EFFICIENT ONE-VS-ONE KERNEL RIDGE REGRESSION FOR SPEECH RECOGNITION

Jie Chen[†*], *Lingfei Wu*[‡*], *Kartik Audhkhasi*[†], *Brian Kingsbury*[†], *Bhuvana Ramabhadran*[†]

[†]IBM Thomas J. Watson Research Center, Yorktown Heights, NY 10598, USA
[‡]Computer Science Department, College of William and Mary, Williamsburg, VA 23185, USA
chenjie@us.ibm.com, lfwu@cs.wm.edu, {kaudhkha, bedk, bhuvana}@us.ibm.com

ABSTRACT

Recent evidences suggest that the performance of kernel methods may match that of deep neural networks (DNNs), which have been the state-of-the-art approach for speech recognition. In this work, we present an improvement of the kernel ridge regression studied in Huang et al., ICASSP 2014, and show that our proposal is computationally advantageous. Our approach performs classifications by using the one-vs-one scheme, which, under certain assumptions, reduces the costs of the one-vs-rest scheme by asymptotically a factor of c^2 in training time and c in memory consumption. Here, c is the number of classes and it is typically on the order of hundreds and thousands for speech recognition. We demonstrate empirical results on the benchmark corpus TIMIT. In particular, the classification accuracy is one to two percentages higher (in the absolute term) than the best of the kernel methods and of the DNNs reported by Huang et al, and the speech recognition accuracy is highly comparable.

Index Terms— large-scale kernel machines, one-vs-one multiclass classification, random Fourier features, deep neural networks, speech recognition

1. INTRODUCTION

The advent of deep neural networks (DNNs) substantially improved classification performance in many artificial intelligence and pattern recognition applications, speech recognition being one important example. State-of-the-art speech recognizers are typically built on a network of several hidden layers, with thousands of units per layer [1, 2]. The success of DNNs is, in part, due to the affordable computational resources (memory and flops) spent on solving a large-scale optimization problem, whose size is proportional to the number of connections between the units.

Recent evidences suggest, however, that DNNs may not be the only approach for achieving such a performance; kernel methods are competitive. Two notable examples that demonstrate the matching performance of kernel methods are the work by Huang et al [3] published in ICASSP 2014 (which used a kernel ridge regression), and the work by Lu et al [4] that was based on a form of kernel logistic regression. Both works apply a mapping from frame-level speech features to high-dimensional random Fourier features, which form a low-rank kernel that reduces the notorious computational expenses due to a fully dense kernel matrix.

A drawback of these methods is that a particularly large number of random features are needed to achieve a stable and a comparable performance. For a training set of size approximately 2M, the work [3, 5] used 400K random features to show a matching classification accuracy with that of the DNNs. Such a phenomenon may not be atypical given the argument made in [6]: In a low-rank approximate kernel, the rank (i.e., the number of random features) needs to be linear in the number of training size in order to maintain a comparable generalization error with that of the nonapproximate kernel.

In this paper, we consider the one-vs-one classification scheme, which substantially reduces the required number of random features for achieving a similar performance. The rationale is intuitive: In each subproblem the training size is only a portion of the whole data. Although the method itself is standard and it has been applied to support vector machines [7], we investigate in the context of kernel ridge regression and we propose an improvement that improves the efficiency of the computation. We also analyze that under certain assumptions, the training time and the memory consumption are asymptotically only $1/c^2$ and $1/c$, respectively, of those of the one-vs-rest scheme, where c is the number of classes. Considering that a speech recognition problem typically poses hundreds to thousands of classes, the one-vs-one scheme is clearly appealing. Experimentally, we demonstrate that this scheme achieves a slightly higher classification accuracy than do the kernel method and the state-of-the-art DNNs reported in [3].

We conclude the introduction by providing the details of the kernel method on which our proposal is built.

