Pattern Analysis and Applications (2019) 22:1221–1231
https://doi.org/10.1007/s10044-018-0697-0

SHORT PAPER

Training neural networks on high-dimensional data using random projection

Piotr Iwo Wójcik[1] · Marcin Kurdziel[1]

Received: 30 November 2016 / Accepted: 9 March 2018 / Published online: 19 March 2018

Abstract

Training deep neural networks (DNNs) on high-dimensional data with no spatial structure poses a major computational problem. It implies a network architecture with a huge input layer, which greatly increases the number of weights, often making the training infeasible. One solution to this problem is to reduce the dimensionality of the input space to a manageable size, and then train a deep network on a representation with fewer dimensions. Here, we focus on performing the dimensionality reduction step by randomly projecting the input data into a lower-dimensional space. Conceptually, this is equivalent to adding a random projection (RP) layer in front of the network. We study two variants of RP layers: one where the weights are fixed, and one where they are fine-tuned during network training. We evaluate the performance of DNNs with input layers constructed using several recently proposed RP schemes. These include: Gaussian, Achlioptas', Li's, subsampled randomized Hadamard transform (SRHT) and Count Sketch-based constructions. Our results demonstrate that DNNs with RP layer achieve competitive performance on high-dimensional real-world datasets. In particular, we show that SRHT and Count Sketch-based projections provide the best balance between the projection time and the network performance.

Keywords Random projection · Neural networks · High-dimensional data · Sparse data

1 Introduction

Deep-learning methods excel in many classical machine learning tasks, such as image and speech recognition or sequence modelling [1]. Unlike conventional machine learning techniques, they do not require handcrafted features, but instead discover features during learning. However, the dimensionality of input data in neural network applications is relatively low; for example, networks trained for speech recognition tasks employ input vectors with size on the order of hundreds of dimensions [2]. Learning with larger input dimensionality typically requires some structure in the input data. This is the case in, e.g. convolutional neural networks (CNNs), which take advantage of the spatial structure of images by exploiting the local connectivity and sharing the weights between spatial locations. This greatly reduces the number of learnable parameters, enabling CNNs to work with up to hundreds of thousands of input pixels.

The motivation for this work stems from the problem of training DNNs on unstructured data with a large number of dimensions. Such data often arise in social media, web crawling, gene sequencing or biomedical applications. When there is no exploitable input structure, training DNNs on high-dimensional data poses a significant computational problem. The reason for this is the implied network architecture, and in particular an input layer which may contain billions of weights. Even with recent advances in GPGPU computing, training networks with this number of parameters is infeasible. Therefore, learning in such applications is often performed with linear classifiers, usually support vector machines or logistic regression [3]. We show that this problem can be solved by incorporating random projection into the network architecture. In particular, we propose to prepend the network with an input layer whose weights are initialized with a random projection matrix. We study two ways of training this architecture: one where the parameters of the RP layer are fixed during training, and one where they are fine-tuned with error backpropagation.

✉ Piotr Iwo Wójcik
pwojcik@agh.edu.pl

Marcin Kurdziel
kurdziel@agh.edu.pl

[1] Department of Computer Science, Faculty of Computer Science, Electronics and Telecommunications, AGH University of Science and Technology, al. A. Mickiewicza 30, 30-059 Kraków, Poland

Our results show that on high-dimensional real-world datasets neural networks with RP layer achieve performance competitive with the state-of-the-art alternatives.

This work is organized as follows. First, in Sect. 2, we briefly review most popular RP schemes. In Sect. 3, we introduce the RP layer. Then, in Sect. 4, we evaluate networks with fixed and fine-tuned RP layers on several synthetic and real-world datasets.

2 Random projection matrices

Let $\mathbf{A} \in \mathbb{R}^{n\times d}$ denote a data matrix consisting of n data points in $\mathbb{R}^d$. The key idea behind random projection stems from the Johnson–Lindenstrauss Lemma [4], which states that a set of n points in a high-dimensional vector space can be embedded into $k = \mathcal{O}(\epsilon^{-2}\log n)$ dimensions, with the distances between these points preserved up to a factor of $1+\epsilon$. This limit can be realized with a linear projection $\tilde{\mathbf{A}} = \mathbf{AR}$, for a carefully designed random matrix $\mathbf{R} \in \mathbb{R}^{d\times k}$ $(k \ll d)$.

Several constructions have been recently proposed for the RP matrix $\mathbf{R}$. The simplest and most straightforward construction is the Gaussian random matrix, whose entries r_{ij} are i.i.d. samples from a normal distribution $\mathcal{N}(0, \frac{1}{k})$ [5, 6]. In [7] Achlioptas proposed a sparser projection matrix, in which only one third of the elements are nonzero:

$$r_{ij} = \sqrt{\frac{3}{k}} \cdot \begin{cases} 1 & \text{with probability} \quad 1/6 \\ 0 & \text{with probability} \quad 2/3 \\ -1 & \text{with probability} \quad 1/6 \end{cases}.$$

Li et al. [8] introduced an even sparser random matrix, by extending Achlioptas' construction:

$$r_{ij} = \sqrt{\frac{s}{k}} \cdot \begin{cases} 1 & \text{with probability} \quad \frac{1}{2s} \\ 0 & \text{with probability} \quad 1-\frac{1}{s} \\ -1 & \text{with probability} \quad \frac{1}{2s} \end{cases}.$$

Note that setting $s = 3$ in the above construction yields Achlioptas' matrix. Li et al. showed, however, that one can use s as high as $\sqrt{d}$ to significantly sparsify the projection matrix and in turn to speed up the projection.

