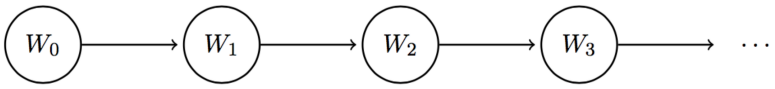
CS 188 Introduction to Artificial Intelligence
Summer 2023 Note 12

Markov Models

In previous notes, we talked about Bayes' nets and how they are a wonderful structure used for compactly representing relationships between random variables. We'll now cover a very intrinsically related structure called a **Markov model**, which for the purposes of this course can be thought of as analogous to a chain-like, infinite-length Bayes' net. The running example we'll be working with in this section is the day-to-day fluctuations in weather patterns. Our weather model will be **time-dependent** (as are Markov models in general), meaning we'll have a separate random variable for the weather on each day. If we define W_i as the random variable representing the weather on day i, the Markov model for our weather example would look like this:



What information should we store about the random variables involved in our Markov model? To track how our quantity under consideration (in this case, the weather) changes over time, we need to know both it's **initial distribution** at time $t = 0$ and some sort of **transition model** that characterizes the probability of moving from one state to another between timesteps. The initial distribution of a Markov model is enumerated by the probability table given by $P(W_0)$ and the transition model of transitioning from state i to $i+1$ is given by $P(W_{i+1}|W_i)$. Note that this transition model implies that the value of W_{i+1} is conditionally dependent only on the value of W_i. In other words, the weather at time $t = i+1$ satisfies the **Markov property** or **memoryless property**, and is independent of the weather at all other timesteps besides $t = i$.

Using our Markov model for weather, if we wanted to reconstruct the joint between W_0, W_1, and W_2 using the chain rule, we would want:

$$P(W_0, W_1, W_2) = P(W_0)P(W_1|W_0)P(W_2|W_1, W_0)$$

However, with our assumption that the Markov property holds true and $W_0 \perp\!\!\!\perp W_2|W_1$, the joint simplifies to:

$$P(W_0, W_1, W_2) = P(W_0)P(W_1|W_0)P(W_2|W_1)$$

And we have everything we need to calculate this from the Markov model. More generally, Markov models make the following independence assumption at each timestep: $W_{i+1} \perp\!\!\!\perp \{W_0, ..., W_{i-1}\}|W_i$. This allows us to reconstruct the joint distribution for the first $n+1$ variables via the chain rule as follows:

$$P(W_0, W_1, ..., W_n) = P(W_0)P(W_1|W_0)P(W_2|W_1)...P(W_n|W_{n-1}) = P(W_0)\prod_{i=0}^{n-1} P(W_{i+1}|W_i)$$

A final assumption that's typically made in Markov models is that the transition model is **stationary**. In other words, for all values of i (all timesteps), $P(W_{i+1}|W_i)$ is identical. This allows us to represent a Markov model with only two tables: one for $P(W_0)$ and one for $P(W_{i+1}|W_i)$.

The Mini-Forward Algorithm

We now know how to compute the joint distribution across timesteps of a Markov model. However, this doesn't explicitly help us answer the question of the distribution of the weather on some given day t. Naturally, we can compute the joint then **marginalize** (sum out) over all other variables, but this is typically extremely inefficient, since if we have j variables each of which can take on d values, the size of the joint distribution is $O(d^j)$. Instead, we'll present a more efficient technique called the **mini-forward algorithm**.

Here's how it works. By properties of marginalization, we know that

$$P(W_{i+1}) = \sum_{w_i} P(w_i, W_{i+1})$$

By the chain rule we can re-express this as follows:

$$\boxed{P(W_{i+1}) = \sum_{w_i} P(W_{i+1}|w_i)P(w_i)}$$

This equation should make some intuitive sense — to compute the distribution of the weather at timestep $i+1$, we look at the probability distribution at timestep i given by $P(W_i)$ and "advance" this model a timestep with our transition model $P(W_{i+1}|W_i)$. With this equation, we can iteratively compute the distribution of the weather at any timestep of our choice by starting with our initial distribution $P(W_0)$ and using it to compute $P(W_1)$, then in turn using $P(W_1)$ to compute $P(W_2)$, and so on. Let's walk through an example, using the following initial distribution and transition model:

W_0	$P(W_0)$
sun	0.8
rain	0.2

W_{i+1}	W_i	$P(W_{i+1}\|W_i)$
sun	*sun*	0.6
rain	*sun*	0.4
sun	*rain*	0.1
rain	*rain*	0.9

Using the mini-forward algorithm we can compute $P(W_1)$ as follows:

$$\begin{aligned}
P(W_1 = sun) &= \sum_{w_0} P(W_1 = sun|w_0)P(w_0) \\
&= P(W_1 = sun|W_0 = sun)P(W_0 = sun) + P(W_1 = sun|W_0 = rain)P(W_0 = rain) \\
&= 0.6 \cdot 0.8 + 0.1 \cdot 0.2 = \boxed{0.5} \\
P(W_1 = rain) &= P(W_1 = rain|w_0)P(w_0) \\
&= P(W_1 = rain|W_0 = sun)P(W_0 = sun) + P(W_1 = rain|W_0 = rain)P(W_0 = rain) \\
&= 0.4 \cdot 0.8 + 0.9 \cdot 0.2 = \boxed{0.5}
\end{aligned}$$

Hence our distribution for $P(W_1)$ is

W_1	$P(W_1)$
sun	0.5
rain	0.5

Notably, the probability that it will be sunny has decreased from 80% at time $t = 0$ to only 50% at time $t = 1$. This is a direct result of our transition model, which favors transitioning to rainy days over sunny days. This gives rise to a natural follow-up question: does the probability of being in a state at a given timestep ever converge? We'll address the answer to this problem in the following section.

Stationary Distribution

To solve the problem stated above, we must compute the **stationary distribution** of the weather. As the name suggests, the stationary distribution is one that remains the same after the passage of time, i.e.

$$P(W_{t+1}) = P(W_t)$$

We can compute these converged probabilities of being in a given state by combining the above equivalence with the same equation used by the mini-forward algorithm:

$$P(W_{t+1}) = P(W_t) = \sum_{w_t} P(W_{t+1}|w_t)P(w_t)$$

For our weather example, this gives us the following two equations:

$$\begin{aligned} P(W_t = sun) &= P(W_{t+1} = sun|W_t = sun)P(W_t = sun) + P(W_{t+1} = sun|W_t = rain)P(W_t = rain) \\ &= 0.6 \cdot P(W_t = sun) + 0.1 \cdot P(W_t = rain) \\ P(W_t = rain) &= P(W_{t+1} = rain|W_t = sun)P(W_t = sun) + P(W_{t+1} = rain|W_t = rain)P(W_t = rain) \\ &= 0.4 \cdot P(W_t = sun) + 0.9 \cdot P(W_t = rain) \end{aligned}$$

We now have exactly what we need to solve for the stationary distribution, a system of 2 equations in 2 unknowns! We can get a third equation by using the fact that $P(W_t)$ is a probability distribution and so must sum to 1:

$$\begin{aligned} P(W_t = sun) &= 0.6 \cdot P(W_t = sun) + 0.1 \cdot P(W_t = rain) \\ P(W_t = rain) &= 0.4 \cdot P(W_t = sun) + 0.9 \cdot P(W_t = rain) \\ 1 &= P(W_t = sun) + P(W_t = rain) \end{aligned}$$

Solving this system of equations yields $P(W_t = sun) = 0.2$ and $P(W_t = rain) = 0.8$. Hence the table for our stationary distribution, which we'll henceforth denote as $P(W_\infty)$, is the following:

W_∞	$P(W_\infty)$
sun	0.2
rain	0.8

To verify this result, let's apply the transition model to the stationary distribution:

$$\begin{aligned} P(W_{\infty+1} = sun) &= P(W_{\infty+1} = sun|W_\infty = sun)P(W_\infty = sun) + P(W_{\infty+1} = sun|W_\infty = rain)P(W_\infty = rain) \\ &= 0.6 \cdot 0.2 + 0.1 \cdot 0.8 = \boxed{0.2} \\ P(W_{\infty+1} = rain) &= P(W_{\infty+1} = rain|W_\infty = sun)P(W_\infty = sun) + P(W_{\infty+1} = rain|W_\infty = rain)P(W_\infty = rain) \\ &= 0.4 \cdot 0.2 + 0.9 \cdot 0.8 = \boxed{0.8} \end{aligned}$$

