Automatic Kernel Construction During the Neural Network Learning by Modified Fast Singular Value Decomposition

Norbert Jankowski[1](✉) and Grzegorz Dudek[2]

[1] Department of Informatics, Nicolaus Copernicus University, Toruń, Poland
norbert@umk.pl

[2] Faculty of Electrical Engineering, Technical University of Częstochowa, Częstochowa, Poland
grzegorz.dudek@pcz.pl

Abstract. Thanks to the broad application fields, learning neural networks is still a more significant problem nowadays. Any attempt in the construction of faster learning algorithms is highly well come. This article presents a new way of learning neural networks with kernels with modified pseudo-inverse learning by modified SVD.

The new algorithm constructs the kernels during the learning and estimates the right number in the results. There is no longer a need to define their number of kernels before the learning. This means there is no need to test networks with a number of kernels that is too large, and the number of kernels is no longer a parameter in the selection process (in cross-validation).

The results show that the proposed algorithm constructs reasonable kernel bases, and final neural networks are accurate in classification.

Keywords: neural network learning · kernels · classification · singular value decomposition

1 Introduction

The classification or approximation problems are represented by a training dataset as a matrix X, which consists of learning vectors $\mathbf{x}_i$ ($\mathbf{x}_i \in R^n, i \in [1,\ldots,m]$), and each vector has corresponding $y_i \in \{0,\ldots,K\}$ (in binary case $y_i = \pm 1$).

We can look at a nonlinear model constructed as a linear combination of nonlinear functions:

$$F(\mathbf{x},\mathbf{w}) = \sum_{j=1}^{l} w_j g_j(\mathbf{x}) + w_0, \tag{1}$$

which is a combination ($\mathbf{w}$) of kernels g_j. The above form is fully consistent with the Radial Basis Function Network (RBFN) [1] and the Extreme Learning

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2024
L. Franco et al. (Eds.): ICCS 2024, LNCS 14834, pp. 205–212, 2024.
https://doi.org/10.1007/978-3-031-63759-9_25

Machine (ELM) [4,5] as well. The same form also applies to nonlinear Support Vector Machines [8].

The sigmoidal function was the original kernel used in the ELM, while in the RBFN the Gaussian kernel is usually chosen (although it is sometimes used in ELM as well [2,5]). [7] has introduced multilayer perceptron learned as a multilayered ELM. It can be seen as a special case of deep learning—first layers are the layers of autoencoders and the final layer is a typical ELM.

The goal (the error function) for the neural networks can be defined by:

$$J_n(\mathbf{w}, G) = ||G\mathbf{w} - \mathbf{y}||^2 \tag{2}$$

where G is defined by: $G = \begin{bmatrix} 1 & g_1(\mathbf{x}_1) & g_2(\mathbf{x}_1) & \cdots & g_l(\mathbf{x}_1) \\ 1 & g_1(\mathbf{x}_2) & g_2(\mathbf{x}_2) & \cdots & g_l(\mathbf{x}_2) \\ \cdots & & & & \\ 1 & g_1(\mathbf{x}_m) & g_2(\mathbf{x}_m) & \cdots & g_l(\mathbf{x}_m) \end{bmatrix}$.

To find the solution we have to compute the gradient:

$$\nabla J(\mathbf{w}, G) = 2G^T(G\mathbf{w} - \mathbf{y}). \tag{3}$$

After a few substitutions, we finally have:

$$\mathbf{w} = (G^T G)^{-1} G^T \mathbf{y} = G^{\dagger}\mathbf{y}. \tag{4}$$

where $G^{\dagger}$ is the Moore-Penrose pseudo-inverse matrix of G. The SVD can be used to compute the pseudo-inverse of G.

The costs of learning via SVD are relatively low—it is a sum of costs of construction of G and costs of SVD. The complexity of construction of G is $O(mln)$ (computation of single G_{ij} is $O(n)$). The complexity of SVD of matrix G is $O(ml^2)$.

In the above learning scheme, we see that for the learning to be based on kernels, the kernel has to be created earlier, while we do not know the correct number of kernels before the learning. Therefore, the main contribution of this article is to propose a way in which the kernels necessary to train the neural network are created and used during the training process, i.e. in the above scheme, it would be during the execution of the SVD procedure. This is presented in the next section. The last section presents analysis of empirical results.