1.1. Kernel Ridge Regression

The standard setting for binary classification is that given data $\{(\boldsymbol{x}_i, y_i)\}_{i=1,\ldots,n}$, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{\pm 1\}$, find a function f that minimizes the discrepancy between $f(\boldsymbol{x}_i)$ and y_i under certain regularity conditions. In the kernel approach [8, 9], let $\mathcal{X} \subset \mathbb{R}^d$ denote a set where data are drawn from and $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ denote a positive-definite kernel function. For any k, there associates a reproducing kernel Hilbert space $\mathcal{H}_k$ with inner product $\langle \cdot, \cdot \rangle_{\mathcal{H}_k}$. Then, the function f is sought from $\mathcal{H}_k$ to minimize the risk functional

$$\mathcal{L}(f) = \sum_{i=1}^{n} V(f(\boldsymbol{x}_i), y_i) + \lambda \langle f, f \rangle_{\mathcal{H}_k}$$

where V denotes a loss function and $\lambda \geq 0$ is the regularization. The Representer Theorem [10] states that the minimizer is in the form

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \alpha_i k(\boldsymbol{x}, \boldsymbol{x}_i), \tag{1}$$

for some set of coefficients $\{\alpha_i\}$. When V is the squared loss $V(t, y) = (t - y)^2$, the vector of α_i's is simply the solution of the linear system

$$(K + \lambda I)\boldsymbol{\alpha} = \boldsymbol{y}, \tag{2}$$

where K is the kernel matrix of elements $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and $\boldsymbol{y}$ is the vector of labels y_i. Such a computation is nothing but the kernel ridge

[*]Both authors contributed equally to this manuscript.

regression. Another well-known example of $V(t, y)$ is the hinge loss $\max\{0, 1 - ty\}$, which leads to the support vector machine.

1.2. Random Fourier Features Approximation

Kernel ridge regression is computationally challenging because the $n \times n$ matrix K is fully dense. A straightforward solution of (2) requires at lest n^2 memory and $O(n^2)$–$O(n^3)$ flops depending on the linear solver one uses. A stream of research on kernel methods approximates the kernel function k by some structures (e.g., low-rank) such that the solution of (2) is less expensive (see, e.g., [11, 12, 13]). The random Fourier features method [12] approximates k by

$$k_{\text{RF}}(\boldsymbol{x}, \boldsymbol{x}') = \frac{2}{r} \sum_{i=1}^{r} \cos(\boldsymbol{\omega}_i^T \boldsymbol{x} + b_i) \cos(\boldsymbol{\omega}_i^T \boldsymbol{x}' + b_i),$$

where r is the number of random features, each scalar b_i is an iid sample of Uniform$[0, 2\pi]$, and each iid vector $\boldsymbol{\omega}_i$ comes from a distribution with density $\hat{k}$, the Fourier transform of k in the Bochner sense. Rahimi and Recht [12] showed a uniform convergence of the approximation.

This method admits a more economic computation than directly solving (2). Let $\boldsymbol{z}(\boldsymbol{x})$ be a row vector of elements $\sqrt{2/r} \cos(\boldsymbol{\omega}_i^T \boldsymbol{x} + b_i)$ for $i = 1, \ldots, r$ and let Z be a matrix stacking the rows $\boldsymbol{z}(\boldsymbol{x}_i)$ for $i = 1, \ldots, n$. Then, the prediction function f in (1) becomes

$$\begin{aligned} f_{\text{RF}}(\boldsymbol{x}) &= \boldsymbol{z}(\boldsymbol{x}) \cdot Z^T (Z Z^T + \lambda I)^{-1} \boldsymbol{y} \\ &= \boldsymbol{z}(\boldsymbol{x}) \cdot (Z^T Z + \lambda I)^{-1} (Z^T \boldsymbol{y}). \end{aligned} \tag{3}$$

Hence, as oppose to solving an $n \times n$ system with $K + \lambda I$ in (2), one solves only an $r \times r$ system with $Z^T Z + \lambda I$ in (3).

1.3. Multiclass Classification

When the classification problem has $c > 2$ classes, a few approaches adapting the preceding discussions exist. The approach in [3] converts the problem into c binary classifications, in each of which the task is to determine how likely a point $\boldsymbol{x}$ belongs to the class i as oppose to other classes, for $i = 1, \ldots, c$. Specifically, let $\boldsymbol{y}_i$ be a label vector whose elements are $+1$ if the corresponding training points belong to class i and -1 otherwise. Correspondingly, let the prediction function (3), with subscript "RF" dropped for clarity, be named $f_i(\boldsymbol{x})$. Then, the predicted class of $\boldsymbol{x}$ is $\operatorname{argmax}_i \{f_i(\boldsymbol{x})\}$.

