Distilling weighted finite automata from arbitrary probabilistic models

Ananda Theertha Suresh, Brian Roark, Michael Riley, Vlad Schogol
{ theertha, roark, riley, vlads }@google.com
Google Research

Abstract

Weighted finite automata (WFA) are often used to represent probabilistic models, such as n-gram language models, since they are efficient for recognition tasks in time and space. The probabilistic source to be represented as a WFA, however, may come in many forms. Given a generic probabilistic model over sequences, we propose an algorithm to approximate it as a weighted finite automaton such that the Kullback-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, both of which can be performed efficiently. We demonstrate the usefulness of our approach on some tasks including distilling n-gram models from neural models.

1 Introduction

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

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

Such a model might be Markovian, where

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

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 language model (Sundermeyer et al., 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 et al., 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 models for on-device (virtual) keyboard decoding (Ouyang et al., 2017), where WFA models are used due to space and time constraints. However, storing the training data in a centralized server and training k-gram or other WFA models directly may not be feasible due to privacy constraints (Hard et al., 2018). Alternatively, an LSTM model can be trained by federated learning (Konečnỳ et al., 2016; Hard et al., 2018), then converted to a WFA at the server for fast on-device inference. This not only may improve performance, but also provide additional privacy.

We 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 et al., 2003; Novak et al., 2013; Hellsten et al., 2017). The inclusion of failure transitions will complicate our analysis and algorithms but is highly desirable in applications such as keyboard decoding. Further, to avoid redundancy that leads to inefficiency, we assume the target model is *deterministic*, which requires at each state there is at most one transition labeled with a given symbol.

The approximation problem can be divided into two steps: (1) select an unweighted automaton A that will serve as the *topology* of the target automaton and (2) weight the automaton A to form our weighted approximation $\hat{A}$. The main goal of this paper is the latter determination of the automaton's

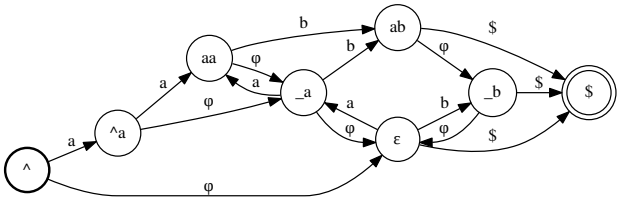

Figure 1: 3-gram topology example derived from the corpus aab. States are labeled with the context that is remembered, $\wedge$ denotes the initial context, ϵ the empty context, \$ the final context (and terminates accepted strings), and _ matches any symbol in a context. Failure transitions, labeled with φ, implement backoff from histories xy to $\_y$ to ϵ.

weighting in the approximation.

In some applications, the topology may be unknown. In such cases, one choice is to build a k-gram deterministic finite automaton (DFA) topology from a corpus drawn from S (Allauzen et al., 2003). This could be from an existing corpus or from random samples drawn from S. Figure 1 shows a trigram topology for the very simple corpus aab. This representation makes 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.

In previous work, there have been various approaches for estimating weighted automata. Methods include state merging and weight estimation from a prefix tree data representation (Carrasco and Oncina, 1994, 1999), the EM algorithm (Dempster et al., 1977) applied to fully connected HMMs or specific topologies (Eisner, 2001) and spectral methods applied to automata (Balle and Mohri, 2012; Balle et al., 2014). For approximating neural network (NN) models as WFAs, methods have been proposed to build n-gram models from RNN samples (Deoras et al., 2011), from DNNs trained at different orders (Arisoy et al., 2014; Adel et al., 2014), and from RNNs with quantized hidden states (Tiňo and Vojtek, 1997; Lecorvé and Motlicek, 2012).

Our paper is distinguished in several respects from previous work. First, our general approach does not depend on the form the source distribution. Second, our targets are a wide class of deterministic automata with failure transitions. Third, we search for the minimal KL divergence between the source and target distributions, given a fixed target topology.

We remark that if the source probabilistic model is represented as a WFA, our approximation will in general give a different solution than forming the finite-state intersection with the topology and *weight-pushing* to normalize the result (Mohri, 2009; Mohri et al., 2008). Our approximation has the same states as the topology whereas a weight-pushed intersection could have many more states and and is not an approximation, but an exact representation, of the source distribution.

