



MSML610: Advanced Machine Learning

Reasoning Over Time

Instructor: Dr. GP Saggese - gsaggese@umd.edu

References: - AIMA 14: Probabilistic reasoning over time

Reasoning over time

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference

Reference

- AIMA: 14

Agents

- Agents in partially observable environments track the current state using sensor information
 1. **Belief state**
 - Store possible world states (by enumeration or logical formulas)
 - Use probability theory to quantify belief
 - Belief state is the posterior distribution of the current state given all evidence so far
 2. **Belief state** + **Transition model**
 - Predict how the world might evolve in the next step
 3. **Sensor model** + **Percepts**
 - Update belief state
- Time is handled by making each quantity a function of time

Static vs dynamic probabilistic reasoning: example

- **Static probabilistic reasoning**

- Each random variable has a single fixed value over time
- E.g., when repairing a car:
 - Whatever is broken stays broken during the diagnosis
 - Observed evidence remains fixed

- **Dynamic probabilistic reasoning**

- Random variables change over time
- E.g.,
 - Tracking the location of a plane
 - Tracking the economic activity of a nation
- E.g., treating a diabetic patient
 - Goal: assess the state of the patient and decide on insulin dose
 - Evidence: previous insulin doses, food intake, blood sugar (which change over time)
 - Dependency on time (e.g., metabolic activity and time of day)

Model components

1. State of the world: $\underline{\mathbf{X}}_t$
2. Prior probability of the state at time 0: $\underline{\mathbf{X}}_0$
3. Evidence variables: $\underline{\mathbf{E}}_t$
4. Transition model: $\Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_{0:t-1})$
 - How the world evolves
 - Specifies the probability distribution of the state $\underline{\mathbf{X}}_t$, given all previous values
5. Sensor model: $\Pr(\underline{\mathbf{E}}_t | \underline{\mathbf{X}}_{0:t}, \underline{\mathbf{E}}_{0:t-1})$
 - How the evidence variables $\underline{\mathbf{E}}_t$ are generated

Discrete vs continuous time models

- **Discrete time models**

- View the world as a series of time slices (“snapshots”)
 - Assume time intervals are equal, so samples are equispaced
 - Label times $t = 0, 1, 2, \dots$
- Each time slice contains random variables:
 - Some RVs are not observable $\underline{\mathbf{X}}_t$ (hidden)
 - Other RVs are observable $\underline{\mathbf{E}}_t$ (evidence)
 - $\underline{\mathbf{X}}_{a:b}$ represents variables in $[a, b]$

- **Continuous time models**

- Uncertainty over continuous time can be modeled by stochastic differential equations (SDEs)
- Discrete time models can be discrete approximations to SDEs

Markov property

- In general, the current state $\underline{\mathbf{X}}_t$ depends on a growing number of past states:

$$\Pr(\underline{\mathbf{X}}_t | \text{history}) = \Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_0, \underline{\mathbf{X}}_1, \dots, \underline{\mathbf{X}}_{t-1})$$

- **Markov property:** the current state (conditionally) depends only on a finite fixed number of k previous states:

$$\Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_0, \underline{\mathbf{X}}_1, \dots, \underline{\mathbf{X}}_{t-k-1}, \underline{\mathbf{X}}_{t-k}, \dots, \underline{\mathbf{X}}_{t-1}) = \Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_{t-1:t-k})$$

Markov process

- **Markov processes** (aka **Markov chains**) have the Markov property

$$\Pr(\underline{\mathbf{X}}_t | \text{history}) = \Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_{t-1:t-k})$$

- **First-order Markov process**: the current state depends only on the previous state (and no other earlier state):

$$\Pr(\underline{\mathbf{X}}_t | \text{history}) = \Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_{t-1})$$

- E.g., the probability of rain today depends only on what happened yesterday $\Pr(R_t | R_{t-1}) \forall t$
 - The Bayesian network for a first-order Markov process looks like:
- **Second-order Markov process**: the current state $\underline{\mathbf{X}}_t$ is conditionally dependent only on $\underline{\mathbf{X}}_{t-1}$ and $\underline{\mathbf{X}}_{t-2}$ and no other earlier state

Time-homogeneous process

- Even with the Markov assumption, there are infinite probability distributions $\Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_{t-1})$, one for every value of t
- **Stationarity**: the transition probability doesn't change over time

$$\Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_{0:t-1}) = \Pr(\underline{\mathbf{X}}_{t-k} | \underline{\mathbf{X}}_{0:t-k-1}) \forall k, t$$

- The process evolves over time, but the governing laws don't change
- **First-order time-homogeneous**: only one conditional probability table is needed

$$\Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_{t-1}) = \Pr(\underline{\mathbf{X}}_{t-k} | \underline{\mathbf{X}}_{t-k-1}) \forall k, t$$

- E.g., the probability of rain depends on what happened yesterday and is the same every day: $\Pr(R_t | R_{t-1}) = f(R_{t-1}) \forall t$

Sensor model

- Aka "observation model"
- In general, the evidence variables \underline{E}_t could depend on previous \underline{X} (state of the world) and \underline{E} (sensor value) variables:

$$\Pr(\underline{E}_t | \underline{X}_{0:t}, \underline{E}_{0:t-1})$$

- Sensor Markov property

- We assume that the sensor value \underline{E}_t depends only on the current state of the world \underline{X}_t , not on previous sensor values

$$\Pr(\underline{E}_t | \underline{X}_{0:t}, \underline{E}_{0:t-1}) = \Pr(\underline{E}_t | \underline{X}_t)$$

- In a Bayesian network, even if \underline{X}_t and \underline{E}_t are contemporaneous in the time step, the arrow goes from the state of the world \underline{X}_t to the sensor value \underline{E}_t , i.e., $\underline{X}_t \rightarrow \underline{E}_t$ since the world causes the sensor to take on particular values

Sensor model: rain example

- In a Bayesian network, $\underline{X}_t \rightarrow \underline{E}_t$ since the world causes the sensor to take on particular values
 - E.g., the rain “causes” the umbrella to appear
 - The inference goes in the other direction: we see the umbrella and need to guess if it's raining
- E.g.,
 - The transition model is $\Pr(Rain_t | Rain_{t-1})$
 - $\Pr(R_t | R_{t-1} = T) = 0.7$
 - $\Pr(R_t | R_{t-1} = F) = 0.3$
 - The sum doesn't have to be 1 since it's a conditional probability
 - The sensor model is $\Pr(Umbrella_t | Rain_t)$
 - $\Pr(U_t | R_t = T) = 0.9$
 - $\Pr(U_t | R_t = F) = 0.2$

Prior probability

- To complete the system specification, we need the prior probability of the state variables at time 0, $\Pr(\underline{\mathbf{X}}_0)$
 - Represents the initial belief about the state of the system before any observations are made
 - It is crucial for initializing the state estimation process
- E.g.,
 - $\underline{\mathbf{X}}_0$ represents the position and velocity of a moving object
 - $\Pr(\underline{\mathbf{X}}_0)$ could be a Gaussian distribution centered around an initial guess of the object's position and velocity with some uncertainty

First-order Markov process: Joint Distribution

- Model a sequence of states $\underline{\mathbf{X}}_0, \underline{\mathbf{X}}_1, \dots, \underline{\mathbf{X}}_t$ and observations $\underline{\mathbf{E}}_1, \dots, \underline{\mathbf{E}}_t$ over time, with the simplifying assumptions:
 - First-order Markov assumption: $\Pr(\underline{\mathbf{X}}_i | \underline{\mathbf{X}}_{0:i-1}) = \Pr(\underline{\mathbf{X}}_i | \underline{\mathbf{X}}_{i-1})$
 - Sensor model: $\Pr(\underline{\mathbf{E}}_i | \underline{\mathbf{X}}_{0:i}, \underline{\mathbf{E}}_{1:i-1}) = \Pr(\underline{\mathbf{E}}_i | \underline{\mathbf{X}}_i)$
- The joint distribution of n random variables:

$$\Pr(X_1, \dots, X_n) = \prod_{i=1}^n \Pr(X_i | \text{parents}(X_i))$$

- The joint distribution probability can be written for any t :

$$\begin{aligned} \Pr(\underline{\mathbf{X}}_{0:t}, \underline{\mathbf{E}}_{1:t}) &= \text{Pr}(\underline{\mathbf{X}}_0) \prod_{i=1}^t \Pr(\underline{\mathbf{X}}_i | \underline{\mathbf{X}}_{i-1}) \Pr(\underline{\mathbf{E}}_i | \underline{\mathbf{X}}_i) \\ &= \text{prior} \times \prod_i \text{transition model} \times \text{sensor model} \end{aligned}$$

