Computational Morphology: FSAs and FSTs: Weights and Probabilities

Yulia Zinova

09 April 2014 – 16 July 2014

Weights and Probabilities: Inventory

- Why? Disambiguation in morphological and syntactic processing.
- We need: sums, products, logarithms and exponents.
- Shorthand notation: Σ for sum, Π for product.
- Convention: we use natural logarithms (base e).

Properties of log and exp

- Reminder 1: basic relationship between log and exp

$$\log(\exp(x)) = log(e^x) = x \tag{1}$$

$$\exp(\log(y)) = e^{\log(y)} = y \tag{2}$$

- Remainder 2: the log of the product is a sum of logs. Useful, because logs can be taken prior to combination when product gives extremely small floats.
- Remainder 3: log preserves order (if $x > y$, then $\log(x) > \log(y)$)

Probabilistic models

- The probabilistic models we are going to use are discrete distributions.
- This means there are k discrete outcomes (such as different words from a vocabulary Σ of size k) each with its own parameter.
- When $k = 2$, it is a binominal distribution; when $k > 2$, it is a multinominal distribution.
- If we assign a probability to each word w in a vocabulary Σ, it is as multinominal distribution with $|\Sigma|$ parameters where $\Sigma_{w \in \Sigma} P(w) = 1$.

Relative frequency estimation

- Suppose we have
  - a corpus of N words
  - words are taken from vocabulary Σ
  - $f(w)$ is the frequency of the word (its count)
- Relative frequency estimation is

$$P(w) = \frac{f(w)}{N} \tag{3}$$

- **Question:** Problem with low-frequency words?

Relative frequency estimation

- Suppose we have
  - a corpus of N words
  - words are taken from vocabulary Σ
  - $f(w)$ is the frequency of the word (its count)
- Relative frequency estimation is

$$P(w) = \frac{f(w)}{N} \tag{3}$$

- **Question:** Problem with low-frequency words?
- **Answer:** Zero probability is given to all the words that have not occured in our corpus.

Most probable...

- $\hat{p}$ is the maximum probability of a word in the corpus

$$\hat{p} = \max_{w} \; P(w) \tag{4}$$

- $\hat{w}$ is the word that has the highest probability

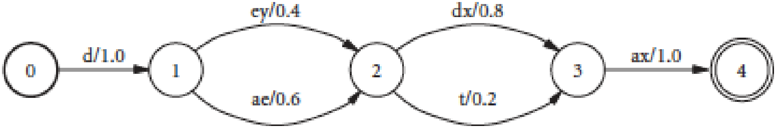
$$\hat{p} = \arg\max_{w} \; P(w) \tag{5}$$

$$P(\hat{w}) = \hat{p} \tag{6}$$

Weights and costs

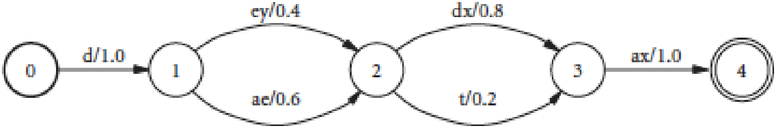
- FSAs and FSTs can be extended to include *weights* or *costs* on the arcs $\rightarrow$ *weighted finite-state automata (WFSA)* and *weighted finite-state transducers*.
- Weights usually represent probabilities, or negative log probabilities, or different analyses.
- The sum of probabilities on all the arcs leaving the given state must sum to 1.
- The probability of a particular path is given by multiplying the individual arc probabilities.

Example



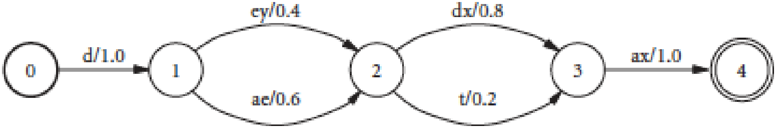
- **Question** What is the probability of the pronunciation /deytax/?

Example



- **Question** What is the probability of the pronunciation /deytax/?
- **Answer** 1 * 0.4 * 0.2 * 1 = 0.08

Example



- **Question** What is the probability of the pronunciation /deytax/?
- **Answer** 1 * 0.4 * 0.2 * 1 = 0.08
- In a toy example probabilities are fine, but if the system is real, this will lead to difficulties in float point representation of the values.
- Because of this, negative log probabilities are used: they must be summed, not multiplied, and smaller numbers correspond to more probable events.

