Approximating Probabilistic Models as Weighted Finite Automata

Ananda Theertha Suresh
Google Research
theertha@google.com

Brian Roark
Google Research
roark@google.com

Michael Riley
Google Research
riley@google.com

Vlad Schogol
Google Research
vlads@google.com

Weighted finite automata (WFAs) are often used to represent probabilistic models, such as n*-gram language models, because among other things, 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 WFA such that the Kullback-Leibler 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 step, 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. The algorithms used for these experiments are available in an open-source software library.*

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

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

Some of the results in this article were previously presented in Suresh et al. (2019).

Submission received: 13 January 2020; revised version received: 11 September 2020; accepted for publication: 6 January 2021.

https://doi.org/10.1162/COLI_a_00401

Such a model might be Markovian, where

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

such as a k-gram language model (LM) (Chen and Goodman 1998) or it might be non-Markovian such as a long short-term memory (LSTM) neural network LM (Sundermeyer, Schlüter, and Ney 2012). 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-Leibler (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, interpretability, and composability. WFA models have been used in many applications including speech recognition (Mohri, Pereira, and Riley 2008), speech synthesis (Ebden and Sproat 2015), optical character recognition (Breuel 2008), machine translation (Iglesias et al. 2011), computational biology (Durbin et al. 1998), and image processing (Albert and Kari 2009). One particular problem of interest is language modeling for on-device (virtual) keyboard decoding (Ouyang et al. 2017), where WFA models are widely used because of space and time constraints. One example use of methods we present in this article comes from this domain, involving approximation of a neural LM. Storing training data from the actual domain of use (individual keyboard activity) in a centralized server and training k-gram or WFA models directly may not be feasible because of privacy constraints (Hard et al. 2018). To circumvent this, an LSTM model can be trained by federated learning (Konečnỳ et al. 2016; McMahan et al. 2017; Hard et al. 2018), converted to a WFA at the server, and then used for fast on-device inference. This not only may improve performance over training the models just on out-of-domain publicly available data, but also benefits from the additional privacy provided by federated learning. Chen et al. (2019) use the methods presented in this article for this very purpose.

There are multiple reasons why one may choose to approximate a source model with a WFA. One may have strong constraints on system latency, such as the virtual keyboard example above. Alternatively, a specialized application may require a distribution over just a subset of possible strings, but must estimate this distribution from a more general model—see the example below regarding utterances for setting an alarm. To address the broadest range of use cases, we aim to provide methods that permit large classes of source models and target WFA topologies. We explore several distinct scenarios experimentally in Section 5.

Our methods allow **failure transitions** (Aho and Corasick 1975; Mohri 1997) 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 (Katz 1987; Chen and Goodman 1998; Allauzen, Mohri, and Roark 2003; Novak, Minematsu, and Hirose 2013; Hellsten et al. 2017). The inclusion of failure transitions complicates 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 that 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 approximation $\hat{A}$. The main goal of this article

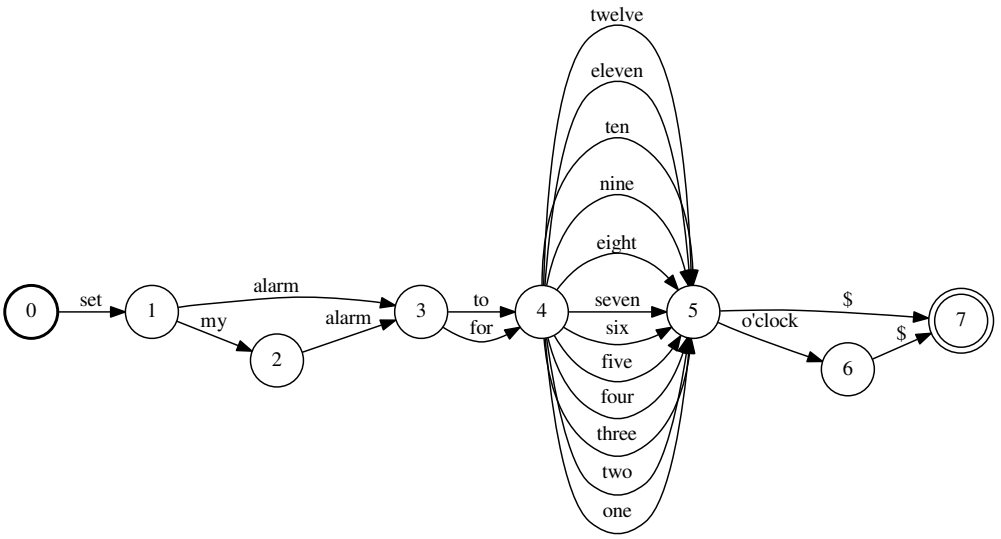

Figure 1
An unweighted automaton that specifies what one might say to set an alarm. The initial state is the bold circle and the final state is the double circle. By convention, we terminate all accepted strings with the symbol $.

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 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 one 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. For example, people may be more likely to set their alarms for six or seven than four. In the absence of data specifically for this scenario, we can fall back on some available background LM M, trained on a large suitable corpus. In particular, we can use the conditional distribution

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

where 1 is the indicator function and $\mathcal{L}(A)$ 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 (Mohri, Pereira, and Riley 2008; Mohri 2009). Our approximation has the same states as A whereas weight-pushed $M \cap A$ has $O(|M||A|)$ states. Furthermore, weight-pushed $M \cap A$ is an exact WFA representation of the distribution in Equation (1).

As stated earlier, some applications may simply require smaller models or those with lower latency of inference, and in such scenarios the specific target topology

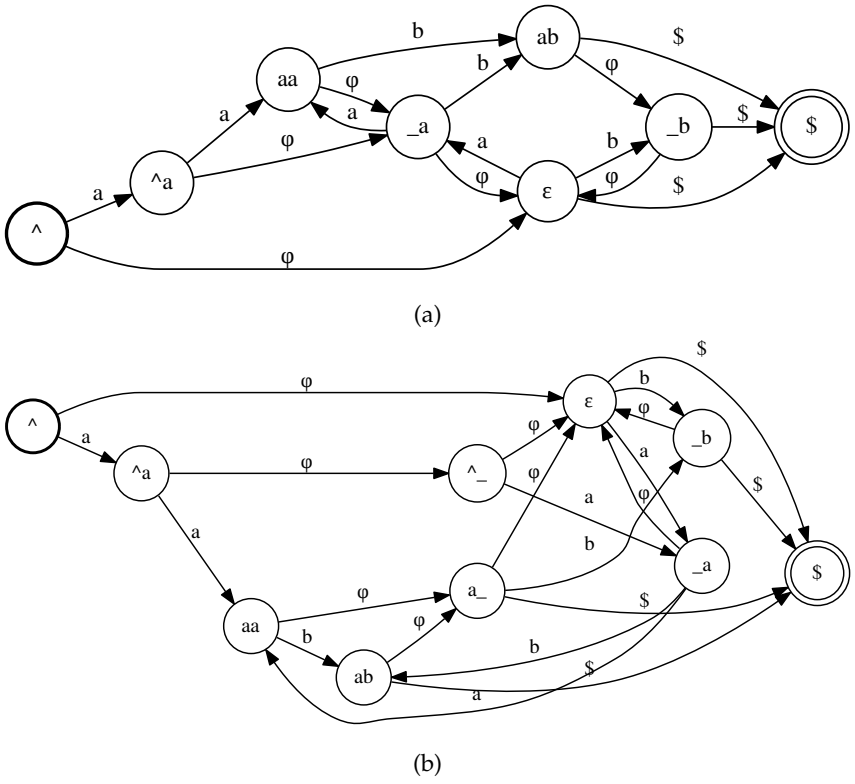

Figure 2
Topology examples derived from the corpus *aab*. States are labeled with the context that is remembered, ∧ denotes the initial context, ϵ the empty context, $ the final context (and terminates accepted strings), and _ matches any symbol in a context. (a) 3-gram topology: failure transitions, 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_*.

