NLP 201: HMMs, Semirings, and WFSAs

Jeffrey Flanigan

November 2, 2021

University of California Santa Cruz
jmflanig@ucsc.edu

Slides and figures adapted from David Bamman, Diyi Yang, Yejin Choi, and Noah Smith

Announcements

- Midterm on Friday
  - On Canvas, take home
  - Available from 9am - 1pm on Saturday
  - Covers material up to this lecture
- Assignment 2 out, due Tuesday 11/9

Parts of speech

- Parts of speech are categories of words defined distributionally by the morphological and syntactic contexts a word appears in.

Morphological distribution

POS often defined by distributional properties; verbs = the class of words that each combine with the same set of affixes

	-s	-ed	-ing
walk	walks	walked	walking
slice	slices	sliced	slicing
believe	believes	believed	believing
of	*ofs	*ofed	*ofing
red	*reds	*redded	*reding

Bender 2013

Morphological distribution

We can look to the function of the affix (denoting past tense) to include irregular inflections.

	-s	-ed	-ing
walk	walks	walked	walking
sleep	sleeps	slept	sleeping
eat	eats	ate	eating
give	gives	gave	giving

Bender 2013

Syntactic distribution

- Substitution test: if a word is replaced by another word, does the sentence remain grammatical?

Kim saw the	elephant	before we did
	dog	
	idea	
	*of	
	*goes	

Bender 2013

Syntactic distribution

- These can often be too strict; some contexts admit substitutability for some pairs but not others.

Kim saw the	elephant	before we did
	*Sandy	

both nouns but common vs. proper

Kim *arrived the	elephant	before we did

both verbs but transitive vs. intransitive

Bender 2013

Nouns	People, places, things, actions-made-nouns ("I like swimming"). Inflected for singular/plural
Verbs	Actions, processes. Inflected for tense, aspect, number, person
Adjectives	Properties, qualities. Usually modify nouns
Adverbs	Qualify the manner of verbs ("She ran downhill extremely quickly yesteray")
Determiner	Mark the beginning of a noun phrase ("a dog")
Pronouns	Refer to a noun phrase (he, she, it)
Prepositions	Indicate spatial/temporal relationships (on the table)
Conjunctions	Conjoin two phrases, clauses, sentences (and, or)

Sequence labeling

$$x = \{x_1, \ldots, x_n\}$$

$$y = \{y_1, \ldots, y_n\}$$

- For a set of inputs x with n sequential time steps, one corresponding label y_i for each x_i

Named entity recognition

B-PERS I-PERS O O O O ORG

tim cook is the ceo of apple

3 or 4-class:

- person
- location
- organization
- (misc)

7-class:

- person
- location
- organization
- time
- money
- percent
- date

BIO tagging

For segmentation tasks, can use BIO tagging.

- B (with a label) denotes the start of a segment (with that label)
- I (with a label) denotes continuing the segment
- O denotes “outside” i.e. not a segment

Variations to this tagging scheme can improve performance.
A popular one: use B only if two spans are next to each other, otherwise default to I for the start.

Supersense tagging

O B-artifact I-artifact B-motion O B-time O O O B-group

The station wagons arrived at noon, a long shining line

O B-motion O O B-location I-location

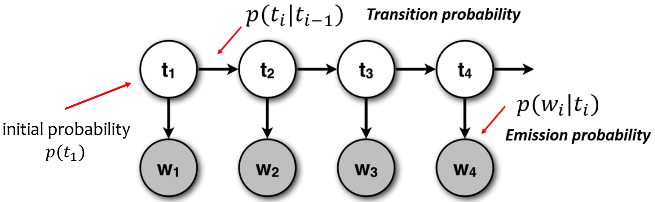
that coursed through the west campus.