Ailon and Chazelle [9] proposed a fast embedding based on the Hadamard–Walsh matrix, the so-called subsampled randomized Hadamard transform (SRHT). The Hadamard–Walsh matrix $\mathbf{H}_t \in \mathbb{R}^{t\times t}$ is defined recursively as:

$$\mathbf{H}_1 = 1, \qquad \mathbf{H}_t = \begin{bmatrix} \mathbf{H}_{t/2} & \mathbf{H}_{t/2} \\ \mathbf{H}_{t/2} & -\mathbf{H}_{t/2} \end{bmatrix},$$

for any t that is a power of two. Ailon and Chazelle defined their projection matrix $\mathbf{R}_{HT}$ as a scaled product of three matrices:

$$\mathbf{R}_{HT} = \frac{1}{\sqrt{k}}\mathbf{DHP},$$

where $\mathbf{D}$ is a $d\times d$ diagonal matrix with random entries drawn uniformly from $\{1, -1\}$, $\mathbf{H}$ is a $d\times d$ normalized Hadamard–Walsh matrix: $\mathbf{H} = \sqrt{\frac{1}{t}}\mathbf{H}_t$ and $\mathbf{P}$ is a sparse $d\times k$ random matrix. This construction assumes that d is a power of two and is greater or equal to the dimensionality of the data. If the dimensionality of the data is lower than d we pad $\mathbf{A}$ with zeros. Elements of $\mathbf{P}$ are set to 0 with probability $1-q$ and are drawn from a normal distribution $\mathcal{N}(0, \frac{1}{q})$ otherwise; q is a sparsity parameter. Using matrix multiplication akin to fast Fourier transform, the product $\mathbf{AR}_{HT}$ can be computed in $\mathcal{O}(nd\log d)$ operations, as opposed to $\mathcal{O}(ndk)$ if the projection was done by a naive matrix multiplication. In [10] the running time of SRHT was further improved to $\mathcal{O}(nd\log k)$.

Another family of RP methods is based on the Count Sketch algorithm. The Count Sketch algorithm was initially proposed by Charikar et al. [11] as a method to estimate the frequency of items in a data stream. In [12, 13] it was used as a dimensionality reduction method. The explicit form of the projection matrix was then given in [14]. The Count Sketch projection matrix can be given as: $\mathbf{R}_{CS} = \mathbf{DC}$, where $\mathbf{D}$ is defined as in SRHT and $\mathbf{C}$ is a $d\times k$ sparse matrix, with each row chosen randomly from the k standard basis vectors of $\mathbb{R}^k$. Similarly to SRHT, in Count Sketch scheme the projection can be performed without a naive multiplication of the data matrix by $\mathbf{R}_{CS}$. Specifically, the result matrix $\tilde{\mathbf{A}}$ is initialized with zeros and then each column of the data matrix $\mathbf{A}$ is multiplied by -1 with probability 50% and added to a randomly selected column of $\tilde{\mathbf{A}}$. The computational complexity of this projection scheme is $\mathcal{O}(nd)$, i.e. linear w.r.t. the size of the input data. Because of its low computational cost, Count Sketch-based projections have recently drawn considerable attention [15–17].

We summarize the time complexity of different RP schemes in Table 1.

3 Neural networks with random projection layer

The weights in the random projection layer can be either fixed or seen as model parameters that are fine-tuned during training. The first variant, further called **fixed-weight RP layer**, can be interpreted as training a standard network

Table 1 Time complexity of random projection schemes

RP scheme	Matrix construction time	Projection time	
		Dense input	Sparse input
Gaussian	$\mathcal{O}(dk)$	$\mathcal{O}(ndk)$	$\mathcal{O}(\mathrm{nnz}(\mathbf{A})k)$
Achlioptas'	$\mathcal{O}(dk)$	$\mathcal{O}(ndk)$	$\mathcal{O}(\mathrm{nnz}(\mathbf{A})k)$
Li's	$\mathcal{O}(\sqrt{d}k)$	$\mathcal{O}(n\sqrt{d}k)$	$\mathcal{O}(\mathrm{nnz}(\mathbf{A})k)$[a]
SRHT	$\mathcal{O}(dk + d\log d)$	$\mathcal{O}(nd\log k)$	$\mathcal{O}(nd\log k)$
Count Sketch	$\mathcal{O}(d)$	$\mathcal{O}(nd)$	$\mathcal{O}(\mathrm{nnz}(\mathbf{A}))$

$\mathbf{A}$ is a $n \times d$ dataset matrix and k is the target dimensionality. If $\mathbf{A}$ is sparse nnz($\mathbf{A}$) denotes the number of nonzero elements in $\mathbf{A}$

[a]For a standard implementation of sparse-dense matrix multiplication

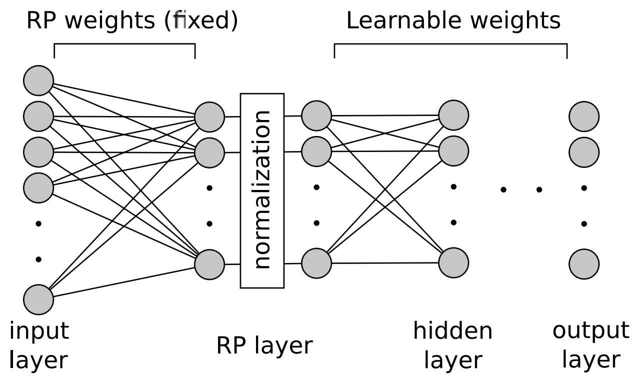

Fig. 1 Neural network with fixed-weight random projection layer

architecture on data whose dimensionality has been reduced with random projection. A theoretical motivation for learning on such randomly projected data has been given in [18, 19]. In particular, Arriaga and Vempala [18] provided a clear motivation that can be summarized in two points: (i) learning from randomly projected data is possible since random projection preserves a lot of the input structure in the lower-dimensional space, and (ii) learning in the lower-dimensional space should require fewer examples and therefore be faster. The second RP layer variant, which we call **fine-tuned RP layer**, may improve the network performance compared to fixed RP weights. However, it has a significantly higher computational cost. Nevertheless, we show that with carefully designed architecture and training regime it can be applied to real-world problems.

3.1 Fixed-weight random projection layer

There are two main reasons for using DNNs with fixed-weight RP layer (Fig. 1). First, with fixed weights we can perform the projection and normalization of the whole dataset only once prior to network training. Such optimization is especially beneficial when the lower-dimensional projection fits in the operating memory, while the original input data require out-of-core processing. Second, for dense RP constructions, such as Gaussian, Achlioptas' or SRHT, an update of the weights in the RP layer may have a prohibitive computational cost; for example, dense RP matrices for some of the tasks reported in Sect. 4 have up to tens of billions of weights. Fine-tuning weights in the RP layer is more practical for sparse RP constructions, especially if we restrict the updates to the weights that are initially nonzero. We further elaborate on this approach in Sect. 3.2.