First-order Markov process: intuition

- The joint distribution probability for a time-homogeneous first-order Markov process can be written, for any t :

$$\begin{aligned}\Pr(\underline{\mathbf{X}}_{0:t}, \underline{\mathbf{E}}_{1:t}) &= \Pr(\underline{\mathbf{X}}_0) \prod_{i=1}^t \Pr(\underline{\mathbf{X}}_i | \underline{\mathbf{X}}_{i-1}) \Pr(\underline{\mathbf{E}}_i | \underline{\mathbf{X}}_i) \\ &= \text{prior} \times \prod_i \text{transition model} \times \text{sensor model}\end{aligned}$$

- Intuition:
 - Each observation depends only on the current state (sensor model)
 - The state evolves probabilistically from the previous state (transition model)
 - This structure reduces complexity and enables tractable inference
- How to represent this process?
 - A Bayesian network can represent a temporal model by modeling time with indices t , i.e., “unrolling the model”
 - Problem: There are infinite t , even assuming the Markov property

Improving approximation of real-world systems

- A first-order Markov process can be reasonable or not, e.g.,
 - A particle following a random walk is well represented by Markov process (by definition)
 - In the umbrella example the rain depends only on what happened the previous day
- How to improve the approximation
 1. Increase the order of the Markov process model
 - E.g., to model “rarely rains more than two days in a row”, we need a second-order Markov model $\Pr(Rain_t | Rain_{t-1}, Rain_{t-2})$
 2. Increase the set of state variables
 - E.g., add $Season_t$ to incorporate the historical records
 - This makes the transition model more complicated
 3. Increase the number of sensor variables
 - E.g., $Location_t, Temperature_t, Humidity_t, Pressure_t$
 - This can simplify modeling of the state

Inference tasks in temporal models

- There are several possible applications that we will consider in details

| Task | Description | Estimate |
|-------------------------|-----------------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| Filtering | Estimate <i>current</i> state given past / current obs | $\Pr(\underline{X}_t \underline{E}_{1:t})$ |
| Prediction | Estimate <i>future</i> state given past / current obs | $\Pr(\underline{X}_{t+k} \underline{E}_{1:t})$ for $k > 0$ |
| Smoothing | Estimate <i>past</i> state given past, current, and <i>future</i> obs | $\Pr(\underline{X}_k \underline{E}_{1:T})$ for $k < T$ |
| Most Likely Explanation | Find most probable sequence of states given the evidence | $\operatorname{argmax}_{\underline{x}_{1:T}} \Pr(\underline{X}_{1:T} \underline{E}_{1:t})$ |
| Learning | Learn model parameters or structure from data | θ of a model |

Task 1: Filtering

- Aka "state estimation"
- **Filtering** computes the posterior distribution of the *current* state (belief state) given all evidence to date:

$$\Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{E}}_{1:t} = \underline{\mathbf{e}}_{1:t})$$

- E.g., estimate the probability of rain today, given all umbrella observations so far $\Pr(\text{Rain}_t | \text{Umbrella}_{1:t})$
- Filtering is needed by a rational agent to track the current state of the world
 - The agent has a belief about the current state $\Pr(\underline{\mathbf{X}}_{t-1})$ at time $t - 1$
 - New evidence $\underline{\mathbf{e}}_t$ arrives for time t
 - The agent updates its belief about the current state $\Pr(\underline{\mathbf{X}}_t)$ at time t
- The term "filtering" refers to the problem in signal processing of filtering out noise in a signal by estimating system parameters

Task 2: Prediction

- **Prediction** involves predicting the posterior distribution over a *future* state, given all evidence to date:

$$\Pr(\underline{\mathbf{X}}_{t+k} | \underline{\mathbf{e}}_{1:t}) \text{ with } k > 0$$

- E.g., compute the probability of rain three days from now:

$$\Pr(Rain_{t+3} | Umbrella_{0:t})$$

- Prediction helps a rational agent evaluate possible actions based on expected outcomes

Task 3: Smoothing

- **Smoothing** is the task of computing the posterior distribution over a *past* state given *all* the past, present, and future evidence:

$$\Pr(\underline{\mathbf{X}}_k | \underline{\mathbf{e}}_{1:t}) \text{ with } 0 \leq k < t$$

- Note: you have information about the “future” of the evidence, but you don’t know the state
- Smoothing provides a better estimate of the state since it incorporates more evidence about the future
- E.g., compute the probability that it rained last Wednesday, given all the observations made up to today
- The name “smoothing” refers to the fact that the state estimate is smoother than filtering

Task 4: Most-likely explanation

- **Most-likely explanation** finds the sequence of states $\underline{x}_{1:t}$ most likely to have generated observations $\underline{e}_{1:t}$:

$$\operatorname{argmax}_{\underline{x}_{1:t}} \Pr(\underline{x}_{1:t} | \underline{e}_{1:t})$$

- E.g.,
 - Umbrella appeared on 3 days, not on the fourth
 - Most likely explanation: rained for 3 days, then stopped
- Applications
 - Speech recognition: most likely sequence of words given sounds
 - Digital processing: reconstruct bit strings over a noisy channel

Task 5: Learning

- Learning involves estimating the transition model $\Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{X}}_{0:t-1})$ and the sensor model $\Pr(\underline{\mathbf{E}}_i | \underline{\mathbf{X}}_i)$ from observations
- **Learning:**
 - Can be a byproduct of inference
 - Requires smoothing rather than filtering for better state estimates
 - Smoothing uses all available data to estimate states, leading to more accurate models
 - E.g., in a weather prediction system, smoothing might use past, present, and future data to better estimate the current weather state

Task 1: Recursive filtering: goal

- Aka “recursive state estimation”
- A practical filtering algorithm updates the current state estimate $\underline{\mathbf{X}}_{t+1}$ using the previous state $\underline{\mathbf{X}}_t$ and the new evidence $\underline{\mathbf{e}}_{t+1}$, rather than recomputing it by going over the entire history of the percepts

$$\Pr(\underline{\mathbf{X}}_{t+1} | \underline{\mathbf{e}}_{1:t+1}) = f(\Pr(\underline{\mathbf{X}}_t | \underline{\mathbf{e}}_{1:t}), \underline{\mathbf{e}}_{t+1})$$

$$\text{NextState} = f(\text{PreviousState}, e_{t+1})$$

- **Why?**
 - Time and space requirements for updating must be constant if a (finite) agent needs to keep track of current state indefinitely
- **Is it possible?**
 - What is the formula $f(\dots)$?

Task 1: Recursive filtering: update formula

- Compute the state at time $t + 1$ with all the evidence up to that time
- Assume that state and evidence are scalar and not vector: $\Pr(X_{t+1}|e_{1:t+1})$

| | |
|----------------------------|---------------------------------------------------------------------------------------------|
| Divide up the evidence | $= \Pr(X_{t+1} e_{1:t}, e_{t+1})$ |
| Bayes rule given $e_{1:t}$ | $= \alpha \Pr(e_{t+1} X_{t+1}, e_{1:t}) \Pr(X_{t+1} e_{1:t})$ |
| Markov sensor assumption | $= \alpha \Pr(e_{t+1} X_{t+1}) \Pr(X_{t+1} e_{1:t})$ |
| Condition on current state | $= \alpha \Pr(e_{t+1} X_{t+1}) \sum_{x_t} \Pr(X_{t+1} x_t, e_{1:t}) \Pr(x_t e_{1:t})$ |
| Markov assumption | $= \alpha \Pr(e_{t+1} X_{t+1}) \sum_{x_t} \Pr(X_{t+1} x_t) \Pr(x_t e_{1:t})$ |

$$\Pr(X_{t+1} | e_{1:t+1}) = f(\Pr(X_t | e_{1:t}), e_{t+1})$$

- The next state is “Sensor model x Transition model x Recursive state”
 - Sensor model: $\Pr(e_{t+1} | X_{t+1})$
 - Transition model: $\Pr(X_{t+1} | x_t)$
 - Recursive term: $\Pr(x_t | e_{1:t})$

Task 1: Recursive filtering: intuition

- Recursive state estimation updates in **two steps** the belief about a system's state over time as new evidence arrives

$$\Pr(X_{t+1}|e_{1:t+1}) = \alpha \Pr(e_{t+1}|X_{t+1}) \sum_{x_t} \Pr(X_{t+1}|x_t) \Pr(x_t|e_{1:t})$$