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* (Allauzen, Mohri, and Roark 2003). This could be from an existing corpus or from random samples drawn from *S*. Figure 2(a) shows a trigram topology for the very simple corpus *aab*. Figure 2(b) shows an alternative topology that allows *skip*-grams. Both of these representations make use of failure transitions. These allow modeling strings unseen in the corpus (e.g., *abab*) 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. We hence focus our evaluation on intrinsic measures of model quality such as perplexity or bits-per-character.[1]

1 Model size or efficiency of inference may be a common motivation for the model approximations we present, which may suggest a demonstration of the speed/accuracy tradeoff for some downstream use of the models. However, as stated earlier in this Introduction, there are mulitple reasons why an approximation may be needed, and our goal in this article is to establish that our methods provide a well-motivated approach should an approximation be required for any reason.

This article expands upon an earlier, shorter version (Suresh et al. 2019) by also providing, beyond the additional motivating examples that have already been presented in this Introduction: an extended related work section and references; expansion of the theoretical analysis from three lemmas without proofs (omitted for space) to five lemmas (and a corollary) with full proofs; inclusion of additional algorithms for counting (for general WFA source and target in addition to φ-WFA); inclusion of additional experiments, including some illustrating the exact KL-minimization methods available for WFA sources of different classes; and documentation of the open-source library that provides the full functionality presented here.

The article 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. 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. In Section 6 we briefly describe the open-source software used in our experiments. Finally, in Section 7 we discuss the results and offer conclusions.

2. Related Work

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 (Parekh and Honavar 2000; Cicchello and Kremer 2003). Gold (1967) showed that an arbitrary regular set L cannot 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 (Gold 1978). 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 (Oncina and Garcia 1992; Dupont 1996). 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 (Giles et al. 1992; Jacobsson 2005).

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 L or, if not, provides a counterexample. In this case the minimal m-state DFA corresponding to L can be learned in time polynomial in m (Angluin 1987). Weiss, Goldberg, and Yahav (2018) apply this method for 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 that the probabilities are generated from a weighted automaton A with $L = \mathcal{L}(A)$, then L is identifiable in the limit with "high probability" (Angluin 1988; Pitt 1989).

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 (Carrasco and Oncina 1994, 1999). Approaches that produce (possibly highly) non-deterministic results include the Expectation-Maximization (EM) algorithm (Dempster, Laird, and Rubin 1977)

applied to fully connected Hidden Markov models or spectral methods applied to automata (Balle and Mohri 2012; Balle et al. 2014). Eisner (2001) describes an algorithm for estimating probabilities in a finite-state **transducer** from data using EM-based methods. Weiss, Goldberg, and Yahav (2019) and Okudono et al. (2020) provide adaptations to the Weiss, Goldberg, and Yahav (2018) DFA extraction algorithm to yield weighted automata.

For approximating neural network (NN) models as WFAs, Deoras et al. (2011) used an RNN LM to generate samples that they then used to train a k-gram LM. Arisoy et al. (2014) 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. (2014) 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 (1997) quantized the continuous state space of an RNN and then estimated the transition probabilities from the RNN. Lecorvé and Motlicek (2012) quantized the hidden states in an LSTM to form a finite-state model and then used an entropy criterion to back off to low-order k-grams to limit the number of transitions. See Section 4.3 for a more detailed comparison with the most closely related methods, once the details of our algorithms have been provided.

Our article is distinguished in several respects from previous work. First, our general approach does not depend on the form of 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 approach. 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, which is 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 $x_i x_{i+1} \dots x_n$ 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}, such that[2]

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

Without loss of generality, we assume that the model maintains an internal state q and updates it after observing the next symbol.[3] Furthermore, the probability of the subsequent state just depends on the state q

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

2 We define $x^0 \triangleq \epsilon$, the empty string, and adopt $p(\epsilon) = 0$.
3 In the most general case, $q(x^n) = x^n$.

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

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

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

The KL divergence between the source model p_s and the target model p_a is given by

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

where for notational simplicity, we adopt the notion $0/0 = 1$ and $0 \log(0/0) = 0$ throughout the article. Note that for the KL divergence to be finite, we need $\mathcal{L}(p_s) \subseteq \mathcal{L}(p_a)$. We will assume throughout that the source entropy $H(p_s) = -\sum_{x^n} p_s(x^n) \log p_s(x^n)$ is finite.[4] We first reduce the KL divergence between two models as follows (cf. Carrasco 1997; Cortes et al. 2008). In the following, let q_* denote the states of the probability distribution p_*.

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

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

where

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

Proof.

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

4 If $|Q(p_s)|$ is finite, it can be shown that $H(p_s)$ is necessarily finite.

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

$$= \sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1}) \sum_{x_i} p_s(x_i|x^{i-1}) \log \frac{p_s(x_i|x^{i-1})}{p_a(x_i|x^{i-1})}$$

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

$$\sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1}) \sum_{x_i} p_s(x_i|x^{i-1}) \log \frac{p_s(x_i|x^{i-1})}{p_a(x_i|x^{i-1})}$$

$$= \sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1}) \sum_{x_i} p_s(x_i|q_s(x^{i-1})) \log \frac{p_s(x_i|q_s(x^{i-1}))}{p_a(x_i|q_a(x^{i-1}))}$$

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

$$= \sum_{q_a \in Q_a} \sum_{q_s \in Q_s} \sum_{x_i} \gamma(q_s, q_a)\, p_s(x_i|q_s) \log \frac{p_s(x_i|q_s)}{p_a(x_i|q_a)}$$

Replacing x_i by x yields the lemma. □

The quantity $\gamma(q_s, q_a)$ counts each string x^i that reaches both state q_s in Q_s and state q_a in Q_a weighted by its probability according to p_s. Equivalently, it counts each string x^i reaching state (q_s, q_a) in $Q_s \times Q_a$ (an interpretation we develop in Section 4.1.1).[5] Note that γ does not depend on distribution p_a.

The following corollary is useful for finding the probabilistic model p_a that has the minimal KL divergence from a model p_s.

Corollary 1
Let $\mathscr{P}$ be a set of probabilistic models p_a for which $\mathcal{L}(p_s) \subseteq \mathcal{L}(p_a)$. Then

$$\operatorname*{argmin}_{p_a \in \mathscr{P}} D(p_s||p_a) = \operatorname*{argmax}_{p_a \in \mathscr{P}} \sum_{q_a \in Q_a} \sum_{x \in \Sigma} c(x, q_a) \log p_a(x|q_a)$$

where

$$c(x, q_a) = \sum_{q_s \in Q_s} \gamma(q_s, q_a)\, p_s(x|q_s) \tag{6}$$

5 γ is not (necessarily) a probability distribution over $Q_s \times Q_a$. In fact, $\gamma(q_s, q_a)$ can be greater than 1.

Proof. By Lemma 1

$$\begin{aligned}
\operatorname*{argmin}_{p_a \in \mathscr{P}} D(p_s||p_a) &= \operatorname*{argmin}_{p_a \in \mathscr{P}} \sum_{q_a \in Q_a} \sum_{q_s \in Q_s} \sum_{x \in \Sigma} \gamma(q_s, q_a)\, p_s(x|q_s) \log \frac{p_s(x|q_s)}{p_a(x|q_a)} \\
&= \operatorname*{argmin}_{p_a \in \mathscr{P}} \left\{ \sum_{q_a \in Q_a} \sum_{q_s \in Q_s} \sum_{x \in \Sigma} \gamma(q_s, q_a)\, p_s(x|q_s) \log p_s(x|q_s) \right. \\
&\qquad \left. - \sum_{q_a \in Q_a} \sum_{q_s \in Q_s} \sum_{x \in \Sigma} \gamma(q_s, q_a)\, p_s(x|q_s) \log p_a(x|q_a) \right\} \qquad (7) \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}} \sum_{q_a \in Q_a} \sum_{x \in \Sigma} c(x, q_a) \log p_a(x|q_a)
\end{aligned}$$

since the first term in Equation (7) does not depend on p_a. □

The quantity $c(x_i, q_a)$ counts each string x^i that reaches a state (q_s, q_a) in $Q_s \times \{q_a\}$ by x_1^{i-1} weighted by its probability according to p_s (cf. Equation (18)).