2 Automatic Kernel Construction During the Pseudo-inverse Learning

The above algorithm can be sped up and simplified by changing two things. The first is related to the fact that the number of kernels l as well as the G matrix must be known before starting pseudo-inversion training. Note that the selection process of the number of kernels is typically done by repeated training and testing with a different number of kernels [2]. Here we propose that the G matrix as well as the number of kernels will be determined in the modified SVD algorithm. The second change will consist in changing the fast version of the SVD determination to enable the dynamic creation of the G matrix.

Thus, the starting point is the fast SVD algorithm proposed in [3]. The main idea of this algorithm lies in construction of matrix Q of the following property:

$$G \approx QQ^T G \tag{5}$$

where Q is matrix composed of orthonormal columns ($m \times k$, $k \le l$,). Q is constructed by low-rank approximation of a matrix.

Now to compute the SVD of G we have to: 1 – construct matrix Q, 2 – construct matrix B such that $B = Q^T G$ (B is $k \times l$), 3 – compute $SVD(B) = \tilde{U}\Sigma V^T$ and 4 – define the final matrix U as $U = Q\tilde{U}$. In the next step final vector $\mathbf{w}$ can be computed as in Eq. 4.

2.1 Standard Q Matrix Construction by Gaussian Randomization and Ortogonalization

The proposed construction of the matrix Q in [3] was performed by iterative Gaussian randomization and orthogonalization. First let the matrix H be defined by

$$\mathbf{h}^{(i)} = G\boldsymbol{\omega}^{(i)} \qquad i = 1, \ldots, k, \tag{6}$$

where $\boldsymbol{\omega}^{(i)}$ is a random Gaussian vector $1 \times n$, and $H \in \mathcal{M}_{m\times k}(\mathbb{R})$. The Gaussian random vector is drawn from a Gaussian distribution with a mean of 0 and variance equal to 1.

The next step is to use an orthonormalization algorithm to build an orthonormal matrix Q from matrix H. The two steps, the randomization and the orthonormalization, can be performed iteratively.

The selection of k as the number of columns in the matrix Q is helped by the following lemma from [3]:

Lemma 1. *Let $B \in \mathcal{M}_{m\times m}(\mathbb{R})$, r be a positive integer and $\alpha > 1$. Draw an independent family $\{\boldsymbol{\omega}^{(i)} : i = 1, \ldots, r\}$ of standard Gaussian vectors. Then:*

$$||B|| \le \alpha\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,r} ||B\boldsymbol{\omega}^{(i)}||$$

with a probability of at least $1 - \alpha^{-r}$.

In the context of the above definition of the orthonormal matrix Q and the above lemma, the direct conclusion is that the decomposition error is bounded as below:

$$||(I - QQ^T)G|| \le 10\sqrt{\frac{2}{\pi}} \max_{i=1,\ldots,r} ||(I - QQ^T)G\boldsymbol{\omega}^{(i)}|| \tag{7}$$

with a probability of at least $1 - 10^{-r}$. The detailed algorithm for construction of Q is presented as Algorithm 4.2 in [3] on page 25.

The complexity of the above algorithm is $O(mlk)$ where $m \times l$ is the size of G and k is the number of columns in Q ($k \leq l$). After the construction of Q, the SVD is calculated on the matrix $B = Q^T G$ and the matrix $B \in \mathcal{M}_{k \times l}(\mathbb{R})$. That means that the complexity of SVD on B is $O(kl^2)$. This leads us to a final complexity of the fast version of SVD: $O(mlk)$.

2.2 New Way of Q Matrix Construction Directly from Learning Data

The concept of the algorithm that we want to publish is an attempt to avoid building kernel matrices G that are too large. Then the complexity of building kernel matrices could change from $O(mnl)$ to $O(m^2n)$ when we have to insert kernels in too many data vectors. This also significantly impacts the final complexity of training via SVD, where the complexity will go from $O(ml^2)$ to $O(m^3)$. In the old procedure of creating the Q matrix, unfortunately, we had to have a ready-to-use kernel matrix G. Training ends with poorer classification accuracy when the kernel matrix is too small. However, training on a large kernel matrix has high (sometimes unnecessarily) complexity. Thanks to this, the new Q matrix creation procedure dynamically adjusts the number of kernels, i.e., the size of the kernel matrix increases dynamically in the new Q matrix creation algorithm. As a result, in the process of creating the Q matrix, the necessary number of kernels will be generated, and the further SVD learning process will take place on a possibly small matrix, reducing the complexity of both stages and removing the need to sample the appropriate number of kernels repeatedly. So, we do not want to build an array of 5000 kernels if 48 kernels are enough.

