COMP 424 - Artificial Intelligence
Lecture 19: Temporal Inference 2

Instructor: Jackie CK Cheung (jcheung@cs.mcgill.ca)

Readings: R&N Ch 15.3-15.4

Plan for today

- Review and warm-up
- Sequence models with continuous state spaces
- Applications of sequence models: speech recognition and language modelling

Review of last class

Match the objective with the term:

1. $P(X_k \mid e_{1:t})$ for $0 \le k < t$	*Smoothing*
2. $P(X_t \mid e_{1:t})$	*Prediction*
3. $argmax_{X1:t} P(X_{1:t} \mid e_{1:t})$	*Filtering*
4. $P(X_{t+k} \mid e_{1:t})$ for $k > 0$	*Most likely explanation*

Review of algorithms

Match the algorithm with its purpose:

Backward algorithm	$\text{argmax}_{\mathbf{X}}\ P(\boldsymbol{X}, \boldsymbol{E}\|\theta)$
Viterbi algorithm	$P(\boldsymbol{E}\|\theta)$
Baum-Welch algorithm	$\text{argmax}_{\theta}\ P(\boldsymbol{E}\|\theta)$
Forward algorithm	

Continuous state spaces

- HMMs from last lecture had **discrete** state spaces
- But we often also want **continuous** state spaces:
  - Positions: Airplanes, robots, …
  - Population dynamics: ecosystems, …
  - Other quantities: economies, chemical plants, planets, …
- Algorithm: **Kalman filtering**

Filtering problem

$P(X_t | e_{1:t})$

- E.g., we are designing a spacecraft, that needs to keep track of its position and velocity.
- Given sensor information (possible from multiple sources) over time

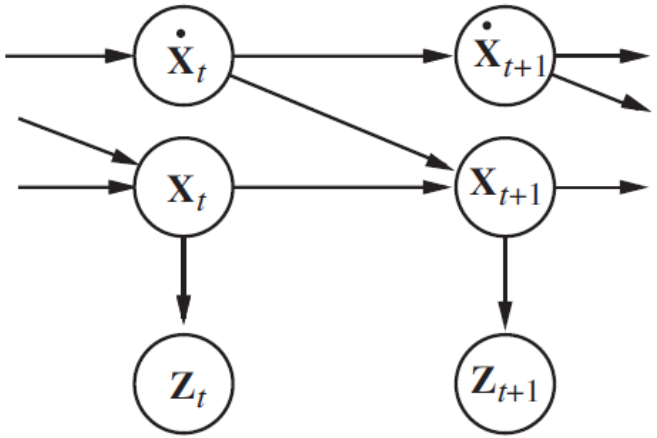
- Called filtering because there is noise in the sensor data, and we need to filter that out to determine the most probable state of the spacecraft.
  - There are usually multiple sensor inputs -> **sensor fusion**
  - **Kalman filters** were used in the Apollo navigation computer!

Kalman filtering

- Another example: suppose we are tracking a bird flying. Define a set of continuous variables:

$\boldsymbol{X}_t = \{ X_t, Y_t, Z_t,$ *Position at time t*

$X_t', Y_t', Z_t' \}$ *Velocity at time t*



- Assume: Gaussian prior distribution, Gaussian transition model, Gaussian sensor model, Gaussian posterior distribution.

Linear Gaussian transition

- Kalman filtering assumes the following transition model:

$$P(X_{t+\Delta} = x_{t+\Delta} | X_t = x_t, X'_t = x'_t) = N(x_t + x'_t\Delta, \sigma^2)(x_{t+\Delta})$$

  - Δ is the time between observations

- This means that the next position is a linear function of the current position plus some **Gaussian noise**.
  - e.g., bird is flying smoothly
  - Some deviations from expectations because of wind resistance, other factors

Updating

- Similar recursive state update equations can be defined as for HMMs:

One-step prediction:

$$P(\boldsymbol{X}_{t+1}|\boldsymbol{e}_{1:t}) = \int_{\boldsymbol{x}_t} P(\boldsymbol{X}_{t+1}|\boldsymbol{x}_t)P(x_t|\boldsymbol{e}_{1:t})d\boldsymbol{x}_t$$