Weights along different paths

- In addition to specifying how weights are combined along a path, one must also specify how weights are combined between the paths.
- When probabilities are used, the probabilities of two paths are summed.
- Combining weights along one paths will be called *times* operation, between two paths – *plus* operation

Some mathematics: monoid

Definition A monoid is a pair $(M, \bullet)$, where M is a set and $\bullet$ is a binary operation on M, obeying the following rules:

1. **closure**: for all a, b in M, $a \bullet b$ is in M
2. **identity**: there exists an element e in M, such that for all a in M, $a \bullet e = e \bullet a = a$. This is termed the neutral element.
3. **associativity**: $\bullet$ is an associative operation; that is, for all a, b, c in M, $(a \bullet b) \bullet c = a \bullet (b \bullet c)$

- A monoid $(M, \bullet)$ is *commutative* if $a \bullet b = b \bullet a$ for all a, b in M.

Some mathematics: semiring

Definition A semiting is a triple $(\mathbb{K}, \bigoplus, \bigotimes)$, where $\mathbb{K}$ is a set and $\bigoplus$ and $\bigotimes$ are binary operations on $\mathbb{K}$, obeying the following rules:

1. $(\mathbb{K}, \bigoplus)$ is a commutative monoid with neutral element denoted by 0;
2. $(\mathbb{K}, \bigotimes)$ is a monoid with neutral element denoted by 1;
3. The product $(\bigotimes)$ distributes with respect to the sum $(\bigoplus)$, i.e., $a \bigotimes (b \bigoplus c) = (a \bigotimes b) \bigoplus (a \bigotimes c)$;
4. For all a in $\mathbb{K}$, $a \bigotimes 0 = 0 \bigotimes a = 0$.

Semirings

- Common semirings used in speech and language processing:
  1. $(+, \times)$, or "real" semiring;
  2. $(\min, +)$, or "tropical"semiring.
- The $(+, \times)$ semiring is appropriate for use with probabilities:
  - to get the probability of a path, one multiplies along the path;
  - to get the probability of a set of paths, one sums the probabilities of those paths.
- The $(min, +)$ semiring is appropriate for use with negative log probabilities:
  - one sums the weights along the path,
  - one computes the minimum of a set of paths (useful if looking for the best scoring path, since lower scores are better with negative logs).

Weighted finite-state automaton

Definition A weighted finite-state automaton is an octuple $A = (Q, s, F, \Sigma, \delta, \lambda, \sigma, \rho)$, where

- $(Q, s, F, \Sigma, \delta)$ is a finite-state automaton;
- an initial output function $\lambda : s \rightarrow \mathbb{K}$ assigns a weight to entering the automaton;
- an output function $\sigma : \delta \rightarrow \mathbb{K}$ assigns a weight to transitions in the automaton;
- a final output function $\rho : F \rightarrow \mathbb{K}$ assigns a weight to leaving the automaton.

Label, origin, destination

- For any transition $d \in \delta$, let $i[d] \in (\Sigma \cup \epsilon)$ be its label; $p[d] \in Q$ its origin state; and $n[d] \in Q$ its destination state. A path $\pi = d_1 \ldots d_k$ consists of k transitions $d_1, \ldots, d_k \in \delta$, where $n[d_j] = p[d_{j+1}]$ for all j, i.e., the destination state of transition d_j is the origin state of transition $d_j + 1$.
- Extending the definitions of label, origin and destination to paths: let $i[\pi] = i[d_1] \ldots i[d_k]$; $p[\pi] = p[d_1]$; and $n[\pi] = n[d_k]$. A *cycle* is a path π such that $p[\pi] = n[\pi]$, i.e., a path that starts and ends at the same state.
- An acyclic automaton or transducer has no cycles.