1	person	7	cognition	13	attribute	19	quantity	25	plant
2	communication	8	possession	14	object	20	motive	26	relation
3	artifact	9	location	15	process	21	animal		
4	act	10	substance	16	Tops	22	body		
5	group	11	state	17	phenomenon	23	feeling		
6	food	12	time	18	event	24	shape		

Noun supersenses (Ciarmita and Altun 2003)

Tagging

- What is the **most likely sequence of tags** for the given **sequence of words w**



Three Basic Problems for HMMs

- **Likelihood** of the input: How likely the sentence “I love cat” occurs
  - Forward algorithm
- **Decoding** (tagging) the input: POS tags of ”I love cat” occurs
  - Viterbi algorithm
- **Estimation** (learning): How to learn the model?
  - Find the best model parameters
    - Case 1: supervised – tags are annotated (MLE)
    - Case 2: unsupervised -- only unannotated text (Forward-backward algorithm)

Last time: Viterbi Algorithm

END							v_T(END)
DT		v_1(DT)	v_2(DT)	v_3(DT)	v_4(DT)	v_5(DT)	
NNP		v_1(NNP)	v_2(NNP)	v_3(NNP)	v_4(NNP)	v_5(NNP)	
VB		v_1(VB)	v_2(VB)	v_3(VB)	v_4(MD)	v_5(MD)	
NN		v_1(NN)	v_2(NN)	v_3(NN)	v_4(NN)	v_5(NN)	
MD		v_1(MD)	v_2(MD)	v_3(MD)	v_4(MD)	v_5(MD)	
START							
		Janet	will	back	the	bill	

Each cell keeps track of the score of the best tag sequence ending in that tag at time i. Let's call this $\pi(i, y_i)$

Derivation: Viterbi Algorithm (Dynamic Programming)

$\hat{y} = \text{argmax}_{y_1 \dots y_n} p(x_1 \dots x_n, y_1 \dots y_n)$

- Define $\pi(i, y_i)$ to be the max score of a sequence of length i ending in tag y_i

$$
\begin{aligned}
\pi(i, y_i) &= \max_{y_1 \cdots y_{i-1}} p(x_1 \dots x_i, y_1 \dots y_i) \\
&= \max_{y_{i-1}} p(x_i | y_i) p(y_i | y_{i-1}) \max_{y_1 \cdots y_{i-2}} p(x_1 \dots x_i, y_1 \dots y_{i-1}) \\
&= \max_{y_{i-1}} p(x_i | y_i) p(y_i | y_{i-1}) \pi(i-1, y_{i-1})
\end{aligned}
$$

- We now have an efficient algorithm. Start with $i = 0$ and work your way to the end of the sentence.

Sum over all tag sequences: Forward Algorithm

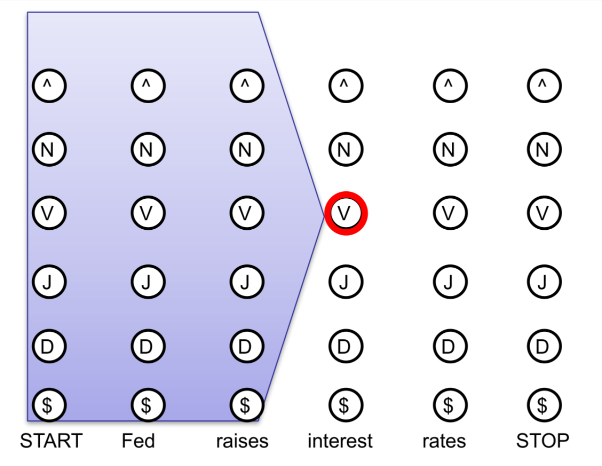
$p(x_1 \ldots x_n) = \sum_{y_1 \ldots y_n} p(x_1 \ldots x_n, y_1 \ldots y_n)$

- Define $\pi(i, y_i)$ to be the sum over all tag sequences of length i ending in tag y_i

$$\begin{aligned}
\pi(i, y_i) &= \sum_{y_1 \ldots y_{i-1}} p(x_1 \ldots x_i, y_1 \ldots y_i) \\
&= \sum_{y_{i-1}} p(x_i|y_i) p(y_i|y_{i-1}) \sum_{y_1 \ldots y_{i-2}} p(x_1 \ldots x_i, y_1 \ldots y_{i-1}) \\
&= \sum_{y_{i-1}} p(x_i|y_i) p(y_i|y_{i-1}) \pi(i-1, y_{i-1})
\end{aligned}$$

- Again, we an efficient algorithm. Start with $i = 0$ and work your way to the end of the sentence.