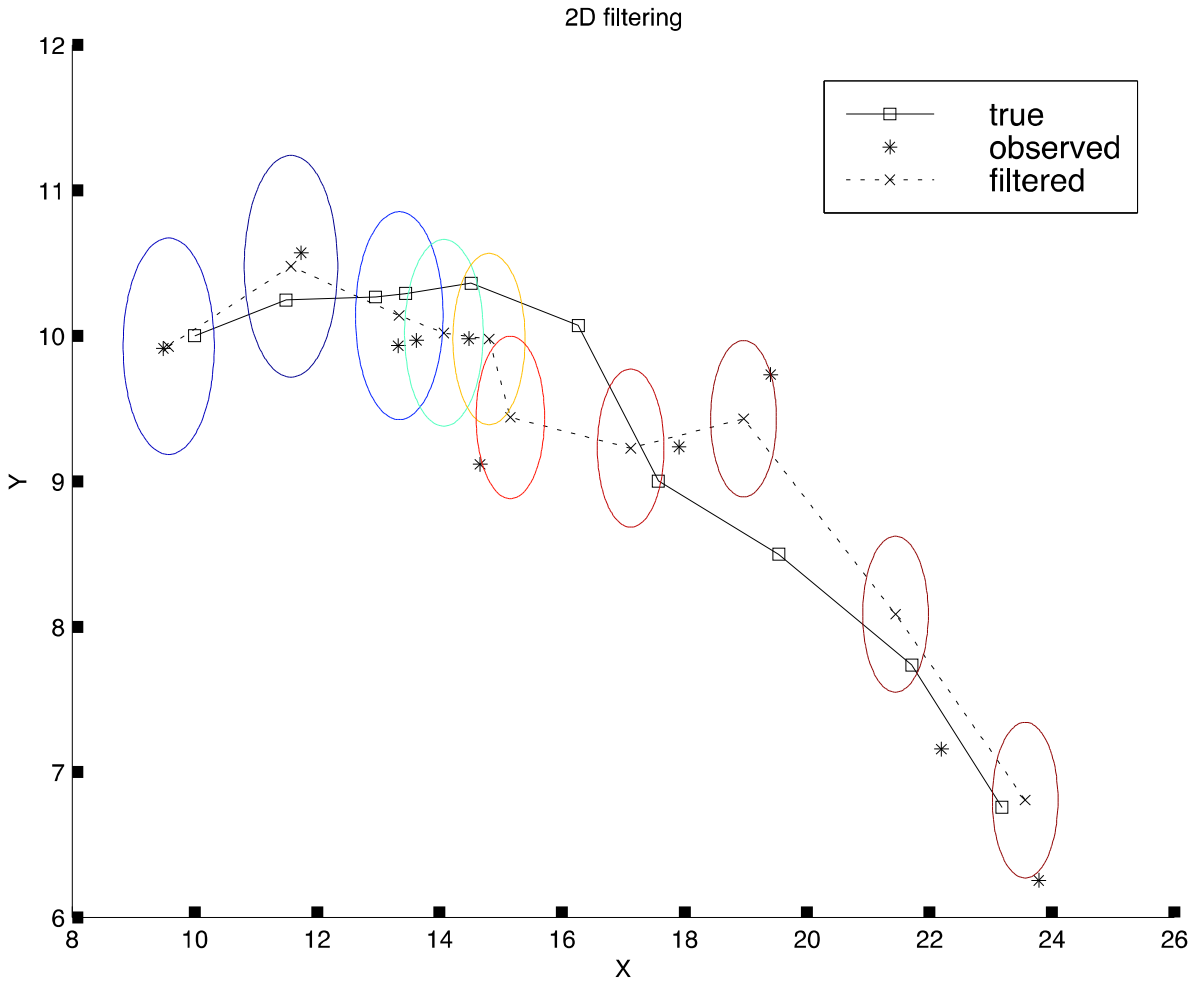
Important: The new distribution is also Gaussian.

Filtering:

