UNIFIED TRAINING OF WFSA THROUGH A GENERIC INTERFACE

Mikel Penagarikano, German Bordel, Luis Javier Rodriguez

Department of Electricity and Electronics
University of the Basque Country, 48940 Leioa, Spain
E-mail: mikel.penagarikano@ehu.es

ABSTRACT

This paper describes a unified approach to the training of Weighted Finite-State Automata (WFSA) that is based on a generic interface. Regardless of their internal structure, any automata implementing a simple interface can be managed by the system, not only for decoding but also for training purposes. This novel approach drastically simplifies the effort to incorporate new formalisms into a pattern recognition engine. This methodology has been integrated into *Sautrela*, a highly modular and pluggable open source package for generic purpose signal processing, focused on speech recognition.[1]

***Index Terms*—** speech recognition, model integration, weighted finite-state automata

1. INTRODUCTION

Since the beginning of the speech recognition research, many different speech recognition systems have been developed [1, 9, 5]. All of them were mainly focused on the research of a few stages of the recognition process, and turned out to be quite hardwired implementations, since they were optimized for a particular methodology. So, an innovation in speech recognition technology meant the development of an entire new system from scratch. More recently, flexibility has played a major role in the design of new recognition engines, which have evolved into modular frameworks that incorporate state-of-the-art methodologies but also address the needs for emerging research areas. However, these systems are designed as decoder machines, whereas other matters such as training of acoustic, lexical and language models, are achieved by external packages.

Sautrela [8] is a highly modular and pluggable open source package for generic purpose signal processing applications that is focused on speech recognition. The system has been developed using the Java^{TM} Technology, thus ensuring its portability to a large variety of computer platforms. Unlike the previously cited systems, Sautrela defines a single framework that allows not only decoding the input streams, but also training all the required models.

[1]This work has been partially funded by the Basque Government, under program SAIOTEK , projects S-PE04UN26 and S-PE05UN32.

This paper focuses on solving the problem of training a set of models that implement a simple WFSA interface, with no assumption about their internal structure. This novel approach drastically simplifies the effort to incorporate new formalisms into a pattern recognition engine.

The rest of the paper is organized as follows. Section 2 defines the WFSA interface hierarchy used for handling the automata and Section 3 introduces the so called *Trainable* interface that generalizes the reestimation of model parameters. Section 4 reviews two well-known training criteria and describes their implementations. Section 5 shows the training interface implementation for three different model formalisms. Finally, Section 6 summarizes the main contributions of this work.

2. THE WFSA INTERFACE

2.1. The WFSA interface hierarchy

Sautrela uses a simple interface hierarchy to unify many different stochastic model techniques related to speech recognition, on top of which lies the WFSA interface (see Figure 1). The WFSA interface is extended by two subinterfaces: the Deterministic Weighted Finite-State Automata (DWFSA) and the Non-Deterministic Weighted Finite-State Automata (NdWFSA). Each subinterface consists of a single method that handles the deterministic and non-deterministic parts of the automaton. Basically, a WFSA consists of an initial state, an input alphabet and a weighted transition function that maps an input symbol and the current state to a next state (DWFSA) or a set of states (NdWFSA). Furthermore, *State*, *Symbol* and *Transition* are in fact relaxed interfaces that allow the system to deal with several types of objects, as for instance string input alphabets (language models) or continuous features (acoustic models). Regardless of its internal structure, any model that implements one of the mentioned subinterfaces can be managed by the system.

However, a speech recognition system is typically composed of different models standing for the corresponding knowledge sources. Hence, a single integrated stochastic formalism is needed. Layered Markov Models (LMM) [7] arise in response to this problem.

(a) The WFSA interface

```
// Returns the name of the WFSA.
String getName();

// Returns the so named Symbol
Symbol getSymbolByName(String name);

// Returns the initial State
State getIniState();

// Returns the probability of being final
double getFinProb(State state);

// Returns all possible transitions
Transition[] getTrans(State state);
```

(b) The DWFSA interface (extends WFSA)

```
// Returns the transition for the
// given state and symbol
Transition getTrans(State state, Symbol sy);
```

(c) The NdWFSA interface (extends WFSA)

```
// Returns all possible transitions for the
// given source state and symbol
Transition[] getTrans(State from, Symbol sy);
```

Fig. 1. The WFSA interface (a) consists of a minimum set of methods to handle an automaton. Each subinterface (b and c) adds a single method that accounts for the deterministic or non-deterministic nature of the weighted transition function.

