Week 11

Video Transcripts

Video 1 (16:03): Hidden Markov Models

In this lecture, we're going to discuss a variation of this technique called hidden Markov models. To motivate this problem, we've seen how in the previous lecture–how Markov models are models for sequential data where we assumed that we could observe the sequence of states. Either we could observe them, or we could build a model in which we are able to observe them. So, in the two cases that we looked at ranking, and semi-supervised classification, we actually are never given a sequence of states. But the models that we construct assume that we can generate that sequence of states and it's going to give us something we care about. So, we work at the state level where the sequences are only of states and nothing else. However, in many cases, the model might not be flexible or expressive enough to work on a discreet state level.

So, again, remember that the states as we constructed them previously were discrete. They were finite and discrete...they were a finite and discrete number of states. But, the data we get might not really work in that setting. So, we might want to instead let each state in our model correspond to a probability distribution on the observable part of the sequence. So, this leads to the idea of a hidden Markov model, hidden being because we don't get to observe a latent state's sequence. So, as just mentioned, the hidden Markov model is different from a Markov model or a Markov chain in that it treats the sequence of states differently. So, here we're going to assume that our sequence of states is a latent variable in our model and that the observed variable that we get to see, which is part of our data, depends on the state but it's not one to one mapping to the state. So, it's a variation of the state. So, here's a graphical model representation of the two before we get into more detailed explanation.

So, with the Markov model– the way we constructed the generative process was–we assumed that we were transitioning from state to state. So, here's a sequence of four different states– so our data sequence would be four-dimensional and each of these nodes would be a number between one to 's', if we have 's' different states. And the value of the state at time two depends on the value of the state at time one, based on the Markov transition matrix. Now, we're going to end up with something that looks like this, where the state sequence that we have here is exactly the same as what we described in the last lecture, only it's not colored in because it's not observed. So, here we color it in because we assume that we're observing these state sequences. In the hidden Markov model, we have the same exact generative process for generating these state sequences but we don't get to observe it. Instead what we get to observe is a conditional–rand–probability distribution or a random variable generated from a conditional probability distribution picked out by the state.

So, given that I'm in a certain state at time two, I then generate an observed random variable 'x_2', that is conditionally dependent on whatever state I'm in here at this time. So, let's look at an example where we transition from a Markov chain to a hidden Markov model. So, let's imagine in this toy example in the next couple of slides that we have three possible states in R^2. So, here's the

second state, the third state, and the first state. We're giving it a physical representation in R^2. So, we're going to say that the third state has this point in R^2.

So, it's a two-dimensional vector that represents the third state, a two-dimensional vector that represents the first state, and a two-dimensional vector that represents the second state. And so, now since we have only three unique positions that we could transition between, we simply index them. We don't give the two-dimensional vector corresponding to the first state, we give the index of the first state, one. And so, when we generate a Markov–tran–sequence, so we have, for example, the Markov transition matrix 'A'. So, in this lecture we're going to switch to the standard notation 'A' for a hidden Markov model. So, if we assume that we have a three by three Markov transition matrix 'A', then the ij^{th} value is the probability of transitioning from state 'i' to state 'j'. So for example, 'A_{21}' is the probability of going from the second state to the first state, and 'A_{12}' is the probability of going from the first state to the second state. So here's how we can physically represent the transition. And 'A_{11}' is the probability of transitioning from state one to itself, etc. So, last time we didn't need to think about these embeddings of these states in some space, we simply generated a sequence using the state index. So, we would get a sequence one two one three two and that would indicate that our transition was from this point to this point to this point, to this point to this point. Something like that. Okay, but now in the case of a hidden Markov model, it's useful to think of each state as being embedded in some location, some latent space. Because now, let's imagine that we have the same exact three states, so, the center of these three ellipses corresponds exactly to the location of these three states. The color's exactly the same.

But now, each time we arrive at one of the three states, we don't get to observe the exact location of the state. It's not a deterministic, you know, discrete thing we get to see. We get to see the location of the state with some random permutation. So, instead of seeing these three vectors over and over again and only these three two-dimensional vectors, we get to see that vector plus some noise. So, again we have the same sequence of states according to a Markov transition matrix 'A' where we have the same transitions, for example, one two one three two which would be one two one three two. Same positions in R^2. However, if we now let 'Mu_1' be the position of state one– so 'Mu_1' corresponds to this point here– this is 'Mu_1', this point here–then what we get to observe is not 'Mu_1' which would be the case of a Markov chain. What we get to observe at time 'i' if we're in state one at time 'i' is 'Mu_1' plus some noise epsilon 'i'. So, we get to observe 'x_i' which is 'Mu_1' plus some noise. And the noise, for example, could have this contour. And for example, we could assume that noise is Gaussian, in which case we get to observe a Gaussian random variable with this distribution.

And so, now we don't get to observe the sequence of means 'Mu_1', 'Mu_2', or 'Mu_3', we get to observe a sequence 'x_1' to 'x_t' where each of these vectors 'x'...'x_i' is a randomly permuted mean vector. So, here's again the sequence we're thinking about. So, this describes what's called a continuous hidden Markov model. We assumed we have some Markov chain like this, for example, three states, except we don't get to observe the three states exactly. We get to observe some random permutation of those states. And so, a sequence of data might look like this where this is 'x_1', this is 'x_2', and we have all of these transitions. For example, we have this point at 'x_i', and then at 'x_i' plus one we move from this point to this point. So, we now have a data set in this case of two-dimensional vectors and we have an order–we have that set ordered. So, previous, for example, with the Gaussian mixture model which also kind of looked like this, we didn't care about the ordering of the data. We never said there was something significant about the fact that 'x_i' was what it was and that 'x_i' plus one was what it was.

Now, the ordering of the data's going to matter, because we're going to assume that there's some first order Marko–hidden Markov model underlying that generation. So, again on the left, we have a Markov transition matrix. In the middle, we've taken this Markov chain, and made, and we've now defined a state dependent probability distribution on the observed data. And on the right, we now have a sequence of observations where, for example, in all of these points we're in this second state. And all of the blue points we're in the third state. And all of the red points are in the red state meaning that every blue point is a random variable with this distribution.

The red and the green points are also random variables with their corresponding distributions. And the order in which we generated these points was like this. So, we started in this state, moved to this state, and stayed here for a while. Finally, we moved to the red state and stayed there for a while and then we made it back to the green state. All right, now let's look at a formal definition of a hidden Markov model. A hidden Markov model consists of three things. If we assume it's an 'S' state hidden Markov model, so we have 'S' different states. We have an 'S' by 'S' transition matrix 'A', which gives the probability of transitioning from any state to any other state.

We assume an initial state distribution. So this Pi is an 'S' dimensional probability vector that gives a distribution on the first state. Since we have to start somewhere we can't make–we can't start by making a transition, we have to have a distribution on where we start. So that's encoded by Pi. And then, finally we have a state dependent emission distribution. So, that's the probability distribution on the observed data at time 'i' given that I'm in state 'k' at time 'i'. So, that's this distribution. So, some density 'p' of 'x_i' given a parameter indexed by the state at time 's'– that I'm in at time 'i'. So these are the three different parts of the HMM and now if I want to generate data, I have to define a generative process because this is a probability model.

So I want to generate a sequence, 'x_1', 'x_2', 'x_3', and so on. And I do so as follows: first I sample the first state from a discrete distribution using Pi as the distribution on the first state. So, I get a first state, it's an index between one and 's'. That state then picks out a parameter associated with a distribution for that state. And I generate my observation 'x_1' from that distribution using the parameter indexed by the state that I'm in at time one. And then I follow that by generating the rest of the sequence, through a Markov chain. So, I simply generate a sequence of states where at time 'i' the state–the distribution of my state at time 'i' given that I'm in state 'k' prime at time 'i' minus one is simply a discreet distribution, where I pick out a probability distribution on states from the K^{th} row, 'k' primeth row of my transition matrix. So, this matrix, again, has 's' different transition distributions to the 's' states depending on which state I'm at. And I pick out the state that I'm at at time 'i' minus one by picking out my transition distribution to all of the states given where I'm at to generate my state at time 'i'. Following that, when I generate the state for the i^{th} observation, I actually generate the data, I generate the i^{th} observation itself from a probability distribution where I pick out the parameter associated with the state that I'm in at time 'i', and so we've left this general. There are two different distributions, of course we could have on the data.

We could have continuous distributions or discreet distributions depending on whether the data, the sequence of data that we have, 'x_1' to 'x_t', depending on whether we want to model each of these observed data points as a continuous random variable or discreet random variable. When we want to model these as continuous, then the distribution on the data is a continuous distribution, often a Gaussian, and in that case we say we're doing a continuous HMM. When we model the data as discrete, then the parameter for the 's', the state 's', is simply a vector of probabilities on the

discreet set of observed values that 'x' can take. In that case, we say we're doing a discreet HMM. So HMM's are always discreet in the states.

