In Proc. International Joint Conf. on Neural Networks (IJCNN 2005). Funded by Swiss NSF grant 200021-100005/1.

Deriving Kernels from MLP Probability Estimators for Large Categorization Problems

Ivan Titov
Department of Computer Science
University of Geneva
24, rue Général Dufour
CH-1211 Genève 4, Switzerland
E-mail: ivan.titov@cui.unige.ch

James Henderson
School of Informatics
University of Edinburgh
2 Buccleuch Place
Edinburgh EH8 9LW, United Kingdom
E-mail: james.henderson@ed.ac.uk

***Abstract*— In multi-class categorization problems with a very large or unbounded number of classes, it is often not computationally feasible to train and/or test a kernel-based classifier. One solution is to use a fast computation to pre-select a subset of the classes for reranking with a kernel method, but even then tractability can be a problem. We investigate using trained multi-layer perceptron probability estimators to derive appropriate kernels for such problems. We propose a kernel derivation method which is specifically designed for reranking problems, and a more efficient variant of this method which is specifically designed for neural networks with large numbers of output units. When applied to a neural network model of natural language parsing, these new methods achieve state-of-the-art performance which improves over the original model.**

I. Introduction

There is increasing interest in applying kernel methods to large complex problems, but these efforts are often compromised by the computational cost of these methods. This is particularly true when the space of possible output categories is unbounded, or even just very large, as in structural classification problems. One common solution is to use a fast computation to select a subset of the categories, and then use a kernel-based classifier to discriminate the best class from amongst this subset (called reranking) [1], [2], [3], [4]. But this approach does not completely address the computational costs of kernel methods for complex domains, and kernel designers must be careful to balance the accuracy of their proposals against their computational tractability.

In most complex domains, previous work on probability models have already developed proposals which achieve good accuracy while staying tractable. We can potentially improve on these models, while still exploiting the domain knowledge they embody, by applying methods which derive kernels from probabilistic models [5], [6]. Such methods not only exploit the domain-specific design of the probabilistic model, they also exploit the information extracted from the data during training of the probabilistic model.

In this paper, we first propose a new method for deriving kernels from probabilistic models which is specifically designed for reranking problems, and then investigate several ways of applying this and previous kernel derivation methods to multi-layer perceptron probability estimators. We test these approaches on a neural network for estimating the probabilities of natural language parses [7]. This approach achieves state-of-the-art performance on the parsing task, and the kernels designed for reranking show a significant improvement over the original model. This includes one kernel which greatly improves computational efficiency by selecting kernel features based on the subset of categories to be reranked.

II. Kernels Derived from Probabilistic Models

In recent years, several methods have been proposed for constructing kernels from trained probabilistic models. As usual, these kernels are then used with linear classifiers to learn the desired task. As well as some empirical successes, these methods are motivated by theoretical results which suggest we should expect some improvement with these classifiers over the classifier which chooses the most probable answer according to the probabilistic model (i.e. the maximum a posteriori (MAP) classifier). There is guaranteed to be a linear classifier for the derived kernel which performs at least as well as the MAP classifier for the probabilistic model. So, assuming the classifier learning is sufficiently good, we should generally expect the derived kernel's classifier to perform at least as well as the probabilistic model's classifier.

In this section, we first present two previous methods for deriving kernels and then propose a new method specifically for reranking tasks. Before discussing the different methods for deriving kernels from probabilistic models, we need to introduce the class of discriminant functions, defined in terms of feature extractors from input-output pairs. We consider the problem of learning a mapping function from an input space $\mathcal{X}$ to a structured output space $\mathcal{Y}$. For example, in parsing, $\mathcal{X}$ is the space of all possible sentences and $\mathcal{Y}$ is the space of parse trees. On the basis of training data, we learn a discriminant function $F : \mathcal{X} \times \mathcal{Y} \rightarrow \mathcal{R}$. For an input x, the model outputs the category y with the largest value of the discriminant function $F(x, y)$. We focus on linear discriminant functions:

$$F_w(x, y) = w^T \phi(x, y), \tag{1}$$

where $\phi(x, y)$ is a feature vector for the input-output pair x, y, and w is the parameter vector for the discriminant function. In

the remainder of this section, we will characterize the kernel methods we consider in terms of the feature extractor $\phi(x, y)$.