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 a **previous** (or **source**) state $p[e]$ to the **next** (or **destination**) state $n[e]$ with the **label** $\ell[e]$ and **weight** $w[e]$. The transitions with previous 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 previous state of a path we denote by $p[\pi]$ and the next 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

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

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

$$\operatorname*{argmin}_{p_a \in \mathscr{P}(A)} D(p_s||p_a) = \tilde{p}(x|q_a) \triangleq \frac{c(x, q_a)}{c(q_a)} \tag{8}$$

where

$$c(q_a) = \sum_{x \in \Sigma} c(x, q_a)$$

Proof. From Corollary 1

$$\begin{aligned}
\operatorname*{argmin}_{p_a \in \mathscr{P}(A)} D(p_s||p_a) &= \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \sum_{q_a \in Q_a} \sum_{x \in \Sigma} c(x, q_a) \log p_a(x|q_a) \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \sum_{q_a \in Q_a} \sum_{x \in \Sigma} c(q_a)\, \tilde{p}(x|q_a) \log p_a(x|q_a) \\
&= \operatorname*{argmin}_{p_a \in \mathscr{P}(A)} \sum_{q_a \in Q_a} c(q_a) \Big\{ - \sum_{x \in \Sigma} \tilde{p}(x|q_a) \log p_a(x|q_a) \Big\}
\end{aligned}$$

The quantity in braces is minimized when $p_a(x|q_a) = \tilde{p}(x|q_a)$ because it is the cross entropy between the two distributions. 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.[6] However, it is followed only when the input cannot 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 all k, $i \leq k < j$, 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.

6 In other words, a φ label, like an ϵ label, acts as an identity element in string concatenation (Allauzen and Riley 2018).

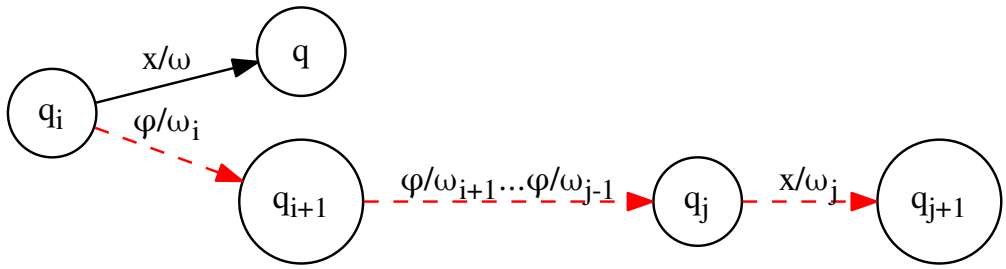

Figure 3
The (dashed red) path $e_i = (q_i, \varphi, \omega_i, q_{i+1})$ to $e_j = (q_j, x, \omega_j, q_{j+1})$ is disallowed since x can be read already on $e = (q_i, x, \omega, q)$.

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 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

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

This is a set of (possibly new) transitions (q, x, ω, q'), one for each allowed path from previous state q to next state q' with optional leading failure transitions and a final x-labeled transition. Denote the labels of $E^*[q]$ by $L^*[q]$. Note the WFA (Σ, Q, E^*, i, f) accepts the same strings with the same weights as φ-WFA A and thus $\mathcal{L}(A)$ is regular (Allauzen and Riley 2018).

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 restrict our target automata to have a topology with the backoff-complete property since it will simplify our analysis, make our algorithms efficient, and is commonly found in applications. For example, backoff n-gram models, such as the Katz model, are φ-cycle-free, backoff-complete φ-WFAs (Katz 1987; Chen and Goodman 1998).

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 $p_a \in \mathscr{P}(A)$ to p_a^* as follows:[7] Given a symbol $x \in \Sigma \cup \{\varphi\}$ and state $q \in Q$, define $p_a(x|q) \triangleq p_a^*(x|q)$ when $x \in L[q] \cap \Sigma$, $p_a(\varphi|q) \triangleq 1 - \sum_{x \in L[q] \cap \Sigma} p_a^*(x|q)$, 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 $p_a \in \mathscr{P}(A)$ 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

$$d(q, q') = 1 - \sum_{x \in L[q] \cap \Sigma} p_a(x|q') \tag{9}$$

Note that we have specified the weights on the automaton that represents $p_a^* \in \mathscr{P}^*(A)$ entirely in terms of the companion distribution $p_a \in \mathscr{P}(A)$. The failure transition weight α is determined from the requirement that $\sum_x E^*[q] = 1$ (Katz 1987, Equation (16)). Thanks to the backoff-complete property, this transition weight depends only on its previous state q and its next state q' and not all the states in the failure path from q. This is a key reason for assuming that property; otherwise our algorithms could not benefit from this locality.

Conversely, each distribution $p_a \in \mathscr{P}(A)$ can be associated with 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

$$\alpha(q, q') = \prod_{e \in \pi_\varphi(q, q')} \frac{p_a(\varphi|p[e])}{d(p[e], n[e])} \tag{10}$$

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

$$p_a^*(x|q) = \begin{cases} \alpha(q, q^x)\, p_a(x|q^x), & x \in L^*[q] \\ 0, & \text{otherwise} \end{cases} \tag{11}$$

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 Equation (10) 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 $x \in L[q]$.[8]

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.

7 The meaning of $\mathscr{P}(A)$ when A is φ-WFA is to interpret it as a WFA with the failure labels as regular symbols.

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

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 cannot be assured. While membership in $\mathscr{P}(A)$ principally requires the weights of the transitions leaving a state are non-negative and sum to 1, membership in $\mathscr{P}^*(A)$ imposes additional constraints due to the failure transitions, indicated in Equation (11).

As such, we restate our goal in terms of the companion distribution $p_a \in \mathscr{P}(A)$ rather than its corresponding distribution $p_a^* \in \mathscr{P}^*(A)$ directly. Let $B_n(q)$ be the set of states in A that back off to state q in n failure transitions and let $B(q) = \bigcup_{n=0}^{|Q_a|} B_n(q)$.

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

$$C(x,q) = \sum_{q_a \in B(q)} c(x, q_a) \mathbf{1}_{q = q_a^x}, \quad x \in \Sigma \tag{12}$$

$$C(\varphi,q) = \sum_{q_a \in B(q)} \sum_{x \in \Sigma} c(x, q_a) \mathbf{1}_{x \notin L[q]} \tag{13}$$

and do not depend on p_a.

Proof. From Corollary 1, Equation (11) and the previously shown 1:1 correspondence between each distribution $p_a^* \in \mathscr{P}^*(A)$ and its companion distribution $p_a \in \mathscr{P}(A)$

$$\begin{aligned}
\operatorname*{argmin}_{p_a^* \in \mathscr{P}^*(A)} D(p_s||p_a^*) &= \operatorname*{argmin}_{p_a^* \in \mathscr{P}^*(A)} \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log p_a^*(x|q_a) \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \alpha(q_a, q_a^x) p_a(x|q_a^x) \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \prod_{e \in \pi_\varphi(q_a, q_a^x)} \frac{p_a(\varphi|p[e])}{d(p[e], n[e])} p_a(x|q_a^x) \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \left\{ A_x + A_\varphi - A_d \right\}
\end{aligned} \tag{14}$$

where we distribute the factors inside the logarithm in Equation (14) as follows:

$$\begin{aligned} A_x &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log p_a(x|q_a^x) \\ &= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in L[q] \cap \Sigma} c(x, q_a) \mathbf{1}_{q=q_a^x} \log p_a(x|q) \\ &= \sum_{q \in Q_a} \sum_{x \in L[q] \cap \Sigma} C(x, q) \log p_a(x|q) \end{aligned} \tag{15}$$

Equation (15) follows from $q = q_a^x$ implying $q_a \in B(q)$.

$$\begin{aligned} A_\varphi &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \prod_{e \in \pi_\varphi(q_a, q_a^x)} p_a(\varphi|p[e]) \\ &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \sum_{e \in \pi_\varphi(q_a, q_a^x)} \log p_a(\varphi|p[e]) \\ &= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in L^*[q_a]} c(x, q_a) \sum_{e \in \pi_\varphi(q_a, q_a^x)} \mathbf{1}_{q=p[e]} \log p_a(\varphi|q) \\ &= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in \Sigma} c(x, q_a) \mathbf{1}_{x \notin L[q]} \log p_a(\varphi|q) \\ &= \sum_{q \in Q_a} C(\varphi, q) \log p_a(\varphi|q) \end{aligned} \tag{16}$$

Equation (16) 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 (14) proves the lemma. □

If there are no failure transitions in the target automaton, then $C(x, q) = c(x, q)$, if x is in Σ, and is 0 otherwise. In this case, the statement of Lemma 4 simplifies to that of Corollary 1.