There are always a set of 's' different state that we can visit. So, each HMM is discrete in that sense, but when we model the observed data as either a continuous or discrete random variable, that then differentiates...differentiates whether we're doing a continuous HMM or discrete HMM. In the rest of this lecture I want to focus on the simpler case which is the discrete HMM. And so, now I'm going to define this probability distribution more precisely and say that we have a matrix 'B' where 'B' has 's' rows and it has 'v' columns if 'x_1' can be a number between one to 'v'. So, it can take one of 'v' different discrete values, then the s^{th} row, the row 's' of my matrix 'B' says the probability of my observation for a data point that comes from that state. So I'm indexing now my distribution on the data for each state along the rows of 'B_s'.

Video 2 (13:09): Examples

Let's look at an example of generating data from a discrete HMM to make it perhaps a bit more concrete. So, here's an example of a dishonest casino. And again, this is an example of a discrete HMM. Imagine that we have two dice, one of them is fair and one of them is unfair. What that means is that the fair dice has six sides and the probability of any individual side is one over six. And the unfair dice has also six sides, but we have different probabilities of which side it will show up when we roll it. Then at each roll, we have a sequence of rolls, we're going to keep the current dice that we're rolling, or we're going to switch to the other dice. But we're not going to get to see which dice is used, we just get to see the roll. But underneath the table perhaps is the choices being made of which dice to pick next, given the dice that was used previously. And all we get to observe is the sequence of the rolls, the numbers that came up.

So, in this case, we have the three parameters for the HMM we need to define. We have a transition matrix, and so, let's think of the transition matrix like this. So, the first state, we're going to call the first state the fair dice, the second state, the unfair dice. And so, the probability of using the fair dice or the fair dice next...in the next roll given that I use it in the current roll is equal to '0.95'. And the probability of switching to the unfair dice given that I'm–that I use the fair dice in the most recent roll is '0.05'. So, that's the probability of switching from a fair to an unfair dice. The probability of switching from a fair–unfair to the fair dice is '0.1'. And the probability of staying with the fair–unfair dice is '0.9'. So, this is how we choose the sequence of dice that we're going to use in the rolls.

Given that we have the fair dice, we use the fair dice, the probability is one over six of the six different observations. So, a dice has six sides. I can see... I see a number, I observe a number between one and six, and so my matrix 'B' has six columns. And my distribution on those six different numbers I get to observe, given that I use the first–the fair dice, which is the first state, is one over six. And then, if i'm in–if I use the unfair dice with probability one half, I have a six roll up. And then, with the remaining probabilities, I have anything else. And then I have to also pick the first dice that's used, and I'll just say that it's an even probability on either. So, this is an example that we'll return to in a later slide of how you would generate data from an HMM. All we get to observe

are the results of the rolls, that's the observed sequence. But the hidden process, the hidden state transition, is picking between one of the two dice.

We've defined the generative process for the HMM. Now we need to discuss what are some of the quantities we might want to learn for the HMM. So, again we generate the generative process. Now we have to–we get some data, and we have to do the inverse process of learning some of the underlying parameters of this model. So with the HMM, there are three–it's usually broken down into three values of interest that we would want to learn. The first is the state estimation. So, in this case we're assuming that we're actually given the HMM. We're given the initial state Pi, the state transition matrix 'A', and the emission probability matrix 'B'. So we're going to assume discrete HMM for the rest of the lecture. And we get the observed sequence 'x_1' through 'x_t'. So, we assume that we always have the data. And we assume we're given an HMM. Now, quantity of interest that we might want to estimate, is what is the state that we were in at time 'i'. So, we have this sequence and this HMM, and now we ask, at time 'i' in this sequence, which of the 's' states was 'i' in? So, we don't know this because that's assumed latent and we assume that we are not given that bit of information. And so, we need to infer it. And so, we don't actually find out which state I was in at time 'i'. I get a distribution on the states that I could have been in that time 'i'. And to do this we use something called the forward-backward algorithm.

And we're not going to be able to discuss this thing, this algorithm, in much detail in this course, but it's a fundamental algorithm for the HMM, and my goal is to motivate where it's used, so that it makes it more clear what the purpose of learning this is. So, we use the forward-backward algorithm. And out of this algorithm–which takes in the HMM parameters and a sequence–what we can get out of it is the posterior probability, the conditional posterior probability, of being in state 'k' at time 'i' given the entire sequence and given the HMM. So, one of the outputs of the forward-backward algorithm is this conditional posterior probability for any time points, for any 'i' and any 'k'. We also might want to estimate the most probable state sequence.

So, in this case we're given an HMM, and we're given an observed sequence. And now we want to find, what is the most probable sequence of states? So, here we aren't giving a distribution on the state sequence, we're actually given...giving a certain sequence of state transitions 's_1' through 's_t', corresponding to the data 'x_1' through 'x_t', where the entire sequence is the most probable one. So we want to find the arg max of the full pos...full conditional posterior on the entire sequence, given the entire sequence of data, and given the HMM parameters.

So, it's a point estimate is this case. It's the MAP solution. And so we actually get a sequence of numbers. This is something that's done through what's called the Viterbi algorithm. This algorithm is a little less fundamental than forward-backward in my opinion, but also very important and something that we also won't describe in this course. But the Viterbi algorithm is something that is run to give the most probable sequence of state transitions, given a sequence of data, and given a Markov model, hidden Markov model. Finally, we might want to learn the HMM parameters, themselves. So, in this case we're just given a sequence of data. And we are...we are also making the assumption of number of states 's'. So, we're given a sequence and we make a number of states equals 's'. So we make this assumption. And now our goal is to find the parameters of the HMM, and especially we might want to find the maximum likelihood.

So, that's the first thing we would want to see if we could calculate. The maximum likelihood parameters of an HMM that most–are most probable given some data. So, this one we'll discuss a

little bit more detail in this course. Let's look at an example of the first two goals that we might want to pursue with an HMM. Learning the posterior distribution on which state we're in at a given time, and also learning the most probable state sequence using the Viterbi algorithm. So, this is for the dishonest casino. What is shown here is shaded–a shaded part and an unshaded part as a function of '300' rolls.

So we have a sequence of length '300'. During the shaded part, the loaded dice, the unfair dice, is used. So the unfair dice is used for the first '20' or '15' rolls or so. Then we switch in the white part to the fair dice. The grey part is the unfair dice. Then, we switch back to the fair dice, and so on. So, this is the underlying–the true underlying state of the model. And now, what we observe is not this underlying state. So, we don't get to observe the shaded or not shaded, which would be the binary indicator of the state. We get to observe the sequence of rolls. So, that's not shown here, the sequence of numbers, that came up during the roll. However, given that particular sequence of length '300', the forward-backward algorithm can be run. And what we get now is a sequence of prob–of conditional posterior probabilities of a given state. So, for example, at this time point here, the probability of being in the third-using the load...loaded dice, we're saying is very low.

So, in this blue line, when it's very low, that's saying the probability of using the unfair dice at that time point is very low. And when this number is very high–for example, in this region, we're saying that at these time points we think the probability that the unfair dice was used, is very high. And so, these are the marginal conditional posterior probabilities. And so, you see that we get very low, for example, in the fair dice period, we say, okay, I believe things are looking fair now. Then we switch the unfair dice and suddenly we start moving up and saying, I think the unfair dice is being used here. Then we move down.

And so, you can see in general, it's able to track which dice is being used, not perfectly, and we're never certain, we only have probabilities in this case. The second problem is trying to find the most probable state sequence. And so, this uses the Viterbi algorithm. The most important thing to know right now is that this is not simply a process of rounding this plot. We don't simply take these probabilities and round them to the most probable state, and then return that as the most probable state sequence. Because, remember, we have a sequential dependence here.

So, the most probable state sequence takes into account on more global view of the situation. It's not simply a local view. So what this, remember, is plotting is if I only look at time '0.50', what is my belief about which state I'm in? Forget all the other time points. Just tell me right at this time point, what's my conditional posterior probability of where I'm at right now? And so that's how–that's what this is plotting as a function of time. Here, we're plotting the most probable sequence overall. And so, it's more probable to stay in zero–say we're saying the fair dice at this time point and then switch the loaded here. And so, we get something very smooth. If we rounded this left plot, we would get things that jump very chaotically at different points.

And those jumps are very low probability events given our transition matrix, which says that we have–we stay at the same state with a high probability. So, the low probability of those jumps that we would get rounding here are removed using the Viterbi algorithm, which is taking into account not just trying to explain the data, but trying to explain a likely state sequence–transition of state sequence between states. And so, you see here, we are–we give up some accuracy. For some of these we might have said, it's more probable, for example, here to say we're using the loaded dice. Whereas here, because we're taking a global view into account, we say, no, this entire period is

using the fair dice. Now, this doesn't necessarily look good for this problem. And it's an example perhaps of where MAP–doing a MAP point is not ideal. We might want to use probabilities to express things.

