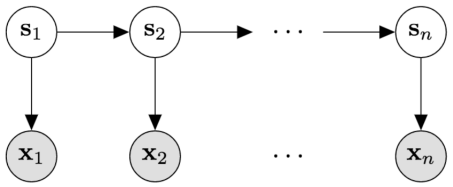
# CS 181 Spring 2022 Section 9

HMMs, Kalman Filters, and MDP's

## 1 Hidden Markov Models

A Hidden Markov Model (HMM) is useful for inferring a sequence of unknown or hidden states from a corresponding sequence of observed evidence.

### 1.1 Graphical Model




Consider a set of possible states $\mathbf{s}_i \in \{S_k\}_{k=1}^K$, which are one-hot encoded, and a set of possible observations $\mathbf{x}_i \in \{O_j\}_{j=1}^M$. That is, there are $K$ possible options for states and $M$ possible options for the observations. A *data point* would be a sequence of observations $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_n$. As shown in the figure above, we believe that each individual $\mathbf{x}_i$ is generated from some hidden state $\mathbf{s}_i$, which also gives rise to the next hidden state $\mathbf{s}_{i+1}$. We also denote by $N$ to be the number of data points we have.

### 1.2 Model Assumptions

HMMs are characterized by and allow us to reason about the following joint distribution

$$p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n, \mathbf{x}_1, \ldots, \mathbf{x}_n) = p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) p(\mathbf{x}_1, \ldots, \mathbf{x}_n \mid \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n)$$

However, it's not immediately obvious how we should optimize this model, and the following assumptions make this easier:

- The next hidden state depends only on the current state. This is also known as the *Markov property*. In other words, we assume that

  $$p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_1, \ldots \boldsymbol{s}_t, \mathbf{x}_1, \ldots, \mathbf{x}_t) = p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t),$$

  for $t = 1, 2, \ldots n-1$.

- The current observation depends only on the current hidden state. That is, we have

  $$p(\mathbf{x}_t \mid \boldsymbol{s}_1, \ldots, \boldsymbol{s}_t, \mathbf{x}_1, \ldots, \mathbf{x}_{t-1}) = p(\mathbf{x}_t \mid \boldsymbol{s}_t),$$

  for $t = 1, 2, \ldots n$.

The graphical model above is a useful way to more easily remember these assumptions.

The above assumptions are quite powerful and allow us to simplify the joint probability we desire to model as follows:

$$\begin{aligned} p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n, \mathbf{x}_1, \ldots, \mathbf{x}_n) &= p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) p(\mathbf{x}_1, \ldots, \mathbf{x}_n \mid \boldsymbol{s}_1, \ldots, \boldsymbol{s}_n) \\ &= p(\boldsymbol{s}_1) \prod_{t=1}^{n-1} p(\boldsymbol{s}_{t+1} \mid \boldsymbol{s}_t) \prod_{t=1}^{n} p(\mathbf{x}_t \mid \boldsymbol{s}_t) \end{aligned}$$

### 1.3 Exercise: When to Use HMMs (Source: CMU)

For each of the following scenarios, is it appropriate to use a Hidden Markov Model? Why or why not? What would the observed data be in each case, and what would the hidden states capture?

1. Stock market price data
2. Recommendations on a database of movie reviews
3. Daily precipitation data in Boston
4. Optical character recognition for identifying words

### 1.4 Parameterization

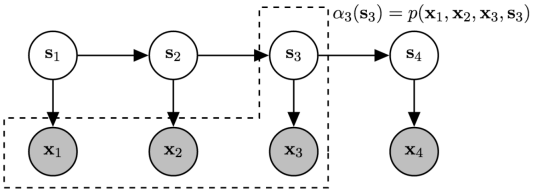
- $\boldsymbol{\theta} \in \mathbb{R}^K$: defines the prior distribution over initial hidden states
- $\mathbf{T} \in \mathbb{R}^{K \times K}$: transition matrix where $T_{ij}$ is the probability of transitioning from $S_i$ to $S_j$
- $\boldsymbol{\pi} \in \mathbb{R}^{K \times M}$: conditional probabilities of observations given hidden states such that $p(\mathbf{x}_t = O_j | \mathbf{s}_t = S_k; \{\boldsymbol{\pi}\}) = \pi_{kj}$. In other words, $\pi_{kj}$ is the probability that $\mathbf{x}_t$ is in class $j$ if $\mathbf{s}_t$ is in class $k$.