For this purpose, the new procedure for determining the Q matrix will not work on the basis of the already prepared G matrix, but on the basis of the X data matrix, which is necessary to construct kernels, and the kernel columns of the G matrix will be created as needed.

To make this process workable, random linear combinations of columns of matrix G will gradually cover more and more of the target number of columns (kernels) of the final matrix G. Initially, the algorithm will use the starting amount of kernels $\nu = 16$. When the number of columns of the Q matrix will increase from time to time algorithm will have to create new groups of kernels by increasing $\nu = 2\nu$. This scheme of increase prevents the increase in complexity due to the necessary matrix manipulations (increase = new memory allocations and copying). More precisely, when the number of columns of the Q matrix reaches half the number of kernels, new kernels will be added to the G matrix.

The algorithm of the Q matrix calculation with the proposed changes can be found in Algorithm 1. It can be seen here that changes have been made mainly in two places related to generating a random combination of columns of the G matrix, but also to the fact that the data matrix X is the input for this algorithm. The original matrix X is necessary for the construction of kernels. And this, in turn, we see in separate Algorithm 2. Here the new kernels are periodically added to the G matrix and based on them, random combinations of columns of the current G matrix are generated. In this procedure, when Gaussian

Algorithm 1: New way of Q matrix construction directly from X

```
function constructQ($X$)
for $i = 1$ to $r$ do
    $\mathbf{h}^{(i)} = \text{newRandComb}(X)$
end for
$Q^{(0)} = []$, the $m \times 0$ matrix
$j = 0$
while $\left(\max_{k=1,\ldots,r} \|\mathbf{h}^{(j+k)}\|\right) > \epsilon/(10\sqrt{2/\pi})$ do
    $j = j + 1$
    $\mathbf{h}^{(j)} = (\mathbf{I} - Q^{(j-1)}(Q^{(j-1)})^T)\mathbf{h}^{(j)}$
    $q^{(j)} = \mathbf{h}^{(j)}/\|\mathbf{h}^{(j)}\|$
    $Q^{(j)} = [Q^{(j-1)} q^{(j)}]$
    $\mathbf{h} = \text{newRandComb}(X)$
    $\mathbf{h}^{(j+r)} = (\mathbf{I} - Q^{(j)}(Q^{(j)})^T)\,\mathbf{h}$
    for $i = (j+1), (j+2), \ldots, (j+r-1)$ do
        $\mathbf{h}^{(i)} = \mathbf{h}^{(i)} - q^{(j)}\langle q^{(j)}, \mathbf{h}^{(i)}\rangle$
    end for
end while
$Q = Q^{(j)}$
return $Q$
```

Algorithm 2: Random combination over current set of kernels

```
$\nu = 16$
$kCount = 0$
$z = 0$
$G = []$, the $m \times 0$ matrix
$Y = []$, the $m \times 0$ matrix

function newRandComb($X$)
if $G$ is empty matrix then
    $G =$ add a column of 1's
end if
if $kCount = 0 \ \vee \ z = kCount * 3/4$ then
    add $\nu$ next kernel columns to $G$, but not more than $m$ in total
    $\Omega =$ Gaussian random matrix (size: number of columns in $G \times \nu$)
    $Y = [Y \ \ G\Omega]$
    $kCount = kCount + \nu$
    $\nu = 2 * \nu$
end if
$z = z + 1$
return $z$'s column of $Y$
```

kernels are created, they are placed in random instances of the X data matrix (no instance is selected twice).

The algorithm working in this way may finish its work faster or slower depending on the data and performance of built kernels in the context of a given classification problem.

The final learning algorithm of a neural network with the above fast SVD algorithm is presented in Algorithm 3.

3 Empirical Analysis of Proposed Algorithm

In order to compare the above proposed algorithm, it was decided to select a number of different data sets from the UCI Machine Learning Repository [6]. The number of instances of selected datasets are summarized in Table 2. In all tests, we used 10-fold stratified cross-validation and all learning machines were trained on the same sets of data partitions.