Video 3 (16:53): Learning the HMM

So, we're going to, again, focus on the discrete HMM. And for this lecture, we are concerned with the maximum likelihood solution. So, we want to find the parameters of the HMM Pi, 'A', and 'B' that maximizes this likelihood of the observed data sequence given those parameters, and so notice that what's not visible in this likelihood is the sequence of states. So, we're integrating out, or summing out the entire state sequence, which we can write this way.

If we write the joint probability distribution of the observed sequence and the hidden sequence, given an HMM, and then we sum over the entire sequence of states, we can get this term here. And so, now we have this representation. And using the Markov property, we can factorize this as follows, where we have the conditional distribution of the observed data, given the hidden state, and given the emission probability matrix, times the probability of this transition from state 'i' minus on...at time 'i' minus one to state at time 'i', given a Pi and 'A'. And so, Pi is here just for the first, the first state, in which case 's_o' is missing, it doesn't belong. And then 'A' is here for the remaining transitions after the first state. And so, we already know that this probability, for our discrete Markov model, hidden Markov models can be written this way, we have pick out the distribution on the data for the i^{th}–the state at time 'i', so 's_i' indexes the discrete distribution vector we're going to use. And then 'x_i' picks out the dimension of that vector. So the...the element in B that corresponds to the state and the observation is the probability of observing– that observation given that we're in that state. Similarly, the probability of this transition is equal to the inde–is equal to the value and the 's_{i-1}' row, and the 's_i' column, because here we're saying that we transition from state at 's_{i-1}' to the state of 's_i'.

So, this probability is encoded in this transition matrix, or if it's the first state in the sequence, it's encoded in this initial state vector. So, we can plug in all three of these things in...in this distribution. But, we don't know what the states are. We want to sum them out and maximize this marginal likelihood. However, it's very hard to do. Because if we wanted to maximize this log marginal likelihood, we're maximizing the log of a sum, and we remember that that's a difficult thing to do. If we started to take derivatives, we would find very quickly that we're in trouble, that we can't optimize these things in close form. However, we also know, or can observe that if we get to know the state sequence 's_1' through 's_t', that this would become a much easier problem. So if we treated these latent ste– sequences as missing data and did a joint likelihood, we would certainly have a very easy model to learn.

In addition, we can observe, I'm not going to derive it, but, what we can do is we can show how we can calculate the conditional posterior of the entire state sequence or the important parts of it given the data and given an HMM. And this is done through the forward-backward algorithm. So therefore, we conclude that we can use the EM algorithm for this problem. And that's how it's usually done is with the EM algorithm. And so in the following slides, I'm going to give a high level

overview of the EM algorithm for the HMM. As we recall, the EM algorithm has two steps. First, we have to define the conditional posterior distribution of the hidden data that we're introducing that we want to integrate out, given the observed data, and given the model parameters, we want to do the point of submit of.

So, we have to calculate this 'q' distribution on the entire state sequence, given the observed sequence, and the HMM parameters. Then given that distribution, we have to calculate the expectation of the log of the joint likelihood of the observed, and unobserved sequence, given the HMM parameters using this 'q' distribution. And so, that's the E-step, where we give this objective function, where we have integrated out the state sequence or summed out, in this case, and now it's a function of the HMM parameters, which leads to the M-step, where we maximize this thing with respect to Pi 'A' and 'B'. So, this part now is tricky, because if we look at the log of the joint likelihood of the observed and unobserved state sequen–data sequence, given the HMM parameters, we'll–we observe that it's equal to something like this.

So, here I'm summing over all of the observations, of the log of the probability of...of observing data at time point 'i', given that I'm in state 'k' at time point 'i', and so this sum here is simply picking out the correct probability. If I want to say that I'm in the state 'k', then this indicator will be an equal to one, for when 's_i' equals 'k', which we'll pick out the correct probability. So, this is the data level log likelihood corresponding to this term here.

So, we're taking the log. In this case, we're taking the log, so we're now assuming up the joint likelihood so, we can get rid of this portion. And so, now we have the log of this thing. And so, this first term is the log of the data level likelihood. And then these two terms are the logs of the state sequence. So, this term simply corresponds to the first state in the sequence. So 's_1'. I have the sum over all of the states, an indicator of which state I'm going to say I'm in, times the log of the probability of that state to start. And then here, for the remaining portion of the sequence, so, for state 'i', at time 'i' equals two to the end, I now have to have an indicator of being in a particular state 'j' at time 'i' minus one, and then transitioning to a state 'k' at time 'i', times the log of the probability of that transition.

And now this thing has to be summed over all of the state transitions in order to pick out the right one. So, this is how I can represent the log of the joint likelihood here. And so, the following results are taken from this classic tutorial by 'Rabiner' called 'A Tutorial on Hidden Markov Models and Selected Applications in Speech Recog–Recognition'. So, that...that tutorial will have much more discussion on the forward-backward algorithm, and the Viterbi algorithm as well; as well as the results that I'm going to use from that tutorial in the following slides. So, notice with the E-step that we take the expectation of this thing with respect to this 'q' distribution. So, this is the full posterior distribution of the entire state sequence given da–the observed data sequence and the HMM. And now I'm going to use this, I haven't defined what it is yet, to calculate the expectation here. And so, now we are going to make some crucial observations.

The first observation is the fundamental fact that the expectation of an indicator of an event is equal to the probability of that event according to the distribution. So, if I have some distribution 'q', and I have some indicator of an event 'x' as in a set 'A' for example, that's simply equal to 'q', the probability that 'x' is in 'A', according to the distribution 'q'. So I have a bunch of indicators here that I'm taking the expectation of with respect to this full posterior distribution. And so, notice that when I take the expectation of the event of being in state 'k' at time 'i' according to this, I'm actually

summing over all the states at different time points except for at time point 'i'. So, this expectation of this event according to this distribution is the marginal conditional posterior of being in this state at time 'i', given that, I'm sorry, of being in state 'k' at time 'i'. So, the expectation, I'll just do it for this one example, the expectation according to 'q' of the indicator of being in state 'k' at time 'i' is equal to the conditional posterior of being in state 'k' at time 'i' given X and given 'A', 'B' and Pi, the distribution of the HMM.

So, remember, this is a full conditional posterior. The expectation of this event of being in state 'k' at time 'i' is simply the marginal focus, of the full conditional posterior, where I've summed up over all states except for the ith at time 'i'. Okay, so that's where these new definitions are going to come in. So, in the E-step, let's define two things. Let's define 'Gamma$_i$''k' to be the conditional posterior probability of being in state 'k' at time step 'i'. So, that corresponds to this. This...this expectation of this thing is equal to 'Gamma$_i$' 'k'. Also we have expectation here, the expectation of being in state 'j' at time 'i' minus one and state 'k' at time 'i'. This thing we're going to call 'Si$_i$' 'j' 'k'.

So this matrix, it's an 'S' by 'S' matrix, if I have 'S' total states, is equal to the conditional posterior probability that I'm in state 'j' at time 'i' minus one. So here, the 'i', the first state is 'i' minus one. I'm in state 'j'. And then I transition to state 'k' at time 'i'. So, at time 'i' minus one, I transition from state 'j' to state 'k'. This matrix encodes the posterior probability. So, this is a matrix 'S' by 'S' where the sum of the entire matrix equals one, and all the values are non-negative. It's a posterior probability matrix. Both of these things are outputs of what is called the forward-backward algorithm. So, again, we won't cover it, but the tutorial is good for covering that. So, we calculate these two probabilities using the forward-backward algorithm, which we can find the details of that in this tutorial. Then, we take the expectations, and we get this.

So, notice that what I've simply done here is I have taken this very complicated writing of these expectations and replaced them with their values that we defined. So, here we defined the expectation of this to be 'Gamma$_i$' 'k'. That's what this is here. And then one more. We define this expectation of this thing to be 'Si$_i$' 'j' 'k'. That's what's going on here. So here I've taken the expectation of the log, of the joint likelihood using 'k', and I get something that looks like this.

Now we need to just maximize this thing with respect to Pi, 'A', and 'B'. And so, we take derivatives. We use the constraints that they have to be probability distributions that sum to one. And we obtain this final result. And so, these are the EM. I'm sorry, this is the M-step after calculating Gamma and 'Xi' in the E-step. We have this M-step update. So let's look at these one by one. First, let's look at the update for the probability of transitioning from state 'j' to state 'k'. So given I'm in state 'j', what's the probability I transition to state 'k'? This is simply the expected number, a fraction of transitions from state 'j' to state 'k', when we start in state 'j'. So the numerator here is summing over each transition 'i', the conditional posterior probability of making that transition at that time point.