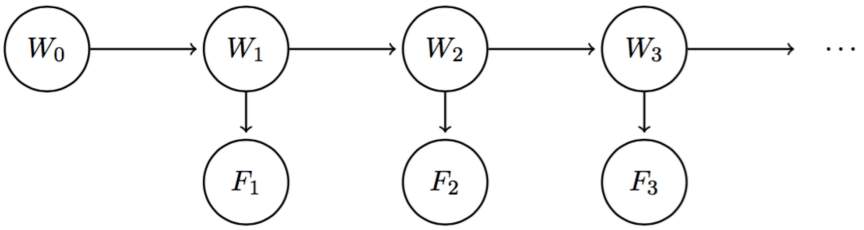
As expected, $P(W_{\infty+1}) = P(W_\infty)$. In general, if W_t had a domain of size k, the equivalence

$$P(W_t) = \sum_{w_t} P(W_{t+1}|w_t)P(w_t)$$

yields a system of k equations, which we can use to solve for the stationary distribution.

Hidden Markov Models

With Markov models, we saw how we could incorporate change over time through a chain of random variables. For example, if we want to know the weather on day 10 with our standard Markov model from above, we can begin with the initial distribution $P(W_0)$ and use the mini-forward algorithm with our transition model to compute $P(W_{10})$. However, between time $t = 0$ and time $t = 10$, we may collect new meteorological evidence that might affect our belief of the probability distribution over the weather at any given timestep. In simpler terms, if the weather forecasts an 80% chance of rain on day 10, but there are clear skies on the night of day 9, that 80% probability might drop drastically. This is exactly what the **Hidden Markov Model** helps us with - it allows us to observe some evidence at each timestep, which can potentially affect the belief distribution at each of the states. The Hidden Markov Model for our weather model can be described using a Bayes' net structure that looks like the following:



Unlike vanilla Markov models, we now have two different types of nodes. To make this distinction, we'll call each W_i a **state variable** and each weather forecast F_i an **evidence variable**. Since W_i encodes our belief of the probability distribution for the weather on day i, it should be a natural result that the weather forecast for day i is conditionally dependent upon this belief. The model implies similar conditional indepencence relationships as standard Markov models, with an additional set of relationships for the evidence variables:

$$
\begin{array}{lcll}
F_1 & \perp\!\!\!\perp & W_0|W_1 & \\
\forall i & = & 2,\ldots,n; & W_i \perp\!\!\!\perp \{W_0,\ldots,W_{i-2},F_1,\ldots,F_{i-1}\}|W_{i-1} \\
\forall i & = & 2,\ldots,n; & F_i \perp\!\!\!\perp \{W_0,\ldots,W_{i-1},F_1,\ldots,F_{i-1}\}|W_i
\end{array}
$$

Just like Markov models, Hidden Markov Models make the assumption that the transition model $P(W_{i+1}|W_i)$ is stationary. Hidden Markov Models make the additional simplifying assumption that the **sensor model** $P(F_i|W_i)$ is stationary as well. Hence any Hidden Markov Model can be represented compactly with just three probability tables: the initial distribution, the transition model, and the sensor model.

As a final point on notation, we'll define the **belief distribution** at time i with all evidence $F_1,\ldots,F_i$ observed up to date:

$$B(W_i) = P(W_i|f_1,\ldots,f_i)$$

Similarly, we'll define $B'(W_i)$ as the belief distribution at time i with evidence $f_1,\ldots,f_{i-1}$ observed:

$$B'(W_i) = P(W_i|f_1,\ldots,f_{i-1})$$

Defining e_i as evidence observed at timestep i, you might sometimes see the aggregated evidence from timesteps $1 \leq i \leq t$ reexpressed in the following form:

$$e_{1:t} = e_1,\ldots,e_t$$

Under this notation, $P(W_i|f_1,\ldots,f_{i-1})$ can be written as $P(W_i|f_{1:(i-1)})$. This notation will become relevant in the upcoming sections, where we'll discuss time elapse updates that iteratively incorporate new evidence into our weather model.