Unfortunately, we do not have a closed-form solution, analogous to Lemma 2 or Lemma 3, for the general φ-WFA case. Instead we will present numerical optimization algorithms to find the KL divergence minimum in the next section. This is aided by the observation that the quantity in braces in the statement of Lemma 4 depends on the distribution p_a only at state q. Thus the KL divergence minimum can be found by maximizing that quantity independently for each state.

4. Algorithms

Approximating a probabilistic source algorithmically as a WFA requires two steps: (1) compute the quantity $c(x, q_a)$ found in Corollary 1 and 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, followed by an explicit comparison of the presented algorithms with closely related prior work.

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 (6) this can be written as

$$c(x, q_a) = \sum_{q_s \in Q_s} \gamma(q_s, q_a) p_s(x|q_s) \tag{17}$$

where

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

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 (Mohri 2009). 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* (Mohri 2002; Allauzen and Riley 2018). Algorithms to efficiently compute the intersection and shortest distance on WFAs are available in `OpenFst` (Allauzen et al. 2007), an open-source WFA library.

Then from Equation (17) we can form the sum

$$c(x, q_a) = \sum_{((q_s, q_a), x, w, (q_s', q_a')) \in E_{S \cap A}} \gamma(q_s, q_a)\, w \tag{18}$$

Equation (18) 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.

The worst-case time complexity for the counting step is dominated by the shortest distance algorithm on the intersection $S \cap A$. The shortest distance computation is a meta-algorithm that depends on the queue discipline selected (Mohri 2002). If s is the maximum number of times a state in the intersection is inserted into the shortest distance queue, C the maximum cost of a queue operation, and D the maximum out-degree in S and A (both assumed deterministic), then the algorithm runs in $O(s(D + C)|Q_S||Q_A|)$ (Mohri 2002; Allauzen and Riley 2018). The worst-case space complexity is in $O(D|Q_S||Q_A|)$, determined by the intersection size.

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 (12) and the previous case this can be written as

$$C(x, q) = \sum_{q_a \in B(q)} \sum_{q_s \in Q_s} \gamma(q_s, q_a) p_s(x|q_s) \mathbf{1}_{q=q_a^x}, \quad x \in \Sigma \tag{19}$$

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 (2018). Equation (19) is the 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 (Allauzen and Riley 2018). The result is then promoted to a transducer T with the output label used to keep track of the previous state in A of the compensated

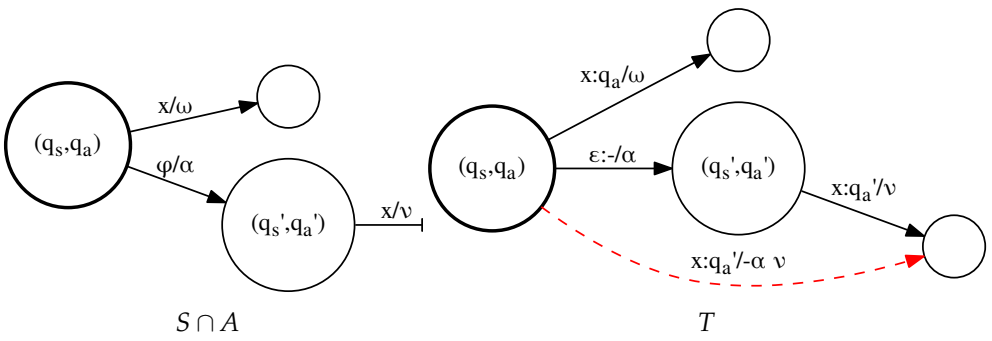

Figure 4
A φ-WFA is transformed into an equivalent WFA by replacing each failure transition by an ϵ-transition. To compensate for the formerly disallowed paths, new (dashed red) negatively weighted transitions are added. The result is promoted to a transducer T with the output label used to keep track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

positive transition (see Figure 4).[9] Algorithms to efficiently compute the intersection and shortest distance on φ-WFAs are available in the `OpenGrm` libraries (Allauzen and Riley 2018).

Then

$$C(x,q) = \sum_{((q_s,q_a),x,q,w,(q_s',q_a'))\in E_T} \gamma_T(q_s,q_a)w, \quad x \in \Sigma \tag{20}$$

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 (2018). Equation (20) 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 q must equal the count mass leaving a state

$$\begin{aligned}\sum_{(q_a,x,1,q)\in E_A} C(x,q_a) &= \sum_{(q,x',1,q_a')\in E_A} C(x',q) \\ &= \sum_{(q,x',1,q_a')\in E_A, x'\in\Sigma} C(x',q) + C(\varphi,q)\end{aligned}$$

Thus

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

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

The worst-case time and space complexity for the counting step for φ-WFAs is the same as for WFAs (Allauzen and Riley 2018).

4.1.3 Arbitrary Source and φ-WFA Target. In some cases, the source is a distribution with possibly infinite states, for example, LSTMs. For these sources, computing $C(x,q)$ can be computationally intractable as Equation (19) 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 $x(1), x(2), \ldots, x(N)$ be independent random samples from p_s. Instead of $C(x,q)$, we propose to use

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

9 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 (Allauzen and Riley 2018).

where

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

Observe that in expectation,

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

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 $C(\varphi, q)$ similarly to the previous section. If ℓ is the expected number of symbols per sample, then the computational complexity of counting in expectation is in $O(N\ell|\Sigma|)$.

4.2 KL Divergence Minimization

4.2.1 WFA Target. When the target topology is a deterministic 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 deterministic, backoff-complete 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 $\mathbf{y} \triangleq [y_x]_{x \in L[q]}$.[10] Then our goal reduces to

$$\operatorname*{argmax}_{\mathbf{y}} \sum_{x \in L[q]} C(x, q) \log y_x - \sum_{q_0 \in B_1(q)} C(\varphi, q_0) \log \left(1 - \sum_{x \in L[q_0] \cap \Sigma} y_x\right) \tag{21}$$

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

This is a difference of two concave functions in $\mathbf{y}$ since $\log(f(\mathbf{y}))$ is concave for any linear function $f(\mathbf{y})$, $C(x, q), C(\varphi, q_0)$ are always non-negative, and the sum of concave functions is also concave. We give a **DC programming** solution to this optimization (Horst and Thoai 1999). Let

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

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

and let $u(\mathbf{y}) = \sum_{x \in L[q]} C(x,q) \log y_x$ and $v(\mathbf{y}) = \sum_{q_0 \in B_1(q)} C(\varphi, q_0) \log(1 - \sum_{x \in L[q_0] \cap \Sigma} y_x)$. 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 (Horst and Thoai 1999), that is,

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

Substituting u and ∇v gives

$$\mathbf{y}^{n+1} = \operatorname*{argmax}_{\mathbf{y} \in \Omega} \sum_{x \in L[q]} \left\{ C(x,q) \log y_x + y_x f(x,q,\mathbf{y}^n) \right\} \tag{22}$$

where

$$f(x,q,\mathbf{y}^n) = \sum_{q_0 \in B_1(q)} \frac{C(\varphi, q_0) \mathbf{1}_{x \in L[q_0] \cap \Sigma}}{1 - \sum_{x' \in L[q_0] \cap \Sigma} y^n_{x'}} \tag{23}$$

Observe that $1 - \sum_{x' \in L[q_0] \cap \Sigma} y^n_{x'} \geq \epsilon$ as the automaton is backoff-complete and $\mathbf{y}^n \in \Omega$.

Let $C(q)$ be defined as:

$$C(q) = \sum_{x' \in L[q]} C(x', q)$$

The following lemma provides the solution to the optimization problem in Equation (22) that leads to a stationary point of the objective.

Lemma 5
Solution to Equation (22) is given by

$$y^{n+1}_x = \max\left(\frac{C(x,q)}{\lambda - f(x,q,\mathbf{y}^n)}, \epsilon \right) \tag{24}$$

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

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

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

We divide the proof into two cases depending on the value of $C(x,q)$. Let $C(x,q) \neq 0$. Differentiating this equation with respect to y_x and equating to zero, we get

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

Furthermore, by the KKT condition, $\mu_x(\epsilon - y_x^{n+1}) = 0$. Hence, μ_x is only non-zero if $y_x^{n+1} = \epsilon$ and if μ_x is zero, then $y_x^{n+1} = \frac{C(x,q)}{\lambda - f(x,q,\mathbf{y}^n)}$. Furthermore, because for all x, $\mu_x \leq 0$, for y_x^{n+1} to be positive, we need $\lambda \geq \max_x f(x,q,\mathbf{y}^n)$. Hence, the above two conditions can be re-expressed as Equation (24). 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 cannot be positive, we have $f(x,q,\mathbf{y}^n) \leq \lambda$ for all x. Hence, irrespective of the value of $C(x,q)$, the solution is given by Equation (24).

