Int. J. Appl. Math. Comput. Sci., 2019, Vol. 29, No. 3, 581–594
DOI: 10.2478/amcs-2019-0043

A FAST NEURAL NETWORK LEARNING ALGORITHM WITH APPROXIMATE SINGULAR VALUE DECOMPOSITION

NORBERT JANKOWSKI [a,*], RAFAŁ LINOWIECKI [a]

[a]Department of Informatics, Faculty of Physics, Astronomy and Informatics
Nicolaus Copernicus University, ul. Grudziądzka 5, 87-100 Toruń, Poland
email: norbert@is.umk.pl

The learning of neural networks is becoming more and more important. Researchers have constructed dozens of learning algorithms, but it is still necessary to develop faster, more flexible, or more accurate learning algorithms. With fast learning we can examine more learning scenarios for a given problem, especially in the case of meta-learning. In this article we focus on the construction of a much faster learning algorithm and its modifications, especially for nonlinear versions of neural networks. The main idea of this algorithm lies in the usage of fast approximation of the Moore–Penrose pseudo-inverse matrix. The complexity of the original singular value decomposition algorithm is $O(mn^2)$. We consider algorithms with a complexity of $O(mnl)$, where $l < n$ and l is often significantly smaller than n. Such learning algorithms can be applied to the learning of radial basis function networks, extreme learning machines or deep ELMs, principal component analysis or even missing data imputation.

Keywords: Moore–Penrose pseudo-inverse learning, radial basis function network, extreme learning machines, kernel methods, machine learning, singular value decomposition, deep extreme learning, principal component analysis.

1. Introduction

Singular value decomposition (SVD) is one of the major algorithms used frequently in neural network and machine learning as an element of more complicated algorithms. Let us assume we have a matrix A, an $(m \times n)$ one ($A \in \mathcal{M}_{m\times n}(\mathbb{R})$). Then the goal of SVD is to decompose this to

$$A = U \times \Sigma \times V^T, \tag{1}$$

where U and V^T are orthogonal matrices, and Σ is a diagonal matrix with singular values. In a more practical version, $U \in \mathcal{M}_{m\times n}(\mathbb{R})$, Σ and $V^T \in \mathcal{M}_{n\times n}(\mathbb{R})$. In the second case, $U \in \mathcal{M}_{m\times m}(\mathbb{R})$, $\Sigma \in \mathcal{M}_{m\times n}(\mathbb{R})$ and $V^T \in \mathcal{M}_{n\times n}(\mathbb{R})$. This case is not very convenient because, when m is huge, the matrix U may not fit in memory, while in the first case, if only n is not huge either, there is no problem with keeping all the matrices in memory.

SVD in artificial neural networks (ANNs) and machine learning (ML) is used in unsupervised and supervised learning. In the former case, it is employed as a very efficient tool for principal component analysis (PCA). PCA can be used to extract the most important features from data (dimensionality reduction). Another known application of SVD in unsupervised learning is latent semantic analysis (Dumais, 2005) or face recognition (Heseltine *et al.*, 2003).

In the case of supervised learning, SVD is used to obtain solutions of linear and nonlinear discrimination and regression problems by building the pseudo-inverse just from the results of SVD. For a deeper investigation of Moore–Penrose pseudo-inverse learning, see the work of Górecki and Łuczak (2013). Whenever we consider classification or approximation problems, they are represented by a training dataset $\mathcal{D}$, which consists of learning vectors $\mathbf{x}_i$ ($\mathbf{x}_i \in \mathbb{R}^n, i \in [1,\ldots,m]$) with the corresponding labels y_i ($\mathbf{y} = [y_1,\ldots,y_m]$). In the case of classification $y_i = 1,\ldots,k$ (in the binary case, $y_i = \pm 1$), and in the case of regression $y_i \in \mathbb{R}$. Then we expect our model $F(\mathbf{x},\mathbf{w})$ to satisfy

$$F(\mathbf{x}_i,\mathbf{w}) \approx y_i, \qquad i \in [1,\ldots,m], \tag{2}$$

and to be as close as possible to the Bayesian optimal classifier (Mitchell, 1997) in the context of classification and, similarly, in the case of regression. This means

[*]Corresponding author

that we want to estimate a hidden relation between input $(\mathbf{x}_1, \dots, \mathbf{x}_m)$ and output $(\mathbf{y})$ in $\mathcal{D}$.

If we consider a linear model, it can be defined as

$$F(\mathbf{x}; \mathbf{w}) = \sum_{j=1}^{n} w_j x_j + w_0, \tag{3}$$

where $\mathbf{w}$ are parameters of a hyperplane.[1] In the case of nonlinear models, we can look at a nonlinear model constructed as a linear combination of nonlinear functions

$$F(\mathbf{x}; \mathbf{w}) = \sum_{j=1}^{l} w_j g_j(\mathbf{x}) + w_0, \tag{4}$$

which is a combination $(\mathbf{w})$ of kernels g_j. The above form is fully consistent with the radial basis function network (RBFN) (Broomhead and Lowe, 1988) and the extreme learning machine (ELM) (Huang *et al.*, 2004; 2006) as well. The same form also applies to nonlinear support vector machines (Vapnik, 1995; Boser *et al.*, 1992) (except for the learning algorithm). Additionally, it is a special case of a multilayer perceptron (Rumelhart *et al.*, 1986) with one hidden layer, one linear neuron in the output layer and fixed weights between the first and the second layer, but the MLP is typically trained by back propagation and usually no weights are fixed.

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 the ELM as well (Huang *et al.*, 2006)). The sigmoidal kernels in ELMs are constructed by randomizing their weights and thresholds. In the case of Gaussian kernels, they can be initialized by a subset of vectors of the training data $\mathcal{D}$. In both cases the number of kernels has to be chosen manually. Automatic kernel selection (for the ELM or RBFN) was described by Jankowski (2018).

Now we can define the goal (the error function) of linear learning by

$$\begin{aligned} J_l(\mathbf{w}, X) &= ||X\mathbf{w} - \mathbf{y}||^2 \\ &= \sum_{j=1}^{m} (\mathbf{w}^T \mathbf{x}_j + w_0 - y_i)^2, \end{aligned} \tag{5}$$

where X is defined by

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & x_{2n} \\ \dots & \dots & \dots & \dots & \dots \\ 1 & x_{m1} & x_{m2} & \cdots & x_{mn} \end{bmatrix}, \tag{6}$$

and in nonlinear learning, the goal is defined by

$$\begin{aligned} J_n(\mathbf{w}, G) &= ||G\mathbf{w} - \mathbf{y}||^2 \\ &= \sum_{i=1}^{m} \Big(\sum_{j=1}^{l} w_j g_j(\mathbf{x}_i) + w_0 - y_i \Big)^2, \end{aligned}$$

where G is given as

$$G = \begin{bmatrix} 1 & g_1(\mathbf{x}_1) & g_2(\mathbf{x}_1) & \cdots & g_l(\mathbf{x}_1) \\ 1 & g_1(\mathbf{x}_2) & g_2(\mathbf{x}_2) & \cdots & g_l(\mathbf{x}_2) \\ \dots & \dots & \dots & \dots & \dots \\ 1 & g_1(\mathbf{x}_m) & g_2(\mathbf{x}_m) & \cdots & g_l(\mathbf{x}_m) \end{bmatrix}. \tag{7}$$

Note that Eqn. (7) simply uses the matrix G in place of the matrix X in Eqn. (5). For simplicity, we can assume the general form of the learning goal J,

$$J(\mathbf{w}, A) = ||A\mathbf{w} - \mathbf{y}||^2, \tag{8}$$

which covers both the linear and the nonlinear case.

