ON-LINE ALGORITHMS FOR COMBINING LANGUAGE MODELS

Adam Kalai, Stanley Chen, Avrim Blum, Ronald Rosenfeld

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
{akalai,sfc,avrim,roni}@cs.cmu.edu

ABSTRACT

Multiple language models are combined for many tasks in language modeling, such as domain and topic adaptation. In this work, we compare *on-line algorithms* from machine learning to existing algorithms for combining language models. On-line algorithms developed for this problem have parameters that are updated dynamically to adapt to a data set during evaluation. *On-line analysis* provides guarantees that these algorithms will perform nearly as well as the best model chosen in hindsight from a large class of models, *e.g.*, the set of all static mixtures. We describe several on-line algorithms and present results comparing these techniques with existing language modeling combination methods on the task of domain adaptation. We demonstrate that in some situations, on-line techniques can significantly outperform static mixtures (by over 10% in terms of perplexity), and are especially effective when the nature of the test data is unknown or changes over time.

1. INTRODUCTION

Multiple language models are combined for many aspects of language modeling, including domain adaptation, topic adaptation, and the application of class n-gram models [12, 10, 3]. In this work, we compare *on-line algorithms* from machine learning to existing algorithms for combining language models.

Consider the situation where we are combining m language models $p_j(\text{word } w|\text{history } h)$ (which we refer to as *submodels*) into a composite model $p_{\text{super}}(w|h)$ (which we refer to as a supermodel) for an evaluation test set $T = w_1 \cdots w_t$. Supermodels often have the form

$$p_{\text{super}}(w_i|h_i) = \sum_{j=1}^{m} \lambda_j^i p_j(w_i|h_i = w_1 \cdots w_{i-1}) \qquad (1)$$

where $\lambda_j^i \geq 0$ is the weight of the j^{th} model on the i^{th} word and where $\sum_{j=1}^{m} \lambda_j^i = 1$ for all i. The most common method for combining language models is a *static mixture*, where the weights λ_j^i of each submodel p_j are taken to be fixed over all words in the test set. In on-line algorithms, the weights λ_i^j are chosen to dynamically adapt to the characteristics of the test data.

In the algorithms we present here, the weight updates are chosen so that *on-line analysis* provides guarantees that these algorithms will perform nearly as well as the best model chosen in hindsight from some class of models. Our first algorithm, SELECTOR, adapts on-line with the guarantee that its performance is virtually identical to that of the best of the component submodels. The MIXER algorithm adapts on-line with the guarantee that its performance will be nearly as good as the optimal static mixture, chosen in hindsight. Finally, the SWITCHER algorithm is compared to the class of language models that may switch between submodels from time to time. These three algorithms are closely related to certain types of Hidden Markov Models, but involve extensions that are not easily expressible within the Hidden Markov Model framework.

We performed experiments to compare these adaptive techniques to more traditional static methods on the task of *domain adaptation*, using data from four different domains: Wall Street Journal text, Associated Press text, Broadcast News transcriptions, and Switchboard transcriptions. We show that on-line algorithms generally perform at least as well as static mixtures in terms of perplexity, and in some cases considerably better. Furthermore, we show these algorithms are extremely robust, due of their ability to dynamically adapt to test data.

We would like to thank Doug Beeferman and John Lafferty for many helpful discussions. This work was supported in part by a NSF Graduate Fellowship.

2. ON-LINE ALGORITHMS

We begin by stating the on-line problem of *predicting from expert advice*, as we see it applied to the field of language modeling. A learning algorithm, in this case a supermodel, is repeatedly given the task of predicting the next word given the previous words. In addition, the algorithm is given as input the advice of m "experts," in this case language models. After each word, each expert predicts the next word with a probability distribution, and then the supermodel must combine these predictions in order to make its own prediction. Suppose we make no assumptions about the quality or independence of the expert language models, so we cannot hope to guarantee any absolute level of quality in our predictions. In that case, a natural first goal is to perform nearly as well as the best language model so far: that is, to guarantee that at any time, our algorithm has not performed much worse than whichever expert has been the most accurate to date.