- Prediction step:** Use the transition model to predict the next state based on the current belief

$$\Pr(X_{t+1}|e_{1:t}) = \sum_{x_t} \Pr(X_{t+1}|x_t) \Pr(x_t|e_{1:t})$$

- Intuition: Project the current belief forward using the model of system evolution
- Update step:** Incorporate the new observation to refine the prediction

$$\Pr(X_{t+1}|e_{1:t+1}) = \alpha \Pr(e_{t+1}|X_{t+1}) \Pr(X_{t+1}|e_{1:t})$$

- Intuition: Correct the prediction using the likelihood of the new evidence
- Maintain $\Pr(X_t|e_{1:t})$, the probability of the current state given all past evidence
 - E.g., in a weather model, if it was likely to rain today and rain usually continues, the prediction leans toward rain tomorrow
 - Seeing an umbrella supports this and updates the belief accordingly

Task 1: Forward update

- We achieved:

$$\begin{aligned}\Pr(\underline{\mathbf{X}}_{t+1}|\underline{\mathbf{e}}_{1:t+1}) &= \alpha \Pr(\underline{\mathbf{e}}_{t+1}|\underline{\mathbf{X}}_{t+1}) \sum_{\mathbf{x}_t} \Pr(\underline{\mathbf{X}}_{t+1}|\mathbf{x}_t) \Pr(\mathbf{x}_t|\underline{\mathbf{e}}_{1:t}) \\ &= f(\Pr(\underline{\mathbf{X}}_t|\underline{\mathbf{e}}_{1:t}), \underline{\mathbf{e}}_{t+1})\end{aligned}$$

- The filtered estimate $\underline{\mathbf{f}}_{1:t} = \Pr(\underline{\mathbf{X}}_t|\underline{\mathbf{e}}_{1:t})$ is propagated forward and updated by each transition and new observation

$$\underline{\mathbf{f}}_{1:t+1} = \text{Forward}(\underline{\mathbf{f}}_{1:t}, \underline{\mathbf{e}}_{t+1})$$

starting with the initial condition $\underline{\mathbf{f}}_{1:0} = \Pr(\underline{\mathbf{X}}_0)$

- This is called “forward update”
- This process allows efficient online inference without storing the full history
 - Time and space requirements for updating is constant
 - A (finite) agent can keep track of current state indefinitely

Task 2: Prediction: update formula

- Prediction is equivalent to filtering without updating the state with new evidence (since we lack evidence)
 - Only the transition model is needed, not the sensor model
- The rule predicting state $\underline{\mathbf{X}}_{t+k+1}$ given state $\underline{\mathbf{X}}_{t+k}$ and evidence $\underline{\mathbf{E}}_{1:t}$ is:

$$\Pr(\underline{\mathbf{X}}_{t+k+1}|\underline{\mathbf{e}}_{1:t}) = \sum_{\underline{\mathbf{x}}_{t+k}} \Pr(\underline{\mathbf{X}}_{t+k+1}|\underline{\mathbf{x}}_{t+k}) \Pr(\underline{\mathbf{x}}_{t+k}|\underline{\mathbf{e}}_{1:t})$$

- This equation can be used recursively to advance over time
 - Predicting even a few steps ahead generally incurs large uncertainty

Task 3: Smoothing: intuition

- We want to calculate the probability distribution over the hidden state at time k , given all evidence up to time t (in the future!)

$$\Pr(X_k | e_{1:t}) \text{ where } 0 \leq k < t$$

- Filtering gives $\Pr(X_k | e_{1:k})$ using past and present evidence
- Smoothing refines the estimate of past states using later evidence
- **E.g.,**
 - You're tracking whether it was raining yesterday
 - You had some evidence up to yesterday (e.g., a cloudy sky)
 - Today you see puddles on the ground
 - That new observation supports the idea that it was raining

Task 3: Smoothing: update formula

- Using the same math as for filtering and the two key assumptions of Markov process and Markov sensor

- **Forward Pass (aka filtering):**

- Move forward through time, using the filtering algorithm to compute:

$$f_{1:k} = \Pr(X_k | e_{1:k})$$

- This gives you a “best guess” of the state at time k , based only on evidence up to k

- **Backward Pass (aka smoothing):**

- Move backward through time from time t , computing:

$$b_{k+1:t} = P(e_{k+1:t} | X_k)$$

- This captures how likely the future evidence is, given a particular value of X_k

- **Combine them:**

- Multiply forward and backward messages to get:

$$P(X_k | e_{1:t}) \propto f_{1:k} \times b_{k+1:t}$$

Task 4: Most likely explanation: Intuition 1/2

- You are tracking the weather (sunny or rainy) based on whether someone carries an umbrella
- You can't see *Weather* directly (hidden state), but you observe umbrellas (which is a noisy observation)
 - You have 5 observations $Umbrella = [T, T, F, T, T]$
- Question: what is the most likely sequence of *Weather* states that explains the *Umbrella* observations?
 - You know something about the transition model (i.e., "it tends to rain several days in a row") and the sensor model (i.e., "people often forget the umbrella")
- Mathematically

$$\operatorname{argmax}_{x_{1:t}} \Pr(x_{1:t} | e_{1:t}) = \operatorname{argmax}_{Weather_{1:t}} \Pr(Weather_{1:t} | Umbrella_{1:t})$$

Task 4: Most likely explanation: Intuition 2/2

- **Naive approach:** Use smoothing to choose the most likely state at each time step
 - Cons
 - Might lead to an implausible overall path
 - Suboptimal since the question addresses joint probability and we are not using all the information (only one step at the time!)
- **Viterbi algorithm:**
 - Constructs a path through a state-time graph with states as nodes and transitions as edges
 - Finds the most likely entire path through the hidden states
- **Key difference:**
 - E.g., in speech recognition, find the most likely word sequence behind a noisy audio signal
 - Smoothing: Best guess per time step (may not find words that are not English and / or suboptimal sequence)
 - Viterbi: Best overall path (maximizes joint probability of the entire sequence)

Viterbi algorithm: Intuition

- Objective: Find the most likely **sequence** of hidden states given observations
1. Initialization
 - At $t = 1$, estimate the probability of starting in each state using the initial state distribution and observation likelihood
 2. Recursion via dynamic programming
 - At each time $t > 1$, for each state x_t :
 - Compute the maximum probability path to x_t from any previous state
 - Use:
 - $\Pr(x_t|x_{t-1})$: transition model
 - $\Pr(e_t|x_t)$: sensor model
 - Best path probability to x_{t-1} from prior step
 - Store the probability and the corresponding back-pointer to x_{t-1}
 3. Termination and backtrace
 - At final time $t = T$, identify the state with the highest final probability
 - Trace back through the stored pointers to reconstruct the optimal path

Viterbi algorithm: Example 1/2

- You observe a friend carrying an umbrella over 3 days:
 $Umbrella = [Yes, Yes, No]$
- You want to infer the most likely sequence of hidden *Weather* states
 - States: $S = \{Sunny, Rainy\}$ (weather)
 - Observations: $O = \{Yes, No\}$ (umbrella)

- Initial Probabilities:

$$\Pr(Sunny) = 0.6, \quad \Pr(Rainy) = 0.4$$

- Transition Probabilities:

$$\Pr(Sunny \rightarrow Sunny) = 0.7, \quad \Pr(Sunny \rightarrow Rainy) = 0.3$$

$$\Pr(Rainy \rightarrow Sunny) = 0.4, \quad \Pr(Rainy \rightarrow Rainy) = 0.6$$

- Observation (Emission) Probabilities:

$$\Pr(Yes|Sunny) = 0.1, \quad \Pr(No|Sunny) = 0.9$$

$$\Pr(Yes|Rainy) = 0.8, \quad \Pr(No|Rainy) = 0.2$$

Viterbi algorithm: example 2/2

- Viterbi table

| Day | State | Probability | Backpointer |
|-----|-------|---------------------------------------------------------------------------|-------------|
| 1 | Sunny | $0.6 \times 0.1 = \mathbf{0.06}$ | — |
| | Rainy | $0.4 \times 0.8 = \mathbf{0.32}$ | — |
| 2 | Sunny | $\max(0.06 \times 0.7, 0.32 \times 0.4) \times 0.1 = \mathbf{0.0128}$ | Rainy |
| | Rainy | $\max(0.06 \times 0.3, 0.32 \times 0.6) \times 0.8 = \mathbf{0.1536}$ | Rainy |
| 3 | Sunny | $\max(0.0128 \times 0.7, 0.1536 \times 0.4) \times 0.9 = \mathbf{0.0553}$ | Rainy |
| | Rainy | $\max(0.0128 \times 0.3, 0.1536 \times 0.6) \times 0.2 = \mathbf{0.0184}$ | Rainy |