So, here we're summing over the expected number of transitions from state 'j' to state 'k'. That's what this numerator is doing. And then in the denominator, we're summing over all of the expected transitions from state 'j' to any other state. So, here we pick a state 'l', we then sum, the expected number of transitions from state 'j' to state 'l', and then we sum over all possible values for 'l'. And so, this is simply taking this normalized, this numerator, and then normalizing along the row. So, here's the expected probability, the probability of transitioning from–to state 'k' given I'm in state 'j', calculated from these probabilities from the forward-backward algorithm.

'B' is essentially the same. The numerator is the expected number of times I observed 'v', the output 'v' from state 'k'. So, here I have an indicator that the i^{th} observed value is equal to 'v'. We have a discrete HMM here. So, an indicator of observing 'v' at time 'i'. And then the probability of being in state 'k' at time 'i'. So, I simply multiply these two things together, and what I'm doing here is summing the expected total number of times that I see the output 'v' from state 'k'. So, I'm summing over all the time points. And then I divide by the total number of observations that I'm going to see from state 'k'. That simply takes the numerator and normalizes it so the rows sum to one. And similarly, for the initial state, we can use the probability distribution of being in any particular state at time one, and then normalize that.

In this case, we don't need to, since this is already distribution, this sums to one, to get the initial state probability. Okay, so this is the maximum likelihood solution for an HMM, when we have one sequence. So, how do we modify this if we have many different sequences? So, it's a simple modification. Simply, all we do is we calculate these probabilities for each individual sequence. We literally perform the same steps for each individual sequence, separately. And then we sum over all of those sequences.

So, here we have a conditional posterior distribution of transitioning from state 'j' to state 'k' at time 'i' in the nth sequence. And we have a conditional posterior of being in state 'k' at time 'i' for the n^{th} sequence. We sum these things up over each individual sequence, and then we sum over all of the sequence, all of the sequences, and then normalize, and we get something like this. So here we–in red, I'm showing that we now simply sum these values over each sequence.

Video 4 (13:25): Application: Speech Recognition

Let's look at an example of–an application of hidden Markov model through speech recognition. So, we're going to discuss the problem at a very simple level. The problem itself is very straightforward and well understood by everybody. If your iPhone and you talk into it, it takes your audio signal and it extracts the words that you're saying and puts it into text to then do some task. So, speech recognition is a very straightforward problem. And the HMM, is a fundamental approach to it.

As you can tell from the title of the tutorial that I recommended to you by Rabiner under classical...the classic tutorial, speech recognition is actually in the title there. So what the HMM does is it models transitions between basic units of sound in your voice called phonemes. So, phonemes are thought of as the smallest unit of sound in a language that can be made. So, English, it's assumed–it's thought has about '50' phonemes, and if you want a few examples of phonemes, you can have–you can look down here. So, here's the words 'Zero' through 'Nine'. And they're broken down according to phonemes.

So, 'Zero' has about four phonemes. 'One' has three phonemes. 'Two' has two phonemes. And each of these phonemes, for example, the word 'Two' has 'T' and 'UW'. In each of these phonemes, it's thought that the same sound is being produced. So, for however long I say 'T', and however long I saw 'UW', my voice is making identically the same sound or trying to make identically the same sound there, some natural variation, but there's no change in the sound that's being made. And so,

'Zero' has four different points in that word where the same exact sound is being made at each of those four points.

So, what an HMM tries to do when modeling this is, is assign each state–or learn, it can't assign it, but it's hoping to learn for each state one of the phonemes. And so, each state is a distribution on a sound 'F' or 'AY', for example. So, if I say 'F' or somebody else says it, we're not going to generate identically the same sound, but you could imagine that the sounds that we do generate will be very close to each other. You could say that they came from the same probability distribution on whatever the features are that are going to be extracted from that specific point in time.

So, let's look at how we actually process speech in one slide before we discuss applying HMM to it. So, we get this sort of a speech signal. So, we have time. It's here normalized to be between zero and one. And the speech signal is an amplitude, and there are many, many measured values between zero and one. We then extract features from this using the Fourier Transform along the sliding window. So, at different points in time, we expand the window at, for example, at this point here, we expand a very small window, for example, perhaps '50' milliseconds between–around this point. And we then look within that '50' millisecond window at the audio signal at that time point and extract a feature vector for it.

The first feature vector that would be extracted and possibly post-processed would be the frequency content. So, imagine that we look at about ten thousand frequencies. We get a breakdown at any given time point, of what frequencies are being used. So, at this time point, it's very quiet corresponding to this point here. So, there's almost no sound being made. Then, suddenly, there's sound, and we get a frequency breakdown that looks like this. So, at this time slice we have a lot of this frequency here, some here, here, here, here, and then less other places.

So, this is the distribution on the frequencies over a '50'–sliding '50' millisecond window. And so, you can see, it's very smooth. I believe what's said here is Bayes theorem. So, this is probably BAYE and

then 'S' and then THEOREM. And so, what's happening is you can see that there's very smooth transitions between BAYE and then 'S'. So, this would be one phoneme. This would be, I suppose, two phonemes. So, at some point in here it would have changed. And then THEOREM, for example, this could be 'TH' and then suddenly it changes to 'EO' and then 'REM'. So, we have these different– we have this sort of a frequency content. And then using things like MFCCs, Mel Frequency Cepstral

Coefficients, or other feature extracting techniques, we can then reduce this to being a sequence of, for example, '40' dimensional vectors. So, we have a feature extraction process, where we have a sequence of vectors, and each vector is supposed–is intended to capture the frequency content at that moment in time. And now we want to use a Markov– hidden Markov model to model that frequency context–content sequentially. So, we could either use a continuous hidden Markov model, in which case if we extrac–if we reduce this down to '40' dimensions using MFCCs, we would then perhaps say that each state has a '40' dimensional Multivariate Gaussian associated with it. And, for example, all data generated in this time period would be potentially in the same state with perhaps a small change here. And therefore, have the same distribution on this–these different vectors and so on. If we wanted to discretize it, we could use K-means. And so this is often where K-means is extremely useful, and I would say one of the most practical uses for K-means is in discretizing data for use in other modeling problems. So, for example, if we had a training set like

this of many different instances of a person speaking a word, and we had–we extracted a sequence of feature vectors in some space 'D' dimensional space.

We could then take this data and quantize it using K-means to get a codebook of size 'V', where each vector in that codebook is going to mark off a region in this–in the space that is–that the data lives in R^d. So, then we learn this codebook, and we rep, and we a represent. Each of the signals as the sequence of which codes it was assigned to. So, we learn a codebook. We get a new signal. We quantize each column by assigning each column to the most nearest–to the nearest centroid. And then instead of reporting the sequence of 'D' dimensional continuous vectors, we simply represent this sequence as a sequence of centroids. So, for example, we could say the first two centroids were cluster two.

The third centroi–the third vector in this sequence was mapped to centroid six. And then the following three vectors in this sequence was mapped centroid four or something like that. So, we've taken a 'D' dimensional sequence sequence of 'D' dimensional vectors and quantize them to now represent each signal as a sequence of discrete values. So, we could now transition to a discrete HMM for this type of a model. So, let's now look at a simple speech recognition model. This is not the state of the art. This is not what your iPhone's doing. This is, for example, a scenario where we're doing automated phone conversation, like, where you call your credit card, and you tell them, names of countries that you're going to be traveling to, something like that. And they want to extract that. They want to recognize the names of those countries. They want recognize the purpose of your call.

So, a simple question answer format. What we have as training data are quantized feature sequences of the different words that we want to model. So, we have many instances of a person saying 'Yes', 'No', 'travel', and we quantize each of these sequences. And so then what we do is we learn an HMM for each individual word. For the instance of word–for the word 'Yes', perhaps we have hundreds of thousands of instances of a person saying the word 'Yes', and for each of these we extract a sequence. And now we want to learn an HMM that models the sequence of the word 'Yes'. So, we learn an HMM for generating the word 'Yes'.

We learn a different HMM for the word 'No'. We learn an individual HMM for each word that we're interested in learning. And then we wanna–we have an HMM representation of all the possible words that we think could occur, and if we, for example, don't know what was said, we would tell the automated system to say "I'm sorry, repeat again," something like that. And so, now what happens is we want to predict a word. We ask the–the automated system asks a question. Here's the 'Yes'. It captures the 'Yes'. It extracts the features from the audio word 'Yes'. It quantizes the sequence 'Yes'. All of all of this done very quickly.

And now it wants to say, "what is the word that corresponds to this quantized sequence?" And so, what would it do? What we want to do is we want to say, "What's the probability of this new sequence of– that I have?" We'll imagine it corresponds to 'Yes', but we don't know that, because it's an automated system. We calculate the probability of the sequence, given the particular HMM. Imagine that we want to know what–we want to query, whether this word that was said corresponds to word 'W'. So, we take the HMM we learned for word 'W'. We calculate the probability of the sequence given the word 'W' which requires the forward-backward algorithm, because we've integrated out the state sequence. And we get the probability of that sequence given the HMM for a particular word.

