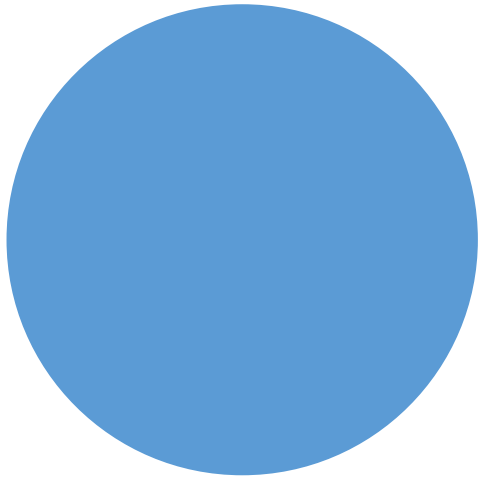
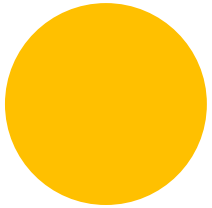
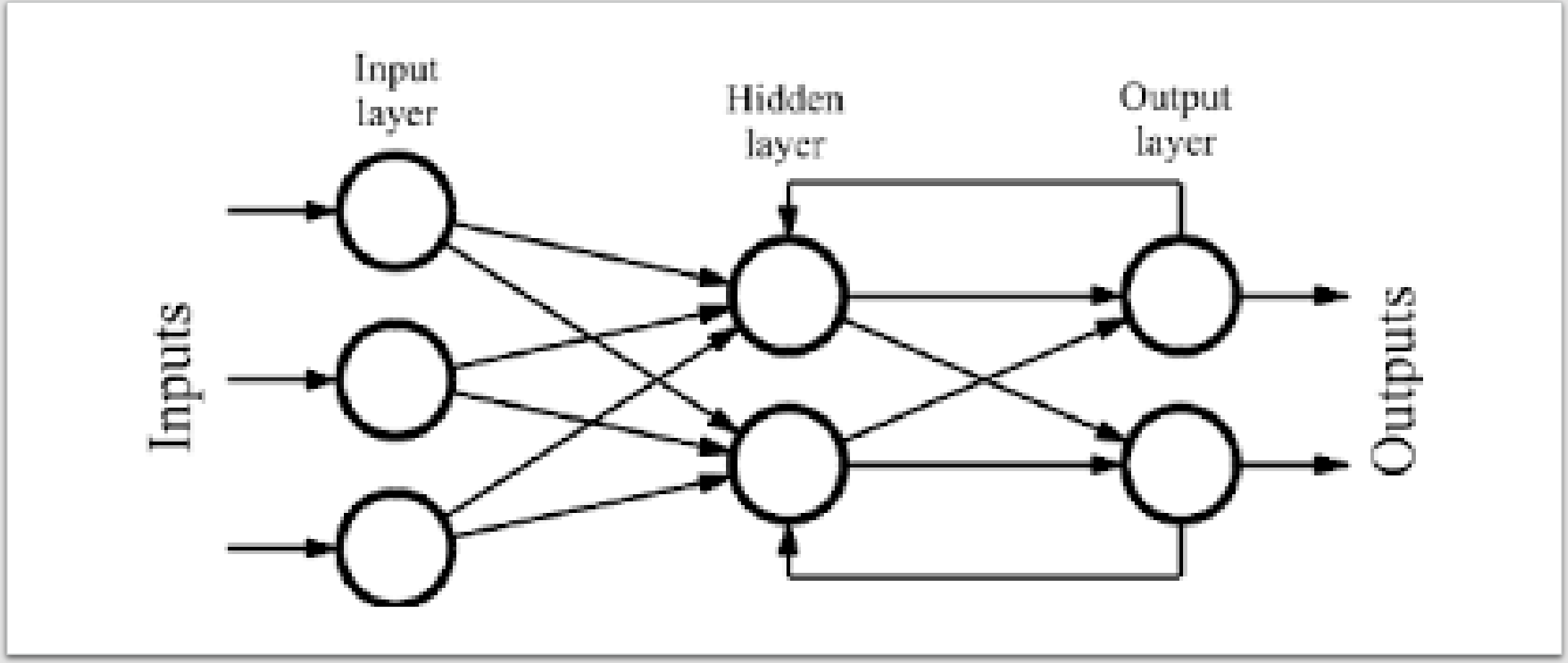



Hidden Markov Models

Scott O'Hara

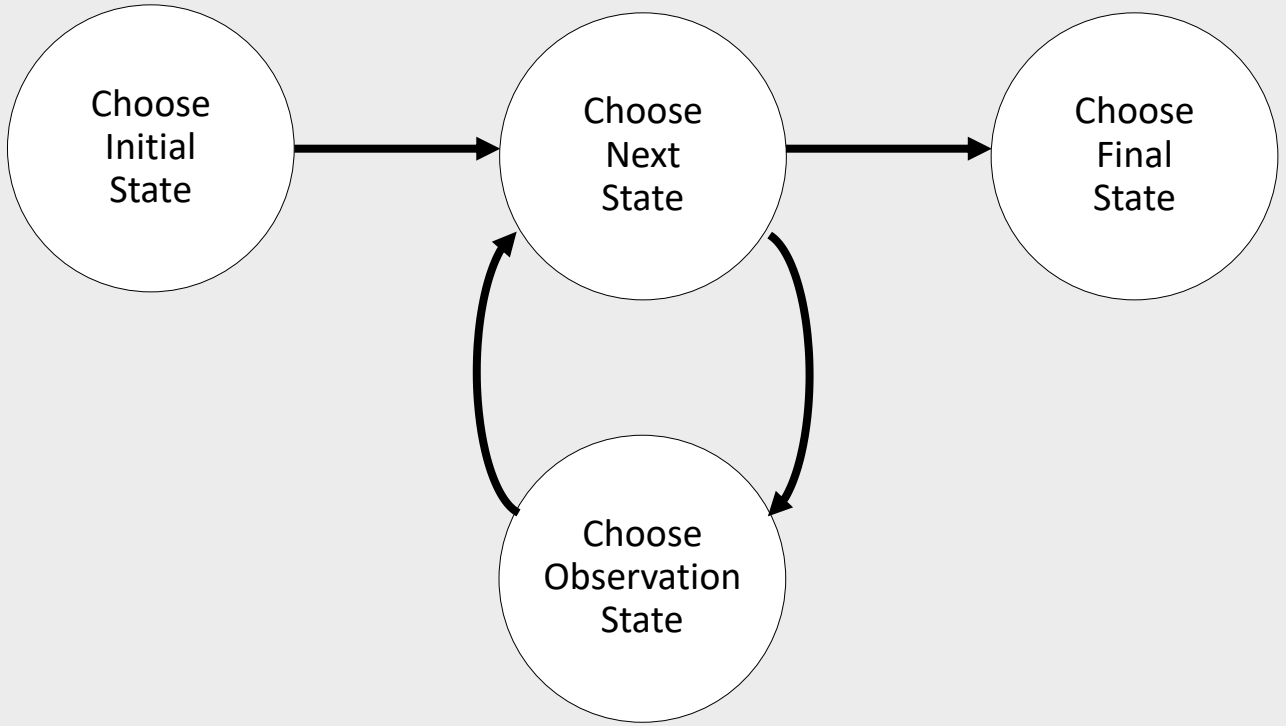
Metrowest Developers
Machine Learning Group

05/23/2018



RNN Discussion Reminded Me of HMMs

HMM Data Flow



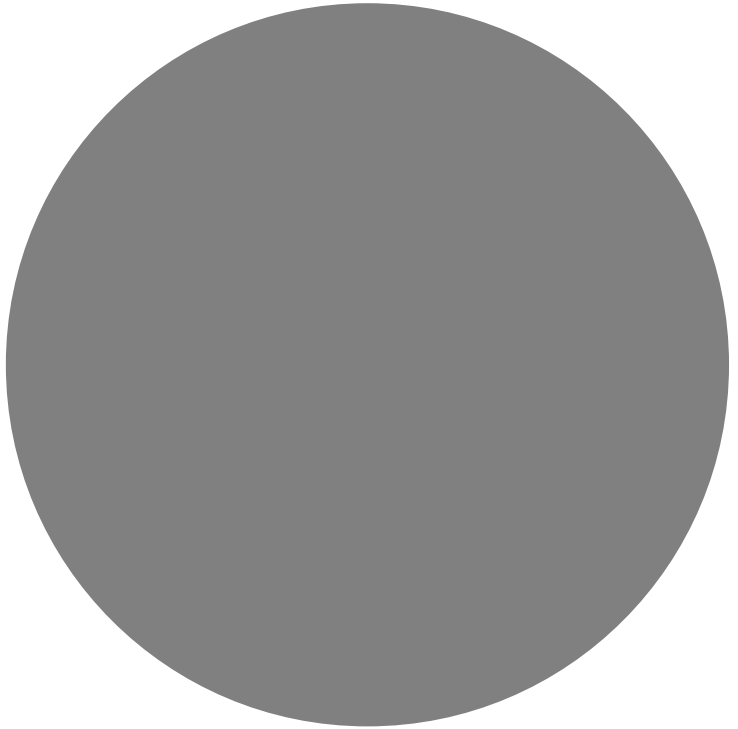
Contents:

- Introducing Hidden Markov Models (HMMs)
- Some Probability Preliminaries
- HMM Algorithms
- HMM Applications
- Training HMMs: Baum-Welch

Maybe Next Time:

- *Computing Belief States*
- *The Backward-Forward Algorithm*
- *The Viterbi Algorithm*
- *The Baum-Welch Algorithm*

Introducing Hidden Markov Models



- Markov Processes
- Hidden Markov Models (HMM)
- The Markov Assumptions

Markov Process

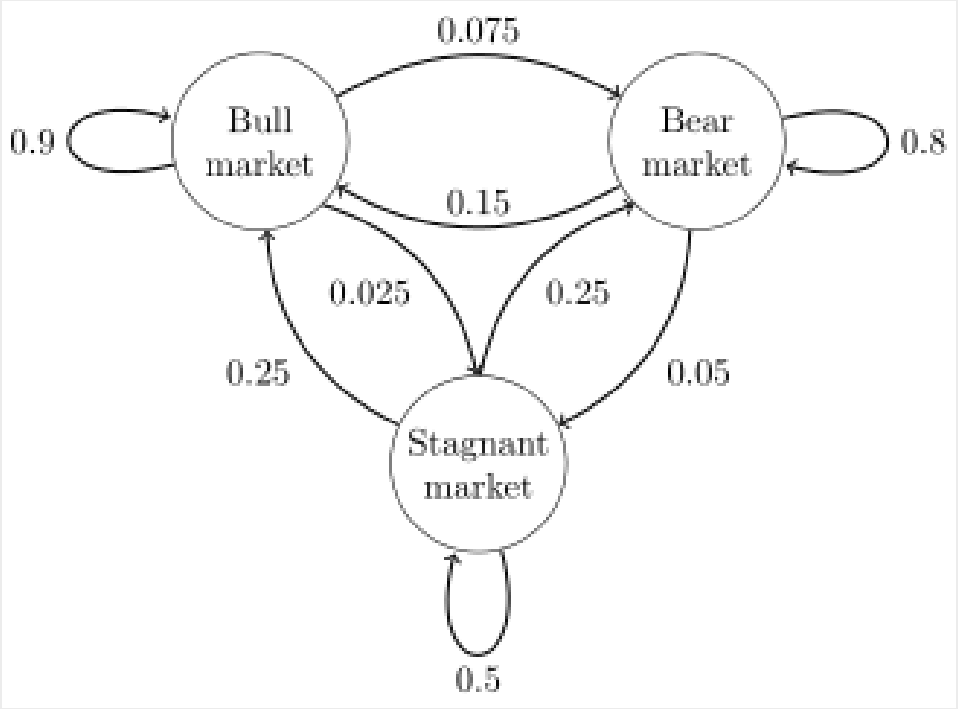
- A Markov process (or chain) is a stochastic model describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event.

  Source: https://en.oxforddictionaries.com/definition/us/markov_chain

- A Markov process has the following components:
  - A **set of N states**
  - A **transition probability matrix** representing the probability of moving from state i to state j. . For any state i, all probabilities must sum to 1.
  - A special **start state** and **end state**.