To find the solution—the minimum of Eqn. (8)—we have to compute the gradient:

$$\nabla J(\mathbf{w}, A) = 2A^T(A\mathbf{w} - \mathbf{y}). \tag{9}$$

To find $\mathbf{w}$, we equate the gradient to zero:

$$\nabla J(\mathbf{w}, A) = 0. \tag{10}$$

After a few substitutions, we finally have

$$\mathbf{w} = (A^T A)^{-1} A^T \mathbf{y} = A^{\dagger} \mathbf{y}, \tag{11}$$

where $A^{\dagger}$ is the Moore–Penrose pseudo-inverse matrix of A. Now, SVD can be used to compute the pseudo-inverse of A. In this way, SVD can be employed to build both linear and nonlinear discrimination/regression. In the case of nonlinear discrimination, we have solutions for RBFNs and ELMs. Finally, $A^{\dagger}$ is

$$A^{\dagger} = V \times \Sigma^{-1} \times U^T, \tag{12}$$

where Σ^{-1} is a matrix with the dimensions of Σ swapped symmetrically, and all of its non-zero elements are on its diagonal. $\Sigma^{-1}_{ii} = 1/\Sigma_{ii}$ for each i, where $\Sigma_{ii} \neq 0$.

The costs of learning via SVD are relatively low. The complexity of SVD in linear cases is $O(mn^2)$ and $O(ml^2)$ in nonlinear cases (the difference lies in the sizes of the matrices X and G, respectively). The goal of this article is to reduce the costs stemming from the number of columns of the decomposed matrix (n^2 and l^2 in the above complexities, respectively).

The usage of SVD for learning RBFNs or ELMs does not constitute the whole of pseudo-inverse learning applicability in ANNs. Tang *et al.* (2016) introduced a multilayer perceptron learned as a multi-layered ELM. It can be seen as a special case of deep learning—the first layers are the layers of autoencoders and the final layer is a typical ELM. This kind of learning enables a significant shift of quality in comparison with the abilities of single hidden layer ELMs, especially in computer vision problems. Another application is local receptive fields-based learning in the domain of ELMs, introduced by Huang *et al.* (2015). This is also an example of

[1] In the case of more than two-class classification, one model per class should be constructed.

a deep structure which is more similar to convolutional neural networks (CNNs) (Goodfellow *et al*., 2016)—it consists of several random feature (sub-)layers and the corresponding pooling maps which finally compose an input to the standard extreme learning layer. RBFNs and ELMs are also used in meta-learning (Jankowski, 2013) as an elementary learning algorithm for searching in wide model spaces. Additionally, pseudo-inverses and ELMs can be used in several other applications, one of the more interesting and somewhat non-typical usages dealing with missing data (Eirola *et al*., 2014; Sovilj *et al*., 2016).

The computational costs of SVD are crucial for the efficiency of the above ANN. What is more, in the search for an appropriate ANN architecture, we test multiple configurations (different layers, different parameters of neurons, etc.), and the learning of each configuration consumes its own amount of time. This is why reducing SVD costs is very important, as it provides a useful speedup in each learning process with little or no tradeoff in accuracy.

The next section describes the idea of fast approximate SVD computation and lays out a proposition of a modification which additionally reduces the computational costs of SVD. The following section presents several numerical analyses which compare normal SVD with fast versions in terms of accuracy and execution time.

2. Analysis and modification of fast approximate singular value decomposition

In the work by Halko *et al.* (2011) we can find a review of a few attempts at reducing the cost of matrix decomposition by randomization techniques. Most of them use a low-rank approximation of a matrix. In the first part of this section we will focus on the general idea, and then on one algorithm proposed by Halko *et al.* (2011). Finally, we will analyze its convergence and propose a modification of the algorithm to reduce the computational costs.

Generally, the idea is based on intermediate matrix decomposition which has to produce a matrix of a significantly smaller size, and after that SVD will be used to decompose that smaller matrix with (possibly) much lower computational costs. Such a decomposition may be strongly dependent on the rank of the decomposed matrix A. It is especially helpful when the rank of A is smaller than its size, or if there exists a low-rank approximation of A.

The low-rank approximation of a given matrix A is defined by

$$\underset{m \times n}{A} \quad \approx \quad \underset{m \times k}{C} \quad \times \quad \underset{k \times n,}{B,} \tag{13}$$

where k bounds the rank of the above decomposition.

Assuming that $m > n$, SVD decomposition of the matrix B is computationally simpler than that of A. However, these are not sufficient conditions for efficient construction of the pseudo-inverse matrix of A without additional restrictions on the C matrix. When C is an orthonormal matrix, the computation of the pseudo-inverse of the above decomposition is simple.

That is why the following matrix Q with orthonormal columns is so useful:

$$A \approx QQ^T A. \tag{14}$$

Now, to compute the SVD of A, we have to

- construct the matrix Q,
- construct the matrix B such that $B = Q^T A$ ($B \in \mathcal{M}_{k\times n}(\mathbb{R})$),
- compute $SVD(B) = \tilde{U}\Sigma V^T$, and
- define the final matrix U as $U = Q\tilde{U}$.

The proposed construction of the matrix Q was performed by Gaussian randomization and orthogonalization. First, let the matrix H be defined by

$$\mathbf{h}^{(i)} = A\boldsymbol{\omega}^{(i)}, \quad i = 1, \dots, k, \tag{15}$$

where $\boldsymbol{\omega}^{(i)}$ is a random Gaussian vector $1 \times n$ and $H \in \mathcal{M}_{m\times k}(\mathbb{R})$.

The next step is to build an orthonormal matrix Q from the matrix H using an orthonormalization algorithm. The two steps, the randomization and the orthonormalization, can be performed as in Algorithm 1.

Now the columns of the matrix Q form an orthonormal basis of the range of H.

The selection of k as the number of columns in the above algorithm is very important. This is because we expect the error

$$||A - QQ^T A||, \tag{16}$$

stemming from the decomposition, to be small. The choice of k below the rank of A can lead to a poor decomposition of A.

However, thanks to the randomness of the vectors $\boldsymbol{\omega}^{(i)}$, the set of $\boldsymbol{\omega}^{(i)}$ is likely to be linearly independent. As presented by Halko *et al.* (2011), the following lemma holds.

Algorithm 1. Orthogonal randomized matrix.

```
for $i = 1$ to $k$ do
    $\boldsymbol{\omega}^{(i)}$ = Gaussian random vector $(1 \times n)$
    $\mathbf{h}^{(i)} = A\boldsymbol{\omega}^{(i)}$
    $\tilde{\mathbf{q}}^{(i)} = (I - QQ^T)\mathbf{h}^{(i)}$
    $\mathbf{q}^{(i)} = \tilde{\mathbf{q}}^{(i)}/||\tilde{\mathbf{q}}^{(i)}||$
    $Q = [Q\ \mathbf{q}^{(i)}]$
end for
```

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

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

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:

$$\begin{aligned}&||(I-QQ^T)A||\\ &\qquad \leq 10\sqrt{\frac{2}{\pi}}\max_{i=1,\dots,r}||(I-QQ^T)A\boldsymbol{\omega}^{(i)}||, \quad (17)\end{aligned}$$

with a probability of at least $1-10^{-r}$.

In further parts of the article, by the decomposition error we will refer to

$$10\sqrt{\frac{2}{\pi}}\max_{i=1,\dots,r}||(I-QQ^T)A\boldsymbol{\omega}^{(i)}||. \qquad (18)$$

Halko *et al.* (2011) present the final algorithm for construction of matrix Q as a combination of the previous orthonormalization steps in connection with the above property to control the decomposition error incrementally (see Algorithm 2). Thanks to this, the number of columns in the matrix Q is incrementally adjusted according to the decreasing decomposition error in low costs.