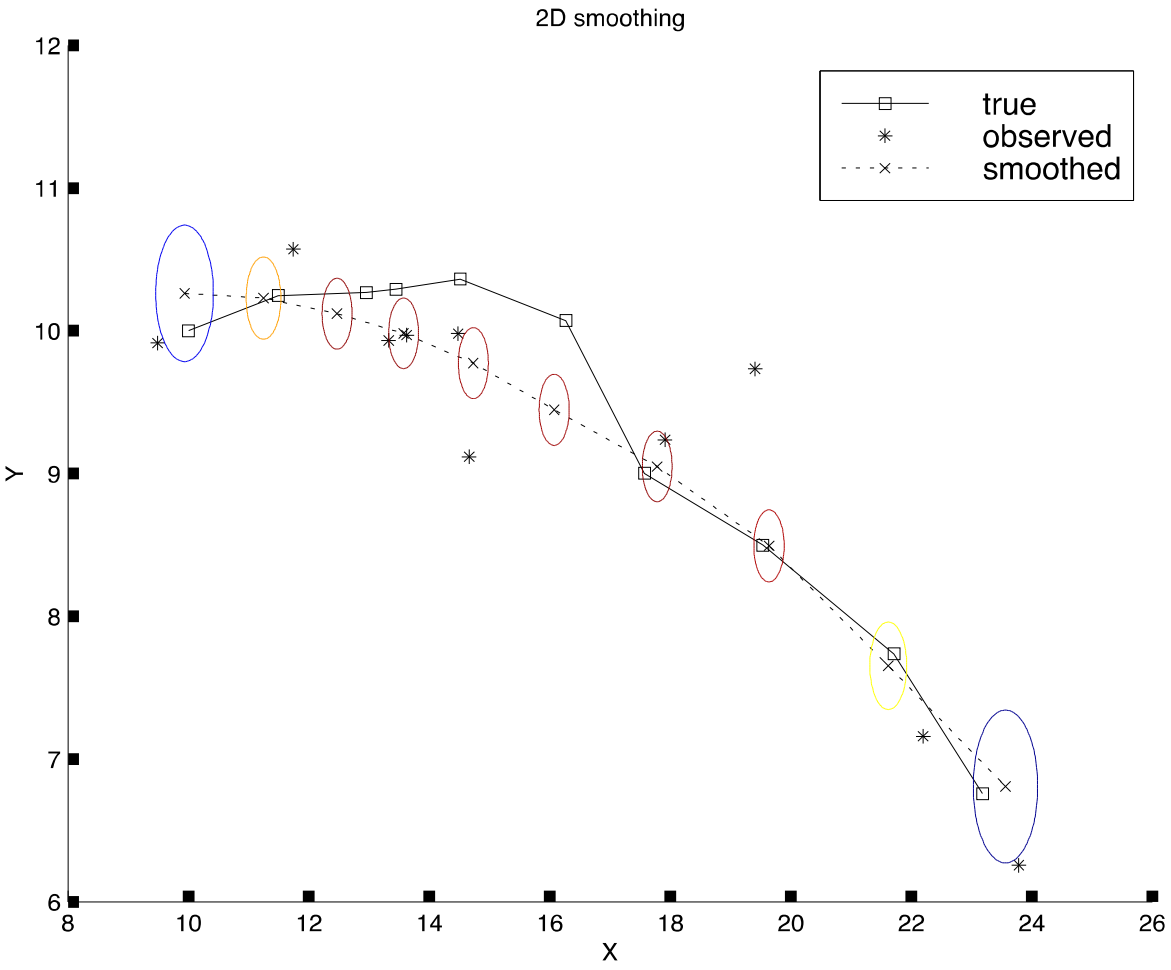
$$P(\boldsymbol{X}_{t+1}|\boldsymbol{e}_{1:t+1}) = \alpha P(\boldsymbol{e}_{t+1}|\boldsymbol{X}_{t+1})P(\boldsymbol{X}_{t+1}|\boldsymbol{e}_{1:t})$$

Important: The new distribution is also Gaussian, assuming that the sensor model is linear Gaussian.