Sum over all tag sequences: Forward Algorithm



Each cell keeps track of the score of the sum over all sequences ending in that tag at time i. We call this $\pi(i, y_i)$

Runtime of Viterbi and Forward algorithms

- Linear in sentence length n
- Polynomial in the number of possible tags |K|

$$\pi(i, y_i) = \max_{y_{i-1}} e(x_i|y_i)q(y_i|y_{i-1})\pi(i-1, y_{i-1})$$

- Specifically:

$$O(n|\mathcal{K}|) \text{ entries in } \pi(i, y_i)$$

$$O(|\mathcal{K}|) \text{ time to compute each } \pi(i, y_i)$$

- Total runtime: $O(n|\mathcal{K}|^2)$

- Q: Is this a practical algorithm?
- A: depends on |K|….

Speeding Up Viterbi: Beam search

- **Beam search**: At each timestep, only keep the k best tags
  Only need to check k previous tags at each timestep.
  k is called the "beam size"
- This is an **approximate algorithm** (no guarantee it will produce the Viterbi tag sequence)
- Runtime becomes $O(nk|T|)$
- If choose $k = 1$ we get a **greedy algorithm** (It never goes back to consider alternatives to choices it makes. It chooses the best tag sequence in a greedy fashion left to right)

Higher-Order HMMs

- Second order HMM: $P(\mathbf{t}, \mathbf{w}) = \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}, t_{i-2})$
- Nth-order HMM:
  $P(\mathbf{t}, \mathbf{w}) = \prod_{i=1}^{n} P(w_i|t_i)P(t_i|t_{i-1}, t_{i-2}, \ldots, t_{i-(N-1)})$
- Viterbi and Forward algorithms exist for higher-order HMMs
- Need to keep track of $N - 1$ tags in each Viterbi cell
- Runtime: $O(n|T|^N)$

Working in log space

- Similar to n-gram LMs, if we work with probabilities, we will get underflows from multiplying many small numbers
- $log(a * b) = log(a) + log(b)$
- Viterbi algorithm
  - Use $log(Pr(t_i|t_{i-1}))$ and $log(Pr(w_i|t_i))$
  - Instead of multiply, add
  - The Viterbi array stores log probabilities
- Forward algorithm
  - Use $log(Pr(t_i|t_{i-1}))$ and $log(Pr(w_i|w_{i-1}))$
  - Instead of multiply, add
  - To add log probabilities, use **log-sum**:
    If $a > b$ (otherwise, switch them):
    $log(a + b) = log(a(1 + b/a)) = log(a) + log(1 + b/a) = log(a) + log(1 + e^{log(b)-log(a)})$

Viterbi Algorithm

function VITERBI(*observations* of len *T*,*state-graph* of len *N*) **returns** *best-path*, *path-prob*

create a path probability matrix *viterbi[N,T]*
for each state *s* **from** 1 **to** *N* **do** ; initialization step
  $viterbi[s,1] \leftarrow \pi_s * b_s(o_1)$
  $backpointer[s,1] \leftarrow 0$
for each time step *t* **from** 2 **to** *T* **do** ; recursion step
  for each state *s* **from** 1 **to** *N* **do**
    $viterbi[s,t] \leftarrow \max_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
    $backpointer[s,t] \leftarrow \operatorname{argmax}_{s'=1}^{N} viterbi[s',t-1] * a_{s',s} * b_s(o_t)$
$bestpathprob \leftarrow \max_{s=1}^{N} viterbi[s,T]$; termination step
$bestpathpointer \leftarrow \operatorname{argmax}_{s=1}^{N} viterbi[s,T]$; termination step
bestpath ← the path starting at state *bestpathpointer*, that follows backpointer[] to states back in time
return *bestpath*, *bestpathprob*

Figure A.9 Viterbi algorithm for finding optimal sequence of hidden states. Given an observation sequence and an HMM $\lambda = (A,B)$, the algorithm returns the state path through the HMM that assigns maximum likelihood to the observation sequence.

For details, see Jurafsky and Martin Appendix A.

Forward Algorithm

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
  $forward[s,1] \leftarrow \pi_s * b_s(o_1)$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] * a_{s',s} * b_s(o_t)$$

$$forwardprob \leftarrow \sum_{s=1}^{N} forward[s,T] \quad \text{; termination step}$$

return *forwardprob*

Figure A.7 The forward algorithm, where $forward[s,t]$ represents $\alpha_t(s)$.

For details, see Jurafsky and Martin Appendix A.