In all experiments ϵ was set to 0.1 and $r = 10$; ϵ defines the accuracy which limits the error in Eqn. (18), r controls the reliability of the test defined by Lemma 1 (larger r means stronger reliability and slightly more costly computations of the test). The chosen values of r and ϵ give sufficient accuracy and reliability of the test. The Gaussian random vector is drawn from a Gaussian distribution with zero mean and unit variance.

The complexity of the above algorithm is $O(mnk)$, where $m \times n$ is the size of A and k is the number of columns in Q. After the construction of Q, SVD is calculated on the matrix $B = Q^TA$ and the matrix $B \in \mathcal{M}_{k\times n}(\mathbb{R})$. This means that the complexity of SVD on B is $O(nk^2)$. This leads us to the final complexity of the fast version of SVD: $O(mnk)$. Two plots in Fig. 1 show examples of convergence of the decomposition error (Eqn. (18)) for two data sets from the UCI Machine Learning Repository, `cardiotocography-1` and `musk2`. The same decomposition error is used in further figures as well. In the first step the original data sets were standardized, and after that transformed (with a Gaussian kernel) to 2000-dimensional kernel spaces according to Eqn. (7). Such matrices were used to construct matrices Q, in accordance with Algorithm 2.

We can see two different cases: the first example stops after 788 iterations, while in the second case the algorithm stops after 2000 iterations. In the first case, the rank of the data matrix of `cardiotocography-1` is smaller than 2000, and the matrix could be approximated by a matrix of rank 788. In the second case the rank of the data matrix was huge and the algorithm constructing Q was not able to reduce its size.

Additionally, we can see some interesting behavior on another example: in this case we add 1% of noise to `cardiotocography-1`, after the transformation to the 2000-dimensional kernel space. The results are presented in Fig. 2. The expression 'ν% of noise' means that we added Gaussian noise with zero mean and the standard deviation equal to $\nu/100$ to the data in the interval $[0,1]$ (data constructed by Gaussian kernels). Now, the noise added to the transformed data significantly impacted the convergence and finally we have no reduction in the size of the matrix Q, even with such a small amount of noise.

Such behaviour is not optimistic from the perspective of machine learning and artificial neural networks. Let us remember that adding an appropriate amount of noise is equivalent to the Tikhonov regularization (Tikhonov and Arsenin, 1977) learning in neural networks (Bishop, 1991). It would, of course, be better if the addition of noise did not affect the convergence so strongly. But we can observe that there is a strong difference between the convergence progressions of `cardiotocography-1` with noise (Fig. 2) and `musk2` (Fig. 1). In the case of `cardiotocography-1`, due to the addition of a small amount of noise, after initially rapid convergence, we can see that the shape becomes very flat, especially compared

Algorithm 2. Forming matrix Q.

```
function constructQ(A)
for $i = 1$ to $r$ do
    $\boldsymbol{\omega}^{(i)}$ = Gaussian random vector $(1 \times n)$
    $\mathbf{h}^{(i)} = A\boldsymbol{\omega}^{(i)}$
end for
$Q^{(0)} = []$, the $m \times 0$ matrix
$j = 0$
while $\left(\max_{k=1,\dots,r}\|\mathbf{h}^{(j+k)}\|\right) > \epsilon/(10\sqrt{2/\pi})$ do
    $j = j + 1$
    $\mathbf{h}^{(j)} = (\mathbf{I} - Q^{(j-1)}(Q^{(j-1)})^T)\mathbf{h}^{(j)}$
    $q^{(j)} = \mathbf{h}^{(j)}/\|\mathbf{h}^{(j)}\|$
    $Q^{(j)} = [Q^{(j-1)}q^{(j)}]$
    $\boldsymbol{\omega}^{(j+r)}$ = Gaussian random vector $(1 \times n)$
    $\mathbf{h}^{(j+r)} = (\mathbf{I} - Q^{(j)}(Q^{(j)})^T)A\boldsymbol{\omega}^{(j+r)}$
    for $i = (j+1), (j+2), \dots, (j+r-1)$ do
        $\mathbf{h}^{(i)} = \mathbf{h}^{(i)} - q^{(j)}\langle q^{(j)}, \mathbf{h}^{(i)}\rangle$
    end for
end while
$Q = Q^{(j)}$
```

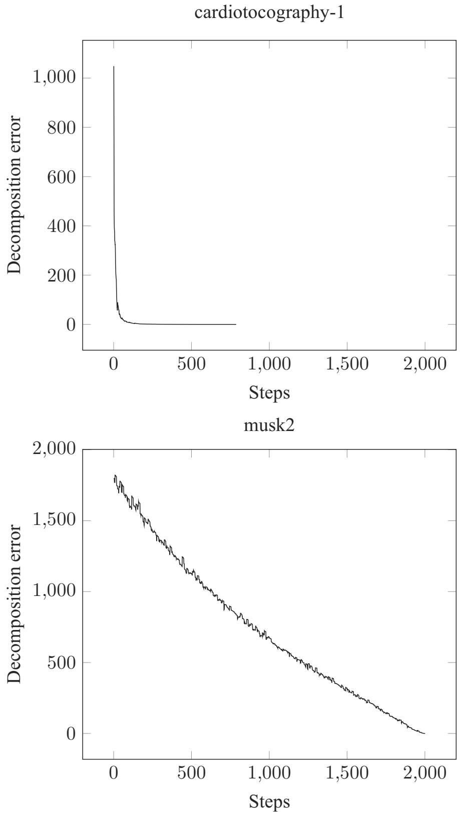

Fig. 1. Examples of convergence of the decomposition error during the construction of the matrix Q. Those plots were obtained for benchmarks from UCI: `cardiotocography-1` and `musk2`.

with the convergence progression of `musk2`. This clearly suggests the existence of two separate causes for those different instances of convergence behaviour.

Smoothed gradient stop criterion in the algorithm of constructing matrix Q. Based on the above experience, we decided to introduce another type of stopping criterion for the algorithm of forming matrix Q.

The previous stopping criterion was based on the lemma presented in the earlier part of this section, which extracts the maximal norm of the last few vectors $\mathbf{h}_i$; see Eqn. (18). Our idea was to add an additional condition which tests whether the subsequent norms are not overly flat. However, this is not as easy as just testing the gradient, as demonstrated by several specific incidents which may occur from time to time, depending on the dataset; see Fig. 3. Sometimes we may find several points where the convergence is not monotonic and, in consequence, the gradient flips in opposite directions. Hopefully, such *flips* exhibit very local behaviour and to overcome this problem we propose the criterion as presented Algorithm 3.

The function presented in Algorithm 3 is a smoothed version of the gradient of the decomposition error. It rejects the flips the of gradient (line 9 of the code) and analyzes the average of maximal norms over the last r vectors $\mathbf{h}_i$. Here τ defines the minimal decrease of decomposition error needed to continue the construction of matrix Q; τ was set manually to 0.02 and this value was used in all tests. After the introduction of the new stopping criterion, the computation of matrix Q slightly changes; compare this with Algorithm 4. It is important to note that the new stopping criterion does not influence the complexity, as it relies only on values that have already been computed.

The above version of constructing matrix Q has the same complexity as the previous one, i.e., $O(mnk)$. However, as is shown in next section, k is very often significantly smaller than in the previous version.

