a robot that has slippery wheels so that sometimes when you make a movement—a left or a right action—the wheels slip and you stay in the same location. And sometimes they work and you arrive where you expected to go. And let’s assume that the suck action always works perfectly. We get a belief state space that looks something like this. Notice that the results of actions will often result in a belief state that’s larger than it was before—that is, the action will increase uncertainty because we don’t know what the result of the action is going to be. And so here for each of the individual world states belonging to a belief state, we have multiple outcomes for the action, and that’s what stochastic means. And so we end up with a larger belief state here. But in terms of the observation, the same thing holds as in the deterministic world. The observation partitions the belief state into smaller belief states. So in a stochastic partially observable environment, the actions tend to increase uncertainty, and the observations tend to bring that uncertainty back down. Now, how would we do planning in this type of environment? I haven’t told you yet, so you won’t know the answer for sure, but I want you to try to figure it out anyways, even if you might get the answer wrong. Imagine I had the whole belief state from which I’ve diagrammed just a little bit here and I wanted to know how to get from this belief state to one in which all squares are clean. So I’m going to give you some possible plans, and I want you to tell me whether you think each of these plans will always work or maybe sometimes work depending on how the stochasticity works out. Here are the possible plans. Remember I’m starting here, and I want to know how to get to a belief state in which all the squares are clean. One possibility is suck right and suck, one is right suck left suck, one is suck right right suck, and the other is suck right suck right suck. So some of these actions might take you out of this little belief state here, but just use what you knew from the previous definition of the state space and the results of each of those actions and the fact that the right and left actions are nondeterministic and tell me which of these you think will always achieve the goal or will maybe achieve the goal. And then I want you to also answer for the fill-in-the-blank plan— that is, is there some plan, some ideal plan, which always or maybe achieves the goal?

31 – Stochastic Environment Problem Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

And the answer is that any plan that would work in the deterministic world might work in the stochastic world if everything works out okay and all of these plans meet that criteria. But no finite plan is guaranteed to always work because a successful plan has to include at least 1 move action. And if we try a move action a finite number of times, each of those times, the wheels might slip, and it won’t move, and so we can never be guaranteed to achieve the goal with a finite sequence of actions. Now, what about an infinite sequence of actions? Well, we can’t represent that in the language we have so far where a plan is a linear sequence. But we can introduce a new notion of plans in which we do have infinite sequences.

32 – Infinite Sequences – lang_en_vs1.srt (9. Logic and Planning Subtitles)

In this new notation, instead of writing plans as a linear sequence of, say, suck, move right, and suck, I’m going to write them as a tree structure. We start off in this belief state here, which we’ll diagram like this. And then we do a suck action. We end up in a new state. And then we do a right action, and now we have to observe the world, and if we observe that we’re still in state A, we loop back to this part of the plan. And if we observe that we’re in B, we go on and then execute the suck action. And now we’re at the end of the plan. So, we see that there’s a choice point here, which we indicate with this sort of tie to say we’re following a straight line, but now we can branch. There’s a

conditional, and we can either loop, or we can continue on, so we see that this finite representation represents an infinite sequence of plans. We could write it in a more sort of linear notation as S, while we observe A, do R, and then do S. Now, what can we say about this plan? Does this plan achieve the goal? Well, what we can say is that if the stochasticity is independent, that is, if sometimes it works and sometimes it doesn't, then with probability 1 in the limit, this plan will, in fact, achieve the goal, but we can't state any bounded number of steps under which it's guaranteed to achieve the goal. We can only say it's guaranteed at infinity.

33 – Finding A Successful Plan – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

Now, I've told you what a successful plan looks like, but I haven't told you how to find one. The process of finding it can be done through search just as we did in problem solving. So, remember in problem solving, we start off in a state, and it's a single state, not a belief state. And then we start searching a tree, and we have a big triangle of possible states that we search through, and then we find one path that gets us all the way to a goal state. And we pick from this big tree a single path. So, with belief states and with branching plan structures, we do the same sort of process, only the tree is just a little bit more complicated. Here we show one of these trees, and it has different possibilities. For example, we start off here, and we have one possibility that the first action will be going right, or another possibility that the first action will be performing a suck. But then it also has branches that are part of the plan itself. This branch here is actually part of the plan as we saw before. It's not a branch in the search space. It's a branch in the plan, so what we do is we search through this tree. We try right as a first action. We try suck as a first action. We keep expanding nodes until we find a portion of the tree like this path is a portion of this search tree. We find that portion which is a successful plan according to the criteria of reaching the goal.

34 – Finding A Successful Plan Question – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

Let's say we performed that search. We had a big search tree, and then we threw out all the branches except one, and this branch of the search tree does itself have branches, but this branch of the search tree through the belief state represents a single plan, not multiple possible plans. Now, what I want to know is, for this single plan, what can we guarantee about it? So, say we wanted to know is this plan guaranteed to find the goal in an unbounded number of steps? And what do we need to guarantee that? So, it's an unbounded solution. Do we need to guarantee that some leaf node is a goal? So, for example, here's a plan to go through, and at the bottom, there's a leaf node. Now, if this were in problem solving, then remember, it would be a sequence of steps with no branches in it, and we know it's a solution if the one leaf node is a goal. But for these with branches, do we need to guarantee that some leaf is a goal, or do we need to guarantee that every leaf is a goal, or is there no possible guarantee that will mean that for sure we've got a solution, although the solution may be of unbounded length? Then I also want you to answer what does it take to guarantee that we have a bounded solution? That is, a solution that is guaranteed to reach the goal in a bounded, finite number of steps. Do we need to have a plan that has no branches in it, like this branch? Or a plan that has no loops in it, like this loop that goes back to a previous state? Or is there no guarantee that we have a bounded solution?

35 – Finding A Successful Plan Question Solution – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

And the answer is we have an unbounded solution if every leaf in the plan ends up in a goal. So, if we follow through the plan, no matter what path we execute based on the observations— and remember, we don't get to pick the observations. The observations come into us, and we follow one path or another based on what we observe. So, we can't guide it in one direction or another, and so we need every possible leaf node. This one only has one, but if a plan had multiple leaf nodes, every one of them would have to be a goal. Now, in terms of a bounded solution, it's

okay to have branches but not to have loops. If we had branches and we ended up with one goal here and one goal here in 1, 2, 3, steps, 1, 2, 3, steps, that would be a bounded solution. But if we have a loop, we might be 1, 2, 3, 4, 5– we don't know how many steps it's going to take.