2.2. Integration of knowledge sources using LMMs

A LMM consists of a number of layers, each composed of a finite set of WFSA. Each layer represents a knowledge source that models its units in terms of lower level layer units. Regardless of the number of layers and the types of WFSA involved, a LMM can be seen as a non-deterministic WFSA.

There are no limitations on the number of layers, models, states or symbols that set up a LMM. Therefore, integrating different knowledge sources into a single stochastic automaton turns out to be a highly flexible solution. State of the art speech recognition systems are made up of various models which are based on different formalisms, such as Hidden Markov Models (HMM), pronunciation dictionaries, graph-driven grammars or stochastic n-grams. All of them can easily implement the WFSA interface, and thus, hardwired model coupling found on most systems can be formalized as a single LMM. Similar approaches can be found in [1] and [6]. However, they are mainly focused on the decoding stage and do not provide generic tools for training purposes.

3. THE TRAINABLE INTERFACE

The WFSA interface, along with the DWFSA and NdWFSA subinterfaces, make up a minimum set of methods capable of addressing the decoding problem. Any training or reestimation process starts from a decoding-like stage that involves the computation of an objective function and then reestimates the internal parameters in order to increase the value of that function. In the same way, the WFSA interface parameters (transition weights and final probabilities) can be reestimated in order to maximize the target function. The recomputed parameters can be then transmitted to the underlying models, which will be responsible of transforming their internal representation to add that information.

```
// Initializes the training counts.
void initTrCounts();

// Increments the training counts associated
// to a Transition.
void incTrTransCount(Transition tr, double c);

// Increments the training counts associated
// to a final state.
void incTrFinalCount(State state, double c);

// Dumps the trained data to the model.
void dumpTrCounts();
```

Fig. 2. Using the *Trainable* interface, the counts can be externally obtained, whereas the model just deals with translating those counts to its internal representation.

The reestimated parameters are transmitted as incremental counts that must be later normalized (a count is nothing but the probability of an event at time t, given the input source sequence and the target objective function). The set of methods that conforms the *Trainable* interface allows to initialize, transmit and dump the counts to update the model parameters (see Figure 2).

4. TRAINING CRITERIA

Different training criteria (and the corresponding transformations) can be applied using this methodology. In the following paragraphs two well-known criteria are reviewed: Maximum Likelihood and Maximum Mutual Information.

4.1. Maximum Likelihood

Maximum Likelihood (ML) is the most widely used estimation criterion. Let $\mathbf{X}$ be a random variable wich is drawn according to the probability distribution function $p_\phi(x)$, where the parameter vector ϕ belongs to some parameter space Φ. Given a sequence of independent observations $\mathbf{x} = \{x_1, ..., x_n\}$, the likelihood function is given by:

$$L = p_\phi(\mathbf{x}) = \prod_{i=1}^{n} p_\phi(x_i)$$

The ML estimation of ϕ is obtained by maximizing L:

$$\phi_{\mathrm{ML}} = \arg\max_\phi p_\phi(\mathbf{x})$$

The Baum-Sell theorem for growth transformations [3] can be applied for such maximization, and reestimation formulae can be calculated for all parameters. In the case of WFSA, the parameters are restricted to transition and final probabilities, which must add up to one for each source state. Let $p_\phi(\tau)$ denote the probability of transition $\tau = (s, d, y)$ from source state s to destination state d and emitting the symbol y. The final probability can be seen as a special external transition with null emission, $p_\phi(s, out, null)$, and therefore can be integrated in the transition function. The reestimation of $p_\phi(\tau)$ is given by:

$$p_{\tilde{\phi}}(\tau) = \frac{p_\phi(\tau) \cdot \left(\frac{\partial L}{\partial P(\tau)}\right)_\phi}{\sum\limits_{\tau'=(s,d',y')} p_\phi(\tau') \cdot \left(\frac{\partial L}{\partial P(\tau)}\right)_\phi} \tag{1}$$

This expression can be rewritten in terms of the incremental counts computed for all the training samples. Let $count(\tau)_{t,i}$ be the probability of transition τ at time t for the observation x_i. These counts can be efficiently computed by using the so called *forward* and *backward* probabilities (see [2]). Finally, the expression (1) can be rewritten as follows:

$$p_{\tilde{\phi}}(\tau) = \frac{\sum\limits_{i=1}^{n}\sum\limits_{t=1}^{m_i} count(\tau)_{t,i}}{\sum\limits_{\tau'=(s,d',y')}\sum\limits_{i=1}^{n}\sum\limits_{t=1}^{m_i} count(\tau')_{t,i}} \tag{2}$$