Before presenting and validating algorithms for a minimum KL divergence approximation when either the source itself is finite-state or not (in which case sampling is involved), we next present the theoretical formulation of the problem and the minimum KL divergence approximation.

2 Theoretical analysis

2.1 Probabilistic models

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

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

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

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

for all i, n, x^i, x_{i+1}^n, where $q(x^i)$ is the state 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

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

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

$$\mathcal{L}(p) \triangleq \{x^n \in \Sigma^* : p(x^n) > 0, \\ x_n = \$ \ \wedge \ x_i \neq \$\,, i < n\}. \quad (1)$$

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

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

$$D(p_s||p_a) = \sum_{x^n} p_s(x^n) \log \frac{p_s(x^n)}{p_a(x^n)}, \quad (2)$$

where for notational simplicity, we adopt the notion $0/0 = 1$ and $0 \log(0/0) = 0$ throughout the paper. Note that for the KL divergence to be finite, we need $\mathcal{L}(p_s) \subseteq \mathcal{L}(p_a)$. We first reduce the minimum KL divergence to a family of probabilistic models $\mathscr{P}$ (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*

$$\operatorname*{argmin}_{p_a \in \mathscr{P}} D(p_s||p_a) \\ = \operatorname*{argmax}_{p_a \in \mathscr{P}} \sum_{q_a \in Q_a} \sum_{x \in \Sigma} c(x, q_a) \log p_a(x|q_a) \quad (3)$$

where $c(x, q_a)$ is given by

$$\sum_{q_s \in Q_s} \sum_{i=0}^{\infty} \sum_{x^i : q_s(x^i) = q_s, q_a(x^i) = q_a} p_s(x^i)\, p_s(x|q_s) \quad (4)$$

and does not depend on p_a.

Proof is omitted due to space limitations.

2.2 Weighted finite automata

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

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

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

Transitions e_1 and e_2 are *consecutive* if $n[e_i] = p[e_{i+1}]$. A path $\pi = e_1 \cdots e_n \in E^*$ is a finite sequence of consecutive transitions, the source and destination states of which we denote by $p[\pi]$ and $n[\pi]$, respectively. 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 1, we assume a symbol $\$ \in \Sigma$ such that

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

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

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

2.3 Weighted finite automata with failure transitions

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

A φ transition does not add to a path label; it consumes no input. However it is followed only when the input can not be read immediately. Specifically, a path $e_1 \cdots e_n$ in a φ-WFA is *disallowed* if it contains a subpath $e_i \cdots e_j$ such that $\ell[e_k] = \varphi$ for all k, $i \leq k < j$, and there is another transition $e \in E$ such that $p[e_i] = p[e]$ and

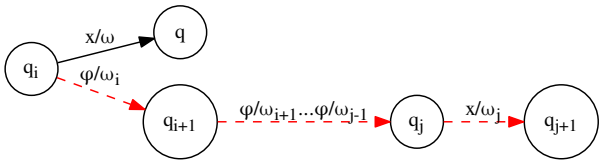

Figure 2: The (dashed red) path $e_i = (q_i, \varphi, \omega_i, q_{i+1})$ to $e_j = (q_j, x, \omega_j, q_{j+1})$ is disallowed since x can be read already on $e = (q_i, x, \omega, q)$.

$\ell[e_j] = \ell[e] \in \Sigma$ (see Figure 2). Since the label $x = l[e_j]$ can be read on e, we do not follow the failure transitions to read it on e_j as well.

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

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

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

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

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

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

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

A deterministic φ-WFA is *backoff-complete* if a failure transition from state q to q' implies $L[q] \cap \Sigma \subseteq L[q'] \cap \Sigma$. Further, if $\varphi \notin L[q']$, then the containment is strict: $L[q] \cap \Sigma \subset L[q'] \cap \Sigma$. In other words, if a symbol can be read immediately from a state q it can also be read from a state failing (*backing-off*) from q and if q' does not have a backoff arc, then at least one additional label can be read from q' that cannot be read from q. For example, the topology depicted in Figure 1 has 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 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 2.1. Note the distribution p_a^* at a state q is defined over the $\varphi-$extended transitions $E^*[q]$ where p_a in the previous section is defined over the transitions $E[q]$. It is convenient to define a companion distribution $p_a \in P(A)$ to p_a^* as follows:[3] given a symbol $x \in \Sigma \cup \{\varphi\}$ and state $q \in Q$, define $p_a(x|q) \triangleq p_a^*(x|q)$ when $x \in L[q] \cap \Sigma$, $p_a(\varphi|q) \triangleq 1 - \sum_{x \in L[q] \cap \Sigma} p_a^*(x|q)$, 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