Viterbi vs Forward

If we just want the score of the best sequence (not the actual sequence), then Viterbi and Forward are the same except:

- We have a max in Viterbi, and a sum in Forward
- Everything else is the same!
- **If we use semirings, we can use the same code to do either algorithm**

(Also easier to code, less likely to have bugs due to backpointers, get k-best outputs for free)

Definition: Semiring

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- The elements A (so far real numbers)
- $\otimes$ generalized times operation
- $\oplus$ generalized plus operation
- 0_s semiring 0 (identity for $\oplus$: $a \oplus 0_s = a$ for all $a \in A$)
- 1_s semiring 1 (identity for $\otimes$: $a \otimes 1_s = a$ for
- $\otimes$, $\oplus$ must satisfy the semiring axioms

Semiring Axioms

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- Satisfies commutative law (for $\oplus$), distributive law, associative law
- Missing: no inverse for addition (not true: all $a \in A$, exists b such that $a \oplus b = 0_s$)
  Rings have additive inverses
- Missing: no inverse for multiplication (not true: all $a \in A$, exists b such that $a \otimes b = 1_s$)
  Fields have multiplicative inverses and additive inverses

Example: Real Semiring

Used in Forward algorithm

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- The elements $A = \mathbb{R}$: Real numbers
- $a \otimes b = a \times b$
- $a \oplus b = a + b$
- $0_s = 0$ (identity for $\oplus$: $a \oplus 0_s = a$ for all $a \in A$)
- $1_s = 1$ (identity for $\otimes$: $a \otimes 1_s = a$ for all $a \in A$)

Example: Viterbi Semiring

Used in Viterbi algorithm to get the max probability

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- The elements $A = \mathbb{R}^+$: Real numbers ≥ 0
- $a \otimes b = a \times b$
- $a \oplus b = \max(a, b)$
- $0_s = 0$ (identity for $\oplus$: $a \oplus 0_s = a$ for all $a \in A$)
- $1_s = 1$ (identity for $\otimes$: $a \otimes 1_s = a$ for all $a \in A$)

Board work: Viterbi Semiring

The elements in the semiring: $A = \mathbb{R}^+$
$a \otimes b = a \times b$ and $a \oplus b = \max(a, b)$

(5 min) On your own

- Verify semiring 0 and 1 ($0_s = 0$ and $1_s = 1$) satisfy $a \oplus 0_s = a$ for all $a \in A$ and $a \otimes 1_s = a$ for all $a \in A$
- Verify semiring distributive law: $a \oplus (b \otimes c) = (a \otimes b) \oplus (a \otimes c)$

(3 min) Discuss with a partner

Example: Log-Real Semiring

Used in Forward algorithm, in log space to avoid underflows

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- The elements $A = \mathbb{R}$: Real numbers
- $a \otimes b = a + b$
- $a \oplus b = log(e^a + e^b)$
- $0_s = -\infty$
- $1_s = 0$

Example: Max-Plus (Tropical) Semiring

Used in Viterbi algorithm, in log space to avoid underflows

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- The elements $A = \mathbb{R}$: Real numbers
- $a \otimes b = a + b$
- $a \oplus b = \max(a, b)$
- $0_s = -\infty$
- $1_s = 0$

Example: Viterbi-Derivation Semiring

Use in Viterbi algorithm to get the max tag sequence and probability score

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- The elements $A = \mathbb{R} \times T^*$: Real number and tag sequence
- $(a_v, a_s) \in A$ **value and sequence**
- $(a_v, a_s) \otimes (b_v, b_s) = (a_s \times b_s, a_s.b_s)$ **multiply the numbers, concatenate the tag sequence**
- $(a_v, a_s) \oplus (b_v, b_s) = (a_v, a_s)\ if\ a_v > b_v,\ else\ (b_v, b_s)$
- $0_s = (-\infty, \epsilon)$
- $1_s = (1, \epsilon)$

Example: Viterbi-Derivation Semiring, Tropical Version

Use in Viterbi algorithm to get the max tag sequence and log-probability score

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- The elements $A = \mathbb{R} \times T^*$: Real number and tag sequence
- $(a_v, a_s) \in A$ **value and sequence**
- $(a_v, a_s) \otimes (b_v, b_s) = (a_s + b_s, a_s.b_s)$ **add the numbers, concatenate the tag sequence**
- $(a_v, a_s) \oplus (b_v, b_s) = (a_v, a_s) \ if \ a_v > b_v, \ else \ (b_v, b_s)$
- $0_s = (-\infty, \epsilon)$
- $1_s = (0, \epsilon)$