Automata Intersection

Given two automata $M = (Q, s, F, \Sigma, \delta)$ and $M' = (Q', s', F', \Sigma', \delta')$, construct a new automaton M'' such that:

- Its set of states $Q'' = Q \times Q'$ is the cross-product of the states of the individual machines.
- $s'' = (s, s')$
- $F'' = F \times F'$
- $\Sigma'' = \Sigma \cap \Sigma'$
- $\delta''((p, p'), x) = (q, q')$ just in case $\delta(p, x) = q$ is in M and $\delta'(p', x) = q'$ is in M'.

Composition of transducers

- The basic algorithm for transducer composition is essentially the same;
- the output label of one transducer is matched with the input label of the other;
- The resulting arc has
    - as its input label the input label of the arc from the first machine
    - as its output label the output label of the arc from the second machine.
- Automata can be seen as a special case of transducers, where the input and output symbols are always identical.
- For weighted intersection or composition, the weights of the resulting path as is the extend ($\bigotimes$) of the weights of the two input paths.

Deterministic automata

- Non-deterministic finite-state automata accept the same class of languages as DFSA: regular languages.
- For every NFSA there is an DFSA that accepts the same language.
- DFSA has usually more states than NFSA for the same language.
- Efficiency: DFSA is more efficient $\rightarrow$ used for computation.
- After making an automaton deterministic, it is important to minimize it (if possible).

From NFSA to DFSA: powerset construction

- Powerset construction applied to NFSA that does not allow state transformations without consuming input symbols (ϵ-transitions).
- Our NFSA: a quintuple $(Q, s, F, \Sigma, \delta)$. **Question:** What is what in this quintuple?

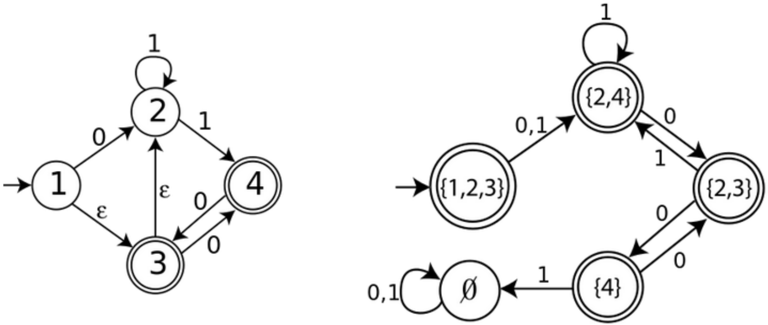
From NFSA to DFSA: powerset construction

- Powerset construction applied to NFSA that does not allow state transformations without consuming input symbols (ϵ-transitions).
- Our NFSA: a quintuple $(Q, s, F, \Sigma, \delta)$. **Question:** What is what in this quintuple?
- **Answer:** Q is the set of states, s is the initial state, F – the set of accepting states, Σ – the alphabet, δ – the transition function.
- The corresponding DFSA has states corresponding to subsets of Q.
- The initial state of the DFSA is s, the (one-element) set of initial states.
- The transition function of the DFSA maps a state S (representing a subset of Q) and an input symbol x to the set $\delta(S, x) = \cup\delta(q, x)|qS$, (the set of all states that can be reached by an x-transition from a state in S).
- A state S of the DFSA is an accepting state if and only if at least one member of S is an accepting state of the NFSA.

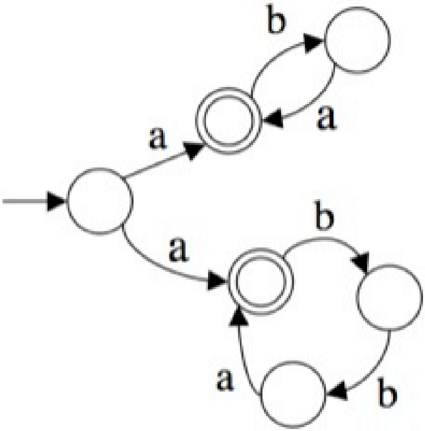
From NFSA to DFSA: powerset construction with ϵ

- For an NFA with ϵ-transitions:
  - the initial state consists of all NFSA states reachable by ϵ-transitions from s
  - the value $\delta(S, x)$ of the transition function is the set of all states reachable by ϵ-transitions from $\cup\delta(q, x)|qinS$.