  Source: adapted from https://web.stanford.edu/~jurafsky/slp3/9.pdf

Markov Process Example



Source: https://upload.wikimedia.org/wikipedia/commons/4/47/MarkovChain1.png

A possible sequence of states:

Bull, Bull, Bull, Stagnant, Stagnant, Bull, Stagnant, Bear, Bear

Hidden Markov Models

- A Hidden Markov Model (HMM) is a statistical model in which the system being modeled is assumed to be a Markov process with unobserved (i.e. hidden) states.

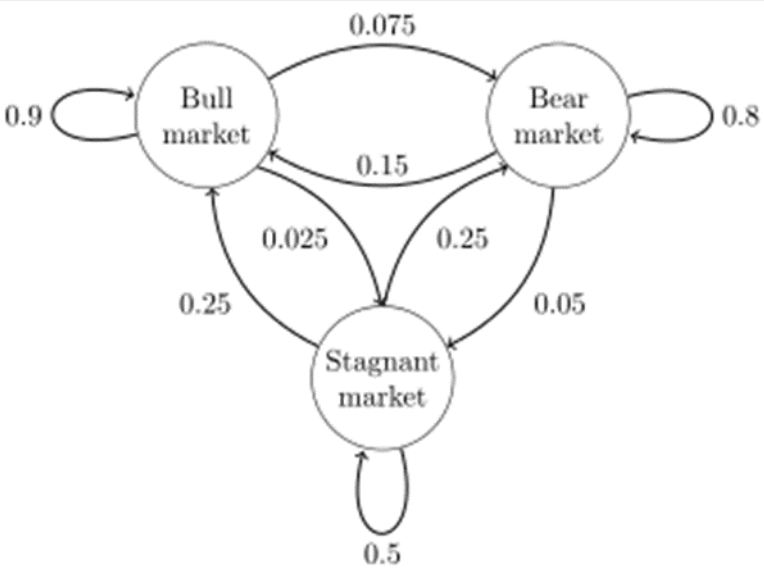
  Source: https://en.wikipedia.org/wiki/Hidden_Markov_model

- A Hidden Markov model has the following components:
  - A **set of N states**
  - An **initial probability distribution** over the states specifying the probability that state i is the first state in a sequence.
  - A **transition probability matrix** representing the probability of moving from state i to state j. . For any state i, all probabilities must sum to 1.
  - A sequence of **observation states**.
  - a sequence of **observation likelihoods** each expressing the probability of an observation i being generated from state i.

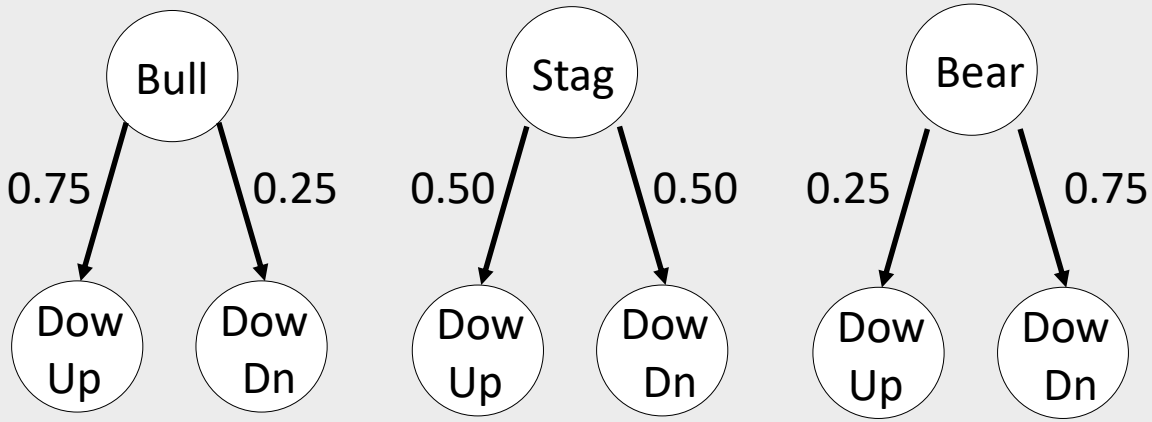
  Source: adapted from https://web.stanford.edu/~jurafsky/slp3/9.pdf

HMM Example (1)

States and Transitions:

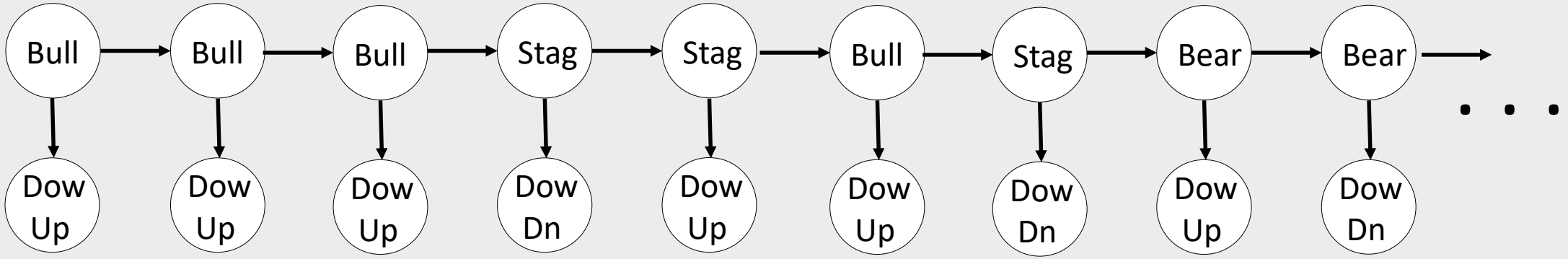


States and Observations:

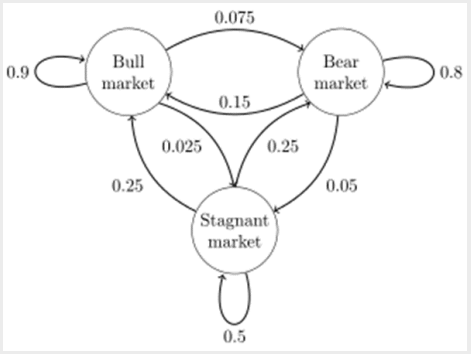


HMM Example (2)

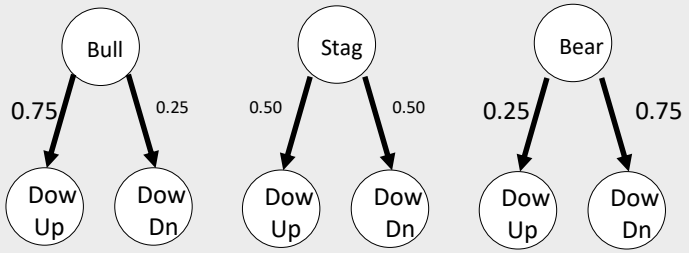
An Observation Sequence:



States and Transitions:



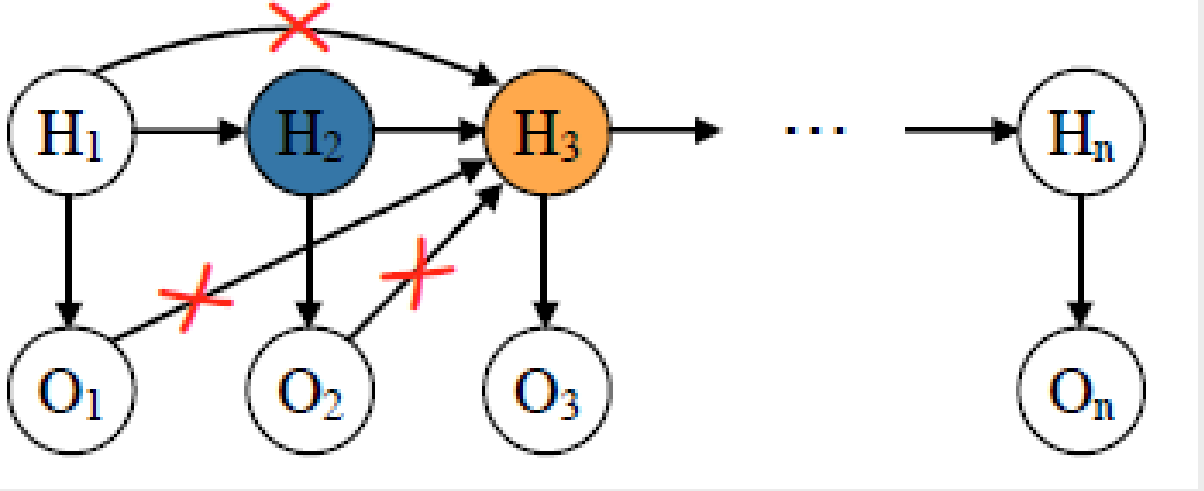
States and Observations:



Markov Assumption 1

The future is independent of the past given the present:

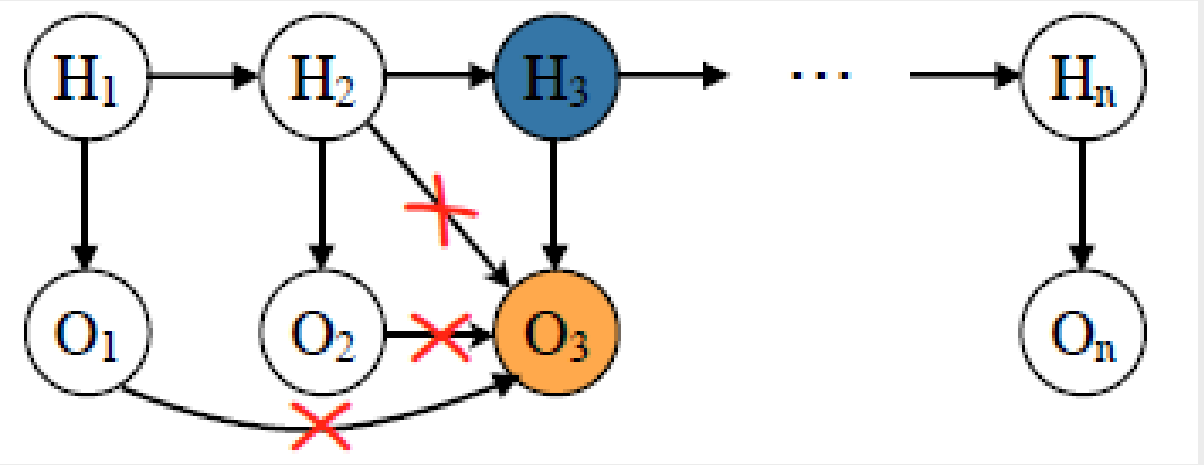
$$P(H_{t+1}|H_1, \dots, H_t, O_1, \dots, O_t) = P(H_{t+1}|H_t)$$