A. Fisher Kernels

The Fisher kernel [5] is one of the best known methods for deriving kernels from probability models. Given a generative model of $P(z|\hat{\theta})$ with smooth parameterization, the Fisher score of an example z is a vector of partial derivatives of the log-likelihood of the example with respect to the model parameters:

$$\phi_{\hat{\theta}}(z) = (\frac{\partial \log P(z|\hat{\theta})}{\partial \theta_1}, \ldots, \frac{\partial \log P(z|\hat{\theta})}{\partial \theta_l}). \quad (2)$$

This score can be regarded as specifying how the model should be changed in order to maximize the likelihood of the example z. Then we can define the similarity between data points as the inner product of the corresponding Fisher scores. This kernel is often referred to as the practical Fisher kernel. The theoretical Fisher kernel depends on the inverse of the Fisher information matrix, which is not feasible to compute for most practical tasks and is usually omitted.

The Fisher kernel is only directly applicable to binary classification tasks. We can apply it to our task by considering an example z to be an input-output pair (x, y), and classifying the pairs into correct ones versus incorrect ones. When we use the Fisher score $\phi_{\hat{\theta}}(x, y)$ in the discriminant function F, we can interpret the value as the confidence that the output y is correct, and choose the y in which we are the most confident.

B. TOP Kernels

Tsuda et al. [6] proposed another kernel constructed from a probabilistic model, called the Tangent vectors Of Posterior log-odds (TOP) kernel. Their TOP kernel is also motivated only for binary classification tasks, so, as above, we treat the input z as a input-output pair and the output category $c \in \{-1, +1\}$ as incorrect/correct. It is assumed that the true probability distribution is included in the class of probabilistic models and that the true parameter vector $\theta^\star$ is unique. The feature extractor of the TOP kernel for the input z is defined by:

$$\phi_{\hat{\theta}}(z) = (v(z, \hat{\theta}), \frac{\partial v(z,\hat{\theta})}{\partial \theta_1}, \ldots, \frac{\partial v(z,\hat{\theta})}{\partial \theta_l}), \quad (3)$$

where

$$v(z, \hat{\theta}) = \log P(c = +1|z, \hat{\theta}) - \log(1 - P(c = +1|z, \hat{\theta})). \quad (4)$$

In addition to being at least as good as the MAP classifier, the choice of the TOP kernel feature extractor is motivated by the minimization of the classification error of a linear classifier $w^T \phi_{\hat{\theta}}(z) + b$.

The motivation for the TOP kernel isn't quite appropriate for structural classification problems because of the different definition of the classifier error. Also the computation of the probabilities $P(c = +1|z, \hat{\theta}) = P(y|x, \hat{\theta})$ on the basis of a generative probability model isn't straightforward for structural classification problems, where the number of possible outputs y is usually very large:

$$P(y|x, \hat{\theta}) = \frac{P(x, y|\hat{\theta})}{\sum_{y'} P(x, y'|\hat{\theta})}. \quad (5)$$

Here the sum in the denominator is taken over all the possible outputs y' for the input x. In section III-A we describe a method which can be used with a neural network generative probability estimator to approximate the normalization factor and its derivatives.

C. A TOP Kernel for Reranking

We propose a new TOP kernel which is motivated by the minimization of error in reranking problems. We define the reranking task as choosing an output from the list of candidates selected by some model. Furthermore, we only consider learning to rerank the output of a particular candidate selection model, without requiring the classifier to have good performance when applied to a candidate list provided by a different model. To construct the kernel we apply an approach similar to that used in [6] but adapt it to this reranking task.