The performance of a language model $p(w|h)$ on some test data $T = w_1 \ldots w_t$ is measured as the probability it assigns to the text, $p(T) = \prod p(w_i|h_i)$. The cross-entropy $H_p(T)$ of a model $p(w|h)$ on data T is defined as

$$H_p(T) = -\frac{1}{t}\log_2 p(T) = -\frac{1}{t}\sum_{i=1}^{t}\log_2 p(w_i|h_i)$$

and can be interpreted as the average number of bits needed to encode each of the t words in the test data using the compression algorithm associated with model $p(w|h)$. The perplexity of a text T, $PP_p(T)$, which we use to report our results, is defined as follows,

$$PP_p(T) = 2^{H_p(T)} = \frac{1}{p(T)}$$

.

This first goal is addressed by what we call the SELECTOR algorithm, which has been analyzed in several fields [6]. We view it as a special case of the problem of predicting from expert advice, described further in [1].

2.1. Selector

The SELECTOR is a supermodel which does almost as well as the single best of its constituent language models, regardless of the text. One way to describe a language model is to give the probability assigned to any text T, $p(T)$. In this way we define the SELECTOR $(\{p_1, p_2, \ldots, p_m\})$ by,

$$p_{\text{sel}}(T) = \frac{1}{m} \sum p_j(T). \tag{2}$$

As an average of m probability distributions, p_{sel} is clearly a valid distribution, and it also satisfies, for all j,

$$p_{\text{sel}}(T) \geq \frac{p_j(T)}{m} \Longrightarrow H_{p_{\text{sel}}}(T) \leq H_{p_j}(T) + \frac{\log_2 m}{t}.$$

In particular, the cross-entropy of SELECTOR is at most $log(m)/t$ bits more than the cross-entropy of the best model, regardless of the domain. We use the term *cross-entropy overhead* for the difference between the cross entropy of the supermodel and the cross entropy of the best of the competing models. In this case the overhead is very small, even for millions of models on any reasonably sized text.

While technically we have described the language model, we have not given an efficient algorithm for calculating $p(w|h)$. One implementation is a simple m-state Hidden Markov Model with zero probability of changing state. Each state refers j to a different model and predicts the next word based on $p_j(w|h)$. The initial distribution is uniform over the states. Thus, we can view λ_j^i in equation (1) as the probability of being in state j after the i^{th} word. The update rule becomes

$$\lambda_j^{i+1} = \frac{\lambda_j^i p_j(w_i|h_i)}{\sum_{j'=1}^m \lambda_{j'}^i p_{j'}(w_i|h_i)},$$

and the algorithm runs in linear time in the length of the text.

2.2. Mixer

Often, combining language models yields better results than any of the individual language models themselves. The most common combination method, described in the introduction, is

$$p_{\vec{\lambda}}(w|h) = \sum \lambda_j p_j(w|h),$$

with $\lambda \in \Delta, i.e. \sum_{j=1}^m \lambda_j = 1$ and $\lambda_j \geq 0$. Thus, a second goal is to perform almost as well as the best static mixture of language models, where the best is chosen in hindsight. We get just such a guarantee by using an on-line algorithm for investing in a stock market due to Cover [4, 5]. Cover's algorithm can be written simply,

$$\text{MIXER} = \text{SELECTOR}\left(\{p_{\vec{\lambda}} | \vec{\lambda} \in \Delta\}\right).$$

This combination of infinitely many language models is a description of a probability distribution rather than a description of an algorithm for computing $p_{\text{mix}}(w|h)$. An $O(t^{m-1})$ implementation is described in [4], but there are faster approximations such as tiling [6] or sampling [2].

Cover's algorithm, translated to the language modeling domain, comes with the following guarantee for all j,

$$H_{p_{\text{mix}}}(T) \leq H_{p_j}(T) + (m-1)\frac{\log_2 t}{t}.$$

Guarantees of the performance of Cover's algorithm and the two approximations can be found in [4, 5], [2], and [6], respectively. The above cross-entropy overhead of $(m-1)\frac{\log_2 t}{t}$ is small for large test sets. For example, over 10,000 words and 10 language models, the maximum overhead is about 0.01 bits per word. However, this grows linearly in the number of models instead of logarithmically as with SELECTOR, so we cannot guarantee performance with large numbers of models, at which point the computational costs become prohibitive as well.