36 – Problem Solving Via Mathematical Notation – lang_en_vs1.srt (9. Logic and Planning Subtitles)

Now, some people like manipulating trees and some people like a more-sort of formal-mathematical notation. So if you're one of those, I'm going to give you another way to think about whether or not we have a solution; and let's start with a problem-solving where a plan consists of a straight line sequence. And we said one way to decide if this is a plan that satisfies the goal is to say, "Is the end state a goal state?" If we want to be more formal and write that out mathematically, what we can say is-what this plan represents is-we started in the start state, and then we transitioned to the state that is the result of applying the action of going from A to S, to that start state; and then we applied to that, the result of starting in that intermediate state and applying the action of going from S to F. And if that resulting state is an element of the set of Goals, then this plan is valid; this plan gives us a solution. And so that's a mathematical formulation of what it means for this plan to be a Goal. Now, in stochastic partially observable worlds, the equations are a little bit more complicated. Instead of just having S Prime is a result of applying some action to the initial state, we're dealing with belief states, rather than individual states. And what we say is our new belief state is the result of updating what we get from predicting what our action will do; and then updating it, based on our observation, O, of the world. So the prediction step is when we start off in a belief state; we look at the action, we look at each possible result of the action- because they're stochastic-to each possible member of the belief state, and so that gives us a larger belief state; and then we update that belief state by taking account of the observation- and that will give us a smaller-or same size-belief state. And now, that gives us the new state. Now, we can use this to predict and update cycle to keep track of where we are in a belief state.

37 – Tracking The Predict Update Cycle – lang_en_vs1.srt (9. Logic and Planning Subtitles)

Here's an example of tracking the Predict Update Cycle; and this is in a world in which the actions are guaranteed to work, as advertised- that is, if you start to clean up the current location, and if you move right or left, the wheels actually turn; and you do move. But we can call this the kindergarten world because there are little toddlers walking around who can deposit Dirt in any location, at any time. So if we start off in this state, and execute the Suck action, we can predict that we'll end up in one of these 2 states. Then, if we have an observation-well, we know what that observation's going to be because we know the Suck action always works, and we know we were in A; so the only observation we can get is that we're in A-and that it's Clean- so we end up in that same belief state. And then, if we execute the Right action- well, then lots of things could happen; because we move Right, and somebody might have dropped Dirt in the Right location, and somebody might have dropped Dirt in the Left location-or maybe not. So we end up with 4 possibilities, and then we can update again when we get the next observation- say, if we observed that we're in B and it's Dirty, then we end up in this belief state. And we can keep on going-specifying new belief states- as a result of success of predicts and updates. Now, this Predict Update Cycle gives us a kind of calculus of belief states that can tell us, really, everything we need to know. But there is one weakness with this approach- that, as you can see here, some of the belief states start to get large; and this is a tiny little world. Already, we have a belief state with 4 world states in it. We could have one with 8, 16, 10, 24-or whatever. And it seems that there may be more succinct representations of a belief state, rather than to just list all the world states. For example, take this one here: If we had divided the world up-not into individual world states, but into variables describing that state, then this whole belief state could be represented just by: Vacuum is on the Right. So the whole world could be represented by 3 states-or 3 variables: One, where is the Vacuum-is it on the Right, or

not? Secondly, is there Dirt in the Left location? And third, is there Dirt in the Right location? And we could have some formula, over those variables, to describe states. And with that type of formulation, some very large states—in terms of enumerating the world states— can be made small, in terms of the description.

38 – Classical Planning 1 – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Norvig] I want to describe a notation which we call classical planning, which is a representation language for dealing with states and actions and plans, and it's also an approach for dealing with the problem of complexity by factoring the world into variables. So under classical planning, a state space consists of all the possible assignments to k Boolean variables. So that means they'll be 2^k states in that state space. And if we think about the 2 location vacuum world, we would have 3 Boolean variables. We could have dirt in location A, dirt in location B, and vacuum in location A. The vacuum has to be in either A or B. So these 3 variables will do, and there will be 8 possible states in that world, but they can be succinctly represented through the 3 variables. And then a world state consists of a complete assignment of true or false through each of the 3 variables. And then a belief state. Just as in problem solving, the belief state depends on what type of environment you want to deal with. In the core classical planning, the belief state had to be a complete assignment, and that was useful for dealing with deterministic fully observable domains. But we can easily extend classical planning, and we can deal with belief states that are partial assignments— that is, some of the variables have values and others don't. So we could have the belief state consisting of vacuum in A is true and the others are unknown, and that small formula represents 4 possible world states. We can even have a belief state which is an arbitrary formula in Boolean logic, and that can represent anything we want. So that's what states look like. Now we have to figure out what actions look like and what the results of those actions look like. These are represented in classical planning by something called an action schema. It's called a schema because it represents many possible actions that are similar to each other. So let's take an example of we want to send cargo around the world, and we've got a bunch of planes in airports, and we have cargo and so on. I'll show you the action for having a plane fly from one location to another. Here's one possible representation. We say it's an action schema, so we write the word Action and then we write the action operator and its arguments, so it's a Fly of P from X to Y. And then we list the preconditions, what needs to be true in order to be able to execute this action. We can say something like P better be a plane. It's no good trying to fly a truck or a submarine. And we'll use the And formula from Boolean propositional logic. X better be an airport. We don't want to try to take off from my backyard. And similarly, Y better be an airport. And, most importantly, P better be at airport X in order to take off from there. And then we represent the effects of the action by saying what's going to happen. Once we fly from X to Y, the plane is no longer at X, so we say not at P,X—the plane is no longer at X— and the plane is now at Y. This is called an action schema. It represents a set of actions for all possible planes, for all X and for all Y, represents all of those actions in one schema that says what we need to know in order to apply the action and it says what will happen. In terms of the transition from state spaces, this variable will become false and this one will become true. When we look at this formula, this looks like a term in first order logic, but we're actually dealing with a completely propositional world. It just looks like that because this is a schema. We can apply this schema to specific ground states, specific world states, and then P and X would have specific values, and you could just think of it as concatenating their names all together, and that's just the name of one variable. The name just happens to have this complex form with parentheses and commas in it to make it easier to write one schema that covers all the individual fly actions.

39 – Classical Planning 2 – lang_en_vs1.srt (9. Logic and Planning Subtitles)

[Norvig] Here we see a more complete representation of a problem solving domain in the language of classical planning. Here's the Fly action schema. I've made it a little bit more explicit with from and to airports rather than X or Y. We want to deal with transporting cargo. So in addition to flying, we have an operator to load cargo, C, onto a

plane, P, at airport A— you can see the preconditions and effects there— and an action to unload the cargo from the plane with preconditions and effects. We have a representation of the initial state. There's 2 pieces of cargo, there's 2 planes and 2 airports. This representation is rich enough and the algorithms on it are good enough that we could have hundreds or thousands of cargo planes and so on representing millions of ground actions. If we had 10 airports and 100 planes, that would be 100, 1,000, 10,000 different Fly actions. And if we had thousands of pieces of cargo, there would be even more Load and Unload actions, but they can all be represented by the succinct schema. So the initial state tells us what's what, where everything is, and then we can represent the goal state: that we want to have this piece of cargo has to be delivered to this airport, and another piece of cargo has to be delivered to this airport. So now we know what actions and problems of initial and goal state looks like in this representation, but how do we do planning using this?