First, we need to estimate the parameters from the data, which we can do with a variant of EM. Then, with our trained HMM, we are able to perform several inference tasks on our data. As an aside, we sometimes denote by $\pi_k$ the column vector corresponding to the $k^{\text{th}}$ row of $\pi$. Intuitively, this corresponds to the "transitions" to the possible $\mathbf{x}$'s coming from state class $S_k$.

### 1.5 Forward-Backward Algorithm

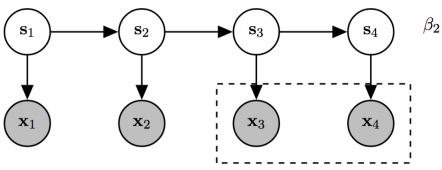
The HMM model is characterized by the joint distribution $p(\boldsymbol{s}_1, \ldots, \boldsymbol{s}_n, \mathbf{x}_1, \ldots, \mathbf{x}_n)$, which means that many of our training and inference tasks require marginalization to obtain conditionals. Thus, naive algorithms can be expensive (they require lots of nested summations over states), and we use EM instead. We define the recurrence relations $\alpha_t(\boldsymbol{s}_t)$ and $\beta_t(\boldsymbol{s}_t)$ in the E-Step:

- $\alpha_t(\mathbf{s}_t)$ represents the joint probability of observations $\mathbf{x}_1, \ldots, \mathbf{x}_t$ and state $\mathbf{s}_t$: $\alpha_t(\mathbf{s}_t) = p(\mathbf{x}_1, \ldots \mathbf{x}_t, \mathbf{s}_t)$. Intuitively, this allows us to quickly answer the question: "how likely are we to currently be in state $\mathbf{s}_t$, if we observed a specific list of values?" It turns out that $\alpha_t$ can be defined in terms of $\alpha_{t-1}$. That is, we move **forwards** through the sequence to calculate the $\alpha$'s.
- $\beta_t(\mathbf{s}_t)$ represents the joint probability of observations $\mathbf{x}_{t+1}, \ldots, \mathbf{x}_n$ conditioned on state $\mathbf{s}_t$: $\beta_t(\mathbf{s}_t) = p(\mathbf{x}_{t+1}, \ldots \mathbf{x}_n | \mathbf{s}_t)$. Similarly, we can think about this intuitively by asking "what are the chances of the next observations if we are currently in state $\mathbf{s}_t$?" It turns out that $\beta_t$ can be defined in terms of $\beta_{t+1}$. That is, we move **backwards** through the sequence to calculate the $\beta$'s.




$$\forall \mathbf{s}_3 : \quad \alpha_3(\mathbf{s}_3) = p(\mathbf{x}_3 \mid \mathbf{s}_3) \sum_{\mathbf{s}_2} p(\mathbf{s}_3 \mid \mathbf{s}_2) \alpha_2(\mathbf{s}_2)$$

(a) alpha




$$\forall \mathbf{s}_2 : \quad \beta_2(\mathbf{s}_2) = \sum_{s_3} p(\mathbf{s}_3 \mid \mathbf{s}_2) p(\mathbf{x}_3 \mid \mathbf{s}_3) \beta_3(\mathbf{s}_3)$$

(b) beta

Note that the probabilities we use for calculating $\alpha$ and $\beta$ are given by the parameters that we fix in the E-Step.

$$\forall \mathbf{s}_t: \quad \alpha_t(\mathbf{s}_t) = \begin{cases} p(\mathbf{x}_t \mid \mathbf{s}_t) \sum_{\mathbf{s}_{t-1}} p(\mathbf{s}_t \mid \mathbf{s}_{t-1}) \alpha_{t-1}(\mathbf{s}_{t-1}) & \text{if } 1 < t \leq n \\ p(\mathbf{x}_1 \mid \mathbf{s}_1) p(\mathbf{s}_1) & \text{o.w.} \end{cases}$$

$$\forall \mathbf{s}_t: \quad \beta_t(\mathbf{s}_t) = \begin{cases} \sum_{\mathbf{s}_{t+1}} p(\mathbf{s}_{t+1} \mid \mathbf{s}_t) p(\mathbf{x}_{t+1} \mid \mathbf{s}_{t+1}) \beta_{t+1}(\mathbf{s}_{t+1}) & \text{if } 1 \leq t < n \\ 1 & \text{o.w.} \end{cases}$$

## 1.6 EM for HMMs

Given data points $\{\mathbf{x}^i\}_{i=1}^N$ defined by sequences $(x_1^i, \ldots, x_n^i)$ of length $n$ represented as row vectors, we want to infer the parameters $\{\mathbf{T}, \boldsymbol{\theta}, \boldsymbol{\pi}\}$. Had we been given the true states, we could easily compute joint probability $p(\mathbf{x}^i, \mathbf{s}^i)$ and write the complete-data log likelihood, and maximize with respect to the parameters. Instead, we need to estimate state distributions and parameters iteratively.