2.3. Switcher

In this section, we describe a novel on-line algorithm, SWITCHER. SWITCHER has the property that it does almost as well as the best submodel on any segment of the text. Put another way, SWITCHER is designed to compete with supermodels that predict each word according to a single submodel but which are allowed to switch between submodels from time to time. Since it may be possible to get absurdly high performance by switching very frequently, the guaranteed bounds, which are asymptotically optimal, depend on the switching frequency.

First, we define a fixed switching algorithm $p_{\vec{j}}(w|h)$ based on the fixed sequence of choices it makes, $\vec{j} = (j_1, \ldots, j_t)$, for which model to use to predict the next word. In other words,

$$p_{\vec{j}}(w_i|h_i) = p_{j_i}(w_i|h_i).$$

Next, as in [11, 8], we define SWITCHER$_\gamma$ specifically to be competitive against algorithms which switch between submodels with frequency γ. Like the SELECTOR, this can be described by an m-state Hidden Markov Model with a uniform starting distribution. Just as before, state j represents a language model and predicts the word according to $p_j(w|h)$. However, now there is now a fixed probability, $1-\gamma$, of staying in a state and $\gamma/(m-1)$ of going from a state to a different state. We can use the same description as we did for SELECTOR in equation (1), where λ_j^i is the probability of being in state j after the i^{th} word. We have the new update,

$$\lambda_j^{i+1} = \left(1 - \gamma\left(\frac{m}{m-1}\right)\right) \frac{\lambda_j^i p_j(w_i|h_j)}{\sum_{j'} \lambda_{j'}^i p_{j'}(w_i|h_i)} + \frac{\gamma}{m-1}.$$

It is not hard to bound the performance of SWITCHER$_\gamma$ relative to $p_{\vec{j}}(w|h)$. We say $p_{\vec{j}}$ has a switching frequency f when,

$$f = \frac{|\{i : j_i \neq j_{i+1}\}|}{t-1}.$$

Singer shows in equation (2) of [11],

$$\frac{p_{\mathrm{swi}_\gamma}(T)}{p_{\vec{j}}(T)} \geq \frac{1}{m}\left[\left(\frac{\gamma}{m-1}\right)^f (1-\gamma)^{1-f}\right]^{t-1} \tag{3}$$

The general SWITCHER, which has no parameter γ, combines SWITCHER$_\gamma$'s so as to adapt to the parameter γ. On a sequence of length t, for any $\vec{j}$, the number of times $p_{\vec{j}}(w|h)$ switches models is an integer between 0 and $t-1$. Thus, we define SWITCHER to simply select among these values:

$$\text{SWITCHER} = \text{SELECTOR}\left(\left\{\text{SWITCHER}_{\frac{s}{t-1}} | s \in \{0,\ldots,t-1\}\right\}\right)$$

By combining equations (2) and (3), for any $\vec{j}$ with switching frequency f, we get a maximum cross-entropy overhead of

$$\frac{\log_2 mt}{t} + f\log_2(m-1) - f\log_2 f - (1-f)\log_2(1-f)$$

which, because $-(1-f)\log_2(1-f) \leq f\log_2 e$, is no more than,

$$\frac{\log_2 mt}{t} + f\log_2\frac{(m-1)e}{f}$$

3. EXPERIMENTS

To compare on-line algorithms with existing techniques for combining language models, we performed experiments on the task of domain adaptation. In domain adaptation, one attempts to improve a language model for one domain (*e.g.*, Switchboard) using training data from additional domains (*e.g.*, North American Business news). Typically, the target domain is known ahead of time; however, to highlight the potential advantages of on-line algorithms, we also consider the situation where the domain of the test data is hidden and where the test data contains text from multiple domains. Previous work in domain adaptation [12, 9] has shown that static mixtures achieve competitive performance in terms of both perplexity and speech recognition word-error rate with other existing combination methods.

We constructed four trigram language models (with a variation of Kneser-Ney smoothing) using training data from the following four sources: Wall Street Journal text (5M words), Associated Press text (5M words), Broadcast News transcriptions (5M words), and Switchboard transcriptions (3M words). For each domain, we extracted a held-out set and test set of about 25,000 words.

