Chapter 18

Lecture 18: Recursive tracking

Monday, April 11, 2011 MIT EECS course 6.869, Bill Freeman

18.1 Driving with my friend Tim

Let me tell you about driving with my friend, Tim. His tracking and control loop may be the same as everybody else's, but it is so spread out over time that you can clearly identify all the different steps in his algorithm. When he's driving along the road, he is very engaged in whatever he is talking with you about, and not really looking at the road, but just predicting where he thinks it might be. Finally, he looks at the road, suddenly sees where everything is relative to the car, and makes some relatively dramatic correction with the steering wheel. Then he resumes talking, and the process repeats until the next road observation. Making a tight curve is an exciting experience of approximating a curve with a polygon of only a few sides.

Let's see what Tim's driving algorithm looks like in terms of a graphical model. We have a sequence of graphical models. First he starts driving, and follows his prior model of where the road is. Then he looks at the road and makes the observation, y_1. Now, based on y_1, he has new estimate of where the road is (and adjusts his steering accordingly). Then he talks, makes eye contact, and gestures, in the meantime, guessing about where the road is at time 2. He then finally looks at where the road is, makes observation y_2, and proceeds accordingly. At each time step, he first predicts where the road is without looking, then looks at the road and updates his estimate for where the car is on the road. In the context of estimation of a hidden state, we call this online inference task "filtering". Examples include the Kalman filter, and particle filters, which we'll discuss below. In vision, these filters are commonly used, for tracking people, tracking roads in automobiles, etc.

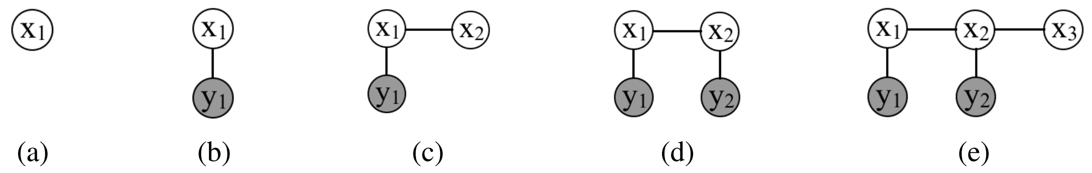
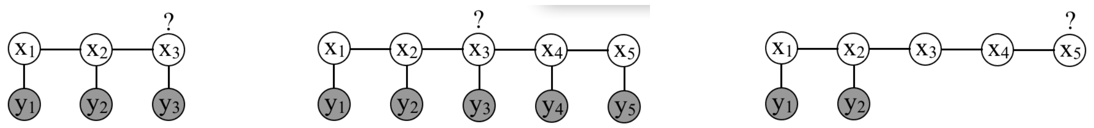


Figure 18.1: (a - e) Graphical models depicting first five steps in an online tracking or filtering algorithm. For each step, the task is to estimate the state at the highest-numbered x node, given the y observations seen so far.



(a) filtering (b) smoothing (c) prediction

Figure 18.2: Graphical models showing examples of the (a) filtering, (b) smoothing, and (c) prediction tasks.

18.2 Prediction, filtering, and smoothing

Having specified the graphical model for filtering (eg as shown in Fig. (18.2) (a)), we have thus described the conditional independence structure and a joint probability factorization. Before we proceed to develop the corresponding online estimation algorithms, while we are at the level of generality of a graphical model, let's examine two other problems that we may want to solve, and their corresponding graphical models.

The first is called "smoothing", and an example is shown in Fig. (18.2) (b). We use observations from both the future and the past to make a best estimate of the current node's state. The second is "prediction", illustrated in Fig. (18.2) (c). We estimate the hidden state 1 or more time steps beyond a time for which we have observations.

The graphical models for filtering, smoothing, and prediction are all trees, so we know that we can find the exact marginal probabilities at any hidden node, given the data, using sum-product belief propagation (BP), and that we can find states maximizing the posterior probability using max-product BP. Furthermore, if there are other, unnamed, online estimation problems we want to solve (suppose a few, but not all, of the observations are missing), we can do that using BP, too.