Again we make the assumption that the true probability distribution is in the class of probabilistic models, with the parameter vector $\theta^\star$. The classification error of an optimal linear classifier in the feature space is defined as

$$R(\phi_{\hat{\theta}}) = \min_{w \in \mathcal{R}^{l+1}} E_{x,y} \Phi(\min_{t=1,s} \{w^T(\phi_{\hat{\theta}}(x, y) - \phi_{\hat{\theta}}(x, y_t))\}), \quad (6)$$

where the expectation is taken over all the possible inputs x and the outputs y which appear in the candidate list, $\Phi(a)$ denotes the step function which equals to 1 if the argument is negative and to 0 otherwise. To obtain a bound for $R(\phi_{\hat{\theta}})$, we define the error of the posterior probability estimator of the form $g(w^T \phi_{\hat{\theta}}(x, y))$ where g is the logistic sigmoid function:

$$D(\phi_{\hat{\theta}}) = \min_{w \in \mathcal{R}^{l+1}} E_x \max_{k=1,s} |g(w^T \phi_{\hat{\theta}}(x, y_k)) - P(y_k|x, y_1 \ldots y_s, \theta^\star)|, \quad (7)$$

here $P(y_k|x, y_1 \ldots y_s, \theta^\star)$ is the probability that y_k is the output for the input x when selecting from the list of candidates $y_1 \ldots y_s$. Let us denote the candidate selected by the linear classifier as $\hat{y}$ and the most probable candidate as $y^\star$, the following result is straightforward to obtain:

$$P(y^\star|x, y_1 \ldots y_s, \theta^\star) - P(\hat{y}|x, y_1 \ldots y_s, \theta^\star) \leq 2 \max_{k=1,s} |g(w^T \phi_{\hat{\theta}}(x, y_k)) - P(y_k|x, y_1 \ldots y_s, \theta^\star)|, \quad (8)$$

which immediately gives us

$$R(\phi_{\hat{\theta}}) - L^\star \leq 2D(\phi_{\hat{\theta}}), \quad (9)$$

where $L^\star$ is the Bayes error.

We can expect that if we minimize $D(\phi_{\hat{\theta}})$ then we will also (approximately) minimize the loss $R(\phi_{\hat{\theta}})$, so the next step is to define the feature extractor which minimizes $D(\phi_{\hat{\theta}})$. To achieve the minimum $D(\phi_{\hat{\theta}}) = 0$, the feature extractor $\phi_{\hat{\theta}}(x, y_k)$ and the parameter vector w have to satisfy the condition

$$w^T \phi_{\hat{\theta}}(x, y_k) = g^{-1}(P(y_k|x, y_1 \ldots y_s, \theta^\star)) = \log P(y_k|x, y_1 \ldots y_s, \theta^\star) - \log(1 - P(y_k|x, y_1 \ldots y_s, \theta^\star)) \quad (10)$$

for all data points x. Since the true distribution is unknown we need to construct the feature extractor that approximately

satisfies the equation with certain values of the vector w. Let us introduce the notation:

$$v(x, y_k, \theta) = \log P(y_k|x, y_1 \dots y_s, \theta) - \log(1 - P(y_k|x, y_1 \dots y_s, \theta)). \quad (11)$$

We approximate $v(x, y_k, \theta^\star)$ by Taylor expansion around $\hat{\theta}$:

$$v(x, y_k, \theta^\star) \approx v(x, y_k, \hat{\theta}) + \sum_{t=1}^{l} \frac{\partial v(x, y, \hat{\theta})}{\partial \theta_t}(\theta_t^\star - \hat{\theta}_t). \quad (12)$$

This gives us useful features for the feature extractor:

$$\phi_{\hat{\theta}}(x, y_k) = (v(x, y_k, \hat{\theta}), \frac{\partial v(x, y_k, \hat{\theta})}{\partial \theta_1}, \dots, \frac{\partial v(x, y_k, \hat{\theta})}{\partial \theta_l}). \quad (13)$$

Observing that the discriminative probability can be estimated from the generative model by normalizing over candidates,

$$P(y_k|x, y_1 \dots y_s, \hat{\theta}) = \frac{P(x, y_k|\hat{\theta})}{\sum_{j=1}^{s} P(x, y_j|\hat{\theta})}, \quad (14)$$

we obtain

$$v(x, y_k, \hat{\theta}) = \log P(x, y_k|\hat{\theta}) - \log \sum_{j \neq k} P(x, y_j|\hat{\theta}). \quad (15)$$

We will call this kernel the *TOP reranking kernel*.