- Final most probable state: **Sunny**
(Day 3)
- Find the most likely sequence:
Rainy → *Rainy* → *Sunny*

HMMs

- Reasoning over time
- **HMMs**
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference

Algorithms for Specific Models

- General temporal probabilistic reasoning makes minimal assumptions:
 - Markov property for transitions
 - Sensor model depends only on current state
 - No constraints on:
 - Mathematical form of transition/sensor models
 - Nature of state and evidence variables (discrete or continuous)
- Efficiency and accuracy can improve by exploiting specific model structures:
 - **Hidden Markov Models (HMMs):**
 - State is a single discrete variable
 - Transition and observation models are discrete probability tables
 - Enables fast algorithms like the Viterbi algorithm, forward-backward, etc
 - **Kalman Filters** (for continuous domains):
 - State variables are continuous and normally distributed
 - Linear-Gaussian models for transitions and observations
 - Allows exact, efficient updates using matrix operations
- Tailored algorithms can be orders of magnitude faster and more accurate than general methods

Hidden Markov Model: State and Transition Model

- **Hidden Markov Model (HMM)**: A temporal model with simplified structure for efficiency
 - **State model**:
 - The system state at time t is a discrete random variable $X_t \in \{1, \dots, S\}$
 - E.g., in the umbrella domain, $X_t = \text{Rain}_t$ with states $\{\text{rain}, \text{no rain}\}$
 - Generality: Multiple variables can be combined into one “mega-state” variable
 - **Transition model** $\Pr(X_t|X_{t-1})$:
 - Described by a transition matrix \underline{T} of size $S \times S$
 - Entry $T_{ij} = \Pr(X_t = j|X_{t-1} = i)$: probability of transitioning from state i to j
 - **Sensor model**:
 - Defined as $\Pr(E_t|X_t = i)$ for each state i
 - Representable as a vector (discrete observations) or a diagonal matrix \underline{O} (for convenience)
 - No assumptions about the number or type (discrete / continuous) of observation variables
- **Benefit**
 - This structure enables efficient algorithms like forward, backward, and Viterbi

Hidden Markov Model: umbrella example

- E.g., if $Rain = T$ is state 1 and $Rain = F$ is state 2, then the transition matrix for the umbrella world

| R_{t-1} | $\Pr(R_t R_{t-1})$ |
|-----------|--------------------|
| T | 0.7 |
| F | 0.3 |

becomes the transition model

$$\underline{\underline{T}} = \begin{pmatrix} 0.7 & 0.3 \\ 0.3 & 0.7 \end{pmatrix}$$

- On day 1 we observe $U_1 = T$ and on day 3, $U_3 = F$, we have the observation matrices

$$\underline{\underline{O}}_1 = \begin{pmatrix} 0.9 & 0 \\ 0 & 0.2 \end{pmatrix} \quad \underline{\underline{O}}_3 = \begin{pmatrix} 0.1 & 0 \\ 0 & 0.8 \end{pmatrix}$$

Hidden Markov Model: algorithms

- Using the matrix representation all the forward / backward computations become matrix operations:

$$\mathbf{f}_{1:t+1} = \alpha \mathbf{O}_{t+1} \mathbf{T}^\top \mathbf{f}_{1:t}$$

$$\mathbf{b}_{k+1:t} = \mathbf{T} \mathbf{O}_{k+1} \mathbf{b}_{k+2:t}$$

- So all the inference tasks (e.g., filtering, smoothing) can be expressed as matrix multiplication which are typically efficient
- There are several improvements that can be done to reduce time and space complexity
 - Baum-Welch
 - Special case of Expectation-Maximization (EM) algorithm
 - Pros: Converge to a local maximum of the likelihood
 - Cons: Only point-estimation of params, no uncertainty estimation
 - Viterbi
 - Find the most likely sequence of hidden states
 - Pros: Fast approximation of BW
 - Cons: returns a local optimum
 - Gradient-based methods
 - For more complex models with differentiable form
 - Use gradient descent to optimize parameter

Hidden Markov Model: applications

- HMMs are very versatile to model systems that have hidden states which produce observable outputs
- Audio / speech
 - Speech recognition: map audio to phonemes, words
 - Speaker identification: model vocal traits to recognize a speaker
 - Music generation and transcription
- Biology / genomics
 - Gene prediction: find regions of DNA
 - Protein structure prediction
- Finance / economics
 - Market regime detection: e.g., bull/bear markets, volatility regimes
 - Credit scoring: observe purchases and estimate financial health (hidden variable)
- Security / anomaly detection
 - User behavior modeling: detect anomalous login patterns or usage activities
 - Intrusion detection: model normal traffic patterns to spot attacks

HMMs: limitations

- Short memory
 - Markov assumption, i.e., current state depends only on the previous state
 - Inefficient for capturing long-range dependencies or context
- Predefined and fixed number of states
 - Underestimating / overestimating the number of states can lead to underfitting or overfitting
- Stationarity assumption
 - Transition and sensor probabilities are assumed to be constant over time
- Use an atomic representation
 - The states have no internal structure and are simply labels
- Training is computationally expensive for large datasets
- Struggles with sparse data
- Hard to interpret when there is a lot of states or states don't have a clean meaning
- Alternatives

Markov random fields

- Reasoning over time
- HMMs
- **Markov random fields**
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference

Markov logic network

- Reasoning over time
- HMMs
- Markov random fields
- **Markov logic network**
- State space models and Kalman filter
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference

State space models and Kalman filter

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- **State space models and Kalman filter**
 - g-h filter
 - One dimensional Kalman filters
 - Multivariate Gaussians
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference

Reference

- <https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python>

Tracking objects

- Many problems can be formulated as “tracking objects”, e.g.,
 - Navigation of aircraft, drones, autonomous cars
 - Robotics: arm kinematics to predict the position of joints
 - Sensor fusion: merge multiple sensor readings
 - Finance: predict economic variables (e.g., stock prices)
 - Computer vision: track moving objects across video
 - Aerospace: radar tracking, missile, satellite
- Kalman filter is widely used for state estimation in dynamic systems, when measurements are noisy and uncertain
 - Track something over time using a combination of predictions (i.e., a model) and observations

Some guiding principles

- **The world is noisy**
 - A car might swerve around a pothole or brake for a pedestrian
 - Wind or ice might change the car's path
- **Sensors are noisy**
 - E.g., a kitchen scale gives different readings for the same object
- Data is better than a guess
 - Even if noisy
- **Knowledge is uncertain**
 - We alter beliefs based on evidence strength
- Use past information and system knowledge to estimate future information
 - E.g., if a car moves at a certain speed at time t , the speed at time $t + 1$ is likely close to the previous speed
- Never discard information, no matter how poor
 - E.g., two sensors, even if one is less accurate, are better than one

g-h filter

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
 - **g-h filter**
 - One dimensional Kalman filters
 - Multivariate Gaussians
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference

Example of weight: blending predictions and measurements

- Imagine going to the gym to gain muscle mass
 - Estimate your weight over time
- You could:
 1. **Predict your weight**
 - Track calorie intake and energy expense
 - Compute expected weight gain
 - Cons: Difficult to track food intake and exercise accurately
 2. **Measure your weight**
 - Use a scale
 - Cons: Scale is noisy, water weight fluctuates, different clothes
- Prediction doesn't match measurements
 - At time $t - 1$
 - Estimate: $\hat{x}_{t-1} = 158$
 - At time t :
 - Scale measures 164
 - Estimate $\hat{x}_{t|t-1} = 159$ based on calorie intake
- **What's your real weight?**
 - You need to blend prediction and measurement

Example of weight: correct gain_rate

- **Blend the estimates like:**

$$\text{estimate} = 0.6 \times \text{prediction} + (1 - 0.6) \times \text{measurement}$$

- You believe the prediction is more likely correct than the measurement
- **Algorithm**
 1. Start with an initial guess
 - Assume it's correct for now
 2. Predict the next weight based on the model
 3. Measure the weight
 4. Estimate the next weight by merging values:
 - The prediction is always between the prediction and the measurement
 5. Go back to first step

Example of weight:

- The black line is the actual weight, i.e., **ground truth**
- The initial guess is 160 lbs
- The **red line** is the **prediction** from previous day's weight
- The **measurements** are the circles
- The **blue line** is the **estimate** from the filter
 - Always falls between measurement and prediction
- It's not impressive since the prediction model describes the ground truth, so you don't need the measurements