The results obtained with the new stop criterion can be seen in Figs. 4–7 with 0%, 1%, 4% and 32% of noise, respectively (on the `cardiotocography-1` dataset again). Comparing those figures with the previous ones, it is easy to observe that the adoption of the new criterion leads to a significant reduction in the size of the final

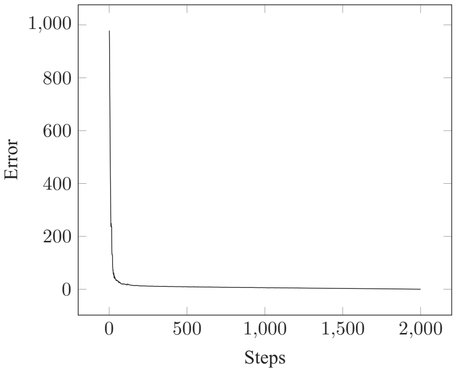

Fig. 2. Example of convergence of the process of forming matrix Q for the `cardiotocography-1` dataset with 1% of noise.

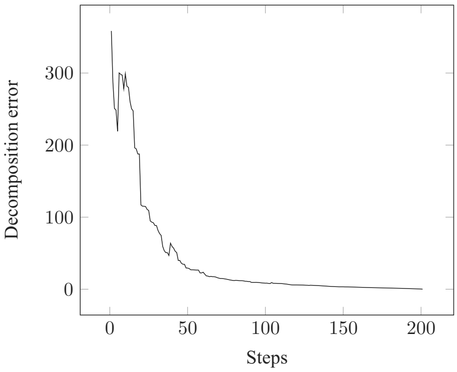

Fig. 3. Non-monotonic convergence of the decomposition error.

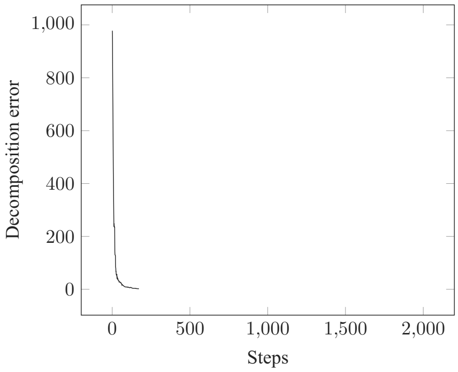

Fig. 4. Convergence with the gradient stopping criterion: 0% of noise.

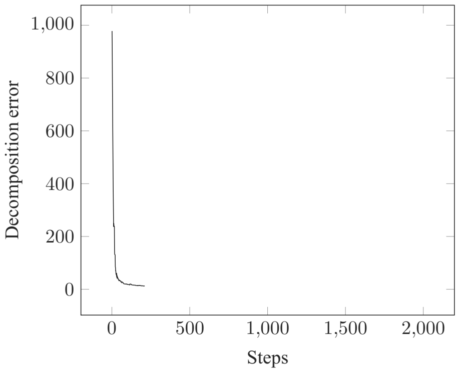

Fig. 5. Convergence with the gradient stopping criterion: 1% of noise.

matrix Q. Table 1 presents the number of columns in the matrix Q depending on the percentage of noise.

The final learning algorithm of a neural network or linear discrimination with the above fast SVD algorithm is presented as Algorithm 5.

In the next section, a comparative analysis of three versions of SVD is presented: the classic and the fast one, and that with the gradient stop.

3. Results analysis of pseudo-inverse learning

Three versions of SVD are compared in this section: normal SVD, fast approximate SVD, and its version with the gradient stop as an element of neural network learning. All experiments were performed on a typical laptop with an Intel i7-4550U processor, 1.5 GHz/2.10 GHz, with 8 GB RAM (DDR3 1.33 GHz).

Table 1. Relation between the noise level and the number of columns in matrix Q.

noise %	0	1	2	4	8	16	32
# columns	172	211	165	227	233	638	878

The tests concentrated mainly on three aspects:

- convergence analysis of fast SVD and its modified version,
- comparison of execution time of SVD and fast versions of SVD, and
- classification accuracy (because we are interested in applications of SVD in neural networks and machine learning).

In Table 2 we summarize the most important information about selected datasets from the UCI Machine Learning Repository (Merz and Murphy, 1998) used for testing.

All tests concern the learning of a neural network equivalent with the RBFN or the ELM with a Gaussian kernel. The tests were computed with second layer sizes of 100, 200, 1000 or 2000 kernels, depending on the test

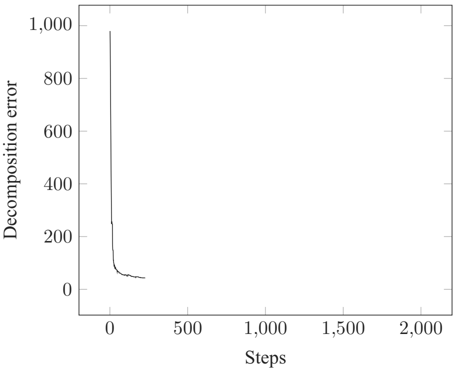

Fig. 6. Convergence with the gradient stopping criterion: 4% of noise.

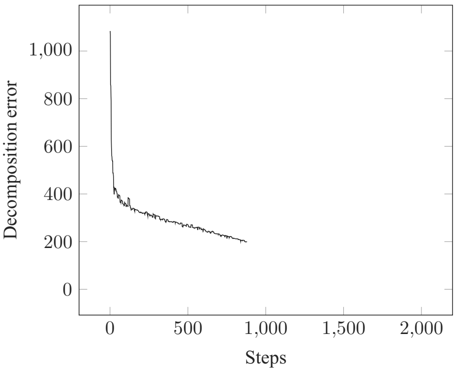

Fig. 7. Convergence with the gradient stopping criterion: 32% of noise.

Table 2. Information on the datasets used for tests.

Dataset	# classes	# instances	# features	# ordered
cardiotocography-1	10	2126	21	21
cardiotocography-2	3	2126	21	21
chess-king-rook-vs-king-pawn	2	3196	38	38
spambase	2	4601	57	57
thyroid-disease	3	7200	21	21
abalone	29	4177	8	7
image	7	2310	19	19
letter-recognition	26	20000	16	16
magic04	2	19020	10	10
musk2	2	6598	168	166
nursery	6	12960	8	0
sat-all	6	6435	36	36
segmentation-all	7	2310	19	19
shuttle-all	7	58000	9	9
waveform	3	5000	21	21

Algorithm 3. Gradient stop criterion.

```
$i = 0$, lastMax $= 0$, $\tau$ = 0.02;
$\mathbf{gradT} = []$, the $r$ elements empty vector

function GradientDesc($\mathbf{h}^{(j+1)}, \ldots, \mathbf{h}^{(j+r)}$)
$m = \max_{i \in [j+1,\ldots,j+r]} \|\mathbf{h}^{(i)}\|$
**if** $m \leq \epsilon/(10\sqrt{2/\pi})$ **then**
    return false
**end if**
$\mathbf{gradT}$[i % r] = $\max\{0, \text{lastMax} - m\}$
$i = i + 1$
lastMax $= m$
**if** $i \leq r$ **then**
    return true;
**else**
    return Average($\mathbf{gradT}$) > $\tau$;
**end if**
```

Algorithm 4. Fast SVD.

```
function constructQG (A)
**for** $i = 1$ to $r$ **do**
    $\boldsymbol{\omega}^{(i)}$ = Gaussian random vector $(1 \times n)$
    $\mathbf{h}^{(i)} = A\boldsymbol{\omega}^{(i)}$
**end for**
$Q^{(0)} = []$, the $m \times 0$ matrix
$j = 0$
$grad = true$
**while** grad **do**
    $j = j + 1$
    $\mathbf{h}^{(j)}$ = $(\mathbf{I} - Q^{(j-1)}(Q^{(j-1)})^T)\mathbf{h}^{(j)}$
    $q^{(j)}$ = $\mathbf{h}^{(j)}/\|\mathbf{h}^{(j)}\|$
    $Q^{(j)} = [Q^{(j-1)}q^{(j)}]$
    $\boldsymbol{\omega}^{(j+r)}$ = Gaussian random vector $(1 \times n)$
    $\mathbf{h}^{(j+r)} = (\mathbf{I} - Q^{(j)}(Q^{(j)})^T)A\boldsymbol{\omega}^{(j+r)}$
    **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**
    $grad$ = GradientDesc($\mathbf{h}^{(j+1)}, \ldots, \mathbf{h}^{(j+r)}$)
**end while**
$Q = Q^{(j)}$
```

type, in accordance with the learning algorithm presented as Algorithm 5.