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

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

$$d(q, q') = 1 - \sum_{x \in L[q] \cap \Sigma} p_a(x|q'). \tag{5}$$

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

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

$$\alpha(q, q') = \prod_{e \in \pi_\varphi(q, q')} \frac{p_a(\varphi|p[e])}{d(p[e], n[e])} \tag{6}$$

where this quantity is taken to be 1 when the fail-

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

ure path is empty ($q = q'$). Finally define

$$p_a^*(x|q) = \begin{cases} \alpha(q, q^x) p_a(x|q^x), & x \in L^*[q] \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

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 (6) to be well-defined, we need $d(p[e], n[e]) > 0$. To ensure this condition, we restrict $\mathscr{P}(A)$ to contain distributions such that $p_a(x|q) \geq \epsilon$ for each $x \in L[q]$.[4]

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. We can restate our goal in terms of the companion distribution $p_a \in \mathscr{P}(A)$. Let $B_n(q)$ be the set of states in A that back-off to state q in n failure transitions and let $B(q) = \bigcup_{n=0}^{|Q_a|} B_n(q)$.

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

$$\operatorname*{argmin}_{p_a^* \in \mathscr{P}^*(A)} D(p_s||p_a^*) = \tag{8}$$
$$\operatorname*{argmax}_{p_a \in \mathscr{P}(A)} \sum_{q \in Q_a} \Bigg\{ \sum_{x \in L[q]} C(x, q) \log p_a(x|q) - \sum_{q_0 \in B_1(q)} C(\varphi, q_0) \log d(q_0, q) \Bigg\},$$

where

$$C(x, q) = \sum_{q_a \in B(q)} c(x, q_a) \mathbf{1}_{q = q_a^x}, \quad x \in \Sigma \tag{9}$$

$$C(\varphi, q) = \sum_{q_a \in B(q)} \sum_{x \in \Sigma} c(x, q_a) \mathbf{1}_{x \notin L[q]} \tag{10}$$

and do not depend on p_a.

Proof is omitted due to space limitations.

The quantity in braces in the statement of Lemma 2 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.

3 Algorithms

Approximating a probabilistic source algorithmically as a weighted finite automaton requires two steps: (1) compute the quantity $C(x, q)$ in Lemma 2 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.

3.1 Counting

How the counts are computed will depend on the source model form. We divide this into two cases.

3.1.1 φ-WFA source and target

When the source and target models are represented as φ-WFAs we compute $C(x, q_a)$ from Lemma 2. From Equation 9 this can be written as

$$C(x, q) = \sum_{q_a \in B(q)} \sum_{q_s \in Q_s} \gamma(q_s, q_a) p_s(x|q_s) \mathbf{1}_{q = q_a^x} \tag{11}$$

with $x \in \Sigma$ and

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

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

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

where $S \cap A$ is the weighted intersection of automata S and A formed using an efficient φ-WFA intersection that compactly retains failure transitions in the result, as described in Allauzen and Riley (2018). The quantity $\gamma(q_s, q_a)$ 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). Equation 11 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.

This computation can be simplified 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 source state in A of the compensated positive transition (see Figure 3).[5]

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

[5] The construction illustrated in Figure 3 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).