Example of weight: learning gain_rate

- Consider when the model predicts a gain of -10lb/day, which is incorrect
 - Estimates diverge from measurements
- The filter needs a correct guess of the weight change rate
 - Also the rate of change can vary over time
- Solution: estimate the rate of change from measurements
 - “Data is better than a guess, even if it’s noisy”
 - Refine the estimate of the gain rate:

$$\text{new gain} = \text{old gain} + 0.3 (\text{measurement} - \text{prediction}) / 1 \text{ day}$$

- The “state” is given by weight and gain_rate, so you need to predict and update both

g-h filter

- The previous algorithm is called **g-h filter**
 - g : scaling used to blend predicted state and measurement
 - h : scaling used to update the parameter of the system model based on the measurements
- g-h filters have different values of g and h to achieve different properties
 - E.g., pick g to minimize the transient error when the derivative of the signal has a step (i.e., a discontinuity of the slope)
 - Many filters (including Kalman filter) are just generalizations of a g-h filter

Control theory nomenclature

- State space models were developed in control theory, so there is a different nomenclature
- **System**: object you want to estimate/track
- **Filter**: algorithm to estimate the state of the system
- **State of the system** x : current values you are interested in
 - E.g., weight
 - Part of the state might be hidden (i.e., not observable)
 - You cannot observe the entire state directly, only measure it indirectly
- **Measurement** z : the measured value of the system
 - It is observable
 - It can be inaccurate
 - E.g., 99.3kg instead of 100kg
- **State estimate** x_{est} : filter estimate of the state
- **System model**: mathematical model of the system
 - E.g., "weight today = weight yesterday + weight gain"
 - The system model is typically imperfect

g-h filter algorithm: pseudo-code

1. Initialization

- Initialize the state of the filter
- Initialize your belief in the state

2. Predict

- Use system model to predict state at next time step
- Adjust belief to account for uncertainty in prediction

3. Update

- Get measurement and associated belief about its accuracy
- Use as estimate of the next state a point between estimated state and measurement

Interpretation of g

- If $g = 0$:
 - The filter follows the system model, ignoring the measurements
- If g increases:
 - The filter follows the measurements more, ignoring the prediction
 - Useful when measurements are accurate and the system model is inaccurate
- If $g = 1$:
 - The filter follows only the measurements, ignoring the system model

Interpretation of h

- You might need to estimate some model parameters from data, e.g.,
 - The change of weight
 - The rate of change of the measurements
 - The speed of the car on different terrains
- If $h = 0$:
 - The filter follows the previous values of the rate of change of the underlying model
 - I.e., it adapts slowly to the change of the signals
- If $h = 1$:
 - The filter reacts to the transient rapidly if the signal varies significantly with respect to the time step
- **Note:** an incorrect initial state (e.g., initial value/rate of change) is similar to a changing state

One dimensional Kalman filters

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
 - g-h filter
 - **One dimensional Kalman filters**
 - Multivariate Gaussians
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference

Updating belief using Gaussians

- The Bayes theorem tells that:

$$\text{posterior} = \text{normalized}(\text{prior} \times \text{likelihood})$$

- If the prior and the likelihood are Gaussian the result is also Gaussian (conjugate prior)
 - The belief and probability are represented as a Gaussian
 - We can encode the PDF in terms of mean and std dev
 - Updating belief is equivalent to sum and multiplication of Gaussians
- Algorithm:

Create prior (using current estimate and system model)

```
prior = predict(x, process_model)
```

Create likelihood (using measurement).

```
likelihood = gaussian(z, sensor_var)
```

Update belief using prior and likelihood

```
posterior = update(prior, likelihood)
```

Sum of Gaussians

- The sum of two independent Gaussians

$$Normal(\mu_1, \sigma_1^2)$$

$$Normal(\mu_2, \sigma_2^2)$$

is a Gaussian $Normal(\mu, \sigma^2)$ with:

$$\mu = \mu_1 + \mu_2$$

$$\sigma^2 = \sigma_1^2 + \sigma_2^2$$

- The mean is the sum of the mean (by linearity)
- The variance always increases

Product of Gaussians

- The product of two independent Gaussians

$$\text{Normal}(\mu_1, \sigma_1^2)$$

$$\text{Normal}(\mu_2, \sigma_2^2)$$

is a Gaussian $N(\mu, \sigma^2)$ with:

$$\mu = \frac{\mu_1 \sigma_2^2 + \mu_2 \sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$

$$\sigma^2 = \frac{\sigma_1^2 \sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

- **Interpretation:**
 - The variance may be reduced as more information is incorporated
 - If one Gaussian N_1 is much narrower than the other (i.e., one measure is more accurate), the result is pushed towards N_1
 - If two Gaussians are similar (i.e., two measures corroborate each other), the result becomes more certain

Kalman Gain

- Assume that:
 - x is the model prediction
 - z indicates the measurements
- The mean of the posterior is:

$$\mu = \frac{\sigma_x^2 \mu_z + \sigma_z^2 \mu_x}{\sigma_x^2 + \sigma_z^2} = \frac{\sigma_x^2}{\sigma_x^2 + \sigma_z^2} \mu_z + \frac{\sigma_z^2}{\sigma_x^2 + \sigma_z^2} \mu_x = K \mu_z + (1 - K) \mu_x$$

- The Kalman Gain K :
 - Is the scaling term that mixes the prediction and the measurement
 - Depends on the ratio of uncertainty of prior and measurement

Kalman pseudo-algorithm

- The typical formulation of the Kalman filter is in terms of the “orthogonal projection” approach to minimize mean squared error
 - Instead of a Bayesian formulation
- Typical symbols used in Kalman literature:
 - x : state
 - P : variance of state (uncertainty, belief)
 - $f()$: system model
 - Q : system model error
 - z : measurement
 - R : measurement noise
- **Initialization**
 - Initialize state of filter $x = x_0$
 - Initialize belief in the state $P = P_0$
- **Predict**
 - Use system model to predict state at the next time step $x = f(x)$
 - Adjust belief to account for uncertainty in prediction $P = P + Q$
- **Update**
 - Get measurement z and belief about its accuracy R
 - Compute residual between estimated state x and z : $y = z - x$
 - Compute scaling factor (Kalman K) based on accuracy of prediction P and measurement R

Multivariate Gaussians

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
 - g-h filter
 - One dimensional Kalman filters
 - **Multivariate Gaussians**
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference

Multivariate state

- Often the state variable is multivariate, e.g.,
 - Position and velocity of a dog (probably uncorrelated)
 - Height and weight of an adult (correlated)
- **Variance** is a measure of how a population varies, e.g.,
 - Variance = 0 means constant
 - Large variance means lots of variation
- **Covariance** are correlated variances
 - E.g., height and weight are generally positively correlated
- **Covariance matrix**
 - The diagonal contains the variance for each variable
 - The off-diagonal elements contain the covariance between i and j variables
 - The covariance matrix is symmetric
- Correlation allows prediction
 - E.g., “as winter comes you predict you will spend more on heating your house”

Multivariate Gaussian

- The marginal of a multivariate Gaussian is 1-d Gaussian
- Consider a contour plot (i.e., the intersection of a 2-d Gaussian $z = f(x, y)$ with a plane $z = c$)
 - The contour plot is always an ellipses

Multiplying two multivariate Gaussians

- Given two multivariate Gaussians $\sim \text{Normal}(\underline{\mu}_i, \underline{\Sigma}_i)$
- The product of the Gaussians is still Gaussian $\sim \text{Normal}(\underline{\mu}, \underline{\Sigma})$

$$\underline{\mu} = \underline{\Sigma}_2(\underline{\Sigma}_1 + \underline{\Sigma}_2)^{-1}\underline{\mu}_1 + \underline{\Sigma}_1(\underline{\Sigma}_1 + \underline{\Sigma}_2)^{-1}\underline{\mu}_2$$
$$\underline{\Sigma} = \underline{\Sigma}_1(\underline{\Sigma}_1 + \underline{\Sigma}_2)^{-1}\underline{\Sigma}_2$$

- **Note:** this is a generalization of the 1-d case to multivariate

$$\mu = \frac{\mu_1\sigma_2^2 + \mu_2\sigma_1^2}{\sigma_1^2 + \sigma_2^2}$$
$$\sigma^2 = \frac{\sigma_1^2\sigma_2^2}{\sigma_1^2 + \sigma_2^2}$$

replacing:

- σ^2 with covariance matrix $\underline{\Sigma}$
- Division with matrix inversion

