Approximating probabilistic models as weighted finite automata

Ananda Theertha Suresh, Brian Roark, Michael Riley ,Vlad Schogol

{ theertha, roark, riley, vlads }@google.com
Google Inc.

May 20, 2019

Abstract

Weighted finite automata (WFA) are often used to represent probabilistic models, such as n-gram language models, since they are efficient for recognition tasks in time and space. The probabilistic source to be represented as a WFA, however, may come in many forms. Given a generic probabilistic model over sequences, we propose an algorithm to approximate it as a weighted finite automaton such that the Kullback-Leiber divergence between the source model and the WFA target model is minimized. The proposed algorithm involves a counting step and a difference of convex optimization, both of which can be performed efficiently. We demonstrate the usefulness of our approach on various tasks, including distilling n-gram models from neural models, building compact language models, and building open-vocabulary character models.

1 Introduction

Given a sequence of symbols $x_1, x_2, \ldots, x_{n-1}$, where symbols are drawn from the alphabet Σ, a probabilistic model S assigns probability to the next symbol $x_n \in \Sigma$ by

$$p_s[x_n|x_{n-1} \ldots x_1].$$

Such a model might be Markovian of order k, where

$$p_s[x_n|x_{n-1} \ldots x_1] = p_s[x_n|x_{n-1} \ldots x_{n-k+1}],$$

such as a k-gram language model (LM) [17] or it might be non-Markovian such as a long-short memory (LSTM) neural network language model [51]. Our goal is to approximate a probabilistic model as a *weighted finite automaton* (WFA) such that the weight assigned by the WFA is close to the probability assigned by the source model. Specifically, we will seek to minimize the Kullback-Leiber (KL) divergence between the source S and the target WFA model.

Representing the target model as a WFA has many advantages including efficient use, compact representation, interpretabilty, and composability. WFA models have been used in many applications including speech recognition [41], speech synthesis [24], optical character recognition [12], machine translation [32], computational biology [23], and image processing [3]. One particular problem of interest is language models for on-device (virtual) keyboard decoding [44], where WFA models are widely used due to space and time constraints. However, storing the training data in a

centralized server and training k-gram or WFA models directly may not be feasible due to privacy constraints [29]. To circumvent this, an LSTM model can be trained by federated learning [35, 29], converted to a WFA at the server, and then used for fast on-device inference. This not only may improve performance, but also provide additional privacy.

We allow *failure transitions* [2, 37] in the target WFA, which are taken only when no immediate match is possible at a given state, for compactness. For example, in the WFA representation of a backoff k-gram model, failure transitions can compactly implement the backoff [34, 17, 4, 42, 30]. The inclusion of failure transitions will complicate our analysis and algorithms but is highly desirable in applications such as keyboard decoding. Further, to avoid redundancy that leads to inefficiency, we assume the target model is *deterministic*, which requires at each state there is at most one transition labeled with a given symbol.

The approximation problem can be divided into two steps: (1) select an unweighted automaton A that will serve as the *topology* of the target automaton and (2) weight the automaton A to form our weighted approximaton $\hat{A}$. The main goal of this paper is the latter determination of the automaton's weighting in the approximation. If the topology is not known, we suggest a few techniques for inferring topology later in the introduction.

We will now give some very simple topology examples to illustrate the approximation idea. In Section 5 we will give larger-scale examples. Consider the unweighted automaton A in Figure 1 that was designed for what you might say to set an alarm. To use this in an application such as speech recognition, we would want to weight the automaton with some reasonable probabilities for the alternatives. In the absence of data specifically for this scenario, we can fall back on some available background language model M, trained on a large suitable corpus. In particular, we can use the conditional distribution

$$p_m[x_1 \dots x_n | x_i \dots x_n \in \mathcal{L}(A)] = \frac{p_m[x_1 \dots x_n] 1_{x_i \dots x_n \in \mathcal{L}(A)}}{\sum_{x_i \dots x_n \in \mathcal{L}(A)} p_m[x_1 \dots x_n]}, \tag{1}$$

where $\mathcal{L}(A)$ is the regular language accepted by the automaton A, as our source distribution S. We then use the unweighted automaton A as our target topology.

If M is represented as a WFA, our approximation will in general give a different solution than forming the finite-state intersection with A and *weight-pushing* to normalize the result [40, 41]. Our approximation has the same states as A whereas weight-pushed $M \cap A$ has $O(|M||A|)$ states and does not approximate the source distribution. Instead, weight-pushed $M \cap A$ is an exact WFA representation of the distribution in Equation 1.

In some applications, the topology may be unknown. In such cases, one choice is to build a k-gram deterministic finite automaton (DFA) topology from a corpus drawn from S [4]. This could be from an existing corpus or from random samples drawn from S. Figure 2a shows a trigram topology for the very simple corpus *aab*. Figure 2b shows an an alternative topology that allows *skip*-grams. Both of these representations make use of failure transitions. These allow modeling strings unseen in the corpus in a compact way by failing or *backing-off* to states that correspond to lower-order histories. Such models can be made more elaborate if some transitions represent classes, such as names or numbers, that are themselves represented by sub-automata. As mentioned previously, we will mostly assume we have a topology either pre-specified or inferred by some means and focus on how to weight that topology to best approximate the source distribution.

The paper is organized as follows. In Section 2 we review previous work in this area. In Section 3 we give the theoretical formulation of the problem and the minimum KL divergence approximation.

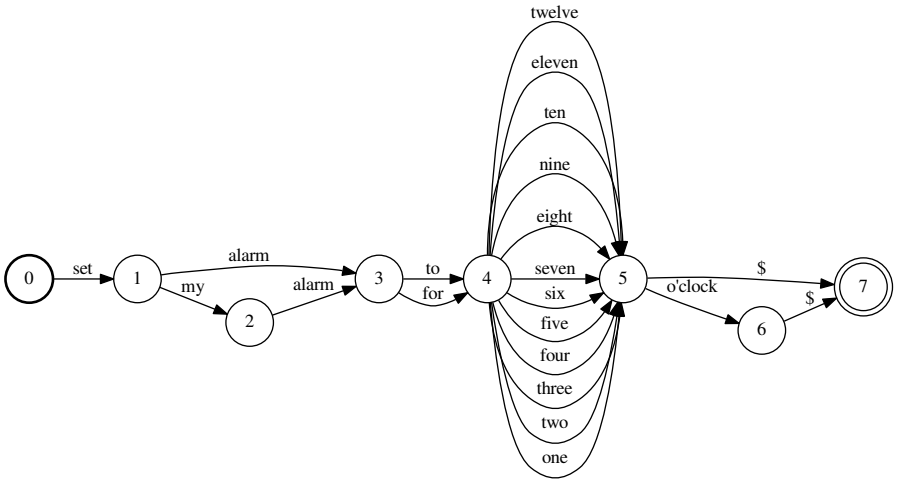


Figure 1: An unweighted automaton that specifies what you might say to set an alarm. The initial state is the bold circle and the final state is double circle. By convention, we terminate all accepted strings with the symbol \$.

In Section 4 we present algorithms to compute that solution. One algorithm is for the case that the source itself is finite-state. A second algorithm is for the case when it is not and involves a sampling approach. In Section 5 we show experiments using the approximation. Finally, in Section 6 we discuss the results and offer conclusions.

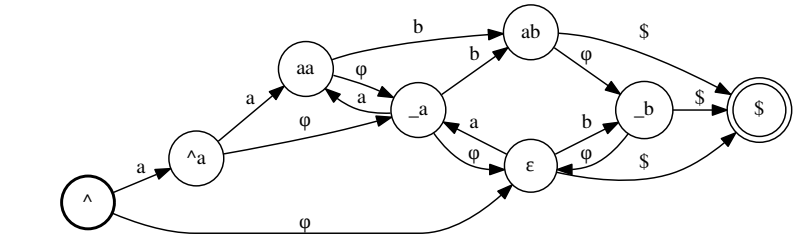
2 Related works

In this section we will review methods both for inferring unweighted finite-state models from data and estimating the weight distribution as well in the weighted case. We start with the unweighted case.

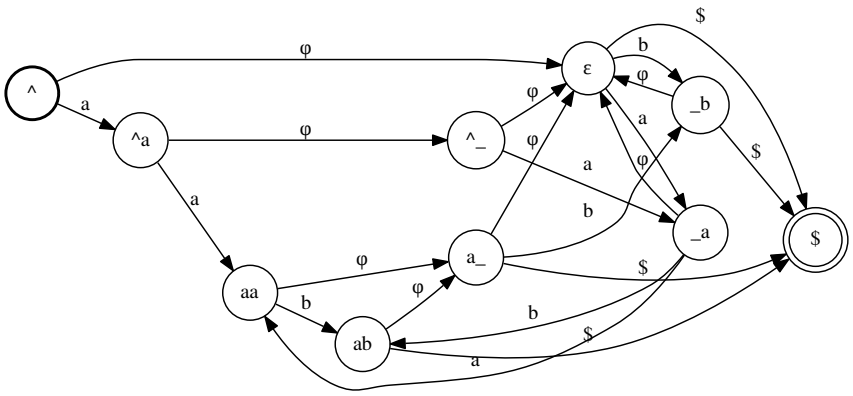
There is a long history of unweighted finite-state model inference [45, 18]. Gold [28] showed that an arbitrary regular set L can not be learned, *identified in the limit*, strictly from the presentation of a sequence of positive examples that eventually includes each string in L. This has led to several alternative lines of attack.