2-D tracking example: Filtering

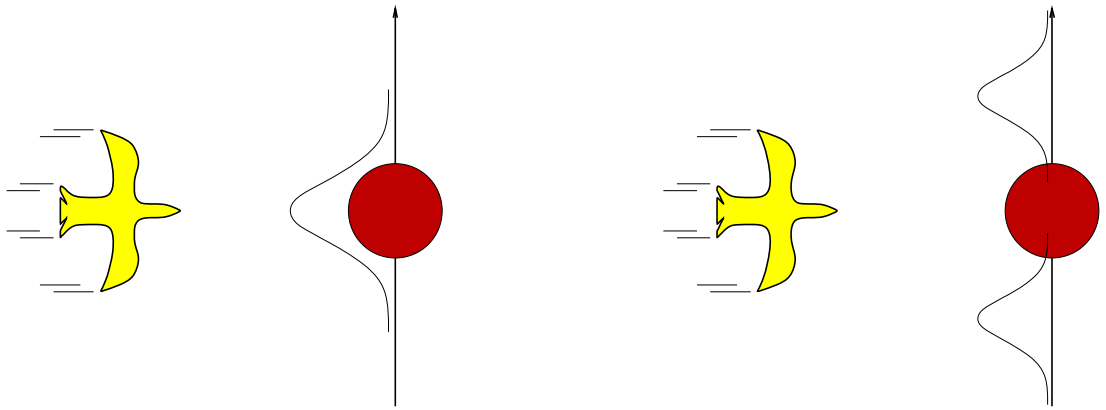


2-D tracking example: Smoothing



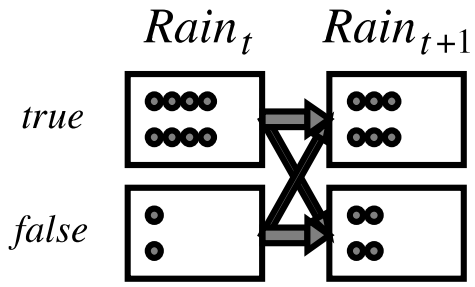
Limitations of Kalman filters?

- Cannot be applied if the **transition model is nonlinear**.
  - Extended Kalman Filter models transition as locally linear around $x_t = \mu_t$. Fails if system is locally unsmooth.
- KFs assume the **belief state stays Gaussian**. Often wrong in the real world.
  - **Particle filters** can represent arbitrary distributions of the posterior belief.



Particle filtering

- Use set of samples of possible states to approximate the belief state.
- The population of samples ("particles") tracks the high-likelihood regions of the state space.
- Replicate particles proportional to likelihood for e_t.

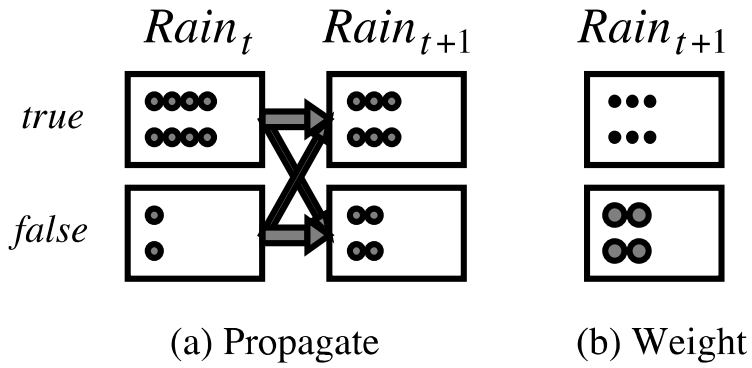


(a) Propagate

using $P(Rain_{t+1}|Rain_t)$

Particle filtering

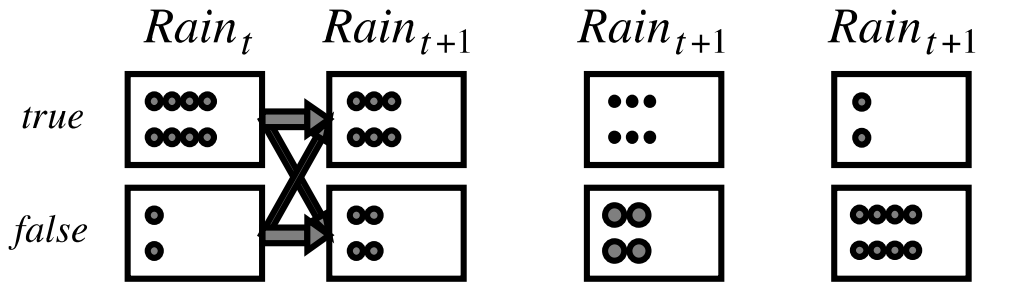
- Use set of samples of possible states to approximate the belief state.
- The population of samples ("particles") tracks the high-likelihood regions of the state space.
- Replicate particles proportional to likelihood for e_t.



using $P(E_{t+1}|Rain_{t+1})$

Particle filtering

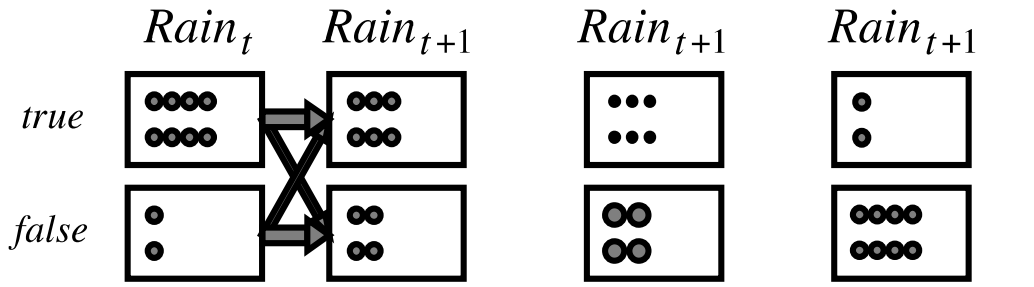
- Use set of samples of possible states to approximate the belief state.
- The population of samples ("particles") tracks the high-likelihood regions of the state space.
- Replicate particles proportional to likelihood for e_t.