The Forward Algorithm

Using the conditional probability assumptions stated above and marginalization properties of conditional probability tables, we can derive a relationship between $B(W_i)$ and $B'(W_{i+1})$ that's of the same form as the update rule for the mini-forward algorithm. We begin by using marginalization:

$$B'(W_{i+1}) = P(W_{i+1}|f_1,\ldots,f_i) = \sum_{w_i} P(W_{i+1},w_i|f_1,\ldots,f_i)$$

This can be reexpressed then with the chain rule as follows:

$$B'(W_{i+1}) = P(W_{i+1}|f_1,\ldots,f_i) = \sum_{w_i} P(W_{i+1}|w_i,f_1,\ldots,f_i)P(w_i|f_1,\ldots,f_i)$$

Noting that $P(w_i|f_1,\ldots,f_i)$ is simply $B(w_i)$ and that $W_{i+1} \perp\!\!\!\perp \{f_1,\ldots f_i\}|W_i$, this simplies to our final relationship between $B(W_i)$ to $B'(W_{i+1})$:

$$\boxed{B'(W_{i+1}) = \sum_{w_i} P(W_{i+1}|w_i)B(w_i)}$$

Now let's consider how we can derive a relationship between $B'(W_{i+1})$ and $B(W_{i+1})$. By application of the definition of conditional probability (with extra conditioning), we can see that

$$B(W_{i+1}) = P(W_{i+1}|f_1,\ldots,f_{i+1}) = \frac{P(W_{i+1},f_{i+1}|f_1,\ldots,f_i)}{P(f_{i+1}|f_1,\ldots,f_i)}$$

When dealing with conditional probabilities a commonly used trick is to delay normalization until we require the normalized probabilities, a trick we'll now employ. More specifically, since the denominator in the above expansion of $B(W_{i+1})$ is common to every term in the probability table represented by $B(W_{i+1})$, we can omit actually dividing by $P(f_{i+1}|f_1,\ldots,f_i)$. Instead, we can simply note that $B(W_{i+1})$ is proportional to $P(W_{i+1},f_{i+1}|f_1,\ldots,f_i)$:

$$B(W_{i+1}) \propto P(W_{i+1},f_{i+1}|f_1,\ldots,f_i)$$

with a constant of proportionality equal to $P(f_{i+1}|f_1,\ldots,f_i)$. Whenever we decide we want to recover the belief distribution $B(W_{i+1})$, we can divide each computed value by this constant of proportionality. Now, using the chain rule we can observe the following:

$$B(W_{i+1}) \propto P(W_{i+1},f_{i+1}|f_1,\ldots,f_i) = P(f_{i+1}|W_{i+1},f_1,\ldots,f_i)P(W_{i+1}|f_1,\ldots,f_i)$$

By the conditional independence assumptions associated with Hidden Markov Models stated previously, $P(f_{i+1}|W_{i+1},f_1,\ldots,f_i)$ is equivalent to simply $P(f_{i+1}|W_{i+1})$ and by definition $P(W_{i+1}|f_1,\ldots,f_i) = B'(W_{i+1})$. This allows us to express the relationship between $B'(W_{i+1})$ and $B(W_{i+1})$ in it's final form:

$$\boxed{B(W_{i+1}) \propto P(f_{i+1}|W_{i+1})B'(W_{i+1})}$$

Combining the two relationships we've just derived yields an iterative algorithm known as the **forward algorithm**, the Hidden Markov Model analog of the mini-forward algorithm from earlier:

$$\boxed{B(W_{i+1}) \propto P(f_{i+1}|W_{i+1})\sum_{w_i} P(W_{i+1}|w_i)B(w_i)}$$

The forward algorithm can be thought of as consisting of two distinctive steps: the **time elapse update** which corresponds to determining $B'(W_{i+1})$ from $B(W_i)$ and the **observation update** which corresponds to determining $B(W_{i+1})$ from $B'(W_{i+1})$. Hence, in order to advance our belief distribution by one timestep (i.e. compute $B(W_{i+1})$ from $B(W_i)$), we must first advance our model's state by one timestep with the time elapse update, then incorporate new evidence from that timestep with the observation update. Consider the following initial distribution, transition model, and sensor model:

W_0	$B(W_0)$
sun	0.8
rain	0.2

W_{i+1}	W_i	$P(W_{i+1}\|W_i)$
sun	*sun*	0.6
rain	*sun*	0.4
sun	*rain*	0.1
rain	*rain*	0.9

F_i	W_i	$P(F_i\|W_i)$
good	*sun*	0.8
bad	*sun*	0.2
good	*rain*	0.3
bad	*rain*	0.7

To compute $B(W_1)$, we begin by performing a time update to get $B'(W_1)$:

$$\begin{aligned}
B'(W_1 = sun) &= \sum_{w_0} P(W_1 = sun|w_0)B(w_0) \\
&= P(W_1 = sun|W_0 = sun)B(W_0 = sun) + P(W_1 = sun|W_0 = rain)B(W_0 = rain) \\
&= 0.6 \cdot 0.8 + 0.1 \cdot 0.2 = \boxed{0.5} \\
B'(W_1 = rain) &= \sum_{w_0} P(W_1 = rain|w_0)B(w_0) \\
&= P(W_1 = rain|W_0 = sun)B(W_0 = sun) + P(W_1 = rain|W_0 = rain)B(W_0 = rain) \\
&= 0.4 \cdot 0.8 + 0.9 \cdot 0.2 = \boxed{0.5}
\end{aligned}$$

Hence:

W_1	$B'(W_1)$
sun	0.5
rain	0.5

Next, we'll assume that the weather forecast for day 1 was good (i.e. $F_1 = good$), and perform an observation update to get $B(W_1)$:

$$\begin{aligned}
B(W_1 = sun) &\propto P(F_1 = good|W_1 = sun)B'(W_1 = sun) = 0.8 \cdot 0.5 = \boxed{0.4} \\
B(W_1 = rain) &\propto P(F_1 = good|W_1 = rain)B'(W_1 = rain) = 0.3 \cdot 0.5 = \boxed{0.15}
\end{aligned}$$

The last step is to normalize $B(W_1)$, noting that the entries in table for $B(W_1)$ sum to $0.4 + 0.15 = 0.55$:

$$\begin{aligned}
B(W_1 = sun) &= 0.4/0.55 = \frac{8}{11} \\
B(W_1 = rain) &= 0.15/0.55 = \frac{3}{11}
\end{aligned}$$

Our final table for $B(W_1)$ is thus the following:

W_1	$B'(W_1)$
sun	$8/11$
rain	$3/11$

Note the result of observing the weather forecast. Because the weatherman predicted good weather, our belief that it would be sunny increased from $\frac{1}{2}$ after the time update to $\frac{8}{11}$ after the observation update.

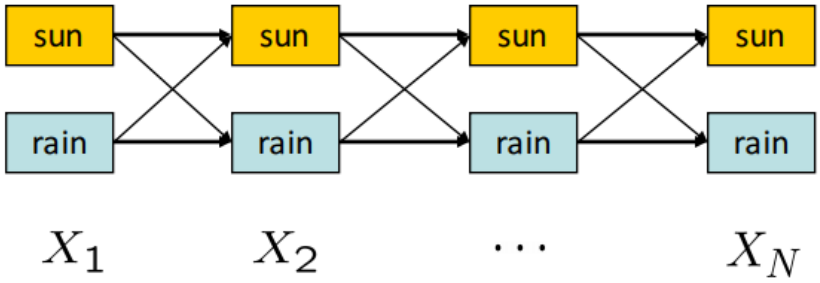
As a parting note, the normalization trick discussed above can actually simplify computation significantly when working with Hidden Markov Models. If we began with some initial distribution and were interested in computing the belief distribution at time t, we could use the forward algorithm to iteratively compute $B(W_1), \ldots, B(W_t)$ and normalize only once at the end by dividing each entry in the table for $B(W_t)$ by the sum of it's entries.

