CPSC 440: Advanced Machine Learning

Message Passing

Mark Schmidt

University of British Columbia

Winter 2022

Last Time: Chapman-Kolmogorov Equations

- Chapman-Kolmogorov (CK) equations:
  - Recursive formula for computing $p(x_j = s)$ for all j and s in a Markov chain.

$$p(x_j) = \sum_{x_{j-1}=1}^{k} p(x_j \mid x_{j-1}) p(x_{j-1}),$$

  - Allows us to compute all these marginal probabilities $p(x_j = s)$ in $O(dk^2)$.
    - For a length-d chain with k states.

- We also discussed stationary distributions of homogeneous Markov chains.

$$\pi(c) = \sum_{c'} p(x_j = c \mid x_{j-1} = c') \pi(c'),$$

which are sets of marginal probabilities π that don't change over time.

  - You can think of this as the "long-run average probability of being in each state".
  - Stationary distribution exists and is unique if all transitions are positive.

Application: Voice Photoshop

- Adobe VoCo uses decoding in a Markov chain as part of synthesizing voices:

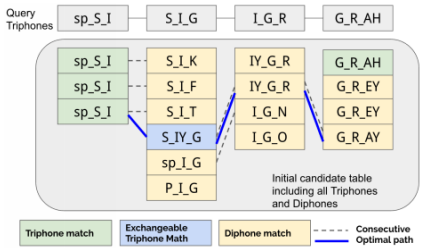

Fig. 7. Dynamic triphone preselection. For each query triphone (top) we find a candidate set of good potential matches (columns below). Good paths through this set minimize differences from the query, number and severity of breaks, and contextual mismatches between neighboring triphones.

http://gfx.cs.princeton.edu/pubs/Jin_2017_VTI/Jin2017-VoCo-paper.pdf

- https://www.youtube.com/watch?v=I314XLZ59iw

Decoding: Maximizing Joint Probability

- Decoding in density models: finding x with highest joint probability:

$$\underset{x_1,x_2,\ldots,x_d}{\text{argmax}}\ p(x_1, x_2, \ldots, x_d).$$

- For CS grad student ($d = 60$) the decoding is "industry" for all years.
  - The decoding often doesn't look like a typical sample.
  - The decoding can change if you increase d.

- Decoding is easy for independent models:
  - Here, $p(x_1, x_2, x_3, x_4) = p(x_1)p(x_2)p(x_3)p(x_4)$.
  - You can optimize $p(x_1, x_2, x_3, x_4)$ by optimizing each $p(x_j)$ independently.

- Can we also maximize the marginals to decode a Markov chain?

Example of Decoding vs. Maximizing Marginals

- Consider the "plane of doom" 2-variable Markov chain:

$$X = \begin{bmatrix} \text{"land"} & \text{"alive"} \\ \text{"land"} & \text{"alive"} \\ \text{"crash"} & \text{"dead"} \\ \text{"explode"} & \text{"dead"} \\ \text{"crash"} & \text{"dead"} \\ \text{"land"} & \text{"alive"} \\ \vdots & \vdots \end{bmatrix}.$$

- 40% of the time the plane lands and you live.
- 30% of the time the plane crashes and you die.
- 30% of the time the explodes and you die.

Example of Decoding vs. Maximizing Marginals

- Initial probabilities are given by

$$p(x_1 = \text{“land”}) = 0.4, \quad p(x_1 = \text{“crash”}) = 0.3, \quad p(x_1 = \text{“explode”}) = 0.3,$$

and transition probabilites are:

$$p(x_2 = \text{“alive”} \mid x_1 = \text{“land”}) = 1, \quad p(x_2 = \text{“alive”} \mid x_1 = \text{“crash”}) = 0,$$
$$p(x_2 = \text{“alive”} \mid x_1 = \text{“explode”}) = 0.$$

- If we apply the CK equations we get

$$p(x_2 = \text{“alive”}) = 0.4, \quad p(x_2 = \text{“dead”}) = 0.6,$$

so maximizing the marginals $p(x_j)$ independently gives (“land”, “dead”).
  - This actually has probability 0, since $p(\text{“dead”} \mid \text{“land”}) = 0$.