We do this for every word, and then we define–we assign the label for this...this sequence to be the most probable word. So, we simply pick the HMM that most likely generated the sequence that we've observed, and that HMM will correspond to a word, and we declare that the word spoken was that particular word. So, this is a very simple example. And notice that what we've actually done is define a base classifier here. So, this is an example of a base classifier where we have a uniform prior on all of the words, and we might want to improve this by learning the frequencies of different words and having a prior probability on each of the words and then multiplying this by a prior. So, this is actually what we've defined here as a simple base classifier. Notice, however, that this is much more complicated than the base classifiers we thought about previously. So, in this case, what we had to do was for each of the potential classes–each word is a class now–we had to learn a class specific model that was much more complicated than a simple Multivariate Gaussian. So, we had to run a more complicated algorithm in order to learn the parameters of this class conditional distribution. And in this case, the class conditional distribution is a–is an entire model. It's not simply a distribution on one vector that is like a Multivariate Gaussian. It's an actual model for generating a sequence of observations.

And so, we could have actually, you know, if it's a base classifier, we could have tried different things. We could have tried a Gaussian mixture model if we though there was no temporal information worth modeling. We could have simple treated the original vectors as if there were no time dependents between them and learned a word-specific or class-specific Gaussian mixture model in which case we would calculate the likelihood of all of the features extracted for the incoming audio signal given that Gaussian mixture model. And so we could have tried that and seen which does better. However, the HMM most likely because it takes into consideration a sequential information, we anticipate and observe that it's going to outperform the Gaussian mixture model, because, in this case, we have chosen a more appropriate class-conditional density or class-conditional model, in this case.

Video 5 (8:06): Markov Models

Let's go back and review the previous two sequential models we discussed. The first was a Markov model where we had a sequence 'S_1' through 'S_4' or S...'S_1' through 'S_t'. And this sequence was observed and we modeled it as having a Markov property, meaning that for all values 't' the distribution on the sequence at time 't' given all the previous time points, is only conditionally dependent on the previous time point. So, we can throw away all the information in the previous points of the sequence except for the last point. In this case, we assumed that the number of states were finite and discrete, meaning that we could index them with a number from one to 's'. This allowed us to then represent the transition probabilities in a matrix; so, we could say that the probability that the state at time 't' is equal to 'j' given the state at time 't' minus one is equal to 'i' is just the ij^{th} value in a matrix, in 'S' bias matrix. We then broadened this definition, to define a hidden Markov model and this was a model where the state sequence still followed a Markov process, but the sequence itself was latent.

And what we observed were state dependent observations from a state dependent distribution. So, in this case again the state sequence 'S_1' through 'S_n' followed a hidden Markov chain, but we didn't

get to observe it. And then, at any given particular time points say that time point 'S_n' we observed a corresponding value 'X_n' which was from a probability distribution where the parameter used was dependent on the state. So, for example, at time 't' the distribution on the obs...observed sequence at time 't' 'x_t' given the state at time 't' 'S_t' is enerated from some probability distribution with parameter selected by the state 's_t'. So, these distributions were always the same family, but the parameter itself was state dependent. Similar to a mixture model this allowed for a few distributions to generate the entire this–data set, but it added an extra transition rule not present in the Gaussian mixture model or in the general mixture model framework where the distribution chosen at one time point was dependent on what distribution was chosen at the previous time point.

So, both of these models, the Markov model, and the hidden Markov model used discrete state space. In this case, what this meant was that even though the data itself could be continuous, the state space or the number of data generating distributions was finite. So, if we saw this plot for the Markov chain where we had these latent three states and the data that we observed were the exact discrete state locations, so, we had three unique locations or values that we could observe. And then when we generalized this, we allowed a particular state to then be a random permutation where we had Gaussian noise, in which case we had the same three states, but now we observed some random variable that was generated from a distribution dependent on that state. But still the number of states underlying the number of distributions or states underlying the data generating process was discrete. So, now the question is, what if we model the underlying state as being continuous as well. Is this something that we can athematically do easily and what do we gain from a modeling standpoint in making this modification?

So, in this case, rather than allowing the number of states to be discrete and therefore have a mapping to an index from one to 'S', we're going to now modify the continuous state Markov model so, that the state 's' is a vector in R^d. And so, the state now can take any value in R^d. So, again let's compare the discrete state Markov model. In that case, the states lived in a discrete space, we could give them a label from one to 'S' if they were 'S' states. Now with the continuous state Markov model the states live in a continuous space.

So, what is an example of a continuous state Markov model? The simplest possible example is something that's like what can be called Brownian motion. So, in this case the state at time 't' is simply equal to the state at the previous time point plus some random noise where the random noise, the noise is generated *iid* from some Gaussian distribution. This random–continuous state random walk–model is a continuous state Markov model because the–the latent state at time 't' is only dependent on the value of the latent state of time 't' minus one. But it's continuous because the noise added to it is a continuous valued random variable. So, this is the model that we're going to discuss in more detail. In this lecture what we're going to discuss today is something called a linear Gaussian Markov model, which is also called a common filter. With this model what we have is a latent state process and an observed process.

So, this is the unobserved process and the observed process similar to a hidden Markov model, except now we have a continuous functional form for the state process and the data process. In this case, with the latent process, we say that the state at time 't' which is a vector in R^p is equal to the state at time 't' minus one pre-multiplied by some known matrix 'C', so we assume that we know the matrix 'C', plus some random noise that is unique to time 't' minus one. And we're going to assume that that's generated *iid*. from a Gaussian with zero mean and covariance equal to 'Q'.

So, this is a latent continuous state process, notice it's continuous because the vectors 'S'...the vectors 's_t' for all 't' are a continuous value random variable in R^p. Then given this latent unobserved process we have at the observed process that corresponds to it where the observed vector at time 't', which is a vector in R^d, is equal to some known matrix 'D' multiplied by the state, the latent underlying state at time 't'. So we take our unobserved state at time 't' multiply it by some matrix, some known matrix 'D' and then add some noise to that as well. Where the noise process here is also a Gaussian assumed *iid*, but with a different covariance. So we in both cases have zero mean noise, but the process noise which is the noise corresponding to the latent state process is a covariance 'Q'. And the measurement noise which corresponds to the observed process has noise 'V'.

Video 6 (10:20): Examples

So, again let's notice the difference between the two models, the continuous state Markov model or the common filter or the linear Gaussian Markov model, the hidden Markov model which was a discrete state model. In both cases, we have exactly the same graphical model representation. We say that the distribution on 'S_2' is only dependent on what happens at with 'S' at time one. And at any given times point the observed vector, the observation at that time point is conditioned only on the latent state at that time point. So, we have exactly the same graphical representation for both the discrete state and continuous state Markov models.

The key difference is that the distribution on 'S_2' is a continuous value–continuous probability distribution given 'S_1' whereas with the hidden Markov model, the distribution on 'S_2' was a discrete distribution on 'S' possible states given the state of 'S_1'. So, hidden Markov model had a discrete distribution on each of these states. The continuous state Markov model has a continuous distribution on each of these states. And the–these continuous state Markov models, the linear Gaussian Markov model and its many variants which we'll discuss at the end, have many applications. Primarily, they're used for tracking objects. They're also used in automatic control systems. They're used in economics and finance, for example, to model stocks, as well as other applications. So, let's look at a concrete example before we become more abstract in deriving the algorithm for the common filter. So, here's a classic very specific tracking problem. We're assuming that what we get are noisy and we're going to assume very noisy measurements of an object's position as a function of time. And we're going to assume that we're taking these measurements high up, so, we can model them as being in 'R_2', just an 'x', 'y' coordinate.

So, at time 't' we get a two-dimensional vector that gives a very noisy measurement of the location of an object in a two-dimensional plane. The underlying state vector, which is also a function of time is now going to be a sixth dimensional vector, so, this is the unobserved state. And each dimension in space in the measurement corresponds to three underlying states. For the first dimension in the observation, so, for say latitude we have the position as one of the latent states. We have the velocity and the acceleration in that dimension–in that direction. In the other direction, we say longitude we have the position, the velocity in that direction, and the components of the acceleration in that direction. So, for the two-dimensional locations we have this sixth dimensional

latent states vector which we don't get to observe, which has the position of...the position of the object encoded in it, as well as the velocity and the acceleration in the two dimensions.

Okay! So now motivated by the underlying physics of the problem of motion we model this as follows. We say that the state vector time 't' plus one is equal to this matrix 'C' times the state vector time's 't'...time 't' plus some random perturbation. And for this specific problem we choose 'C', again motivated by the underlying physics to have this exact form. So, notice what this is modeling, this is–this first dimension corresponds to the position in the first dimension of the object that's equal to mapping again to this vector here that's equal to the position at the previous time point times the amount of time that is passed.