18.3 Filtering, in more detail

Even though we know, from belief propagation, what messages need to be sent where for the filtering graphical model, it is useful to work through the message passing algorithm for this special case.There are several reasons for this. Very little of the extensive literature on this problem is written in terms of belief propagation, so it is helpful to see a derivation of the optimal inference algorithm for this specific problem. This special case is important enough that it is worthwhile to solve it at this specific level. Finally, this calculation will let us see what we avoid having to do for each new graphical when we use the general machinery of BP.

We see a sequence of observations, y_t, and we seek the marginal probability for the hidden variable x_t at each time step i. (The calculations needed to find the states maximizing the joint can be derived similarly, although with a slight extra complication in order to account for ties.)

We write a recursive update procedure for our online estimate of the most recent state. We'll write our formulas assuming continuous state variables, but the modifications for discrete state variables are straightforward–the integrals go to sums. We assume we know the state transition probabilities, $P(x_t|x_{i-1})$ for each pair of nodes, and the probabilities describing the relationship of the local observations to the underlying state, $P(y_t|x_t)$. (Note we use the notational shortcut (or abuse) of describing the probability function P by its arguments.)

The recursion begins with knowing the prior probability, $P(x_1)$. For the general recursion, we assume we have computed $P(x_t|y_1 \dots y_t)$, and we seek to compute $P(x_{t+1}|y_1 \dots y_{t+1})$.

We form the recursion over two steps. First, we find $P(x_{t+1}|y_1 \dots y_t)$, the state probabilities at the next time step, given all the observations up to the current time. From the relation $P(a,b) = P(b|a)P(a)$, we have

$$P(x_t, x_{t+1}|y_1 \dots y_t) = P(x_{t+1}|x_t)P(x_t|y_1 \dots y_t), \tag{18.1}$$

where we have used the conditional independence relation that knowing x_t tells node x_{t+1} everything it can know about the past, so we can drop the conditioning on $y_1 \dots y_t$ for x_{t+1}: $P(x_{t+1}|x_t, y_1 \dots y_t) = P(x_{t+1}|x_t)$.

To find $P(x_{t+1}|y_1 \dots y_t)$, our prediction for node x_{t+1} given all the observations through time t, we just marginalize Eq. (18.1) over the variable x_t:

$$\begin{aligned} P(x_{t+1}|y_1 \dots y_t) &= \int_{x_t} P(x_t, x_{t+1}|y_1 \dots y_t) && (18.2) \\ &= \int_{x_t} P(x_{t+1}|x_t)P(x_t|y_1 \dots y_t) && (18.3) \end{aligned}$$

This is called the prediction step. Before seeing the new observations, we predict the new state.

We can finish the recursion once we include the observations at the new time, $t+1$ By Bayes rule, $P(a|b) = P(b|a)P(a)/P(b)$, we have

$$P(x_{t+1}|y_1 \dots y_{t+1}) = \frac{1}{P(y_{t+1})} P(y_{t+1}|x_{t+1})P(x_{t+1}|y_1 \dots y_t) \tag{18.4}$$

This is called the update step or measurement step (We have again used the Markov structure to write $P(y_{t+1}|x_{t+1}, y_1 \dots y_t)$ as $P(y_{t+1}|x_{t+1})$).

Next, we need to specify the representation and conditional probabilities of our HMM model. But before we do that, let's take a brief detour to show that this is the same result that we would have obtained had we blindly applied the belief propagation rules to the graphical model of Fig. 18.2 (a).

18.3.1 Equivalence to belief propagation

To run the BP update equations, we first need to describe the clique potentials. One set of potentials that repects the conditional independence relations specified by the graph, and which multiplies out to the desired joint probability, is to assign the following. For the local evidence potentials, take $\Phi_t(x_t, y_t) = \frac{1}{P(y_{t+1})} P(y_t|x_t)$. For the compatibility functions, take $\Psi_t(x_t, x_{t+1}) = P(x_{t+1}|x_t)$. The reader can verify that the product of the clique potentials gives desired joint probability for all the observed and hidden variables, and that that product obeys the conditional independence relations specified by the filtering graphical model. We include the prior factor $P(x_1)$ at the first node, multiplied together with its local evidence $P(y_1|x_1)$.