- Decoding considers the joint assignment to x_1 and x_2 maximizing probaiblity.
  - In this case it's (“land”, “alive”), which has probability 0.4.

Decoding with Dynamic Programming

- Note that decoding can't be done forward in time as in CK equations.
  - Even if $p(x_1 = 1) = 0.99$, the most likely sequence could have $x_1 = 2$.
  - So we need to optimize over all k^d assignments to all variables.

- Fortunately, we can solve this problem using dynamic programming.

- Ingredients of dynamic programming:
  1. Optimal sub-structure.
     - We can divide the problem into sub-problems that can individual be solved.
  2. Overlapping sub-problems.
     - The same sub-problems are reused several times.

Decoding with Dynamic Programming

- For decoding in Markov chains, we will use the following sub-problem:
  - Compute the highest probability sequence of length j ending in state s.
  - We'll use $M_j(s)$ as the probability of this sequence.

$$M_j(s) = \max_{x_1, x_2, \ldots, x_{j-1}} p(x_1, x_2, \ldots, x_{j-1}, x_j = s).$$

- Optimal sub-structure:
  - We can find the decoding by finding the s maximizing $M_d(s)$ (then "backtracking").
  - We can compute other $M_j(s)$ recursively (derivation of this coming up),

$$M_j(s) = \max_{x_{j-1}} \underbrace{p(x_j = s \mid x_{j-1})}_{\text{given}} \underbrace{M_{j-1}(x_{j-1})}_{\text{recurse}},$$

  with a base case of $M_1(s) = p(x_1 = s)$ (which is given by the initial probability).
- Overlapping sub-problems:
  - The same k values of $M_{j-1}(s)$ are used to compute the k values of $M_j(s)$.

Digression: Recursive Joint Maximization

- To derive the M_j formula, it will be helpful to re-write joint maximizations as

$$\max_{x_1,x_2} f(x_1,x_2) = \max_{x_1} f_1(x_1),$$

  where $f_1(x_1) = \max_{x_2} f(x_1,x_2)$ (this f_1 "maximizes out" over x_2).
  - This is similar to the marginalization rule in probability.

- Plugging in the definition of $f_1(x_1)$ we obtain:

$$\max_{x_1,x_2} f(x_1,x_2) = \max_{x_1} \underbrace{\max_{x_2} f(x_1,x_2)}_{f_1(x_1)}.$$

- You can do this trick repeatedly and/or with any number of variables.

Decoding with Dynamic Programming

- Derivation of recursive calculation $M_j(x_j)$ for decoding Markov chains:

$$
\begin{aligned}
M_j(x_j) &= \max_{x_1,x_2,\ldots,x_{j-1}} p(x_1, x_2, \ldots, x_j) && \text{(definition of } M_j(x_j)\text{)} \\
&= \max_{x_1,x_2,\ldots x_{j-1}} p(x_j \mid x_1, x_2, \ldots x_{j-1}) p(x_1, x_2, \ldots, x_{j-1}) && \text{(product rule)} \\
&= \max_{x_1,x_2,\ldots x_{j-1}} p(x_j \mid x_{j-1}) p(x_1, x_2, \ldots, x_{j-1}) && \text{(Markov property)} \\
&= \max_{x_{j-1}} \left\{ \max_{x_1,x_2,\ldots x_{j-2}} p(x_j \mid x_{j-1}) p(x_1, x_2, x_{j-1}) \right\} && (\max_{a,b} f(a,b) = \max_a \{\max_b f(a,b)\}) \\
&= \max_{x_{j-1}} \left\{ p(x_j \mid x_{j-1}) \max_{x_1,x_2,\ldots x_{j-2}} p(x_1, x_2, x_{j-1}) \right\} && (\max_i \alpha a_i = \alpha \max_i a_i \text{ for } \alpha \geq 0) \\
&= \max_{x_{j-1}} \underbrace{p(x_j \mid x_{j-1})}_{\text{given}} \underbrace{M_{j-1}(x_{j-1})}_{\text{recurse}} && \text{(definition of } M_{j-1}(x_{j-1})\text{)}
\end{aligned}
$$

- We also store the argmax over x_{j-1} for each (j, s) .
  - Once we have $M_j(x_j = s)$ for all j and s values, backtrack using these values to solve problem.