III. Kernels Derived from MLP Probability Estimators

Previous work on kernels derived from probabilistic models has applied the derivation method directly to the parameters in a symbolic specification of the model. In this paper we investigate using a second level of parameterization, namely the weights of a neural network which has been trained to estimate the symbolic parameters. This has the advantage that the set of symbolic parameters does not have to be finite, a feature we exploit in our application of this method to natural language parsing. In this section we discuss several alternative ways in which the weights of the neural network can be mapped to a set of parameters to which we apply the kernel derivation methods.

A. MLP Probability Estimation for Large Categorization Problems

Methods for estimating probability distributions with multi-layer perceptrons are well understood. For multi-class categorization problems, the appropriate output function is the normalized exponential (i.e. softmax) [8]:

$$P(c_k|h) \approx \frac{\exp(\theta_k^T h)}{\sum_j \exp(\theta_j^T h)}, \quad (16)$$

where h is the hidden layer vector.

But for very large or unbounded categorization problems, we need to allow for structure in the set of categories. Such structure is manifested in a probability model where the probability of a category is computed from several more primitive probabilities, which form the parameters of the probability model. For example, in the parsing task we will discuss below, the probability of a given output parse tree is the multiplication of the probabilities for each decision made in constructing the parse tree. The neural network's job is then to estimate the probabilities of the individual decisions.

For structured domains such as natural language, issues of structural locality mean that generative probability models can be estimated more accurately [9], so for simplicity we will restrict our attention to such models. In a generative model, the probability of a category c_k is the joint probability of the output y and the input x, so maximizing $P(c_k)$ over categories which generate x is the same as maximizing $P(c_k|x)$. Given a category c_k which is decomposed into a sequence of decisions $d_1,\dots,d_m$, we need to calculate

$$P(c_k) \approx \prod_i P(d_i|h_i) \approx \prod_i \frac{\exp(\theta_{d_i}^T h_i)}{\sum_j \exp(\theta_{d_j}^T h_i)}, \quad (17)$$

where $h_1,\dots,h_m$ is the sequence of hidden layer vectors for the decisions $d_1,\dots,d_m$.

As mentioned in section II, the original TOP kernel requires a probability conditioned on the input x, but to compute this from the joint probability we need to normalize over all categories c_k which generate x. A computationally tractable approximation to computing this normalization factor was proposed in [9]. For each category c_k in the candidate list, this method first computes all the $P(d_i|h_i)$ required by equation (17). These computed decision sequences represent a sample of all possible decision sequences for the input x. The remaining non-computed sequences all consist of some (possible zero length) prefix $d_1,\dots,d_i$ of a computed sequence plus a postfix which generates the portion of x not generated by $d_1,\dots,d_i$. The approximation assumes that the probability of generating this remaining portion of x with some non-computed sequence with prefix $d_1,\dots,d_i$ is the same as the probability of generating this portion of x with some computed sequence with prefix $d_1,\dots,d_i$. Assuming that the candidates have been selected to have high probabilities (and thus to be particularly good at generating x), this is an over-estimate of the probabilities of the non-computed decision sequences. However, the success of a neural network training method which uses this approximation [9] indicates that it is a good basis for computing the derivatives of the conditional probability, which determines most of the features for the TOP kernel.

B. Mapping Network Weights to Parameters for a Kernel

When applying kernels with a large training corpus, we face efficiency issues because of the large number of neural network weights. Empirical results with parsing suggested that usage of the parameters other than the network output layer leads to a decrease in accuracy (see section VI). But even the size of an output layer might be very large if there are a large number of elementary decisions at each step. In addition, the kernels presented in section II all lead to feature vectors without many zero values. This happens because $P(d_i|h_i)$ depends on the output layer weights for all the alternative decisions d_j involved in its normalization factor (see section III-A). Thus the derivatives with respect to these

weights are all nonzero, even if the decisions d_j never appear in the parse. This makes an application of the VP algorithm infeasible in the case of a large number of decisions.