Algorithm 3: Neural network learning with SVD

```
function networkLearnig ($X$, $\mathbf{y}$)
$Q$ = constructQ($X$)
$[\tilde{U}, \Sigma, V^T]$ = SVD($Q^T G$)
$\mathbf{w} = V\Sigma^{-1}\tilde{U}^T Q^T \mathbf{y}$
return $\mathbf{w}$
```

To visualize the performance of all algorithms we present average accuracy for each benchmark dataset and for each learning machine. Additionally, we present the average reduction of dataset size in separate tables. *Ranks* are calculated for each machine for a given dataset. The ranks are calculated as follows: First, for a given benchmark dataset the averaged accuracies of all learning machines are sorted in descending order. The machine with the highest average accuracy is ranked 1. Then, the following machines in the accuracy order whose accuracies are not statistically different[1] from the result of the first machine are ranked 1, until a machine with a statistically different result is encountered. That machine starts the next rank group (2, 3, and so on), and an analogous process is repeated on the remaining (yet unranked) machines. Notice that each cell of the main part of Table 1 is in a form: $acc+std(rank)$, where acc is average accuracy (for a given data set and given learning machine), std is its standard deviation and $rank$ is the rank described just above. If a given cell of the table is in bold it means that this result is the best for given data set or not worse than the best one (rank 1 = winners).

Table 1 compares the average accuracies of the classification of the proposed algorithm that automatically determines kernels during modified SVD method with standard training of the neural network through SVD from 20, 200 and 2000 random Gaussian kernels, respectively.

As you can see, the proposed method is not always the best one, but always it achieves results close to the best without any manual selection of the number of kernels. It is worth comparing this data with the data from Table 2. This allows us to see how different the number of columns of the Q matrix was needed to achieve convergence and this has a direct impact on the working time of the entire learning process.

Even for a large data set like shuttle-all, the averaged number of columns in the Q matrix is only 48 out of 58,000 instances. This clearly shows that the size of the data set is not responsible for the number of necessary kernels, and the proposed method deals with it in an extremely interesting way. This is much simpler computationally than selection by many trials and testing, like via cross-validation. However, the proper selection of the number of kernels needs to start cross-validation at several points, which makes the process several times more time-consuming. Time consumption grows with the square of l in the complexity of SVD and $O(mlk)$ complexity of Q matrix construction. In conclusion, testing huge numbers of kernels in cross-validation provides huge costs. However, the

[1] We use the paired t-test to test the significance of statistical differences.

Table 1. Comparison of classification average accuracies for methods: auto-kernel SVD, SVD with 20, 200 and 2000 kernels.

Dataset	Auto-kernel	SVD 20	SVD 200	SVD 2000
cardiotocography-1	**83.1 ± 2.3(1)**	67.9 ± 2.8(3)	81.2 ± 2.2(2)	**83.3 ± 2.3(1)**
cardiotocography-2	92.4 ± 1.7(2)	87.5 ± 2.1(4)	91.6 ± 1.7(3)	**92.6 ± 1.8(1)**
chess-king-rook-vs-king-pawn	99.3 ± 0.54(2)	84.5 ± 4.5(4)	97.2 ± 1.1(3)	**99.4 ± 0.45(1)**
spambase	91.9 ± 1.2(3)	87.4 ± 1.8(4)	92.4 ± 1.3(2)	**92.7 ± 1.2(1)**
thyroid-disease	94.5 ± 0.89(3)	93.8 ± 0.28(4)	94.7 ± 0.5(2)	**95.4 ± 0.51(1)**
abalone	26.2 ± 1.9(2)	25.7 ± 2.1(3)	**26.3 ± 1.9(1)**	**26.4 ± 1.8(1)**
image	**94.4 ± 1.5(1)**	89.1 ± 2(3)	**94.5 ± 1.5(1)**	94.2 ± 1.6(2)
letter-recognition	**87.9 ± 0.76(1)**	58.4 ± 1.6(3)	82.5 ± 0.94(2)	**87.9 ± 0.75(1)**
magic04	86.1 ± 0.68(2)	82.4 ± 0.93(4)	86 ± 0.69(3)	**86.3 ± 0.66(1)**
musk2	**100 ± 0.04(1)**	85.3 ± 0.55(4)	89.9 ± 1(3)	99.4 ± 0.37(2)
nursery	**95.9 ± 0.43(1)**	87.2 ± 0.92(3)	93.8 ± 0.58(2)	**95.9 ± 0.49(1)**
sat-all	**90.1 ± 1(1)**	83.8 ± 1.1(4)	88.5 ± 1.3(3)	90 ± 1.1(2)
segmentation-all	**94.4 ± 1.5(1)**	89.1 ± 2(3)	**94.5 ± 1.5(1)**	94.2 ± 1.6(2)
SHUTTLE-all	98.2 ± 0.22(3)	96.9 ± 0.71(4)	98.4 ± 0.17(2)	**98.5 ± 0.17(1)**
Waveform	84.3 ± 1.5(2)	83.7 ± 1.7(3)	**86.4 ± 1.4(1)**	84.3 ± 1.6(2)
Mean Accuracy	87.9 ± 1.1	80.2 ± 1.7	86.5 ± 1.2	88 ± 1.1
Mean Rank	1.73 ± 0.21	3.53 ± 0.14	2.07 ± 0.21	1.33 ± 0.13
Wins[unique]	7[2]	0[0]	4[1]	10[6]