Viterbi Algorithm

In the Forward Algorithm, we used recursion to solve for $P(X_N|e_{1:N})$, the probability distribution over states the system could inhabit given the evidence variables observed so far. Another important question related to Hidden Markov Models is: *What is the most likely sequence of hidden states the system followed given the observed evidence variables so far?* In other words, we would like to solve for $\arg\max_{x_{1:N}} P(x_{1:N}|e_{1:N}) = \arg\max_{x_{1:N}} P(x_{1:N}, e_{1:N})$. This trajectory can also be solved for using dynamic programming with the **Viterbi algorithm**.

The algorithm consists of two passes: the first runs forward in time and computes the probability of the best path to each (state, time) tuple given the evidence observed so far. The second pass runs backwards in time: first it finds the terminal state that lies on the path with the highest probability, and then traverses backward through time along the path that leads into this state (which must be the best path).

To visualize the algorithm, consider the following **state trellis**, a graph of states and transitions over time:



In this HMM with two possible hidden states, sun or rain, we would like to compute the highest probability path (assignment of a state for every timestep) from X_1 to X_N. The weights on an edge from X_{t-1} to X_t is equal to $P(X_t|X_{t-1})P(E_t|X_t)$, and the probability of a path is computed by taking the *product* of its edge weights. The first term in the weight formula represents how likely a particular transition is and the second term represents how well the observed evidence fits the resulting state.

Recall that:

$$P(X_{1:N}, e_{1:N}) = P(X_1)P(e_1|X_1)\prod_{t=2}^{N} P(X_t|X_{t-1})P(e_t|X_t)$$

The Forward Algorithm computes (up to normalization)

$$P(X_N, e_{1:N}) = \sum_{x_1,..,x_{N-1}} P(X_N, x_{1:N-1}, e_{1:N})$$

In the Viberbi Algorithm, we want to compute

$$\arg\max_{x_1,..,x_N} P(x_{1:N}, e_{1:N})$$

to find the maximum likelihood estimate of the sequence of hidden states. Notice that each term in the product is exactly the expression for the edge weight between layer $t-1$ to layer t. So, the product of weights along the path on the trellis gives us the probability of the path given the evidence.

We could solve for a joint probability table over all of the possible hidden states, but this results in an exponential space cost. Given such a table, we could use dynamic programming to compute the best path in

polynomial time. However, because we can use dynamic programming to compute the best path, we don't necessarily need the whole table at any given time.

Define $m_t[x_t] = \max_{x_{1:t-1}} P(x_{1:t}, e_{1:t})$, or the maximum probability of a path starting at any x_0 and the evidence seen so far to a given x_t at time t. This is the same as the highest weight path through the trellis from step 1 to t. Also note that

$$m_t[x_t] = \max_{x_{1:t-1}} P(e_t|x_t)P(x_t|x_{t-1})P(x_{1:t-1}, e_{1:t-1}) \tag{1}$$

$$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) \max_{x_{1:t-2}} P(x_{1:t-1}, e_{1:t-1}) \tag{2}$$

$$= P(e_t|x_t) \max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]. \tag{3}$$

This suggests that we can compute m_t for all t recursively via dynamic programming. This makes it possible to determine the last state x_N for the most likely path, but we still need a way to backtrack to reconstruct the entire path. Let's define $a_t[x_t] = P(e_t|x_t) \arg\max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}] = \arg\max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$ to keep track of the last transition along the best path to x_t. We can now outline the algorithm.

Result: Most likely sequence of hidden states $x^*_{1:N}$
`/* Forward pass */`
for $t = 1$ *to* N **do**
  for $x_t \in \mathscr{X}$ **do**
    if $t = 1$ **then**
      $m_t[x_t] = P(x_t)P(e_0|x_t)$
    else
      $a_t[x_t] = \arg\max_{x_{t-1}} P(x_t|x_{t-1}) m_{t-1}[x_{t-1}]$;
      $m_t[x_t] = P(e_t|x_t)P(x_t|a_t[x_t]) m_{t-1}[a_t[x_t]]$;
    end
  end
end
`/* Find the most likely path's ending point */`
$x^*_N = \arg\max_{x_N} m_N[x_N]$;
`/* Work backwards through our most likely path and find the hidden states */`
for $t = N$ *to* 2 **do**
  $x^*_{t-1} = a_t[x^*_t]$;
end

Notice that our a arrays define a set of N sequences, each of which is the most likely sequence to a particular end state x_N. Once we finish the forward pass, we look at the likelihood of the N sequences, pick the best one, and reconstruct it in the backwards pass. We have thus computed the most likely explanation for our evidence in polynomial space and time.