The prescribed approach is named "one-vs-rest." In what follows, we consider a more favored approach: "one-vs-one."

2. ONE-VS-ONE MULTICLASS CLASSIFICATION

A drawback of "one-vs-rest" is that the number r of random features needs to be particularly large for a matching performance with DNNs. Experiments on TIMIT in [3, 5] allured to setting $r = 400$K, which is approximately $1/5$ of the training size n. Whereas theoretical guidance on the choice of r is rare, Dai et al. [6] argued that the generalization error of the random features method is $O(1/\sqrt{r} + 1/\sqrt{n})$. This bound implies that r cannot be too small compared with n for maintaining the predictive power. A linear relationship is often desired.

One natural idea for reducing r, then, resorts to the "one-vs-one" scheme, because the training size is amortizd by the number of classes. This scheme converts the multiclass classification problem into $c(c-1)/2$ binary classifications, each of which uses only the training points from a pair (i, j) of classes and trains a classifier that votes between the two classes. Then, for any point, the class with the largest vote is the prediction.

2.1. Algorithm

Since the "one-vs-one" scheme loops over all pairs (i, j), we index the prediction function (3) by using ij (and same as before, removing the subscript "RF" for clarity):

$$f_{ij}(\boldsymbol{x}) = \boldsymbol{z}(\boldsymbol{x}) \cdot \boldsymbol{\beta}_{ij} \quad \text{with} \quad \boldsymbol{\beta}_{ij} = (Z_{ij}^T Z_{ij} + \lambda I)^{-1} (Z_{ij}^T \boldsymbol{y}_{ij}), \tag{4}$$

where Z_{ij} is a matrix by stacking the rows $\boldsymbol{z}(\boldsymbol{x}_l)$ for all points $\boldsymbol{x}_l$ from class i or j, and $\boldsymbol{y}_{ij}$ is a vector whose elements are $+1$ if $\boldsymbol{x}_l$ is in class i and -1 otherwise.

If one naively forms $Z_{ij}^T Z_{ij}$ and $Z_{ij}^T \boldsymbol{y}_{ij}$ each time for a pair (i, j), then much work is redundant. For example, if $\boldsymbol{x}_l$ is in class i, the row $\boldsymbol{z}(\boldsymbol{x}_i)$ appears $c - 1$ times, each for a different j. This redundancy can be eliminated by observing that

$$Z_{ij}^T Z_{ij} = Z_i^T Z_i + Z_j^T Z_j \quad \text{and} \quad Z_{ij}^T \boldsymbol{y}_{ij} = Z_i^T \mathbf{1} - Z_j^T \mathbf{1},$$

where Z_i is a matrix by stacking the rows $\boldsymbol{z}(\boldsymbol{x}_l)$ for all points $\boldsymbol{x}_l$ from class i and $\mathbf{1}$ is a vector of all ones with a matching length. Therefore, we propose precomputing

$$A_i := Z_i^T Z_i \quad \text{and} \quad \boldsymbol{g}_i := Z_i^T \mathbf{1}$$

for all classes i. Then, (4) becomes

$$\boldsymbol{\beta}_{ij} = (A_i + A_j + \lambda I)^{-1} (\boldsymbol{g}_i - \boldsymbol{g}_j). \tag{5}$$

Because forming the linear systems (5) is much more expensive than solving them, the saving in flops through such a simple rearrangement is substantial.

Algorithm 1 summarizes the computations.

Algorithm 1 One-vs-one classification with random Fourier features

```
    // Training
 1: Generate random ω_i and b_i for i = 1, ..., r
 2: for all classes i do
 3:     Generate temporary Z_i
 4:     Compute A_i = Z_i^T Z_i and g_i = Z_i^T 1
 5: end for
 6: for all pairs of classes i, j where i < j do
 7:     Compute β_ij = (A_i + A_j + λI)^{-1} (g_i - g_j)
 8: end for
    // For a test point x
 9: for all pairs of classes i, j where i < j do
10:     Generate z(x)
11:     Compute f_ij(x) = z(x) · β_ij
12: end for
```

2.2. Linear Solves

A straightforward method for solving (5) is Cholesky factorizations, because the matrix is positive definite [14]. This approach requires $n^3/3$ flops for factorization and $2n^2$ flops for triangular solves. Here, the notation n generically denotes the matrix size, which in the case of (5) is equal to the number of random features.