### 1.6.1 Inference Patterns with $\boldsymbol{\alpha}, \boldsymbol{\beta}$

The following patterns are useful for inference with a trained HMM as well as during the E-Step:

- $\alpha_t(\mathbf{s}_t)\beta_t(\mathbf{s}_t) = p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{s}_t) \propto p(\mathbf{s}_t | \mathbf{x}_1, \ldots, \mathbf{x}_n)$
- joint of observations: $p(\mathbf{x}_1, \ldots, \mathbf{x}_n) = \sum_{\mathbf{s}_t} \alpha_t(\mathbf{s}_t)\beta_t(\mathbf{s}_t)$ (for any $t$)
- smoothing: $p(\mathbf{s}_t \mid \mathbf{x}_1, \ldots, \mathbf{x}_n) \propto p(\mathbf{x}_1, \ldots, \mathbf{x}_n, \mathbf{s}_t) = \alpha_t(\mathbf{s}_t)\beta_t(\mathbf{s}_t)$
- prediction: $p(\mathbf{x}_{n+1} \mid \mathbf{x}_1, \ldots, \mathbf{x}_n) \propto \sum_{\mathbf{s}_n, \mathbf{s}_{n+1}} \alpha_n(\mathbf{s}_n) p(\mathbf{s}_{n+1} \mid \mathbf{s}_n) p(\mathbf{x}_{n+1} \mid \mathbf{s}_{n+1})$
- transition: $p(\mathbf{s}_t, \mathbf{s}_{t+1} \mid \mathbf{x}_1, \ldots, \mathbf{x}_n) \propto \alpha_t(\mathbf{s}_t) p(\mathbf{s}_{t+1} \mid \mathbf{s}_t) p(\mathbf{x}_{t+1} \mid \mathbf{s}_{t+1}) \beta_{t+1}(\mathbf{s}_{t+1})$

The derivations of the above equations can be found in the textbook. As we can see above, the $\alpha$'s and $\beta$'s allow us to concisely capture the quantities we would like to model, which allows us to quickly compute important quantities.

### 1.6.2 E-Step

The goal of the expectation step is to compute the expected values of the hidden states given a fixed set of parameters $\mathbf{w} = \{\mathbf{T}, \boldsymbol{\theta}, \boldsymbol{\pi}\}$. That is, we estimate the state distribution for $p(\mathbf{s}_1^i, \ldots, \mathbf{s}_n^i | \mathbf{x}^i)$, which we will call $\mathbf{q}^i$. Note that this is a matrix with $n$ rows and $K$ columns, where the rows correspond to each $\mathbf{s}_j$, and the columns correspond to the possible classes of the $\mathbf{s}_j$.

The idea is to find successive approximations of this quantity based on the data we have available. We let $s_{t,k}^i$ be the indicator that $\mathbf{s}_t = S_k$. Then we define the $t, k$ element of $\mathbf{q}$ to be

$$q_{t,k}^i = E[s_{t,k}^i | \mathbf{x}^i] = P(\mathbf{s}_t^i = S_k | \mathbf{x}^i).$$

That is, this is the probability that the state at time $t$ is in class $k$ given all the observed emissions. Notice how this is exactly the smoothing quantity we had in the previous subsection, which is the motivation for defining $\alpha_t$ and $\beta_t$.

We would also need to consider the expectation of the *joint* of two consecutive states. Mathematically, this is written as $\mathbf{Q}^i_{t,t+1} = E[\mathbf{s}^i_t, \mathbf{s}^i_{t+1}|\mathbf{x}^i]$. Note that to encapsulate all possible values of the states, this would mean that $\mathbf{Q}^i_{t,t+1}$ is a matrix. We then define the $k, l$ element to be

$$Q^i_{t,t+1,k,l} = E[\mathbf{s}^i_t = S_k, \mathbf{s}^i_{t+1} = S_l|\mathbf{x}^i] = P(\mathbf{s}^i_t = S_k, \mathbf{s}^i_{t+1} = S_l|\mathbf{x}^i).$$

Notice how this is exactly the transition equation in the previous subsection!