using weights from prev. step

Particle filtering

- Use set of samples of possible states to approximate the belief state.
- The population of samples ("particles") tracks the high-likelihood regions of the state space.
- Replicate particles proportional to likelihood for e_t.



- Widely used for tracking nonlinear dynamic systems.
  E.g. Localization of mobile robots.

Particle filtering equations

1. Assume belief at time t: $N(x_t \mid e_{1:t}) / N = P(x_t \mid e_{1:t})$
2. Propagate forward: $N(x_{t+1} \mid e_{1:t}) = \Sigma_{x_t} P(x_{t+1} \mid x_t)\, N(x_t \mid e_{1:t})$
3. Weight samples by their likelihood for e_{t+1}:

   $W(x_{t+1} \mid e_{1:t+1}) = P(e_{t+1} \mid x_{t+1})\, N(x_{t+1} \mid e_{1:t})$
4. Resample to obtain population proportional to W:

   $N(x_{t+1} \mid e_{1:t+1}) / N = \alpha\, W(x_{t+1} \mid e_{1:t+1})$ resampling

   $= \alpha P(e_{t+1} \mid x_{t+1})\, N(x_{t+1} \mid e_{1:t})$ plug in 3

   $= \alpha P(e_{t+1} \mid x_{t+1}) \Sigma_{x_t} P(x_{t+1} \mid x_t)\, N(x_t \mid e_{1:t})$ plug in 2

   $= \alpha' P(e_{t+1} \mid x_{t+1}) \Sigma_{x_t} P(x_{t+1} \mid x_t)\, P(x_t \mid e_{1:t})$

   $= P(x_{t+1} \mid e_{1:t+1})$ plug in 1

   by definition of $P(x_{t+1} \mid e_{1:t+1})$
   and assumptions of filtering problem

Particle filtering performance

- Easy to use in continuous state/observation spaces!
- Easy to implement. Useful for many different distributions (not just Gaussian).
- To improve precision of belief tracking, increase the number of particles.
  - Time/space complexity grow linearly with the number of particles.
- Hard to track very rare events (need too many particles).

Speech Recognition

What makes it challenging?

- Noise – background, digitization artifacts.
- Ambiguities
  - same word pronounced differently by different people or even by the same person.
  - different words sound the same.
- Co-articulation – word sound changes depending on the preceding and following words.
- Intonation, accents.
- Time variations

Signal processing methods deal with some of these issues.

Speech as probabilistic inference

- Decompose into two separate problems (by Bayes rule):

$$P(\text{Words} \mid \text{Signal}) = \eta\, P(\text{Signal} \mid \text{Words})\, P(\text{Words})$$

- P(Signal | Words) = Acoustic model
  - Modeled as an HMM.
  - Words are the hidden state variables
  - Signal is the observation (evidence) variable.

- P(Words) = Language model

Choosing a set of states

At what level of detail should we model speech phenomena?

Choosing a set of states

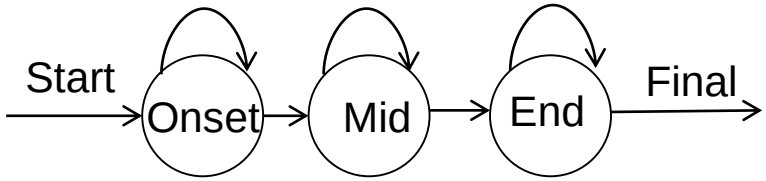
English uses around 50 **phones** (distinct speech sounds).