It is yet often advantageous to use an iterative method to solve (5) if the iterations converge rapidly. Two iterative methods with convergence guarantee for positive-definite systems are

CG and GMRES [15]. Let k be the number of iterations, with a subscript distinguishing the method. Without preconditioning, CG requires $O(k_{\text{CG}}n^2)$ flops whereas a full GMRES requires $O(k_{\text{GMRES}}n^2 + k_{\text{GMRES}}^2 n)$ flops.

In theory, the number of iterations is tied to the condition number, the spectral gap, and the desired solution accuracy, but not the size of the matrix. In practice, experience points to the observation that full GMRES converges much more quickly than does CG, and it also completes much faster than do Cholesky factorizations, for a reasonable accuracy that does not deteriorate the prediction quality. The following table gives a flavor of the number of GMRES iterations for one particular system (5) at 10K random features. Prediction results verify that it suffices to set the tolerance to 1e-3.

	Relative tolerance of solution accuracy					
λ	1e-1	1e-2	1e-3	1e-4	1e-5	1e-6
1e-1	4	12	44	83	105	123
1e-2	4	12	48	111	158	196
1e-3	4	12	49	127	210	283
1e-4	4	12	49	131	228	316
1e-5	4	12	49	131	230	320
1e-6	4	12	49	131	230	320

2.3. Computational Costs, Part 1

We now analyze the computational costs of the proposed method (Algorithm 1) and compare them with those of the standard kernel ridge regression (cf. Section 1.1) and of the 1-vs-rest random features method (cf. Section 1.2). The results are summarized in Part 1 of Table 1. Note that all the expressions in the table are in the big-O sense.

The costs of the standard method are straightforward. For training, the computation is to construct the kernel matrix ($O(dn^2)$ time and $O(n^2)$ memory) and to solve (2) with c different $\boldsymbol{y}$'s. We use $T(n, c)$ to generically denote the time for solving an $n \times n$ system with c right-hand sides. For testing on each point $\boldsymbol{x}$, the computation is to construct a vector of elements $k(\boldsymbol{x}, \boldsymbol{x}_i)$ in $O(dn)$ time and to multiply this vector with the c different $\boldsymbol{\alpha}$'s resulting from training (with $O(cn)$ time and memory).

Next, we consider the one-vs-rest random features method. For training, the dominant computation is to generate the random numbers $\boldsymbol{\omega}_i$ and b_i (in $O(dr)$ time and memory), to construct the matrix $Z^T Z$ (whose storage is $O(r^2)$) and c different vectors $Z^T \boldsymbol{y}$ (whose storage is $O(rc)$), and to solve c linear systems indicated by (3) (in $T(r, c)$ time). Note that the construction of $Z^T Z$ and $Z^T \boldsymbol{y}$ can be more efficient than does the naive approach of generating the whole matrix Z. The economic approach is to generate a temporary storage Z_i each time for a class i, compute $Z_i^T Z_i$ and $Z_i^T \mathbf{1}$, and accumulate them to the correct memory location of $Z^T Z$ and $Z^T \boldsymbol{y}$. Such an approach costs $O(drn + r^2 n + c^2 r)$ time and $O(\max_i(rn_i))$ temporary memory, where n_i denotes the number of points in class i. For testing, it requires the storage of the random numbers $\boldsymbol{\omega}_i$ and b_i (with $O(dr)$ cost) and the storage of the linear system solutions (with $O(cr)$ cost). Additionally, constructing $\boldsymbol{z}(\boldsymbol{x})$ takes $O(dr)$ time and multiplying $\boldsymbol{z}(\boldsymbol{x})$ to the linear system solutions takes $O(cr)$ time.