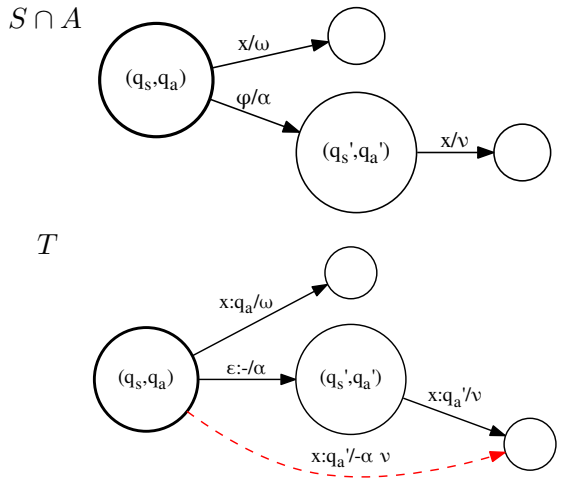

Figure 3: A φ-WFA is transformed into an equivalent WFA by replacing each failure transition by an ϵ-transition. To compensate for the formerly disallowed paths, new (dashed red) negatively-weighted transitions are added. The result is promoted to a transducer T with the output label used to keep track of the source state in A of the compensated positive transition.

Then, for $x \in \Sigma$,

$$C(x, q) = \sum_{((q_s,q_a),x,q,w,(q'_s,q'_a))\in E_T} \gamma_T(q_s, q_a)w \tag{12}$$

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

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

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

Thus

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

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

3.1.2 Arbitrary source and φ-WFA target

In some cases, the source is a distribution with possibly infinite states, e.g., LSTMs. For these sources, computing $C(x, q)$ can be computationally intractable as (11) requires a summation over all possible states in the source machine, Q_s. We propose to use a sampling approach to approximate $C(x, q)$ for these cases. Let $x(1), x(2), \ldots, x(m)$ be independent random samples from p_s. Instead of $C(x, q)$, we propose to use

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

with $x \in \Sigma$ and where

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

Observe that the expectation $\mathbb{E}[\hat{\gamma}(q_s, q_a)]$ is given by

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

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.

3.2 KL divergence minimization

As noted before, the quantity in braces in the statement of Lemma 2 depends on the distribution p_a only at state q so the minimum KL divergence $D(p_s||p_a^*)$ can be found by maximizing that quantity independently for each state.

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

$$\begin{aligned} \operatorname*{argmax}_{\mathbf{y}} &\sum_{x\in L[q]} C(x, q) \log y_x - \\ &\sum_{q_0\in B_1(q)} C(\varphi, q_0) \log\Big(1 - \sum_{x\in L[q_0]\cap\Sigma} y_x\Big) \end{aligned} \tag{13}$$

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

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

Algorithm KL-MINIMIZATION

Notation:

- $y_x = p_a(x|q)$ for $x \in L(q)$
- $C(x, q)$ from Equations 9 and 10
- $C(q) = \sum_{x' \in L[q]} C(x', q)$
- $f(x, q, \mathbf{y}^n)$ from Equation 15
- $\text{lb} = \max_{x \in L[q]} f(x, q, \mathbf{y}^n) + C(x, q)$
- $\text{ub} = \max_{x \in L[q]} f(x, q, \mathbf{y}^n) + C(q)$
- $k = |L[q]|$
- $\epsilon =$ lower bound on y_x

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

Initialization: Initialize:

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

Iteration: Until convergence do:

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

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

Figure 4: KL-MINIMIZATION Algorithm

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

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

be the function domain. 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. Using this procedure on Equation 13 gives $\mathbf{y}^{n+1}$ as

$$\operatorname*{argmax}_{\mathbf{y} \in \Omega} \sum_{x \in L[q]} \left\{ C(x, q) \log y_x + y_x f(x, q, \mathbf{y}^n) \right\} \tag{14}$$

where

$$f(x, q, \mathbf{y}^n) = \sum_{q_0 \in B_1(q)} \frac{C(\varphi, q_0) \mathbf{1}_{x \in L[q_0] \cap \Sigma}}{1 - \sum_{x' \in L[q_0] \cap \Sigma} y_{x'}^n}. \tag{15}$$

Observe that $1 - \sum_{x' \in L[q_0] \cap \Sigma} y_{x'}^n \geq \epsilon$ 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 (14) which leads to a stationary point of the objective.

Lemma 3. *Solution to* (14) *given by*

$$y_x^{n+1} = \max \left(\frac{C(x, q)}{\lambda - f(x, q, \mathbf{y}^n)}, \epsilon \right), \tag{16}$$

where λ *is chosen so that* $\sum_x y_x^n = 1$ *and lies in*

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

Proof is omitted due to space limitations.