E.g. English phones

[iy]	beat	[b]	bet	[p]	pet
[ih]	bit	[ch]	Chet	[r]	rat
[ey]	bet	[d]	debt	[s]	set
[ao]	bought	[hh]	hat	[th]	thick
[ow]	boat	[hv]	high	[dh]	that
[er]	Bert	[l]	let	[w]	wet
[ix]	roses	[ng]	sing	[en]	button
⋮	⋮	⋮	⋮	⋮	⋮

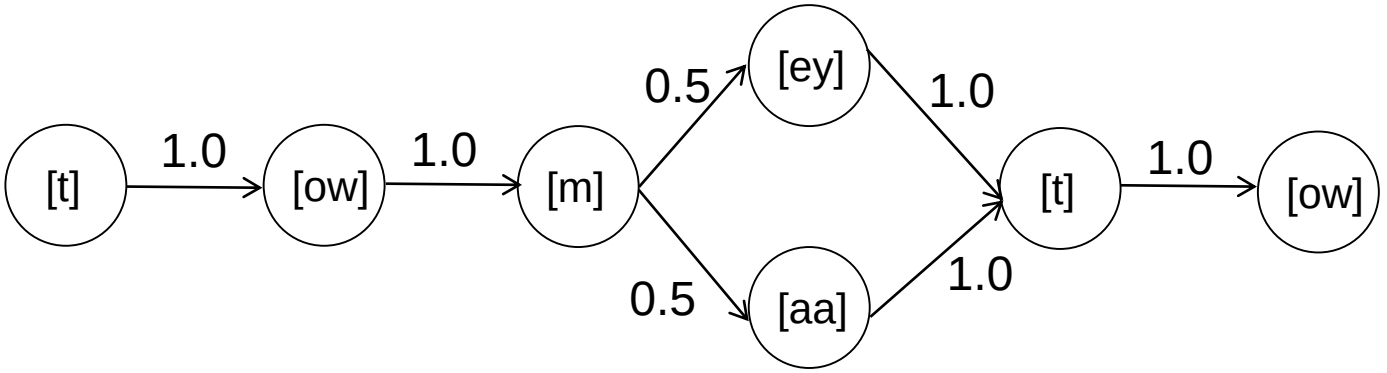
E.g., "ceiling" is [s iy l ih ng] / [s iy l ix ng] / [s iy l en]

- Phones have structure. Generally, use 3-state phones.



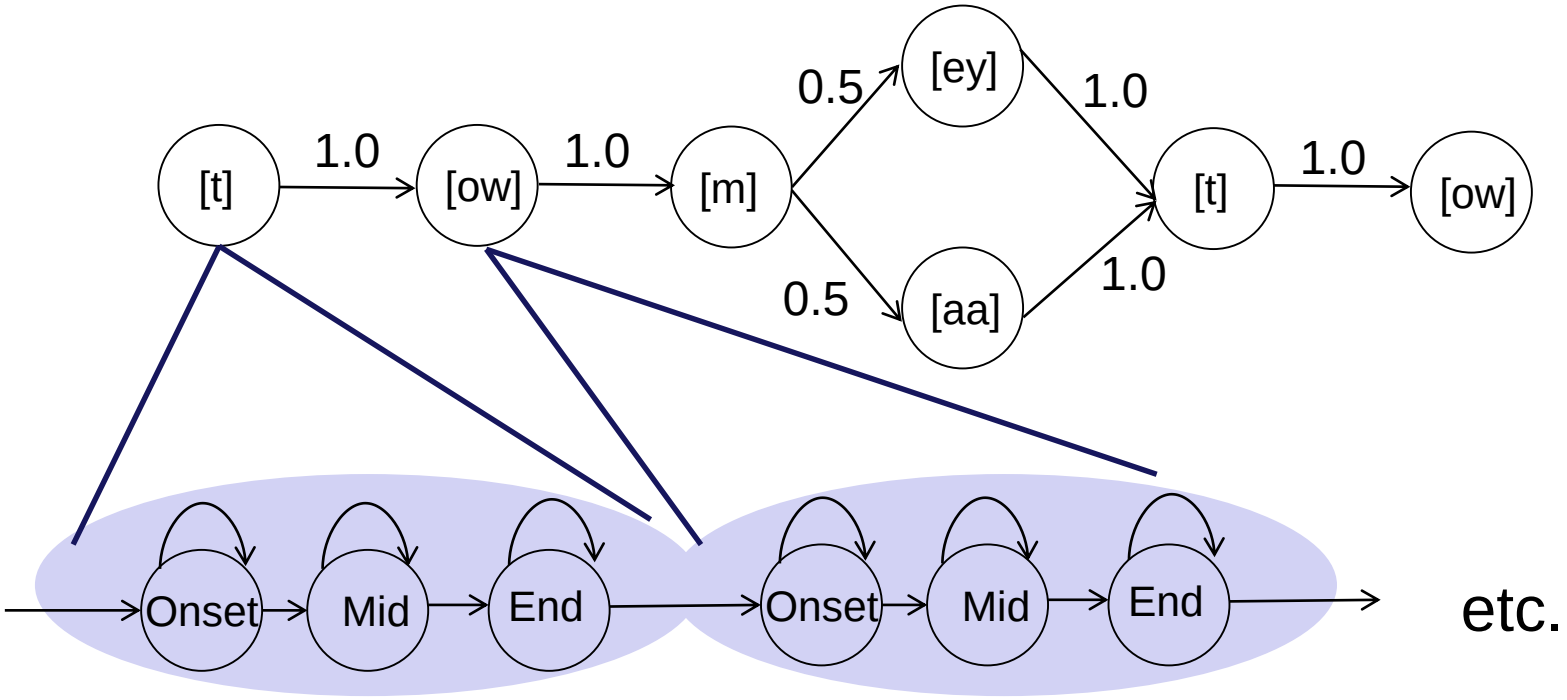
Learn probabilities using Baum-Welch.