Convergence of fast SVD algorithms. Figures 8 and 9 present the plots of convergence for the selected datasets. Those figures present all types of behaviour in terms of convergence—this was the goal in the selection of the datasets. As before, the y-axis presents the decomposition error defined by Eqn. (18). The solid-dashed curve represents the convergence of the fast SVD algorithm. The solid part of those curves represents the fast SVD with the gradient stop algorithm. It can be easily seen that the new stop criterion stops much earlier than the fast SVD algorithm. Now the new criterion stops if the convergence curve becomes too flat.

In Tables 3 and 4, comparisons of proportions of the numbers of steps with the numbers of kernels in the input spaces[2] are shown. The fraction means the ratio of 1000 and 2000 or 100 and 200, respectively, to the initial number of kernels.

Comparison of the learning time of SVD, fast SVD and its version with the gradient stop. The learning time presented in the following tables was computed by Algorithm 5 from line 7 to 9, such that we ignore

[2] The numbers of kernels are equivalent to those of columns in the matrices A with the transformed datasets.

Table 3. Comparison of the fractions of steps of the fast SVD algorithm and of its version with the gradient stop for input spaces with 1000 and 2000 kernels.

	FastSVD-1000	GradFastSVD-1000	FastSVD-2000	GradFastSVD-2000
cardiotocography-1	0.601±0.0068	0.157±0.017	0.414±0.0035	0.0921±0.0091
cardiotocography-2	0.6±0.0072	0.155±0.016	0.414±0.0034	0.0928±0.0094
chess-king-rook-vs-king-pawn	0.99±0.002	0.327±0.048	0.878±0.0049	0.191±0.028
spambase	0.908±0.0075	0.352±0.046	0.837±0.0055	0.223±0.032
thyroid-disease	0.355±0.01	0.18±0.014	0.256±0.0055	0.108±0.0074
abalone	0.0701±0.0025	0.0433±0.0028	0.0382±0.0011	0.023±0.0014
image	0.219±0.0047	0.0986±0.0068	0.133±0.0022	0.0566±0.0043
letter-recognition	0.868±0.0072	0.196±0.018	0.616±0.0042	0.108±0.011
magic04	0.296±0.0061	0.114±0.0079	0.191±0.0037	0.0644±0.0056
musk2	1±0	1±0	1±0	1±0
nursery	0.617±0.0047	0.17±0.016	0.363±0.0027	0.0928±0.0068
sat-all	0.806±0.0054	0.147±0.018	0.616±0.0041	0.0828±0.0095
segmentation-all	0.219±0.0047	0.0986±0.0068	0.133±0.0022	0.0566±0.0043
SHUTTLE-all	0.0772±0.0028	0.0512±0.003	0.0439±0.0016	0.0284±0.0016
Waveform	1±0.00074	0.207±0.033	0.966±0.0014	0.119±0.02
Mean	0.575±0.0049	0.22±0.017	0.46±0.0031	0.156±0.01

Table 4. Comparison of fractions of steps of the fast SVD algorithm and of its version with the gradient stop for input spaces with 100 and 200 kernels.

	FastSVD-100	GradFastSVD-100	FastSVD-200	GradFastSVD-200
cardiotocography-1	1±0.0019	0.889±0.061	0.986±0.0051	0.535±0.042(4)
cardiotocography-2	1±0.00099	0.871±0.069	0.987±0.0056	0.532±0.044(4)
chess-king-rook-vs-king-pawn	1±0	1±0	1±0	0.976±0.07(2)
spambase	0.98±0.016	0.98±0.016	0.965±0.015	0.934±0.065
thyroid-disease	0.84±0.038	0.716±0.039	0.671±0.032	0.493±0.029
abalone	0.458±0.021	0.324±0.017	0.267±0.011	0.179±0.011
image	0.841±0.027	0.575±0.048	0.596±0.02	0.347±0.023
letter-recognition	1±0.0022	1±0.0022	0.999±0.0029	0.75±0.068
magic04	0.97±0.017	0.681±0.035	0.775±0.019	0.395±0.021
musk2	1±0	1±0	1±0	1±0
nursery	1±0	0.997±0.025	1±0	0.671±0.061
sat-all	1±0	0.898±0.087	1±0	0.528±0.049
segmentation-all	0.841±0.027	0.575±0.048	0.596±0.02	0.347±0.023
SHUTTLE-all	0.465±0.02	0.354±0.023	0.278±0.011	0.199±0.013
Waveform	1±0	1±0	1±0	0.768±0.1
Mean	0.893±0.011	0.791±0.031	0.808±0.0093	0.577±0.042

the kernel construction time. In the case of normal SVD, the code lines from 7 to 9 are substituted by only computing SVD on the matrix A as well as multiplication for computing its pseudo-inverse. Tables 5 and 6 present the fractions of the execution time of the normal SVD algorithm that were used by fast SVD or by fast SVD with the gradient stop. It can be seen that fast SVD with the gradient stop uses much less time than with the previous stop criterion. Only in case of the `musk2` dataset are the times similar, while in most cases the execution time is around 10 times smaller.

Comparison of classifier accuracies obtained with SVD, fast SVD and its version with the gradient stop. To present a trustworthy investigation of applicability of the fast versions of SVD algorithms, the accuracies of classifiers were tested in 10-fold stratified cross-validation repeated 10 times. The accuracies were averaged from the test parts only. Tables 7 and 8 present the classifiers' average accuracies. The columns correspond to different classifier configurations and rows correspond to datasets. The cells in boldface mean that a given classifier is a winner in the meaning of the paired t-test. The cells consist of averaged accuracy, accompanied by its standard deviation and the classifier's rank (in line with the pattern $accuracy \pm std(rank)$)).

The *ranks* are calculated for each machine for a given dataset $\mathcal{D}$. They are determined as follows. First, for a given benchmark dataset $\mathcal{D}$ the averaged accuracies of all learning machines are sorted in descending order. The