Layers that follow fixed-weight random projection can be trained from scratch with error backpropagation. However, we found that the performance can be improved by pretraining these layers with deep belief networks (DBNs) [20], and then fine-tuning with error backpropagation. Before the projected data are used to train the "learnable" part of the network, we normalize each dimension to zero mean and unit variance. Initial evaluation showed that this is necessary: pretraining on unnormalized data was unstable, especially for highly sparse datasets. One particular advantage of pretraining a DBN on normalized data is that we can use Gaussian units [21] in the first layer of the "learnable" part of the network.

The choice of the projection matrix in the RP layer is not trivial and depends on the dimensionality and sparsity of the input data. In particular, these two factors have a significant impact on the computational cost of training. While the projection time is usually negligible in comparison with the training time, this may not be the case when the data dimensionality is very high. Fortunately, especially for large unstructured datasets, high dimensionality often goes hand in hand with high sparsity. This is beneficial from the computational point of view since sparse representation facilitates faster projection. In particular, the performance of RP schemes that involve matrix multiplication can be improved by fast algorithms for sparse matrix multiplication, e.g. [22, 23]. Some other RP schemes can also be optimized to take advantage of the data sparsity [15].

Another aspect to consider is the sparsity of the projection matrix itself. Random projection matrices that provide the best quality of embedding are typically dense [24]. Unfortunately, applying dense projection schemes to huge datasets can be computationally prohibitive. In this case, one needs to resort to more efficient projection schemes. One possibility is to employ a projection scheme that does not require matrix multiplication. A good example of such random projection method is SRHT [10]. Another approach is to use a sparse projection matrix, e.g. Li's construction. Moreover, these two approaches can be combined into a projection scheme where the RP matrix is sparse and the projection does not require an explicit matrix multiplication. This results in very efficient projection methods, such as the Count Sketch projection. However, projecting sparse data with sparse RP matrices, regardless if they are explicitly or implicitly constructed, can introduce significant distortions in the embedding [9]. These distortions may, in turn, affect the network accuracy. Therefore, for large datasets, the choice of the RP layer type is a trade-off between the network accuracy and the computational complexity of the RP embedding. Investigating this trade-off, apart from enabling training of neural networks on high-dimensional data, is one of the goals of this work.

3.2 Fine-tuned random projection layer

While the idea of fine-tuning weights in the RP layer may seem straightforward, there are several technical difficulties that make its implementation challenging. They stem, primarily, from the high computational cost of performing the weight updates and normalizing the layer outputs. As we discussed in the previous section, for large-scale datasets performing even a single weights' update in a dense RP layer is computationally prohibitive. For example, for the `KDD2010-a` dataset, a fully dense RP layer with 1000 output units contains more than 2×10^{10} weights—nearly twice as much as the number of weights in the largest currently used networks [25]. Fortunately, we can reduce the number of weights to the order of millions by choosing a sparse variant of the RP layer. In this work, we propose to construct fine-tuned RP layers using two sparse random projection schemes presented in Sect. 2, i.e. Li's and Count Sketch-based projections. Compared to a dense RP matrix, Li's and Count Sketch constructions reduce the total number of weights by a factor of $\sqrt{d}$ and k, respectively, where d is the number of input units and k is the number of output units.

To ensure that the number of model parameters, and in turn the computational cost, does not increase during training, we update only these elements in the fine-tuned RP matrix that are initially nonzero. This construction can be interpreted as a network layer with sparse connectivity (Fig. 2). To further improve the training performance, we can restrict the weight updates in the RP layer to a fraction of training mini-batches. We found that even with sparse RP layers this approximation is necessary for our largest benchmark datasets. Importantly, to reduce the bias introduced by skipping some of the weight updates, the updates are performed for randomly selected mini-batches.

When training networks with the fixed-weight RP layer, we normalize RP activations to zero mean and unit variance (using moments calculated on the training set). Since the weights in the RP layer do not change during training, this operation can be performed only once, unlike in networks with fine-tuned random projection. When training networks with fine-tuned RP layer, we normalize activations by adding batch normalization [26] between the random projection and the activation function.

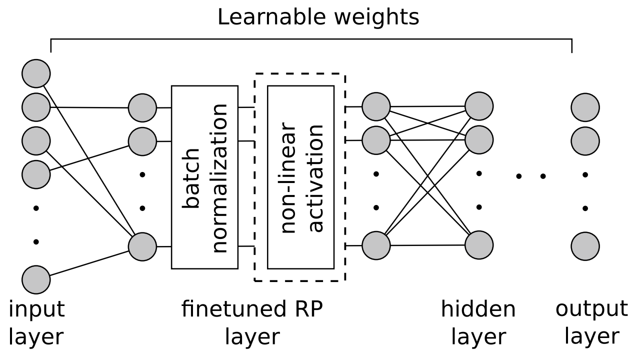

Fig. 2 Neural network with fine-tuned random projection layer. Weights in the layer are initialized to a sparse RP matrix. Only weights that are initially nonzero are part of the model. The output of the projection is batch normalized and optionally transformed with a nonlinear activation function

We found that networks with fine-tuned RP layer are best trained end-to-end starting from random initialization. Initially, we also considered different training regimes. For example, we experimented with networks that were first trained without changing the RP layer and then fine-tuned end-to-end with backpropagation. However, this training regime yielded inferior results.

4 Experiments

To evaluate the performance of DNNs with RP layer, we carried out a set of comparative experiments on several classification tasks. We begin with experimental evaluation of fixed-weight RP layer trained on synthetic and real-world datasets.

4.1 Fixed-weight random projection layer

Experiments with fixed-weight RP layer were carried out using all RP techniques described in Sect. 2, namely Gaussian, Achlioptas', Li's, SRHT and Count Sketch. For Gaussian and Achlioptas' schemes, we implemented a memory efficient projection procedure that avoids allocating the whole projection matrix at once. For SRHT we did not explicitly construct $\mathbf{R}_{HT}$, but instead implemented the projection using the fast Walsh–Hadamard Transform[1]. These optimizations were necessary since in most of our experiments the projection matrix would not fit in the operating memory.