Word Pronunciation Models



Pronunciation model for “tomato”

Word Model with states



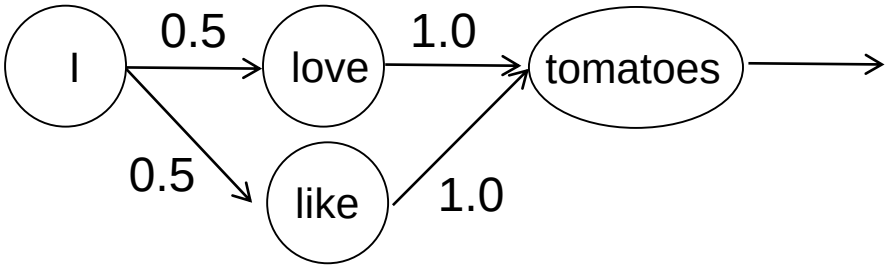
A word with *p* phones would have 3*p* states

HMM for Speech Recognition

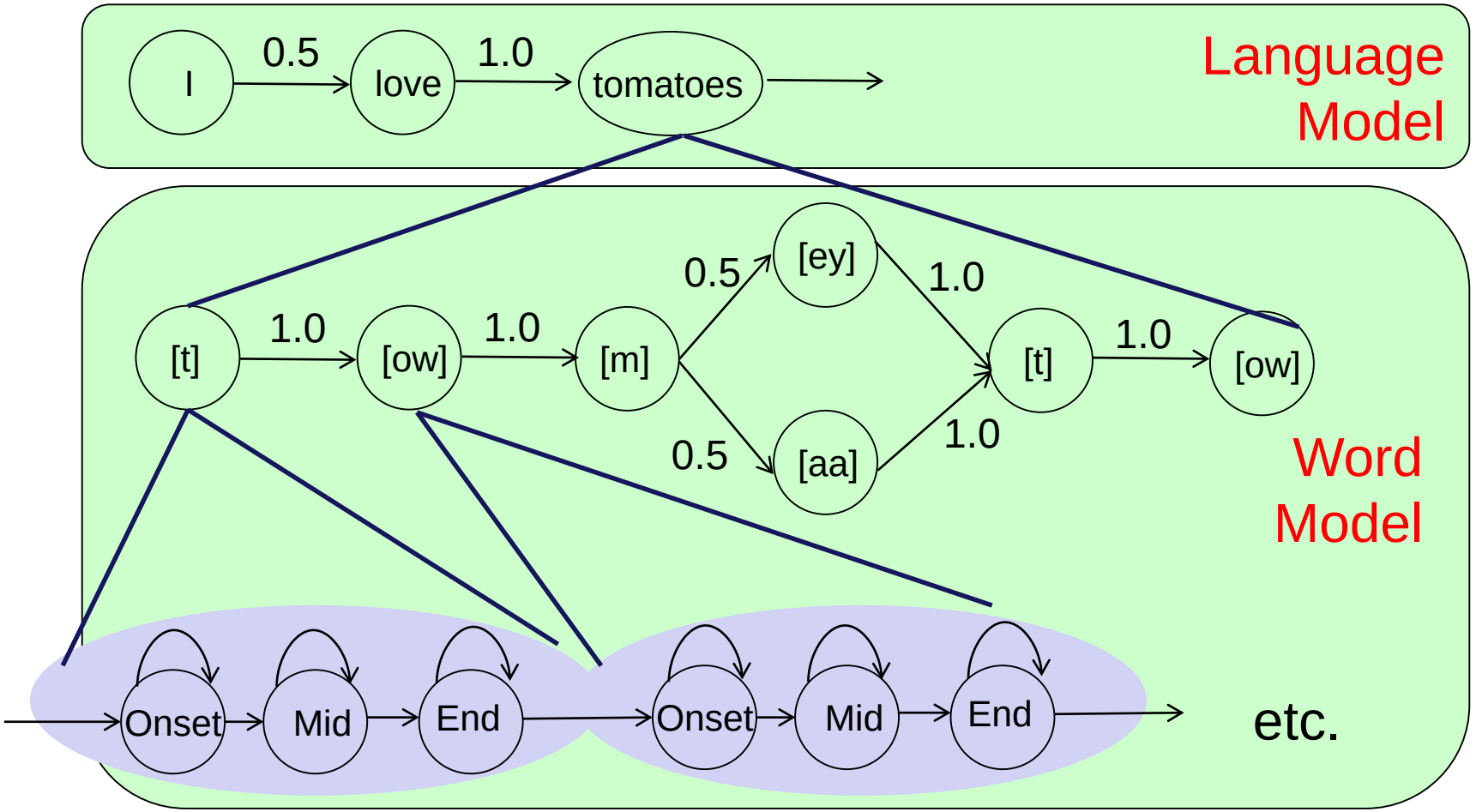
- Training phase for Word model:
  - Start with initial setting and train with continuous speech signals and Baum-Welch algorithm.
    - Observation data corresponding to the same phone (in different words) are pooled together: parameter-tying.