40 – Progression Search – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Norvig] The simplest way to do planning is really the exact same way that we did it in problem solving. We start off in an initial state. So P1 was at SFO, say, and cargo, C1, was also at SFO, and all the other things that were in that initial state. And then we start branching on the possible actions, so say one possible action would be to load the cargo, C1, onto the plane, P1, at SFO, and then that would bring us to another state which would have a different set of state variables set, and we'd continue branching out like that until we hit a state which satisfied the goal predicate. So we call that forward or progression state space search in that we're searching through the space of exact states. Each of these is an individual world state, and if the actions are deterministic, then it's the same thing as we had before. But because we have this representation, there are other possibilities that weren't available to us before.

41 – Regression Search – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Norvig] Another way to search is called backwards or regression search in which we start at the goal. So we take the description of the goal state. C1 is at JFK and C2 is at SFO, so that's the goal state. And notice that that's the complete goal state. It's not that I left out all the other facts about the state; it's that that's all that's known about the state is that these 2 propositions are true and all the others can be anything you want. And now we can start searching backwards. We can say what actions would lead to that state. Remember in problem solving we did have that option of searching backwards. If there was a single goal state, we could say what other arcs are coming into that goal state. But here, this goal state doesn't represent a single state; it represents a whole family of states with different values for all the other variables. And so we can't just look at that, but what we can do is look at the definition of possible actions that will result in this goal. So let's look at it one at a time. Let's first look at what actions could result at C1, JFK. We look at our action schema, and there's only 1 action schema that adds an At, and that would be the Unload schema. Unload of C, P, A adds C, A. And so what we would know is if we want to achieve this, then we would have to do an Unload where the C variable would have to be C1, the P variable is still unknown—it could be any plane— and the A variable has to be JFK. Notice what we've done here. We have this representation in terms of logical formula that allows us to specify a goal as a set of many world states, and we also can use that same representation to represent an arrow here not as a single action but as a set of possible actions. So this is representing all possible actions for any plane, P, of unloading cargo at the destination. And then we can regress this state over this operator and now we have another representation of this state here. But just as this state was uncertain—not all the variables were known— this state too will be uncertain. For example, we won't know anything about what plane, P, is involved, and now we continue searching backwards until we get to a state where enough of the variables are filled in and where we match against the initial state. And then we have our solution. We found it going backwards, but we can apply the solution going forwards.

42 – Regression Vs Progression – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Norvig] Let's show an example of where a backwards search makes sense. I'm going to describe a world in which there is one action, the action of buying a book. And the precondition is we have to know which book it is, and let's identify them by ISBN number. So we can buy ISBN number B, and the effect is that we own B. And probably there should be something about money, but we're going to leave that out for now to make it simple. And then the goal would be to own ISBN number 0136042597. Now, if we try to solve this problem with forward search, we'd start in the initial state. Let's say the initial state is we don't own anything. And then we'd think about what actions can we apply. If there are 10 million different books, 10 million ISBN numbers, then there is a branching factor of 10 million coming out of this node, and we'd have to try them all in order until we happened to hit upon one that was the right one. It seems very inefficient. If we go in the backward direction, then we start at the goal. The goal is to own this number. Then we look at our available actions, and out of the 10 million actions there's only 1 action schema, and that action schema can match the goal in exactly one way, when B equals this number, and therefore we know the action is to buy this number, and we can connect the goal to the initial state in the backwards direction in just 1 step. So that's the advantage of doing backwards or regression search rather than forward search.

43 – Plan Space Search – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Norvig] There's one more type of search for plans that we can do with the classical planning language that we couldn't do before, and this is searching through the space of plans rather than searching through the space of states. In forward search we were searching through concrete world states. In backward search we were searching through abstract states in which some of the variables were unspecified. But in plan space search we search through the space of plans. And here's how it works. We start off with an empty plan. We have the start state and the goal state, and that's all we know about the plan. So obviously, this plan is flawed. It doesn't lead us from the start to the goal. And then we say let's do an operation to edit or modify that plan by adding something in new. And here we're tackling the problem of how to get dressed and put on all the clothes in the right order, so we say out of all the operators we have, we could add one of those operators into the plan. And so here we say what if we added the put on right shoe operator. Then we end up with this plan. That still doesn't solve the problem, so we need to keep refining that plan. Then we come here and say maybe we could add in the put on left shoe operator. And here I've shown the plan as a parallel branching structure rather than just as a sequence. And that's a useful thing to do because it captures the fact that these can be done in either order. And we keep refining like that, adding on new branches or new operators into the plan until we got a plan that was guaranteed to work. This approach was popular in the 1980s, but it's faded from popularity. Right now the most popular approaches have to do with forward search. We saw some of the advantages of backward search. The advantage of forward search seems to be that we can come up with very good heuristics. So we can do heuristic search, and we saw how important it was to have good heuristics to do heuristic search. And because the forward search deals with concrete plan states, it seems to be easier to come up with good heuristics.

44 – Sliding Puzzle Example – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Norvig] To understand the idea of heuristics, let's talk about another domain. Here we have the sliding puzzle domain. Remember we can slide around these little tiles and we try to reach a goal state. A 16 puzzle is kind of big, so let's show you the state space for the smaller 8 puzzle. Here is just a small portion of it. Let's figure out what the action schema looks like for this puzzle. We only need to describe one action, which is to slide a tile, T, from location A to location B. The precondition: the tile has to be on location A and has to be a tile and B has to be blank and A and B have to be adjacent. This should be an And sign, not an A. So that's the action schema. Oops. I forgot we need an effect, which should be that the tile is now on B and the blank is now on A and the tile is no longer on

A and the blank is no longer on B. We talked before about how a human analyst could examine a problem and come up with heuristics and encode those heuristics as a function that would help search do a better job. But with this kind of a formal representation we can automatically come up with good representations of heuristics. For example, if we came up with a relaxed problem by automatically going in and throwing out some of the prerequisites— if you throw out a prerequisite, you make the problem strictly easier— then you get a new heuristic. So for example, if we crossed out the requirement that B has to be blank, then we end up with the Manhattan or city block heuristic. And if we also throw out the requirement that A and B have to be adjacent, then we get the number of misplaced tiles heuristic. So that means we could slide a tile from any A to any B, no matter how far apart they were. That's the number of misplaced tiles. Other heuristics are possible. For example, one popular thing is to ignore negative effects, to say let's not say that this takes away the blank being in B. So if we ignore that negative effect, we make the whole problem strictly easier. We'd have a relaxed problem, and that might end up being a good heuristic. So because we have our actions encoded in this logical form, we can automatically edit that form. A program can do that, and the program can come up with heuristics rather than requiring the human to come up with heuristics.