So, we're going to make an assumption that the sampling time is discrete–discretized say every few milliseconds, and so, the amount of time that has passed since the previous time point times the velocity. So, this is the amount of motion that has taken place in that direction in the interval from time 't' to 't' plus one, which has this delta 't' amount of time–time-lapse plus one half two times the time-lapse squared, times the acceleration. So what this–these three values are modeling is the position as a function of the previous position, the velocity in that–the first dimension, and the acceleration in the first dimension. These three values multiplied by these three values will give the estimated position at a time step Delta 't' in the future. If we look in the second dimension of the state vector that corresponds to the velocity along the first dimension of the observed space.

So, for example, this could be the velocity in one direction, in say the longitudinal direction. That's going to be equal to zero times the position at the previous time point which makes sense, the position doesn't impact the velocity; plus one times the previous velocity. So, this says that the velocity at the next time point should be like the velocity of the previous time point plus delta 't' times the acceleration, and so, that takes into account how much we're speeding up or slowing down, at the previous time point. So, we have a–we the velocity at the next time point factors in both the velocity at the previous time point plus the acceleration at the previous time point to give a prediction of the velocity of the next time point. And then finally for the acceleration along that first dimension we're going to have this dampening effect that says that the acceleration should always be either decreasing or increasing toward zero.

So, this is a dampening that takes the old acceleration, it multiplies it by a number that's slightly less than one to dampen the acceleration. Okay! So this is the underlying state vector, and then the observation is simply equal to this matrix 'D' times the state vector plus noise. Notice that all this matrix 'D' is doing is picking out the position along the first dimension and the position along the second dimension of the state vector to then say that the position is equal to whatever these two values are equal to plus noise. So, this is a simple common filter for modeling a moving ob–object where we're going to try to then use an inference algorithm to both predict where the location is, as well as predicting what this underlying state is. So, this predictions for these underlying state vectors will not only capture the position of the object, but also the velocity and the acceleration of it, as a function of time.

All right! Let's look at an example of what we would learn from this algorithm on tracking a real moving object. So, we're going to discuss the actual algorithm in the second part of this lecture, first I want to show the results. So, what we have here in these four time slices I'm showing this solid red line which is the same at all four time points. This is the actual trajectory of the object as a function of time, so I'm showing the entire trajectory of a particular object. The black dots are showing the

measurements at various time points. So, I'm showing all of the location measurements of that object. You can't tell the ordering of these measurements from the way that they're plotted, but you can tell that they're very noisy measurements. So, on average, the measurements are following the location of the object, but they're very noisy. So, for example at this point when the object is right here we might've measured the object as being right here, but at the next time point when it's right here we might've measured it as being way over there. So that's the noise in 'x'. So again, these dots correspond to the measurements 'x_t' here and the noise is fairly large for this problem. Now what I'm showing in these four slices is the state vector at that specific time point and I'm not showing the entire sixth dimensional state vector. I'm just showing four of the dimensions of the state vector. I'm showing the first and the fourth dimension as the red dot.

So, remember the first and the fourth dimension correspond to the position in the first dimension of the observations space and the second dimension of the observation space, so the position of the object. So, that's what this red dot is. And so, we can see that it's a function of time, we're accurately measuring the position of the object so, we're averaging out the noise, and then this blue line is showing the velocity vector. So, I'm plotting an–a vector of this velocity in dimension one and the velocity in dimension two, that's the blue line which gives us a estimate of the direction and the speed in which the object is moving. So, the speed actually is the length of the arrow and the direction of the arrow is the direction the object is estimated as moving at that time point. So, we can see that as a function of time, what we're learning with this...this tracking model is both the location and the direction and speed of the movement of the object as a function of time based only on these black dot measurements.

Video 7 (5:00): The Learning Problem

As with the hidden Markov model discussed in the previous lecture, all we're given is the observations the seque...sequence 'x_1' through 'x_t', where each observation is in R^d. And now, the goal is to learn the underlying state sequence 's_1' through 's_t'. So, similar to HMM, we want to learn what's going on with this hidden state sequence. Unlike the HMM, the only thing we're learning is this underlying state sequence. We're not learning a– we're not learning the emission distributions, called 'B' last time, or the transition matrix 'A', because we now have a continuous transition distribution. So, for this model, again, all distributions are Gaussian, so here's another way of writing the generative process we discussed previously.

The distribution on the state at time 't' plus one given the state at time 't' is a Multivariate Gaussian with mean equal to 'C' times 's_t' and covariance 'Q'. So, we're assuming that 'Q' and 'C' are known or can be estimated in advance. And then the distribution on the observed sequence, 'x_t', the observation of time 't', given the underlying state at time 't', is a–also a Multivariate Gaussian with mean equal to 'D' times the underlying state st and covariance equal to 'V'. And again, we're assuming 'D' and 'V' are either known or can be estimated. So, for the previous problem that we discussed, 'C' and 'D' were known from the physics of the problem, and 'Q' and 'V' were estimated somehow in advance. So, notice that, again, with

HMM, with the discrete HMM, we had these other model variables, Pi, 'A' and 'B', and we wanted to learn those as well. Here, we don't have any of those. All we want to learn is the underlying state vector as a function of time. So, again, just to emphasize, let's notice how this is very different from the discrete HMM. With the discrete HMM, we wanted to learn Pi, 'A' and 'B', where Pi was an initial state distribution, where we had a discrete set of states, 'A' was a transition matrix among the discrete set of states and then 'B' contained the state dependent probability distributions. In this–in the linear Gaussian Markov model, of the situation is different because we don't have to learn 'B' because since each state is going to be unique because the state is in a continuous space with a continuous transition distribution, the distribution on each 'x' is also going to be different. The way that we constrain it, is by letting the distribution family be parameterized by 's' in this way. However, because 's_t' is continuous valued and unique, this distribution –the parameters–are going to be unique for every value of 't'–of 'x'. Also, we don't have the matrix 'A' to learn because we don't have a discrete set of states, again, 's' is a continuous valued random vector with a continuous distribution, meaning that every state and every time point is going to be something brand new, a new continuous rand–valued random vector.

However, it's constrained by this distribution, which is what is going to make the problem of learning the underlying state possible. What we can learn in this case are these two posterior distributions. And for the rest of the lecture, we're going to be interested in the first one. But we can learn the posterior distribution of the state at time 't', given the sequence that we've observed up until time 't'. So, this is a distribution on the current state, given all of the data I've seen up until that time.

We also might want to be–we also might be interested in learning the posterior distribution of the state at time 't', given all of the data, including future data, so, this would not be real time model. The first problem has a name, it's called common filtering. It's what we'll focus on today. It's learning the continuously evolving distributions on the states as a– in a real time setting. The second problem is known as common smoothing. It requires one extra step that we won't discuss. It's similar to the backward algorithm for the hidden Markov model. This would be something, where we have all the data in advance and we want to then do some sort of a post processing. So, the first problem has to do with tracking objects and that's what we're going to focus on for the rest of the lecture.

Video 8 (9:11): The Kalman Filter I

The goal of the Kalman filter is going to be to learn the time evolving conditional posterior distribution of the state sequence, the hidden state sequence, given the observed data up until that time for a given sequence 'x_1' through 'x_t'. And, again, to review, we have this sort of a model where we say that the underlying state at time 't' plus one given the underlying stated time 't' is a Multivariate Gaussian like this.

And the distribution on the observed value at time 't', given the underlying state at time 't', is a Multivariate Gaussian parametrized like this. So, let's look at how we can calculate this conditional posterior distribution. Well, we use Bayes' rule, as usual, in order to figure this thing out. We can say that the conditional posterior of my hidden state at time 't', given all of the data up until and including time 't' is proportional to the likelihood of what I observe at time 't', given the state at time

't' times the conditional posterior distribution of the state at time 't', given all of the data up until time 't' minus one but not including time 't'. So, this is the first step in solving this problem.

We then represent this prior distribution as a marginal distribution. So, we say that the distribution on the state at time 't', given the data up until time 't' minus one is equal to an integral. So, to be more explicit, we have this integral of 's_t' and 's_{t-1}' given the data from one to 't' minus one 'Ds_{t-1}'. So, we simply add the stated time 't' minus one and then integrate it out to get a marginal. But then, we write this joint distribution as a conditional distribution on the stated time 't', given the stated time 't' minus one times the posterior distribution on the stated time 't' minus one, given the sequence up until time 't' minus one. So, originally, we had 'x_1' through 'x_{t-1}' in here.

But if you tell me the stated time 't' minus one, then the stated time 't' is conditionally independent of the sequence 'x' up until time 't' minus one. So, we could get rid of the sequence in this conditioning here. So, we write this prior distribution as this marginal, and notice that this marginal includes the transition distribution and the posterior distribution for the previous time point. So, this is part of the story of Bayes online learning was Bayes' in– Bayes' rule, where the posterior at the previous time point is used to form the prior at the next time point.