### 1.6.3 M-Step

Now we need to update our parameters to maximize the expected complete-data log likelihood $\mathbb{E}_{\mathbf{S}}[\ln p(\mathbf{x}, \mathbf{s}; \mathbf{w})]$. It becomes a very nasty expression, so we will not include it here. It can be found in the textbook for reference. Applying the appropriate Lagrange multipliers and maximizing with respect to each of the parameters of interest, we recover the following update equations:

$$\theta_k = \frac{\sum_{i=1}^N q^i_{1,k}}{N},$$

which makes intuitive sense as the sample averages of our estimated probabilities for each possible value of the initial state. Next,

$$T_{k,l} = \frac{\sum_{i=1}^N \sum_{t=1}^{n-1} Q^i_{t,t+1,k,l}}{\sum_{i=1}^N \sum_{t=1}^{n-1} q^i_{tk}},$$

which has the intuitive interpretation of the (normalized) average of the transition probabilities, and finally

$$\pi_{k,m} = \frac{\sum_{i=1}^N \sum_{t=1}^{n} q^i_{t,k} x^i_{t,m}}{\sum_{i=1}^N \sum_{t=1}^{n} q^i_{tk}},$$

which has the intuitive interpretation of a weighted average of the emissions given the state. After updating these parameters, we repeat the EM algorithm until convergence of said parameters.

### 1.7 Exercise: Parameter Estimation in Supervised HMMs

You are trying to predict the weather using an HMM. The hidden states are the weather of the day, which may be sunny or rainy, and the observable states are the color of the clouds, which can be white or gray. You have data on the weather and clouds from one sequence of four days (note: the hidden states are observed here):

| Day | Weather | Clouds |
|---|---|---|
| 1 | Sunny | White |
| 2 | Rainy | Gray |
| 3 | Rainy | Gray |
| 4 | Sunny | Gray |

1. Draw a graphical model representing the HMM.
2. Give the values of $N, n, c$ and of the one-hot vectors $\mathbf{s}_1^1, \ldots, \mathbf{s}_4^1, \mathbf{x}_1^1, \ldots, \mathbf{x}_4^1$.
3. Estimate and interpret the values of the parameters $\boldsymbol{\theta}, \mathbf{T}, \{\boldsymbol{\pi}\}$ using the MLE estimators for the supervised HMM provided in the previous subsection.

## 1.8 Exercise: EM for HMMs

You are trying to model a toy's state using an HMM. At each time step, the toy can be active (state 1) or inactive (state 2), but you can only observe the color of the indicator light, which can be red (observation state 1) or green (observation state 2). You have collected data from one sequence:

| Time | Light |
|---|---|
| 1 | Green |
| 2 | Red |
| 3 | Green |

You initialize your EM with $\boldsymbol{\theta} = [\frac{1}{2}\ \frac{1}{2}]^\top, \mathbf{T} = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \\ \frac{1}{3} & \frac{2}{3} \end{bmatrix}, \boldsymbol{\pi}_1 = [\frac{1}{4}\ \frac{3}{4}]^\top, \boldsymbol{\pi}_2 = [\frac{3}{4}\ \frac{1}{4}]^\top$.

1. Compute $\alpha_1, \alpha_2, \alpha_3, \beta_1, \beta_2, \beta_3$ for the forward-backward algorithm using the initial parameter values.
2. Refer to the definition of $\mathbf{q}_t^1$ in Section 1.6.2. Now, compute the values of $\mathbf{q}_1^1, \mathbf{q}_2^1$ using the $\alpha$ and $\beta$ values.
3. Refer to the definition of $\mathbf{Q}_{t,t+1}^1$ in Section 1.6.2. Compute the value of $\mathbf{Q}_{1,2}^1$ using the $\alpha$ and $\beta$ values.