45 – Situation Calculus 1 – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Norvig] Now I want to talk about 1 more representation for planning called situation calculus. To motivate this, suppose we wanted to have the goal of moving all the cargo from airport A to airport B, regardless of how many pieces of cargo there are. You can't express the notion of All in propositional languages like classical planning, but you can in first order logic. There are several ways to use first order logic for planning. The best known is situation calculus. It's not a new kind of logic; rather, it's regular first order logic with a set of conventions for how to represent states and actions. I'll show you what the conventions are. First, actions are represented as objects in first order logic, normally by functions. And so we would have a function like the function Fly of a plane and a From Airport and a To Airport which represents an object, which is the action. Then we have situations, and situations are also objects in the logic, and they correspond not to states but rather to paths— the paths of actions that we have in state space search. So if you arrive at what would be the same world state by 2 different sets of actions, those would be considered 2 different situations in situation calculus. We describe the situations by objects, so we usually have an initial situation, often called S0, and then we have a function on situations called Result. So the result of a situation object and an action object is equal to another situation. And now instead of describing the actions that are applicable in a situation with a predicate Actions of S, situation calculus for some reason decided not to do that and instead we're going to talk about the actions that are possible in the state, and we're going to do that with a predicate. If we have a predicate Possible of A and S, is an action A possible in a state? There's a specific form for describing these predicates, and in general, it has the form of some precondition of state S implies that it's possible to do action A in state S. I'll show you the possibility axiom for the Fly action. We would say if there is some P, which is the plane in state S, and there is some X, which is an airport in state S, and there is some Y, which is also an airport in state S, and P is at location X in state S, then that implies that it's possible to fly P from X to Y in state S. And that's known as the possibility axiom for the action Fly.

46 – Situation Calculus 2 – lang_en_vs1.srt (9. Logic and Planning Subtitles)

.....

[Norvig] There's a convention in situation calculus that predicates like At— we said plane P was at airport X in situation S— these types of predicates that can vary from 1 situation to another are called fluents, from the word fluent, having to do with fluidity or change over time. And the convention is that they refer to a specific situation, and we always put that situation argument as the last in the predicate. Now, the trickiest part about situation calculus is describing what changes and what doesn't change as a result of an action. Remember in classical planning we had action schemas where we described 1 action at a time and said what changed. For situation

calculus it turns out to be easier to do it the other way around. Instead of writing 1 action or 1 schema or 1 axiom for each action, we do 1 for each fluent, for each predicate that can change. We use the convention called successor state axioms. These are used to describe what happens in the state that's a successor of executing an action. And in general, a successor state axiom will have the form of saying for all actions and states, if it's possible to execute action A in state S, then—and I'll show in general what they look like here— the fluent is true if and only if action A made it true or action A didn't undo it. So that is, either it wasn't true before and A made it be true, or it was true before and A didn't do something to stop it being true. For example, I'll show you the successor state axiom for the In predicate. And just to make it a little bit simpler, I'll leave out all the For All quantifiers. So wherever you see a variable without a quantifier, assume that there's a For All. What we'll say is it's possible to execute A in situation S. If that's true, then the In predicate holds between some cargo C and some plane in the state, which is the result of executing action A in state S. So that In predicate will hold if and only if either A was a load action— so if we load the cargo into the plane, then the result of executing that action A is that the cargo is in the plane— or it might be that it was already true that the cargo was in the plane in situation S and A is not equal to an unload action. So for all A and S for which it's possible to execute A in situation S, the In predicate holds if and only if the action was a load or the In predicate used to hold in the previous state and the action is not an unload.

10. PLANNING UNDER UNCERTAINTY

1 – Introduction – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

So today is an exciting day. We'll talk about planning under uncertainty, and it really puts together from the material we've talked about in past classes. We talked about planning, but not under uncertainty, and you've had many, many classes of under uncertainty, and now it gets to the point where we can make decisions under uncertainty. This is really important for my own research field like robotics where the world is full of uncertainty, and the type of techniques I'll tell you about today will really make it possible to drive robots in actual physical roles and find good plans for these robots to execute.

2 – Planning Under Uncertainty MDP – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] Planning under uncertainty. In this class so far we talked a good deal about planning. We talked about uncertainty and probabilities, and we also talked about learning, but all 3 items were discussed separately. We never brought planning and uncertainty together, uncertainty and learning, or planning and learning. So the class today, we'll fuse planning and uncertainty using techniques known as Markov decision processes or MDPs, and partial observer Markov decision processes or POMDPs. We also have a class coming up on reinforcement learning which combines all 3 of his aspects, planning, uncertainty, and machine learning. You might remember in the very first class we distinguished very different characteristics of agent tasks, and here are some of those. We distinguished deterministic was the casting environments, and we also talked about photos as partial observable. In the area of planning so far all of our evidence falls into this field over here, like A*, depth first, right first and so on. The MDP algorithms which I will talk about first fall into the intersection of fully observable yet stochastic, and just to remind us what the difference was, stochastic is an environment where the outcome of an action is somewhat random. Whereas an environment that's deterministic where the outcome of an action is predictable and always the same. An environment is fully observable if you can see the state of the environment which means if you can make all decisions based on the momentary sensory input. Whereas if you need memory, it's partially observable.

Planning in the partially observable case is called POMDP, and towards the end of this class, I'll briefly talk about POMDPs but not in any depth. So most of this class focuses on Markov decision processes as opposed to partially observable Markov decision processes. So what is a Markov decision process? One way you can specify a Markov decision process by a graph. Suppose you have states S_1 , S_2 , and S_3 , and you have actions A_1 and A_2 . In a state transition graph, like this, is a finite state machine, and it becomes Markov if the outcomes of actions are somewhat random. So for example if A_1 over here, with a 50% probability, leads to state S_2 but with another 50% probability leads to state S_3 . So put differently, a Markov decision process of states, actions, a state's transition matrix, often written of the following form which is just about the same as a conditional state transition probability that a state is prime is the correct posterior state after executing action A in a state S , and the missing thing is the objective for the Markov decision process. What do we want to achieve? For that we often define a reward function, and for the sake of this lecture, I will attach rewards just to states. So each state will have a function R attached that tells me how good the state is. So for example it might be worth \$10 to be in the state over here, \$0 to be in the state over here, and \$100 to be in a state over here. So the planning problem is now the problem which relies on an action to each possible state. So that we maximize our total reward.