We can address this problem by freezing the normalization factor when computing the feature vector. If we consider output y as an output category for an input x, then we can rewrite expression (17) as:

$$\log P(x,y|\theta) = \textstyle\sum_i \log \frac{\exp(\theta^T_{d_i} h_i)}{\sum_j \exp(\theta^T_{d_j} h_i)} = \\ \textstyle\sum_i \theta^T_{d_i} h_i - \sum_i \log (\sum_j \exp(\theta^T_{d_j} h_i)). \quad (18)$$

We treat the parameters used to compute the first term $\sum_i \theta^T_{d_i} h_i$ as different from the parameters used to compute the second term, and we define our kernel only using the parameters in the first term. This means that the second term does not effect the derivatives in the formula for the feature vector $\phi(x,y)$. Thus the feature vector for the kernel will contain non-zero entries only in the components corresponding to the decisions actually made in the parse. We have applied this technique to the TOP reranking kernel, the result of which we will call the *efficient TOP reranking kernel*.

IV. The Probabilistic Model of Parsing

We apply the above kernel derivation methods to a neural network based probabilistic model of natural language parsing, as proposed in [7]. The first level of parameterization is a history-based probability model without independence assumptions, giving it an infinite number of parameters. The second level of parameterization is a recurrent neural network specifically designed for processing structures, namely Simple Synchrony Networks.

A history-based model of parsing decomposes a parse tree into a sequence of decisions about the tree, and then computes the probability of the tree as the multiplication of the probabilities of each decision conditioned on the history of previous decisions in the sequence:

$$P(d_1,...,d_m) = \Pi_i P(d_i|d_1,...,d_{i-1}). \quad (19)$$

Henderson [7] uses a form of left-corner parsing strategy to define a one-to-one mapping from phrase structure trees to parse sequences.

The number of parameters $P(d_i|d_1,...,d_{i-1})$ in this probability model is infinite, since the length of the history sequence is unbounded. In most models, independence assumptions would be made to keep the information in the conditional finite. We make no independence assumptions, making equation (19) an exact computation, and instead let the training of a recurrent neural network probability estimator learn an appropriate function for compressing the unbounded history.

The estimation of $P(d_i|d_1,...,d_{i-1})$ is done in two stages, first computing a finite representation $h(d_1,...,d_{i-1})$ of the unbounded history, and then estimating a probability distribution over d_i given this representation. The output probabilities are computed with the normalized exponential function, as was discussed in section III-A. The history representation is the hidden layer activation vector from a neural network architecture called Simple Synchrony Networks (SSNs). This hidden vector is incrementally computed from pre-defined features of the history (including the previous decision d_{i-1}) plus a finite set of previous history representations $h(d_1,...,d_j)$, $j < i-1$. As long as the history representation for position $i-1$ is always included in the inputs to the history representation for position i, any information about the entire sequence could be passed from history representation to history representation and be used to estimate the desired probability. However, learning is biased towards paying more attention to information which passes through fewer history representations.

To exploit this learning bias, structural locality between parse tree nodes is used to determine which history representations are input to which others. In this way, the number of representations which information needs to pass through in order to flow from history representation i to history representation j is determined by the structural distance between i's tree node and j's tree node, and not just the distance between i and j in the parse sequence. This provides the neural network with a linguistically appropriate inductive bias when it learns the history representations, as explained in more detail in [7].

Statistical parsers can be evaluated in a variety of ways, but the standard measure by which statistical parsers are compared is the F-measure over labeled constituents (i.e. the nonterminal nodes of the parse tree). This measure is a combination of recall and precision, where a constituent is correct if it has the same label and spans the same portion of the sentence as a constituent in the gold-standard parse. The method we use for large-margin optimization, discussed in the next section, allows us to incorporate a loss function, which indicates the relative importance of making one mistake versus making another. Our loss function at each individual sentence is defined as the difference of the F-measure of the best candidate and the F-measure of the candidate selected by the model. The best candidate has the largest F-measure, so the loss value is always non-negative.

V. Large-Margin Optimization

Once we have defined a kernel, general techniques for linear classifier optimization can be used to learn the given task. The most sophisticated of these techniques (such as Support Vector Machines) are unfortunately too computationally expensive to be used on large datasets like the Penn Treebank [10]. Instead we use a method which has often been shown to be virtually as good, the Voted Perceptron (VP) [11] algorithm.

The Voted Perceptron algorithm is an ensemble method for combining the various intermediate models which are produced during training a perceptron.