In our first set of experiments, we evaluated several methods on each domain test set separately. We calculated the perplexity of each test set using only the language model from the matching domain; using a trigram model constructed from the training data from each domain merged together; using a static mixture of the four domain language models with weights optimizing the perplexity of the matching held-out set; and using a static mixture with weights optimizing the perplexity of the given test set, *i.e.*, the optimal static mixture chosen in hindsight. These perplexities are displayed in order at the top of Table 1.

We ran the SELECTOR, MIXER, and SWITCHER algorithms on these data sets. As can be seen in Table 1, SELECTOR did exactly as theoretically guaranteed, matching the performance of the best single submodel for each test set, for an average perplexity of 174.7. The MIXER performed significantly better, with an average

	domain				
algorithm	AP	WSJ	SWB	BN	avg.
corr. domain only	266.2	201.9	83.2	209.8	174.7
merge train. data	238.9	195.1	103.1	193.6	174.2
mixt., h.-o. wgts.	234.0	184.5	79.4	182.9	157.9
best mixtures	231.8	184.2	79.4	180.0	156.7
SELECTOR	266.2	201.9	83.2	209.8	174.7
SWITCHER	234.5	187.0	82.1	182.2	159.6
MIXER	233.0	184.5	80.2	180.4	157.4

Table 1: Perplexities of various models on test set from each of the four domains; the *average* column represents average performance over each domain test set weighted by test set length

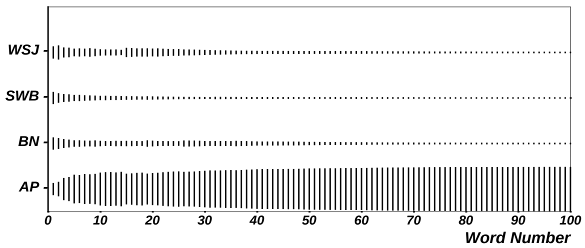

Figure 1: The weights λ_j^i of MIXER over the first 100 words of the AP test set

perplexity of 157.4, close to the performance of the optimal static mixture. In Figure 1, we display the weights λ_j^i that MIXER places on the different models as we predict the first 100 words of the AP data. As expected, these weights quickly devote the majority of their mass to the AP model.

Surprisingly, the SWITCHER almost matched the performance of the best static mixture, even though it employs only a single submodel at a time. On closer examination, we see that these models are switching more often than one might expect. Recall that a SWITCHER is simply an average of SWITCHER$_\gamma$'s for different values of $\gamma \in [0,1]$, where γ corresponds to the natural switching frequency. On the four test sets, the highest weighted SWITCHER$_\gamma$'s switched with probabilities 0.13, 0.05, 0.02, and 0.27. In Figure 2, we show the weights λ_j^i of the SWITCHER algo-

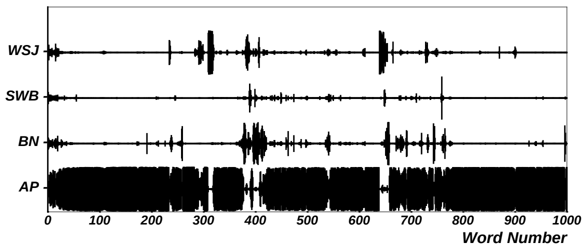

Figure 2: The weights λ_j^i of SWITCHER over the first 1000 words of the AP test set

algorithm	perplexity
mixture, held-out wgts.	185.1
best mixture	184.7
MIXER	184.7
SWITCHER	160.8

Table 2: Perplexities of various models on test set combining four domain test sets

rithm over the first 1,000 words of the AP test set.

In our second set of experiments, we concatenated the four domain test sets into a single long test set. We calculated the perplexity of this test set using a static mixture with weights optimizing the perplexity of the four held-out sets concatenated; using the optimal static mixture in hindsight; and using the MIXER and SWITCHER supermodels. These perplexities are displayed in Table 2. Not surprisingly, the performance of the MIXER matches that of the best static mixture (184.7 PP). However, the SWITCHER performed almost as well on the concatenation (160.8 PP) as it did on the individual test sets (159.6 PP). As expected, this test set highlights the strength of the SWITCHER, resulting in a 13% decrease in perplexity over the other models.