Table 2. Reduction strength—the number of columns in matrix Q.

Dataset	# Instances	Reduction		Time	
	in dataset	Total	Fraction	Auto-kernel	CV-kernel
cardiotocography-1	2126	640	0.334	2.14	86.6
cardiotocography-2	2126	637	0.333	2.46	83.8
chess-king-rook-vs-king-pawn	3196	2.3E+03	0.799	51.4	143
spambase	4601	2.98E+03	0.719	103	151
thyroid-disease	7200	162	0.0123	2	187
abalone	4177	46.4	0.0123	1.43	189
image	2310	137	0.0659	1.68	111
letter-recognition	20000	1.21E+03	0.067	45.7	996
magic04	19020	206	0.012	7.95	1.03E+03
musk2	6598	5.83E+03	0.981	888	609
nursery	12960	645	0.0553	19.5	849
sat-all	6435	2.19E+03	0.378	99.2	720
segmentation-all	2310	137	0.0659	15	415
SHUTTLE-all	58000	48	0.00092	14.8	4.58E+03
Waveform	5000	3.51E+03	0.779	259	1.35E+03

proposed algorithm immediately determines the correct number of kernels. This was proven by the next experiment (see Table 2 columns Reduction), which compares the CPU times of the proposed Auto-kernel algorithm and the SVD-based learning with cross-validation for the selection of kernel count. In the second algorithm, cross-validation is used to select kernel counts just between three values: 20, 200, and 2000. In many benchmarks, the Auto-kernel algorithm is several times faster than learning with CV-based selection of kernels, even in a simple three point selection. If we added 20,000 kernels to CV training as an additional optionpoint, learning would become terribly slower comparing to the Auto-kernel.

4 Summary

The process of selection of kernels for neural networks was never simple. The methods that were based on finding the right number of kernels required many learning processes and tests of the learned neural networks. Such a scheme of learning strongly uses the CPU. In the proposed algorithm, we modify a fast SVD method by automatically generating kernels and their appropriate number in the learning process, specifically in the sub-process of the fast SVD.

As can be seen, such learning is characterized by good classification quality for various data sets. What's more, the selection of the number of columns in the Q matrix is quite impressive, which translates into shortening the learning process if the classifier does not have to use many kernels.

References

1. Broomhead, D.S., Lowe, D.: Multivariable functional interpolation and adaptive networks. Complex Syst. **2**(3), 321–355 (1988)
2. Dudek, G.: A constructive approach to data-driven randomized learning for feedforward neural networks. Appl. Soft Comput. **112** (2021). https://doi.org/10.1016/j.asoc.2021.107797
3. Halko, N., Martinsson, P.G., Tropp, J.A.: Finding structure with randomness: probabilistic algorithms for constructing approximate matrix decompositions. SIAM Rev. **53**(2), 217–288 (2011)
4. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: a new learning scheme of feedforward neural networks. In: International Joint Conference on Neural Networks, Budapest, Hungary, pp. 985–990. IEEE Press (2004)
5. Huang, G.B., Zhu, Q.Y., Siew, C.K.: Extreme learning machine: theory and applications. Neurocomputing **70**(1–3), 489–501 (2006)
6. Merz, C.J., Murphy, P.M.: UCI repository of machine learning databases (1998). http://www.ics.uci.edu/~mlearn/MLRepository.html
7. Tang, J., Deng, C., Member, S., Huang, G.B.: Extreme learning machine for multilayer perceptron. IEEE Trans. Neural Netw. Learn. Syst. **27**(4), 809–821 (2016)
8. Vapnik, V.: The Nature of Statistical Learning Theory. Springer, New York (1995)