Multivariate filtering

- Covariance structure helps improve the estimate, e.g.,
 - You know an airplane direction can't change quickly
 - Knowing an approximate value for the velocity helps constrain possible next positions
- **E.g., airplane**
 - You are tracking a plane moving in a direction (1-d problem)
 - At time 1, you are fairly certain about the position $x = 0$, but you don't know the velocity
 - You plot position and velocity on an x-y plane
 - The covariance matrix between position and velocity is narrow and tall
 - It is narrow on the x-axis since you know that the position is around $x = 0$
 - It is tall on the y-axis because of your lack of knowledge about velocity
 - After 1 sec, you get a position update of $x = 5$
 - You can infer that the velocity is 5/s
 - The covariance matrix is then stretched diagonally

Multivariate Kalman filters

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- **Multivariate Kalman filters**
 - Tracking a dog with a Kalman filter
 - Non-linear filtering
- Dynamic Bayesian networks
- State space model
- Variational Inference

Notation

- A Bayesian notation $a|b$ means “ a given the evidence of b ”
 - The prior is $\hat{x}_{t|t-1}$, since you know only the information at time $t - 1$, i.e., the previous state
 - The posterior is $\hat{x}_{t|t}$, since you know all the information at time t , i.e., the measurement
- **A simpler notation:**
 - Indicate the “prior” version of the variables (i.e., after the system update) with an overline (E.g., \bar{x} , \bar{x} , $\bar{\mathbf{X}}$)
 - Omit the indices $t + 1$ and t and use an assignment notation (representing “update in place” of a variable):

$$x = x + 1$$

instead of the mathematical notation using a different variable for each time step:

$$x_{t+1} = x_t + 1$$

- With this notation:
 - The prior is $\bar{x} = \hat{x}_{t|t-1}$
 - The posterior is $x = \hat{x}_{t|t}$

Multivariate Kalman filter

- With the previous notation:
 - State update: $\bar{\mathbf{x}} = \mathbf{F}\mathbf{x} + \mathbf{B}\mathbf{u}$
 - State uncertainty: $\bar{\mathbf{P}} = \mathbf{F}\mathbf{P}\mathbf{F}^T + \mathbf{Q}$
 - Residual: $\mathbf{y} = \mathbf{z} - \mathbf{H}\bar{\mathbf{x}}$
 - Kalman gain: $\mathbf{K} = \bar{\mathbf{P}}\mathbf{H}^T(\mathbf{H}\bar{\mathbf{P}}\mathbf{H}^T + \mathbf{R})^{-1}$
 - Updated state: $\mathbf{x} = \bar{\mathbf{x}} + \mathbf{K}\mathbf{y}$
 - Update state uncertainty: $\mathbf{P} = (\mathbf{I} - \mathbf{K}\mathbf{H})\bar{\mathbf{P}}$
- Where
 - \mathbf{x} and \mathbf{P} are the state mean and covariance
 - \mathbf{F} is the state transition function
 - \mathbf{Q} is the system error (i.e., the noise in the model assessment)
 - \mathbf{B} and \mathbf{u} model the control inputs to the system
 - \mathbf{H} is the measurement function
 - \mathbf{z} and \mathbf{R} are the measurement mean and covariance
 - \mathbf{y} is the residual
 - \mathbf{K} is the Kalman gain
- Use the system model to predict the next state
 - When we multiply \mathbf{F} to \mathbf{x} we get the prior (i.e., the state before seeing any measurement)
- Form an estimate between the prior and the measurement

From univariate to multivariate Kalman filter

- Let's compare

| Definition | Univariate (Bayesian) | Univariate (Kalman) | Multivariate (Kalman) |
|------------------------|-----------------------------------------------------------------------------------------------|-----------------------------------|----------------------------------------|
| State update | $\bar{\mu} = \mu + \mu_f$ | $\bar{x} = x + dx$ | $\bar{x} = Fx + Bu$ |
| State uncertainty | $\bar{\sigma}^2 = \sigma^2 + \sigma_f^2$ | $\bar{P} = P + Q$ | $\bar{P} = FPF^T + Q$ |
| Residual | | $y = z - \bar{x}$ | $y = z - H\bar{x}$ |
| Kalman gain | | $K = \frac{\bar{P}}{\bar{P} + R}$ | $K = \bar{P}H^T(H\bar{P}H^T + R)^{-1}$ |
| Updated state | $\hat{\mu} = \frac{\bar{\sigma}^2 \mu_z + \sigma_z^2 \bar{\mu}}{\bar{\sigma}^2 + \sigma_z^2}$ | $x = \bar{x} + Ky$ | $x = \bar{x} + Ky$ |
| Upd. state uncertainty | $\sigma^2 = \frac{\bar{\sigma}^2 \sigma_z^2}{\bar{\sigma}^2 + \sigma_z^2}$ | $P = (1 - K)\bar{P}$ | $P = (I - KH)\bar{P}$ |

Designing a Kalman filter

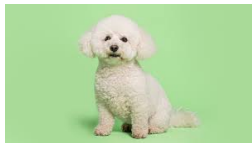
- The designer of the model needs to design:
 - The form of the state \underline{x} and \underline{P}
 - The system model \underline{F} and \underline{Q}
 - The measurement \underline{z} and \underline{R}
 - The measurement function \underline{H}
 - The control inputs \underline{B} and \underline{u} if there are control inputs

Tracking a dog with a Kalman filter

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
 - **Tracking a dog with a Kalman filter**
 - Non-linear filtering
- Dynamic Bayesian networks
- State space model
- Variational Inference

Tracking 1d dog: Problem formulation

- There is a dog moving on a 1-d track
- The dog moves approximately 1 meter per step
 - The velocity has variance due to noise/imperfect model specification
- There is a sensor that measures the position of the dog
 - The sensor has a certain error
- Time is discrete



(Nuvolo)

Tracking dog: Predict step

- At each step, the position is described with a Gaussian distribution $Normal(\mu, \sigma^2)$
- The position is part of the system's state, along with the velocity
 - The position is “observed” by a sensor
 - The velocity is a “hidden” variable
 - You could use more variables (E.g., acceleration, jerk, etc.)

Tracking dog: Design state covariance

- Initialize variances to reasonable values
 - E.g., $\sigma_{position} = 500m$ due to uncertainty about initial position
 - Top speed for a dog is 21m/s, so set $3\sigma_{velocity} = 21$
 - Assume covariances to be zero due to unknown initial correlation between position and velocity
 - P is diagonal

Tracking dog: Design system model

- Describe mathematically the behavior of the system

$$x_{t+1} = x_t + v\Delta t$$

- No model to predict how dog velocity changes over time
 - Assume it remains constant

$$\dot{x}_{t+1} = \dot{x}_t$$

- This is not correct, but if velocity doesn't change much, the filter will perform well
- Put the model in matrix form $\underline{x}_{t+1} = \underline{\underline{F}}\underline{x}_t$

Tracking dog: Predicting the system

- If we predict the system without measurements:
 - The state follows the system model
 - The state uncertainty grows
 - This is true even without system error (noise)

Tracking dog: Design system noise

- Consider a car driving on a road with cruise control on
- It should travel at constant speed:

$$x_t = \dot{x}_{t-1} \Delta t + x_{t-1}$$

- In reality, it is affected by unknown factors:
 - The cruise control is not perfect
 - Wind, hills, potholes affect the car
 - Passengers roll down windows, changing the drag profile of the car
- Model this as:

$$\dot{x}_t = \dot{x}_{t-1} + w$$

- Model all of this with a covariance matrix $\underline{Q} = \mathbb{E}[\underline{w} \cdot \underline{w}^T]$:
 - Assume the noise is iid, has zero mean, and is independent from the system
 - For these reasons, you don't have to change the position, only the velocity

Tracking dog: Design the control function

- Incorporate control inputs to predict state based on this information

$$\Delta \underline{\bar{x}} = \underline{\underline{B}} \underline{u}$$

- E.g., in the case of the car
 - Steering
 - Acceleration
- E.g., in the case of the dog, control inputs can be
 - The voice of its master
 - Seeing a squirrel

Tracking dog: Design the measurement function

- Kalman filter computes the update step in the measurement space
- If the measurement is in the same units as the state, the residual is simple to compute:

$$\text{residual} = \text{measured position} - \text{predicted position}$$

- E.g., assume we are tracking the position of the dog using a sensor that outputs a voltage
 - We cannot compute the residual as:

$$\text{measure voltage} - \text{predicted position}$$

- We need to convert the position into voltage
- The Kalman space allows to have a measurement matrix H to convert the state into a measurement

$$\underline{y} = \underline{z} - \underline{\underline{H}}\underline{\bar{x}}$$

Why working in measurement and not in state space?