We modify the perceptron training algorithm by introducing a classification loss function to make it more suitable for structured classification, where zero-one classification loss is not the evaluation measure usually employed. This modification enables us to treat differently the cases when the classifier predicts a category with a large loss function value from the cases where the predicted category incurs a small classification

loss. Our approach is very similar to slack variable rescaling for Support Vector Machines proposed in [12].

We denote the *k*'th candidate output for the *j*'th data point x^j by y_k^j. Without loss of generality, let us assume that y_1^j is the best candidate in the list. Let $\Delta(y_k^j, y_1^j)$ be the classification loss for the candidate y_k^j. The learning algorithm we employ is presented in figure 1.

$w = 0$
for j = 1 .. n
 for k = 2 .. s
  if $w^T \phi(x^j, y_k^j) > w^T \phi(x^j, y_1^j)$
   $w = w + \Delta(y_k^j, y_1^j)(\phi(x^j, y_1^j) - \phi(x^j, y_k^j))$

Fig. 1. The modification of the perceptron algorithm

VI. The Experimental Results

To perform empirical evaluations of the proposed kernels, we considered the standard task of parsing the Penn Treebank WSJ corpus [10].

The input to the neural network parser is a sequence of tag-word pairs. The number of tag-word pairs in the vocabulary has a large impact on computation time, because a generative model needs an output for every tag-word pair for predicting the next word. We report results for two different vocabulary sizes, varying in the frequency with which tag-word pairs must occur in the training set in order to be included explicitly in the vocabulary. A frequency threshold of 200 resulted in a vocabulary of 508 tag-word pairs and a threshold of 20 resulted in 4215 tag-word pairs. We denote the probabilistic model trained with the vocabulary of 508 by SSN-Freq≥200, and the model trained with the vocabulary of 4215 by SSN-Freq≥20.

The same model was used for choosing the list of candidate parses and for the generative probabilistic model used for kernel feature extraction. For training and testing of the kernel models, we provided a candidate list consisting of the top 20 parses found by the generative probabilistic model. When using the Fisher kernel, we added the log-probability of the tree given by the probabilistic model as a feature. This was not necessary for the TOP kernels because they already contain a feature corresponding to the probability estimated by the probabilistic model (see section II).

We trained the VP model with five kernels using the 508 word vocabulary. Four of them were trained taking the derivatives with respect to only the output layer weights: the Fisher Kernel (Fisher-Out-Freq≥200), the original TOP kernel estimated by the method discussed in III-A (TOP-Out-Freq≥200), the TOP reranking kernel (TOP-Rerank-Out-Freq≥200), the efficient TOP reranking kernel (Eff-TOP-Rerank-Out-Freq≥200). Also we trained the reranking TOP kernel using all the weights of the neural network (TOP-Rerank-All-Freq≥200). Only the efficient TOP reranking kernel model (with features only for the output layer weights) was trained with the vocabulary of 4215 words (Eff-TOP-Rerank-Out-Freq≥20). The non-sparsity of the feature vectors for other kernels led to excessive memory requirements and larger testing time. In each case, the VP model was run for only one epoch. We would expect some improvement if we ran it for more epochs, as has been empirically demonstrated in other domains [11].

	LR	LP	$F_{\beta=1}$
TOP-Rerank-All-Freq≥200	86.8	88.4	87.6
SSN-Freq≥200	87.2	88.5	87.8
TOP-Out-Freq≥200	87.1	88.8	87.9
Fisher-Out-Freq≥200	87.2	88.8	87.9
TOP-Rerank-Out-Freq≥200	87.3	88.9	88.1
Eff-TOP-Rerank-Out-Freq≥200	87.3	88.9	88.1
SSN-Freq≥20	88.1	89.2	88.6
Eff-TOP-Rerank-Out-Freq≥20	88.2	89.7	88.9

TABLE I

Percentage labeled constituent recall (LR), precision (LP), and a combination of both ($F_{\beta=1}$) on validation set sentences of length at most 100

To avoid repeated testing on the standard testing set, we first compare the different models with their performance on the validation set. Note that the validation set wasn't used during learning of the kernel models or for adjustment of any parameters.