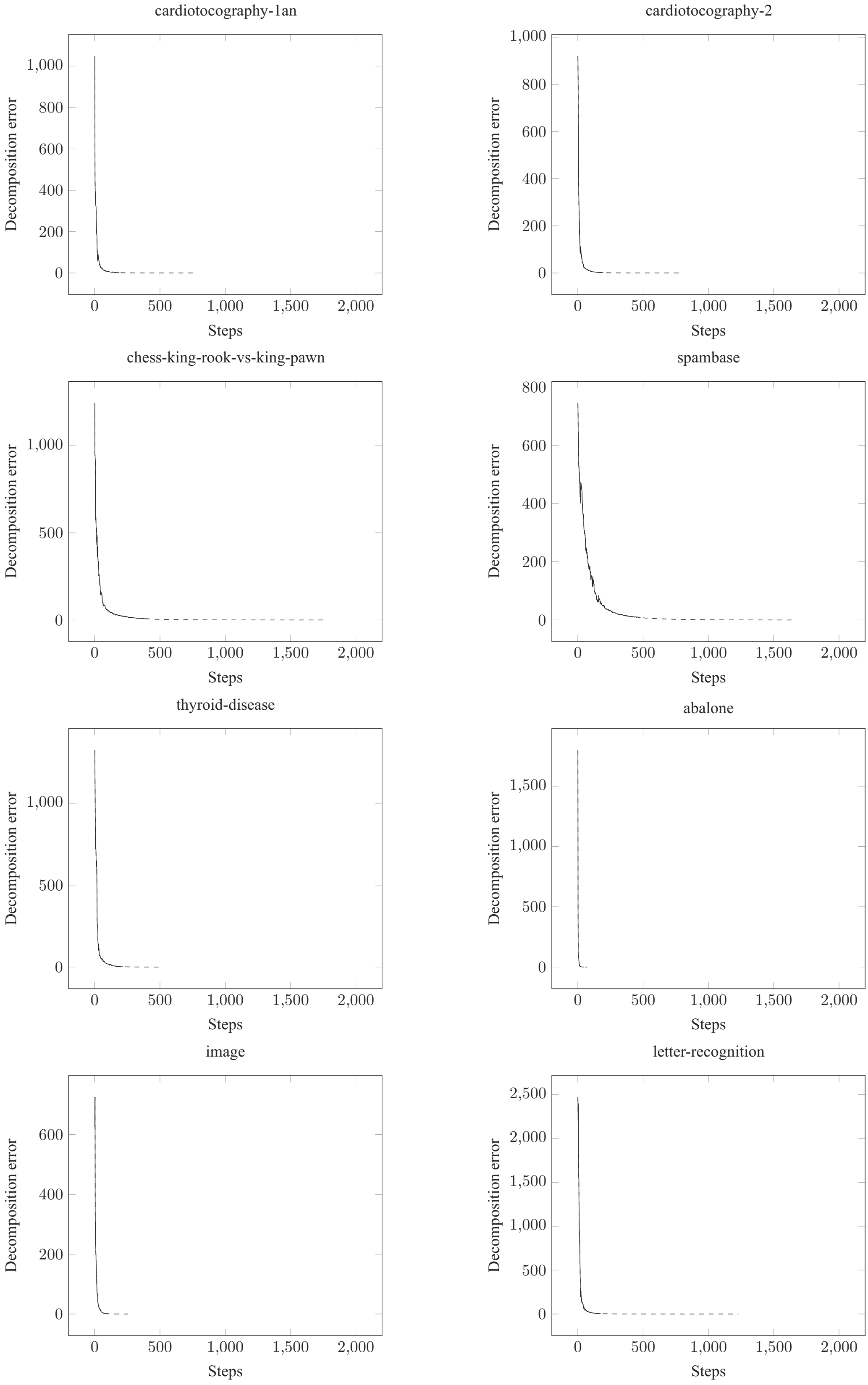

Fig. 8. Convergence of fast SVD and its version with the gradient stop criterion (part A).

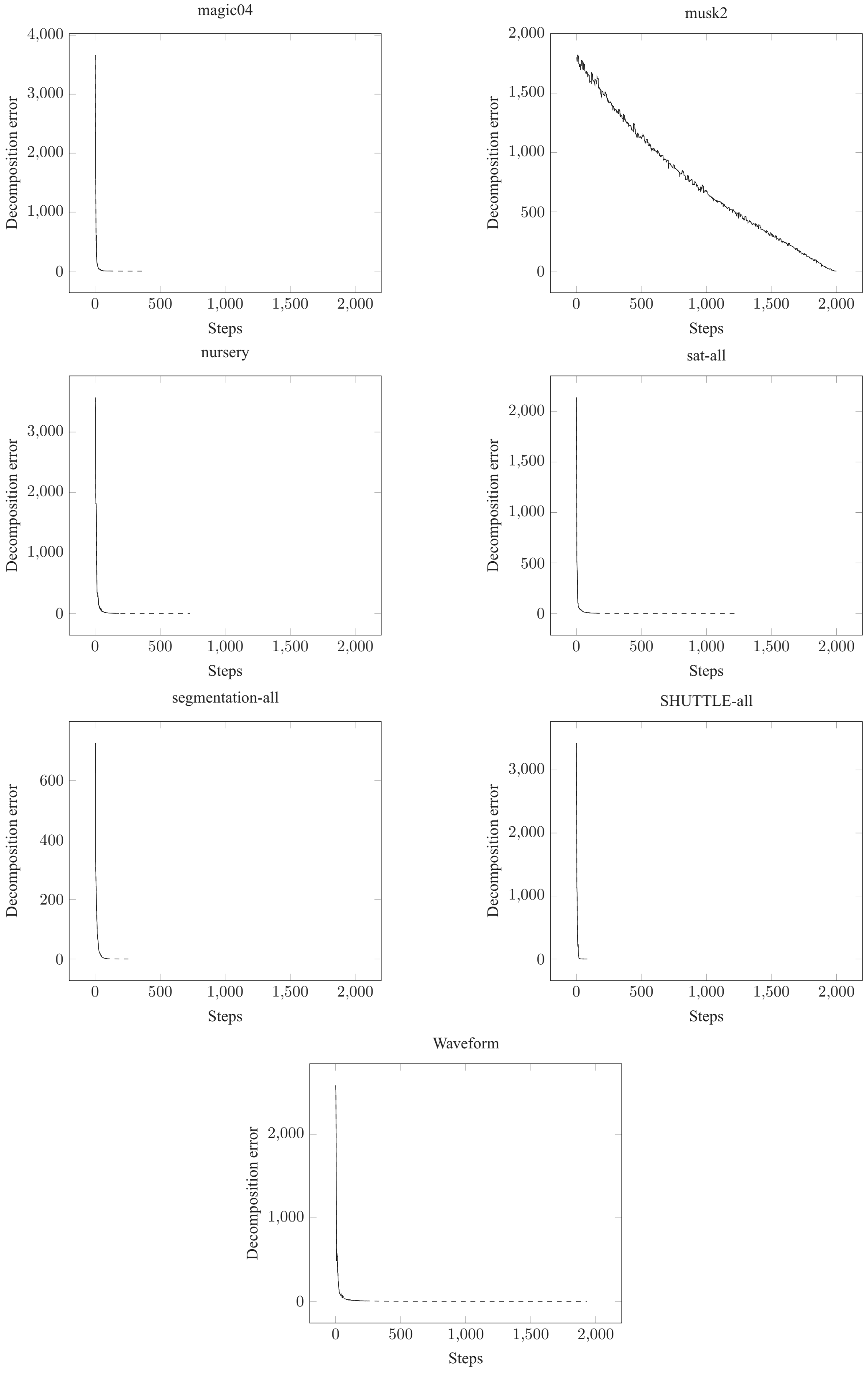

Fig. 9. Convergence of fast SVD and its version with the gradient stop criterion (part B).

Table 5. Comparison of real time fractions used by fast SVD and its version with the gradient stop compared to the execution time of SVD for 1000 kernels.

Name	FastSVD-1000	GradFastSVD-1000	SVD
cardiotocography-1	0.0969	0.0127	1
cardiotocography-2	0.5056	0.0653	1
chess-king-rook-vs-king-pawn	2.2424	0.3168	1
spambase	1.5718	0.2875	1
thyroid-disease	0.6413	0.2303	1
abalone	0.0601	0.0409	1
image	0.0950	0.0361	1
letter-recognition	3.1637	0.2599	1
magic04	0.7527	0.2067	1
musk2	2.3530	2.3977	1
nursery	2.1806	0.3182	1
sat-all	2.0816	0.1611	1
segmentation-all	0.1102	0.0472	1
SHUTTLE-all	0.2325	0.1541	1
Waveform	3.1955	0.1730	1

Table 6. Comparison of real time fractions used by fast SVD and its version with the gradient stop compared to the execution time of SVD for 2000 kernels.