All right, so we've decomposed this problem into parts that we do and don't know. So, we've said that the posterior distribution on the stated time 't', given the sequence of observations up until and including time 't' is proportional to the likelihood of the observation at time 't', given the stated time 't'. By definition of the model, that's a Gaussian with this mean and this variance. So, notice that the Gaussian has the state and the mean there times the prior which we've written as a marginal of the transition distribution from state at time 't' minus one to time 't', which we've defined by the model to be a Gaussian with this distribution times the posterior of the previous state at the last time point, given the observations up until the last time point. So, we don't know what this one is yet. So, let's make some observations before we continue working through this problem. Again, first, we want to note that the left-hand side has the posterior, the conditional posterior on 's_t', and the right has the conditional posterior on 's_{t-1}'. So, we already kind of have a hint that if we can solve the posterior at the previous time, we might be able to solve the posterior at the current time. We know that we want this integral to be in closed form, in order to make things work out well.

So, this is a motivation, in how we want to pick distributions. And, we also want this likelihood and this prior so that integral constitutes the prior. We want that pair to also lead to a posterior that is known. So, we would like these two to be conjugate to each other if possible, meaning that the posterior has a nice closed form solution and is in the same family as the prior, and we want whatever this turns out to be to then also make future calculations easy. So, we want this to work out well.

And so, it all hinges on what this distribution is. We've already picked Gaussians here, and so, we need this to be something nice. So, for now, let's just hypothesize temporarily that this unknown Gausian–this posterior distribution at time 't' minus one is a Gaussian. Let's say that this conditional posterior at the last time point is a Gaussian with mean vector Mu and covariant Sigma by hypothesis. And if this were true, what would the solution to this posterior be?

Okay, so proceeding from this hypothesis now we have all the distributions in here, and we can start calculating things. First, we have to calculate this marginal. So, a nice property of Gaussians is that the marginals are still Gaussian, meaning that if we have a distribution like this, where we have 'st'

as a Gaussian with this mean and this covariance and then a prior distribution on 's_t' like this, which is a Gaussian with mean Mu and covariant Sigma. We...we multiply those two together, and then we integrate out the state at time 't' minus one. So, we're integrating out 's_{t-1}' minus one here and here in this multiplication. The result is still a Gaussian, so we still have a distribution on 's_t', which is a Gaussian with mean equal to 'C' Mu. So, notice that we've taken 's_{t-1}' here and replaced it with the mean of the prior, so we've got 'C' now times Mu as the mean. And the covariance equal to 'Q'. That's from this likelihood; plus 'C' time Sigma times 'C' transposed. So, there's the Sigma there,and there's the 'C' there.

We assume we know all of these. We're assuming we know this...this distribution by hypothesis, and we know 'C' and 'Q' by the model definition. So, this marginal likelihood to calculate the prior on 's_t' is a Gaussian with a mean and a covariance all of which we can calculate. So, now we've solved for the integral. We still want to calculate this conditional posterior at time 't'. So, now it all hinges on whether this likelihood times this prior is something that we can now normalize and calculate the posterior.

So, again, the likelihood of 'x_t' is a Gaussian with mean 'D' times the state at 's_t' times the covariance. And the prior on the state at 's'–at time 't' is equal to a Multivariate Gaussian with this mean and this covariance calculated on the previous slide. And so, again, we're working with Gaussians, and a nice property is that the posterior is again, a Gaussian. So, the posterior distribution on 's_t' is a Gaussian, so when we normalize this thing so that it...it's a function of 's_t' that integrates to one we find that we have a Gaussian with mean Mu Prime and covariant Sigma Prime where the covariance is equal to this and the mean is equal to this.

So, it looks complicated, but notice that it's a function of everything that we know. So, here we have the prior covariance. Here we have a function of 'D' and 'V' which is simply from the likelihood. In this term, we have, again, a function of 'D' and 'V', and the observed data, and a prior term. So, all of these things are something we can calculate. So, we plug in the relevant values and we get this conditional posterior.

Video 9 (06:47) : The Kalman Filter II

Okay! So we've solved for the conditional posterior, but we made a hypothesis. Remember that we said that we're going to hypothesize that this conditional posterior time 't' minus one, given all the data up until time 't' minus one is a Multivariate Gaussian with a mean and a covariant. So, we hypothesize that, and, based on that hypothesis, we were able to show that, therefore, the conditional posterior at the next time point would also be a Multivariate Gaussian with a mean and a covariance equal to a function of things that we know. Therefore, in order to turn that hypothesis into the truth, all we need to do is define a Gaussian prior on the first state. So, for the prior definition on the state at time zero starting state, remember, if 't' is equal to one, then here we have a prior distribution on the state at time zero. We don't have any data yet, so that's just the starting state–the prior on the starting state.

We get to make that...We get to define that prior distribution, and so, if we define that to be a Gaussian, for example, a Gaussian with zero, meaning identity covariance, or any Multivariate

Gaussian that we want that would make sense for our problem. Then, by definition, this first prior at time equal to zero is a Multivariate Gaussian, and, therefore, everything follows...follows through in order where the conditional posterior at time one is a Multivariate Gaussian, because we've defined the prior at time zero to be a Multivariate Gaussian. That can...that can get the ball rolling. That will make everything Gaussian downstream. So, we've shown with the Kalman filter that when we want to calculate the sequence of conditional posteriors on the state...the latent state. Since we're working with Gaussians, since we've made modeling assumptions where everything's Gaussian, it all is in enclosed form and can be done using simple functions of things that we know.

So, there is one other quantity that we might want to calculate with this Kalman filter, and that's predicting the observation at the next time point. So, we've shown how to calculate the conditional posterior on the state at time 't', given the observations up until time 't'. But what if want to then take that and predict the observation at time 't' plus one, before we get to see it? And so what we're asking is what is the distribution on the observation at time 't' plus one, given all of the observations up to and including time 't'? This is a distribution that we're interested in is our predicted distribution on the next observation. Well, we write this as a marginal over the state where, again, what I'm doing here is writing 'p' of 'x' at time 't' plus one comma 's_{t+1}'given the data up until time 't' integrating out 's' at time plus one. So, I have this joint distribution of the observation at the time 't' plus one and the state at time 't' plus one. I integrate out the state at time 't' plus one to get this marginal distribution. But now, I write this joint distribution as a conditional distribution on the observation at time 't' plus one, given the state at time 't' plus times the conditional distribution of the state at time 't' plus one, given the data up until, and including time 't', but not 't' plus one. So, here, I'm not conditioning on any of the other data, because if you tell me what the state is at time 't' plus one, my observation at time 't' plus one is independent of all of the other data.

All right, and now we, again, do the same breakdown of this distribution where I write this thing as a marginal of the state at time 't' plus one and the state at time 't', given the data up until and including time 't'. But now I marginalize out the state at time 't'. But I take this joint distribution, and I factorize this as well into the conditional of the state at time 't' plus one, given the state of time 't' times the conditional of the state at time 't' min...state at time 't', given all the data up until and including time 't', which is just the posterior distribution, which I have discussed how to calculate this. So, this is how we can take the marginal predictive distribution and write it as an integral over distributions we know. Again, the distribution of the data at time 't' plus one, given the state at time 't' plus one is just a Gaussian with this parametrization by the definition of the model. The condi–transition distribution on the state at 't' plu–time 't' plus one, given the state of time 't' is just this Gaussian, again, by definition of the model. And the conditional posterior of the state at time 't', given all the data up until and including time 't' has been calculated using the common filtering algorithm from the previous slides.

So, we plug in a Gaussian here with mean Mu, Prime and Sigma, a covariant Sigma Prime. Plug in a Gaussian here with parameter as defined by the model and a Gaussian here. And notice that we're integrating out 'st' and 'st' plus one. So, we end up with just the function of the data at time 't' plus one and the data up until time 't'. Again, these are all Gaussians. We've already discussed how we can take this marginal and get a Gaussian out of it. And then, if we have a Gaussian here, we can, again, take the same marginal and get another Gaussian. I'm omitting the mean and the covariance of this Gaussian, but it's still a Gaussian. It's going to be much more complicated than what we've discussed previously, so, we showed this marginal distribution. This was one marginal that we took

of a Gaussian with another Gaussian, became this Gaussian. We would simply have an even more complex Gaussian. For example, here the mean and the covariance will have this function. And so, then we take that, integrate that out, and integrate again so, it was very complicated function, but we ultimately get a Gaussian with a mean vector that's a function of things that we can...that we know and a covariance matrix that's also a function of things that we know.