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 $y_x^{n+1} > 1$ and if $\lambda > \max_x f(x,q,\mathbf{y}^n) + \frac{C(q)}{1-|L[q]|\epsilon}$, then $\sum_x y_x^{n+1} < 1$. Hence λ needs to lie in

$$\left[\max_{x \in L[q]} f(x,q,\mathbf{y}^n) + C(x,q), \max_{x \in L[q]} f(x,q,\mathbf{y}^n) + \frac{C(q)}{1 - |L[q]|\epsilon}\right]$$

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

Algorithm KL-MINIMIZATION

Notation:

- $y_x = p_a(x|q)$ for $x \in L(q)$
- $C(x,q)$ from Equations (12) and (13)
- $C(q) = \sum_{x' \in L[q]} C(x',q)$
- $f(x,q,\mathbf{y}^n)$ from Equation (23)
- $k = |L[q]|$
- $\text{lb} = \max_{x \in L[q]} f(x,q,\mathbf{y}^n) + C(x,q)$
- $\text{ub} = \max_{x \in L[q]} f(x,q,\mathbf{y}^n) + \frac{C(q)}{1-k\epsilon}$
- $\epsilon =$ 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 $\sum_{x \in L(q)} y_x = 1$.

Figure 5
KL-MINIMIZATION algorithm.

From this, we form the KL-MINIMIZATION algorithm in Figure 5. 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 a 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. Because 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 (Sriperumbudur and Lanckriet 2009, Theorem 4). For each state q, the computational complexity of KL-MINIMIZATION is in $O(|E[q]|)$ per iteration. Hence, if the maximum number of iterations per each state is s, the overall computational complexity of KL-MINIMIZATION is in $O(s|E|)$.

4.3 Discussion

Our method for approximating from source φ-WFAs, specifically from backoff k-gram models, shares some similarities with entropy pruning (Stolcke 2000). Both can be used to approximate onto a more compact k-gram topology and both use a KL-divergence criterion to do so. They differ in that entropy pruning, by sequentially removing transitions, selects the target topology and in doing so uses a greedy rather than global entropy reduction strategy. We will empirically compare these methods in Section 5.1 for this use case.

Perhaps the closest to our work on approximating arbitrary sources is that of Deoras et al. (2011), where they used an RNN LM to generate samples that they then used to train a k-gram LM. In contrast, we use samples to approximate the joint state probability $\gamma(q_s, q_a)$. If ℓ is the average number of words per sentence and N is the total number of sampled sentences, then the time complexity of Deoras et al. (2011) is $\mathcal{O}(N\ell|\Sigma|)$ as it takes $\mathcal{O}(|\Sigma|)$ time to generate a sample from the neural model for every word. This is the same as that of our algorithm in Section 4.1.3. However, our approach has several advantages. First, for a given topology, our algorithm provably finds the best KL minimized solution, whereas their approach is optimal only when the size of the k-gram model tends to infinity. Second, as we show in our experiments, the proposed algorithm performs better compared with that of Deoras et al. (2011).

Arisoy et al. (2014) and Adel et al. (2014) both trained DNN models of different orders as the means to derive probabilities for a k-gram model, rather than focusing on model approximation as the above-mentioned methods do. Further, the methods are applicable to k-gram models specifically, not the larger class of target WFA to which our methods apply.

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. Finally, we present experiments in Section 5.4 addressing scenarios with source WFAs, hence not requiring sampling. First, motivated by low-memory applica-

tions such as (virtual) keyboard decoding (Ouyang et al. 2017), we use our approach to create compact language models in Section 5.4.1. Then, in Section 5.4.2, we use our approach to create compact open-vocabulary character language models from count-thresholded n-grams.

For most experiments (unless otherwise noted) we use the 1996 CSR Hub4 Language Model data, LDC98T31 (English 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 data set 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 consisting of all words that appeared more than 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 (Stolcke 2000). We use this pruned model as a baseline in all our word-based experiments. We use Katz smoothing because it is amenable to pruning (Chelba et al. 2010). The perplexity of this model on the test set is 144.4.[11] All algorithms were implemented using the open-source `OpenFst` and `OpenGrm` n-gram and stochastic automata (`SFst`) libraries[12] with the last library including these implementations (Allauzen et al. 2007; Roark et al. 2012; Allauzen and Riley 2018).

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 Figure 6. 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 (Stolcke 2000) 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}. As before, 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 the source model p_s on A_{greedy}, the

11 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 data set.

12 These libraries are available at `www.openfst.org` and `www.opengrm.org`.

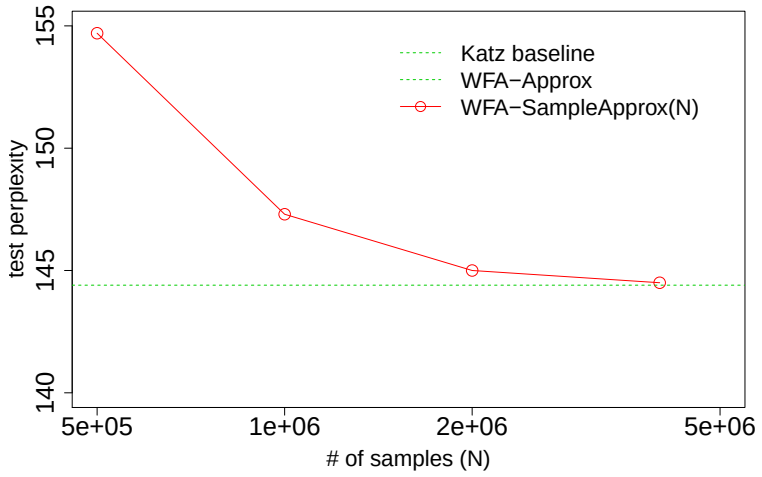

Figure 6
Idempotency: Test set perplexity for Katz baseline and approximations of that baseline trained on the same data. The Katz baseline and Katz WFA-APPROX plots are identical (and thus overlap), while WFA-SAMPLEAPPROX(N) plot converges to the baseline.

Table 1
Test perplexity of greedy pruning and approximated models.

Model size	Greedy pruning	Approximated model
250K	205.7	198.3
500K	177.3	173.0
1M	157.4	155.7
1.5M	149.0	148.4

topology of p_{greedy}. The resulting model has a test perplexity of 155.7, which is smaller than the test perplexity of p_{greedy}. This shows that the approximation algorithm indeed finds a good solution. We repeat the experiment with different pruning thresholds and observe that the approximation algorithm provides a good solution across the resulting model sizes. The results are in Table 1.

5.2 Neural Models to WFA Conversion

Since neural models such as LSTMs give improved performance over n-gram models, we investigate whether 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, which was done in Chen et al. (2019) using the methods presented here.

To explore this, we train an LSTM language model on the training data. The model has 2 LSTM layers with 1,024 states and an embedding size of 1,024. The resulting model has a test perplexity of 60.5. We approximate this model as a WFA in three ways.

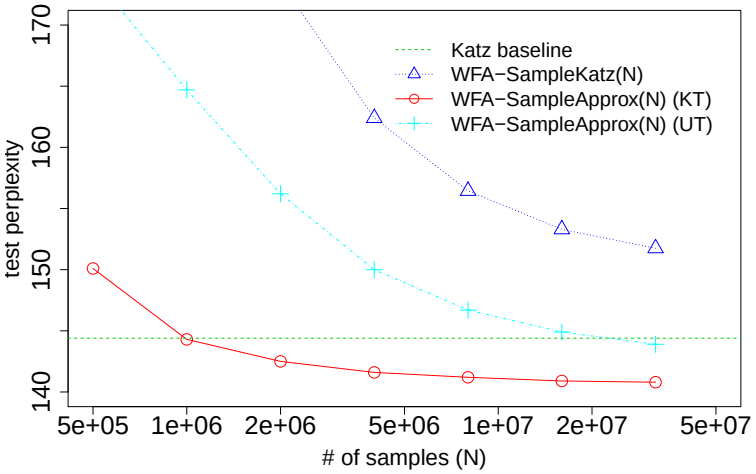

Figure 7
Approximated Neural Models for the English Broadcast News corpus. Test set perplexity for Katz baseline and LSTM models approximated in three ways. One uses LSTM samples to build a new Katz model (WFA-SAMPLEKATZ(N)). The remaining two use our approximation algorithm (WFA-SAMPLEAPPROX(N)), but with different topologies. One topology (KT) is known, using the baseline Katz topology, and one (UT) is unknown, using the samples drawn for WFA-SAMPLEAPPROX(N) to also infer the topology.