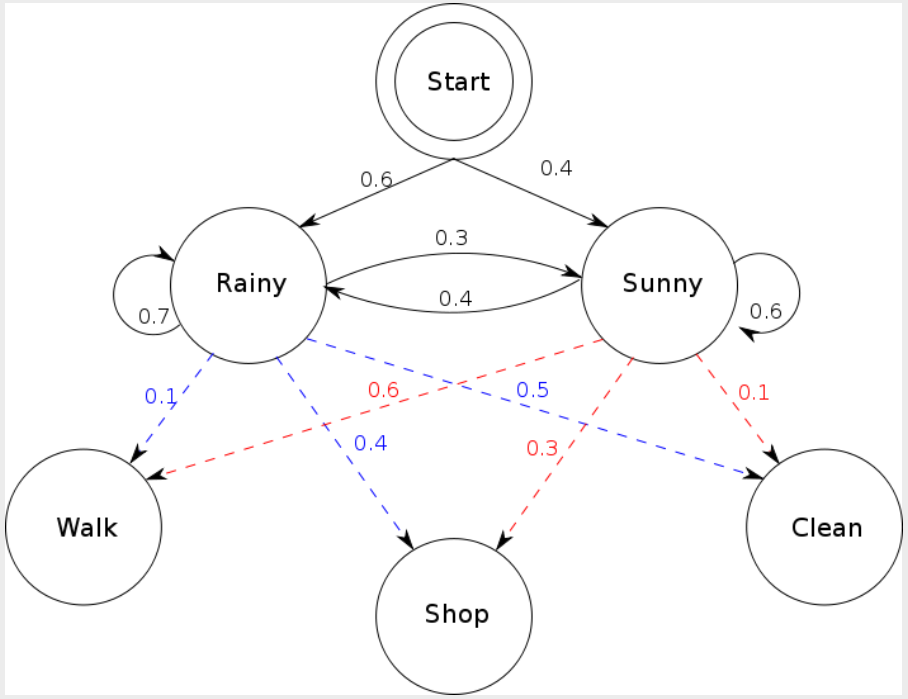
Markov Assumption 2

Observations depend only on the current hidden state:

$$P(O_t|H_1, \dots, H_t, O_1, \dots, O_t) = P(O_t|H_t)$$



HMM Data Structures

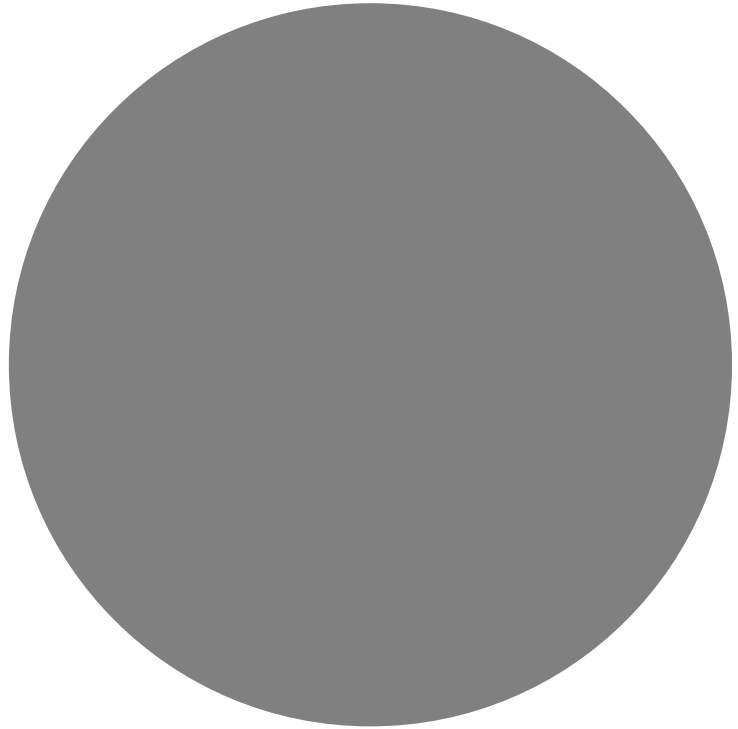


By Terencehonles - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=8384168

```
# n = 2 hidden states
states = ('Rainy', 'Sunny')

# m = 3 observation states
observations = ('walk', 'shop', 'clean')

# n x 1 start table
start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

# n x n transition table
transition_probability = { 'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
                           'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6}, }

# n x m observation table
emission_probability = { 'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
                         'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1}, }
```

Adapted from https://en.wikipedia.org/w/index.php?title=Hidden_Markov_model&oldid=841523243

Some Probability Preliminaries



Some Probability Preliminaries

- Discrete Random Variables
- Probabilities and the Sum Rule
- Joint Probability
- Conditional Probability
- The Product Rule
- The Chain Rule
- Bayes' Rule
- Marginalization

Discrete Random Variables

- *Discrete Variable* – a variable that can only take on a finite number of values.
- *Random Variable* - a variable whose possible values are outcomes of a random process.
- Random variables are represented as bold-faced capital letters e.g., **A** or **B.**
- For random variable **X**, the set of possible values it may take is represented by the lowercase X.
- A ***Discrete Random Variable*** is a discrete variable whose possible values are outcomes of a random process.

Probabilities and the **Sum Rule**

- *P(**X**=x)* – probability that discrete random variable ***X*** equals the value *x ϵ X.*
- *P(**X**)* is always a value between 0.0 and 1.0.

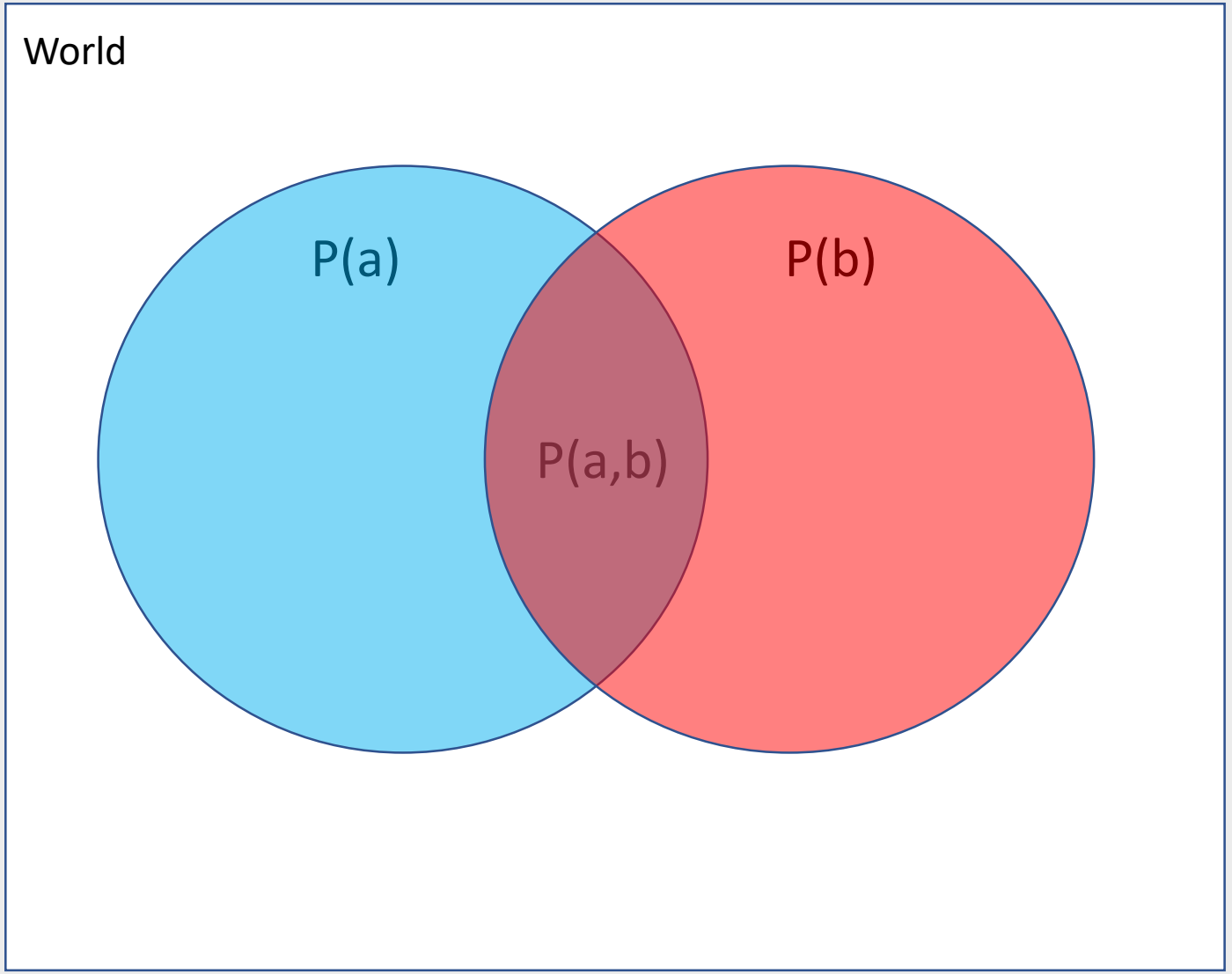
$$0.0 \leq P(X) \leq 1.0$$

$$\sum_{x \in X} P(X) = 1$$

Joint Probability

Identities:

P(**A**,**B**) = P(**B**,**A**)

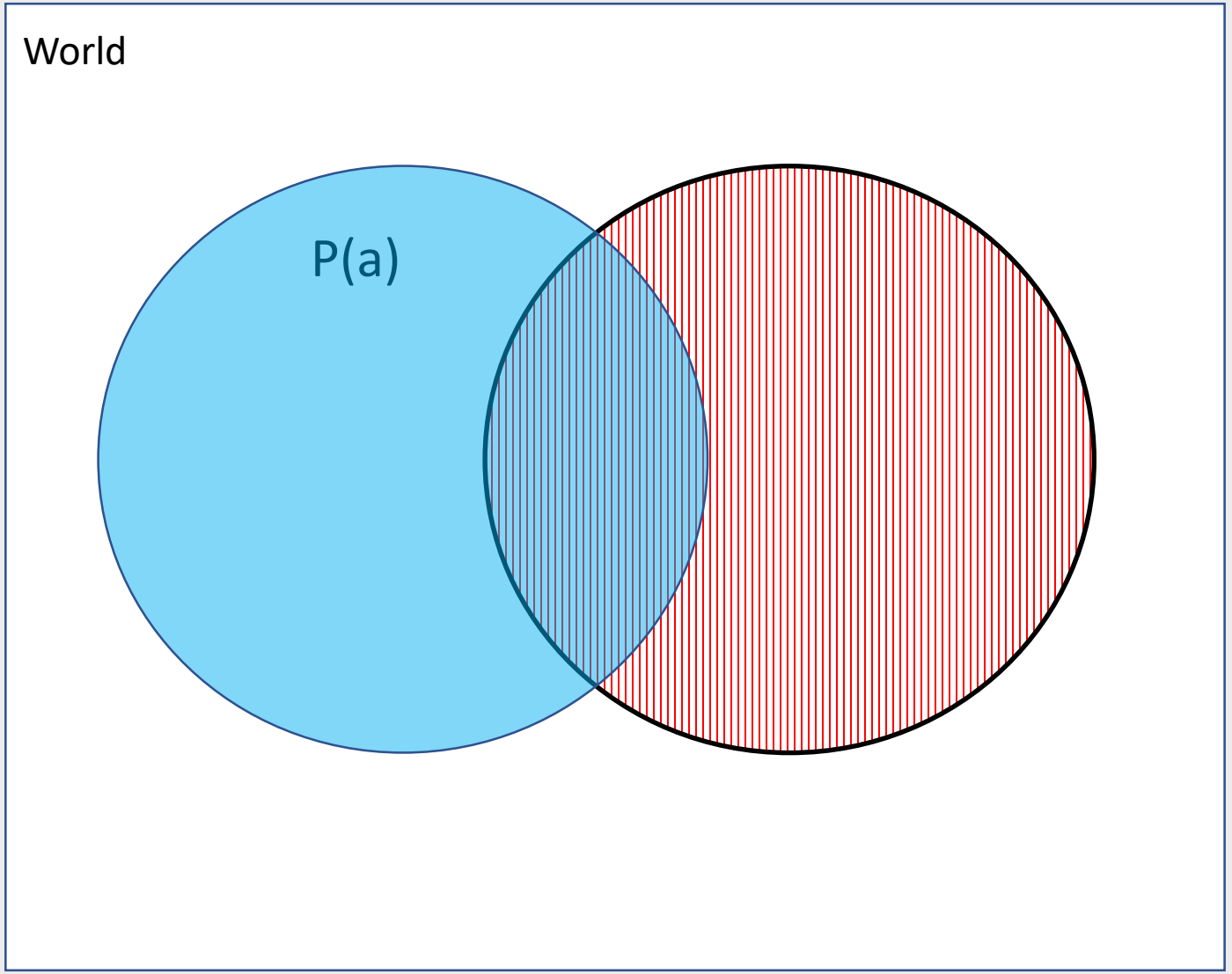


Conditional Probability

P(**A**|**B**) – What is the probability that **A** is true given that **B** is true?