Belief propagation updates

For continuous variables, the BP message update equation to update the message from node x_t to node x_{t+1} is

$$M_{t,t+1}^{xx} = \int_{x_t} \Psi_t(x_t, x_{t+1}) M_{t,t}^{yx} M_{t,t+1}^{xx} dx_t \tag{18.5}$$

The marginal probability at a node is the product of all the incoming messages. So the marginal probability, $P(x_t|y_1 \dots y_t)$ of Eq. (18.4) must be the product of the incoming messages. From that, we see that $M_{t-1,t}^{xx} = P(x_{t-1}|y_1 \dots y_{t-1})$, and $M_{t,t}^{yx} = \frac{1}{P(y_t)} P(y_t|x_t)$ are the incoming messages to node

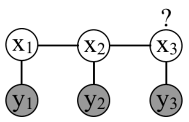


Figure 18.3: Showing the message passing algorithm of BP.

x_t. Substituting in those identifications, we can see that the BP update equation, Eq. (18.5) is the same calculation as the prediction and update equations we derived for the online state updates, Eqs. (18.4) and (18.3). (Substituting the prediction equation into the update equation gives Eq. (18.5), with the messages identified as above).

18.3.2 Linear state dynamics

Now let's return to the task of specifying pariticular values of the functions that define our model, $P(x_{t+1}|x_t)$ and $P(y_t|x_t)$. We have a choice of state (continuous or discrete) and a choice of the dynamics and measurement functions.

We can make a very powerful filter with a very simple set of assumptions. Let us assume that all the x_t and y_t together form jointly Gaussian random variables. While this assumption restricts us to unimodal state estimates, at the same time, it allows efficient computation with continuous variables and can lead to a powerful algorithm for tracking and control, called the Kalman filter.

For Gaussian random variables that respect the graphical models of Fig. 18.2, the linear dynamics and observation equations below allow all the relevant probabilities of the filtering problem to stay in a Gaussian form:

$$x_{t+1} = Ax_t + v_t \qquad (18.6)$$

$$y_t = Cx_t + w_t, \qquad (18.7)$$

where v_t is a mean zero, normally distributed Gaussian random variable, of covariance matrix Q, independent of x_t. w_t is a mean zero, normally distributed Gaussian random variable, of covariance matrix R, independent of v_t and of x_t. A and C are some matrices, potentially different for each time step, t.

With this model for the state dynamics and the observations, we have specified things well enough to calculate the conditional probabilities we need to work out the Kalman filter algorithm. By Eq. (18.6), we have the state transition probabilities

$$P(x_{t+1}|x_t) = N_{x_{t+1}}(Ax_t, Q), \qquad (18.8)$$

where $N_x(\mu, \Lambda) = \frac{1}{(2\pi)^{d/2}|\Lambda|^{\frac{1}{2}}} \exp -\frac{1}{2}(x-\mu)^T \Lambda^{-1}(x-\mu)$, where d is the dimensionality of x.

By Eq. (18.7), we have the local evidence potentials,

$$P(y_t|x_t) = N_{y_t}(Cx_t, R) \qquad (18.9)$$

18.3.3 Representation

Because the variables in the probem are all jointly Gaussian, all marginals will also be Gaussian, including our online state estimates. The marginalizations and multiplications of the state prediction and measurement equations confirm this, by involving products and integrals, both of which maintain Gaussian random variables as Gaussians.

18.3.4 Kalman filter equations

Inserting the conditional probabilities of Eqs. (18.8) and (18.9) for the state transition and measurement probabilities into the equations for the online state estimate updates yields equations that give recursive update equations for the Gaussian parameters of the online state estimates. Mathematically, as described, this will involve a matrix version of "completing the square", and is algebraically involved. Other approaches, some simpler, can also be used to derive the equations below. It is easier to verify that these equations satisify the state estimate update relations, and we leave that as an exercise for the motivated reader.