One approach is to include the negative examples in the sequence. Given such a *complete sample*, there are polynomial-time algorithms that identify a regular set in the limit [27]. For example, a *prefix tree* of the positive examples can be built and then states can be merged so long as they do not cause a negative example to be accepted [43, 22]. Another approach is to train a recurrent neural network (RNN) on the positive and negative examples and then extract a finite automaton by quantizing the continuous state space of the RNN [26, 33].

A second approach is to assume a teacher is available that determines not only if a string is a positive or negative example but also if the language of the current hypothesized automaton equals



(a)



(b)

Figure 2: Topology examples derived from the corpus *aab*. States are labeled with the context that is remembered, ∧ denotes the initial context, ϵ the empty context, \$ the final context (and terminates accepted strings), and _ matches any symbol in a context. (a) 3-gram topology: failure transitions, labeled with φ, implement backoff from histories xy to $\_y$ to ϵ. (b) *skip*-gram topology: failure transitions implement backoff instead from histories xy to $x\_$.

L or if not, provides a counterexample. In this case the minimal n-state DFA corresponding to L can be learned in time polynomial in n [7]. Weiss et al.[53] apply this method in a DFA extraction from an RNN.

A third approach is to assume a probability distribution over the (positive only) samples. With some reasonable restrictions on the distribution, such as the probabilities are generated from a weighted automaton A with $L = \mathcal{L}(A)$, then L is identifiable in the limit with 'high probability' [8, 46].

There have been a variety of approaches for estimating weighted automata. A variant of the prefix tree construction can be used that merges states with sufficiently similar suffix distributions, estimated from source frequencies [14, 15]. Approaches that produce (possibly highly) nondeterministic results include the Expectation-Maximization (EM) algorithm [20] applied to a fully connected Hidden Markov models or spectral methods applied to automata [11, 10]. Eisner [25] describes an algorithm for estimating probabilities in a finite-state *transducer* from data using EM-based methods.

For approximating neural network (NN) models as WFAs, Deoras et al.[21] used an RNN LM to generate samples that they then train an k-gram LM. Arisoy et al.[9] used deep neural network

(DNN) models of different orders to successively build and prune a k-gram LM with each new order constrained by the previous order. Adel et al.[1] also trained DNNs of different orders, built a k-gram LM on the same data to obtain a topology and then transferred the DNN probabilities of each order onto that k-gram topology. Tiňo and Vojtek [52] quantized the continuous state space of an RNN and then estimated the transition probabilities from the RNN. Lecorve and Motlicek [36] quantized the hidden states in an LSTM to form a finite-state model and then used an entropy criterion to backoff to low-order k-grams to limit the number of transitions.

Our paper is distinguished in several respects from previous work. First, our general approach does not depend on the form the source distribution although we specialize our algorithms for (known) finite-state sources with an efficient direct construction and for other sources with an efficient sampling. Second, our targets are a wide class of deterministic automata with failure transitions. These are considerably more general than k-gram models but retain the efficiency of determinism and the compactness failure transitions allow, especially important in applications with large alphabets like language modeling. Third, we show that our approximation searches for the minimal KL divergence between the source and target distributions, given a fixed target topology provided by the application or some earlier computation.

3 Theoretical analysis

3.1 Probabilistic models

Let Σ be a finite alphabet. Let $x_i^n \in \Sigma^*$ denote the string $x_i x_{i+1} \ldots x_n$ and $x^n \triangleq x_1^n$. A probabilistic model p over Σ is a probabilistic distribution over the next symbol x_n, given the previous symbols x^{n-1}

$$\sum_{x \in \Sigma} p(x_n = x | x^{n-1}) = 1 \text{ and } p(x_n = x | x^{n-1}) \geq 0, \forall x \in \Sigma.^{1}$$

Without loss of generality, we assume that the model maintains an internal state q and updates it after observing the next symbol.[2] Furthermore, the probability of the subsequent state just depends on the state q

$$p(x_{i+1}^n | x^i) = p(x_{i+1}^n | q(x^i)),$$

for all i, n, x^i, x_{i+1}^n, where $q(x^i)$ is the state that the model has reached after observing sequence x^i. Let $Q(p)$ be the set of possible states. Let the language $\mathcal{L}(p) \subseteq \Sigma^*$ defined by the distribution p be

$$\mathcal{L}(p) \triangleq \{x^n \in \Sigma^* : p(x^n) > 0 \text{ and } x_n = \$ \text{ and } x_i \neq \$ \, , i < n\}. \tag{2}$$

The symbol $\$$ is used as a stopping criterion. Further for all $x^n \in \Sigma^*$, $p(x_n | x^{n-1} : x_{n-1} = \$ \,) = 0$.

The KL divergence between two models p_s and p_a is given by

$$D(p_s || p_a) = \sum_{x^n} p_s(x^n) \log \frac{p_s(x^n)}{p_a(x^n)}, \tag{3}$$

where for notational simplicity, we adopt the notion $0/0 = 1$ and $0 \log(0/0) = 0$ throughout the paper. Note that for the KL divergence to be finite, we need $\mathcal{L}(p_s) \subseteq \mathcal{L}(p_a)$. We first reduce the KL divergence between two models as follows (cf. [13, 19]).

[1] We define $x^0 \triangleq \epsilon$, the empty string, and adopt $p(\epsilon) = 0$.

[2] In the most general case, $q(x^n) = x^n$.

Lemma 1. *If $\mathcal{L}(p_s) \subseteq \mathcal{L}(p_a)$, then*

$$D(p_s||p_a) = \sum_{q_a \in Q_a} \sum_{x \in \Sigma} c(x, q_a) \log \frac{p_s(x|q_s)}{p_a(x|q_a)}. \tag{4}$$

where

$$c(x, q_a) = \sum_{q_s \in Q_s} \sum_{i=0}^{\infty} \sum_{x^i} p_s(x^i : q_s(x^i) = q_s, q_a(x^i) = q_a)\, p_s(x|q_s) \tag{5}$$

and does not depend on p_a.

Proof.

$$\begin{aligned}
D(p_s||p_a) &= \sum_{n=1}^{\infty} \sum_{x^n} p_s(x^n) \log \frac{p_s(x^n)}{p_a(x^n)} \\
&= \sum_{n=1}^{\infty} \sum_{x^n} p_s(x^n) \sum_{i=1}^{n} \log \frac{p_s(x_i|x^{i-1})}{p_a(x_i|x^{i-1})} \\
&= \sum_{n=1}^{\infty} \sum_{i=1}^{n} \sum_{x^n} p_s(x^n) \log \frac{p_s(x_i|x^{i-1})}{p_a(x_i|x^{i-1})} \\
&= \sum_{n=1}^{\infty} \sum_{i=1}^{n} \sum_{x^n} p_s(x^{i-1}) p_s(x_i|x^{i-1}) p_s(x_{i+1}^n|x^i) \log \frac{p_s(x_i|x^{i-1})}{p_a(x_i|x^{i-1})} \\
&= \sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1}) \sum_{x_i} p_s(x_i|x^{i-1}) \log \frac{p_s(x_i|x^{i-1})}{p_a(x_i|x^{i-1})} \cdot \sum_{n \geq i} \sum_{x_{i+1}^n} p_s(x_{i+1}^n|x^i) \\
&= \sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1}) \sum_{x_i} p_s(x_i|x^{i-1}) \log \frac{p_s(x_i|x^{i-1})}{p_a(x_i|x^{i-1})}.
\end{aligned}$$

By definition, the probability of the next symbol conditioned on the past just depends on the state. Hence grouping terms corresponding to same states both in s and t yields,

$$\begin{aligned}
&\sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1}) \sum_{x_i} p_s(x_i|x^{i-1}) \log \frac{p_s(x_i|x^{i-1})}{p_a(x_i|x^{i-1})} \\
&= \sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1}) \sum_{x_i} p_s(x_i|q_s(x^{i-1})) \log \frac{p_s(x_i|q_s(x^{i-1}))}{p_a(x_i|q_a(x^{i-1}))} \\
&= \sum_{q_s \in Q_s} \sum_{q_a \in Q_a} \sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1} : q_s(x^{i-1}) = q_s, q_a(x^{i-1}) = q_a) \sum_{x_i} p_s(x_i|q_s) \log \frac{p_s(x_i|q_s)}{p_a(x_i|q_a)} \\
&= \sum_{q_s \in Q_s} \sum_{q_a \in Q_a} \sum_{x_i} c(x_i, q_s) \log \frac{p_s(x_i|q_s)}{p_a(x_i|q_a)}.
\end{aligned}$$

Replacing x_i by x yields the lemma. □

3.2 Weighted finite automata

A weighted finite automaton $A = (\Sigma, Q, E, i, f)$ over $\mathbb{R}_+$ is given by a finite alphabet Σ, a finite set of states Q, a finite set of transitions $E \subseteq Q \times \Sigma \times \mathbb{R}_+ \times Q$, an initial state $i \in Q$ and a final state $f \in Q$. A transition $e = (p[e], \ell[e], w[e], n[e]) \in E$ represents a move from the *source* or *previous* state $p[e]$ to the *destination* or *next* state $n[e]$ with the *label* $\ell[e]$ and *weight* $w[e]$. The transitions with source state q are denoted by $E[q]$ and the labels of those transitions as $L[q]$.