Now, we analyze the proposed one-vs-one method. We use $\tilde{r}$ to denote the number of random features here. Most components of the costs follow the same analysis as in the preceding paragraph, by replacing r by $\tilde{r}$; hence, we discuss only the components that are different. First, the number of linear systems are increased from c to $O(c^2)$. Hence, the storage of the linear system solutions is $O(c^2\tilde{r})$ instead of $O(c\tilde{r})$, and the storage of the matrices is $O(c\tilde{r}^2)$ instead of $O(\tilde{r}^2)$. Second, generating the $O(c^2)$ different systems (5) takes $O(d\tilde{r}n + \tilde{r}^2 n + c^2\tilde{r}^2)$ time, and obviously solving them takes $T(\tilde{r}, 1) \cdot c^2$ time. Summarizing these differences we have the last column of Part 1 of Table 1.

2.4. Computational Costs, Part 2

Two factors are not directly comparable in Part 1: the relative magnitude of r and $\tilde{r}$; and the time cost T for varying matrix sizes and numbers of right-hand sides. To allow for a deeper analysis, we make the following assumptions:

1. The class sizes are balanced; i.e., $n_i = n/c$ for all i.
2. All linear systems are solved by using full GMRES and the number of iterations is linear in the matrix size. In other words, $T(n, s) = \epsilon s n^2$ with $\epsilon < 1$. Here, the number of iterations is modeled as ϵn, where ϵ is expected to be $\ll 1$ (see, for example, the table in Section 2.2).
3. The number of random features is linear in the training size, following the discussions at the beginning of Section 2. Hence, we let $r = \delta n$ with $\delta < 1$; and together with Assumption 1, let $\tilde{r} = 2\delta n/c$. Experimental results indicate that δ is a moderate fraction of 1.

With these assumptions, Part 1 of Table 1 is translated to Part 2. If one considers only the training set size n and ignores the other factors, all three methods fall in the notorious $O(n^3)$-time and $O(n^2)$-memory regime for training. However, the significant computational improvements brought about by the methods from left to right are closely tied to the ignored factors $\epsilon \ll 1$, $\delta < 1$, and $c \gg 1$. Comparing the leading terms in the training costs of the proposed one-vs-one scheme with those of the one-vs-rest scheme, we see a factor of c^2 saving in time and a factor of c saving in memory.

2.5. Probability Estimates

As oppose to the prediction of label y, more important in a speech recognition is the posterior probability distribution $\Pr(y = i \mid \boldsymbol{x})$, because the probabilities are fed into a Viterbi decoding for finding the best sequence of labels for an input utterance. To compute the posterior probabilities, we follow the approach of Wu et al. [16]. First, we estimate the pairwise conditional probability $\mu_{ij}(\boldsymbol{x}) := \Pr(y = i \mid y = i \text{ or } j,\ \boldsymbol{x})$ from the prediction function $f_{ij}(\boldsymbol{x})$, based on a logistic mapping

$$\mu_{ij}(\boldsymbol{x}) = \frac{\exp(\alpha_{ij} f_{ij}(\boldsymbol{x}) + \beta_{ij})}{1 + \exp(\alpha_{ij} f_{ij}(\boldsymbol{x}) + \beta_{ij})}. \tag{6}$$

Here, the unknowns α_{ij} and β_{ij} are computed through a maximum likelihood estimation on the validation set. Then, the posterior probabilities $p_i \equiv \Pr(y = i \mid \boldsymbol{x})$ for all i are obtained by solving the optimization problem:

$$\begin{aligned} \min_{\boldsymbol{p}} \quad & \sum_{i=1}^{c} \sum_{j \neq i} (\mu_{ji} p_i - \mu_{ij} p_j)^2, \\ \text{s.t.} \quad & \sum_{i=1}^{c} p_i = 1, \quad p_i \geq 0 \ \forall i. \end{aligned} \tag{7}$$

Let the validation set have size n'. The cost of estimating the unknowns in (6) is $O(k_{\text{Newton}} c^2 n')$, if the maximum likelihood problem is solved by using a Newton solver, where k_{Newton} is the number of Newton iterations. The cost of computing the posterior probabilities is $O(c^3)$, because the nonlinear problem (7) is equivalent to a linear system of size $(c + 1) \times (c + 1)$.