Example: k-Best Semiring

Use to get the k best tag sequences and their probability scores

$S = \langle A, \otimes, \oplus, 0_s, 1_s \rangle$

- The elements $A = (\mathbb{R} \times T^*)^k$: k Real numbers and tag sequences
- $((a_v^1, a_s^1) \ldots (a_v^k, a_s^k)) \in A$
- $\otimes$: compute all combinations from both lists by multiplying the numbers and concatenating the tag sequence. Keep only the k best
- $\oplus$: merge the two lists by keeping the k best
- $0_s = (-\infty, \epsilon)^k$
- $1_s = (1, \epsilon)^k$

Implementing a Semiring Viterbi/Forward algorithm

In the code:

- Create a class for your semiring elements (for example, Viterbi-derivation semiring)
  Implement semiring plus and times functions
  Implement constructors for transition, emission, starting probabilies
  Implement constructors for semiring 0 and 1
- Implement Forward algorithm, using semiring plus and times

- To get the Forward algorithm in log space, use Log-Real semiring
- To get the Viterbi algorithm in log space, use the Viterbi-derivation semiring, Tropical version

Forward Algorithm

function FORWARD(*observations* of len T, *state-graph* of len N) **returns** *forward-prob*

create a probability matrix *forward[N,T]*
for each state s **from** 1 **to** N **do** ; initialization step
  $forward[s,1] \leftarrow \pi_s \, * \, b_s(o_1)$
for each time step t **from** 2 **to** T **do** ; recursion step
  for each state s **from** 1 **to** N **do**

$$forward[s,t] \leftarrow \sum_{s'=1}^{N} forward[s',t-1] \, * \, a_{s',s} \, * \, b_s(o_t)$$

$$forwardprob \leftarrow \sum_{s=1}^{N} forward[s,T] \qquad \text{; termination step}$$

return *forwardprob*

Figure A.7 The forward algorithm, where $forward[s,t]$ represents $\alpha_t(s)$.

For details, see Jurafsky and Martin Appendix A.

Weighted Automata

Many of the models (n-gram language models, HMMs) we've covered so far can be represented as WFSAs and WFSTs

Weighted Automata

- Weighted automata can be understood as transducing strings to weights
- Weighted transducers can be understood as transducing pairs of strings to weights

Weighted Finite-State Automaton

Element	Definition
Q	Finite set of states
Σ	Finite vocabulary
$I \subseteq Q$	Set of initial states
$F \subseteq Q$	Set of final states
$E \subseteq (Q \times (\Sigma \cup \{\varepsilon\}) \times Q)$	Set of transitions (edges)
$\lambda : I \rightarrow \mathbb{R}_{\geq 0}$	Initial weights
$\rho : F \rightarrow \mathbb{R}_{\geq 0}$	Final weights
$w : E \rightarrow \mathbb{R}_{\geq 0}$	Transition weights

Paths in WFSAs

- Define the function $p(e)$ to be the **previous state** of an edge e in E
- Define the function $n(e)$ to pick out the **next state** of an edge e
- A **path** π in E* is a sequence of transitions $e_1 e_2 \ldots e_\ell$ such that $n(e_i)=p(e_{i+1})$ for all $i \in [1,\ell)$
- We overload p and n to be defined on paths

  $p(\pi)=p(e_1)$ $n(\pi)=p(e_\ell)$

Weight of a Path

- We generalize the **transition weight** of the path as the **product** of all transitions

$$w(\pi) = w(e_1) \times w(e_2) \times \cdots \times w(e_\ell)$$
$$= \prod_{i=1}^{\ell} w(e_i)$$

- The **output weight** (or **score**) of the path is defined to be

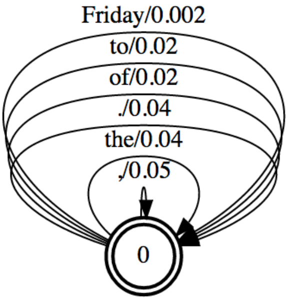
$$score(\pi) = \lambda(p(\pi)) \times w(\pi) \times \rho(n(\pi))$$

Deterministic and Nondeterministic

- Deterministic WFSA
  - There is one initial state q_0
  - For each q in Q and s in Σ, there is at most one r in Q such that $(q, s, r) \in E$.
  - There is no $(q,r) \in (Q \times Q)$ s.t. $(q,\varepsilon,r) \in E$
- Can we determinize WFSAs?
  - Sometimes, but not always
- Can we minimize (deterministic) WFSAs?
  - Yes. There might be more than one way to define "minimization."