Identities:

$$\mathrm{P}(\boldsymbol{A}|\boldsymbol{B}) = \frac{P(\boldsymbol{A},\boldsymbol{B})}{P(\boldsymbol{B})}$$



The Product Rule

- $\mathrm{P}(\boldsymbol{A}|\boldsymbol{B}) = \frac{P(\boldsymbol{A},\boldsymbol{B})}{P(\boldsymbol{B})}$

Conditional Probability

$$P(\boldsymbol{A},\boldsymbol{B}) = \mathrm{P}(\boldsymbol{A}|\boldsymbol{B})\ P(\boldsymbol{B}) = \mathrm{P}(\boldsymbol{B}|\boldsymbol{A})\ P(\boldsymbol{A})$$

The Chain Rule (Generalized Product Rule)

- $P(\boldsymbol{A}, \boldsymbol{B}) = \mathrm{P}(\boldsymbol{A}|\boldsymbol{B})\, P(\boldsymbol{B})$ Product Rule

$$P(\boldsymbol{A_1}, \boldsymbol{A_2}, \ldots, \boldsymbol{A_n}) = \mathrm{P}(\boldsymbol{A_1}|\boldsymbol{A_2}, \ldots, \boldsymbol{A_n})\, P(\boldsymbol{A_2}, \ldots, \boldsymbol{A_n})$$

Chain Rule Example

- $P(\boldsymbol{A_1}, \boldsymbol{A_2}, \ldots, \boldsymbol{A_n}) = \mathrm{P}(\boldsymbol{A_1}|\boldsymbol{A_2}, \ldots, \boldsymbol{A_n})\mathrm{P}(\boldsymbol{A_2}, \ldots, \boldsymbol{A_n})$

  Chain Rule

- Example:
  - $P(\boldsymbol{A_1}, \boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4}) = \mathrm{P}(\boldsymbol{A_1}|\boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4})\mathrm{P}(\boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4})$

    $= \mathrm{P}(\boldsymbol{A_1}|\boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4})\mathrm{P}(\boldsymbol{A_2}|\boldsymbol{A_3}, \boldsymbol{A_4})\mathrm{P}(\boldsymbol{A_3}, \boldsymbol{A_4})$

    $= \mathrm{P}(\boldsymbol{A_1}|\boldsymbol{A_2}, \boldsymbol{A_3}, \boldsymbol{A_4})\mathrm{P}(\boldsymbol{A_2}|\boldsymbol{A_3}, \boldsymbol{A_4})\mathrm{P}(\boldsymbol{A_3}|\, \boldsymbol{A_4})\mathrm{P}(\boldsymbol{A_4})$

Bayes' Rule

- $P(\boldsymbol{A}, \boldsymbol{B}) = \mathrm{P}(\boldsymbol{A}|\boldsymbol{B})\ P(\boldsymbol{B}) = \mathrm{P}(\boldsymbol{B}|\boldsymbol{A})\ P(\boldsymbol{A})$ Product Rule
- $\mathrm{P}(\boldsymbol{A}|\boldsymbol{B})\ P(\boldsymbol{B}) = \mathrm{P}(\boldsymbol{B}|\boldsymbol{A})\ P(\boldsymbol{A})$
- $\mathrm{P}(\boldsymbol{B}|\boldsymbol{A})\ P(\boldsymbol{A}) = \mathrm{P}(\boldsymbol{A}|\boldsymbol{B})\ P(\boldsymbol{B})$

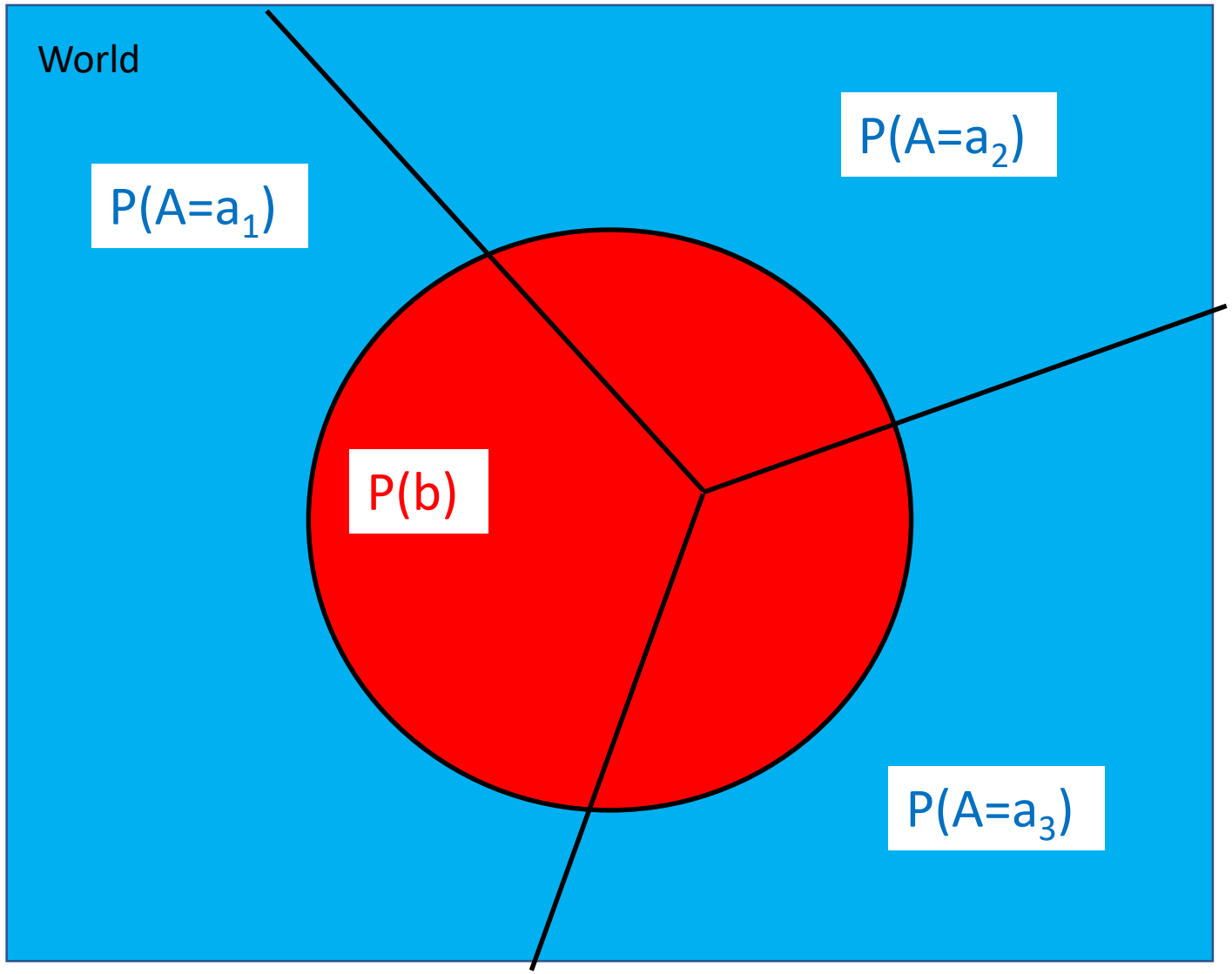
$$\mathrm{P}(\boldsymbol{B}|\boldsymbol{A}) = \frac{\mathrm{P}(\boldsymbol{A}|\boldsymbol{B})\ P(\boldsymbol{B})}{P(\boldsymbol{A})}$$

Marginalization

$$P(\boldsymbol{B}) = \sum_{a} P(\boldsymbol{A} = a, \boldsymbol{B})$$

where $A = \{a_1, a_2, a_3\}$

$\sum_{x \in X} P(\boldsymbol{X}) = 1$ (Sum Rule)



HMM Algorithms

The 3 HMM Algorithms

Forward-Backward – computes the probability of a sequence of observations $o_1, \ldots, o_n$.

Viterbi – computes the most likely sequence of hidden states $h_1, \ldots, h_n$ that would result from an observation sequence $o_1, \ldots, o_n$.

Baum-Welch – trains the parameters of HMM that so that it maximizes the likelihood of the training data.

The Forward-Backward Algorithm

- Given an HMM, the forward-backward algorithm takes as input an observation sequence $o_1, \ldots, o_n$, and computes $P(o_1, \ldots, o_n)$, i.e., the probability that the sequence would be generated by the HMM model.
- Uses a dynamic programming approach, which is possible because of the Markov assumptions.
- Relatively efficient at $O(m^2 n)$ where *m* is the number of hidden states and *n* is the sequence length.
- Forms the basis for *sequence classification*.
- There are 2 forms: the forward algorithm and the backward algorithm. There is no advantage of one over the other.

The Viterbi Algorithm

- Given an HMM, the Viterbi algorithm computes the most likely sequence of hidden states $h_1, \ldots, h_n$ that would result in a given observation sequence $o_1, \ldots, o_n$
- Uses a dynamic programming approach
- Similar to the Forward-Backward algorithm but has both a forward and backward pass.
- $O(m^2 n)$ where *m* is the number of hidden states and *n* is the sequence length.
- Used to find the “best explanation” for a given observation sequence.

The Baum-Welch Algorithm

- Given a set of observation sequences (training data) and the number of hidden states *m*, train the parameters of an *m*-state HMM that maximizes the likelihood of the training data.
- The parameter include the probabilities in the start table, the transition table and the observation table.
- The algorithm is a hill-climbing algorithm and is an application of the more general *expectation-maximization* algorithm.
- Uses Forward-Backward algorithm

Baum-Welch Algorithm Cost

- m hidden states
- N training sequences
- n max length per sequence
- $O(m^2 N n)$
- How many iterations are needed? **Unknown**

Some HMM Variants

There are many HMM variants

- **N-order HMMs** – HMM depends on n previous states.
- **Input/Output HMMs** – adds a visible input node I_t at each time t. Each hidden node and observation node at time t receives an input from I_t.
- **Factorial HMMs** – allow multiple chains of hidden nodes. Does not satisfy Markov property.
- **Switching Auto-Regressive HMMs (SAR-HMMs)** – Use real-valued observations that are correlated to previous observations by linear regression.
- **Kalman Filter** – HMM cousin where hidden states are continuous real values as are the observations.
- **Etc., etc.**

HMM Applications

What Problems HMMs Can Solve

Filtering – given a sequence of observations $o_1, \ldots, o_t$, find the most likely state h_t i.e., where $P(h_t|o_1, \ldots, o_t)$ is maximal.

Smoothing – given a sequence of observations $o_1, \ldots, o_T$ and $t < T$, find the most likely state h_t i.e., where $P(h_t|o_1, \ldots, o_T)$ is maximal.

Prediction – given a sequence of observations $o_1, \ldots, o_t$, predict the most likely observation o_{t+1} i.e., where $P(o_{t+1}|o_1, \ldots, o_t)$ is maximal.

Classification – classify a sequence of observations $o_1, \ldots, o_t$ by assigning a class *c* associated with the HMM_c that computes the highest probability of generating the sequence.

Explanation – Given a a sequence of observations $o_1, \ldots, o_t$, find a sequence of hidden states $h_1, \ldots, h_t$ that best explains the observations.

Some Real HMM Application Areas

- **Computational Finance**
- **Speech Recognition e.g., Siri**
- **Speech Synthesis**
- **Part-of-Speech Tagging**
- **Protein Folding**
- **Single-molecule kinetic analysis**
- **Target tracking (Kalman filter)**
- **etc., etc.**

Sources:

"The Master Algorithm: How the Quest for the Ultimate Learning Machine Will Remake Our World", Pedro Domingos, 2015.

https://en.wikipedia.org/wiki/Hidden_Markov_model

Classifying an Observation Sequence

Approach 1:

- Train an HMM_i for each class c_i in C.
- Given observation sequence $o_1, \ldots, o_n$, for each c_i, compute $P_i(o_1, \ldots, o_n)$.
- Classify sequence as c_{max} for $max = \mathrm{argmax}_i \, P_i(o_1, \ldots, o_n)$

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Classifying an Observation Sequence

Approach 2: Weight probabilities by prior probabilities for each class.

- Compute prior probability $P(c_i)$ for each class c_i in C.
- Train an HMM_i for each class c_i in C.
- Given observation sequence $o_1, \ldots, o_n$, compute $\mathrm{argmax}_i\, P(c_i|o_1, \ldots, o_n)$.
- $\underset{i}{\mathrm{argmax}}\, P(c_i|o_1, \ldots, o_n) = \mathrm{argmax}_i \frac{P(o_1, \ldots, o_n|c_i)P(c_i)}{P(o_1,\ldots,o_n)}$ (by Bayes' rule)

  $= \mathrm{argmax}_i\, P(o_1, \ldots, o_n|c_i)P(c_i)$ ($o_1, \ldots, o_n$ has no effect on max)
- Classify sequence as c_{max} for $\max = \underset{i}{\mathrm{argmax}}\, P(o_1, \ldots, o_n|c_i)P(c_i)$.