We trained the evaluated networks using mini-batch stochastic gradient descent with momentum. Amplitudes of

[1] Based on a Python implementation available at: http://www.quantatrisk.com/2015/04/10/fast-walsh-hadamard-transform-python-matlab/.

weights were limited with the L2 cost. During fine-tuning, the learning rate was decreased according to a slow exponential decay, while the momentum was slowly increased. We also used dropout [27] to prevent overfitting. We employed rectified linear (ReLU) units [28] in hidden layers and Gaussian units in the layer after the random projection. Values of learning hyperparameters were selected with experiments on validation sets constructed from the training data. All experiments were run using a GPU-accelerated library described in [29].

The evaluation begins with experiments on synthetic datasets, where we investigate the impact of data sparsity and the number of significant features on the performance of networks with RP. As a reference in these experiments, we use two linear classifiers, i.e. logistic regression (LR) and support vector machines (SVM). We used implementations of these classifiers provided in the LIBLINEAR package [30]. Subsequent experiments evaluate the performance of networks with RP layer on real-world datasets. First, we show a toy example on the commonly used `MNIST` dataset [31]. Afterwards, we present results on several large-scale sparse datasets.

4.1.1 Experiments on synthetic datasets

We prepared several 10^6-dimensional synthetic datasets, each consisting of 1.25×10^6 examples belonging to two balanced classes. Each dataset was constructed by first generating a ρ-dense matrix S (ρ being the fraction of nonzero elements in S), and selecting a fraction of features, ϕ, that would separate examples from the two classes. Nonzero elements in S were drawn randomly from a normal distribution $\mathcal{N}(0, 1)$. To separate the classes, we picked examples from one class and added a Gaussian noise with nonzero mean to all nonzero elements in significant features. Note that this operation does not alter the sparsity of S. We generated two groups of such sparse datasets:

- with fixed fraction of significant features $\phi = 0.2$ and density ρ ranging from 10^{-6} to 10^{-4},
- with fixed density $\rho = 10^{-4}$ and a fraction of significant features ϕ ranging from 0.01 to 0.2.

The above ranges for ρ and ϕ were chosen so that the most difficult dataset variants had, on average, one or two significant nonzero features per example. We randomly selected 80% rows of S as the training set and the remaining 20% as the test set.

The synthetic datasets were projected to 1000 dimensions and used to pretrain a network with two hidden layers, each consisting of 3, 000 neurons. After pretraining, we added a logistic regression unit on top and fine-tuned the network to minimize binary cross-entropy cost.

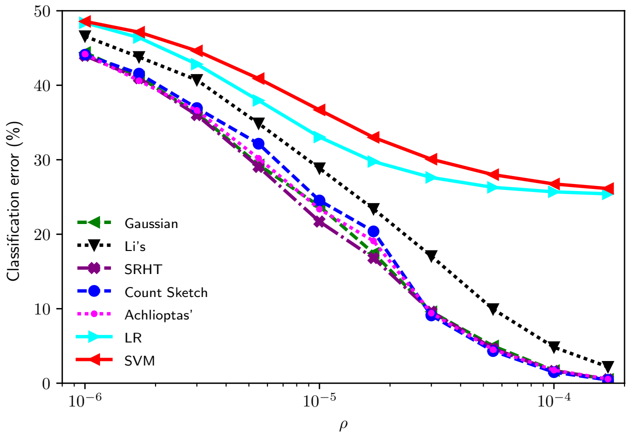

Fig. 3 Classification error on the synthetic datasets with fixed significant feature fraction $\psi = 0.2$ and varying density ρ

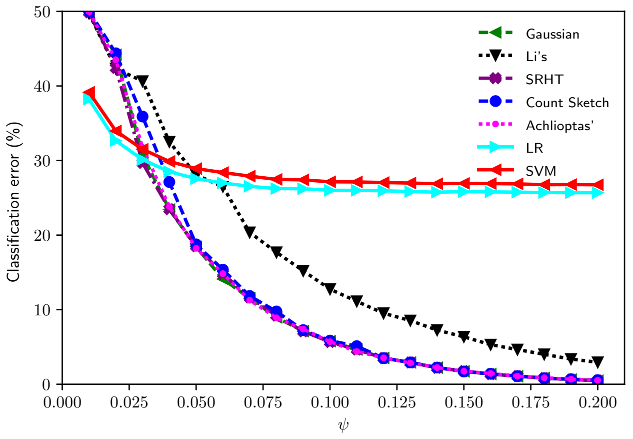

Fig. 4 Classification error on the synthetic datasets with fixed density $\rho = 10^{-4}$ and varying fraction of significant features ψ

The reference logistic regression and SVM models were trained on unprojected data. For both models, we report results obtained with the solvers that performed best on the validation set, namely dual L2-regularized logistic regression and dual L2-regularized L2-loss SVM.

Test classification errors for different density levels and number of significant features are presented in Fig. 3 and Fig. 4. An important observation here is that networks with RP layer significantly outperformed full-dimensional logistic regression and SVM models, except for the tests where each example had, on average, 1–3 nonzero significant features. In practice, we do not expect real-world classification tasks to have an input representation with such a low number of significant features. For example, for all classification algorithms evaluated in this work the accuracy on the hardest variants of the synthetic dataset

Table 2 Summary of real-world datasets used in the evaluation

Dataset	Training set size	Test set size	Dimensions	Classes	Density
`MNIST`	60,000	10,000	784	10	0.191
`webspam`	280,000	70,000	16,609,143	2	2.24×10^{-4}
`url`	1,976,130	420,000	3,231,961	2	3.58×10^{-5}
`KDD2010-a`	8,407,752	510,302	20,216,830	2	1.80×10^{-6}
`KDD2010-b`	19,264,097	748,401	29,890,095	2	9.84×10^{-7}

Density is the fraction of nonzero elements in the training set

Table 3 Classification errors (%) on real-world datasets for networks with fixed-weight RP layers

Dataset	References	Gaussian	Achlioptas'	Li's	SRHT	Count Sketch
`MNIST`	**0.92** [27]	1.06	0.94	1.11	1.04	1.34
`webspam`	**0.32** [33], 0.40 [34]	0.38	0.40	0.36	0.40	**0.32**
`url`	1.23 [35], 1.34 [36]	1.03	1.12	3.75	1.01	**0.96**
`KDD2010-a`	**10.38** [35]	10.86	10.88	11.95	10.86	11.49
`KDD2010-b`	**10.01** [37], 10.42 [34], 13.25 [33]	10.51	10.49	10.98	10.49	10.54

For each dataset, we highlight the result of the best performing method

is much lower than on any of our real-world benchmarks (Sect. 4.1.2), indicating harder input representation.