A deterministic WFA has at most one transition with a given label leaving each state. An *unweighted (finite) automaton* is a WFA that satisfies $w[e] = 1, \forall e \in E$. A *probabilistic (or stochastic) WFA* satisfies

$$\sum_{e \in E[q]} w[e] = 1 \text{ and } w[e] \geq 0, \quad \forall q \in Q - \{f\}.$$

Transitions e_1 and e_2 are *consecutive* if $n[e_i] = p[e_{i+1}]$. A path $\pi = e_1 \cdots e_n \in E^*$ is a finite sequence of consecutive transitions. The source state of a path we denote by $p[\pi]$ and the destination state by $n[\pi]$. The label of a path is the concatenation of its transition labels $\ell[\pi] = \ell[e_1] \cdots \ell[e_n]$. The weight of a path is obtained by multiplying its transition weights $w[\pi] = w[e_1] \times \cdots \times w[e_n]$. For a non-empty path, the i-th transition is denoted by π_i.

$P(q, q')$ denotes the set of all paths in A from state q to q'. We extend this to sets in the obvious way $P(q, R)$ denotes the set of all paths from state q to $q' \in R$ and so forth. A path π is successful if it is in $P(i, f)$ and in that case the automaton is said to accept the input string $\alpha = \ell[\pi]$.

The language accepted by an automaton A is the regular set $\mathcal{L}(A) = \{\alpha \in \Sigma^* : \alpha = \ell[\pi], \pi \in P(i, f)\}$. The weight of $\alpha \in \mathcal{L}(A)$ assigned by the automaton is $A(\alpha) = \Sigma_{\pi \in P(i,f):\, \ell[\pi] = \alpha} w[\pi]$. Similar to Equation 2, we assume a symbol $\$ \in \Sigma$ such that

$$\mathcal{L}(A) \subseteq \{x^n \in \Sigma^* : x_n = \$ \text{ and } x_i \neq \$, i < n\}.$$

Thus all successful paths are terminated by the symbol $\$$.

For a symbol $x \in \Sigma$ and a state $q \in Q$ of a deterministic, probabilistic WFA A, define a distribution $p_a(x|q) \triangleq w$ if $(q, x, w, q') \in E$ and $p_a(x|q) \triangleq 0$ otherwise. Then p_a is a probabilistic model over Σ as defined in the previous section. If $A = (\Sigma, Q, E, i, f)$ is an unweighted deterministic automaton, we denote by $\mathscr{P}(A)$ the set of all probabilistic models p_a representable as a weighted WFA $\hat{A} = (\Sigma, Q, \hat{E}, i, f)$ with the same topology as A where $\hat{E} = \{(q, x, p_a(x|q), q') : (q, x, 1, q') \in E\}$.

Given an unweighted deterministic automaton A, our goal is to find the target distribution $p_a \in \mathscr{P}(A)$ that has the minimum KL divergence from our source probability model p_s.

Lemma 2. *If $\mathcal{L}(p_s) \subseteq \mathcal{L}(A)$, then*

$$\underset{p_a \in \mathscr{P}(A)}{\operatorname{argmin}} D(p_s||p_a) = \tilde{p}(x|q_a) \triangleq \frac{c(x, q_a)}{\sum_{x' \in \Sigma} c(x', q_a)} \tag{6}$$

Proof. The KL divergence between the distributions p_s and p_a is

$$D(p_s||p_a) = \sum_{q_a \in Q_a} \sum_{x \in \Sigma} c(x, q_a) \log \frac{p_s(x|q_s)}{p_a(x|q_a)}.$$

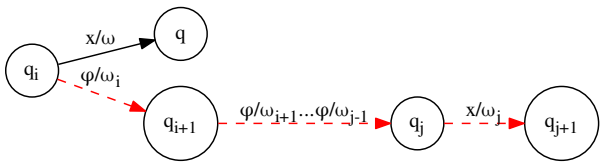


Figure 3: The (dashed red) path $e_i = (q_i, \varphi, \omega_i, q_{i+1})$ to $e_j = (q_j, x, \omega_j, q_{j+1})$ is disallowed since x can be read already on $e = (q_i, x, \omega, q)$.

Minimizing the above is the same as minimizing

$$\sum_{q_a \in Q_a} \sum_{x \in \Sigma} c(x, q_a) \log \frac{\tilde{p}(x|q_a)}{p_a(x|q_a)}.$$

The above quantity can be rewritten as

$$\sum_{q_a \in Q_a} \sum_{x' \in \Sigma} c(x', q_a) \Big\{ \sum_{x \in \Sigma} \tilde{p}(x|q_a) \log \frac{\tilde{p}(x|q_a)}{p_a(x|q_a)} \Big\}.$$

Since the KL divergence between two distributions is always non-negative, the quantity in braces is always non-negative and is 0 if and only if $p_a(x|q_a) = \tilde{p}(x|q_a)$. Since $\mathcal{L}(p_s) \subseteq \mathcal{L}(A)$, it follows that $\tilde{p} \in \mathscr{P}(A)$. □

3.3 Weighted finite automata with failure transitions

A *weighted finite automaton with failure transitions* (φ-*WFA*) $A = (\Sigma, Q, E, i, f)$ is a WFA extended to allow a transition to have a special *failure* label denoted by φ. Then $E \subseteq Q \times (\Sigma \cup \{\varphi\}) \times \mathbb{R}_+ \times Q$.

A φ transition does not add to a path label; it consumes no input. However it is followed only when the input can not be read immediately. Specifically, a path $e_1 \cdots e_n$ in a φ-WFA is *disallowed* if it contains a subpath $e_i \cdots e_j$ such that $\ell[e_k] = \varphi$ for $i \leq k < j$ and there is another transition $e \in E$ such that $p[e_i] = p[e]$ and $\ell[e_j] = \ell[e] \in \Sigma$ (see Figure 3). Since the label $x = l[e_j]$ can be read on e, we do not follow the failure transitions to read it on e_j as well.

We use $P^*(q, q') \subseteq P(q, q')$ to denote the set of (not dis-) allowed paths from state q to q' in a φ-WFA. This again extends to sets in the obvious way. A path π is successful in a φ-WFA if $\pi \in P^*(i, F)$ and only in that case is the input string $\alpha = \ell[\pi]$ accepted.

The language accepted by the φ-automaton A is the regular set $\mathcal{L}(A) = \{\alpha \in \Sigma^* : \alpha = \ell[\pi], \pi \in P^*(i, f)\}$. The weight of $\alpha \in \Sigma^*$ assigned by the automaton is $A(\alpha) = \Sigma_{\pi \in P^*(i,f):\, \ell[\pi] = \alpha} w[\pi]$. We assume each string in $\mathcal{L}(A)$ is terminated by the symbol \$ as before. We also assume there are no φ-labeled cycles and there is at most one exiting failure transition per state.

We express the φ-*extended transitions* leaving q as

$$E^*[q] = \Big\{ (q, x, \omega, q') : \quad \pi \in P^*(q, Q), x = \ell[\pi] = \ell[\pi_{|\pi|}] \in \Sigma, \omega = w[\pi], q' = n[\pi] \Big\}.$$

This is a set of (possibly new) transitions (q, x, ω, q'), one for each allowed path from source state q to destination state q' with optional leading failure transitions and a final x-labeled transition. Denote the labels of $E^*[q]$ by $L^*[q]$.

A *probabilistic (or stochastic) φ-WFA* satisfies

$$\sum_{e \in E^*[q]} w[e] = 1 \text{ and } w[e] \geq 0, \quad \forall q \in Q - \{f\}.$$

A deterministic φ-WFA is *backoff-complete* if a failure transition from state q to q' implies $L[q] \cap \Sigma \subseteq L[q'] \cap \Sigma$. Further, if $\varphi \notin L[q']$, then the containment is strict: $L[q] \cap \Sigma \subset L[q'] \cap \Sigma$. In other words, if a symbol can be read immediately from a state q it can also be read from a state failing (*backing-off*) from q and if q' does not have a backoff arc, then at least one additional label can be read from q' that cannot be read from q. For example, both topologies depicted in Figure 2 have this property. We adopt this topology for our target automata since it will simplify our analysis, make our algorithms efficient and is commonly found in applications.

For a symbol $x \in \Sigma$ and a state $q \in Q$ of a deterministic, probabilistic φ-WFA A, define $p_a^*(x|q) \triangleq w$ if $(q, x, w, q') \in E^*[q]$ and $p_a^*(x|q) \triangleq 0$ otherwise. Then p_a^* is a probabilistic model over Σ as defined in Section 3.1. Note the distribution p_a^* at a state q is defined over the $\varphi-$extended transitions $E^*[q]$ where p_a in the previous section is defined over the transitions $E[q]$. It is convenient to define a companion distribution $p_a \in P(A)$ to p_a^* as follows:[3] given a symbol $x \in \Sigma \cup \{\varphi\}$ and state $q \in Q$, define $p_a(x|q) \triangleq p_a^*(x|q)$ when $x \in L[q] \cap \Sigma$, $p_a(\varphi|q) \triangleq \sum_{x \in L[q] \cap \Sigma} 1 - p_a^*(x|q)$, and $p_a(x|q) \triangleq 0$ otherwise. The companion distribution is thus defined solely over the transitions $E[q]$.