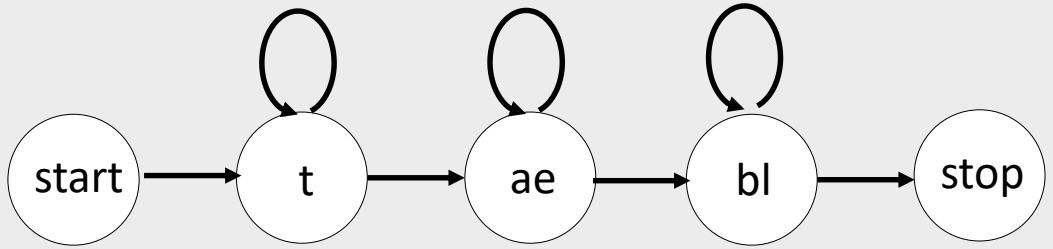
Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Application Example: Word Recognition

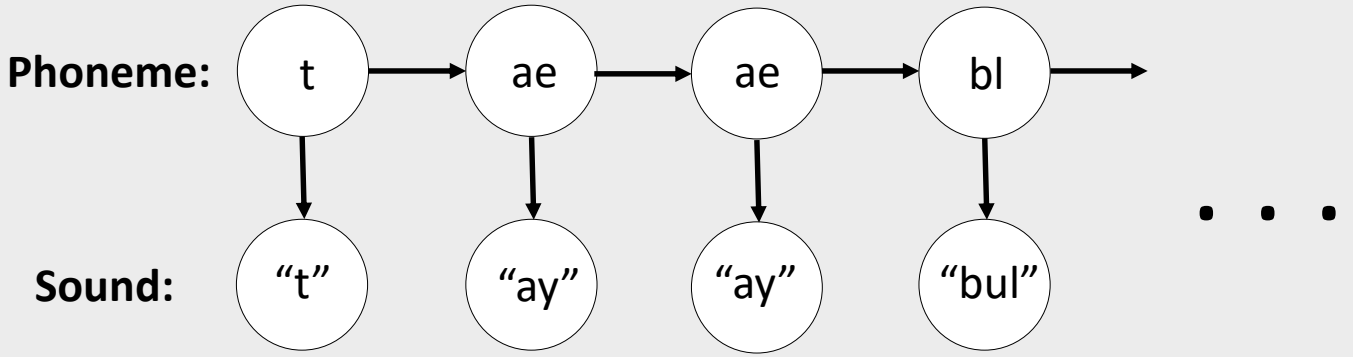
- Train a separate HMM for each word.
- Represent a word as a sequence of phonemes (hidden states)
- Represent observations as frequency vectors.

Application Example: Word Recognition (2)

A Word Model:



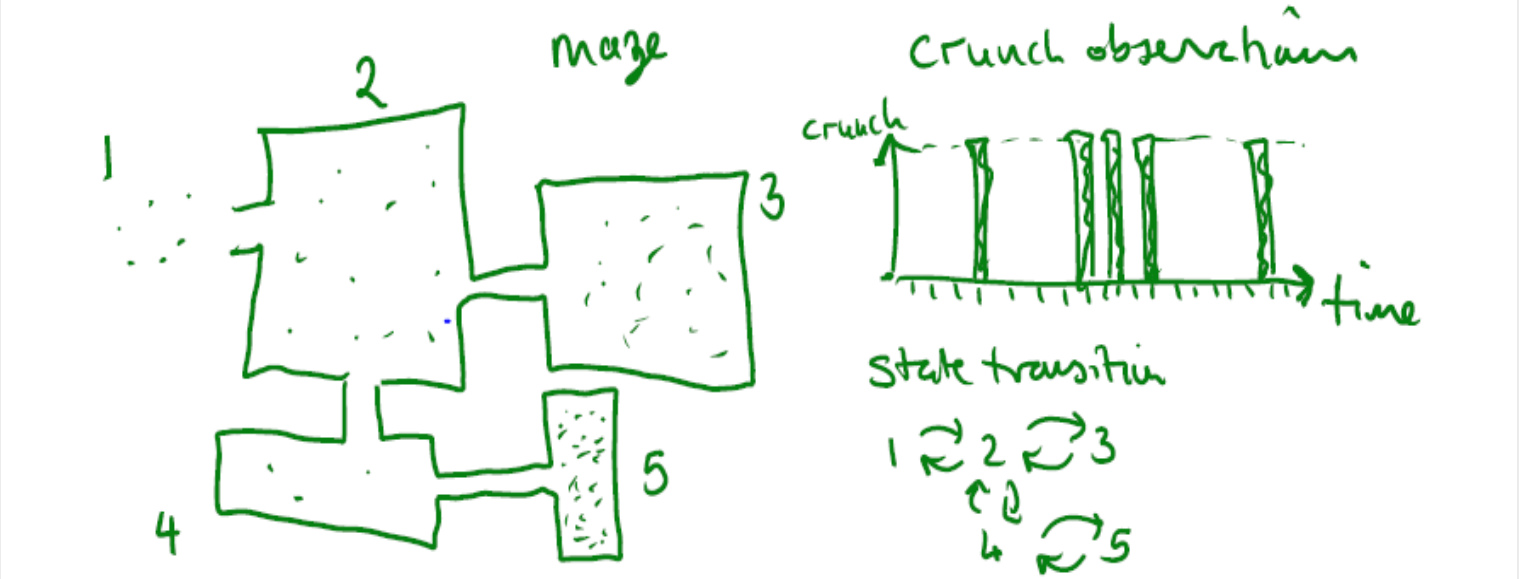
An Observation Sequence:



. . .

Application Example: Tracking

- Suppose we want to track the movement of a mouse in a set of connected rooms.
- There are microphones in each room that can detect various sounds like: “squeak”, “scurry”, “crunch”.
- The microphones don’t always pick up a sound and may detect sounds in other rooms.
- Use filtering, smoothing and prediction to estimate where the mouse is, what it is doing and where it will go next.



Credit: adapted from course notes, Avi Pfeffer; Revised by David Parkes, “CS181: Intelligent Machines Perception, Learning and Uncertainty”, 2011, Harvard University.

Application Example: Spam "Deobfuscation"

obfuscated input	deobfuscated input
u-n-s-u-s-c-r-i-b-e link	unsubscribe link
Re xe finance	refinance
M/\ cromei)i/\	macromedia
gre4t pr1ces	great prices
heyllooo it's me Chelsea	hello it's me chelsea
veerrryyy cheeaapp	very cheap
u.n sabscjbe	unsubscribe
pa, rty for sen, ding this re.port	party for sending this report
get you un iversi t y di pl0 m a	get your university dimploma
ree movee below:	remove below

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

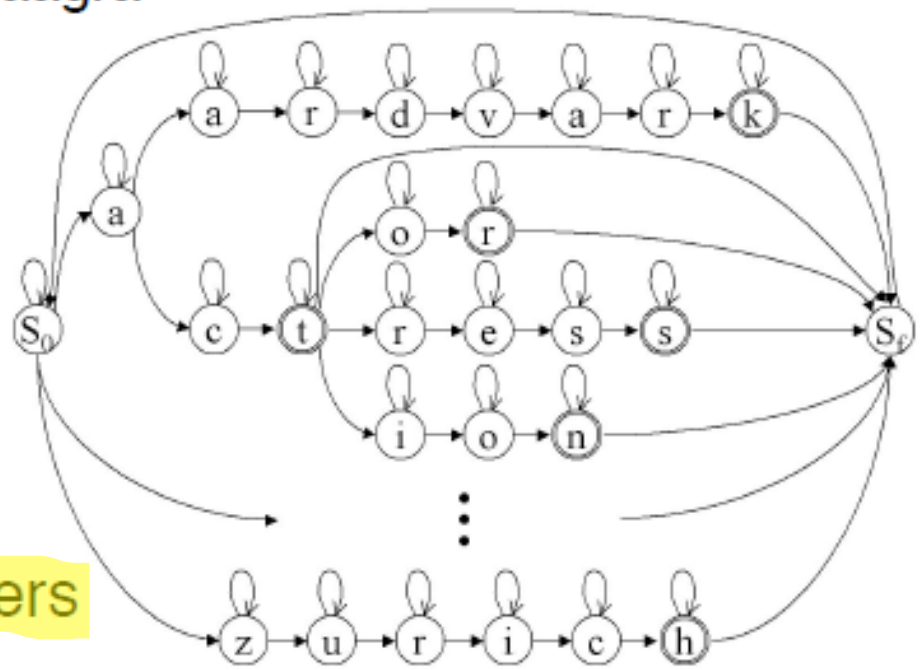
Application Example: Spam "Deobfuscation"

- Decode intentional misspellings

  - e.g. mortg3ge, viaaaagra

- Hidden states: correct letters

- Observation probabilities: similarity features between correct and observed letters



[Lee,Ng,2005]

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Training HMMs: Baum-Welch

Learning Parameters

- Input: observation sequences

$$\langle o_1^1, \ldots, o_{n^1}^1 \rangle$$

$$\langle o_1^2, \ldots, o_{n^2}^2 \rangle$$

...

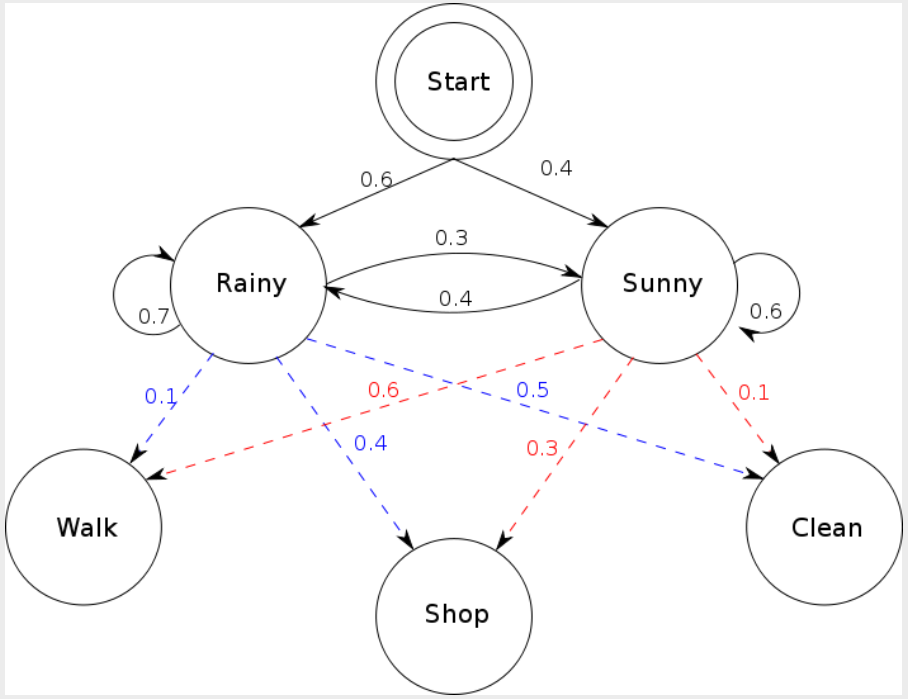
$$\langle o_1^N, \ldots, o_{n^N}^N \rangle$$

- sequences may be different lengths

- Goal: Learn HMM parameters

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

HMM Data Structures



By Terencehonles - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=8384168

```
# n = 2 hidden states
states = ('Rainy', 'Sunny')

# m = 3 observation states
observations = ('walk', 'shop', 'clean')

# n x 1 start table
start_probability = {'Rainy': 0.6, 'Sunny': 0.4}

# n x n transition table
transition_probability = { 'Rainy' : {'Rainy': 0.7, 'Sunny': 0.3},
                           'Sunny' : {'Rainy': 0.4, 'Sunny': 0.6}, }

# n x m observation table
emission_probability = { 'Rainy' : {'walk': 0.1, 'shop': 0.4, 'clean': 0.5},
                         'Sunny' : {'walk': 0.6, 'shop': 0.3, 'clean': 0.1}, }
```

Adapted from https://en.wikipedia.org/w/index.php?title=Hidden_Markov_model&oldid=841523243

Parameters

$$\theta_{h,o} = P(O_i = o \mid H_i = h)$$

Observation Table

$$\theta_{h_1,h_2} = P(H_{i+1} = h_2 \mid H_i = h_1)$$

Transition Table

$$\theta_h^1 = P(H_1 = h)$$

Start Table

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

If only …

- If we could observe hidden states, learning would be easy
  - define sufficient statistics
  - count statisics in data
  - use maximum likelihood formula for each parameter

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Sufficient Statistics

$N_{h,o}$ = number of times state h occurs with observation o

N_{h_1,h_2} = number of times state h_1 is followed by h_2

N_h^1 = number of times initial state is h

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Maximum Likelihood Estimation

$$\theta_{h,o} = P(O_i = o \mid H_i = h) \qquad \theta_{h,o} = \frac{N_{h,o}}{\sum_{o' \in O} N_{h,o'}}$$

$$\theta_{h_1,h_2} = P(H_{i+1} = h_2 \mid H_i = h_1) \qquad \theta_{h_1,h_2} = \frac{N_{h_1,h_2}}{\sum_{h \in H} N_{h_1,h}}$$

$$\theta^1_h = P(H_1 = h) \qquad \theta^1_h = \frac{N^1_h}{N}$$

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Unfortunately . . .