In all tests, SRHT and Achlioptas' projections yielded the best and almost equal performance. The Count Sketch and the Gaussian projections performed slightly worse, especially for datasets with only a few significant features. Li's projection was outperformed by the other four RP schemes. Another scheme with a sparse projection matrix, i.e. Count Sketch, performed better. However, when the input data were very sparse, it was also outperformed by dense projection schemes.

Li's projection matrix in our experiments had the same sparsity as the Count Sketch matrix but nevertheless performed worse. While it has been argued that sparse projection matrices are not suited for sparse data [9], this result demonstrates that the matrix construction itself also plays an important role. Note that columns in the Count Sketch projection matrix are fully orthogonal, unlike in Li's construction. Moreover, orthogonal weight initialization has been shown to improve the performance of deep networks [32]. We believe that this may be the reason behind the better performance of the Count Sketch RP layer.

4.1.2 Experiments on real-world datasets

To demonstrate the practical effectiveness of DNNs with fixed-weight RP layer, we performed experiments on five real-world classification tasks. The datasets for these tasks are summarized in Table 2.

First, as a toy example, we used the popular `MNIST` benchmark. While it is a small and low-dimensional image dataset, it is frequently used to evaluate neural networks and has well-established reference results. Original `MNIST` images are represented by 256 grey-scale levels, but in our work we use pixel intensities rescaled to the $\langle 0, 1 \rangle$ interval. We use the permutation invariant version of the dataset, i.e. we randomly shuffle the pixel order. We projected the original 784-dimensional `MNIST` digits to 400 dimensions and used them to pretrain and fine-tune networks with two hidden layers consisting of 1000 neurons each. The networks were fine-tuned to minimize multinomial cross-entropy cost. With three RP matrices, namely Achlioptas', Gaussian and Li's, they achieved performance close to the reference performance on the unprojected digits (Table 3). When judging this result, it is important to note that `MNIST` is relatively dense, unlike other datasets used in this work.

In subsequent tests, we used four large high-dimensional datasets, namely `webspam` [38], `url` [39][2], `KDD2010-a` and `KDD2010-b`. All four are binary classification tasks. `webspam` is a dataset consisting of 350,000 descriptors of webpages. The challenge is to detect examples of so-called *Webspam* (or *search spam*), i.e. webpages that are designed to manipulate search engine results. We use the normalized high-dimensional trigram version of the dataset. `webspam` is not provided with standardized train/test set split. Therefore, following [36], we use a random 80%/20% train/test split. The `url` dataset consists of descriptors of 2.4 million URL addresses. Descriptors contain lexical features (bag-of-words representations of tokens in the URL) and host-based features (WHOIS information, location, connection, speed, blacklist membership, etc.). The challenge in this dataset

[2] Available at http://sysnet.ucsd.edu/projects/url/.

Table 4 Test errors (%) for networks with fine-tuned random projection layer

Dataset	Li's RP layer			Count Sketch RP layer		
	Fixed weights	Fine-tuned weights		Fixed weights	Fine-tuned weights	
		Linear	ReLU		Linear	ReLU
MNIST	1.11	**1.10**	1.25	1.34	**1.22**	1.41
synthetic $\rho = 10^{-5}$, $\psi = 0.2$	27.49	**26.55**	30.59	20.42	**20.16**	27.16
webspam	0.36	**0.35**	0.38	0.32	**0.25**	0.33
url	3.75	**3.30**	3.78	0.96	**0.75**	0.81

For comparison, we also report errors for networks with fixed-weight random projection layer. For each dataset and random projection scheme we highlight the best performing network architecture

is to distinguish between malicious and benign addresses. Following [36], we used examples from the first 100 days of data collection as the training set and remaining examples for testing. KDD2010-a and KDD2010-b are large student performance prediction datasets from the KDD Cup 2010 challenge. We used pre-processed versions of these datasets made available by the challenge winner [40] and adopted the provided train/test set split. webspam and KDD2010 datasets are available from the LIBSVM datasets webpage[3]. For all four datasets, we randomly projected the input vectors to 1000 dimensions and trained a network with two hidden layers, each one with 3000 neurons. Each network was pretrained and then fine-tuned to minimize the binary cross-entropy cost.

Overall, networks with fixed-weight RP layer significantly improved over the current state-of-the-art results on the url dataset and achieved competitive performance on webspam and KDD2010 datasets (Table 3). Gaussian, Achlioptas' and SRHT projections performed similarly well in these experiments, while Li's method performed the worst. Count Sketch was among the best performing projections for datasets with density between $10^{-5} - 10^{-4}$ (webspam and url), while for sparser datasets it yielded results similar to, and in the case of KDD2010-a even worse than, Gaussian, Achlioptas' and SRHT. This agrees with the results from experiments on the synthetic datasets (Sect. 4.1.1).

4.2 Fine-tuned random projection layer

We evaluated the performance of fine-tuned RP layer on several large-scale datasets: a variant of the synthetic dataset with density $\rho = 10^{-5}$ and the fraction of significant features $\psi = 0.2$, webspam dataset, and url dataset. Additionally, we report results on a toy benchmark—MNIST. We employed network architectures with the same activation functions, number of neurons and number hidden layers as in the experiments with fixed-weight random projection. We also employed the same training setup but performed additional validation experiments to choose the learning rate and L2 cost hyperparameters. In addition to experiments with linear random projection, we also investigated architectures with the ReLU nonlinearity after batch normalization. In experiments with MNIST and synthetic datasets we updated the parameters in the RP layer for every training mini-batch. In experiments with larger datasets, i.e. url and webspam, we updated the RP weights for randomly selected 50% of mini-batches.