When $A = (\Sigma, Q, E, i, f)$ is an unweighted deterministic, backoff-complete φ-WFA, we denote by $\mathscr{P}^*(A)$ the set of all probabilistic models p_a^* representable as a weighted φ-WFA $\hat{A} = (\Sigma, Q, \hat{E}, i, f)$ of same topology as A with

$$\begin{aligned}\hat{E} =& \{(q, x, p_a(x|q), q') : (q, x, 1, q') \in E, x \in \Sigma\} \cup \\ & \{(q, \varphi, \alpha(q, q'), q') : (q, \varphi, 1, q') \in E\}\end{aligned}$$

where $p_a \in P(A)$ is the companion distribution to p_a^* and $\alpha(q, q') = p_a(\varphi|q)/d(q, q')$ is the weight of the failure transition from state q to q' with

$$d(q, q') = 1 - \sum_{x \in L[q] \cap \Sigma} p_a(x|q'). \tag{7}$$

Note we have specified the weights on the automaton that represents $p_a^* \in P^*(A)$ entirely in terms of the companion distribution $p_a \in P(A)$, thanks the the backoff-complete property.

Conversely, each distribution $p_a \in \mathscr{P}(A)$ can be associated to a distribution $p_a^* \in \mathscr{P}^*(A)$ given a deterministic, backoff-complete φ-WFA A. First extend $\alpha(q, q')$ to any failure path as follows. Denote a failure path from state q to q' by $\pi_\varphi(q, q')$. Then define

$$\alpha(q, q') = \prod_{e \in \pi_\varphi(q, q')} \frac{p_a(\varphi|p[e])}{d(p[e], n[e])} \tag{8}$$

where this quantity is taken to be 1 when the failure path is empty ($q = q'$). Finally define

$$p_a^*(x|q) = \begin{cases} \alpha(q, q^x) p_a(x|q^x), & x \in L^*[q] \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

[3] The meaning of $P(A)$ when A is φ-WFA is to interpret it as a WFA with the failure labels as regular symbols.

where for $x \in L^*[q]$, q^x signifies the first state q' on a φ-labeled path in A from state q for which $x \in L[q']$.

For (8) to be well-defined, we need $d(p[e], n[e]) > 0$. To ensure this condition, we restrict $\mathscr{P}(A)$ to contain distributions such that $p_a(x|q) \geq \epsilon$ for each $x \in L[q]$.[4] Let $\mathscr{P}^*(A)$ denote the set of distribution p_a^* that have a companion distribution in $\mathscr{P}(A)$. Given an unweighted deterministic, backoff-complete, automaton A, our goal is to find the target distribution $p_a^* \in \mathscr{P}^*(A)$ that has the minimum KL divergence from our source probability model p_s.

Lemma 3. *Assume $\mathcal{L}(p_s) \subseteq \mathcal{L}(A)$. Let $p^* = \tilde{p}(x|q_a)$ from Lemma 2. If $\mathcal{L}(p^*) \subseteq \mathcal{L}(A)$ and $p^* \in \mathscr{P}^*(A)$ then*

$$p^* = \operatorname*{argmin}_{p_a^* \in \mathscr{P}^*(A)} D(p_s||p_a^*)$$

Proof. This follows immediately from Lemma 2. □

The requirement that the p^* of Lemma 3 is in $\mathscr{P}^*(A)$ will be true if, for instance, the target has no failure transitions or if the source and target are both φ-WFAs with the same topology and failure transitions. In general, this requirement can not be assured and directly minimizing over $\mathscr{P}^*(A)$ is hard. Hence, we restate our goal in terms of the companion distribution $p_a \in \mathscr{P}(A)$. Let $B_n(q)$ be the set of states in A that back-off to state q in n failure transitions and let $B(q) = \sum_{n=0}^{|Q_a|} B_n(q)$.

Lemma 4. *If $\mathcal{L}(p_s) \subseteq \mathcal{L}(A)$ then*

$$\operatorname*{argmin}_{p_a^* \in \mathscr{P}^*(A)} D(p_s||p_a^*) = \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \sum_{q \in Q_a} \left\{ \sum_{x \in L[q]} C(x, q) \log p_a(x|q) - \sum_{q_0 \in B_1(q)} C(\varphi, q_0) \log d(q_0, q) \right\}$$

where

$$C(x, q) = \sum_{q_a \in B(q)} c(x, q_a) \mathbf{1}_{q = q_a^x}, \quad x \in \Sigma \tag{10}$$

$$C(\varphi, q) = \sum_{q_a \in B(q)} \sum_{x \in \Sigma} c(x, q_a) \mathbf{1}_{x \notin L[q]} \tag{11}$$

and do not depend on p_a.

Proof. From Lemma 1, Equation 9 and the previously shown 1:1 correspondence between each distribution $p_a^* \in P^*(A)$ and its companion distribution $p_a \in P(A)$

$$\begin{aligned}
\operatorname*{argmin}_{p_a^* \in \mathscr{P}^*(A)} D(p_s||p_a^*) &= \operatorname*{argmin}_{p_a \in \mathscr{P}(A)} \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \frac{p_s(x|q_s)}{\alpha(q_a, q_a^x) p_a(x|q_a^x)} \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \alpha(q_a, q_a^x) p_a(x|q_a^x) \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \prod_{e \in \pi_\varphi(q_a, q_a^x)} \frac{p_a(\varphi|p[e])}{d(p[e], n[e])} p_a(x|q_a^x) \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \left\{ A_x + A_\varphi - A_d \right\}
\end{aligned} \tag{12}$$

[4] For brevity, we do not include ϵ in the notation of $\mathscr{P}(A)$.

where we distribute the factors inside the logarithm in Equation 12 as follows:

$$\begin{aligned} A_x &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log p_a(x|q_a^x) \\ &= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in L[q] \cap \Sigma} c(x, q_a) \mathbf{1}_{q=q_a^x} \log p_a(x|q) \\ &= \sum_{q \in Q_a} \sum_{x \in L[q] \cap \Sigma} C(x, q) \log p_a(x|q). \end{aligned} \tag{13}$$

Equation 13 follows from $q = q_a^x$ implying $q_a \in B(q)$.

$$\begin{aligned} A_\varphi &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \prod_{e \in \pi_\varphi(q_a, q_a^x)} p_a(\varphi|p[e]) \\ &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \sum_{e \in \pi_\varphi(q_a, q_a^x)} \log p_a(\varphi|p[e]) \\ &= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in L^*[q_a]} c(x, q_a) \sum_{e \in \pi_\varphi(q_a, q_a^x)} \mathbf{1}_{q=p[e]} \log p_a(\varphi|q) \\ &= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in \Sigma} c(x, q_a) \mathbf{1}_{x \notin L[q]} \log p_a(\varphi|q) \\ &= \sum_{q \in Q_a} C(\varphi, q) \log p_a(\varphi|q) \end{aligned} \tag{14}$$

Equation 14 follows from $e \in \pi_\varphi(q_a, q_a^x)$ implying $x \notin p[e]$.

$$\begin{aligned} A_d &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \prod_{e \in \pi_\varphi(q_a, q_a^x)} d(p[e], n[e]) \\ &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \sum_{e \in \pi_\varphi(q_a, q_a^x)} \log d(p[e], n[e]) \\ &= \sum_{q \in Q_a} \sum_{q_a \in B(q_0)} \sum_{q_0 \in B_1(q)} \sum_{x \in L^*[q_a]} c(x, q_a) \sum_{e \in \pi_\varphi(q_a, q_a^x)} \mathbf{1}_{q_0=p[e]} \log d(q_0, q) \\ &= \sum_{q \in Q_a} \sum_{q_a \in B(q_0)} \sum_{q_0 \in B_1(q)} \sum_{x \in \Sigma} c(x, q_a) \mathbf{1}_{x \notin L[q_0]} \log d(q_0, q) \\ &= \sum_{q \in Q_a} \sum_{q_0 \in B_1(q)} C(\varphi, q_0) \log d(q_0, q) \end{aligned}$$

Substituting these results into Equation 12 proves the lemma. □

The quantity in braces in the statement of Lemma 4 depends on the distribution p_a only at state q so the minimum KL divergence $D(p_s||p_a^*)$ can be found by maximizing that quantity independently for each state.

4 Algorithms

Approximating a probabilistic source algorithmically as a weighted finite automaton requires two steps: (1) compute the quantity $c(x, q_a)$ found in Lemma 2 or $C(x, q)$ in Lemma 4 and (2) use

this quantity to find the minimum KL divergence solution. The first step, which we will refer to as *counting*, is covered in the next section and the KL divergence minimization step is covered afterwards.

4.1 Counting

How the counts are computed will depend on the form of the source and target models. We break this down into several cases.