4.2. Maximum Mutual Information

Any pattern recognition problem can be formalized using the noisy channel metaphor. Given a source random variable Ω and the observed data X, the mutual information $I(X;\Omega)$ accounts for the reduction in the uncertainty of X due to the knowledge of Ω. Assuming that the available sample vector $(\mathbf{x}, \omega) = \{(x_1, \omega_1), ..., (x_n, \omega_n)\}$ is representative, the mutual information can be approximated by:

$$I(X;\Omega) \approx \log \frac{\prod_{i=1}^{n} p_\phi(x_i \mid \omega_i)}{\prod_{i=1}^{n} p_\phi(x_i)} = \log \frac{\prod_{i=1}^{n} p_\phi^{sup}(x_i)}{\prod_{i=1}^{n} p_\phi^{unsup}(x_i)}$$

Note that the numerator $p_\phi^{sup}(x_i)$ is the expectation of the observation x_j when the source class ω_j (labelling) is known. The denominator $p_\phi^{unsup}(x_i)$ accounts for the expected probability of x_i in the absence of supervision. Both probabilities can be computed in a straightforward way by building a LMM with an additional knowledge layer that accounts for such supervision or unsupervision. The Maximun Mutual Information (MMI) estimation of ϕ is obtained by maximizing $I(X;\Omega)$:

$$\phi_{\mathrm{MMI}} = \arg\max_\phi \frac{\prod_{i=1}^{n} p_\phi^{sup}(x_i)}{\prod_{i=1}^{n} p_\phi^{unsup}(x_i)}$$

The Gopalakrishnan theorem for rational functions [4] proves that there exists a constant C, such that the following expression is a growth transformation:

$$p_{\tilde{\phi}}(\tau) = \frac{p_\phi(\tau) \cdot \left(\left(\frac{\partial I}{\partial P(\tau)}\right)_\phi + C\right)}{\sum\limits_{\tau'=(s,d',y')} p_\phi(\tau') \cdot \left(\left(\frac{\partial I}{\partial P(\tau)}\right)_\phi + C\right)} \tag{3}$$

As for the ML estimation, forward and backward probabilities can be used to compute the counts. In this case, there will be two partial counts, $count(\tau)_{t,i}^{sup}$ and $count(\tau)_{t,i}^{unsup}$, and the resulting count will be the difference between them:

$$count(\tau)_{t,i} = count(\tau)_{t,i}^{sup} - count(\tau)_{t,i}^{unsup}$$

The aforementioned constant C will ensure the convergence even in the presence of negative counts[2]. Finally, the expression (3) can be rewritten as follows:

$$p_{\tilde{\phi}}(\tau) = \frac{C \cdot p_\phi(\tau) + \sum\limits_{i=1}^{n}\sum\limits_{t=1}^{m_i} count(\tau)_{t,i}}{\sum\limits_{\tau'=(s,d',y')} C \cdot p_\phi(\tau') + \sum\limits_{i=1}^{n}\sum\limits_{t=1}^{m_i} count(\tau')_{t,i}} \tag{4}$$

5. INTERFACE IMPLEMENTATION EXAMPLES

5.1. Layered Markov Models

The core of the training approach presented in this paper lies in the possibility of training a single LMM composed of several layers, corresponding to different, hierarchically organized, knowledge sources. As shown in Section 2, a LMM can be seen as a non-deterministic WFSA. Therefore, its parameters can be iteratively reestimated by computing the counts for all the training samples according to a given criterion (for instance, ML or MMI), and transmitting them to the LMM by means of the *Trainable* interface. Now, the issue arises of how the training information is transmitted to all the inner models the LMM is made up.

The answer turns out to be quite simple. A LMM transition τ consists of a sequence of transitions $\Delta_\tau = \{\delta_1^\tau, ..., \delta_k^\tau\}$ within internal layers. So, $count(\tau)_{t,i}$ is transmitted to all the transitions τ is made up. In other words, the count for the inner transition δ, $count(\delta)_{t,i}$, is computed by adding the counts of the LMM transitions τ such that $\delta \in \Delta_\tau$. Let $T(\delta) = \{\tau \mid \delta \in \Delta_\tau\}$. Then,

$$count(\delta)_{t,i} = \sum_{\tau \in T(\delta)} count(\tau)_{t,i}$$

[2]Although the expression (3) is proven to be a growth transformation for sufficiently large values of C, experiments show that small values of C provide a faster convergence.