During EM, at one point you obtain the following values after the E step:

$$\mathbf{q}_1^1 = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \end{bmatrix}^\top, \quad \mathbf{q}_2^1 = \begin{bmatrix} \frac{1}{3} & \frac{2}{3} \end{bmatrix}^\top, \quad \mathbf{q}_3^1 = \begin{bmatrix} \frac{2}{3} & \frac{1}{3} \end{bmatrix}^\top$$

$$\mathbf{Q}_{1,2}^1 = \begin{bmatrix} \frac{1}{6} & \frac{1}{2} \\ \frac{1}{6} & \frac{1}{6} \end{bmatrix}, \quad \mathbf{Q}_{2,3}^1 = \begin{bmatrix} \frac{1}{6} & \frac{1}{6} \\ \frac{1}{2} & \frac{1}{6} \end{bmatrix}$$

1. Use the above values to compute $\hat{N}_k, \hat{N}_{kl}, \hat{N}_{kj}$.
2. Complete the M step by updating the parameters $\boldsymbol{\theta}, \mathbf{T}, \boldsymbol{\pi}_1, \boldsymbol{\pi}_2$.

## 2 Markov Decision Processes

A Markov Decision Process (MDP) is a framework for modeling an agent's actions in the world. It consists of:

1. A set of states $S$
2. A set of actions $A$
3. A reward function $r : S \times A \to \mathbb{R}$
4. A transition model $p(s'|s, a)$, $\forall s, s' \in S, a \in A$.

A *policy* $\pi$ is a mapping from states to actions, i.e. $\pi : S \to A$.

### 2.1 Finite time horizon MDP

In the finite horizon setting, a policy may vary with the number of time periods remaining. $\pi_{(t)}$ denotes the policy with $t$ time steps to go. $T$ is the decision horizon. The value of a policy with $t$ time steps to go is defined inductively to be:

$$V^{\pi}_{(t)}(s) = \begin{cases} r(s, \pi_{(1)}(s)) & \text{if } t = 1 \\ r(s, \pi_{(t)}(s)) + \sum_{s' \in S} p(s'|s, \pi_{(t)}(s)) V^{\pi}_{(t-1)}(s') & \text{o.w.} \end{cases} \tag{1}$$

The process of computing these values inductively, working from the end of the horizon to the present, is called *value iteration*. If we instead look forward in time, we are computing the expected value of the policy

$$V^{\pi}_{T}(s) = \mathbb{E}_{s_1, \ldots, s_T} \left[ \sum_{t=0}^{T} r(s_t, \pi_{(T-t)}(s_t)) \right] \tag{2}$$

by induction, where $s_1 := s$. $V^{\pi}(s)$ is the MDP value function.

In an MDP, the general goal is to find an optimal policy by maximizing the expected reward under the policy, i.e. maximizing the value function. This is the *planning problem*.

## 2.2 Infinite Horizon MDP

**Policy Evaluation** We can also send $T \to \infty$, i.e. have an infinite time horizon. In that case, we need a discount factor $0 < \gamma < 1$, and we want to compute the value function

$$V^\pi(s) = \mathbb{E}_{s_1,s_2,\ldots}\left[\sum_{t=0}^{\infty} \gamma^t r(s_t, \pi(s_t))\right] \tag{3}$$

where $s_1 := s$, and the $\gamma$ factor ensures convergence (assuming bounded rewards). In this setting, we only worry about stationary policies that don't vary with time. This is the *policy evaluation* problem; for any given policy $\pi$, we can find $V^\pi(s)$ by solving the system of linear equations

$$V^\pi(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V^\pi(s') \tag{4}$$

These capture consistency about the value function. To solve this system, we can use Gaussian elimination, or simply iterate until convergence as in the finite horizon case.

Given a policy $\pi$ and $\theta$ (small positive number), we find $V^\pi$ iteratively as follows:

- Initialize: $V(s) = 0$ for all states $s$.
- Repeat
  - Update step:

$$V'(s) = r(s, \pi(s)) + \gamma \sum_{s' \in S} p(s'|s, \pi(s)) V(s'), \ \forall s \tag{5}$$

  - $\Delta = \max(|V' - V|)$
  - $V \leftarrow V'$

  until $\Delta < \theta$

**Value Iteration** Suppose we have an optimal policy $\pi^*$. This satisfies the following set of equations known as the *Bellman equations*:

$$V^*(s) = \max_{a \in A}\left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^*(s')\right] \tag{6}$$

where $V^* \triangleq V^{\pi^*}$. Assuming we know $V^*$, we can read off the optimal policy by setting

$$\pi^*(s) = \arg\max_{a \in A}\left[r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^*(s')\right] \tag{7}$$

In order to find $V^*$, we can use *value iteration*:

- Initialize: $V(s) = 0$ for all states $s$.