Table 4 reports the early stopping errors achieved by networks with fine-tuned RP layer. Compared to networks with fixed-weight RP layer, networks with fine-tuned linear random projection performed better on all datasets. Importantly, they further improved the state-of-the-art results on webspam and url datasets. However, our results also show that introducing a nonlinearity after the RP layer decreased the network performance. In fact, networks where fine-tuned random projection was followed by ReLU nonlinearity performed very similarly to, or were outperformed by, networks with fixed-weight RP layer. We hypothesize that this poor performance of random projection with ReLU nonlinearity is a consequence of a small size of the RP layer output. (Because of the computational cost, in our main experiments we limited the output of fine-tuned random projection layer to 1000 dimensions.) Note that input and output units in the fine-tuned RP layer are sparsely connected. Therefore, when the RP layer processes a sparse training example, the total input to the nonlinearity is also sparse. If we apply ReLU activation, we effectively zero-out, on average, half of the elements in the sparse input. We believe that this loss of information causes the decrease in network performance. If our hypothesis is correct, random projection with ReLU nonlinearity should perform better with larger output dimensionality.

To verify this hypothesis, we performed additional experiments with larger RP layers. In particular, we experimented

[3] Available at https://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets/.

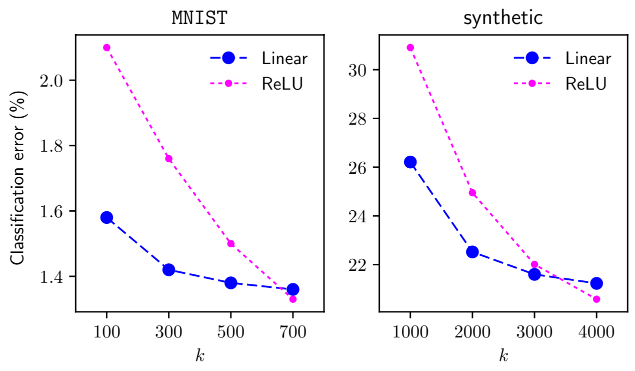

Fig. 5 Performance of networks with fine-tuned RP layer for different activation functions and output dimensionality (k)

with Li's RP layer on the MNIST and synthetic datasets. In experiments on MNIST, we trained networks with 784-k-300-10 architectures, for $k \in \{100, 300, 500, 700\}$, and in experiments on the synthetic dataset we used 10^6-k-3000-1 architecture for $k \in \{1000, 2000, 3000, 4000\}$. We employed the same training settings as in the previous experiments. For each k and each activation function, we selected the learning hyperparameters with experiments on the validation sets. Figure 5 presents the early stopping errors for networks with different activation functions after the RP layer and varying RP layer size. Our results suggest that introducing the ReLU activation function after the RP layer can improve the network performance, provided that the dimensionality of the RP layer is sufficiently high. In our experiments on the synthetic dataset, it was necessary to use 4000 units in the RP layer to make ReLU viable. However, such a large RP layer greatly increases the overall computational cost of training. Therefore, for practical applications involving large, high-dimensional data we recommend using networks with fine-tuned linear random projection.

5 Related work

The idea of using fixed random weights in neural networks is not new and has been incorporated into different models proposed throughout the years. Note, however, that not every layer with random weights realizes a random projection. One important family of shallow networks employing random weights are the random weight feedforward neural networks (RW-FNNs). These models differ from our approach in two important aspects. First, instead of lowering the input data dimensionality, they transform the input data into a higher-dimensional space in which learning should, theoretically, be easier. Importantly, this transformation is most often nonlinear and, in general, does not preserve the distances between training examples. Additionally, after randomly transforming the input, RW-FNNs do not employ any feature normalization. Second, RW-FNNs cast the weight optimization problem as a standard regularized least-squares problem, which can be solved analytically in a single step. While this approach offers a computational advantage compared to stochastic gradient descent, it is suitable only for networks with a single hidden layer. For a more comprehensive overview of RW-FNNs see [41]. Predecessors of these models were proposed in a number of early works on feedforward architectures, e.g. in [42, 43]. A more mature version of RW-FNNs, called Random Vector Functional-Link (RVFL) networks were introduced in [44, 45].

Arriaga and Vempala [18] suggested that the human brain may reduce the amount of information generated by visual stimuli in a process that resembles random projection. They showed that RP can be realized by a shallow neural network with weights drawn from a Gaussian distribution or just set randomly to - 1 or 1 (note that this is a denser variant of the Achlioptas' construction [7]). Arriaga and Vempala used their so-called neuron-friendly RP to show that efficient learning is possible in the projected space. However, similarly to RW-FNNs, they did not train deeper models on the projected data and used a simple learning algorithm instead of error backpropagation.

To the best of our knowledge, the only attempt at training DNNs on randomly projected data, and therefore the approach that is most relevant to our fixed-weight RP layers, was presented in [46]. Therein, Dahl et al. used randomly projected data as input to networks trained for the malware classification task. Specifically, they projected the original 179, 000-dimensional data (trigrams of system API calls) to 4000 dimensions and used the projected data to train a neural network with two hidden layers. With this approach, they achieved 43% relative improvement in classification performance, compared to logistic regression trained on the unprojected data. However, their classification task was fairly simple, with the classes being nearly linearly separable. Unfortunately, Dahl et al. only evaluated Li's random matrix construction [8], which is extremely sparse and, from our experience, is unsuited for projecting sparse data. It is also worth mentioning that in their experiments unsupervised pretraining did not improve network performance, unlike in experiments reported in our work. Finally, Dahl et al. evaluated only networks with the sigmoid activation function and do not report results for the currently state-of-the-art ReLU activation.

Random weight matrices were also used in certain convolutional neural network architectures [47]. In particular, Saxe et al. reported convolutional networks with random weights that performed only slightly worse than networks with learned parameters. Finally, RP was studied as a preprocessing step for SVM models. In particular, several RP schemes were evaluated in [48] as an input to SVM classification and regression, yielding promising results on small- and medium-size datasets. Similarly to our results, they also found Count Sketch to be one of the best performing RP methods.

To the best of our knowledge this work is the first systematic evaluation of different RP schemes for training deep networks on sparse high-dimensional data. Unlike several of the related works, we focus on reducing the dimensionality of data in order to make the training task feasible. Furthermore, we investigate the balance between the density of the RP matrix, which has a significant impact on the computational cost, and the final network performance. Finally, we investigate network architectures where the RP weights are fine-tuned during training.