3 – Robot Tour Guide Examples – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] Before diving into too much detail, let me explain to you why MDPs really matter. What you see here is a robotic tour guide that the University of Bonn, with my assistance, deployed in the German museum in Bonn, and the objective of the this robot was to navigate the museum and guide visitors, mostly kids, from exhibit to exhibit. This is a challenging planning problem because as the robot moves it can't really predict its action outcomes because of the randomness of the environment and the carpet and the wheels of the robot. The robot is not able to really follow its own commands very well, and it has to take this into consideration during the planning process so when it finds itself in a location it didn't expect, it knows what to do. In the second video here, you see a successor robot that was deployed in the Smithsonian National Museum of American History in the late 1990s where it guided many, many thousands of kids through the entrance hall of the museum, and once again, this is a challenging planning problem. As you can see people are often in the way of the robot. The robot has to take detours. Now this one is particularly difficult because there were obstacles that were invisible like a downward staircase. So this is a challenging localization problem trying to find out where you are, but that's for a later class. In the video here, you see a robot being deployed in a nursing home with the objective to assist elderly people by guiding them around, bring them to appointments, reminding them to take their medication, and interacting with them, and this robot has been active for many, many years and been used, and, again, it's a very challenging planning problem to navigate through this elderly home. And the final robot I'm showing you here. This was built with my colleague Will Whittaker at Carnegie Melon University. The objective here was to explore abandoned mines. Pennsylvania and West Virginia and other states are heavily mined. There's many abandoned old coal mines, and for many of these mines, it's unknown what the conditions are and where exactly they are. They're not really human accessible. They tend to have roof fall and very low oxygen levels. So we made a robot that went inside and built maps of those mines. All these problems have in common that they have really challenging planning problems. The environments are stochastic. That is the outcome of actions are unknown, and the robot has to be able to react to all kinds of situations, even the ones that it didn't plan for.

4 – MDP Grid World – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] Let me give a much simpler example often called grid world for MDPs, and I'll be using this insanely simple example over here throughout this class. Let's assume we have a starting state over here, and there's 2 goal states who are often called absorbing states with very different reward or payout. Plus 100 for the state over here, minus 100 for the state over here, and our agent is able to move about the environment, and when it reaches one of

those 2 states, the game is over and the task is done. Obviously the top state is much more attractive than the bottom state with minus 100. Now to turn this into an MDP, let's assume actions are somewhat stochastic. So suppose we had a grid cell, and we attempt to go north. The deterministic agent would always succeed to go to the north square if it's available, but let's assume that we only have an 80% chance to make it to the cell in the north. If there's no cell at all, there's a wall like over here, we assume with 80% chance, we just bounce back to the same cell, but with 10% chance, we instead go left. Another 10% chance, we go right. So if an agent is over here and wishes to go north, then with 80% chance, it finds itself over here, 10% over here, 10% over here. If it goes north from here, because there's no north cell, it'll bounce back with 80% probability, 10% left, 10% right. In a cell like this one over here, it'll bounce back with 90% probability, 80 from the top and 10 from the left, but it still has a 10% chance of going right. This is a stochastic state transition which we can equally define for actions, south, west and east, and now we can see a situation like this conventional planning is insufficient. So for example if you're plan a sequence of actions starting over here, you might go north, north, east, east, east to reach our plus 100 absorbing or final state, but with this state transition model over here, even with the first step it might happen with 10% chance do you find yourselves over here, in which case conventional planning would not give us an answer. So we wish to have a planning method that provides an answer no matter where we are and that's called a policy. A policy assigns actions to any state. So for example a policy might look as follows: for this state, we wish to go north, north, east, east, east, but for this state over here, we wish to go north, maybe east over here, and maybe west over here. So each state, except for the absorbing states, we have to define an action to define a policy. The planning problem we have becomes one of finding the optimal policy

5 – Problems With Conventional Planning 1 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] To understand the beauty of a policy, let me look into stochastic environments, and let me try to apply conventional planning. Consider the same grid I just gave you, and let's assume there's a discrete start state, the one over here, and we wish to find an action sequence that leads us to the goal state over here. In conventional planning we would create a tree. In C1, we're given 4 action choices, north, south, west, and east. However, the outcome of those choices is not deterministic So rather than having a single outcome, nature will choose for us the actual outcome. In the case of going north, for example, we may find ourselves in B1, or back into C1. Similarly for going south, we might find ourselves in C1, or back in C2, and so on. This tree has a number of problems. The first problem is the branching factor. While we have 4 different action choices, nature will give us up to 3 different outcomes which makes up to 12 different things we have to follow. Now in conventional planning we might have to follow just 1 of those, but here we might have to follow up to 3 of those things. So every time we plan a step ahead, we might have to increase the breadth of the search of tree by at least a factor of 3. So one of the problem is the branching factor is too large.

6 – Branching Factor Question – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] To understand the branching factor, let me quiz you on how many states you can possibly reach from any other states, and as an example from C1, you can reach under any action choice B1, C1, and C2, but it will give you an affective branching factor of 3. So when I ask you what's the affective branching factor in B3? What is the maximum level of states you can reach under any possible action from B3? So how many states can we reach from B3 over here?

7 – Branching Factor Question Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] And the answer is 8. If you go north, you might reach this state over here, this one over here, this one over here. If you go east, you might reach this state over here, or this one over here, or this one over here, or this one over here. When you put it all together, you can reach all of those 8 states over here.

8 – Problems With Conventional Planning 2 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] There are other problems with the search paradigm. The second one is that the tree could be very deep, and the reason is we might be able to circle forever in the area over here without reaching the goal state, and that makes for a very deep tree, and until we reach the goal state, we won't even know it's the best possible action. So conventional planning might have difficulties with basically infinite loops. The third problem is that many states recur in the search. In a star, we were careful to visit each state only once, but here because the actions might carry you back here to the same state, C1 is, for example, over here and over here. You might find that many states in the tree might be visited many, many different times. Now if you had a state it doesn't really matter how you got there. Yet, the tree doesn't understand this, and it might expand states more than once. These are the 3 problems that are overcome by our policy method, and this motivates in part by calculating policies is so much better of an idea than using conventional planning and still casting environments. So let's get back to the policy case.

9 – Policy Question 1 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] Let's look at the grid world, again, and let me ask you a question. I wish to find an optimal policy for all these states that with maximum probability leads me to the absorbing state plus 100, and as I just discussed, I assume there's 4 different actions, north, south, west, and east that succeed with probability 80% provided that the corresponding grid cell is actually attainable. I wish to know what is the optimal action in the corner set over here, A1, and I give you 4 choices, north, south, west, and east.

10 – Policy Question 1 Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] And the answer is east. East in expectation transfers you to the right side, and you're one closer to your goal position.

11 – Policy Question 2 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] Let me ask the same question for the state over here, C1, which one is the optimal action for C1?

12 – Policy Question 2 Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] And the answer is north. It gets you one step closer. There is 2 equally long paths, but over here you risk falling into the minus 100; therefore, you'd rather go north.

13 – Policy Question 3 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

[Narrator] The next question is challenging. Consider state C4, which one is the optimal action provided that you can run around as long as you want. There's no costs associated with steps, but you wish to maximize the probability of ending up in plus 100 over here. Think before you answer this question.

14 – Policy Answer 3 Question 4 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

[Narrator] And the answer is south. The reason why it's south is if we attempt to go south, an 80% probability we'll stay in the same cell. In fact, a 90% probability because we can't go south and we can't go east. In a 10% probability, we find ourselves over here which is a relatively safe state because we can actually go to the left side. If we were to go just west which is the intuitive answer, then there's a 10% chance we end up in the minus 100 absorbing state. You can convince yourself if you go south, find ourselves eventually in state C3, and then go west, west, north, north, east, east, east. You will never ever run risk of falling into the minus 100, and that argument is tricky and to convince ourselves let me ask the other hard question: so what shall we do in state B3 that's the optimal action?