- Update step (Bellman operator):

$$V'(s) = \max_{a \in A} \left[ r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V(s') \right], \ \forall s \tag{8}$$

- $V \leftarrow V'$

where we iterate until convergence of $V$, which is guaranteed. With our converged $V$, we can then find $\pi^*$ as in Equation 7.

**Policy Iteration** Another approach to planning is called *policy iteration*. To do policy iteration, we evaluate a proposed policy $\pi$ by finding $V^\pi$ as in Equation 4. This is Evaluation step (E step). Then, we do a policy improvement step (I step) by the equation

$$\pi'(s) \leftarrow \arg\max_{a \in A} \left[ r(s,a) + \gamma \sum_{s' \in S} p(s'|s,a) V^\pi(s') \right], \quad \forall s \tag{9}$$

We repeat the E and I steps until the policy $\pi$ converges (stops changing).

Note that policy iteration takes more computation per iteration, but tends to converge faster in practice.

### 2.3 Exercise: Markov Decision Process

(Sutton & Barto 2012) Consider an MDP on the following grid:

| | | |
|---|---|---|
| | A | |
| | | |
| | B | |

At each square, we can go left, right, up, or down. Normally we get a reward of 0 from moving, but if we attempt to move off the grid, we get a reward of $-1$ and stay where we are. Also, if we move onto square A, we get a reward of 10 and are teleported to square B.

Suppose our actions also fail with probability 0.5, i.e. with probability 0.5 we stay on the current square. Also suppose our MDP is infinite horizon, and take $\gamma = 0.9$ to be the discount factor.

1. **Defining the MDP** Identify the states $S$, actions $A$, rewards, and transition probabilities $p(s'|s,a)$ in this problem.
2. **Policy Evaluation** Suppose $\pi$ is the policy where we always choose to go right. Write the equations to find the values $V^\pi(s)$.
3. **Value Iteration** Write the second iteration of value iteration, i.e. starting by initializing $V(s) = \max_{a \in A} r(s,a)$.
4. **Policy Iteration** Write the first iteration of policy iteration, starting with $V^\pi(s) = 0$ for all $s$. (We could also initialize a policy, and do the Evaluation step to get started.)

## 3 Kalman Filters (Bonus Material)

Now consider the following dynamical system model:

$$z_{t+1} = \Phi z_t + \epsilon_t$$

$$x_t = A z_t + \gamma_t$$

where $z$ are the hidden variables and $x$ are the observed measurements. $\Phi$ and $A$ are known constants, while $\epsilon$ and $\gamma$ are random variables drawn from the following normal distributions:

$$\epsilon_t \sim \mathcal{N}(\mu_\epsilon, \sigma_\epsilon^2)$$

$$\gamma_t \sim \mathcal{N}(\mu_\gamma, \sigma_\gamma^2)$$

This is called a (one-dimensional) linear Gaussian state-space model. It is closely related to an HMM – try drawing out the graphical model! – but here the hidden states and the observations are now continuous and normally distributed. Linear Gaussian state-space models have convenient mathematical properties and can be used to describe noisy measurements of a moving object (e.g. missiles, rodents, hands), market fluctuations, etc.

The Kalman filter is an algorithm to perform filtering in linear Gaussian state-space models, i.e. to find the distribution of $z_t$ given observations $x_1, ..., x_t$. The distribution of $z_t \,|\, x_1, ..., x_s$ will be $\mathcal{N}(\mu_{t|s}, \sigma_{t|s}^2)$. If we start with $\mu_{t-1|t-1}$ and $\sigma_{t-1|t-1}^2$, the algorithm tells us to

1. Define the distribution of $z_t \,|\, x_1, ..., x_{t-1}$ by computing $\mu_{t|t-1}$ and $\sigma_{t|t-1}^2$. This is called the prediction step.
2. Define the distribution of $z_t \,|\, x_1, ..., x_t$ by computing $\mu_{t|t}$ and $\sigma_{t|t}^2$. This is called the update step.

The Kalman filter alternates between prediction and update steps, assimilating observations one at a time. It requires one forward pass through the data, and is analogous to obtaining the $\alpha$'s in an HMM.