Standard measures of accuracy are shown in table I.[1] First note that taking into account non-output weights leads to worse results. Among the models which used only the output weights, both the Fisher kernel and all the TOP kernels show better accuracy than the baseline probabilistic model, but only the improvements of the TOP reranking kernels are statistically significant.[2] For the TOP reranking kernels, the improvement over the baseline is about the same with both vocabulary sizes. Also, note that the performance of the efficient TOP reranking kernel is the same as that of the original TOP reranking kernel for the same vocabulary. This kernel allows us to deal with arbitrarily large output vocabularies without any substantial increase in the kernel's computational costs.

In our opinion, the fact that the TOP reranking kernels are specifically designed for the reranking task led to the better results comparing both to the original TOP kernel based model and the Fisher kernel based model.

For comparison to previous results, table II lists the results on the testing set for our best model (Eff-TOP-Rerank-Out-Freq≥20) and several other statistical parsers [13], [1], [15], [7], [16], [17], [4], [3], [9], [18]. First note that the parser based on the TOP efficient kernel has 0.5% better accuracy than [7], which used the same parsing method as our baseline model, although the trained network parameters were not the same. When compared to other kernel methods, our approach performs better than those based on the Tree kernel [1], [15], and is only 0.2% worse than the best results achieved by a kernel method for parsing [3], [4].

[1] All our results are computed with the evalb program following the standard criteria in [13], using the standard training (sections 2–22, 39,832 sentences, 910,196 words), validation (section 24, 1346 sentence, 31507 words), and testing (section 23, 2416 sentences, 54268 words) sets [13].

[2] We measured significance with the randomized significance test of [14].

	LR	LP	$F_{\beta=1*}$
Collins99 [13]	88.1	88.3	88.2
Collins&Duffy02 [1]	88.6	88.9	88.7
Collins&Roark04 [15]	88.4	89.1	88.8
Henderson03 [7]	88.8	89.5	89.1
Charniak00 [16]	89.6	89.5	89.5
Eff-TOP-Rerank-Out-Freq≥20	89.1	90.1	89.6
Collins00 [17]	89.6	89.9	89.7
Shen&Joshi04 [4]	89.5	90.0	89.8
Shen et al.03 [3]	89.7	90.0	89.8
Henderson04 [9]	89.8	90.4	90.1
Bod03 [18]	90.7	90.8	90.7

* $F_{\beta=1}$ for previous models may have rounding errors.

TABLE II

PERCENTAGE LABELED CONSTITUENT RECALL (LR), PRECISION (LP), AND A COMBINATION OF BOTH ($F_{\beta=1}$) ON THE ENTIRE TESTING SET

VII. RELATED WORK

Most of the work on kernels for structured domains has focused on Convolution kernels [19], where the features of the kernel correspond to parts of the structure, and a dynamic programming approach is used to compute the inner product in the feature space. To tackle the sparsity problem and eliminate meaningless features, [3] and [20] both propose different feature selection approaches and demonstrate improvements over the corresponding baselines.

The first application of kernel methods to parsing was proposed by Collins and Duffy [1]. They used the Tree kernel, which is a Convolution kernel where the features of a tree are all its connected tree fragments. The VP algorithm was used to rerank the output of a statistical parser and demonstrated an improvement over the baseline. Later several approaches were considered to improve the Tree kernel and the optimization algorithms [2], [3], [4].

Previous work on deriving kernels from probabilistic models [5], [6] has demonstrated substantial improvement over their baselines for a large number of problems. The application of this approach to trees was suggested in [21], where an extension of Hidden Markov Model is used as the probabilistic model. Their kernel was applied to the problem of predicting the boiling point of a group of alkanes. They achieved results far worse than state-of-the-art results for the domain. This might be explained by the fact that a Fisher kernel is designed specifically for classification problems.

VIII. CONCLUSIONS

This paper proposes a method for deriving a kernel from a probabilistic model, which is specifically designed to maximize the kernel's usefulness for reranking. A modification of this method is also proposed, which enables efficient computation of the kernel when it is applied to a multi-layer perceptron probability estimator. This modification makes it feasible to apply the kernel to problems with very high dimensional outputs, without any significant degradation of accuracy. As an example of such a domain, we consider natural language parsing. We performed experiments on parse reranking using a neural network based statistical parser as both the probabilistic model and the source of the list of candidate parses. The kernel method significantly improved over the baseline model, achieving results amongst the best current statistical parsers and only 0.2% worse than the best current kernel-based parsing method.