Table 1. Computational costs of various kernel methods. The Big-O symbol is omitted for clarity.
Part 1 notation. n: Number of points; n_i: Number of points in class i; d: Dimension; c: Number of classes; r: Number of features in one-vs-rest; $\tilde{r}$: Number of features in one-vs-one; $T(n, s)$: Time solving an $n \times n$ linear system with s right-hand sides.

	Standard one-vs-rest	RF one-vs-rest	RF one-vs-one
Train time	$dn^2 + T(n, c)$	$drn + r^2n + c^2r + T(r, c)$	$d\tilde{r}n + \tilde{r}^2n + c^2\tilde{r}^2 + T(\tilde{r}, 1) \cdot c^2$
Train memory	n^2	$\max_i(rn_i) + r^2 + (c + d)r$	$\max_i(\tilde{r}n_i) + c\tilde{r}^2 + (c^2 + d)\tilde{r}$
Test time	$(c + d)n$	$(c + d)r$	$(c^2 + d)\tilde{r}$
Test memory	cn	$(c + d)r$	$(c^2 + d)\tilde{r}$

Part 2 assumption: (a) Class sizes are balanced, that is, $n_i = n/c$ for all i; (b) Linear systems are solved by using GMRES; (c) $T(n, s) = \epsilon sn^3$ with $\epsilon < 1$; (d) $r = \delta n$ with $\delta < 1$; and (e) $\tilde{r} = 2\delta n/c$.

	Standard one-vs-rest	RF one-vs-rest	RF one-vs-one
Train time	$\epsilon cn^3 + dn^2$	$\delta^2(1 + \epsilon\delta c)n^3 + \delta dn^2 + \delta c^2 n$	$\delta^2/c^2(1 + \epsilon\delta c)n^3 + \delta(\delta + d/c)n^2$
Train memory	n^2	$\delta/c(1 + \delta c)n^2 + \delta(c + d)n$	$\delta/c^2(1 + \delta c)n^2 + \delta(c + d/c)n$
Test time	$(c + d)n$	$\delta(c + d)n$	$\delta(c + d/c)n$
Test memory	cn	$\delta(c + d)n$	$\delta(c + d/c)n$

3. EXPERIMENTAL RESULTS

We perform experiments on the benchmark corpus TIMIT and demonstrate the computational efficiency of the proposed method.

The preparation of the data followed that of [3]. In particular, the data was segmented into 5-millisecond frames, each of which was transformed to a 40-dimensional fMLLR feature vector. Each frame was concatenated with five frames on each side so that the data points for classification are 440-dimensional. One tenth of the data was separated from the training set for parameter tuning and validation. The evaluation was performed on the core test set. The number of classes was 147 (49 phonemes $\times$ 3 states).

The program was implemented in C with the linear algebra computations linked to ESSL. The experiments were performed on a shared memory machine with 72 Power8 cores and 512GB memory.

The kernel function k used for reporting results in this section was the Gaussian kernel. We found that other kernels (e.g., the Laplace kernel popularized by [12]) behaved similarly in terms of the best achievable performance.

The floating-point operations were done with single precision. We experimented with double precision as well and found that the difference in accuracy was negligible. This difference was also comparable with the variation caused by the random nature of the method. Clearly, the gain in computation with reduced precision lies in timing and storage: The time for both training and testing was reduced by approximately one half, and the same for memory consumption.

Table 2. Classification errors, timing, and memory consumption.

#Features	Err. (vote)	Err. (prob)	Train Time	Memory
5K	34.37%	33.12%	240s	21GB
10K	33.70%	32.57%	1188s	69GB
15K	33.39%	32.29%	2544s	143GB
20K	33.16%	32.04%	4254s	249GB

Table 2 shows the results as the number $\tilde{r}$ of random features increases. The classification accuracy can be evaluated in two ways: based on voting or on the probability estimates (see Section 2.5). In fact, the estimation method holds no guarantee that the probabilities fully agree with the votes. The results in Table 2 and our experiences indicate that predictions according to the probability estimates are consistently better than those according to votes.

Table 3. Comparison of the results of the proposed method with the best results in [3]. "Cl. Err." means classification error and "PER" means phone error rate.