Name	FastSVD-2000	GradFastSVD-2000	SVD
cardiotocography-1	0.3963	0.0416	1
cardiotocography-2	0.4594	0.0449	1
chess-king-rook-vs-king-pawn	2.2720	0.1322	1
spambase	1.0279	0.0858	1
thyroid-disease	0.2397	0.0616	1
abalone	0.019	0.0141	1
image	0.0764	0.0273	1
letter-recognition	2.217	0.1655	1
magic04	0.34	0.0825	1
musk2	1.3005	1.1607	1
nursery	0.6835	0.1028	1
sat-all	1.2957	0.0655	1
segmentation-all	0.0946	0.0341	1
SHUTTLE-all	0.0945	0.0755	1
Waveform	2.5319	0.0614	1

Algorithm 5. Neural network learning with SVD.

```
function networkLearning (M, learningType)
if learningType == linear then
    $A$ = construct $X$ according to Eqn. (6)
else
    $A$ = construct $G$ according to Eqn. (7)
end if
$Q$ = constructQG($A$)
$[\tilde{U}, \Sigma, V^T]$ = SVD($Q^T A$)
$\mathbf{w} = V\Sigma^{-1}\tilde{U}^T Q^T \mathbf{y}$
return $\mathbf{w}$
```

machine with the highest average accuracy is ranked 1. Then, the following machines in the accuracy order whose accuracies are not statistically different (according to the paired t-test) 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 ranked group (2, 3, and so on), and an analogous process is repeated on the remaining (yet unranked) machines.

The last three rows in Tables 7 and 8 present cumulative results on mean ranks (and their standard deviation), numbers of wins (how many times a given machine configuration was the winner or statistically not worse) and the average accuracies. Table 7 compares the performances performance of the RBFN/ELM with 1000 and 2000 Gaussian kernels using fast SVD, fast SVD with the gradient stop and standard SVD for learning. In a similar manner, Table 8 presents results for 100 and 200 kernels.

In most cases the best accuracies were of course obtained by non-approximated SVD. However, fast SVD was in many cases as good as SVD. Fast SVD with the

Table 7. Accuracy comparison for fast SVD with 1000 kernels, gradient fast SVD with 1000 kernels, fast SVD with 2000 kernels, gradient fast SVD with 2000 kernels, and SVD with 1000 and 2000 kernels.

	FastSVD-1k	GradFastSVD-1k	FastSVD-2k	GradFastSVD-2k	SVD-1k	SVD-2k
cardiotocography-1	**83.3±2.1(1)**	79.8±2.5(3)	82.9±2.2(2)	80.1±2.6(3)	**83.4±2.1(1)**	**83.1±2.2(1)**
cardiotocography-2	92.4±1.6(2)	91±1.8(3)	**92.7±1.6(1)**	91.2±1.8(3)	**92.6±1.6(1)**	**92.6±1.6(1)**
chess-king-rook-vs-king-pawn	99.4±0.42(2)	98.3±0.77(4)	99.4±0.44(2)	98.5±0.7(3)	**99.4±0.43(1)**	99.4±0.44(2)
spambase	**92.9±1.3(1)**	92.6±1.2(2)	91.9±1.2(3)	**92.8±1.2(1)**	**92.9±1.2(1)**	91.9±1.2(3)
thyroid-disease	95.6±0.53(2)	94.7±0.53(3)	**95.8±0.53(1)**	94.7±0.49(3)	95.6±0.53(2)	**95.9±0.52(1)**
abalone	26.4±2(2)	**26.4±2(1)**	**26.6±2.2(1)**	26.3±2.1(2)	**26.4±2(1)**	**26.5±2(1)**
image	**95.4±1.5(1)**	93.1±1.5(3)	95.1±1.3(2)	93.2±1.5(3)	**95.5±1.3(1)**	95.1±1.3(2)
letter-recognition	92.3±0.57(4)	81.9±1.2(6)	93±0.61(2)	82.8±1.3(5)	92.4±0.55(3)	**93±0.6(1)**
magic04	**86.6±0.69(1)**	86±0.66(3)	**86.6±0.67(1)**	86.1±0.67(2)	**86.6±0.69(1)**	**86.6±0.68(1)**
musk2	97.8±0.65(2)	97.8±0.65(2)	**99.4±0.33(1)**	**99.4±0.33(1)**	97.8±0.65(2)	**99.4±0.33(1)**
nursery	96.9±0.37(4)	92.8±0.67(6)	97±0.36(2)	93±0.6(5)	97±0.37(3)	**97.1±0.37(1)**
sat-all	90.3±1.1(2)	88.1±1(3)	**90.6±1.1(1)**	88.2±1.1(3)	90.3±1.2(2)	**90.6±1.1(1)**
segmentation-all	**95.4±1.5(1)**	93.1±1.5(3)	95.1±1.3(2)	93.2±1.5(3)	**95.5±1.3(1)**	95.1±1.3(2)
SHUTTLE-all	99±0.15(4)	98.7±0.18(6)	99.1±0.13(2)	98.8±0.18(5)	99.1±0.16(3)	**99.2±0.12(1)**
Waveform	84±1.6(3)	**86.3±1.4(1)**	82.7±1.6(4)	86.1±1.6(2)	84±1.6(3)	82.7±1.5(4)
Mean Accuracy	88.5±1.1	86.7±1.2	88.5±1	87±1.2	88.6±1	88.5±1
Mean Rank	**2.13±0.3**	**3.27±0.43**	**1.8±0.23**	**2.93±0.34**	**1.73±0.24**	**1.53±0.24**
Wins[unique]	5[0]	2[1]	6[0]	2[0]	8[1]	10[3]

Table 8. Accuracy comparison for fast SVD with 100 kernels, gradient fast SVD with 100 kernels, fast SVD with 200 kernels, gradient fast SVD with 200 kernels, and SVD with 100 and 200 kernels.

	FastSVD-100	GradFastSVD-100	FastSVD-200	GradFastSVD-200	SVD-100	SVD-200
cardiotocography-1	79±2.6(3)	78.2±2.8(4)	81.3±2.3(2)	79.1±2.5(3)	79±2.6(3)	**81.6±2.2(1)**
cardiotocography-2	91.2±1.8(2)	90.7±1.8(3)	**91.7±1.8(1)**	90.8±1.9(3)	91.2±1.8(2)	**91.8±1.7(1)**
chess-king-rook-vs-king-pawn	95±1.4(3)	95±1.4(3)	**97.1±1(1)**	97±1.1(2)	95±1.4(3)	**97.1±1(1)**
spambase	91.9±1.3(2)	91.9±1.3(2)	**92.4±1.2(1)**	**92.3±1.2(1)**	92±1.3(2)	**92.4±1.2(1)**
thyroid-disease	94.5±0.48(4)	94.4±0.46(5)	94.9±0.52(2)	94.5±0.48(4)	94.5±0.51(3)	**94.9±0.5(1)**
abalone	**26.3±2.1(1)**	26.1±2.2(2)	**26.4±2(1)**	26.1±2(2)	**26.4±2(1)**	**26.3±1.9(1)**
image	94.3±1.6(3)	92.4±1.8(5)	94.9±1.4(2)	92.7±1.6(4)	94.9±1.5(2)	**95.6±1.4(1)**
letter-recognition	76.4±0.88(3)	76.4±0.88(3)	**82.7±0.89(1)**	79.9±1.4(2)	76.4±0.88(3)	**82.7±0.89(1)**
magic04	86.1±0.68(3)	85.8±0.67(4)	86.3±0.65(2)	85.8±0.66(4)	86.1±0.68(3)	**86.4±0.66(1)**
musk2	87.5±0.64(2)	87.5±0.64(2)	**89.8±0.76(1)**	**89.8±0.76(1)**	87.5±0.64(2)	**89.8±0.76(1)**
nursery	91.4±0.67(3)	91.4±0.66(3)	**93.9±0.6(1)**	91.9±0.72(2)	91.4±0.67(3)	**93.9±0.6(1)**
sat-all	87.5±1.1(3)	87.3±1.1(4)	**88.5±1.1(1)**	87.6±1.1(2)	87.5±1.1(3)	**88.5±1.1(1)**
segmentation-all	94.3±1.6(3)	92.4±1.8(5)	94.9±1.4(2)	92.7±1.6(4)	94.9±1.5(2)	**95.6±1.4(1)**
SHUTTLE-all	98.7±0.18(4)	98.4±0.29(6)	98.7±0.18(2)	98.6±0.24(5)	98.7±0.17(3)	**98.8±0.18(1)**
Waveform	**86.6±1.4(1)**	**86.6±1.4(1)**	**86.5±1.4(1)**	86.4±1.4(2)	**86.6±1.4(1)**	**86.5±1.4(1)**
Mean Accuracy	85.4±1.2	85±1.3	86.7±1.1	85.7±1.2	85.5±1.2	86.8±1.1
Mean Rank	**2.67±0.24**	**3.47±0.38**	**1.4±0.14**	**2.73±0.33**	**2.4±0.2**	**1±0**
Wins[unique]	2[0]	1[0]	9[0]	2[0]	2[0]	15[6]