REFERENCES

[1] M. Collins and N. Duffy, "New ranking algorithms for parsing and tagging: Kernels over discrete structures and the voted perceptron," in *Proc. 40th Meeting of Association for Computational Linguistics*, 2002, pp. 263–270.

[2] L. Shen and A. K. Joshi, "An SVM based voting algorithm with application to parse reranking," in *Proc. of the 7th Conf. on Computational Natural Language Learning*, Edmonton, Canada, 2003, pp. 9–16.

[3] L. Shen, A. Sarkar, and A. K. Joshi, "Using LTAG based features in parse reranking," in *Proc. of Conf. on Empirical Methods in Natural Language Processing*, Sapporo, Japan, 2003.

[4] L. Shen and A. K. Joshi, "Flexible margin selection for reranking with full pairwise samples," in *Proc. of the 1st Int. Joint Conf. on Natural Language Processing*, Hainan Island, China, 2004.

[5] T. S. Jaakkola and D. Haussler, "Exploiting generative models in discriminative classifiers," *Advances in Neural Information Processes Systems 11*, 1998.

[6] K. Tsuda, M. Kawanabe, G. Ratsch, S. Sonnenburg, and K. Muller, "A new discriminative kernel from probabilistic models," *Neural Computation*, vol. 14(10), pp. 2397–2414, 2002.

[7] J. Henderson, "Inducing history representations for broad coverage statistical parsing," in *Proc. joint meeting of North American Chapter of the Association for Computational Linguistics and the Human Language Technology Conf.*, Edmonton, Canada, 2003, pp. 103–110.

[8] C. M. Bishop, *Neural Networks for Pattern Recognition*. Oxford, UK: Oxford University Press, 1995.

[9] J. Henderson, "Discriminative training of a neural network statistical parser," in *Proc. 42nd Meeting of Association for Computational Linguistics*, Barcelona, Spain, 2004.

[10] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz, "Building a large annotated corpus of English: The Penn Treebank," *Computational Linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[11] Y. Freund and R. E. Schapire, "Large margin classification using the perceptron algorithm," in *Proc. of the 11th Annual Conf. on Computational Learning Theory*, Madisson WI, 1998, pp. 209–217. [Online]. Available: citeseer.ist.psu.edu/freund98large.html

[12] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun, "Support vector machine learning for interdependent and structured output spaces," in *Proc. 21st Int. Conf. on Machine Learning*, Banff, Alberta, Canada, 2004, pp. 823–830.

[13] M. Collins, "Head-driven statistical models for natural language parsing," Ph.D. dissertation, University of Pennsylvania, Philadelphia, PA, 1999.

[14] A. Yeh, "More accurate tests for the statistical significance of the result differences," in *Proc. 17th International Conf. on Computational Linguistics*, Saarbruken, Germany, 2000, pp. 947–953.

[15] M. Collins and B. Roark, "Incremental parsing with the perceptron algorithm," in *Proc. 42th Meeting of Association for Computational Linguistics*, Barcelona, Spain, 2004.

[16] E. Charniak, "A maximum-entropy-inspired parser," in *Proc. 1st Meeting of North American Chapter of Association for Computational Linguistics*, Seattle, Washington, 2000, pp. 132–139.

[17] M. Collins, "Discriminative reranking for natural language parsing," in *Proc. 17th Int. Conf. on Machine Learning*, Stanford, CA, 2000, pp. 175–182.

[18] R. Bod, "An efficient implementation of a new DOP model," in *Proc. 10th Conf. of European Chapter of the Association for Computational Linguistics*, Budapest, Hungary, 2003.

[19] D. Haussler, "Convolution kernels on discrete structures." University of Santa Cruz, Tech. Rep., 1999.

[20] J. Suzuki, H. Isozaki, and E. Maeda, "Convolutional kernels with feature selection for natural language processing tasks," in *Proc. 42nd Meeting of Association for Computational Linguistics*, Barcelona, Spain, 2004.

[21] L. Nicotra, A. Micheli, and A. Starita, "Fisher kernel for tree structured data," in *Proceedings of International Joint Conference on Neural Networks*, Budapest, Hungary, 2004.