Language Models can be represented as a WFSA

The Simplest Language Model



word	transition weight (*w*)
,	0.05
the	0.04
.	0.04
of	0.02
to	0.02
…	…
Friday	0.002

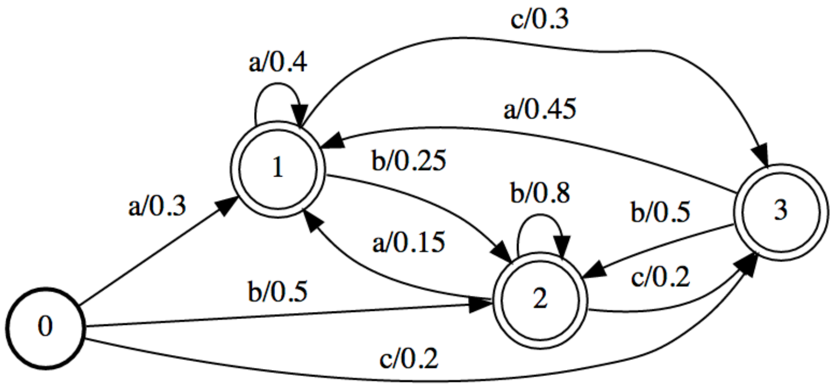
λ
1

ρ
0.04

What Else Might We Do?

- We can encode more information in the states.
  - More states: more information.
  - Often used to encode history and increase the "memory" of the process.
  - Example: create a distinct state for each (n-1) word history … this is an **n-gram** language model.

A Simple Bigram Model on {a, b, c}



WFSA

weighted languages
disambiguation

WFS**T**

weighted string-to-string mappings

FSA

regular languages

FS**T**

regular *relations*
string-to-string mappings

Beyond WFSAs: WFSTs

- Weighted finite-state *transducers*: combine the two-tape idea from last time with the weighted idea from today.
  - Very general framework; key operation is **weighted composition**.
  - Best-path algorithm variations: best path and output given input, best path given input and output, …
- Weights do not have to be numbers.
  - Boolean weights = traditional FSAs/FSTs.
  - Real weights = the case we explored so far.
  - Many other *semirings* (sets of possible weight values and operations for combining weights)

WFSAs and WFSTs: Algorithms

- OpenFST documentation contains high-level view of most algorithms you think you want, and pointers.
- The set of people developing general WFST algorithms largely overlaps with the OpenFST team.
  - Mehryar Mohri, Cyril Allauzen, Michael Reilly
- This is an active area of research!

WFST Examples

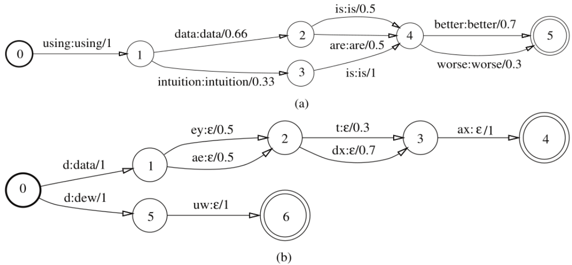


Figure 2: Weighted finite-state transducer examples.

Board Work

HMMs are can be represented as WFSTs: the input is the word sequence, the output is the tag sequence.
The easiest way: one WFST for emission probabilities $Pr(w_i|t_i)$ and one WFST (actually a WFSA as an WFST) for transition probabilities $Pr(t_i|t_{i-1})$.
These two WFSTs are composed to get the HMM

On your own (5 min)
For an HMM with two tags U, V and vocabulary a, b:

- Construct the emission WFST
- Construct the transition WFST

Discuss with a partner (3 min)

Why WFSAs / WFSTs

- Can use off the shelf software to find best path (Viterbi algorithm) and sum over paths (Forward algorithm)
- Example application: speech recognition (WFST for acoustic model, and WFST for n-gram LM)
- Nowadays WFSAs used to analyse seq2seq neural networks (theory)