Particle Filtering

Recall that with Bayes' nets, when running exact inference was too computationally expensive, using one of the sampling techniques we discussed was a viable alternative to efficiently approximate the desired probability distribution(s) we wanted. Hidden Markov Models have the same drawback - the time it takes to run exact inference with the forward algorithm scales with the number of values in the domains of the random variables. This was acceptable in our current weather problem formulation where the weather can only take on 2 values, $W_i \in \{sun, rain\}$, but say instead we wanted to run inference to compute the distribution of the actual temperature on a given day to the nearest tenth of a degree.

The Hidden Markov Model analog to Bayes' net sampling is called **particle filtering**, and involves simulating the motion of a set of particles through a state graph to approximate the probability (belief) distribution of the random variable in question. This solves the same question as the Forward Algorithm: it gives us an approximation of $P(X_N|e_{1:N})$.

Instead of storing a full probability table mapping each state to its belief probability, we'll instead store a list of n **particles**, where each particle is in one of the d possible states in the domain of our time-dependent random variable. Typically, n is significantly smaller than d (denoted symbolically as $n << d$) but still large enough to yield meaningful approximations; otherwise the performance advantage of particle filtering becomes negligible. Particles are just the name for samples in this algorithm.

Our belief that a particle is in any given state at any given timestep is dependent entirely on the number of particles in that state at that timestep in our simulation. For example, say we indeed wanted to simulate the belief distribution of the temperature T on some day i and assume for simplicity that this temperature can only take on integer values in the range $[10, 20]$ ($d = 11$ possible states). Assume further that we have $n = 10$ particles, which take on the following values at timestep i of our simulation:

$$[15, 12, 12, 10, 18, 14, 12, 11, 11, 10]$$

By taking counts of each temperature that appears in our particle list and diving by the total number of particles, we can generate our desired empirical distribution for the temperature at time i:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_i)$	0.2	0.2	0.3	0	0.1	0.1	0	0	0.1	0	0

Now that we've seen how to recover a belief distribution from a particle list, all that remains to be discussed is how to generate such a list for a timestep of our choosing.

Particle Filtering Simulation

Particle filtering simulation begins with particle initialization, which can be done quite flexibly - we can sample particles randomly, uniformly, or from some initial distribution. Once we've sampled an initial list of particles, the simulation takes on a similar form to the forward algorithm, with a time elapse update followed by an observation update at each timestep:

- *Time Elapse Update* - Update the value of each particle according to the transition model. For a particle in state t_i, sample the updated value from the probability distribution given by $P(T_{i+1}|t_i)$. Note the similarity of the time elapse update to prior sampling with Bayes' nets, since the frequency of particles in any given state reflects the transition probabilities.

- *Observation Update* - During the observation update for particle filtering, we use the sensor model $P(F_i|T_i)$ to weight each particle according to the probability dictated by the observed evidence and the particle's state. Specifically, for a particle in state t_i with sensor reading f_i, assign a weight of $P(f_i|t_i)$. The algorithm for the observation update is as follows:

  1. Calculate the weights of all particles as described above.
  2. Calculate the total weight for each state.
  3. If the sum of all weights across all states is 0, reinitialize all particles.
  4. Else, normalize the distribution of total weights over states and resample your list of particles from this distribution.

  Note the similarity of the observation update to likelihood weighting, where we again downweight samples based on our evidence.

Let's see if we can understand this process slightly better by example. Define a transition model for our weather scenario using temperature as the time-dependent random variable as follows: for a particular temperature state, you can either stay in the same state or transition to a state one degree away, within the range $[10, 20]$. Out of the possible resultant states, the probability of transitioning to the one closest to 15 is 80% and the remaining resultant states uniformly split the remaining 20% probability amongst themselves.