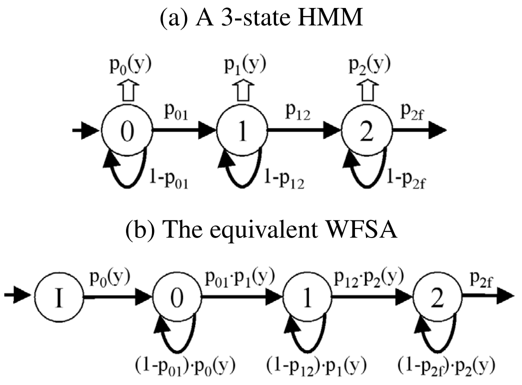

Fig. 3. Each transition in the HMM generates as many transitions in the equivalent WFSA as emissions in the destination HMM state. The extra initial state in the WFSA contains transitions to the equivalent initial states in the HMM.

5.2. Discrete Hidden Markov Models

In HMMs, emissions are tied to states, whereas WFSA emissions happen at transitions. An equivalent WFSA can be obtained by assuming that emissions happen just before arriving to HMM states (see Figure 3).

To reestimate HMM parameters, each WFSA count must be translated into internal HMM counts (initial, transition and emission counts for each HMM state). Let $T_A(s,d) = \bigcup_{\forall y} \{\tau = (s,d,y)\}$ be the set of WFSA transitions with source state s and destination state d, $T_B(d,y) = \bigcup_{\forall s} \{\tau = (s,d,y)\}$ the set of WFSA transitions with destination state d and emission symbol y, and s_I the initial state of the WFSA. Then, the HMM internal counts are given by:

$$count_{init}(s)_{t,i} = \sum_{\tau \in T_A(s_I,s)} count(\tau)_{t,i}$$

$$count_{trans}(s,d)_{t,i} = \sum_{\tau \in T_A(s,d)} count(\tau)_{t,i}$$

$$count_{emit}(s,y)_{t,i} = \sum_{\tau \in T_B(s,y)} count(\tau)_{t,i}$$

That is, each WFSA count is transmitted to either initial or transition HMM counts (depending on the source state), and always to the emission counts of the destination HMM state. In the case of ML training, these HMM counts match exactly those obtained with the Baum-Welch algorithm [2].

5.3. Continuous Hidden Markov Models

The WFSA equivalent to a continuous HMM is analogous to that presented for discrete HMMs, but defining a continuous set of transitions for each state. This is not a problem for the WFSA formalism, since it does no assumption on the nature of the observed data and can handle infinite transition sets.

6. CONCLUSIONS

In this paper, a novel approach to the training of WFSA has been introduced. Any model that implements a simple WFSA interface can be trained regardless of its internal structure. In the case of ML training of Hidden Markov Models, the reestimations match exactly those obtained with the Baum-Welch algorithm. Layered Markov Models have proved to be a good formalism for the integration of knowledge sources in a single WFSA and the transmission of the training information to its inner models. Based on this approach, a general training module has been developed for the *Sautrela* system, drastically simplifying the effort to incorporate new modelling formalisms into this framework.

7. REFERENCES

[1] J.K. Baker. The DRAGON system - An Overview. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 23(1):24–29, 1975.

[2] Leonard. E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. *Inequalities*, 3:1–8, 1972.

[3] Leonard. E. Baum and George R. Sell. Growth transformations for functions on manifolds. *Pacific Journal of Mathematics*, 27:211–227, 1968.

[4] P. S. Gopalakrishnan, D. Kanevsky, A. Nádas, and D. Nahamoo. An inequality for rational functions with applications to some statistical estimation problems. *IEEE Trans. Information Theory*, 37:107–113, 1991.

[5] K. Lee, H. Hon, and R Reddy. An overview of the SPHINX speech recognition system. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 38(1):35 – 45, January 1990.

[6] M. Mohri, F. Pereira, and M. Riley. Weighted finite-state transducers in speech recognition. *Computer Speech & Language*, 16(1):69–88, 2002.

[7] M. Penagarikano and G. Bordel. Layered Markov Models: a new architectural approach to automatic speech recognition. In *Proceedings of the MLSP Workshop*, 2004.

[8] M. Penagarikano and G. Bordel. Sautrela: A highly modular open source speech recognition framework. In *Proceedings of the ASRU Workshop*, 2005.

[9] S. Young. The HTK Hidden Markov Model Toolkit: Design and Philosophy. Technical Report TR.153, Department of Engineering, Cambridge University, 1993.