Chapter 4

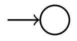
Weighted Automata

If you've taken *Theory of Computing*, you should be quite familiar with finite automata; if not, you may be familiar with regular expressions, which are equivalent to finite automata. Many models in NLP can be thought of as finite automata, or variants of finite automata, including n-gram language models. Although this may feel like overkill at first, we'll soon see that formalizing models as finite automata makes it much easier to combine models in various ways.

4.1 Definitions

A *finite automaton (FA)* is an imaginary machine that reads in a string and outputs an answer, either "accept" or "reject." (For example, a FA could accept only words in an English dictionary and reject all other strings.) At any given time, the machine is in one *state* out of a finite set of possible states. It has rules, called *transitions*, that tell it how to move from one state to another.

A FA is typically represented by a directed graph. We draw nodes to represent the various states that the machine can be in. The node can be drawn with or without the state's name inside. The machine starts in the *initial state* (or *start state*), which we draw as:



The edges of the graph represent transitions, for example:

$$\begin{tikzcd} q \arrow[r, "a"] & r \end{tikzcd}$$

which means that if the machine is in state q and the next input symbol is a, then it can read in a and move to state r. The machine also has zero or more *final states* (or *accept states*), which we draw as:

If the machine reaches the end of the string and is in a final state, then it accepts the string.

We say that a FA is *deterministic* if every state has the property that, for each label, there is exactly one exiting transition with that label. Otherwise, it is *nondeterministic*. We will focus on deterministic FAs for now.

Here's a more formal definition (for those who like formal definitions).

Definition 1. A *finite automaton* is a tuple $M = \langle Q, \Sigma, \delta, s, F \rangle$, where:

- Q is a finite set of *states*
- Σ is a finite alphabet
- δ is a set of *transitions* of the form $q \xrightarrow{a} r$, where $q, r \in Q$ and $a \in \Sigma$
- $s \in Q$ is the *initial state*
- $F \subseteq Q$ is the set of *final states*

A string $w = w_1 \cdots w_n \in \Sigma^*$ is accepted by M iff there is a sequence of states $q_0, \ldots, q_n \in Q$ such that $q_0 = s, q_n \in F$, and for all i, $1 \leq i \leq n$, there is a transition $q_{i-1} \xrightarrow{w_i} q_i$. We write $L(M)$ for the set of strings accepted by M.

A *weighted finite automaton* adds a *weight* to each transition (that is, δ is a mapping $Q \times \Sigma \times Q \to \mathbb{R}$). The weight of an accepting path through a weighted FA is the product of the weights of the transitions along the path. A weighted FA defines a weighted language, or a distribution over strings, in which the weight of a string is the sum of the weights of all accepting paths of the string.

In a *probabilistic FA*, each state has the property that the weights of all of the exiting transitions sum to one. Moreover, there is a special *stop symbol*, which we write as `</s>`, that we assume appears at the end of every string, and only at the end of every string. Then the weighted FA also defines a probability distribution over strings.

4.2 Language Models

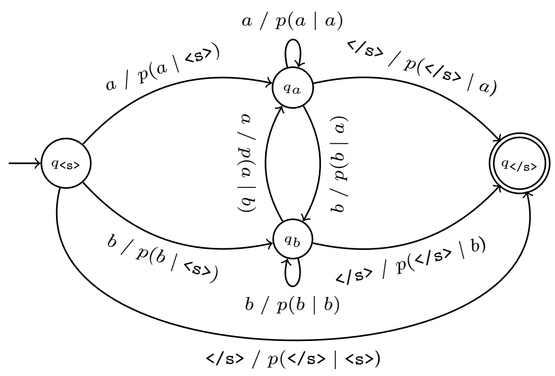
So, an m-gram language model is a probabilistic FA with a very simple structure. If we continue to assume a bigram language model, we need a state for every observed context, that is, one for `<s>`, which we call $q_{\texttt{<s>}}$, and one for each word type a, which we call q_a. And we need a final state $q_{\texttt{</s>}}$. For all a, b, there is a transition

$$q_a \xrightarrow{b/p(b|a)} q_b,$$

and for every state q_a, there is a transition

$$q_a \xrightarrow{\texttt{</s>}/p(\texttt{</s>}|a)} q_{\texttt{</s>}}.$$

The transition diagram (assuming an alphabet $\Sigma = \{a, b\}$) looks like this:



(4.1)

Generalizing to m-grams, we need a state for every $(m-1)$-gram. It would be messy to actually draw the diagram, but the transitions are not hard to describe:

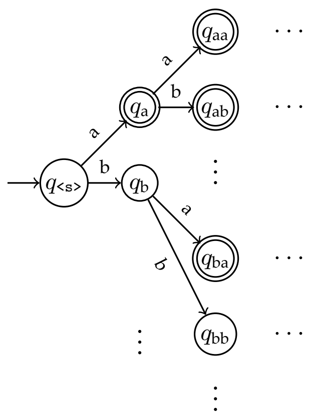
- For all $u \in \Sigma^{m-1}$, there is a state q_u.
- The start state is $q_{\texttt{<s>}^{m-1}}$.
- The accept state is $q_{\texttt{</s>}}$.
- For all $a \in \Sigma, u \in \Sigma^{m-2}, b \in \Sigma$, there's a transition

$$q_{au} \xrightarrow{b/p(b \mid au)} q_{ub}.$$

- For all $u \in \Sigma^{m-1}$, there's a transition

$$q_u \xrightarrow{\texttt{</s>}/p(\texttt{</s>} \mid u)} q_{\texttt{</s>}}.$$

One can imagine designing other kinds of language models as well. A commonly used structure is a *trie*, used for storing lists like dictionaries:



The portion shown accepts the strings {a, aa, ab, ba}. You could also add transitions reading space symbols from the accept states back to the start state to make the automaton recognize strings of words.

4.3 Training

If we are given a collection of strings $w^1, \ldots, w^N$ and a DFA M, we can learn weights very easily. For each string w^i, run M on w^i and collect, for each state q, counts $c(q)$, $c(q, a)$ for each word a, and $c(q, \texttt{</s>})$, which is the number of times that M stops in state q. Then the weight of transition $q \xrightarrow{a} r$ is $\frac{c(q,a)}{c(q)}$.

If the automaton is not deterministic, the above won't work. This is because, for a given string, there might be more than one path that accepts it, and we don't know which path's transitions to count. Training nondeterministic automata is the subject of a later chapter.

4.4 Smoothing

A smoothed m-gram model can also be represented as a weighted automaton. In principle, it should be easy – the only difference would be that the transition weights would be smoothed probabilities instead of unsmoothed probabilities.

However, this would mean creating a state q_u for all $u \in \Sigma^{m-1}$, even if u was not observed in the training data. Suppose that $|\Sigma| = 10^4$ and $m = 5$; then we would need 10^{16} states!

It's possible to reduce the size of the automaton without changing any string weights, but to really do this well, we need nondeterminism. Another solution would be to represent all these states and their transitions *implicitly*: that is, to construct them only as needed (and maybe destroy them when no longer needed).