15 – Policy Answer 3 Question 4 Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

[Narrator] And the answer is west. If you're over here, and we go east, we'd likely end up with minus 100. If you go north, which seems to be the intuitive answer, there's a 10% chance we fall into the minus 100. However, if we go west, then there's absolutely no chance we fall into the minus 100. We might find ourselves over here. We might be in the same state. We might find ourselves over here, but from these states over here, there's safe policies that can safely avoid the minus 100.

16 – MDP And Costs – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

[Narrator] So even for the simple grid world, the optimal control policy assuming stochastic actions and no costs of moving, except for the final absorbing costs, is somewhat nontrivial. Take a second to look at this. Along here it seems pretty obvious, but for the state over here, B3, and for the state over here, C4, we choose an action that just avoids falling into the minus 100, which is more important than trying to make progress towards the plus 100. Now obviously this is not the general case of an MDP, and it's somewhat frustrating they'd be willing to run through the wall, just so as to avoid falling into the minus 100, and the reason why this seems unintuitive is because we're really forgetting the issue of costs. In normal life, there is a cost associated with moving. MDPs are gentle enough to have a cost factor, and the way we're going to denote costs is by defining our award function over any possible state. We are reaching the state A4, gives us plus 100, minus 100 for B4, and perhaps minus 3 for every other state, which reflects the fact that if you take a step somewhere that we will pay minus 3. So this gives an incentive to shorten the final action sequence. So we're now ready to state the actual objective of an MDP which is to minimize not just the momentary costs, but the sum of all future rewards, but you're going to write RT to denote the fact that this reward has received time T , and because our reward itself is stochastic, we have to complete the expectation over those, and that we seek to maximize. So we seek to find the policy that maximizes the expression over here. Now another interesting caveat is a sentence people put a so called discount factor into this equation with an exponent of T , where a discount factor was going to be 0.9, and what this does is it decays future reward relative to more immediate rewards, and it's kind of an alternative way to specify costs. So we can make this explicit by a negative reward per state or we can bring in a discount factor that discounts the plus 100 by the number of steps that it went by before it reached the plus 100. This also gives an incentive to get to the goal as fast as possible. The nice mathematical thing about discount factor is it keeps this expectation bounded. It easy to show that this expression over here will always be smaller or equal to $1 / (1 - \gamma)$ times the absolute reward maximizing value and which in this case would be plus 100.

17 – Value Iteration 1 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

The definition of the expected sum of future possible discounted reward that it has given you allows me to define a value function. For each status, my value of the state is the expected sum of future discounted reward provided that I start in state S , then I execute policy π . This expression looks really complex, but it really means something really simple, which is suppose we start in the state over here, and you get +100 over here, -100 over here. And suppose for now, every other state costs you -3. For any possible policy that assigns actions to the non-absorbing states, you can now simulate the agent for quite a while and compute empirically what is the average reward that is being received until you finally hit a goal state. For example, for the policy that you like, the value would, of course, for any state depend on how much you make progress towards the goal, or whether you bounce back and forth. In fact, in this state over here, you might bounce down and have to do the loop again. But there's a well defined expectation over any possible execution of the policy π that is generic to each state and each policy π . That's called a value. And value functions are absolutely essential to MDP, so the way we're going to plan is we're going to iterate and compute value functions, and it will turn out that by doing this, we're going to find better and better policies as well.

18 – Value Iteration 2 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

Before I dive into mathematical detail about value functions, let me just give you a tutorial. The value function is a potential function that leads from the goal location—in this case, the 100 in the upper right— all the way into the space so that hill climbing in this potential function leads you on the shortest path to the goal. The algorithm is a recursive algorithm. It spreads the value through the space, as you can see in this animation, and after a number of iterations, it converges, and you have a grayscale value that really corresponds to the best way of getting to the goal. Hill climbing in that function gets you to the goal. You can simplify. Think about this as pouring a glass of milk into the 100th state and having the milk descend through the maze, and later on, when you go in the gradient of the milk flow, you will reach the goal in the optimal possible way.

19 – Value Iteration 3 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

Let me tell you about a truly magical algorithm called value iteration. In value iteration, we recursively calculate the value function so that in the end, we get what's called the optimal value function. And from that, we can derive, look up, the optimal policy. Here's how it goes. Suppose we start with a value function of 0 everywhere except for the 2 absorbing states, whose value is +100 and -100. Then we can ask ourselves the question is, for example, for the field A3, 0 a good value. And the answer is no, it isn't. It is somewhat inconsistent. We can compute a better value. In particular, we can understand that if we're in A3 and we choose to go east, then with 0.8 chance we should expect a value of 100. With 0.1 chance, we'll stay in the same state, in which case the value is -3. And with 0.1 chance, we're going to stay down here for -3. With the appropriate definition of value, we would get the following formula, which is 77. So, 77 is a better estimate of value for the state over here. And now that we've done it, we can ask ourselves the question is this a good value, or this a good value, or this a good value? And we can propagate value backwards in reverse order of action execution from the positive absorbing state through this grid world and fill every single state with a better value estimate than the one we assumed initially. If we do this for the grid over here and run value iteration through convergence, then we get the following value function. We get 93 over here. We're very close to the goal. 89, 85, 81, 77, 73, 70, over here. This state will be worth 68, and this state is worth 47, and the reason why these are not so good is because we might stay quite a while in those before we'll be able to execute an action that gets us outside the state. Let me give you an algorithm that defines value iteration. We wish to estimate recursively the value of state S . And we do this based on a possible successor state as prime that we look up in the existing table. Now, actions A are non-deterministic. Therefore, we have to go through all possible as primes and weigh each outcome with the associated probability. The probability of reaching S prime given that we started state S and apply action A . This expression is usually discounted by gamma, and we also add

the reward or the costs of the state. And because there's multiple actions and it's up to us to choose the right action, we will maximize over all possible actions. See, we look at this equation, and it looks really complicated, but it's actually really simple. We compute a value recursively based on successor values plus the reward and minus the cost that it takes us to get us there. Because Mother Nature picks a successor state for us for any given action, if you compute an expectation over the value of the successor state weighted by the corresponding probabilities which is happening over here, and because we can choose our action, we maximize over all possible actions. Therefore, the max as opposed to the expectation on the left side over here. This is an equation that's called backup. In terminal states, we just assign $R(s)$, and obviously, in the beginning of value iteration, these expressions are different, and we have to update. But as Bellman has shown a while ago, this process of updates converges. After convergence, this assignment over here is replaced by the equality sign, and when this equality holds true, we have what is called a Bellman equality or Bellman equation. And that's all there is to know to compute values. If you assign this specific equation over and over again to each state, eventually you get a value function that looks just like this, where the value really corresponds to what's the optimal future cost reward trade off that you can achieve if you act optimally in any given state over here.