The first way is to construct a Katz n-gram model on N LSTM samples and entropy-prune to 2M n-grams, which we denote by WFA-SAMPLEKATZ(N). This approach is very similar to that of Deoras et al. (2011), except that we use Katz smoothing instead of Kneser-Ney smoothing. We used Katz due to the fact that, as stated earlier, Kneser-Ney models are not amenable to pruning (Chelba et al. 2010). The second way is to approximate onto the baseline Katz 2M n-gram topology described above using WFA-SAMPLEAPPROX(N). We refer to this experiment as WFA-SAMPLEAPPROX(N) (KT), where KT stands for known topology. The results are shown in Figure 7. The WFA-SAMPLEKATZ(N) models perform significantly worse than the baseline Katz model even at 32M samples, while the WFA-SAMPLEAPPROX(N) models have better perplexity than the baseline Katz model with as little as 1M samples. With 32M samples this way of approximating the LSTM model as a WFA is 3.6 better in perplexity than the baseline Katz.

Finally, 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. We refer to this experiment as WFA-SAMPLEAPPROX(N) (UT), where UT stands for unknown topology. The results are shown in Figure 7. The approximated models obtained by WFA-SAMPLEAPPROX(N) (UT) perform better than WFA-SAMPLEKATZ(N). However, they do not perform as well as WFA-SAMPLEAPPROX(N) (KT), 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.

We then compare our approximation method with other methods that approximate DNNs with n-gram LMs (Arisoy et al. 2014; Adel et al. 2014). Both these methods require training DNNs of different orders and we used DNNs with two layers with 1,024 nodes and an embedding size of 1,024. The results are in Table 2. The proposed algorithms WFA-SAMPLEAPPROX($\cdot$) perform better than the existing approaches.

Table 2
Test perplexity of k-gram models obtained by different approximation methods for the English Broadcast News corpus. We use 32M samples for both WFA-SAMPLEKATZ(·) and WFA-SAMPLEAPPROX(·).

Model	Test perplexity
Katz	144.4
WFA-SAMPLEKATZ(·)	151.8
Arisoy et al. (2014)	159.6
Adel et al. (2014)	146.7
WFA-SAMPLEAPPROX(·) (KT)	140.8
WFA-SAMPLEAPPROX(·) (UT)	143.9

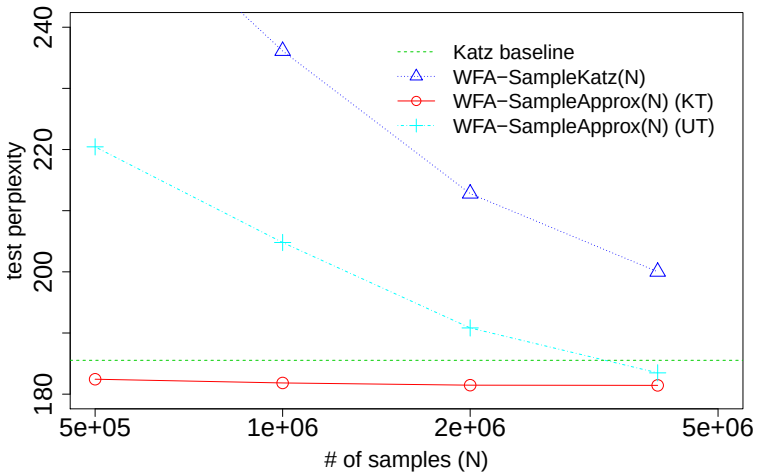

Figure 8
Approximated Neural Models for the Polish (Europarl) corpus. Test set perplexity for Katz baseline and LSTM models approximated in three ways. One uses LSTM samples to build a new Katz model (WFA-SAMPLEKATZ(N)). The remaining two use our approximation algorithm (WFA-SAMPLEAPPROX(N)), but with different topologies. One topology (KT) is known, using the baseline Katz topology, and one (UT) is unknown, using the samples drawn for WFA-SAMPLEAPPROX(N) to also infer the topology.

Experiment on Polish. To repeat the above experiment on a different language and corpus, we turn to the Polish language section[13] of the Europarl corpus (Koehn 2005). We chose Polish for this follow-up experiment due to the fact that it belongs to a different language family than English, is relatively highly inflected (unlike English), and is found in a publicly available corpus. We use the processed form of the corpus and further process it to downcase all the words and remove punctuation. The resulting data set has approximately 13M words in the training set and 210K words in the test set. The selected vocabulary has approximately 30K words, consisting of all words that appeared more than 20 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 (Stolcke 2000). We use this pruned model as a baseline. The results are in Figure 8. The trend

13 https://www.statmt.org/europarl/v7/pl-en.tgz.

Table 3
Test perplexity of k-gram models obtained by different approaches for the Polish (Europarl) corpus. We use 32M samples for both WFA-SAMPLEKATZ(·) and WFA-SAMPLEAPPROX(·).

Model	Test perplexity
Katz	185.5
WFA-SAMPLEKATZ(·)	189.3
Arisoy et al. (2014)	197.8
Adel et al. (2014)	187.1
WFA-SAMPLEAPPROX(·) (KT)	181.4
WFA-SAMPLEAPPROX(·) (UT)	177.0

is similar to that of the English Broadcast News corpus with the proposed algorithm WFA-SAMPLEAPPROX(·) performing better than the other methods. We also compare the proposed algorithms with other neural approximation algorithms. The comparison results are shown in Table 3.

5.3 Lower Bounds on Perplexity

Best Approximation for Target Topology. 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

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

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 English 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 the algorithm finds the global KL divergence minimum, the test perplexity with this topology cannot be improved beyond 121.1, irrespective of the method.

Approximation onto Best Trigram Topology. 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 data sets, the largest shortfall of n-gram models in the approximation is due to the n-gram topology.

5.4 WFA Sources

While distillation of neural models is an important use case for our algorithms, there are other scenarios where approximations will be derived from WFA sources. In such cases, no sampling is required and we can use the algorithms presented in Sections 4.1.1 or 4.1.2, namely, WFA-APPROX. For example, it is not uncommon for applications to place maximum size restrictions on models, so that existing WFA models are too large and must be reduced in size prior to use. Section 5.4.1 presents a couple of experiments focused on character language model size reduction that were motivated by such size restrictions. Another scenario with WFA sources is when, for privacy reasons, no language model training corpus is available in a domain, but only minimum-count-thresholded (i.e., high frequency) word n-gram counts are provided. In Section 5.4.2 we examine the estimation of open-vocabulary character language models from such data.

5.4.1 Creating Compact Language Models

Creating Compact Models for Infrequent Words. In low-memory applications such as on-device keyboard decoding (Ouyang et al. 2017), it is often useful to have a character-level WFA representation of a large set of vocabulary words that act only as unigrams, for example, 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 cutoff) 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 (Mohri 2000) 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 further 25% reduction. Because 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 that is as close as possible to the original, but that is constrained to the minimal deterministic representation that preserves the vocabulary.

To evaluate this approach, we randomly select a 20K sentence subset of the original test set, and represent each selected string as a character-level sequence. 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 that is 25% smaller. Wolf-Sonkin et al. (2019) used the methods presented here to augment the vocabulary of an on-device keyboard to deal with issues related to a lack of standard orthography.

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

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

Topology	Test perplexity	# Transitions
A_{greedy}	155.7	1.13M
Minimize(A_{greedy})	156.4	1.05M
A'_{greedy}	154.1	1.22M
Minimize(A'_{greedy})	154.9	1.13M

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.

5.4.2 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 50), no n-gram can be traced to a specific document, thus providing privacy in the aggregation. This is known as k-anonymity (Samarati 2001).

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. However, character n-gram models might be harder to learn from a k-anonymized corpus.