Our temperature particle list was as follows:

$$[15, 12, 12, 10, 18, 14, 12, 11, 11, 10]$$

To perform a time elapse update for the first particle in this particle list, which is in state $T_i = 15$, we need the corresponding transition model:

T_{i+1}	14	15	16
$P(T_{i+1}\|T_i = 15)$	0.1	0.8	0.1

In practice, we allocate a different range of values for each value in the domain of T_{i+1} such that together the ranges entirely span the interval $[0, 1)$ without overlap. For the above transition model, the ranges are as follows:

1. The range for $T_{i+1} = 14$ is $0 \leq r < 0.1$.
2. The range for $T_{i+1} = 15$ is $0.1 \leq r < 0.9$.
3. The range for $T_{i+1} = 16$ is $0.9 \leq r < 1$.

In order to resample our particle in state $T_i = 15$, we simply generate a random number in the range $[0, 1)$ and see which range it falls in. Hence if our random number is $r = 0.467$, then the particle at $T_i = 15$ remains in $T_{i+1} = 15$ since $0.1 \leq r < 0.9$. Now consider the following list of 10 random numbers in the interval $[0, 1)$:

$$[0.467, 0.452, 0.583, 0.604, 0.748, 0.932, 0.609, 0.372, 0.402, 0.026]$$

If we use these 10 values as the random value for resampling our 10 particles, our new particle list after the full time elapse update should look like this:

$$[15, 13, 13, 11, 17, 15, 13, 12, 12, 10]$$

Verify this for yourself! The updated particle list gives rise to the corresponding updated belief distribution $B(T_{i+1})$:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_{i+1})$	0.1	0.1	0.2	0.3	0	0.2	0	0.1	0	0	0

Comparing our updated belief distribution $B(T_{i+1})$ to our initial belief distribution $B(T_i)$, we can see that as a general trend the particles tend to converge towards a temperature of $T = 15$.

Next, let's perform the observation update, assuming that our sensor model $P(F_i|T_i)$ states that the probability of a correct forecast $f_i = t_i$ is 80%, with a uniform 2% chance of the forecast predicting any of the other 10 states. Assuming a forecast of $F_{i+1} = 13$, the weights of our 10 particles are as follows:

Particle	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}
State	15	13	13	11	17	15	13	12	12	10
Weight	0.02	0.8	0.8	0.02	0.02	0.02	0.8	0.02	0.02	0.02

Then we aggregate weights by state:

State	10	11	12	13	15	17
Weight	0.02	0.02	0.04	2.4	0.04	0.02

Summing the values of all weights yields a sum of 2.54, and we can normalize our table of weights to generate a probability distribution by dividing each entry by this sum:

State	10	11	12	13	15	17
Weight	0.02	0.02	0.04	2.4	0.04	0.02
Normalized Weight	0.0079	0.0079	0.0157	0.9449	0.0157	0.0079

The final step is to resample from this probability distribution, using the same technique we used to resample during the time elapse update. Let's say we generate 10 random numbers in the range $[0,1)$ with the following values:

$$[0.315, 0.829, 0.304, 0.368, 0.459, 0.891, 0.282, 0.980, 0.898, 0.341]$$

This yields a resampled particle list as follows:

$$[13, 13, 13, 13, 13, 13, 13, 15, 13, 13]$$

With the corresponding final new belief distribution:

T_i	10	11	12	13	14	15	16	17	18	19	20
$B(T_{i+1})$	0	0	0	0.9	0	0.1	0	0	0	0	0

Observe that our sensor model encodes that our weather prediction is very accurate with probability 80%, and that our new particles list is consisistent with this since most particles are resampled to be $T_{i+1} = 13$.

Summary

In this note, we covered two new types of models:

- *Markov models*, which encode time-dependent random variables that possess the Markov property. We can compute a belief distribution at any timestep of our choice for a Markov model using probabilistic inference with the mini-forward algorithm.
- *Hidden Markov Models*, which are Markov models with the additional property that new evidence which can affect our belief distribution can be observed at each timestep. To compute the belief distribution at any given timestep with Hidden Markov Models, we use the forward algorithm.

Sometimes, running exact inference on these models can be too computationally expensive, in which case we can use particle filtering as a method of approximate inference.