Example: Decoding the Plane of Doom

- We have $M_1(x_1) = p(x_1)$ so in "plane of doom" we have

$$M_1(\text{"land"}) = 0.4, \quad M_1(\text{"crash"}) = 0.3, \quad M_1(\text{"explode"}) = 0.3.$$

- We have $M_2(x_2) = \max_{x_1} p(x_2 \mid x_1) M_1(x_1)$ so we get

$$M_2(\text{"alive"}) = 0.4, \quad M_2(\text{"dead"}) = 0.3.$$

- $M_2(2) \neq p(x_2 = 2)$ because we needed to choose either "crash" or "explode".
  - And notice that $\sum_{c=1}^{k} M_2(x_j = c) \neq 1$ (this is not a distribution over x_2).

- We maximize $M_2(x_2)$ to find that the optimal decoding ends with "alive".
  - We now need to backtrack to find the state that lead to "alive", giving "land".

Viterbi Decoding

- The Viterbi decoding dynamic programming algorithm:
  1. Set $M_1(x_1) = p(x_1)$ for all x_1.
  2. Compute $M_2(x_2)$ for all x_2, store argmax of x_1 leading to each x_2.
  3. Compute $M_3(x_3)$ for all x_3, store argmax of x_2 leading to each x_3.
  4. ...
  5. Maximize $M_d(x_d)$ to find value of x_d in a decoding.
  6. Bactrack to find the value of x_{d-1} that lead to this x_d.
  7. Backtrack to find the value of x_{d-2} that lead to this x_{d-1}.
  8. ...
  9. Backtrack to find the value of x_1 that lead to this x_2.

- For a fixed j, computing all $M_j(x_j)$ given all $M_{j-1}(x_{j-1})$ costs $O(k^2)$.
  - Total cost is only $O(dk^2)$ to search over all k^d paths.
  - Has numerous applications like decoding digital TV.

Viterbi Decoding

- What Viterbi decoding data structures might look like ($d = 4, k = 3$):

$$M = \begin{bmatrix} 0.25 & 0.25 & 0.50 \\ 0.35 & 0.15 & 0.05 \\ 0.10 & 0.05 & 0.05 \\ 0.02 & 0.03 & 0.05 \end{bmatrix}, \quad B = \begin{bmatrix} \emptyset & \emptyset & \emptyset \\ 1 & 1 & 3 \\ 2 & 1 & 1 \\ 2 & 2 & 1 \end{bmatrix}.$$

- The $d \times k$ matrix M stores the values $M_j(s)$, while B stores the argmax values.

- From the last row of M and the backtracking matrix B, the decoding is $x_1 = 1, x_2 = 2, x_3 = 1, x_4 = 3$.

Conditional Probabilities in Markov Chains: Easy Case

- How do we compute conditionals like $p(x_j = c \mid x_{j'} = c')$ in Markov chains?

- Consider conditioning on an earlier time, like computing $p(x_{10} \mid x_3)$:
  - We are given the value of x_3.
  - We obtain $p(x_4 \mid x_3)$ by looking it up among transition probabilities.
  - We can compute $p(x_5 \mid x_3)$ by adding conditioning to the CK equations,

$$
\begin{aligned}
p(x_5 \mid x_3) &= \sum_{x_4} p(x_5, x_4 \mid x_3) && \text{(marg rule)} \\
&= \sum_{x_4} p(x_5 \mid x_4, x_3) p(x_4 \mid x_3) && \text{(product rule)} \\
&= \sum_{x_4} \underbrace{p(x_5 \mid x_4)}_{\text{given}} \underbrace{p(x_4 \mid x_3)}_{\text{recurse}} && \text{(Markov property).}
\end{aligned}
$$

  - Repeat this to find $p(x_6 \mid x_3)$, then $p(x_7 \mid x_3)$, up to $p(x_{10} \mid x_3)$.