- We don't observe the hidden states
- So we cannot count the statistics
- What can we do?

- Expectation-maximization!

- EM in HMMs is called the **Baum-Welch** algorithm

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

EM

1. Pick initial parameters θ
2. Repeat until convergence:
   a) E-step: compute expected sufficient statistics given θ and the data
   b) M-step: choose θ so as to maximize likelihood of expected sufficient statistics

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Expectation Step

$$E\left[N_h^1\right] = \sum_{j=1}^{N} P(H_1^j = h \mid o_1^j, \ldots, o_{n^j}^j)$$

$$E\left[N_{h_1,h_2}\right] = \sum_{j=1}^{N} \sum_{i=1}^{n^j-1} P(H_i^j = h_1, H_{i+1}^j = h_2 \mid o_1^j, \ldots, o_{n^j}^j)$$

$$E\left[N_{h,o}\right] = \sum_{j=1}^{N} \sum_{i=1}^{n^j} \begin{cases} 0 & \text{if } o_i^j \neq o \\ P(H_i^j = h \mid o_1^j, \ldots, o_{n^j}^j) & \text{if } o_i^j = o \end{cases}$$

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Maximization Step

Just replace sufficient statistics with expected sufficient statistics

$$\theta_{h,o} = \frac{E\left[N_{h,o}\right]}{\sum_{o' \in O} E\left[N_{h,o'}\right]}$$

$$\theta_{h_1,h_2} = \frac{E\left[N_{h_1,h_2}\right]}{\sum_{h \in H} E\left[N_{h_1,h}\right]}$$

$$\theta_h^1 = \frac{E\left[N_h^1\right]}{N}$$

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Cost

- m hidden states
- N sequences
- n max length per sequence
- $O(m^2Nn)$ per iteration
- But how many iterations are needed?
  - Unknown

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

How Many Hidden States?

- Do you want more or fewer hidden states?
  - Which might underfit the data?
  - Which might overfit the data?

- More hidden states:
  - more complex models
  - captures more pattern
  - may overfit

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Amount of Data Needed

- m hidden states
- Size of observation model: $O(m)$
- Size of transition model: $O(m^2)$
- Size of initial model: $O(m)$
- Overall number of parameters: $O(m^2)$

- You need roughly quadratic data to learn a model with more hidden states

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Training Time

- Training time per iteration: quadratic in number of hidden states
- Number of iterations required until convergence may also grow
- Overall cost of computation grows quickly with number of hidden states

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

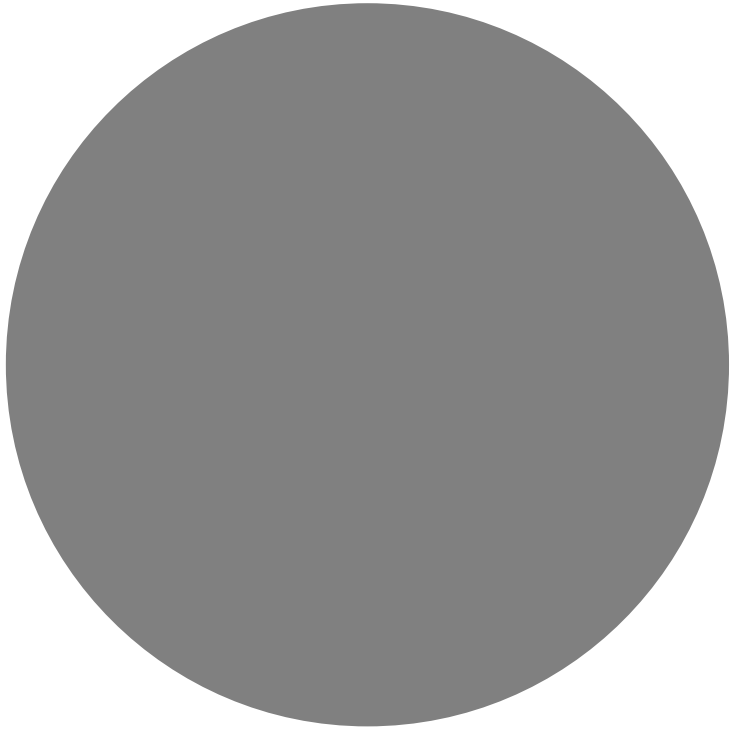
Bottom Line

- While having more hidden states is useful for capturing more pattern in the data, having a very large number of hidden states is often impractical

Credit: adapted from lecture slides Sarah Finney, “CS181: Intelligent Machines Perception, Learning and Uncertainty”, 2009, Harvard University.

HMMs vs RNNs

Look in the slide notes below for topics to consider talking about



Linear Dynamical Systems (engineers love them!)

- These are generative models. They have a real-valued hidden state that cannot be observed directly.
  - The hidden state has linear dynamics with Gaussian noise and produces the observations using a linear model with Gaussian noise.
  - There may also be driving inputs.
- To predict the next output (so that we can shoot down the missile) we need to infer the hidden state.
  - A linearly transformed Gaussian is a Gaussian. So the distribution over the hidden state given the data so far is Gaussian. It can be computed using "Kalman filtering".

Credit: Geoffrey Hinton, *https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf*

Hidden Markov Models (computer scientists love them!

- Hidden Markov Models have a discrete one-of-N hidden state. Transitions between states are stochastic and controlled by a transition matrix. The outputs produced by a state are stochastic.
  - We cannot be sure which state produced a given output. So the state is "hidden".
  - It is easy to represent a probability distribution across N states with N numbers.
- To predict the next output we need to infer the probability distribution over hidden states.
  - HMMs have efficient algorithms for inference and learning.

Credit: Geoffrey Hinton, *https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf*

A fundamental limitation of HMMs

- Consider what happens when a hidden Markov model generates data.
  - At each time step it must select one of its hidden states. So with N hidden states it can only remember log(N) bits about what it generated so far.
- Consider the information that the first half of an utterance contains about the second half:
  - The syntax needs to fit (e.g. number and tense agreement).
  - The semantics needs to fit. The intonation needs to fit.
  - The accent, rate, volume, and vocal tract characteristics must all fit.
- All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half. 2^100 is big!

Credit: Geoffrey Hinton, *https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf*

A fundamental limitation of HMMs

- Consider what happens when a hidden Markov model generates data.
  - At each time step it must select one of its hidden states. So with N hidden states it can only remember log(N) bits about what it generated so far.
- Consider the information that the first half of an utterance contains about the second half:
  - The syntax needs to fit (e.g. number and tense agreement).
  - The semantics needs to fit. The intonation needs to fit.
  - The accent, rate, volume, and vocal tract characteristics must all fit.
- All these aspects combined could be 100 bits of information that the first half of an utterance needs to convey to the second half. 2^100 is big!

Credit: Geoffrey Hinton, *https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf*

Recurrent neural networks

- RNNs are very powerful, because they combine two properties:
  - Distributed hidden state that allows them to store a lot of information about the past efficiently.
  - Non-linear dynamics that allows them to update their hidden state in complicated ways.
- With enough neurons and time, RNNs can compute anything that can be computed by your computer.

Credit: Geoffrey Hinton, *https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf*

Do generative models need to be stochastic?

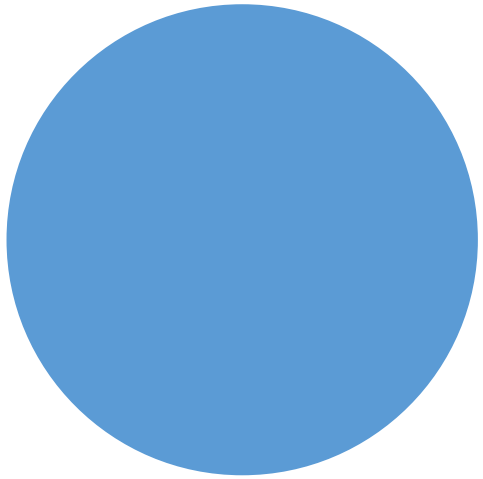
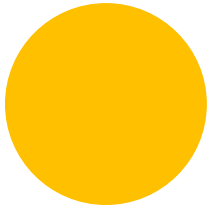
- Linear dynamical systems and hidden Markov models are stochastic models.
  - But the posterior probability distribution over their hidden states given the observed data so far is a deterministic function of the data.
- Recurrent neural networks are deterministic.
  - So think of the hidden state of an RNN as the equivalent of the deterministic probability distribution over hidden states in a linear dynamical system or hidden Markov model.

Credit: Geoffrey Hinton, *https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf*

Recurrent neural networks

- What kinds of behaviour can RNNs exhibit?
  - They can oscillate. Good for motor control?
  - They can settle to point attractors. Good for retrieving memories?
  - They can behave chaotically. Bad for information processing?
  - RNNs could potentially learn to implement lots of small programs that each capture a nugget of knowledge and run in parallel, interacting to produce very complicated effects.
- But the computational power of RNNs makes them very hard to train.
  - For many years we could not exploit the computational power of RNNs despite some heroic efforts (e.g. Tony Robinson's speech recognizer).

Credit: Geoffrey Hinton, *https://www.cs.toronto.edu/~hinton/csc2535/notes/lec10new.pdf*