Let $\hat{x}_{t+1|t}$ be the mean of $P(x_{t+1}|y_1 \ldots y_t)$, ie, the estimate for the mean of x at node $t+1$, given all the observations $y_1 \ldots y_t$. And $\hat{x}_{t|t}$ will be the mean of $P(x_t|y_1 \ldots y_t)$, ie the estimate for the mean of x at node t, given all the observations $y_1 \ldots y_t$. The node marginal posterior covariances $\Lambda_{t+1|t}$ and $\Lambda_{t|t}$ are defined similarly. Putting the conditional probabilities of Eqs. (18.8) and (18.9) into the prediction and measurement equations of Eqs. (18.3) and (18.4) (and algebraic simplification, as described above) gives:

Prediction step

$$\hat{x}_{t+1|t} = A\hat{x}_{t|t} \quad (18.10)$$

$$\Lambda_{t+1|t} = A\Lambda_{t|t}A^T + Q \quad (18.11)$$

$$(18.12)$$

Measurement step

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + \Lambda_{t+1|t}C^T(C\Lambda_{t+1|t}C^T + R)^{-1}(y_{t+1} - C\hat{x}_{t+1|t}) \quad (18.13)$$

$$\Lambda_{t+1|t+1} = \Lambda_{t+1|t} - \Lambda_{t+1|t}C^T(C\Lambda_{t+1|t}C^T + R)^{-1}C\Lambda_{t+1|t} \quad (18.14)$$

We initialize with $\hat{x}_{1|0} = \hat{x_1}$ and $\Lambda_{1|0} = \Lambda_1$.

Some **notes**:

1. We often define a "Kalman gain", $k_{t+1} = \Lambda_{t+1|t}C^T(C\Lambda_{t+1|t}C^T + R)^{-1}$, which then simplifies the measurement update equations.

2. The covariance estimates $\Lambda_{t+1|t+1}$ and $\Lambda_{t+1|t}$ are not a function of the observations.

3. These Kalman filter equations make sense. We add covariance matrices from $A\hat{x}_{t|t}$ and V when we propagate the state forward in time. We reduce the covariance when we make a measurement.

4. the new state estimate is a weighted combination of the previous state estimate and a correction factor from the different between the observations and the predicted observations, given our current state estimate. ($\hat{x}_{t+1|t+1}$ is a weighted combination of $\hat{x}_{t+1|t}$ and $(y_{t+1} - C\hat{x}_{t+1|t})$.

Eqs. (18.12) and (18.14) are the update rules for the Kalman filter. This is often used in vision, and has a long history in online control and estimation. Those equations let spaceships navigate to the moon, among other things.

18.3.5 1-d Kalman filter worked example

It's useful to get some initial experienc with those equations using a worked example. Here's a simple problem: Our observations are a sequence of numbers, and we seek an online estimate of the mean of those numbers, using the Kalman filter. Of course, in "batch mode", given those numbers we would just add them all together with equal weights, and then divide by the number of terms in the sum. Let's see if the Kalman filter gives us that in an online implementation.

In practise, part of the task of applying the Kalman filter to a problem is deciding how to set the parameters of the filter–the state and observation dimensions, the matrices A, C, and the process and observation noise covariances, Q and R. For the online averaging problem, what should the hidden state and observations be, and how should we set the four unknown matrices? Try to work this out before you go on.

We want to pose an estimation problem where the best estimate (making the Gaussian random variable assumptions that the Kalman filter does) will be the average of a set of numbers. Suppose we have a single parmeter that doesn't change, to which we add Gaussian noise and then make observaions. The best estimate for that parameter would then be the average of those noisy samples.

So let x be that parameter we seek to estimate. It will be a scalar (the average of a set of numbers) that won't change over time. So then the state dynamics are simple: $x_{t+1} = x_t$. To accomodate that, we need to set the covariance of the scalar noise that we add to x_t to be $Q = 0$.

We add noise to our parameter-to-be-estimated through our observations: we let $y_t = x_t + w$, where w is a zero-mean scalar Gaussian random variable, of variance R. Since we average the draws from a Gaussian to estimate its mean, regardless of its sigma, let's set $R = 1$.

Now we have the parameters all set, and we just need to specify the initial conditions of the Kalman filter. Let's say the mean is zero, or $\hat{x}_{1|0} = 0$, but with infinitely large covariance to the Gaussian describing our prior distribution, $\Lambda_{1|0} = \text{inf}$. That lets us specify that we have no idea what the mean of these numbers will be, and will force our estimate to rely entirely on the observed data.