From this, we form the KL-MINIMIZATION algorithm in Figure 4. Observe that if all the counts are zero, then 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. Since Ω is a convex compact set and both the functions are continuous and differentiable in Ω, the KL-MINIMIZATION converges to a stationary point (Sriperumbudur and Lanckriet, 2009, Theorem 4).

4 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 3.1.1 followed by the KL-MINIMIZATION algorithm of Section 3.2. Similarly, we use WFA-SAMPLEAPPROX(N) to denote the sampled counting described in Section 3.1.2 with N sampled sentences followed by KL-MINIMIZATION.

We first give experimental evidence that supports the theory in Section 4.1. We then show how to approximate neural models as WFAs in Section 4.2. We also use the proposed method to provide lower bounds on the perplexity given a target topology in Section 4.3.

For all the experiments we use the 1996 CSR Hub4 Language Model data, LDC98T31 from the Broadcast News (BN) task. We use the processed form of the corpus and further process it to downcase all the words and remove punctuation. The resulting dataset has 132M words in the training set, 20M words in the test set, and has 240K unique words. From this, 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), which we use as a baseline in all our experiments. We use Katz smoothing since it is amenable to pruning (Chelba et al., 2010). The perplexity of this model on the test set is 144.4.[7] All algorithms were implemented using the open-source `OpenFst` and `OpenGrm` n-gram and stochastic automata (`SFst`) libraries[8] with the last library including these implementations (Allauzen et al., 2007; Roark et al., 2012; Allauzen and Riley, 2018).

4.1 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 shown in Section 3.2, when 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 5. 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.

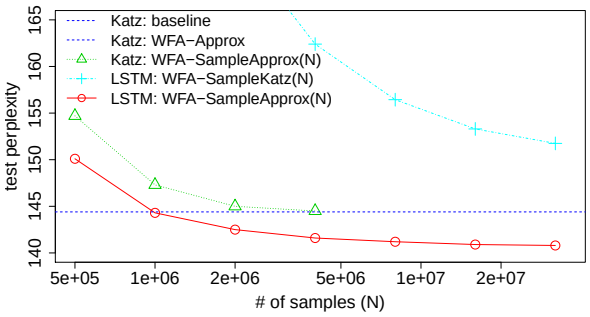

Figure 5: Test set perplexity for Katz baseline and various approximations of that baseline and of an LSTM model trained on the same data. Note that the Katz baseline and Katz WFA-Approx plots are identical.

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}. In particular, let p_s again be the $2M$ n-gram Katz model described above. We prune it to have 1M n-grams and obtain p_{greedy}, which has a test perplexity of 157.4. We then approximate p_s on A_{greedy} and the resulting model has test perplexity of 155.6, which is smaller than the test perplexity of p_{greedy}. This shows that the approximation algorithm indeed finds a good solution.

4.2 Neural models to WFA conversion

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

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

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

vate on-device inference.

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

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). The second way is is to approximate onto the baseline Katz 2M n-gram topology described above using WFA-SAMPLEAPPROX(N). The results are included in Figure 5. It shows that the WFA-SAMPLEKATZ(N) model performs 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.

4.3 Lower bounds on perplexity

The neural model in Section 4.2 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, in our final experiment.

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

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

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

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

which is the model with topology A that has minimal KL divergence from the test distribution $\hat{p}_t$. This can be computed using WFA-APPROX . If we use this approach on the BN test set with the 2M n-gram Katz model, the result has a perplexity of 121.1. This demonstrates 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.

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

5 Summary

In this paper, we have presented an algorithm for minimizing the KL-divergence between a probabilistic source model over sequences and a WFA target model. Our algorithm is general enough to permit source models of arbitrary form (e.g., RNNs) and a wide class of target WFA models, importantly including those with failure transitions, such as n-gram models. We provide some experimental validation of our algorithm, including demonstrating that it is well-behaved in common scenarios and that it yields improved performance over baseline n-gram models using the same WFA topology. Additionally, we use our methods to provide lower bounds on how well a given WFA topology can model a given test set. All of the algorithms reported here are available in the open-source OpenGrm libraries at `http://www.opengrm.org`.