Hidden Markov Models

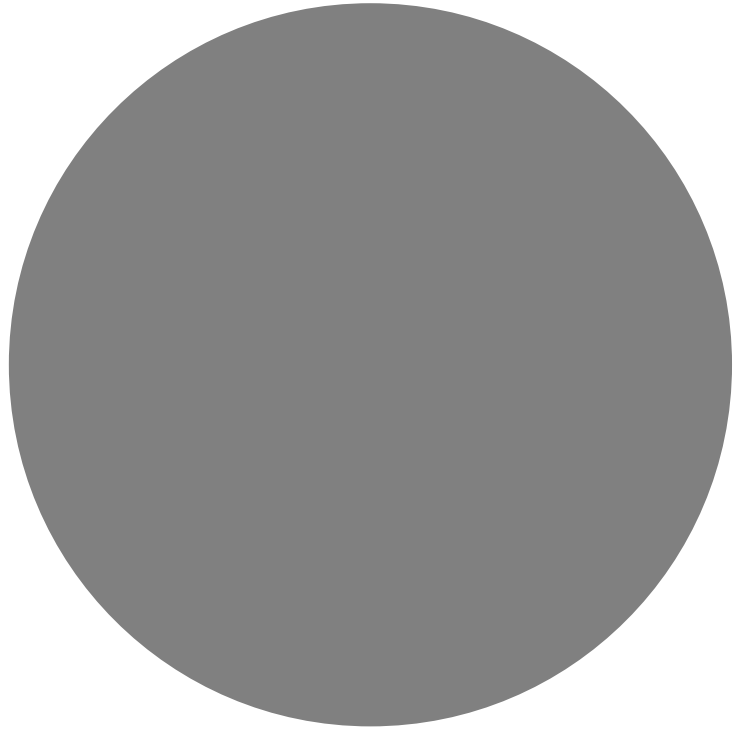
Scott O'Hara

Metrowest Developers
Machine Learning Group

05/23/2018

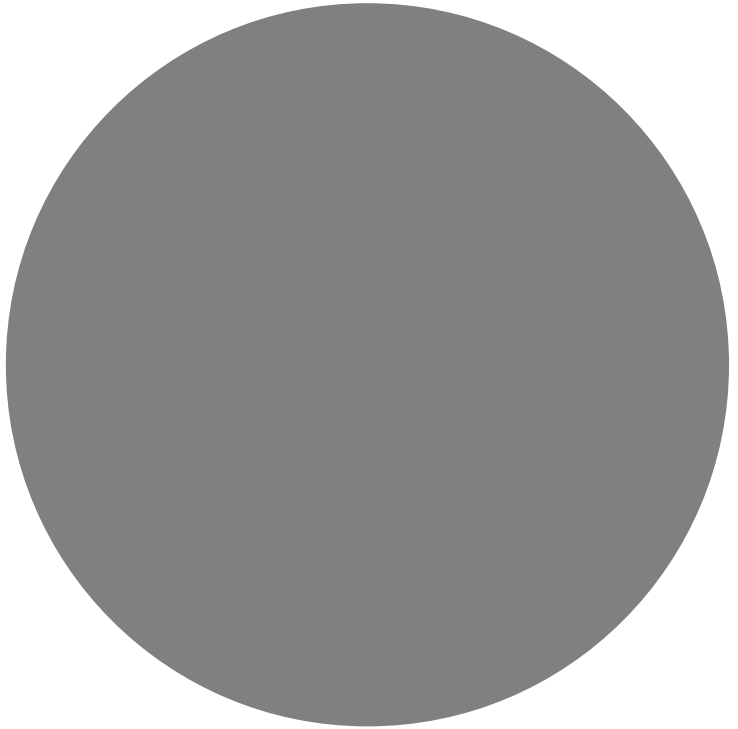
Extra Slides

Look in the slide notes below for topics to consider talking about



Forward-Backward Algorithm

Given an HMM, compute the probability of an observation sequence.



Compute the probability of a sequence of observations

Given: $o_1, \ldots, o_n$

Goal: compute probability $P(o_1, \ldots, o_n)$

Two Algorithms:

- Backward Algorithm
- Forward Algorithm
- Together called the forward-backward algorithm

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Approach 1: Enumerate All Possibilities

Introduce $h_1, \dots, h_n$ with marginalization:

$$P(o_1, \dots, o_n) = \sum_{h_1,\dots,h_n} P(h_1, \dots, h_n, o_1, \dots, o_n)$$

Enumerate all possible configurations of $h_1, \dots, h_n$ and sum the results where:

$$P(h_1, \dots, h_n, o_1, \dots, o_n) = P(o_1|h_1)P(h_1)\prod_{i=2}^{n} P(h_i|h_{i-1})P(o_i|h_i)$$

Problem: the total number of sequences is exponential in the length of sequence.

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

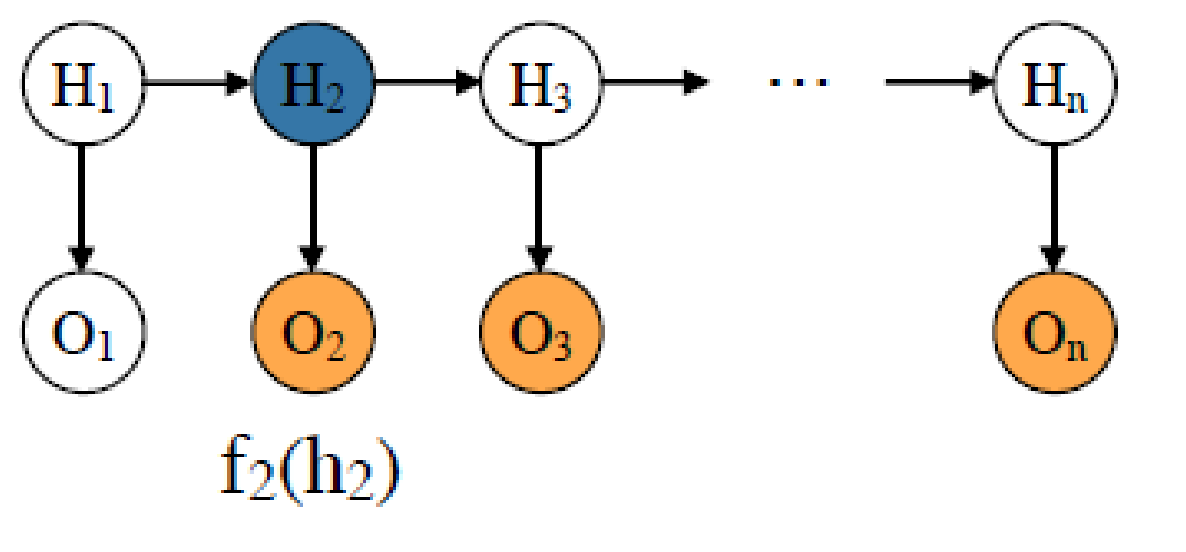
Approach 2: Dynamic Programming

- Build the solution from the bottom up by combining the solution of smaller problems into the solution of larger problems.
- **Backward Algorithm:** compute from the end of the sequence toward the beginning.
- **Forward Algorithm:** computer from the beginning of the sequence toward the end.
- Neither approach has any particular advantage.

Credit: adapted from lecture slides Sarah Finney, “CS181: Intelligent Machines Perception, Learning and Uncertainty”, 2009, Harvard University.

Backward Algorithm

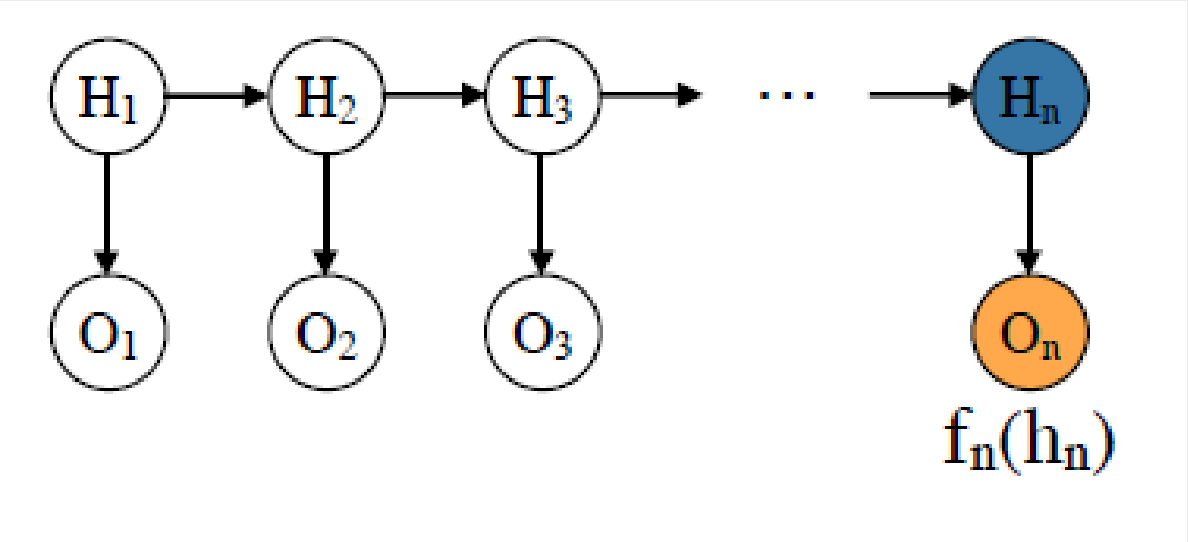
- Subproblem at time i: probability of suffix from i starting from each possible hidden state
- Define $f_i(h_i) = P(o_1, \ldots, o_n | h_i)$



Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Base Case

- $f_n(h_n) = P(o_n|h_n)$, from the observation model.



Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Backward Inductive Step

$$f_i(h_i) = P(o_i, \ldots, o_n | h_i)$$

$$= P(o_i|h_i)P(o_{i+1}, \ldots, o_n|h_i, o_i) \quad \textbf{chain rule}$$

$$= P(o_i|h_i)P(o_{i+1}, \ldots, o_n|h_i\ ) \quad \textbf{Markov property}$$

$$= P(o_i|h_i)\sum_{h_{i+1} \in H} P(h_{i+1}, o_{i+1}, \ldots, o_n|h_i\ ) \quad \textbf{marginalization}$$

$$= P(o_i|h_i)\sum_{h_{i+1} \in H} P(h_{i+1}|h_i\ )P(o_{i+1}, \ldots, o_n|h_i, h_{i+1}) \quad \textbf{chain rule}$$

$$= P(o_i|h_i)\sum_{h_{i+1} \in H} P(h_{i+1}|h_i\ )P(o_{i+1}, \ldots, o_n|h_{i+1}) \quad \textbf{Markov property}$$

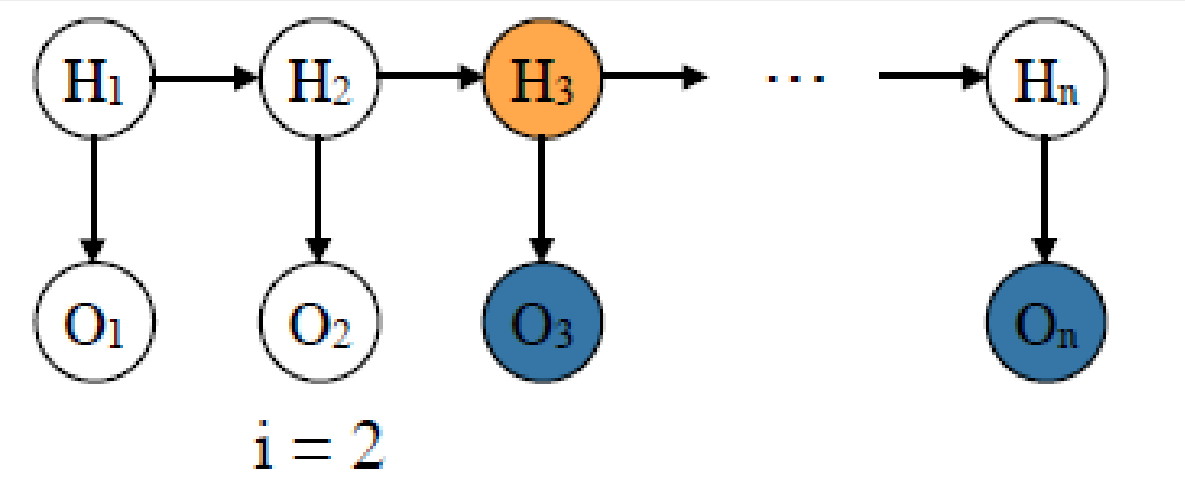
$$= P(o_i|h_i)\sum_{h_{i+1} \in H} P(h_{i+1}|h_i\ )f_{i+1}(h_{i+1}) \quad \textbf{definition of } \boldsymbol{f}$$

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Inductive Step (1)

$$f_i(h_i) = P(o_i|h_i) \sum_{h_{i+1} \in H} P(h_{i+1}|h_i)\, \boldsymbol{f_{i+1}(h_{i+1})}$$