6 Conclusions

In this work, we studied the viability of training deep neural networks with random projection layer. Our results demonstrate that networks with RP layer can match or improve the state-of-the-art classification performance on data with millions of dimensions and no spatial structure. This opens a path to applying deep networks to tasks where directly learning from the data would be infeasible: experiments on the `KDD2010` datasets, for example, involved up to 30, 000-fold reduction of the input dimensionality.

We studied two variants of the random projection layer: one with RP weights that are fixed during training and one where they are fine-tuned with error backpropagation. Our experimental evaluation of fixed-weight RP layers shows that Gaussian, Achlioptas', SRHT and Count Sketch projections perform well, while the Li's projection yields worse results. This could be attributed to the sparsity of the projected data—on the `MNIST` dataset, which is dense, Li's method performed well. Note also that Achlioptas', Count Sketch, Li's and SRHT are fast: first three do not employ dense projection matrices and the last can be computed efficiently using a transform similar to the fast Fourier transform. Taking this into account, SRHT and Count Sketch projections combine the best network performance with efficient data projection. Apart from the results reported in this work, we also experimented with using fixed-weight RP for bag-of-words (BOW) data. Specifically, we experimented with training deep autoencoders similar to the ones described in [49] on randomly projected BOW vectors. While this approach enabled us to train autoencoders on larger dictionaries, it did not achieve performance comparable to the reference networks. This result can be a consequence of two facts. First, the autoencoders with projected data require Gaussian input units. The reference networks employ the constrained Poisson model, which is tailored to BOW data. Second, the dictionary used by the reference models already captured most of the word count in the text.

Our experiments with fine-tuned random projection suggest that adjusting the nonzero weights in a sparse RP layer can significantly improve the overall network performance. In particular, by using the fine-tuned Count Sketch RP layer, we were able to train networks that achieved more than 30% lower classification error than the previously state-of-the-art methods on `webspam` and `url` datasets. To make the task of training the RP layer feasible, we employed several architectural and training modifications. First, instead of normalizing the input data we applied batch normalization after the random projection layer. Second, we fine-tuned only these RP weights that were initially nonzero. Finally, we found that applying a nonlinear activation function after the batch normalization is viable only when the input is projected to high-dimensional space. In practice, the performance gain from this nonlinearity does not justify the additional cost introduced by fine-tuning an RP layer with a high-dimensional output.

Training deep networks with random projection layer is more computationally expensive than training linear classifiers, such as logistic regression or support vector machines. However, with an already trained model the inference time is small: feeding a training example through the RP layer can be realized by a single matrix multiplication. By using fast random projection schemes, this operation can be performed in nearly linear or linear time ($\mathcal{O}(d \log k)$ for SRHT and $\mathcal{O}(d)$ for Count Sketch). The operations in subsequent layers can be implemented efficiently on modern hardware. Therefore, despite the computational cost of training, neural networks with random projection layer can be used to solve practical problems.

Acknowledgements This research is supported by the Polish National Science Centre grant no. DEC-2013/09/B/ST6/01549 "Interactive Visual Text Analytics (IVTA): Development of novel, user-driven text mining and visualization methods for large text corpora exploration." This research was carried out with the support of the "HPC Infrastructure for Grand Challenges of Science and Engineering" Project, co-financed by the European Regional Development Fund under the Innovative Economy Operational Programme. This research was supported in part by PL-Grid Infrastructure. We would like to thank prof. Witold Dzwinel for helpful comments and suggestions during the work on this article.