4.1.1 WFA source and target

When the source and target models are represented as WFAs we compute $c(x, q_a)$ from Lemma 2. From Equation 5 this can be written as

$$c(x, q_a) = \sum_{q_s \in Q_s} \gamma(q_s, q_a) p_s(x|q_s) \tag{15}$$

where

$$\gamma(q_s, q_a) = \sum_{i=0}^{\infty} \sum_{x^i} p_s(x^i : q_s(x^i) = q_s, q_a(x^i) = q_a).$$

The quantity $\gamma(q_s, q_a)$ can be computed as

$$\gamma(q_s, q_a) = \sum_{\pi \in P_{S \cap A}((i_s, i_a),(q_s, q_a))} w[\pi]$$

where $S \cap A$ is the weighted finite-state intersection of automata S and A [40]. The above summation over this intersection is the (generalized) *shortest distance* from the initial state to a specified state computed over the *positive real semiring* [39, 6]. Algorithms to efficiently compute the intersection and shortest distance are available in *OpenFst* [5], an open-source weighted finite automata library.

Then from Equation 15 we can form the sum

$$c(x, q_a) = \sum_{((q_s, q_a), x, w, (q'_s, q'_a)) \in E_{S \cap A}} \gamma(q_s, q_a) w \tag{16}$$

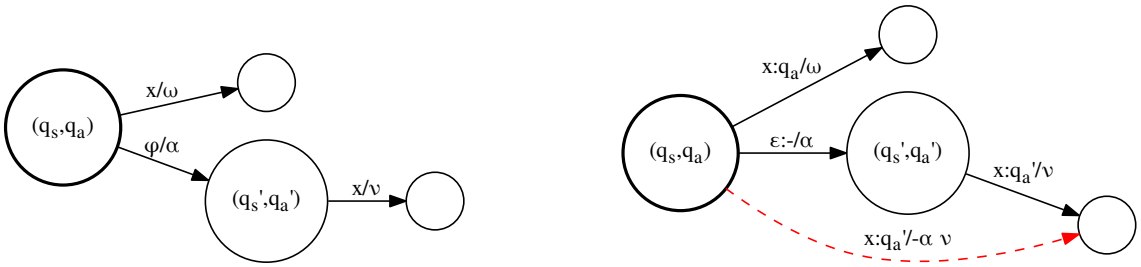
Equation 16 is the weighted count of the paths in $S \cap A$ that begin at the initial state and end in any transition leaving a state (q_s, q_a) labeled with x.

4.1.2 φ-WFA source and target

When the source and target models are represented as φ-WFAs we compute $C(x, q_a)$ from Lemma 4. From Equation 10 and the previous case this can be written as

$$C(x, q) = \sum_{q_a \in B(q)} \sum_{q_s \in Q_s} \gamma(q_s, q_a) p_s(x|q_s) \mathbf{1}_{q = q_a^x}, \quad x \in \Sigma \tag{17}$$

To compute this quantity we first form $S \cap A$ using an efficient φ-WFA intersection that compactly retains failure transitions in the result as described in Allauzen and Riley [6]. Equation 17 is the



$S \cap A$ T

Figure 4: A φ-WFA is transformed into an equivalent WFA by replacing each failure transition by an ϵ-transition. To compensate for the formerly disallowed paths, new (dashed red) negatively-weighted transitions are added. The result is promoted to a transducer T with the output label used to keep track of the source state in A of the compensated positive transition.

weighted count of the paths in $S \cap A$ allowed by the failure transitions that begin at the initial state and end in any transition leaving a state (q_s, q) labeled with x.

We can simplify this computation by the following transformation. First we convert $S \cap A$ to an equivalent WFA by replacing each failure transition with an epsilon transition and introducing a negatively-weighted transition to compensate for formerly disallowed paths [6]. The result is then promoted to a transducer T with the output label used to keep track of the source state in A of the compensated positive transition (see Figure 4).[5]

Then

$$C(x, q) = \sum_{((q_s,q_a),x,q,w,(q_s',q_a'))\in E_T} \gamma_T(q_s, q_a)w, \quad x \in \Sigma \tag{18}$$

where $e = (p[e], il[e], ol[e], w[e], n[e])$ is a transition in T and $\gamma_T(q_s, q)$ is the shortest distance from the initial state to (q_s, q_a) in T computed over the *real semiring* as described in Allauzen and Riley [6]. Equation 18 is the weighted count of all paths in $S \cap A$ that begin at the initial state and end in any transition leaving a state (q_s, q) labeled with x minus the weighted count of those paths that are disallowed by the failure transitions.

Finally, we compute $C(\varphi, q)$ as follows. The count mass entering a state must equal the count mass leaving a state

$$\sum_{(q_a,x,1,q)\in A} C(x, q) = \sum_{(q,x',1,q_a)\in A} C(x', q).$$

Thus

$$C(\varphi, q) = \sum_{(q_a,x,1,q)\in A} C(x, q) - \sum_{(q,x',1,q_a)\in A} C(x', q), \quad x' \in \Sigma$$

This quantity can be computed iteratively in the topological order of states with respect to the φ-labeled transitions.

[5] The construction illustrated in Figure 4 is sufficient when $S \cap A$ is acyclic. In the cyclic case a slightly modified construction is needed to ensure convergence in the shortest distance calculation [6]

4.1.3 Arbitrary source and φ-WFA target

In some cases, the source is a distribution with possibly infinite states, e.g., LSTMs. For these sources, computing $C(x, q)$ can be computationally intractable as (17) requires a summation over all possible states in the source machine, Q_s. We propose to use a sampling approach to approximate $C(x, q)$ for these cases. Let $X(1), X(2), \ldots, X(m)$ be independent random samples from p_s. Instead of $C(x, q)$, we propose to use

$$\hat{C}(x, q) = \sum_{q_a \in B(q)} \sum_{q_s \in Q_s} \hat{\gamma}(q_s, q_a) p_s(x|q_s) \mathbf{1}_{q=q_a^x}, \quad x \in \Sigma,$$

where

$$\hat{\gamma}(q_s, q_a) = \frac{1}{m} \sum_{j=1}^{m} \sum_{i \geq 0} \mathbf{1}_{q_s(x^i(j))=q_s, q_a(x^i(j))=q_a}.$$

Observe that in expectation,

$$\begin{aligned} \mathbb{E}[\hat{\gamma}(q_s, q_a)] &= \frac{1}{m} \sum_{j=1}^{m} \sum_{i \geq 0} \mathbb{E}[\mathbf{1}_{q_s(x^i(j))=q_s, q_a(x^i(j))=q_a}] \\ &= \sum_{i \geq 0} p_s(x^i : q_s(x^i) = q_s, q_a(x^i) = q_a), \end{aligned}$$

and hence $\hat{\gamma}(q_s, q_a)$ is an unbiased, asymptotically consistent estimator of $\gamma(q_s, q_a)$. Given $\hat{C}(x, q)$, we compute $C(\varphi, q)$ similar to the previous section.

4.2 KL divergence minimization

4.2.1 WFA target

When the target topology is a WFA, we use $c(x, q_a)$ from the previous section and Lemma 2 to immediately find the minimum KL divergence solution.

4.2.2 φ-WFA target

When the target topology is a φ-WFA, Lemma 3 can be applied in some circumstances to find the minimum KL divergence solution but not in general. However, as noted before, the quantity in braces in the statement of Lemma 4 depends on the distribution p_a only at state q so the minimum KL divergence $D(p_s||p_a^*)$ can be found by maximizing that quantity independently for each state.

Fix a state q and let $y_x \triangleq p_a(x|q)$ for $x \in L[q]$ and let $\mathbf{y} \triangleq [y_x]_{x \in L[q]}$[6]. Then our goal reduces to

$$\operatorname*{argmax}_{\mathbf{y}} \sum_{x \in L[q]} C(x, q) \log y_x - \sum_{q_0 \in B_1(q)} C(\varphi, q_0) \log \big(1 - \sum_{x \in L[q_0] \cap \Sigma} y_x\big). \tag{19}$$

subject to the constraints $y_x \geq \epsilon$ for $x \in L[q]$ and $\sum_{x \in L[q]} y_x = 1$.

[6] We fix some total order on $\Sigma \cup \{\varphi\}$ so that $\mathbf{y}$ is well-defined.