Video 10 (14:06): Algorithm: Kalman Filtering

Okay! So, let's look at the Kalman filtering algorithm. So, this is an algorithm that can be run in real time for tracking. With to start out, we set the initial state distribution to something. So, I arbitrarily put standard Gaussian here. But, it could be any other Gaussian. So, as long as it's a Gaussian, it will work. Then prior to observing each new observation at a time point. For example, at time 't' prior to observing the value at time 't', 'x_t', we form the predictive distribution, which is a Gaussian with a mean and covariance, as discussed in the previous slide. That's our prediction for what we're going to observe next. Then after observing the value of 'x' at time 't', we update the conditional posterior distribution on the underlying state given the data up until and including time 't'. And that was also something that we showed how to calculate on the previous slide. So, let's look at a very trivial example at this plot. So, here we have the green line, which is the true trajectory. The blue line is the observed trajectory, and then, the red values are the state distribution. So, as a function of time, the 'x''s are the means of the Gaussians of where we believe we are at that time point, and the circles are the covariances.

So, what's some intuition about what this is doing, this tracking algorithm? So, we see that in the prior distribution, we add 'Q' to the covariance. So, when we wanted to calculate this prior distribution on the state at time 't' given the data up until, and including time 't' minus one, we had this distribution. So, this is copied from a previous slide. Notice that we always add this 'Q' to the covariance. What this is doing is adding some drift to the latent states. Adding some variance to the latent state. So, at every time point, we're adding some variance, and allowing the state to move away, to drift away from what it was at time 't' minus one. So, this is what's making it a dynamic model. Not something that's converging to a point estimate as 't' increases. We're always adding this covariance to the state distribution. So, we're always allowing the model to drift away from the previous state. And then, if we look at the posterior distribution of the state given the observation at time 't'. So, this is the prior, not seeing 't', 'x_t', and this is the posterior having seeing 't'. If we then go back and look at those equations, I haven't shown them here. What we'll see is that 'x_t' then pulls the distribution away so that 's_t' is...is pulled away by 'x_t' so that it can model 'x_t' a little bit better.

So, the prior constraints what 's_t' can be, but adds a little bit of drift so, it can move a little bit, be permuted by 'xt', and then 'xt' actually pulls the state in the direction that would have predicted it better. In the next two slides, I want to tie a few of the models we've been discussing together. We can see that in many cases one model is a modification of a different model. So, really they're...they're very closely related, but by making simple modifications, we get a fundamentally new modeling capability. So, first I want to compare the Gaussian mixture model with a continuous hidden Markov model. So, this is a discrete state hidden Markov model with continuous observed values. So, we can write the Gaussian mixture model in this way. Remember, we have 'S' different

Gaussians. So, I'm going to use the same notation that we've used for Markov model. If we imagine that we have 'S' different, capital 'S' different Gaussians. What we do for the t^{th} observation is we first assign the data point to one of the 'S' Gaussians. So, we have a discrete distribution with parameter Pi. Where Pi is our mixing weights, our probability distribution on the 'S' different Gaussians in our mixture model.

So, we first generate a Gaussian from a discrete distribution with mixing weight vector Pi. So, we pick a Gaussian indexed by 'st', and then, the observation at time 't' given the cluster, given the component it was assigned to at time 't'. It's just a Multivariate Gaussian with mean equal to 'Must' and covariance 'Sigmast'. So we pick out the mean and covariance indexed by the state or the cluster at the t^{th} observation. So, we have this type of a graphical model where we pick a state from a probability distribution. So, I could...I could also say that we have a vector Pi here. And then, we simply pick a state based on this vector Pi– this distribution Pi. And then, given the state or given the cluster, we generate the observation from the Gaussian picked out by that cluster. With the continuous state, I'm sorry, with the continuous discrete state hidden Markov model, what we now have is that the state at a particular time is generated from a discrete distribution where we pick out the probability distribution indexed by the previous time point. So 'A' is a transition matrix. It says that if we're at state 's_{t-1}' at time 't' minus one, then we transition to the state at time 't' according to the distribution indexed by the row of our transition matrix 's_t' minus one. And then, given that state, we again generate our observation from a Gaussian, where we use the mean and covariance picked out by the indexed state. So, notice that given 's_t' in both of these models, we simply generate the data from a Multivariate Gaussian with the mean and covariance picked out by the index 's'.

However, for the Gaussian mixture model, we're picking the clusters or picking the states, if you will, from a discrete distribution that's always using the same probability vector Pi. Whereas, for the continuous hidden Markov model, which is a discrete state model, we're using also a discrete distribution. But we're picking out the distribution according to a transition matrix. So, we have–if they're at capital 'S' different states– we have capital 'S' different probability distributions that we could possibly choose, and we choose the one indexed by the previous state in order to generate the next state. So, they're very similar in that way. So, there's a similar comparison that we could make between probabilistic PCA and the linear Gaussian Markov model. The previous ex-example was a comparison between two discrete state models, if you will. The Gaussian mixture model that had no sequential distribution. And the continuous hidden Markov model, which did have a sequential distribution on the discrete states.

Here we have, if we look at probabilistic PCA, we see that there's a similar model structure as the Gaussian mixture model. So, I'm using the notation that we used in this lecture to discuss the algorithm we discussed in a previous lecture probabilistic PCA. So, here we have a vector 's_t' for the t^{th} observation. Which is generated from a Multivariate Gaussian with mean zero and covariance 'Q'. And then given the vector 's_t', we generate the observation at time 't' or the t^{th} observation from a Multivariate Gaussian with mean equal to a matrix 'D' times the latent vector 's_t' and covariance 'V'. So, in the previous lecture, when we discussed probabilistic PCA, we used different notation. I believe this was 'W' and this was 'C'. Something like that. Here I'm using the notation of today's lecture. So, you see that the latent state, if you will, the continuous state vector is a continuous random variable. But there is no sequential information being modeled. So, we have, again, something that looks like this. Where 's_t', each state is generated *iid*. from the same distribution. So,

there's no sequential information. But now, it's a continuous valued vector instead of a discrete index. If we now take probabilistic PCA and we build a sequential model on top of it. We see that we have the linear Gaussian Markov model discussed today.

Where the state at time 't', given the state at time 't' minus one, is a Multivariate Gaussian. Where we have the same covariance. But now the mean is equal to 'C' times the state at time 't' minus one. So, we again see that the latent state at the previous time point informs the latent state at the next time point, according to this function. But then, given the latent state, we have exactly the same distribution on the observation as with probabilistic PCA. So, again, with discrete state models, Gaussian mixture model and the continuous hidden Markov model, are essentially the same in the data generating distribution, but fundamentally different in the way that the distributions are picked. One has an iid distribution on the latent Gaussians. And one has a sequential distribution on how these latent Gaussians are picked. With a continuous state model, we have probabilistic PCA where, again, the distributions are identical on the observations. But on the latent states for probabilistic PCA, they're *iid* from the continuous Gaussian distribution in which case, these aren't even referred to as states whereas, with the linear Gaussian Markov model, we have a Markov sequence generating these latent states. Finally, I want to discuss a few of the many extensions of this modeling framework. Each of these would require a modification to the algorithm we discussed previously.

They're not trivial modifications, but they would look familiar based on what we've discussed. First, there's the non-linear Kalman filtering framework. So, what a non-linear Kalman filter does is, it takes exactly the same state sequence as before, but then it generates the observation by using a non-linear function of the state sequence. So, previously we had a matrix times the latent state. So, it was a linear function of the state, in which case everything was nice in closed form using Gaussians. Now, we have a non-linear function of the latent state, in which case, calculating the posterior of the latent state is in longer in closed form. So, this happens quite a bit with systems where we don't get linear measurements. For example, we might get functions of locations based on angles and what not and distances.

So with a non-linear Kalman filter, there are many ways to approach this problem. One way is to use what's called the extended Kalman filter. What this does is makes a simple approximation to this nonlinear function. It treats this nonlinear function of 's_t' as a, approximately equal to the function of evaluated at some point 'z', plus this gradient matrix times a the difference between the latent state and the point 'z' that we pick. So, it replaces this nonlinear function with an approximating linear function. If we take this function and place it in the mean, again, we have linear functions. and so, everything's Gaussian. We might also want to modify the model so that we can take continuous time into account. By continuous time, what we mean is that the time difference between two time points is not always the same. At one, between, for example, time 't' and 't' plus one, we might have had a few seconds elapse. Whereas, between time 't' plus one and 't' plus two, we might have had a few milliseconds elapse.

In that case, we modify the modeling this way. We have exactly the same observation level distribution. But the state distribution on how the states are evolving now has a function delta 't' times 'Q' on the covariance where delta 't' is the time lapsed from the previous time point. So, we have 'Q', which is a constant matrix now being scaled up or down, based on how much time has passed. And so, this is a Brownian motion representation. We can also add control to this system. So,

in this case, we have...our state evolving distribution like this where we say the distribution on the next state, given the current state, is a Gaussian with this mean equal to a sum of a matrix 'C' times the state 's_t', as before, plus a new thing, 'G' times 'u_t', where 'u_t' is a vector that we get to control. We say what 'u' is. For example, it could be a thruster along a certain dimension. Power that we're going to...to put in...into the system. And so, this is how we can control the latent state in order to control what we observe in the observation sequence 'xt'.