References

1. LeCun Y, Bengio Y, Hinton G (2015) Deep learning. Nature 521(7553):436–444
2. Graves A, Mohamed A, Hinton G (2013) Speech recognition with deep recurrent neural networks. In: Proceedings of 2013 IEEE international conference on acoustics, speech and signal processing (ICASSP). IEEE, pp 6645–6649
3. Yuan G-X, Ho C-H, Lin C-J (2012) Recent advances of large-scale linear classification. Proceedings of the IEEE 100(9):2584–2603
4. Johnson WB, Lindenstrauss J (1984) Extensions of Lipschitz mappings into a Hilbert space. Contemp Math 26:189–206
5. Indyk P, Motwani R (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the 13th annual ACM symposium on theory of computing. ACM, pp 604–613
6. Dasgupta S, Gupta A (2003) An elementary proof of a theorem of Johnson and Lindenstrauss. Random Struct Algorithms 22(1):60–65
7. Achlioptas D (2001)Database-friendly random projections. In: Proceedings of the 20th ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems. ACM, pp 274–281
8. Li P, Hastie TJ, Church KW (2006) Very sparse random projections. In: Proceedings of the 12th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 287–296
9. Ailon N, Chazelle B (2006) Approximate nearest neighbors and the fast Johnson–Lindenstrauss transform. In: Proceedings of the 38th annual ACM symposium on theory of computing. ACM, pp 557–563
10. Ailon N, Liberty E (2009) Fast dimension reduction using Rademacher series on dual BCH codes. Discrete Comput Geom 42(4):615–630
11. Charikar M, Chen K, Farach-Colton M (2004) Finding frequent items in data streams. Theor Comput Sci 312(1):3–15
12. Weinberger K, Dasgupta A, Langford J, Smola A, Attenberg J (2009) Feature hashing for large scale multitask learning. In: Proceedings of the 26th annual international conference on machine learning (ICML'09). ACM, pp 1113–1120
13. Shi Q, Petterson J, Dror G, Langford J, Smola A, Vishwanathan SVN (2009) Hash kernels for structured data. J Mach Learn Res 10:2615–2637
14. Dasgupta A, Kumar R, Sarlós T (2010) A sparse Johnson–Lindenstrauss transform. In: Proceedings of the 42nd annual ACM symposium on theory of computing. ACM, pp 341–350
15. Clarkson KL, Woodruff DP (2013) Low rank approximation and regression in input sparsity time. In: Proceedings of the 45th annual ACM symposium on theory of computing. ACM, pp 81–90
16. Meng X, Mahoney MW (2013) Low-distortion subspace embeddings in input-sparsity time and applications to robust linear regression. In: Proceedings of the 45th annual ACM symposium on theory of computing. ACM, pp 91–100
17. Nelson J, Nguyên HL (2013) OSNAP: faster numerical linear algebra algorithms via sparser subspace embeddings. In: Proceedings of the 54th annual IEEE symposium on foundations of computer science. IEEE, pp 117–126
18. Arriaga RI, Vempala S (2006) An algorithmic theory of learning: robust concepts and random projection. Mach Learn 63(2):161–182
19. Hegde C, Davenport MA, Wakin MB, Baraniuk RG (2007) Efficient machine learning using random projections. In: Proceedings of the NIPS workshop on efficient machine learning
20. Hinton GE, Salakhutdinov RR (2006) Reducing the dimensionality of data with neural networks. Science 313(5786):504–507
21. Welling M, Rosen-Zvi M, Hinton GE (2004) Exponential family harmoniums with an application to information retrieval. In: Advances in neural information processing systems 17 (NIPS'04). MIT Press, pp 1481–1488
22. Bank RE, Douglas CC (1993) Sparse matrix multiplication package (SMMP). Adv Comput Math 1(1):127–137
23. Greiner G et al (2012) Sparse matrix computations and their I/O complexity. Ph.D. thesis, Dissertation, Technische Universität München, München
24. Nelson J, Nguyễn HL (2014) Lower bounds for oblivious subspace embeddings. In: International colloquium on automata, languages, and programming. Springer, pp 883–894
25. Coates A, Huval B, Wang T, Wu D, Catanzaro B, Andrew N (2013) Deep learning with cots HPC systems. In: Proceedings of the 30th international conference on machine learning (ICML'13). PMLR, pp 1337–1345
26. Ioffe S, Szegedy C (2015) Batch normalization: accelerating deep network training by reducing internal covariate shift. In: Proceedings of the 32nd international conference on machine learning (ICML'15). PMLR, pp 448–456
27. Srivastava N, Hinton GE, Krizhevsky A, Sutskever I, Salakhutdinov R (2014) Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res 15(1):1929–1958
28. Nair V, Hinton GE (2010) Rectified linear units improve restricted Boltzmann machines. In: Fürnkranz J, Joachims T (eds) Proceedings of the 27th international conference on machine learning (ICML'10). Omnipress, pp 807–814
29. Grzegorczyk K, Kurdziel M, Wójcik PI (2016) Implementing deep learning algorithms on graphics processor units. In: Parallel processing and applied mathematics: 11th international conference (PPAM2015). Springer, pp 473–482
30. Fan R-E, Chang K-W, Hsieh C-J, Wang X-R, Lin C-J (2008) Liblinear: a library for large linear classification. J Mach Learn Res 9:1871–1874
31. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
32. Mishkin D, Matas J (2005) All you need is a good init. arXiv preprint arXiv:1511.06422
33. Yuan G-X, Ho C-H, Lin C-J (2012) An improved glmnet for l1-regularized logistic regression. J Mach Learn Res 13(1):1999–2030
34. Yuan G-X, Ma K-L (2012) Scalable training of sparse linear svms. In: Proceedings of 2012 IEEE 12th international conference on data mining (ICDM). IEEE, pp 775–784
35. Yang H, Wu J (2012) Practical large scale classification with additive kernels. In: Proceedings of 4th Asian conference on machine learning, pp 523–538
36. Wang Z, Djuric N, Crammer K, Vucetic S (2011) Trading representability for scalability: adaptive multi-hyperplane machine for nonlinear classification. In: Proceedings of the 17th ACM SIGKDD international conference on knowledge discovery and data mining. ACM, pp 24–32
37. Zhang C, Lee H, Shin KG (2012) Efficient distributed linear classification algorithms via the alternating direction method of multipliers. In: Proceedings of the 15th international conference on artificial intelligence and statistics (AISTATS 2012). PMLR, pp 1398–1406
38. Webb S, Caverlee J, Pu C (2006) Introducing the Webb Spam Corpus: using email spam to identify web spam automatically. In: Proceedings of the 3rd conference on email and anti-Spam (CEAS)
39. Ma J, Saul LK, Savage S, Voelker GM (2009) Identifying suspicious URLs: an application of large-scale online learning. In: Bottou L, Littman M (eds) Proceedings of the 26th international

conference on machine learning (ICML'09). Omnipress, pp 681–688

40. Yu H-F, Lo H-Y, Hsieh H-P, Lou J-K, McKenzie TG , Chou J-W, Chung P-H, Ho C-H, Chang C-F, Wei Y-H et al (2010) Feature engineering and classifier ensemble for KDD Cup 2010. In: Proceedings of the KDD Cup 2010 workshop, pp 1–16
41. Scardapane S, Wang D (2017) Randomness in neural networks: an overview. Wiley Interdiscip Rev Data Min Knowl Discov 7(2):1–18
42. Gallant S, Smith D (1987) Random cells: an idea whose time has come and gone... and come again. In: Proceeding of the 1987 IEEE international conference on neural networks. IEEE, pp 671–678
43. Schmidt WF, Kraaijveld MA, Duin RPW (1992) Feedforward neural networks with random weights. In: Proceedings of the 11th IAPR international conference on pattern recognition (IAPR). IEEE, pp 1–4
44. Pao Y-H, Takefuji Y (1992) Functional-link net computing: theory, system architecture, and functionalities. Computer 25(5):76–79
45. Yoh-Han P, Park G-H (1994) Learning and generalization characteristics of the random vector functional-link net. Neurocomputing 6(2):163–180
46. Dahl GE, Stokes JW, Deng L, Yu D (2013) Large-scale malware classification using random projections and neural networks. In: Proceedings of 2013 IEEE international conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE, pp 3422–3426
47. Saxe A, Koh PW, Chen Z, Bhand M, Suresh B, Ng AY (2011) On random weights and unsupervised feature learning. In: Proceedings of the 28th international conference on machine learning (ICML'11). Omnipress, pp 1089–1096
48. Paul S, Boutsidis C, Magdon-Ismail M, Drineas P (2014) Random projections for linear support vector machines. ACM Trans Knowl Discov Data (TKDD) 8(4):22
49. Salakhutdinov R, Hinton GE (2009) Semantic hashing. Int J Approx Reason 50(7):969–978