4. DISCUSSION

On-line algorithms provide a simple and effective set of tools for combining language models. These methods are extremely general as no assumptions are made about the structure of component language models. We demonstrate that on-line algorithms perform comparably with static mixtures, which have achieved excellent performance on many language modeling tasks. In the case where our test set was composed of text from disparate domains, the algorithm SWITCHER resulted in a perplexity decrease of over 10% as compared to the optimal static mixture.

In general, on-line algorithms automatically and dynamically adapt to changing text characteristics. Consequently, on-line algorithms do not require held-out sets for parameter optimization and still perform comparably to or better than static mixtures with weights trained on held-out sets. This is especially advantageous when no appropriate held-out is available. For example, when static mixture weights are optimized on just the Broadcast News held-out set for the experiments in Table 1, the average test perplexity rises from 157.9 to 186.6. Conversely, even though on-line algorithms were blind to the domain of the test set in our experiments, they still performed just as well as other algorithms that took advantage of knowledge of the domain of the test set.

While the guaranteed performance bounds of on-line algorithms are often excellent, in some cases they may not be adequate. For large numbers of component models or very short test sets, guarantees are generally poor. For example, combining 5,000 language models could lead to a sixfold perplexity increase for the MIXER algorithm over that of the optimal static mixture on a data set of 25,000 words. Furthermore, the computational efficiency of the MIXER algorithm is poor for large numbers of submodels; it can require time exponential in the number of submodels. However, there exist other on-line algorithms that do not have theoretical performance guarantees but which still perform well in practice and which are very efficient, *e.g.*, [7].

The on-line algorithms we describe are closely related to Hidden Markov Models. While the more complex algorithms involve theory that falls outside of the conventional HMM framework, the simpler algorithms have direct HMM analogs, and our implementation of each algorithm can be considered to be an HMM. However, on-line analysis provides a different perspective for analyzing and designing these types of models.

In some applications, the characteristics of the test data are known and appropriate held-out data is available and consequently there is little advantage in using on-line algorithms over static mixtures. However, when the nature of the test data is unknown or changing over time, on-line algorithms offer the advantages of performance, convenience, generality, and robustness.

5. REFERENCES

[1] A. Blum. On-line algorithms in machine learning. In *Proceedings of the Workshop on On-Line Algorithms*, Dagstuhl, 1996.

[2] A. Blum and A. Kalai. Universal portfolios with and without transaction costs. In *Proceedings of the 10th Annual Conference on Computational Learning Theory*, 1997.

[3] P. F. Brown, V. J. Della Pietra, P. V. deSouza, J. C. Lai, and R. L. Mercer. Class-based n-gram models of natural language. *Computational Linguistics*, 18(4):467–479, December 1992.

[4] T. Cover. Universal portfolios. *Math. Finance*, 1(1):1–29, January 1991.

[5] T. Cover. Universal data compression and portfolio selection. In *Proceedings of the 37th IEEE Symposium on Foundations of Computer Science*, pages 534–538, October 1996.

[6] D. Foster and R. Vohra. Regret in the on-line decision problem. In *Something for Nothing Workshop*, May 1995.

[7] D. Helmbold, R. Schapire, Y. Singer, and M. Warmuth. On-line portfolio selection using multiplicative updates. In *Machine Learning: Proceedings of the 13th International Conference*, 1996.

[8] M. Herbster and M. Warmuth. Tracking the best expert. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 286–294, 1995.

[9] R. Iyer, M. Ostendorf, and H. Gish. Using out-of-domain data to improve in-domain language models. *IEEE Signal Processing Letters*, 4(8):221–223, August 1997.

[10] K. Seymore and R. Rosenfeld. Using story topics for language model adaptation. In *Proceedings of Eurospeech '97*, 1997.

[11] Y. Singer. Switching portfolios. In *Proceedings of the 14th Conference on Uncertainty in Artificial Intelligence (UAI-98)*, 1998.

[12] M. Weintraub, Y. Aksu, S. Dharanipragada, S. Khudanpur, H. Ney, J. Prange, A. Stolcke, F. Jelinek, and L. Shriberg. Fast training and portability. In *1995 Language Modeling Summer Research Workshop: Technical Reports*, Center for Language and Speech Processing, Johns Hopkins University, Baltimore, 1995.