Conditional Probabilities in Markov Chains with "Forward" Messages

- How do we condition on a future time, like computing $p(x_3 \mid x_6)$?
  - Need to sum over "past" values x_1 and x_2, and "future" values x_4 and x_5.

$$p(x_3 \mid x_6) \propto p(x_3, x_6) = \sum_{x_5}\sum_{x_4}\sum_{x_2}\sum_{x_1} p(x_1, x_2, x_3, x_4, x_5, x_6) \quad \text{(cond. prob. and marg. rule)}$$

$$= \sum_{x_5}\sum_{x_4}\sum_{x_2}\sum_{x_1} p(x_6 \mid x_5)p(x_5 \mid x_4)p(x_4 \mid x_3)p(x_3 \mid x_2)p(x_2 \mid x_1)p(x_1)$$

$$= \sum_{x_5} p(x_6 \mid x_5) \sum_{x_4} p(x_5 \mid x_4)p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1)p(x_1)$$

$$= \sum_{x_5} p(x_6 \mid x_5) \sum_{x_4} p(x_5 \mid x_4)p(x_4 \mid x_3) \sum_{x_2} p(x_3 \mid x_2)M_2(x_2)$$

$$= \sum_{x_5} p(x_6 \mid x_5) \sum_{x_4} p(x_5 \mid x_4)p(x_4 \mid x_3)M_3(x_3)$$

$$= \sum_{x_5} p(x_6 \mid x_5)M_5(x_5)$$

$$= M_6(x_6) \quad \text{(the values } M_j \text{ are called "forward messages")}$$

- $M_j(x_j)$ summarizes "everything you need to know up to time j for this x_j value".
  - Different x_3 will give different M_6 values, normalize these to get final result.

Conditional Probabilities in Markov Chains with "Backward" Messages

- We could exchange order of sums to do computation "backwards" in time:

$$
\begin{aligned}
p(x_3 \mid x_6) &= \sum_{x_1}\sum_{x_2}\sum_{x_4}\sum_{x_5} p(x_1)p(x_2 \mid x_1)p(x_3 \mid x_2)p(x_4 \mid x_3)p(x_5 \mid x_4)p(x_6 \mid x_5) \\
&= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1)p(x_3 \mid x_2) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \\
&= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1)p(x_3 \mid x_2) \sum_{x_4} p(x_4 \mid x_3)V_4(x_4) \\
&= \sum_{x_1} p(x_1) \sum_{x_2} p(x_2 \mid x_1)p(x_3 \mid x_2)V_3(x_3) \\
&= \sum_{x_1} p(x_1)V_1(x_1) \quad \text{(the values } V_j \text{ are called "backward messages")}
\end{aligned}
$$

- The V_j summarize "everything you need to know after time j for this x_j value".
  - Sometimes called "cost to go" function, as in "what is the cost for going to x_j".
  - Sometimes called a value function, as in "what is the future value of being in x_j".

Motivation for Forward-Backward Algorithm

- Why do care about being able to solve this "forward" or "backward" in time?
  - Cost is $O(dk^2)$ in both directions to compute conditionals in Markov chains.
- Consider computing $p(x_1 \mid A)$, $p(x_2 \mid A)$,..., $p(x_d \mid A)$ for some event A.
  - Need all these conditionals to add features or neural networks, and in HMMs.
- We could solve this in $O(dk^2)$ for each time, giving a total cost of $O(d^2k^2)$.
  - Using forward messages $M_j(x_j)$ at each time, or backwards messages $V_j(x_j)$.
- Alternately, the forward-backward algorithm computes all conditionals in $O(dk^2)$.
  - By doing one "forward" pass and one "backward" pass with appropriate messages.

Potential Function Representation of Markov Chains

- Forward-backward algorithm considers probabilities written in the form

$$p(x_1, x_2, \ldots, x_d) = \frac{1}{Z}\left(\prod_{j=1}^{d} \phi_j(x_j)\right)\left(\prod_{j=2}^{d} \psi_j(x_j, x_{j-1})\right).$$

- The ϕ_j and ψ_j functions are called potential functions.
  - They can map from a state (ϕ) or two states (ψ) to a non-negative function.
  - And normalizing constant Z ensures we sum/integrate to 1 (over all x_1, x_2,...,x_d).