In addition to the above-mentioned results, we also demonstrated that optimizing the WFA topology for the given test set yields far better perplexities than were obtained using WFA topologies derived from training data alone, suggesting that the problem of deriving an appropriate WFA topology – something we do not really touch on in this paper – is particularly important.

References

Heike Adel, 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*.

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

Jürgen Albert and Jarkko Kari. 2009. Digital image compression. In *Handbook of weighted automata*. Springer.

Cyril Allauzen, Mehryar Mohri, and Brian Roark. 2003. Generalized algorithms for constructing language models. In *Proceedings of ACL*, pages 40–47.

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

Cyril Allauzen 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. Springer.

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

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

Ebru Arisoy, 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.

Borja Balle, Xavier Carreras, Franco M Luque, and Ariadna Quattoni. 2014. Spectral learning of weighted automata. *Machine learning*, 96(1-2):33–63.

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

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

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

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

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

Ciprian Chelba, 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*.

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

Orlando Cicchello 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.

Corinna Cortes, 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.

Arthur P Dempster, 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)*, pages 1–38.

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

Pierre Dupont. 1996. Incremental regular inference. In *International Colloquium on Grammatical Inference*, pages 222–237. Springer.

Richard Durbin, Sean R. Eddy, Anders Krogh, and Graeme J. Mitchison. 1998. *Biological Sequence Analysis: Probabilistic Models of Proteins and Nucleic Acids*. Camb. Univ. Press.

Peter Ebden and Richard Sproat. 2015. The kestrel tts text normalization system. *Natural Language Engineering*, 21(3):333–353.

Jason Eisner. 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.

C Lee Giles, 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.

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

E Mark Gold and The RAND Corporation. 1967. Language identification in the limit. *Information and control*, 10(5):447–474.

Andrew Hard, 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*.

Lars Hellsten, 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.

Reiner Horst and Nguyen V Thoai. 1999. Dc programming: overview. *Journal of Optimization Theory and Applications*, 103(1):1–43.

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

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

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

Jakub Konečnỳ, 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*.

Gwénolé Lecorvé 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*.

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

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

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

Mehryar Mohri. 2009. Weighted automata algorithms. In *Handbook of Weighted Automata*, pages 213–254. Springer.

Mehryar Mohri, Fernando C. N. Pereira, and Michael Riley. 2008. Speech recognition with weighted finite-state transducers. In *Handbook on speech proc. and speech comm.* Springer.

Josef R Novak, Nobuaki Minematsu, and Keikichi Hirose. 2013. Failure transitions for joint n-gram models and g2p conversion. In *INTERSPEECH*, pages 1821–1825.

José Oncina and Pedro Garcia. 1992. Identifying regular languages in polynomial time. In *Advances in Structural and Syntactic Pattern Recognition*, pages 99–108. World Scientific.

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

Rajesh Parekh and Vasant Honavar. 2000. Grammar inference, automata induction, and language acquisition. *Handbook of natural language processing*, pages 727–764.

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

Brian Roark, 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.

Bharath K Sriperumbudur and Gert RG 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. Curran Associates Inc.

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

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. Lstm neural networks for language modeling. In *Thirteenth annual conference of the international speech communication association*.

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

Gail Weiss, Yoav Goldberg, and Eran Yahav. 2017. Extracting automata from recurrent neural networks using queries and counterexamples. *arXiv preprint arXiv:1711.09576*.

A Proofs

- Proof of Lemma 1 in Figure 6.
- Proof of Lemma 2 in Figure 7.
- Proof of Lemma 3 in Figure 8.

Figure 6: Proof of Lemma 1

Proof.

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

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

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

Therefore

$$
\begin{aligned}
&\operatorname*{argmin}_{p_a \in \mathscr{P}} D(p_s||p_a) \\
&= \operatorname*{argmin}_{p_a \in \mathscr{P}} \sum_{q_s \in Q_s} \sum_{q_a \in Q_a} \sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1} : q_s(x^{i-1}) = q_s, q_a(x^{i-1}) = q_a) \sum_{x_i} p_s(x_i|q_s) \log \frac{p_s(x_i|q_s)}{p_a(x_i|q_a)} \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}} \sum_{q_s \in Q_s} \sum_{q_a \in Q_a} \sum_{i=1}^{\infty} \sum_{x^{i-1}} p_s(x^{i-1} : q_s(x^{i-1}) = q_s, q_a(x^{i-1}) = q_a) \sum_{x_i} p_s(x_i|q_s) \log p_a(x_i|q_a) \\
&= \operatorname*{argmax}_{p_a \in \mathscr{P}} \sum_{q_a \in Q_a} \sum_{x_i} c(x_i, q_a) \log p_a(x_i|q_a)
\end{aligned}
$$

Replacing x_i by x yields the lemma.