Method	Cl. Err.	PER
MSE-DNN, 2K units, 2 layers [3]	34.12%	22.2%
CE-DNN, 4K units, 3 layers [3]	33.34%	20.5%
RF one-vs-rest, 400K features [3]	33.67%	21.3%
RF one-vs-one, 5K features	33.12%	21.8%
RF one-vs-one, 10K features	32.57%	21.5%
RF one-vs-one, 15K features	32.29%	21.0%
RF one-vs-one, 20K features	32.04%	20.9%

Table 3 compares our results with the best results reported in [3], including those of the state-of-the-art DNNs and of the one-vs-rest kernel method. In particular, the proposed method achieves the best classification accuracies and its speech recognition accuracies (PER) are highly comparable with the other methods.

Note that the methods under comparisons are trained under different computer architectures best suited for the nature of the computation: DNNs were trained on a GPU, one-vs-rest was train on a distributed memory BlueGene, and the proposed one-vs-one was trained on a shared memory machine. Also note that the one-vs-rest in [3] was computed by using a reformulated algorithm different from the assumption of our cost analysis in Section 2.3. Hence, the required computational resources were not straightforwardly comparable. Nevertheless, the time and memory listed in Table 2 indicate the economic computations of the proposed method.

4. ACKNOWLEDGMENT

We are grateful to Xing Liu for help on experiments. J. Chen is supported in part by the XDATA program of the Advanced Research Projects Agency (DARPA), administered through Air Force Research Laboratory contract FA8750-12-C-0323. This work was done while L. Wu was a summer intern at IBM Research.

5. REFERENCES

[1] A. Mohamed, G. E. Dahl, and G. Hinton, "Acoustic modeling using deep belief networks," *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 20, no. 1, pp. 14–22, 2012.

[2] Geoffrey Hinton, Li Deng, Dong Yu, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara Sainath George Dahl, and Brian Kingsbury, "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] P. Huang, H. Avron, T. N. Sainath, V. Sindhwani, and B. Ramabhadran, "Kernel methods match deep neural networks on TIMIT," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2014.

[4] Zhiyun Lu, Avner May, Kuan Liu, Alireza Bagheri Garakani, Dong Guo, Aurélien Bellet, Linxi Fan, Michael Collins, Brian Kingsbury, Michael Picheny, and Fei Sha, "How to scale up kernel methods to be as good as deep neural nets," arXiv:1411.4000 [cs.LG], 2015.

[5] V. Sindhwani and H. Avron, "High-performance kernel machines with implicit distributed optimization and randomization," *Technometrics*, 2015, to appear.

[6] Bo Dai, Bo Xie, Niao He, Yingyu Liang, Anant Raj, Maria-Florina Balcan, and Le Song, "Scalable kernel methods via doubly stochastic gradients," in *Advances in Neural Information Processing Systems 27*, 2014.

[7] Chih-Chieh Cheng and B. Kingsbury, "Arccosine kernels: Acoustic modeling with infinite neural networks," in *IEEE International Conference on Acoustics, Speech and Signal Processing*, 2011.

[8] Bernhard Schölkopf and Alexander J. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*, The MIT Press, 2001.

[9] Trevor Hastie, Robert Tibshirani, and Jerome Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, second edition, 2009.

[10] Bernhard Schölkopf, Ralf Herbrich, and Alex J. Smola, "A generalized representer theorem," *Lecture Notes in Computer Science*, vol. 2111, pp. 416426, 2001.

[11] Petros Drineas and Michael W. Mahoney, "On the Nyström method for approximating a Gram matrix for improved kernel-based learning," *Journal of Machine Learning Research*, vol. 6, pp. 2153–2175, 2005.

[12] Ali Rahimi and Ben Recht, "Random features for large-scale kernel machines," in *Neural Infomration Processing Systems*, 2007.

[13] J. Yang, V. Sindhwani, Q. Fan, H. Avron, and M. Mahoney, "Random Laplace feature maps for semigroup kernels on histograms," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014.

[14] Gene H. Golub and Charles F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, 3rd edition, 1996.

[15] Yousef Saad, *Iterative Methods for Sparse Linear Systems*, Society for Industrial and Applied Mathematics, second edition, 2003.

[16] Ting-Fan Wu, Chih-Jen Lin, and Ruby C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *The Journal of Machine Learning Research*, vol. 5, pp. 975–1005, 2004.