- We can write Markov chains in this form by using (in this case $Z = 1$):
  - $\phi_1(x_1) = p(x_1)$ and $\phi_j(x_j) = 1$ when $j \neq 1$.
  - $\psi_j(x_{j-1}, x_j) = p(x_j \mid x_{j-1})$.

- Why do we need the ϕ_j functions?
  - To condition on $x_j = c$, set $\phi_j(c) = 1$ and $\phi_j(c') = 0$ for $c' \neq c$.
  - For "hidden Markov models", (HMMs) the ϕ_j will be the "emission probabilities".
  - For neural networks, ϕ_j will be $\exp$(neural network output) (generalizes softmax).

Forward-Backward Algorithm

- Forward pass in forward-backward algorithm (generalizes CK equations):
  - Set each $M_1(x_1) = \phi_1(x_1)$.
  - For $j = 2$ to $j = d$, set each $M_j(x_j) = \sum_{x_{j-1}} \phi_j(x_j)\psi_j(x_j, x_{j-1})M_{j-1}(x_{j-1})$.
    - "Multiply by new terms at time j, summing up over x_{j-1} values."

- Backward pass in forward-backward algorithm:
  - Set each $V_d(x_d) = \phi_d(x_d)$.
  - For $(d-1)$ to $j = 1$, set each $V_j(x_j) = \sum_{x_{j+1}} \phi_j(x_j)\psi_{j+1}(x_{j+1}, x_j)V_{j+1}(x_{j+1})$.

- We then have that $p(x_j) \propto \frac{M_j(x_j)V_j(x_j)}{\phi_j(x_j)}$.
  - Not obvious, see bonus for how it gives conditional in Markov chain.
  - We divide by $\phi_j(x_j)$ since it is included in both the forward and backward messages.
    - You can alternately shift ϕ_j to earlier/later message to remove division.
- We can also get the normalizing constant as $Z = \sum_{c=1}^{k} M_d(c)$.

Forward-Bacwkard for Decoding and Sampling

- Viterbi decoding can be generalized to use potentials ϕ and ψ:
  - Compute forward messages, but with summation replaced by maximization:

$$M_j(x_j) \propto \max_{x_{j-1}} \phi_j(x_j)\psi_j(x_j, x_{j-1})M_{j-1}(x_{j-1}).$$

  - Find the largest value of $M_d(x_d)$, then backtrack to find decoding.

- Forward-filter backward-sample is a potentials (ϕ and ψ) variant for sampling.
  - Forward pass is the same.
  - Backward pass generates samples (ancestral sampling backwards in time):
    - Sample x_d from $M_d(x_d) = p(x_d)$.
    - Sample x_{d-1} using $M_{d-1}(x_{d-1})$ and sampled x_d.
    - Sample x_{d-2} using $M_{d-2}(x_{d-2})$ and sampled x_{d-1}.
    - (continue until you have sampled x_1)

Outline

Markov Chains for Monte Carlo Estimation

- We have been discussing inference in Markov chains.
  - Sampling, marginals, stationary distribution, decoding, conditionals.

- We can also use Markov chains for inference in other models.
  - Most common way to do this is Markov chain Monte Carlo (MCMC).
  - Widely-used for approximate inference, including in Bayesian logistic regression.

- High-level ideas behind MCMC:
  - We want to use Monte Carlo estimates with a distribution p.
    - But we do not know how to generate IID samples from p.
  - Design a homoeneous Markov chain whose stationary distribution is p.
    - This is usually surprisingly-easy to do.
  - Use ancestral sampling to sample from a long version of this Markov chain.
  - Use the Markov chain samples within the Monte Carlo approximation.