gradient performs a little worse indeed, but the results are often almost on the border of statistical difference, while the computational time is many times smaller. Also, it sometimes happens that even the fastest SVD has the best accuracy.

The most important observation is that, despite the fact that approximation in the proposed method is so strong, the results are not much worse. This is especially important when we browse many configurations before the final learning of the end model.

4. Summary

The complexity of neural network learning algorithms is so often crucial, especially in learning from huge datasets (a huge dataset meaning one with a big number of instances or a big number of attributes). This article was focused on the construction and analysis of fast algorithms of SVD computation for training neural networks like the RBFN, ELM, or deep ELM. The proposed modification of the fast SVD algorithm is on the average 10 times faster than standard SVD. Such algorithms can be used in every place where standard SVD is now employed: in supervised and unsupervised learning and in classification, regression, PCAs and so on.

Although computational time consumption of the new fast SVD algorithm is significantly reduced, the accuracies of classifiers are still satisfactory. This is especially important when the learning phase is repeated many times in order to find an appropriate model of a neural network or another learning machine, which is a typical case.

References

Bishop, C.M. (1991). Training with noise is equivalent to Tikhonov regularization, *Neural Computation* **7**(1): 108–116.

Boser, B.E., Guyon, I.M. and Vapnik, V. (1992). A training algorithm for optimal margin classifiers, *in* D. Haussler (Ed.), *Proceedings of the 5th Annual ACM Workshop on Computational Learning Theory, Pittsburgh, PA, USA*, pp. 144–152.

Broomhead, D.S. and Lowe, D. (1988). Multivariable functional interpolation and adaptive networks, *Complex Systems* **2**(3): 321–355.

Dumais, S.T. (2005). Latent semantic analysis, *Annual Review of Information Science and Technology* **38**(1): 188–230.

Eirola, E., Lendasse, A., Vandewalle, V. and Biernacki, C. (2014). Mixture of Gaussians for distance estimation with missing data, *Neurocomputing* **131**: 32–42.

Goodfellow, I., Bengio, Y. and Courville, A. (2016). *Deep Learning*, MIT Press, Cambridge, MA, http://www.deeplearningbook.org.

Górecki, T. and Łuczak, M. (2013). Linear discriminant analysis with a generalization of the Moore–Penrose pseudoinverse, *International Journal of Applied Mathematics and Computer Science* **23**(2): 463–471, DOI: 10.2478/amcs-2013-0035.

Halko, N., Martinsson, P.G. and Tropp, J.A. (2011). Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, *SIAM Review* **53**(2): 217–288.

Heseltine, T., Pears, N., Austin, J. and Chen, Z. (2003). Face recognition: A comparison of appearance-based approaches, *7th International Conference on Digital Image Computing: Techniques and Applications, Sydney, Australia*, Vol. 1, pp. 59–68.

Huang, G.-B., Bai, Z., Kasun, L.L.C. and Vong, C.M. (2015). Local receptive fields based extreme learning machine, *IEEE Computational Intelligence Magazine* **10**(2): 18–29.

Huang, G.-B., Zhu, Q.-Y. and Siew, C.-K. (2004). Extreme learning machine: A new learning scheme of feedforward neural networks, *International Joint Conference on Neural Networks, Budapest, Hungary*, pp. 985–990.

Huang, G.-B., Zhu, Q.-Y. and Siew, C.-K. (2006). Extreme learning machine: Theory and applications, *Neurocomputing* **70**(1–3): 489–501.

Jankowski, N. (2013). Meta-learning and new ways in model construction for classification problems, *Journal of Network & Information Security* **4**(4): 275–284.

Jankowski, N. (2018). Comparison of prototype selection algorithms used in construction of neural networks learned by SVD, *International Journal of Applied Mathematics and Computer Science* **28**(4): 719–733, DOI: 10.2478/amcs-2018-0055.

Merz, C.J. and Murphy, P.M. (1998). UCI Repository of Machine Learning Databases, https://archive.ics.uci.edu/ml/index.php.

Mitchell, T. (1997). *Machine Learning*, McGraw Hill, New York, NY.

Rumelhart, D.E., Hinton, G.E. and Williams, R.J. (1986). Learning internal representations by error propagation, *in* J.L.M.D.E. Rumelhart (Ed.), *Parallel Distributed Processing: Explorations in Microstructure of Congnition*, Vol. 1: Foundations, MIT Press, Cambridge, MA, pp. 318–362.

Sovilj, D., Eirola, E., Miche, Y., Bjork, K.-M., Nian, R., Akusok, A. and Lendasse, A. (2016). Extreme learning machine for missing data using multiple imputations, *Neurocomputing* **174**(PA): 220–231.

Tang, J., Deng, C., Member, S. and Huang, G.-B. (2016). Extreme learning machine for multilayer perceptron, *IEEE Transactions on Neural Networks and Learning Systems* **27**(4): 809–821.

Tikhonov, A.N. and Arsenin, V.Y. (1977). *Solutions of Ill-posed Problems*, W.H. Winston, Washington, DC.

Vapnik, V. (1995). *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, NY.

Norbert Jankowski has been an associate professor at Nicolaus Copernicus University in Toruń, Poland, since 2012. He received his DSc degree in computer science at the Systems Research Institute, Polish Academy of Sciences, Warsaw. He obtained his PhD in computer science at the Institute of Biocybernetic and Biomedical Engineering, Polish Academy of Sciences, Warsaw. He received his MSc in computer science at the Institute of Computer Science, University of Wrocław, Poland. He is the author of 78 scientific articles and two books. His main research areas are computational intelligence, meta-learning, machine learning, neural networks, data mining, pattern recognition, complexity, algorithms and data structures.

Rafał Linowiecki has been an assistant researcher at Nicolaus Copernicus University in Toruń, Poland, since 2015. He received his MSc degree in applied computer science at the Faculty of Physics, Astronomy and Informatics, Nicolaus Copernicus University. He received a Bachelor's degree in mathematics at the Faculty of Mathematics and Computer Science, Nicolaus Copernicus University. He also received a Bachelor's degree in economics at the Faculty of Economic Sciences and Management, Nicolaus Copernicus University. His scientific interests include machine learning, data mining, computational intelligence, complexity and artificial intelligence. He also specializes in custom software development with the use of cloud services.

Received: 6 September 2018
Revised: 29 December 2018
Re-revised: 9 February 2019
Accepted: 22 February 2019