- Recognition phase:
  - Use Viterbi Algorithm to find the most likely sequence of states -> most likely sequence of words.

Language Model



Language Model



Language Model

How to model sequences of words?

How to estimate the probability of sentences?

Useful in Natural Language Processing tasks:

- Speech recognition, machine translation, natural language generation, spelling correction.

Slide from Ani Nenkova

The chain rule

$$P(w_1^n) = P(w_1)P(w_2|w_1)P(w_3|w_1^2)\ldots P(w_n|w_1^{n-1})$$
$$= \prod_{k=1}^{n} P(w_k|w_1^{k-1})$$

P(its water was so transparent)=

P(its) x

P(water|its) x

P(was|its water) x

P(so|its water was) x

P(transparent|its water was so)

Slide from Ani Nenkova

MLE estimate of probabilities

Unigram:

$$P(w_i) = \frac{count(w_i)}{N}$$

Bigram:

$$P(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_{i-1})}$$

Slide from Ani Nenkova

Berkeley Restaurant Project Sentences

- *can you tell me about any good cantonese restaurants close by*
- *mid priced thai food is what i'm looking for*
- *tell me about chez panisse*
- *can you give me a listing of the kinds of food that are available*
- *i'm looking for a good place to eat breakfast*
- *when is caffe venezia open during the day*

Slide from Ani Nenkova

Bigram Counts

- Out of 9222 sentences
  - Eg. “I want” occurred 827 times

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Slide from Ani Nenkova

Bigram Probabilities

- Divide bigram counts by ***prefix unigram counts*** to get probabilities.

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Slide from Ani Nenkova

Bigram Estimates of Sentence Probabilities

- P(I want english food) =
  P(i) x
  P(want|I) x
  P(english|want) x
  P(food|english)
  =.000031

Slide from Ani Nenkova

Laplace Smoothing

- What to do about never seen two-word combinations? – Smoothing!
  - Also called add-one smoothing

** Not the same kind of smoothing as with temporal inference. **

- Just add one to all the counts!
- MLE estimate:
- Laplace estimate:

$$P(w_i) = \frac{c_i}{N}$$

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V}$$

Slide from Ani Nenkova

Laplace-Smoothed Bigram Counts

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Slide from Ani Nenkova

Laplace-Smoothed Bigram Probabilities

$$P^*(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V}$$

	i	want	to	eat	chinese	food	lunch	spend
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Slide from Ani Nenkova

Summary

- Different models of temporal models:
  - HMMs, Dynamic Bayesian networks, Kalman filter, Particle filter
- Inference tasks:
  - Filtering
  - Prediction
  - Smoothing: Forward-Backward Algorithm
  - Most likely explanation: Viterbi algorithm
- Application to speech recognition
- Language modelling