This is a difference of two concave functions in $\mathbf{y}$ since $\log(f(\mathbf{y}))$ is concave for any linear function $f(\mathbf{y})$, the $C(x,q)$ are always non-negative and the sum of concave functions is also concave. We give a *DC programming* solution to this optimization in Section 4 [31]. Let

$$\Omega = \{\mathbf{y} : \forall x, y_x \geq \epsilon, \sum_{x \in L(q)} y_x \leq 1\},$$

and let $u(\mathbf{y}) = \sum_{x \in L[q]} C(x,q) \log y_x$ and $v(\mathbf{y}) = \sum_{q_0 \in B_1(q)} C(\varphi, q_0) \log \left(1 - \sum_{x \in L[q_0] \cap \Sigma} y_x\right)$. Then the optimization problem can be written as

$$\max_{\mathbf{y} \in \Omega} u(\mathbf{y}) - v(\mathbf{y}).$$

The *DC programming solution* for such a problem uses an iterative procedure that linearizes the subtrahend in the concave difference about the current estimate and then solves the resulting concave objective for the next estimate [31] i.e.,

$$\mathbf{y}^{n+1} = \operatorname*{argmax}_{\mathbf{y} \in \Omega} u(\mathbf{y}) - \mathbf{y} \cdot \nabla v(\mathbf{y}^n).$$

Substituting u and ∇v gives

$$\mathbf{y}^{n+1} = \operatorname*{argmax}_{\mathbf{y} \in \Omega} \sum_{x \in L[q]} \left\{ C(x,q) \log y_x + y_x f(x, q, \mathbf{y}^n) \right\}, \tag{20}$$

where

$$f(x, q, \mathbf{y}^n) = \sum_{q_0 \in B_1(q)} \frac{C(\varphi, q_0) \mathbf{1}_{x \in L[q_0] \cap \Sigma}}{1 - \sum_{x' \in L[q_0] \cap \Sigma} y^n_{x'}}. \tag{21}$$

Observe that $1 - \sum_{x' \in L[q_0] \cap \Sigma} y^n_{x'} \geq \epsilon$ as the automaton is backoff-complete and $\mathbf{y}^n \in \Omega$. The following lemma provides the solution to the optimization problem in (20) which leads to a stationary point of the objective.

Lemma 5. *Solution to (20) is given by*

$$y^{n+1}_x = \max\left(\frac{C(x,q)}{\lambda - f(x, q, \mathbf{y}^n)}, \epsilon \right), \tag{22}$$

where $\lambda \in \left[\max_{x \in L[q]} f(x, q, \mathbf{y}^n) + C(x,q), \max_{x \in L[q]} f(x, q, \mathbf{y}^n) + C(q)\right]$ *such that* $\sum_x y^n_x = 1$.

Proof. With KKT multipliers, the optimization problem can be written as

$$\max_{\mathbf{y}, \lambda, \mu_x : \mu_x \leq 0} \sum_{x \in L[q]} \left\{ C(x,q) \log y_x + y_x f(x, q, \mathbf{y}^n) + \lambda\big(1 - \sum_{x \in L[q]} y_x\big) + \sum_{x \in L[q]} \mu_x(\epsilon - y_x). \right.$$

We divide the proof into two cases depending on the value of $C(x,q)$. Let $C(x,q) \neq 0$. Differentiating the above equation, we get

$$y^{n+1}_x = \frac{C(x,q)}{\lambda + \mu_x - f(x, q, \mathbf{y}^n)} \text{ and } \mu_x(\epsilon - y^{n+1}_x) = 0.$$

Hence, μ_x is only non-zero if $y_x^{n+1} = \epsilon$. Furthermore, Since for all x, $\mu_x \leq 0$, for y_x^{n+1} to be positive, we need $\lambda \geq \max f(x, q, \mathbf{y}^n)$. Hence, the above two conditions can be re-expressed as (22). If $C(x, q) = 0$, then we get

$$f(x, q, \mathbf{y}^n) = \lambda + \mu_x \text{ and } \mu_x(\epsilon - y_x^{n+1}) = 0,$$

and the solution is given by $y_x^{n+1} = \epsilon$ and $\mu_x = f(x, q, \mathbf{y}^n) - \lambda$. Since μ_x can be negative, we have $f(x, q, \mathbf{y}^n) \leq \lambda$. Hence, irrespective of the value of $C(x, q)$, the solution is given by (22).

The above analysis restricts $\lambda \geq \max_x f(x, q, \mathbf{y}^n)$. If $\lambda < f(x, q, \mathbf{y}^n) + C(x, q)$, then $y_x^n > 1$ and if $\lambda > \max_x f(x, q, \mathbf{y}^n) + C(q)$, then $\sum_x y_x^n < 1$. Hence λ needs to lie in

$$\left[\max_{x \in L[q]} f(x, q, \mathbf{y}^n) + C(x, q), \max_{x \in L[q]} f(x, q, \mathbf{y}^n) + C(q)\right]$$

to ensure that $\sum_x y_x^{n+1} = 1$. □

From this, we form algorithm KL-Minimization. Observe that if all the counts are zero, then for any $\mathbf{y}$, $u(\mathbf{y}) - v(\mathbf{y}) = 0$ and any solution is an optimal solution and the algorithm returns uniform distribution over labels. In other cases, we initialize the model based on counts such that $\mathbf{y}^0 \in \Omega$. We then repeat the DC programming algorithm iteratively until convergence. Since, Ω is a convex compact set and functions u, v, and ∇v are continuous and differentiable in Ω, the KL-Minimization converges to a stationary point (Theorem 4 [49]).

Algorithm KL-Minimization

Notation:

- $y_x = p_a(x|q)$ for $x \in L(q)$
- $C(x, q)$ from Equations 10 and 11
- $C(q) = \sum_{x' \in L[q]} C(x', q)$
- $f(x, q, \mathbf{y}^n)$ from Equation 21
- $\text{lb} = \max_{x \in L[q]} f(x, q, \mathbf{y}^n) + C(x, q)$
- $\text{ub} = \max_{x \in L[q]} f(x, q, \mathbf{y}^n) + C(q)$
- $k = |L[q]|$
- $\epsilon =$ lower bound on y_x

Trivial case: If $C(q) = 0$, output $\mathbf{y}$ given by $y_x = 1/k$ for all x.
Initialization: Initialize:

$$y_x^0 = \frac{C(x, q)}{C(q)} (1 - k\epsilon) + \epsilon.$$

Iteration: Until convergence do:

$$y_x^{n+1} = \max \left(\frac{C(x, q)}{\lambda - f(x, q, \mathbf{y}^n)}, \epsilon \right),$$

where $\lambda \in [\text{lb}, \text{ub}]$ is chosen (in a binary search) to ensure $\sum_{x \in L(q)} y_x = 1$.

5 Experiments

We now provide experimental evidence of the theory's validity and show its usefulness in various applications. For the ease of notation, we use WFA-Approx to denote the exact counting algorithm described in Section 4.1.2 followed by the KL-Minimization algorithm of Section 4.2. Similarly, we use WFA-SampleApprox(N) to denote the sampled counting described in Section 4.1.3 with N sampled sentences followed by KL-Minimization.

We first give experimental evidence that supports the theory in Section 5.1. We then show how to approximate neural models as WFAs in Section 5.2. We also use the proposed method to provide lower bounds on the perplexity given a target topology in Section 5.3. Motivated by low-memory applications such as (virtual) keyboard decoding [44], we then use our approach to create compact language models in Section 5.4. Finally, we use our approach to create compact open-vocabulary character language models from count-thresholded data in Section 5.5.

For all the experiments we use the 1996 CSR Hub4 Language Model data, LDC98T31 (broadcast news data). We use the processed form of the corpus and further process it to downcase all the words and remove punctuation. The resulting dataset has 132M words in the training set, 20M words in the test set, and has 240K unique words. For all the experiments that use word models, we create a vocabulary of approximately 32K words that consists of all words that appeared more than

Table 1: Test perplexity of n-gram models approximated onto the same topology.

Model	Test perplexity
Baseline	144.4
WFA-SampleApprox(500K)	154.7
WFA-SampleApprox(1M)	147.3
WFA-SampleApprox(2M)	145.0
WFA-SampleApprox(4M)	144.5
WFA-Approx	144.4

50 times in the training corpus. Using this vocabulary, we create a trigram Katz model and prune it to contain 2M n-grams using entropy pruning [50] that we use as a baseline in all our word-based experiments. We use Katz smoothing since it is amenable to pruning [16]. The perplexity of this model on the test set is 144.4. [7] All algorithms were implemented using the open-source `OpenFst` and `OpenGrm` n-gram and stochastic automata (`SFst`) libraries[8] with the last library including these implementations [5, 47, 6].

5.1 Empirical evidence of theory

Recall that our goal is to find the distribution on a target DFA topology that minimizes the KL divergence to the source distribution. However, as stated in Section 4.2, if the target topology has failure transitions, the optimization objective is not convex so the stationary point solution may not be the global optimum. We now show that the model indeed converges to a good solution in various cases empirically.

Idempotency: When the target topology is the same as the source topology, we show that the performance of the approximated model matches the source model. Let p_s be the pruned Katz word model described above. We approximate p_s onto the same topology using WFA-Approx and WFA-SampleApprox($\cdot$) and then compute perplexity on the test corpus. The results are presented in Table 1. The test perplexity of the WFA-Approx model matches that of the source model and the performance of the WFA-SampleApprox(N) model approaches that of the source model as the number of samples N increases.

Comparison to greedy pruning: Recall that entropy pruning [50] greedily removes n-grams such that the KL divergence to the original model p_s is small. Let p_{greedy} be the resulting model and A_{greedy} be the topology of p_{greedy}. If the KL-Minimization converges to a good solution, then approximating p_s onto A_{greedy} would give a model that is at least as good as p_{greedy}. We show that this is indeed the case; in fact, approximating p_s onto A_{greedy} performs better than p_{greedy}. In particular, let p_s again be the $2M$ n-gram Katz model described above. We prune it to have 1M n-grams and obtain p_{greedy}, which has a test perplexity of 157.4. We then approximate p_s on A_{greedy} and the resulting model has test perplexity of 155.6, which is smaller than the test perplexity of p_{greedy}. This shows that the approximation algorithm indeed finds a good solution.