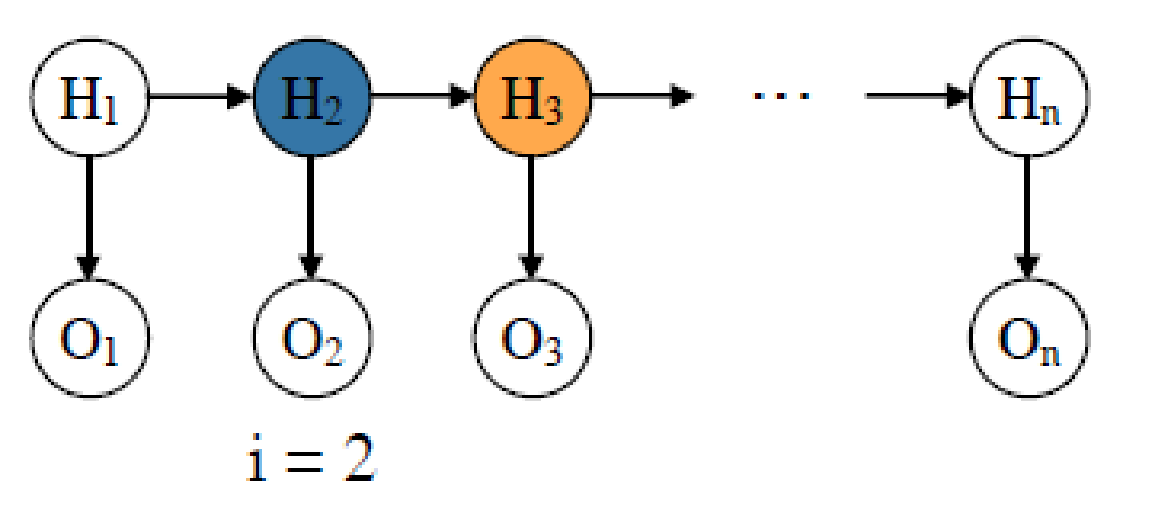
Previously Computed



Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Inductive Step (2)

$$f_i(h_i) = P(o_i|h_i) \sum_{h_{i+1} \in H} P(h_{i+1}|h_i) f_{i+1}(h_{i+1})$$



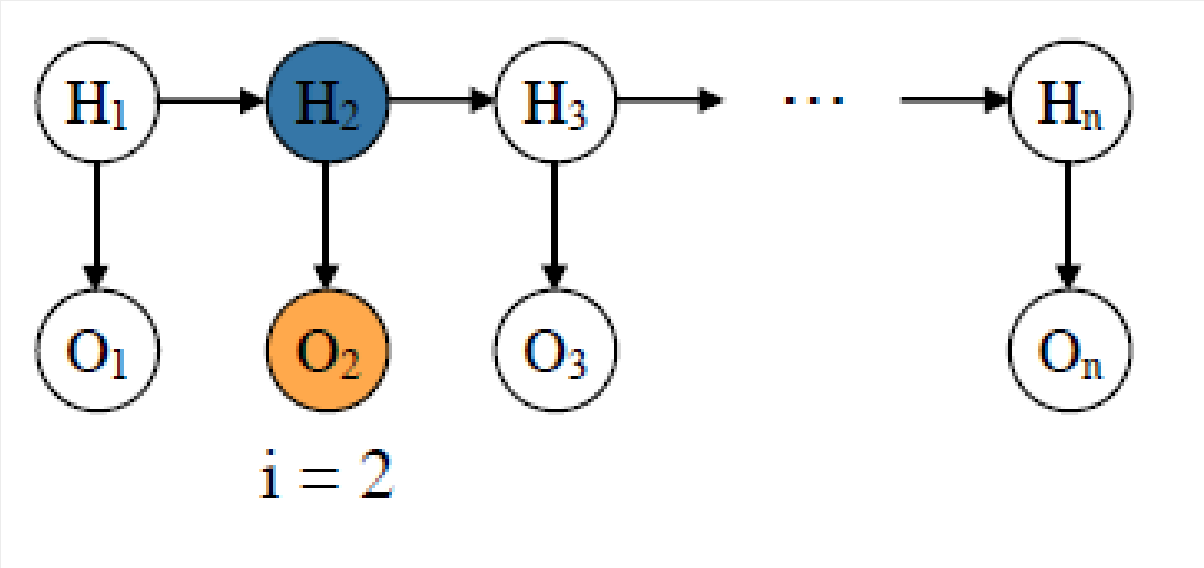
from transition model

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Inductive Step (3)

$$f_i(h_i) = P(o_i|h_i) \sum_{h_{i+1} \in H} P(h_{i+1}|h_i)\, \boldsymbol{f_{i+1}(h_{i+1})}$$

from observation model

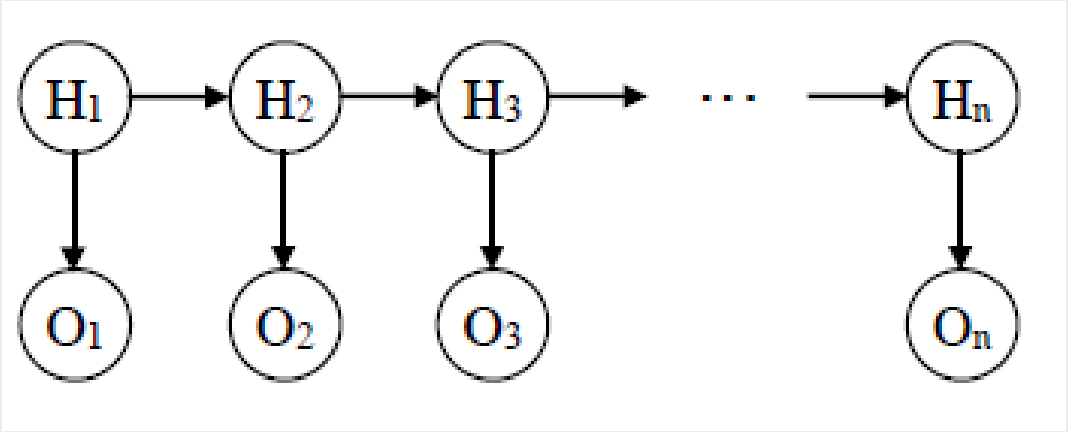


Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Final Step (1)

$$P(o_1, \ldots, o_n) = \sum_{h_1 \in H} P(o_1|h_1) P(o_2, \ldots, o_n|h_1)$$

$$= \sum_{h_1 \in H} P(o_1|h_1) f_1(h_1)$$



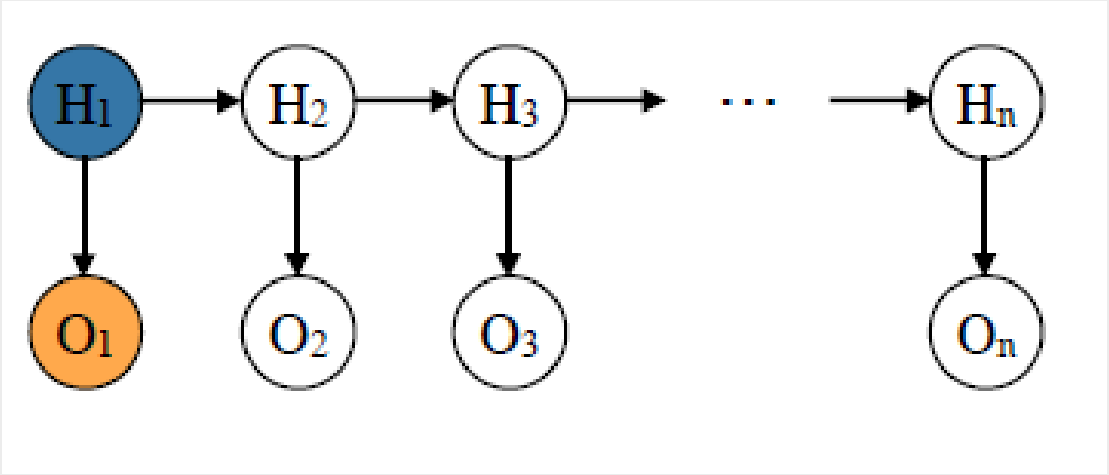
Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Final Step (2)

$$P(o_1, \ldots, o_n) = \sum_{h_1 \in H} P(o_1|h_1) P(o_2, \ldots, o_n|h_1)$$

$$= \sum_{h_1 \in H} P(o_1|h_1) f_1(h_1)$$

observation model



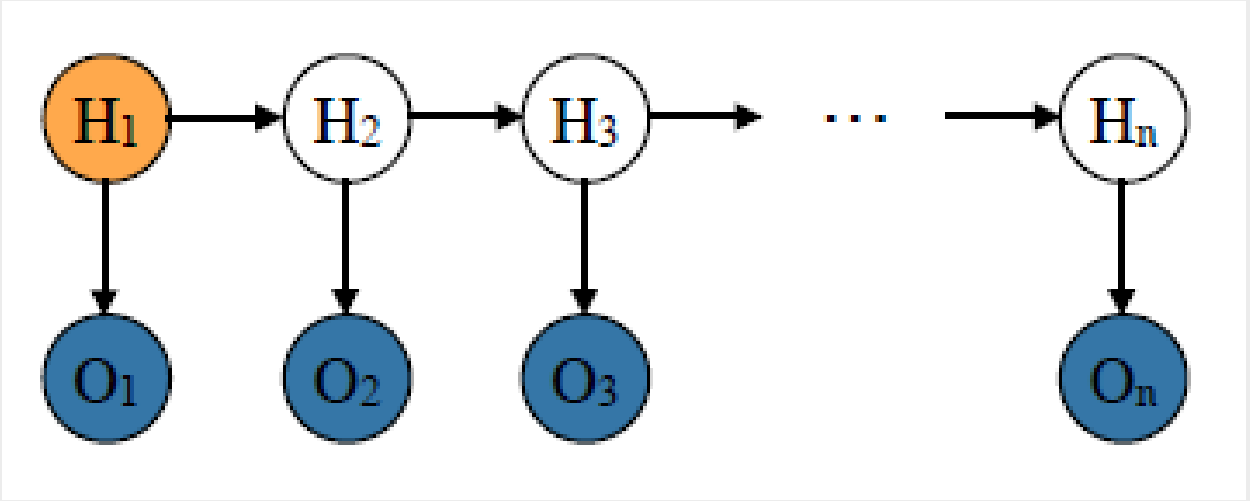
Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Final Step (3)

$$P(o_1, \ldots, o_n) = \sum_{h_1 \in H} P(o_1|h_1) P(o_2, \ldots, o_n|h_1)$$

$$= \sum_{h_1 \in H} P(o_1|h_1) f_1(h_1)$$

previously computed



Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Complexity: m hidden states, n time steps

Init

For each hidden state h_n: **O(m)**

$$f_n(h_n) = P(o_n \mid h_n)$$

Loop

For $i = n\text{-}1$ down to 1: **O(n) times**

For each hidden state h_i: **O(m^2)**

$$f_i(h_i) = P(o_i \mid h_i) \sum_{h(i+1)} P(h_{i+1} \mid h_i) f_{i+1}(h_{i+1})$$

Finish

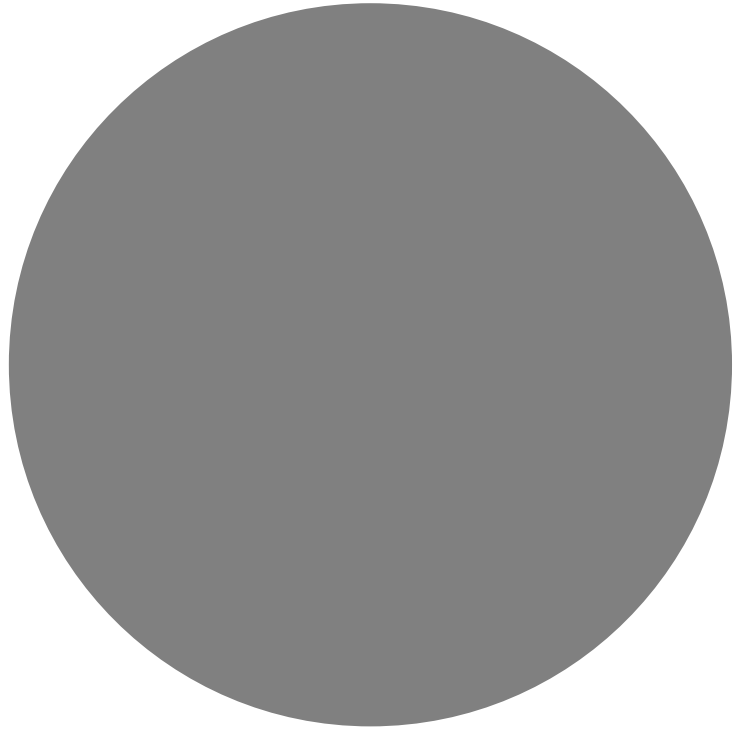
Return $\sum_{h1} P(h_1) f_1(h_1)$ **O(m)**

Total: O(m^2n)

Credit: adapted from lecture slides Sarah Finney, "CS181: Intelligent Machines Perception, Learning and Uncertainty", 2009, Harvard University.

Template

Stuff



Template