20 – Deterministic Question 1 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

Let me take my example world and apply value iteration in a quiz. As before, assume the value is initialized as +100 and -100 for the absorbing states and 0 everywhere else. And let me make the assumption that our transition probability is deterministic. That is, if I execute the east action of this state over here with probability 1 item over here, if I assume the north action over here, probability 1, I will find myself in the same state as before. There is no uncertainty anymore. That's really important for now, just for this one quiz. I'll also assume gamma equals 1, just to make things a little bit simpler. And the cost over here is -3 unless you reach an absorbing state. What I'd like to know, after a single backup, what's the value of A3?

21 – Deterministic Question 1 Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

And the answer is 97. It's easy to see that the action east is the value maximizing action. Let's plug in east over here. Gamma equals 1. This is a single successor state of 100, so if we have 100 over here minus the 3 that it costs us to get there, we get 97.

22 – Deterministic Question 2 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

Let's run value iteration again, and let me ask what's the value for B3, assuming that we already updated the value for A3 as shown over here.

23 – Deterministic Question 2 Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

And again, making use of the observation that our state transition function is deterministic, we get 94, and the logic is the same as before. The optimal action here is going north, which we will succeed with the probability 1. Therefore, we can use the value recursively from A3 to deflect back to B3. $97 - 3$ gives us 94.

24 – Deterministic Question 3 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

And finally, I would like to know what's the value of C1, the figure down here, after we ran value iteration over and over again all the way to a convergence. Again, gamma equals 1. State transition function is deterministic.

25 – Deterministic Question 3 Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

And the answer is easily obtained if you just subtract -3 for each step. We get 88 and 85 over here. We could also reach the same value going around here. So, 85 would have been the right answer, and this will be the value function after convergence. It's beautiful to see that the value function is effectively the distance to the positive absorbing state times 3 subtracted from 100. So, we have 97, 94, 91, 88, 85 and so on. This is a degenerate case. If we have a deterministic state transition function, it gets more tricky to calculate for the stochastic case.

26 – Stochastic Question 1 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

Let me ask the same question for the stochastic case. We have the same world as before, and actions have stochastic outcomes. The probability 0.8, we get the action we commanded, otherwise we get left or right. And assuming that the initial values are all 0, calculate for me for a single backup the value of A3.

27 – Stochastic Question 1 Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

This should look familiar from the previous material. It's 77, and the reason is in A3, we have an 80% chance for the action going east to reach 100. But the remaining 20%, we either stay in place or go to the field down here, both of which have an initial value of 0. That gives us 0, but we have to subtract the cost of 3, and that gives us $0 - 3 = -3$. It's also easy to verify that any of the other actions have lower values. For example, the value of going west will be 0 in all possible outcomes given the current value function minus 3, so the value of going west would right now be estimated as -3, and 77 is larger than -3. Therefore, we'll pick 77 as the action that maximizes the updated equation over here.

28 – Stochastic Question 2 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

And here's a somewhat non-trivial quiz. For the state B3, calculate the value function assuming that we have a value function as shown over here and all the open states have a value of assumed 0, because we're still in the beginning of our value update. What would be our very first value function for B3 that we compute based on the values shown over here?

29 – Stochastic Question 2 Solution – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

And the answer is 48.6. And obviously, it's not quite as trivial as the calculation before because there's 2 competing actions. We can try to go north, which gives us the 77 but risks the chance of falling into the -100. Or we can go west, as before, which gives us a much smaller chance to reach 77, but avoids the -100. Let's do both and see which one is better. If we go north, we have a 0.8 chance of reaching 77. There's now a 10% chance of paying -100 and a 10% chance of staying in the same location, which at this point is still a value of 0. We subtract our costs of 3, and we get $61.6 - 10 - 3 = 48.6$. Let's check the west action value. We reach the 77 with probability 0.1 with 0.8 chance we stay in the same cell, which has the value of 0, and with 0.1 chance, we end up down here, which also has a value of 0. We subtract our costs of -3, and that gives us $7.7 - 3 = 4.7$. At this point, going west is vastly inferior to going north, and the reason is we already propagated a great value of 77 for this cell over here, whereas this one is still set to 0. So, we will set it to 48.6.

30 – Value Iterations And Policy 1 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

So, now that we have a value backup function that we discussed in depth, the question now becomes what's the optimal policy? And it turns out this value backup function defines the optimal policy as completely opposite of which action to pick, which is just the action that maximizes this expression over here. For any state S , any value function V , we can define a policy, and that's the one that picks the action under argmax that maximizes the expression over here. For the maximization, we can safely draw up γ and $R(s)$. Baked in the value iteration function was already an action choice that picks the best action. We just made it explicit. This is the way of backing up values, and once values have been backed up, this is the way to find the optimal thing to do.

31 – Value Iterations And Policy 2 – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

I'd like to show you some value function after convergence and the corresponding policies. If we assume $\gamma = 1$ and our cost for the non-absorbing state equals -3 , as before, we get the following approximate value function after convergence, and the corresponding policy looks as follows. Up here we go right until we hit the absorbing state. Over here we prefer to go north. Here we go left, and here we go north again. I left the policy open for the absorbing states because there's no action to be chosen here. This is a situation where the risk of falling into the -100 is balanced by the time spent going around. We have an action over here in this visible state here that risks the 10% chance of falling into the -100 . But that's preferable under the cost model of -3 to the action of going south. Now, this all changes if we assume a cost of 0 for all the states over here, in which case, the value function after convergence looks interesting. And with some thought, you realize it's exactly the right one. Each value is exactly 100, and the reason is with a cost of 0, it doesn't matter how long we move around. Eventually we can guarantee in this case we reach the 100, therefore each value after backups will become 100. The corresponding policy is the one we discussed before. And the crucial thing here is that for this state, we go south, if you're willing to wait the time. For this state over here, we go west, willing to wait the time so as to avoid falling into the -100 . And all the other states resolve exactly as you would expect them to resolve as shown over here. If we set the costs to -200 , so each step itself is even more expensive than falling into this ditch over here, we get a value function that's strongly negative everywhere with this being the most negative state. But more interesting is the policy. This is a situation where our agent tries to end the game as fast as possible so as not to endure the penalty of -200 . And even over here where it jumps itself into the -100 's it's still better than going north and taking the excess of 200 as a penalty and then leave the $+100$. Similarly, over here we go straight north, and over here we go as fast as possible to the state over here. Now, this is an extreme case. I don't know why it would make sense to set a penalty for life that is so negative that even negative death is worse than living, but certainly that's the result of running value iteration in this extreme case.