□

Figure 7: Proof of Lemma 2

Proof. From Lemma 1, Equation 7 and the previously shown 1:1 correspondence between each distribution $p_a^* \in P^*(A)$ and its companion distribution $p_a \in P(A)$

$$\begin{aligned}
\underset{p_a^* \in \mathscr{P}^*(A)}{\operatorname{argmin}} D(p_s||p_a^*) &= \underset{p_a \in \mathscr{P}(A)}{\operatorname{argmax}} \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \alpha(q_a, q_a^x) p_a(x|q_a^x) \\
&= \underset{p_a \in \mathscr{P}(A)}{\operatorname{argmax}} \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \prod_{e \in \pi_\varphi(q_a, q_a^x)} \frac{p_a(\varphi|p[e])}{d(p[e], n[e])} p_a(x|q_a^x) \\
&= \underset{p_a \in \mathscr{P}(A)}{\operatorname{argmax}} \Big\{ A_x + A_\varphi - A_d \Big\} \qquad (17)
\end{aligned}$$

where we distribute the factors inside the logarithm in Equation 17 as follows:

$$\begin{aligned}
A_x &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log p_a(x|q_a^x) \\
&= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in L[q] \cap \Sigma} c(x, q_a) \mathbf{1}_{q = q_a^x} \log p_a(x|q) \qquad (18) \\
&= \sum_{q \in Q_a} \sum_{x \in L[q] \cap \Sigma} C(x, q) \log p_a(x|q).
\end{aligned}$$

Equation 18 follows from $q = q_a^x$ implying $q_a \in B(q)$.

$$\begin{aligned}
A_\varphi &= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \log \prod_{e \in \pi_\varphi(q_a, q_a^x)} p_a(\varphi|p[e]) \\
&= \sum_{q_a \in Q_a} \sum_{x \in L^*[q_a]} c(x, q_a) \sum_{e \in \pi_\varphi(q_a, q_a^x)} \log p_a(\varphi|p[e]) \\
&= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in L^*[q_a]} c(x, q_a) \sum_{e \in \pi_\varphi(q_a, q_a^x)} \mathbf{1}_{q = p[e]} \log p_a(\varphi|q) \\
&= \sum_{q \in Q_a} \sum_{q_a \in B(q)} \sum_{x \in \Sigma} c(x, q_a) \mathbf{1}_{x \notin L[q]} \log p_a(\varphi|q) \qquad (19) \\
&= \sum_{q \in Q_a} C(\varphi, q) \log p_a(\varphi|q)
\end{aligned}$$

Equation 19 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 17 proves the lemma. □

Figure 8: Proof of Lemma 3

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

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

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

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

Hence, μ_x is only non-zero if $y_x^{n+1}=\epsilon$. Furthermore, Since for all x, $\mu_x\leq 0$, for y_x^{n+1} to be positive, we need $\lambda \geq \max f(x,q,\mathbf{y}^n)$. Hence, the above two conditions can be re-expressed as (16). If $C(x,q)=0$, then we get

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

and the solution is given by $y_x^{n+1}=\epsilon$ and $\mu_x = f(x,q,\mathbf{y}^n)-\lambda$. Since μ_x can be negative, we have $f(x,q,\mathbf{y}^n)\leq\lambda$. Hence, irrespective of the value of $C(x,q)$, the solution is given by (16).
The above analysis restricts $\lambda \geq \max_x f(x,q,\mathbf{y}^n)$. If $\lambda < f(x,q,\mathbf{y}^n)+C(x,q)$, then $y_x^n > 1$ and if $\lambda > \max_x f(x,q,\mathbf{y}^n)+C(q)$, then $\sum_x y_x^n < 1$. Hence λ needs to lie in

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

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