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 count character 5-grams from the k-anonymized corpus, then remove all n-grams that include 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 in the k-anonymized corpus 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.

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 related to the construction presented in Section 6 of Chen et al. (2019), briefly 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 Source to Character Sequence Source Model.* Briefly, for every state q in the n-gram automaton, the set of words labeling transitions leaving q 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 next state as the original word transition. If q has a backoff transition pointing to its backoff state q', then each new internal state in the character trie backs off to the corresponding state in the character trie leaving q'. The presence of the corresponding state in the character trie leaving q' is guaranteed because the n-gram automaton is backoff-complete, as discussed in Section 3.3.

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, that is, this is still closed vocabulary. To make it open vocabulary, we further back off 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 our 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 because 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 slightly lower bits-per-character than even what we get

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

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. Software Library

All of the algorithms presented here are available in the `OpenGrm` libraries at `http://www.opengrm.org` under the topic: *SFST Library: operations to normalize, sample, combine, and approximate stochastic finite-state transducers.* We illustrate the use of this library by showing how to implement some of the experiments in the previous section.

6.1 Example Data and Models

Instead of Broadcast News, we will use the text of Oscar Wilde's *The Importance of Being Earnest* for our tutorial example. This is a small tutorial corpus that we make available at `http://sfst.opengrm.org`.

This corpus of approximately 1,688 sentences and 18,000-words has been uppercased and had punctuation removed. The first 850 sentences were used as training data and the remaining 838 sentences used as test data. From these, we produce two 1,000-word vocabulary Katz bigram models, the $\sim 6k$ n-gram `earnest_train.mod` and the $\sim 4k$ n-gram `earnest_test.mod`. We also used relative-entropy pruning to create the $\sim 2k$ n-gram `earnest_train.pru`. The data, the steps to generate these models, and how to compute their perplexity using *OpenGrm NGram* are all fully detailed in the `QuickTour` topic at `http://sfst.opengrm.org`.

6.2 Computing the Approximation

The following step shows how to compute the approximation of a φ-WFA model onto a φ-WFA topology. In the example below, the first argument, `earnest_train.mod`, is the source model, and the second argument, `earnest_train.pru`, provides the topology. The result is a φ-WFA whose perplexity can be measured as before.

```
$ sfstapprox -phi_label=0 earnest_train.mod earnest_train.pru \
  >earnest_train.approx
```

An alternative, equivalent way to perform this approximation is to break it into two steps, with the counting and normalization (KL divergence minimization) done separately.

```
$ sfstcount -phi_label=0 earnest_train.mod earnest_train.pru \
  >earnest_train.approx_cnts
$ sfstnormalize -method=kl_min -phi_label=0 \
  earnest_train.approx_cnts >earnest_train.approx
```

We can now use these utilities to run some experiments analogous to our larger Broadcast News experiments by using different target topologies. The results are shown in Table 6. We see in the idempotency experiment that the perplexity of the approximation on the same topology matches the source. In the greedy-pruning experiment, the approximation onto the greedy-pruned topology yields a better perplexity than the greedily-pruned model. Finally, the approximation onto the test-set bigram topology gives a better perplexity than the training-set model since we include all the relevant bigrams.

Table 6
Perplexities for example experiments.

Experiment	Source Model	Topology	Approx. Perplexity
Idempotency	earnest_train.mod	earnest_train.mod	73.41
Comparison to greedy pruning	earnest_train.mod	earnest_train.pru	83.12
Approx. onto best bigram topology	earnest_train.mod	earnest_test.pru	69.68

Table 7
Available Operations in the OpenGrm SFst Library.

Operation	Description
`sfstapprox`	`Approximates a stochastic φ-WFA wrt a backoff-complete φ-WFA.`
`sfstcount`	`Counts from a stochastic φ-WFA wrt a backoff-complete φ-WFA.`
`sfstintersect`	`Intersects two φ-WFAs.`
`sfstnormalize -method=global`	`Globally normalizes a φ-WFA.`
`sfstnormalize -method=kl_min`	`Normalizes a count φ-WFA using KL divergence minimization.`
`sfstnormalize -method=local`	`Locally normalizes a φ-WFA.`
`sfstnormalize -method=phi`	`φ-normalizes a φ-WFA.`
`sfstperplexity`	`Computes perplexity of a stochastic φ-WFA.`
`sfstrandgen`	`Randomly generates paths from a stochastic φ-WFA.`
`sfstshortestdistance`	`Computes the shortest distance on a φ-WFA.`
`sfsttrim`	`Removes useless states and transitions in a φ-WFA.`

6.3 Available Operations

Table 7 lists the available command-line operations in the `OpenGrm SFst` library. We show command-line utilities here; there are corresponding C++ library functions that can be called from within a program; see `http://sfst.opengrm.org`.

7. Summary and Discussion

In this article, we have presented an algorithm for minimizing the KL-divergence between a probabilistic source model over sequences and a WFA target model. That our algorithm is general enough to permit source models of arbitrary form (e.g., RNNs) and deriving an appropriate WFA topology—something we do not really touch on in this article—is particularly important.

References

Adel, Heike, Katrin Kirchhoff, Ngoc Thang Vu, Dominic Telaar, and Tanja Schultz. 2014. Comparing approaches to convert recurrent neural networks into backoff language models for efficient decoding. In *Fifteenth Annual Conference of the International Speech Communication Association (Interspeech)*, pages 651–655.

Aho, Alfred V. and Margaret J. Corasick. 1975. Efficient string matching: An aid to bibliographic search. *Communications of the ACM*, 18(6):333–340. https://doi.org/10.1145/360825.360855

Albert, Jürgen and Jarkko Kari. 2009. Digital image compression. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*, Springer, pages 453–479. https://doi.org/10.1007/978-3-642-01492-5_11

Allauzen, Cyril, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing language models. In *Proceedings of ACL*, pages 40–47. https://doi.org/10.3115/1075096.1075102

Allauzen, Cyril, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. 2007. OpenFst Library. http://www.openfst.org.

Allauzen, Cyril and Michael D. Riley. 2018. Algorithms for weighted finite automata with failure transitions. In *International Conference on Implementation and Application of Automata*, pages 46–58. https://doi.org/10.1007/978-3-319-94812-6_5

Angluin, Dana. 1987. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106. https://doi.org/10.1016/0890-5401(87)90052-6

Angluin, Dana. 1988. Identifying languages from stochastic examples. Technical Report YALEU /DCS /RR-614, Yale University.

Arisoy, Ebru, Stanley F. Chen, Bhuvana Ramabhadran, and Abhinav Sethy. 2014. 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. https://doi.org/10.1109/TASLP.2013.2286919

Balle, Borja, Xavier Carreras, Franco M. Luque, and Ariadna Quattoni. 2014. Spectral learning of weighted automata. *Machine Learning*, 96(1-2):33–63. https://doi.org/10.1007/s10994-013-5416-x

Balle, Borja and Mehryar Mohri. 2012. Spectral learning of general weighted automata via constrained matrix completion. In *Advances in Neural Information Processing Systems*, pages 2159–2167.

Breuel, Thomas M. 2008. The OCRopus open source OCR system. In *Proceedings of IS&T/SPIE 20th Annual Symposium*. https://doi.org/10.1117/12.783598

Carrasco, Rafael C. 1994. Accurate computation of the relative entropy between stochastic regular grammars. *RAIRO-Theoretical Informatics and Applications*, 31(5):437–444. https://doi.org/10.1051/ita/1997310504371

Carrasco, Rafael C. and José Oncina. 1994. Learning stochastic regular grammars by means of a state merging method. In *International Colloquium on Grammatical Inference*, pages 139–152. https://doi.org/10.1007/3-540-58473-0_144

Carrasco, Rafael C. and José Oncina. 1999. Learning deterministic regular grammars from stochastic samples in polynomial time. *RAIRO-Theoretical Informatics and Applications*, 33(1):1–19. https://doi.org/10.1051/ita:1999102

Chelba, Ciprian, Thorsten Brants, Will Neveitt, and Peng Xu. 2010. Study on interaction between entropy pruning and Kneser-Ney smoothing. In *Eleventh Annual Conference of the International Speech Communication Association (Interspeech)*, pages 2422–2425.