- The problem is that it is possible to convert state into measurement, but not vice versa because of the hidden variables
 - E.g., transform position (discarding velocity) into voltage
 - If the sensor doesn't read velocity how do we estimate the measured velocity

Tracking dog: Design the measurement

- Typically \underline{z} is easy since it just contains the measurements from the sensor
- The measurement noise matrix $\underline{\underline{R}}$ can be difficult to estimate
 - Noise can be not Gaussian
 - There can be a bias in the sensor
 - The error can be not symmetrical (e.g., temperature sensor is less precise as the temperature increases)

Non-linear filtering

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
 - Tracking a dog with a Kalman filter
 - **Non-linear filtering**
- Dynamic Bayesian networks
- State space model
- Variational Inference

Optimality

- **Assumptions:**
 - Everything is linear
 - System and sensor noise is Gaussian
- Under these assumptions, the Kalman filter is optimal in a least square sense
- The Kalman filter is a mathematical model of the world
 - The output is only as accurate as the model of the world

The world is non-linear

- The Kalman filter uses linear equations and can only handle linear problems
- **The world is non-linear:**
 - System model can be non-linear:
 - Many physical systems are described by non-linear differential equations
 - E.g., a ball flying through air is affected by drag, leading to non-linear behaviors
 - Measurements can be non-linear:
 - To measure the height on a plane, you can measure the distance of the plane from the radar. Given the Pythagorean theorem, you get:

$$x = \sqrt{\text{dist}^2 - \text{height}^2}$$

- Rarely does a physical system have equations that can be solved analytically

Extended Kalman Filter

- Aka EKF
- EKF is a nonlinear version of the Kalman filter
 - Linearize the differential equations to compute the Jacobian (i.e., matrix of partial derivatives) at the point of the current estimate
 - Used for estimating the state of a nonlinear dynamic system
- Pros
 - Use the linear Kalman machinery
- Cons:
 - Analytical solution:
 - Difficult or impossible
 - Numerical solution:
 - Expensive computationally
 - Errors can compound forcing the filter to diverge (unstable)

Unscented Kalman filter

- Aka UKF
- It is superior to EKF in almost every way

Intuition of sampling techniques

- Assume you have a distribution X and a non-linearity ϕ
- For every measurement:
 - Generate many points from X
 - Pass them through the non-linear function ϕ
 - Approximate the result (E.g., compute mean and variance)
- **Problem:**
 - “How many points are needed to build an accurate output distribution”?
 - Even if $n=500,000$ points are enough for 1 dimension, for k dimensions you might need n^k points (curse of dimensionality)

Unscented transform

- Unscented transform estimates the result of applying a non-linear transformation to a probability distribution characterized by a finite number of moments (e.g., mean and covariance)
 - E.g., compute the non-linear transform of a distribution, given mean and covariance estimate
 - Called “unscented” since “it doesn’t stink.”
- **Intuition**
 - Given a PDF C with mean $\underline{\mu}$ and covariance $\underline{\Sigma}$
 - Encode mean and covariance in a set of points (sigma points) that represent a discrete PMF D with the same mean $\underline{\mu}$ and covariance $\underline{\Sigma}$
 - Propagate the discrete PMF D by applying the non-linear function ϕ to each point of the PMF
 - The mean and covariance of $\phi(D)$ approximate the mean and covariance of $\phi(C)$

Unscented transform: 1d case

- The idea is that we need 3 sigma points for a 1-d Gaussian
 - One point for the mean
 - Two points around the mean
- Each point has a weight

Unscented transform: sigma points

- Consider a distribution F and a non-linearity ϕ
- There are algorithms to generate points and weights (given the mean and covariance of F) to evaluate mean and covariance of F transformed through ϕ
- In n dimensions, we need $2n + 1$ points $\underline{\mathbf{x}}_i$ and weights w_i^m, w_i^c

$$\sum_i w_i^m = 1$$

$$\sum_i w_i^c = 1$$

$$\mu(\phi) = \sum_i w_i^m \phi(\underline{\mathbf{x}}_i)$$

$$\Sigma(\phi) = \sum_i w_i^c (\phi(\underline{\mathbf{x}}_i) - \mu(\phi))(\phi(\underline{\mathbf{x}}_i) - \mu(\phi))^T$$

- Note that selecting the sigma points has not a single solution

Monte Carlo sampling

- Use a finite number of randomly sampled points to represent the problem
- Run the points through the transformation (e.g., non-linear function / system you are modeling)
- Compute the results on the transformed points

Particle filters

- Aka Sequential Monte Carlo (SMC) methods
- = Monte Carlo algorithms to solve problems in Bayesian statistical inference (e.g., in filtering problems)
- The goal is to compute posterior distributions of the states, given some data

Generic particle filter algorithm

1. Randomly generate particle
 - Particles have all state variable that needs to be estimated (e.g., position, velocity)
 - Each particle has a weight representing the probability that it represents the actual state of the system
2. Predict next state of the particles
3. Update weighting
 - Update the weighting of the particles based on the measurements
 - Particles that match closely the measurements are weighted higher
4. Resample
 - Discard highly improbable particle
5. Compute estimate
 - Compute weighted mean and covariance of the particles to get an estimate of the state and uncertainty

Dynamic Bayesian networks

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
- **Dynamic Bayesian networks**
- State space model
- Variational Inference

Dynamic Bayesian Networks (DBNs)

- DBNs extend Bayesian networks to model temporal processes
- Main idea
 - “Unroll” the model over time
 - Capture intra-slice (within time) and inter-slice (across time) dependencies
- Each time slice includes:
 - State variables X_t
 - Evidence variables E_t
- Assumptions
 - First-order Markov process: current state depends only on the previous state
 - First-order sensor Markov process: evidence depends only on current state
 - Stationarity: each time slice is the same, both structure and parameters do not change over time
 - Structure and CPTs (Conditional Probability Tables) are the same across slices (time-homogeneous model)
 - No Gaussian distribution

DBNs vs HMMs

- DBNs generalize Hidden Markov Models (HMMs)
- HMMs are a special case with a single hidden and evidence variable per time step
- DBNs model more complex systems than HMMs by:
 - Using multiple state variables
 - Enables modeling large systems like robot localization with many state components
 - Exploiting sparse connections among variables yielding compact model
 - HMM: transition matrix of size $O(d^{2n})$
 - DBN: size $O(nd^k)$ with k bounded parents per variable

DBNs vs Kalman filters

- DBNs generalize Kalman filters
- Every Kalman filter can be represented in a DBN with:
 - Continuous variables
 - Linear / Gaussian conditional distributions
- Not every DBN can be represented by a Kalman filter, since:
 - DBN variables can mix discrete/continuous and non-Gaussian
 - Allow arbitrary conditional dependencies among variables
- **Pros of DBNs**
 - DBNs are applicable to broader domains including:
 - Fault diagnosis in networks
 - Complex system monitoring
- **Pros of Kalman filters:**
 - Optimal for linear systems with Gaussian noise
 - Support exact inference, DBNs often require approximate methods

Constructing a DBN

- Key components of a DBN
 - Prior distribution of state $\Pr(X_0)$
 - Transition model $\Pr(X_{t+1}|X_t)$
 - Sensor model $\Pr(E_t|X_t)$
 - Transition and sensor models are time-homogeneous
- Network topology includes:
 - Intra-slice topology
 - Inter-slice links

DBN example: Tracking a robot (1/3)

- **Problem:**

- Tracking a robot moving randomly on a line X over time

- **Initial model:**

- Position X_t and velocity \dot{X}_t as state variables
- Update via Newton's laws
- Easy to generalize for 2d or 3d by using a \underline{X}_t

- **Issue:**

- Velocity changes over time
- Battery exhaustion affects velocity systematically
- Effect depends on cumulative energy use
- Violates the Markov property (future depends on full history)

- **Solution:**

- Include battery level $Battery_t$ in the state X_t
- Restores the Markov assumption
- Allows motion prediction considering energy constraints
- Enables coherent reasoning about motion and power consumption over time

- **New requirement for state:**

- $S_t = (X_t, \dot{X}_t, BatteryLevel_t)$
- $E_t = (GPS_t, BMeter_t)$

DBN example: Tracking a robot (2/3)