[7] For all perplexity measurements we treat the unknown word as a single token instead of a class. To compute the perplexity with the unknown token being treated as class, multiply the perplexity by $k^{0.0115}$, where k is the number of tokens in the unknown class and 0.0115 is the out of vocabulary rate in the test dataset.

[8] These libraries are available at `www.openfst.org` and `www.opengrm.org`

Table 2: Test perplexity of LSTM models approximated onto a Katz-derived DFA topology.

Model	Test perplexity
WFA-SampleApprox(500K)	150.1
WFA-SampleApprox(1M)	144.3
WFA-SampleApprox(2M)	142.5
WFA-SampleApprox(4M)	141.6
WFA-SampleApprox(8M)	141.2
WFA-SampleApprox(16M)	140.9
WFA-SampleApprox(32M)	140.8

Table 3: Test perplexity of LSTM models approximated onto sampled DFA topology.

Model	Test perplexity
WFA-SampleApprox(500K)	174.0
WFA-SampleApprox(1M)	164.7
WFA-SampleApprox(2M)	156.2
WFA-SampleApprox(4M)	150.0
WFA-SampleApprox(8M)	146.7
WFA-SampleApprox(16M)	144.9
WFA-SampleApprox(32M)	143.9

5.2 Neural models to WFA conversion

Since neural models such as LSTMs give improved performance over n-gram models, we investigated if an LSTM distilled onto a WFA model can obtain better performance than the baseline WFA trained directly from Katz smoothing. As stated in the introduction, this could then be used together with federated learning for fast and private on-device inference.

To explore this, we trained an LSTM language model on the training data. The model has 2 LSTM layers with 1024 states and embedding size of 1024. The resulting model has a test perplexity of 60.5. We approximate this model as a WFA in two ways.

First, we approximate the neural model onto the Katz 2M n-gram topology described above using WFA-SampleApprox($\cdot$). The results are presented in Table 2, showing that the approximated LSTM models have better perplexity than the original Katz model with as little as 1M samples. With 32M samples, the approximated LSTM model is 3.6 better in perplexity than Katz.

The above experiments assume that the topology is known. If the WFA topology is unknown, we use the samples obtained in WFA-SampleApprox($\cdot$) to create a Katz model entropy-pruned to 2M n-grams. The results are shown in Table 3. Observe that the approximated models do not perform as well as the models obtained with the known topology derived from the training data. However with enough samples, their performance is similar to that of the original Katz model.

5.3 Lower bounds on perplexity

The neural model in Section 5.2 has a perplexity of 60.5, but the best perplexity for the approximated model is 140.8. Is there a better approximation algorithm for the given target topology? We place

bounds on that next.

Let T be the set of test sentences. The test-set log-perplexity of a model p can be written as

$$\frac{1}{|T|}\sum_{x^*\in T}\log\frac{1}{p(x^*)} = \sum_{x^*}\hat{p}_t(x^*)\log\frac{1}{p(x^*)},$$

where $\hat{p}_t$ is the empirical distribution of test sentences. Observe that the best model with topology A can be computed as

$$p'_a = \operatorname*{argmin}_{p_a\in\mathcal{P}(A)}\sum_{x^*}\hat{p}_t(x^*)\log\frac{1}{p_a(x^*)},$$

which is the model with topology A that has minimal KL divergence from the test distribution $\hat{p}_t$. This can be computed using WFA-Approx . If we use this approach on the Broadcast News test set with the 2M n-gram Katz model, the resulting model has perplexity of 121.1, showing that, under the assumption that algorithm finds the global KL divergence minimum, the test perplexity with this topology cannot be improved beyond 121.1, irrespective of the method.

What if we approximate the LSTM onto the best trigram topology, how well does it perform over the test data? To test this, we build a trigram model from the test data and approximate the LSTM on the trigram topology. This approximated model has 11M n-grams and a perplexity of 81. This shows that for large datasets, the shortfall of n-gram models in the approximation is in the n-gram topology.

5.4 Creating compact language models

Creating compact models for infrequent words. In low-memory applications such as on-device keyboard decoding [44], it is often useful to have a character-level WFA representation of a large set of vocabulary words that act only as unigrams, e.g. those words beyond the 32K words of our trigram model. We explore how to compactly represent such a unigram-only model.

To demonstrate our approach, we take all the words in the training set (without a count cut-off) and build a character-level deterministic WFA of those words weighted by their unigram probabilities. This is represented as a tree rooted at the initial state (a *trie*). This automaton has 820K transitions. Storing this many transitions can be prohibitive; we can reduce the size in two steps.

The first step is to minimize this WFA using weighted minimization [38] to produce p_{char}, which has a topology A_{char}. Although p_{char} is already much smaller (it has 378K transitions, a 54% reduction), we can go further by approximating onto the minimal deterministic *unweighted* automaton, Minimize(A_{char}). This gives us a model with only 283K transitions, a 25% reduction. Since Minimize(A_{char}) accepts exactly the same words as A_{char}, we are not corrupting our model by adding or removing any vocabulary items. Instead we find an estimate which is as close as possible to the original, but which is constrained to the minimal deterministic representation that preserves the vocabulary.

To evaluate this approach, we convert our test set into a corpus where each entry is the character-level sequence for each word in a 20K sentence subset of the original test set. We evaluate using cross entropy in bits-per-character, common for character-level models. The resulting cross entropy for p_{char} is 1.557 bits-per-character. By comparison, the cross entropy for p_{char} approximated onto Minimize(A_{char}) is 1.560 bits-per-character. In exchange for this small accuracy loss we are rewarded with a model which is 25% smaller.

Table 4: Test perplexity of models when approximated onto smaller topologies.

Topology	Test perplexity	# Transitions
A_{greedy}	155.6	1.13M
Minimize(A_{greedy})	156.4	1.05M
A'_{greedy}	154.1	1.22M
Minimize(A'_{greedy})	154.9	1.13M

Creating compact WFA language models. Motivated by the previous experiment, we also consider applying (unweighted) minimization to A_{greedy}, the word-based trigram topology that we pruned to 1M n-grams described earlier. In Table 4 we show that applying minimization to A_{greedy} and then approximating onto the resulting topology leads to a reduction of 7% in the number of transitions needed to represent the model. However, the test perplexity also increases some. To control for this, we prune the original model to a 1.08M n-gram topology A'_{greedy} instead of the 1M as before and apply the same procedure to obtain an approximation on Minimize(A'_{greedy}). We achieve a 0.4% perplexity reduction compared to the approximation on A_{greedy} with very nearly the same number of transitions.

5.5 Count thresholded data for privacy

One increasingly common scenario that can benefit from these algorithms is modeling from frequency thresholded substring counts rather than raw text. For example, word n-grams and their frequencies may be provided from certain domains of interest only when they occur within at least k separate documents. With a sufficiently large k (say 100), no n-gram can be traced to a specific document, thus providing privacy in the aggregation. This is known as k-anonymity [48].

However, for any given domain, there are many kinds of models that one may want to build depending on the task, some of which may be trickier to estimate from such a collection of word n-gram counts than with standard approaches for estimation from a given corpus. For example, character n-gram models can be of high utility for tasks like language identification, and have the benefit of a relatively small memory footprint and low latency in use.

Here we will compare open-vocabulary character language models, which accept all strings in Σ^* for a character vocabulary Σ, trained in several ways. Each approach relies on the training corpus and 32k vocabulary, with every out-of-vocabulary word replaced by a single OOV symbol ★. Additionally, for each approach we add 50 to the unigram character count of any printable ASCII character, so that even those that are unobserved in the words of our 32k vocabulary have some observations. Our three approaches are:

1. **Baseline corpus trained models:** We counted character 5-grams from the corpus, then removed all n-grams that included the ★ symbol (in any position) prior to smoothing and normalization. Here we present both Kneser-Ney and Witten Bell smoothed models, as both are popular for character n-gram models.

2. **Word trigram sampled model:** First we count word trigrams and discard any n-gram with the ★ symbol (in any position) prior to smoothing and normalization. We then sample one million strings from a Katz smoothed model and build a character 5-gram model from these strings. We also use this as our target topology for the next approach.

Table 5: Comparison of character 5-gram models derived from either the original corpus or a word trigram model. Size of the models is presented in terms of the number of character n-grams, the numbers of states and transitions in the automaton representation, and the file size in MB. The two corpus estimated models have the same topology, hence the same size; as do the two word trigram estimated models.

Source	n-grams (x1000)	states (x1000)	transitions (x1000)	MB	Estimation	bits/char
Corpus	336	60	381	6.5	Kneser-Ney	2.04
					Witten-Bell (WB)	2.01
Word trigram	277	56	322	5.6	Sampled (WB)	2.36
					KL min	1.99

3. **Word trigram KL minimization estimation:** We create a source model by converting the 2M n-gram word trigram model into an open vocabulary model. We do this using a specialized construction, described below, that converts the word model into a character sequence model. As this model is still closed vocabulary (see below), we additionally smooth the unigram distribution with a character trigram model trained from the words in the symbol table (and including the 50 extra counts for every printable ASCII character as with the other methods). From this source model, we estimate a model on the sampled character 5-gram topology from the previous approach, using our KL minimization algorithm.