Chen, Mingqing, Ananda Theertha Suresh, Rajiv Mathews, Adeline Wong, Françoise Beaufays, Cyril Allauzen, and Michael Riley. 2019. Federated learning of N-gram language models. In *Proceedings of the 23rd Conference on Computational Natural Language Learning (CoNLL)*, pages 121–130. https://doi.org/10.18653/v1/K19-1012

Chen, Stanley and Joshua Goodman. 1998. An empirical study of smoothing techniques for language modeling. TR-10-98, Harvard University.

Cicchello, Orlando and Stefan C. Kremer. 2003. Inducing grammars from sparse data sets: A survey of algorithms and results. *Journal of Machine Learning Research*, 4(Oct):603–632.

Cortes, Corinna, Mehryar Mohri, Ashish Rastogi, and Michael Riley. 2008. On the computation of the relative entropy of probabilistic automata. *International Journal of Foundations of Computer Science*, 19(01):219–242. https://doi.org/10.1142/S0129054108005644

Dempster, Arthur P., Nan M. Laird, and Donald B. Rubin. 1977. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39(1):1–38. https://doi.org/10.1111/j.2517-6161.1977.tb01600.x

Deoras, Anoop, Tomáš Mikolov, Stefan Kombrink, Martin Karafiát, and Sanjeev Khudanpur. 2011. Variational approximation of long-span language models for LVCSR. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5532–5535. https://doi.org/10.1109/ICASSP.2011.5947612

Dupont, Pierre. 1996. Incremental regular inference. In *International Colloquium on Grammatical Inference*, pages 222–237. https://doi.org/10.1007/BFb0033357

Durbin, Richard, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison. 1998. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*, Cambridge University Press. https://doi.org/10.1017/CBO9780511790492

Ebden, Peter and Richard Sproat. 2015. The Kestrel TTS text normalization system. *Natural Language Engineering*, 21(3):333–353. https://doi.org/10.1017/S1351324914000175

Eisner, Jason. 2001. Expectation semirings: Flexible EM for learning finite-state transducers. In *Proceedings of the ESSLLI Workshop on Finite-state Methods in NLP*, pages 1–5.

Giles, C. Lee, Clifford B. Miller, Dong Chen, Hsing-Hen Chen, Guo-Zheng Sun, and Yee-Chun Lee. 1992. Learning and extracting finite state automata with second-order recurrent neural networks. *Neural Computation*, 4(3):393–405. https://doi.org/10.1162/neco.1992.4.3.393

Gold, E. Mark. 1967. Language identification in the limit. *Information and Control*, 10(5):447–474. https://doi.org/10.1016/S0019-9958(67)91165-5

Gold, E. Mark. 1978. Complexity of automaton identification from given data. *Information and Control*, 37(3):302–320. https://doi.org/10.1016/S0019-9958(78)90562-4

Hard, Andrew, Kanishka Rao, Rajiv Mathews, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*.

Hellsten, Lars, Brian Roark, Prasoon Goyal, Cyril Allauzen, Françoise Beaufays, Tom Ouyang, Michael Riley, and David Rybach. 2017. Transliterated mobile keyboard input via weighted finite-state transducers. In *FSMNLP 2017*, pages 10–19. https://doi.org/10.18653/v1/W17-4002

Horst, Reiner and Nguyen V. Thoai. 1999. DC programming: Overview. *Journal of Optimization Theory and Applications*, 103(1):1–43. https://doi.org/10.1023/A:1021765131316

Iglesias, Gonzalo, Cyril Allauzen, William Byrne, Adrià de Gispert, and Michael Riley. 2011. Hierarchical phrase-based translation representations. In *EMNLP 2011*, pages 1373–1383.

Jacobsson, Henrik. 2005. Rule extraction from recurrent neural networks: A taxonomy and review. *Neural Computation*, 17(6):1223–1263. https://doi.org/10.1162/0899766053630350

Katz, Slava M. 1987. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustic, Speech, and Signal Processing*, 35(3):400–401. https://doi.org/10.1109/TASSP.1987.1165125

Koehn, Philipp. 2005. Europarl: A parallel corpus for statistical machine translation. In *MT Summit*, 5:79–86.

Konečnỳ, Jakub, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. *arXiv preprint arXiv:1610.05492*.

Lecorvé, Gwénolé and Petr Motlicek. 2012. 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 (Interspeech)*, pages 1668–1671.

McMahan, Brendan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pages 1273–1282.

Mohri, Mehryar. 1997. String-matching with automata. *Nordic Journal of Computing*, 4(2):217–213.

Mohri, Mehryar. 2000. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1-2):177–201. https://doi.org/10.1016/S0304-3975(98)00115-7

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

Mohri, Mehryar. 2009. Weighted automata algorithms. In M. Droste, W. Kuich, and H. Vogler, editors, *Handbook of Weighted Automata*. Springer, pages 213–254. https://doi.org/10.1007/978-3-642-01492-5_6

Mohri, Mehryar, Fernando C. N. Pereira, and Michael Riley. 2008. Speech recognition with weighted finite-state transducers. In J. Benesty, M. M. Sondhi, and Y. A. Huang, editors, *Handbook on Speech Processing and Speech Communication*, Springer, pages 559–584. https://doi.org/10.1007/978-3-540-49127-9_28

Novak, Josef R., Nobuaki Minematsu, and Keikichi Hirose. 2013. Failure transitions for joint n-gram models and G2P conversion. In *Fourteenth Annual Conference of the International Speech Communication Association (Interspeech)*, pages 1821–1825.

Okudono, Takamasa, Masaki Waga, Taro Sekiyama, and Ichiro Hasuo. 2020. Weighted automata extraction from recurrent neural networks via regression on state spaces. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 34:5306–5314. https://doi.org/10.1609/aaai.v34i04.5977

Oncina, José and Pedro Garcia. 1992. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*. World Scientific, pages 99–108. https://doi.org/10.1142/9789812797919_0007

Ouyang, Tom, David Rybach, Françoise Beaufays, and Michael Riley. 2017. Mobile keyboard input decoding with finite-state transducers. *arXiv preprint arXiv:1704.03987*.

Parekh, Rajesh and Vasant Honavar. 2000. Grammar inference, automata induction, and language acquisition. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*, pages 727–764.

Pitt, Leonard. 1989. Inductive inference, DFAs, and computational complexity. In *International Workshop on Analogical and Inductive Inference*, pages 18–44. https://doi.org/10.1007/3-540-51734-0_50

Roark, Brian, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai. 2012. The OpenGrm open-source finite-state grammar software libraries. *Proceedings of the ACL 2012 System Demonstrations*, pages 61–66.

Samarati, Pierangela. 2001. Protecting respondents' identities in microdata release. *IEEE Transactions on Knowledge and Data Engineering*, 13(6):1010–1027. https://doi.org/10.1109/69.971193

Sriperumbudur, Bharath K. and Gert R. G. Lanckriet. 2009. On the convergence of the concave-convex procedure. In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, pages 1759–1767.

Stolcke, Andreas. 2000. Entropy-based pruning of backoff language models. *arXiv preprint cs/0006025*.

Sundermeyer, Martin, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. *Thirteenth Annual Conference of the International Speech Communication Association*, pages 194–197.

Suresh, Ananda Theertha, Brian Roark, Michael Riley, and Vlad Schogol. 2019. Distilling weighted finite automata from arbitrary probabilistic models. In *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP)*, pages 87–97. https://doi.org/10.18653/v1/W19-3112

Tiňo, Peter and Vladimir Vojtek. 1997. Extracting stochastic machines from recurrent neural networks trained on complex symbolic sequences. In *Proceedings of the First International Conference on Knowledge-Based Intelligent Electronic Systems*, volume 2, pages 551–558, IEEE.

Weiss, Gail, Yoav Goldberg, and Eran Yahav. 2018. Extracting automata from recurrent neural networks using queries and counterexamples. In *International Conference on Machine Learning*, pages 5244–5253.

Weiss, Gail, Yoav Goldberg, and Eran Yahav. 2019. Learning deterministic weighted automata with queries and counterexamples. *Advances in Neural Information Processing Systems*, pages 8560–8571.

Wolf-Sonkin, Lawrence, Vlad Schogol, Brian Roark, and Michael Riley. 2019. Latin script keyboards for South Asian languages with finite-state normalization. In *Proceedings of the 14th International Conference on Finite-State Methods and Natural Language Processing (FSMNLP)*, pages 108–117. https://doi.org/10.18653/v1/W19-3114