- The DBN structure models both intra-slice (within time) and inter-slice (across time) dependencies
- Intra-slice dependencies:
 - Position X_t influences velocity \dot{X}_t
 - BatteryLevel $_t$ influences velocity \dot{X}_t
 - Battery $_{t+1}$ depends on Battery $_t$ and \dot{X}_t
 - BMeter $_t$ depends on Battery $_t$
 - GPS $_t$ depends on X_t
- Inter-slice dependencies:
 - Position X_{t+1} depends on Position X_t and velocity \dot{X}_t
 - Velocity \dot{X}_{t+1} depends on \dot{X}_t and Battery $_t$

DBN example: Tracking a robot (3/3)

- **Replicate for Multiple Time Slices:**
 - Create slices for $t = 0, 1, 2, \dots$ with the above variables and dependencies
 - Group each time slice vertically or horizontally for clarity
- **Unrolling:**
 - Visualize the full DBN by unrolling these slices over the desired number of time steps (e.g., three slices for $t = 0, 1, 2$)

Inference in DBNs

- DBNs are Bayesian networks and we can use the same inference algorithms
 - “Unroll” the DBN over time (i.e., replicate slices for each time step) and apply standard BN inference
 - We can’t unroll “forever”, but we limit to a certain number of slices to approximate a fixed amount of time dependency
- Use recursive methods to get a constant time and space update complexity
 - Variable elimination with temporal ordering
 - At time step $t + 1$ add slice $t + 2$ and remove slice t so one has always two slices to do inference
 - Maintains constant memory by keeping only two slices at a time
- Complexity:
 - Exponential in number of state variables ($O(nd^{n+k})$)
 - More efficient than full HMM representation ($O(d^{2n})$)
- Even though we can use DBNs to represent very complex temporal processes with many sparsely connected variables, we cannot reason efficiently and exactly about those processes
 - The prior joint distribution over all the variables is factorizable into its constituents CPTs
 - The posterior joint distribution conditioned on observation sequence is not

Approximate Inference in DBNs

- Particle Filtering:
 - Represent belief state with weighted samples (particles)
 - Steps: propagate, weight, resample
- Benefits:
 - Focuses computation on high-probability regions
 - Maintains manageable memory and time per step
- Challenges:
 - Approximation error
 - Sensitive to transition and observation model assumptions
- Used when exact inference is computationally impractical
- Real-world application: robot localization, speech recognition

DBN to represent changing model

- We can model the fact that the system can change over time
 - Transient failure: a sensor reads wrong measures
 - Persistent failure model: we can model it with additional variables (e.g., *SensorBroken*)

DBN: inference

- We can unroll the DBN and get a BayesNet and then perform exact or approximate inference with the known methods (e.g., MCMC)

DBN: optimization for inference

- Many optimizations are possible, e.g.,
 - Instead of running each sample through the entire DBN one can run all the samples evaluating one slice at a time to compute the posterior distribution

State space model

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
- Dynamic Bayesian networks
- **State space model**
- Variational Inference

Variational Inference

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- **Variational Inference**
 - Expectation-Maximization (EM) Algorithm

Expectation-Maximization (EM) Algorithm

- Reasoning over time
- HMMs
- Markov random fields
- Markov logic network
- State space models and Kalman filter
- Multivariate Kalman filters
- Dynamic Bayesian networks
- State space model
- Variational Inference
 - **Expectation-Maximization (EM) Algorithm**

EM Algorithm: Intuition and Applications

- Expectation-Maximization (EM) is a method for learning with hidden or missing data
 - Useful when some variables influencing the data are not directly observed
 - Works by iteratively improving parameter estimates
 - Alternates between estimating missing data and optimizing parameters
- Two main steps:
 - **E-step (Expectation)**: Estimate distribution over hidden variables using current parameters
 - **M-step (Maximization)**: Update parameters to maximize expected log-likelihood from the E-step
- Used in diverse settings:
 - Unsupervised clustering (e.g., Gaussian Mixture Models)
 - Learning with incomplete data in Bayesian networks
 - Hidden Markov Models (HMMs)
- Key property: EM increases data likelihood at each iteration
- Converges to a local maximum of the likelihood function
- No need for a step size parameter unlike gradient descent

EM Algorithm: Mechanics and Example in Gaussian Mixture Models

- Goal: Recover parameters of Gaussian components from unlabeled data
- **E-step:**
 - Compute $p_{ij} = P(C = i | x_j)$ using Bayes' rule
 - $p_{ij} \propto P(x_j | C = i)P(C = i)$
 - Calculate effective count: $n_i = \sum_j p_{ij}$
- **M-step:**
 - Update means: $\mu_i \leftarrow \sum_j p_{ij} x_j / n_i$
 - Update covariances: $\Sigma_i \leftarrow \sum_j p_{ij} (x_j - \mu_i)(x_j - \mu_i)^T / n_i$
 - Update weights: $w_i \leftarrow n_i / N$
- Intuition: Softly assign points to components, then re-estimate the components
- Example scenario:
 - 500 data points from a mix of 3 Gaussians
 - EM reconstructs original distribution closely after iterations
- Limitations:
 - Sensitive to initialization
 - May converge to poor local optima
 - Component collapse or merging can occur

Introduction to the Expectation–Maximization (EM) Algorithm

- **Purpose of EM Algorithm**

- Iterative method for finding maximum likelihood or maximum a posteriori (MAP) estimates in statistical models with latent variables
- Particularly useful when data is incomplete or has missing values

- **Key Concepts**

- **Observed Data (X)**: The data we can directly observe
- **Latent Variables (Z)**: Hidden or unobserved variables that influence the observed data
- **Parameters (θ)**: Unknown parameters to be estimated

- **Challenge Addressed**

- Direct maximization of the likelihood function $p(\mathbf{X}|\theta)$ is often intractable due to the presence of latent variables

- **EM Algorithm Overview**

- Alternates between estimating the expected value of the log-likelihood (E-step) and maximizing this expectation (M-step)

- **Applications**

- Widely used in clustering (e.g., Gaussian Mixture Models), natural language processing, and image reconstruction

The EM Algorithm: Step-by-Step

- **Initialization**

- Start with initial guesses for the parameters $\theta^{(0)}$

- **E-Step (Expectation Step)**

- Compute the expected value of the log-likelihood function, with respect to the conditional distribution of the latent variables given the observed data and current parameter estimates:

- $Q(\theta|\theta^{(t)}) = \mathbb{E}_{\mathbf{Z}|\mathbf{X},\theta^{(t)}} [\log p(\mathbf{X}, \mathbf{Z}|\theta)]$

- **M-Step (Maximization Step)**

- Maximize the expected log-likelihood found in the E-step to update the parameters:

- $\theta^{(t+1)} = \arg \max_{\theta} Q(\theta|\theta^{(t)})$

- **Iteration**

- Repeat E and M steps until convergence, i.e., until the parameters stabilize or the increase in likelihood is below a threshold

Mathematical Foundation of EM

- **Likelihood with Latent Variables**

- The marginal likelihood of the observed data is:
 - $p(\mathbf{X}|\theta) = \int p(\mathbf{X}, \mathbf{Z}|\theta) d\mathbf{Z}$

- **Intractability**

- The integral is often difficult to compute due to the complexity introduced by the latent variables

- **EM Solution**

- EM circumvents this by iteratively applying the E and M steps to find parameter estimates that locally maximize the likelihood

- **Convergence**

- Each iteration of EM is guaranteed to increase the likelihood function, ensuring convergence to a local maximum

Example: Gaussian Mixture Models (GMM)

- **Problem Setup**
 - Data is assumed to be generated from a mixture of Gaussian distributions, each with its own mean and covariance
- **Latent Variables**
 - Each data point is associated with a latent variable indicating the Gaussian component from which it was generated
- **E-Step in GMM**
 - Compute the posterior probabilities (responsibilities) that each data point belongs to each Gaussian component
- **M-Step in GMM**
 - Update the parameters (means, covariances, and mixing coefficients) of each Gaussian component using the responsibilities computed in the E-step
- **Iteration**
 - Repeat E and M steps until the parameters converge

Properties and Limitations of EM

- **Advantages**

- Can handle missing or incomplete data effectively
- Provides a framework for parameter estimation in complex models

- **Limitations**

- Converges to a local maximum, which may not be the global maximum
- Sensitive to initial parameter estimates; poor initialization can lead to suboptimal solutions

- **Extensions and Variants**

- **Variational Bayes**: Provides a fully Bayesian approach by estimating distributions over parameters
- **Generalized EM (GEM)**: Relaxes the requirement of fully maximizing the expected log-likelihood in the M-step
- **Expectation Conditional Maximization (ECM)**: Breaks the M-step into several conditional maximization steps

- **Practical Considerations**

- Multiple runs with different initializations can help in finding better solutions
- Monitoring the increase in likelihood can help in determining convergence