Let's re-write the Kalman filter prediction and update equations specialized to our problem. Substituting $A = 1$, $Q = \text{inf}$, $C = 1$, and $R = 1$, we have

Prediction step for our problem

$$\hat{x}_{t+1|t} = \hat{x}_{t|t} \quad (18.15)$$

$$\Lambda_{t+1|t} = \Lambda_{t|t} \quad (18.16)$$

$$(18.17)$$

Measurement step for our problem

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + \frac{\Lambda_{t+1|t}}{(\Lambda_{t+1|t} + 1)}(y_{t+1} - \hat{x}_{t+1|t}) \quad (18.18)$$

$$\Lambda_{t+1|t+1} = \Lambda_{t+1|t} - \frac{\Lambda^2_{t+1|t}}{(\Lambda_{t+1|t} + 1)} \quad (18.19)$$

worked example

$$\begin{array}{ccccc}
\text{iteration} & \text{t=0} & \text{t=1} & \text{t=2} & \text{t=3} \\
x_{t+1|t} & 0 & y_1 & \frac{(y_1+y_2)}{2} & \frac{(y_1+y_2+y_3)}{3} \\
x_{t+1|t+1} & y_1 & y_1+\frac{1}{2}(y_2-y_1) & \frac{y_1+y_2}{2}+\frac{1}{3}(y_3-\frac{y_1+y_2}{2}) & \\
\Lambda_{t+1|t} & \text{inf} & 1 & \frac{1}{2} & \\
1\Lambda_{t+1|t+1} & 1 & \frac{1}{2} & \frac{1}{3} &
\end{array} \tag{18.20}$$

For the nth iteration, the mean value is $\frac{\Sigma_n y_i}{n} + \frac{1}{n+1}(y_{n+1} - \frac{\Sigma_n y_i}{n}) = \frac{\Sigma_{n+1} y_i}{n+1}$.

modified example

What would happen if the state dynamics noise were give some non-zero variance? Think this through... You would trust the old data less and less.

$$\begin{array}{cccc}
\text{iteration} & \text{t=0} & \text{t=1} & \text{t=2} \\
x_{t+1|t+1} & y_1 & \frac{y_1+2y_2}{3} & \frac{y_1+2y_2+5y_2}{8}
\end{array} \tag{18.21}$$

If $Q >> R$, then the process noise is much more than the observation noise, and we can't trust anything we have seen in the past. Then

$$\hat{x}_{t+1|t+1} = \hat{x}_{t+1|t} + \frac{\Lambda_{t+1|t}}{(\Lambda_{t+1|t}+R)}(y_{t+1} - \hat{x}_{t+1|t}) \approx y_{t+1} \tag{18.22}$$

When R becomes small relative to $\Lambda_{t+1|t}$, then $\frac{\Lambda_{t+1|t}}{(\Lambda_{t+1|t}+R)} \approx 1$ and in that regime, you forget the previous data altogether. The Kalman filter adjusts properly to give the optimal estimates (within the Gaussian model) in these different cases.

18.4 Particle representation

So far, we've used two different probability representations for the state's we're trying to estimate: (1) a Gaussian distribution, that we've explored just now, and (2) a multinomial distribution over discrete states in our earlier work with belief propagation and graphical models. Each has their advantages. The Gaussian model is parametric, so it only requires a few parameters to specify a pdf over a potentially high dimensional space. The drawback, of course, is that you lose flexibility–you're only able to represent unimodal distributions. (Real-world vision posterior probabilities are often multi-modal). You can adjust the mean and covariance matrix, but that's it. The multinomial distribution is completely general–you can discretize the space as much as you'd like, and describe the probabilities with almost complete generality, but at the cost of having to specify many numbers, even in regions of the parameter space where perhaps not much is going on.

A third probability distribution representation, that we introduce here, offers, in some sense, the best of both worlds. In a *particle representation* for state, we represent the probability distribution as a set of samples–a sum of delta functions in the parameter space. Where there is a lot of mass in the probability density function, we add more samples. In regimes where nothing happens, we have few or no samples.