32 – MDP Conclusion – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

So, we've learned quite a bit so far. We've learned about Markov Decision Processes. We have fully observable with a set of states and corresponding actions where they have stochastic action effects characterized by a conditional probability entity of P of S prime given that we apply action A in state S . We seek to maximize a reward function that we define over states. You can equally define over states in action pairs. The objective was to maximize the expected future accumulative and discounted rewards, as shown by this formula over here. The key to solving them was called value iteration where we assigned a value to each state. There's alternative techniques that have assigned values to state action pairs, often called $Q(s, a)$, but we didn't really consider this so far. We defined a recursive update rule to update $V(s)$ that was very logical after we understood that we have an action choice, but nature chooses for us the outcome of the action in a stochastic transition probability over here. And then we observe the value iteration converged and we're able to define a policy if we're assuming the argmax under the value iteration expression, which I don't spell out over here. This is a beautiful framework. It's really different from planning than before because of the stochasticity of the action effects. Rather than making a single sequence of

states and actions, as would be the case in deterministic planning, now we make an entire field a so-called policy that assigns an action to every possible state. And we compute this using a technique called value iteration that spreads value in reverse order through the field of states.

33 – Partial Observability Introduction – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

So far, we talked about the fully observable case, and I'd like to get back to the more general case of partial observability. Now, to warn you, I don't think it's worthwhile in this class to go into full depth about the type of techniques that are being used for planning and uncertainty if the world is partially observable. But I'd like to give you a good flavor about what it really means to plan in information spaces that we reflect the types of methods that are being brought to bear in planning and uncertainty. Like my Stanford class, I don't go into details here either because the details are much more subject to more specialized classes, but I hope you can enjoy the type of flavor of materials that you're going to get to see in the next couple of minutes.

34 – POMDP Vs MDP – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

So we now learned about fully observable environments, and planning in stochastic environments with MDPs I'd like to say a few words about partially observable environments, or POMDPs—which I won't go into in depth; the material is relatively complex. But I'd like to give you a feeling for why this is important, and what type of problems you can solve with this, that you could never possibly solve with MDPs. So, for example, POMDPs address problems of optimal exploration versus exploitation, where some of the actions might be information-gathering actions; whereas others might be goal-driven actions. That's not really possible in the MDPs because the state space is fully observable and therefore, there is no notion of information gathering.

35 – POMDP – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

.....

I'd like to illustrate the problem, using a very simple environment that looks, as follows: Suppose you live in world like this; and your agent starts over here, and there are 2 possible outcomes. You can exit the maze over here—where you get a plus 100— or you can exit the maze over here, where you receive a minus 100. Now, in a fully observable case, and in a deterministic case, the optimal plan might look something like this; and whether or not is goes straight over here or not, depends on the details. For example, whether the agent has momentum or not. But you'll find a single sequence of actions and states that might cut the corners, as close as possible, to reach the plus 100 as fast as possible. That's conventional planning. Let's contrast this with the case we just learned about, which is the fully observable case or the stochastic case. We just learned that the best thing to compute is a policy that assigns to every possible state, an optimal action; and simplified speaking, this might look as follows: Where each of these arrows corresponds to a sample control policy. And those are defined in part of the state space that are even far away. So this would be an example of a control policy where all the arrows gradually point you over here. We just learned about this, using MDPs and value iteration. The case I really want to get at is the case of partial observability— which we will eventually solve, using a technique called POMDP. And in this case, I'm going to keep the location of the agent in the maze observable. The part I'm going to make unobservable is where, exactly, I receive plus 100 and where I receive minus 100. Instead, I'm going to put a sign over here that tells the agent where to expect plus 100, and where to expect minus 100. So the optimum policy would be to first move to the sign, read the sign; and then return and go to the corresponding exit, for which the agent now knows where to receive plus 100. So, for example, if this exit over here gives us plus 100, the sign will say Left. If this exit over here gives us plus 100, the sign will say Right. What makes this environment interesting is that if the agent knew which exit would have plus 100, it will go north, from a starting position. It goes south exclusively to gather information. So the question becomes: Can we devise a method for planning that understands that, even though we'd wish to

receive the plus 100 as the best exit, there's a detour necessary to gather information. So here's a solution that doesn't work: Obviously, the agent might be in 2 different worlds—and it doesn't know. It might be in the world where there's plus 100 on the Left side or it might be in the world with plus 100 on the Right side, with minus 100 in the corresponding other exit. What doesn't work is you can't solve the problem for both of these cases and then put these solutions together— for example, by averaging. The reason why this doesn't work is this agent, after averaging, would go north. It would never have the idea that it is worthwhile to go south, read the sign, and then return to the optimal exit. When it arrives, finally, at the intersection over here, it doesn't really know what to do. So here is the situation that does work— and it's related to information space or belief space. In the information space or belief space representation you do planning, not in the set of physical world states, but in what you might know about those states. And if you're really honest, you find out that there's a multitude of belief states. Here's the initial one, where you just don't know where to receive 100. Now, if you move around and either reach one of these exits or the sign, you will suddenly know where to receive 100. And that makes your belief state change— and that makes your belief state change. So, for example, if you find out that 100 is Left, then your belief state will look like this— where the ambiguity is now resolved. Now, how would you jump from this state space or this state space? The answer is: when you read the sign, there's a 50 percent chance that the location over here will result in a transition to the location over here— 50 percent because there's a 50 percent chance that the plus 100 is on the Left. There's also a 50 percent chance that the plus 100 is on the Right, so the transition over here is stochastic; and with 50 percent chance, it will result in a transition over here. If we now do the MDP trick in this new belief space, and you pour water in here, it kind of flows through here and creates all these gradients—as we had before. We do the same over here and all these gradients are being created point to this exit on the Left side. Then, eventually, this water will flow through here and create gradients like this; and then flow back through here, where it creates gradients like this. So the value function is plus 100 over here, plus 100 over here that gradually decrease down here, down here; and then gradually further decrease over here— and even further decrease over there, so we've got arrows like these. And that shows you that in this new belief space, you can find a solution. In fact, you can use value iteration—MDP's value iteration— in this new space to find a solution to this really complicated partially observable planning process. And the solution—just to reiterate— we'll suggest: Go south first, read the sign, expose yourself to the random position to the Left or Right world in which you are now able to reach the plus 100 with absolute confidence.

36 – Planning Under Uncertainty Conclusion – lang_en_vs1.srt (10. Planning under Uncertainty Subtitles)

So now we have, learned pretty much, all there is to know about Planning Under Uncertainty. We talked about Markov Decision Processes. We explained the concept of information spaces; and what's better, you can actually apply it. You can apply it to a huge number of problems where the outcome of states are uncertain. There is a huge legislation about robot motion planning. Here are some examples of robots moving through our environments that use MDP-style planning techniques; and these methods have become vastly popular in artificial intelligence— so I'm really glad you now understand the basics of those and you can apply them yourself.

ADVERTISEMENT



REPORT THIS AD

Advertisements



REPORT THIS AD



Published by Aarsh Talati

Artificial Intelligence Developer, Data Scientist [View all posts by Aarsh Talati](#)