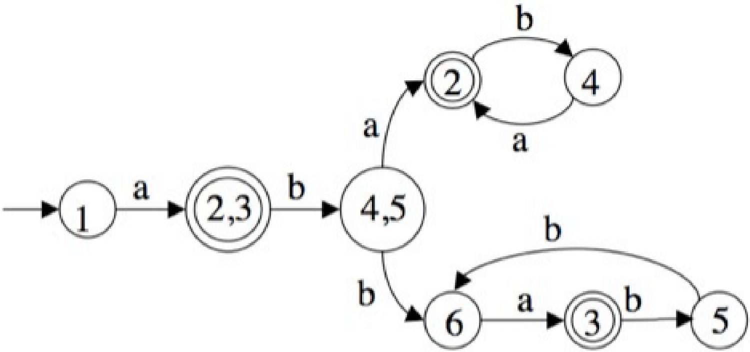
From NFSA to DFSA: example



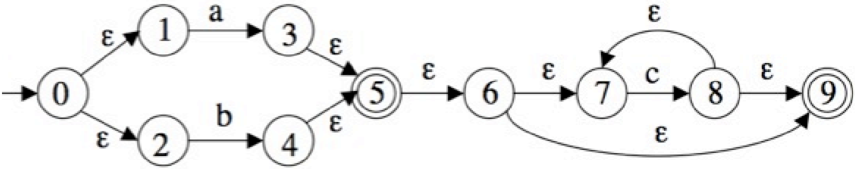
From NFSA to DFSA: exercise



From NFSA to DFSA: answer

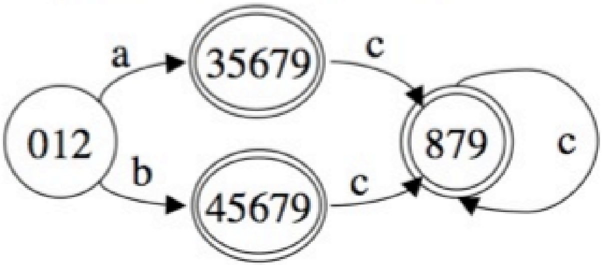


From NFSA to DFSA: exercise with ϵ



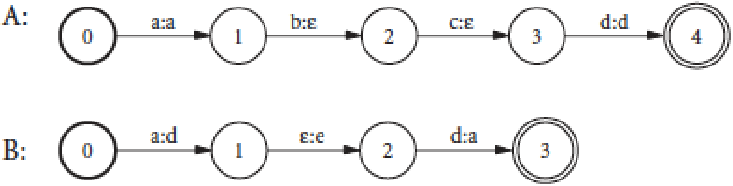
- Which regular language recognizes this automaton?

From NFSA to DFSA: answer

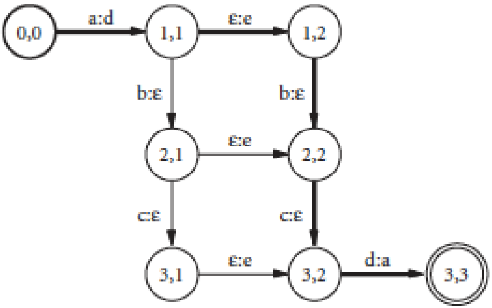


- Language: $(a|b)c*$
- How to minimize the DFSA?

Transducers for composition



Result of a naive composition



- This is correct, but inefficient.
- With weights, such naive algorithm leads to incorrect weights being assigned.
- Solution: inserting some *epsilon-filter* as a middle layer in composition. The algorithm is complicated, so we do not review it.

NFSA → DFSA: weights

- Weighted automata and transducers (whether weighted or not) cannot in general be determinized, but certain types of machines, including acyclic machines can be.
- Since machine minimization requires a determinized machine, this also implies that not all weighted acceptors or transducers can be minimized (some classes can be).
- Transducers and weighted acceptors that fall into the class of determinizable and minimizable machines include machines that are useful in speech and language processing.
- For example, a dictionary can be modeled as an acyclic transducer, mapping input words to some other property such as their part of speech or pronunciation; and a lattice of possible analyses output by a speech recognizer can be modeled as an acyclic weighted acceptor.
- Determinizing and minimizing such machines can provide large efficiency gains.