Degenerate Example: "Stupid MCMC"

- Consider finding the expected value of a fair di:
  - For a 6-sided di, the expected value is 3.5.
- Consider the following "stupid MCMC" algorithm:
  - Start with some initial value, like "4".
  - At each step, roll the di and generate a random number u:
    - If $u < 0.5$, "accept" the roll and take the roll as the next sample.
    - Othewise, "reject" the roll and take the old value ("4") as the next sample.
- Generates samples from a Markov chain with this transition probability:

$$q(x_t \mid x_{t-1}) = \begin{cases} 7/12 & x_t = x_{t-1} \\ 1/12 & x_t \neq x_{t-1} \end{cases}.$$

  - I am using q so we do not confuse with the probability p we want to sample.

Degenerate Example: "Stupid MCMC"

- Stupid MCMC in action:
  - Start with "4", so record "4".
  - Roll a "6" and generate 0.234, so record "6".
  - Roll a "3" and generate 0.612, so record "6".
  - Roll a "2" and generate 0.523, so record "6".
  - Roll a "3" and generate 0.125, so record "3".
  - Roll a "2" and generate 0.433, so record "2".

- So our samples are 4,6,6,6,3,2...
  - If you run this long enough, you will spend $1/6$ of the time on each number.
    - Stationary distribution of stupid MCMC is $\pi(c) = 1/6$, so
    $$\pi(x) = p(x),$$
    which is the key feature underlying MCMC methods.
  - This property lets us use the dependent samples within Monte Carlo.

- It is "stupid" since it assumes we can generate IID samples from p.
  - If you can do that, you do not need MCMC.

Summary

- Viterbi decoding allow efficient decoding with Markov chains.
  - A special case of dynamic programming.
- Potential representation of Markov chains.
  - Non-negative potential ϕ at each time and ψ for each transition.
  - Useful for representing various conditional Markov chains.
- Forward-backward generalizes CK equations for potentials.
  - Allows computing all marginals in $O(dk^2)$.
- Markov chain Monte Carlo (MCMC) approximates complicated expectations.
  - Generate samples from a Markov chain that has p as stationary distribution.
  - Use these samples within a Monte Carlo approximation.

- Next time: the "number one algorithm" of the 20th century.

Computing Markov Chain Conditional using Forward-Backward

$$
\begin{aligned}
p(x_3 \mid x_6) &\propto \sum_{x_4}\sum_{x_5}\sum_{x_2}\sum_{x_1} p(x_1, x_2, x_3, x_4, x_5, x_6) \quad \text{(set up both sums to work "outside in")} \\
&= \sum_{x_4}\sum_{x_5}\sum_{x_2}\sum_{x_1} p(x_4 \mid x_3)p(x_5 \mid x_4)p(x_6 \mid x_5)p(x_3 \mid x_2)p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1)p(x_1) \\
&= \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \sum_{x_2} p(x_3 \mid x_2) \sum_{x_1} p(x_2 \mid x_1)M_1(x_1) \\
&= \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \sum_{x_2} p(x_3 \mid x_2)M_2(x_2) \\
&= \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5)M_3(x_3) \\
&= M_3(x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5) \quad \text{(take } M_3(x_3) \text{ outside sums)} \\
&= M_3(x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)p(x_6 \mid x_5)V_6(x_6) \quad (V_6(x_6) = 1) \\
&= M_3(x_3) \sum_{x_4} p(x_4 \mid x_3) \sum_{x_5} p(x_5 \mid x_4)V_5(x_5) \\
&= M_3(x_3) \sum_{x_4} p(x_4 \mid x_3)V_4(x_4) \\
&= M_3(x_3)V_3(x_3) \quad (\phi_3(x_3) = 1 \text{ so no division, normalize over } x_3 \text{ values to get final answer})
\end{aligned}
$$

Sequential Monte Carlo (Particle Filters)

- For continuous non-Gaussian Markov chains, we usually need approximate inference.
- A popular strategy in this setting is sequential Monte Carlo (SMC).
  - Importance sampling where proposal q_t changes over time from simple to posterior.
  - AKA sequential importance sampling, annealed importance sampling, particle filter.
    - And can be viewed as a special case of genetic algorithms.
  - "Particle Filter Explained without Equations":
    `https://www.youtube.com/watch?v=aUkBa1zMKv4`