Converting word n-gram to character sequence model. Briefly, for every state s in the n-gram automaton, the set of words labeling transitions leaving s are represented as a trie of characters including a final end-of-word symbol. Each resulting transition labeled with the end-of-word symbol represents the last transition for that particular word spelled out by that sequence of transitions, hence is assigned the same destination state as the original word transition. If s has a backoff transition pointing to its backoff state s', then each new internal state in the character trie backs off to the corresponding state in the character trie leaving s'.

As stated above, this construction converts from word sequences to character sequences, but will only accept character sequences consisting of strings of in-vocabulary words, i.e., this is still closed vocabulary. To make it open vocabulary, we further backoff the character trie leaving the unigram state to a character n-gram model estimated from the symbol table (and additional ASCII character observations). This is done using a very similar construction to that described above. The resulting model is used as the source model for your KL minimization algorithm, to estimate a distribution over the sampled character 5-gram topology.

We encode the test set as a sequence of characters, without using the symbol table since our models are intended to be open vocabulary. Following typical practice for open-vocabulary settings, we evaluate with bits-per-character. The results are presented in Table 5. Here we achieve a bits-per-character even slightly better than what we get straight from the corpus, perhaps due to better regularization of the word-based model than with either Witten-Bell or Kneser-Ney on the character n-grams.

6 Discussion

References

[1] H. Adel, K. Kirchhoff, N. T. Vu, D. Telaar, and T. Schultz. Comparing approaches to convert recurrent neural networks into backoff language models for efficient decoding. In *Fifteenth Annual Conference of the International Speech Communication Association*, 2014.

[2] A. V. Aho and M. J. Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6):333–340, 1975.

[3] J. Albert and J. Kari. Digital image compression. In *Handbook of weighted automata*. Springer, 2009.

[4] C. Allauzen, M. Mohri, and B. Roark. Generalized algorithms for constructing language models. In *Proceedings of ACL*, pages 40–47, 2003.

[5] C. Allauzen, M. Riley, J. Schalkwyk, W. Skut, and M. Mohri. OpenFst Library. *http://www.openfst.org*, 2007.

[6] C. Allauzen and M. D. Riley. Algorithms for weighted finite automata with failure transitions. In *International Conference on Implementation and Application of Automata*, pages 46–58. Springer, 2018.

[7] D. Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[8] D. Angluin. Identifying languages from stochastic examples. Technical Report YALEU /DCS /RR-614, Yale University, 1988.

[9] E. Arisoy, S. F. Chen, B. Ramabhadran, and A. Sethy. Converting neural network language models into back-off language models for efficient decoding in automatic speech recognition. *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)*, 22(1):184–192, 2014.

[10] B. Balle, X. Carreras, F. M. Luque, and A. Quattoni. Spectral learning of weighted automata. *Machine learning*, 96(1-2):33–63, 2014.

[11] B. Balle and M. Mohri. Spectral learning of general weighted automata via constrained matrix completion. In *Advances in neural information processing systems*, pages 2159–2167, 2012.

[12] T. M. Breuel. The OCRopus open source OCR system. In *Proceedings of IS&T/SPIE 20th Annual Symposium*, 2008.

[13] R. C. Carrasco. Accurate computation of the relative entropy between stochastic regular grammars. *RAIRO-Theoretical Informatics and Applications*, 31(5):437–444, 1997.

[14] R. C. Carrasco and J. Oncina. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152. Springer, 1994.

[15] R. C. Carrasco and J. Oncina. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO-Theoretical Informatics and Applications*, 33(1):1–19, 1999.

[16] C. Chelba, T. Brants, W. Neveitt, and P. Xu. Study on interaction between entropy pruning and kneser-ney smoothing. In *Eleventh Annual Conference of the International Speech Communication Association*, 2010.

[17] S. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. Technical report, TR-10-98, Harvard University, 1998.

[18] O. Cicchello and S. C. Kremer. Inducing grammars from sparse data sets: a survey of algorithms and results. *Journal of Machine Learning Research*, 4(Oct):603–632, 2003.

[19] C. Cortes, M. Mohri, A. Rastogi, and M. Riley. On the computation of the relative entropy of probabilistic automata. *International Journal of Foundations of Computer Science*, 19(01):219–242, 2008.

[20] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[21] A. Deoras, T. Mikolov, S. Kombrink, M. Karafiát, and S. Khudanpur. Variational approximation of long-span language models for lvcsr. In *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on*, pages 5532–5535. IEEE, 2011.

[22] P. Dupont. Incremental regular inference. In *International Colloquium on Grammatical Inference*, pages 222–237. Springer, 1996.

[23] R. Durbin, S. R. Eddy, A. Krogh, and G. J. Mitchison. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Camb. Univ. Press, 1998.

[24] P. Ebden and R. Sproat. The kestrel tts text normalization system. *Natural Language Engineering*, 21(3):333–353, 2015.

[25] J. Eisner. Expectation semirings: Flexible em for learning finite-state transducers. In *Proceedings of the ESSLLI workshop on finite-state methods in NLP*, pages 1–5, 2001.

[26] C. L. Giles, C. B. Miller, D. Chen, H.-H. Chen, G.-Z. Sun, and Y.-C. Lee. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405, 1992.

[27] E. M. Gold. Complexity of automaton identification from given data. *Information and control*, 37(3):302–320, 1978.

[28] E. M. Gold and T. R. Corporation. Language identification in the limit. *Information and control*, 10(5):447–474, 1967.

[29] A. Hard, K. Rao, R. Mathews, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[30] L. Hellsten, B. Roark, P. Goyal, C. Allauzen, F. Beaufays, T. Ouyang, M. Riley, and D. Rybach. Transliterated mobile keyboard input via weighted finite-state transducers. In *FSMNLP 2017*, pages 10–19, 2017.

[31] R. Horst and N. V. Thoai. Dc programming: overview. *Journal of Optimization Theory and Applications*, 103(1):1–43, 1999.

[32] G. Iglesias, C. Allauzen, W. Byrne, A. de Gispert, and M. Riley. Hierarchical phrase-based translation representations. In *EMNLP 2011*, pages 1373–1383, 2011.

[33] H. Jacobsson. Rule extraction from recurrent neural networks: Ataxonomy and review. *Neural Computation*, 17(6):1223–1263, 2005.

[34] S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recogniser. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 35(3):400–401, 1987.

[35] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*, 2016.

[36] G. Lecorvé and P. Motlicek. Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition. In *Thirteenth Annual Conference of the International Speech Communication Association*, 2012.

[37] M. Mohri. String-matching with automata. *Nord. J. Comput.*, 4(2):217–231, 1997.

[38] M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1-2):177–201, 2000.

[39] M. Mohri. Semiring frameworks and algorithms for shortest-distance problems. *Journal of Automata, Languages and Combinatorics*, 7(3):321–350, 2002.

[40] M. Mohri. Weighted automata algorithms. In *Handbook of Weighted Automata*, pages 213–254. Springer, 2009.

[41] M. Mohri, F. C. N. Pereira, and M. Riley. Speech recognition with weighted finite-state transducers. In *Handbook on speech proc. and speech comm.* Springer, 2008.

[42] J. R. Novak, N. Minematsu, and K. Hirose. Failure transitions for joint n-gram models and g2p conversion. In *INTERSPEECH*, pages 1821–1825, 2013.

[43] J. Oncina and P. Garcia. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, pages 99–108. World Scientific, 1992.

[44] T. Ouyang, D. Rybach, F. Beaufays, and M. Riley. Mobile keyboard input decoding with finite-state transducers. *arXiv preprint arXiv:1704.03987*, 2017.

[45] R. Parekh and V. Honavar. Grammar inference, automata induction, and language acquisition. *Handbook of natural language processing*, pages 727–764, 2000.

[46] L. Pitt. Inductive inference, dfas, and computational complexity. In *International Workshop on Analogical and Inductive Inference*, pages 18–44. Springer, 1989.

[47] B. Roark, R. Sproat, C. Allauzen, M. Riley, J. Sorensen, and T. Tai. The opengrm open-source finite-state grammar software libraries. *Proceedings of the ACL 2012 System Demonstrations*, pages 61–66, 2012.

[48] P. Samarati. Protecting respondents identities in microdata release. *IEEE transactions on Knowledge and Data Engineering*, 13(6):1010–1027, 2001.

[49] B. K. Sriperumbudur and G. R. Lanckriet. On the convergence of the concave-convex procedure. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, pages 1759–1767. Curran Associates Inc., 2009.

[50] A. Stolcke. Entropy-based pruning of backoff language models. *arXiv preprint cs/0006025*, 2000.

[51] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*, 2012.

[52] P. Tiňo and V. Vojtek. Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences. In *Knowledge-Based Intelligent Electronic Systems, 1997. KES'97. Proceedings., 1997 First International Conference on*, volume 2, pages 551–558. IEEE, 1997.

[53] G. Weiss, Y. Goldberg, and E. Yahav. Extracting automata from recurrent neural networks using queries and counterexamples. *arXiv preprint arXiv:1711.09576*, 2017.