To add flexibility to the representation, we include weights on each of the point samples. Thus, we can represent a large local mass in probability in two different ways: by adding more samples there, or by increasing the weights of the samples that are already there. We'll see how we use that flexibility in a minute when we examine the measurement and prediction steps, using the particle filter representation.

Thus, we assume we have a set of samples, x^1, x^2, ..., x^N, and a set of weights w^1, w^2, ..., w^N. Each $w^i \geq 0$, and their sum is one, $\Sigma_{i=1}^N w^i = 1$. Then

$$P(x) = \Sigma_{i=1}^N w^i \delta(x - x^i) \tag{18.23}$$

The x's and particle positions x^i's are, in general, vector quantities, but we're showing them here as scalars for simplicity.

18.4.1 The update and prediction steps for particle filtering

Equations (18.3) and (18.4) apply for other representations for state, too. Let's examine them in our particle representation.

Update or measurement step

Let

$$P(x_t|y_1 \ldots y_{t-1}) = \sum_{k=1}^K w_t^k \delta(x_t - x_t^k) \tag{18.24}$$

Then, by Eq. (18.4), we have the update step,

$$P(x_t|y_1 \ldots y_{t-1}) = \sum_{k=1}^K \tilde{\ } w_t^k \delta(x_t - x_t^k), \tag{18.25}$$

where $\tilde{\ } w_t^k \propto w_g^k P(y_t|x_t^k)$ and $\sum_{k=1}^K \tilde{\ } w_t^k = 1$. By $P(y_t|x_t^k)$, we mean the likelihood function at time t evaluated for the particle x_t^k. Because we're using this overcomplete representation, we don't need to create any new particles, and we can just adjust the weight of each particle x_t^k.

Prediction step

Now let's apply the prediction step. By Eq. (18.3), we have

$$\begin{aligned} P(x_{t+1}|y_1 \ldots y_t) &= \int_{x_t} P(x_{t+1}|x_t) \sum_{k=1}^K \tilde{\ } w_t^k \delta(x - x_t^k) dx_t && (18.26) \\ &= \sum_{k=1}^K \tilde{\ } w_t^k P(x_{t+1}|x_t^k) && (18.27) \end{aligned}$$

$P(x_{t+1}|x_t^k)$ is the conditional probability, evaluated at the point sample x_t^k. It is a function of the continuous variable, x_{t+1}. So right now, our probability distribution is represented by a weighted sum of the continuous functions $P(x_{t+1}|x_t^k)$.

We need to convert that back to a sum of discrete samples in order to be able to continue with the recursion. The simplest way to do that is to generate samples x_{t+1}^k from the continuous probability density, $P(x_{t+1}|x_t^k)$. (We descussed previously how to sample from a function). After that sampling step, then we again have our running probability density in the form

$$P(x_{t+1}|y_1 \ldots y_t) = \sum_{k=1}^K \tilde{\ } w_t^k \delta(x - x_{t+1}^k) \tag{18.28}$$

Particle management

After some iterations, most of the particle weights can become small, leading to fewer effective particles representing the distribution. A solution to this is to *resample*.

We can draw K independent samples from the $x_t^1, \ldots, x_t^K$ particles, sampling from each particle with probability in proportion to its weight, w_t^k. Thus, you'll sample from some particles several times, and from others never. The multiple samples from a single particle will soon go off their own ways in the parameter space, so we don't need to worry about the temporary degeneracy introduced then.

You don't want to resample too often, because you then lose particle diversity. One often uses a measure of weight degeneracy as a guide for when to resample. We can define the quantity,

$$K_{\text{eff}} = \frac{1}{\sum_{k=1}^{K} (w_t^k)^2} \tag{18.29}$$

which is a measure of the effective number of particles. $K_{\text{eff}} = K$ if all weights on all the particles are equal. $K_{\text{eff}} = 1$ in the worst case where one particle has weight 1 and all others have weight 0.

Typically, one only resamples when K_{eff} falls below some thereshold, for example, below $\frac{K}{2}$.

Examples

See videos in the slides for examples of particle filters for tracking leaves, heads, and figures.