Neural network pruning with simultaneous matrix tri-factorization

Anonymous Author(s)
Affiliation
Address
email

Abstract

In this paper we present a novel approach for pruning neural network to reduce the model size and achieve better generalization performance. We apply a simultaneous matrix tri-factorization to map weight matrices to a low-dimensional space, therefore shrinking them and partially eliminating noise. Factorized models are thus more robust and have a better generalization ability...

1 Introduction

Deep neural networks are a popular tool used to solve a certain problem. The advantages of neural networks are that they are relatively easy to use and can approximate any function, regardless of its linearity. They are widely used for complex or abstract problems such as image, sound and text recognition. However, they are computationally intensive to train and are known for black box problem as they will not tell you why they reached a certain conclusion. Success of neural networks largely depends on their architecture. While the size of the input layer and the output layer is known, the number of hidden layers and the number of nodes in each hidden layer depends on the complexity of the problem [1]. Generally, a network with large number of hidden nodes is able to learn fast and avoids local minima, but when a network is oversized, the network may overfit the training data and lose its generalization ability while still having unnecessary calculations as they are using more nodes than necessary. Better generalization performance can be achieved only by small networks. They are easier to interpret but their training may require a lot of effort. Also too small networks are very sensitive to initial conditions and learning parameters and are prone to not learn the given problem. The most popular approach to obtain the most optimal architecture of neural network is pruning. Pruning is defined as a network trimming within the assumed initial architecture, which is larger than necessary. Pruning algorithms are used to remove the redundant connections while maintaining the networks performance. So one can use the larger networks for training and its generalization can be improved by the process of pruning [1].

More recent researches have tackled upon an issue of deep neural network and deep convolutional neural networks which is that they involve many layers with millions of parameters, making the size of the network model to be extremely large to store. This prohibits the usage on resource limited hardware especially mobile devices or other embedded devices even though deep neural networks are increasingly used in applications suited for mobile devices [11].

In this work we present a novel approach using low-dimensional matrix factorization. Because we have more than one weight matrix and because the weight matrices between the layers in a neural network are dependent with their neighbor matrices, we used an upgraded approach of matrix factorization, named simultaneous matrix tri-factorization, or in other words, data fusion. Pruning neural network with simultaneous matrix tri-factorization was names as matrix factorization-based brain pruning (MFBP).

2 Related work

In article [7] they said that giving only a few weight values for each feature it is possible to accurately predict the remaining values while many of them do not need to be learned at all. They exploited the fact that the weights in learned networks tend to be sparse and structured. Another article [1] have shown that, in any case the overall time required for training a large network and then pruning it to a small size compares very favorably with that of simply training a small network.

Because there is significant redundancy in the parametrization of networks, many researchers found solutions to prune neural networks with possible accuracy loss in order to reduce the model size extensively. But were able to fine-tune the compressed layers with added learning iterations to recover the performance and improve the accuracy back.

Compressing the most storage demanding dense connected layers is possible by neural network pruning with low-rank matrix factorization methods [4, 26, 25], where network pruning has been used both to reduce model size and to reduce over-fitting [12]. State-of-the-art approaches are Optimal Brain Damage [18] and Optimal Brain Surgeon [13] which open the rich field of studies using matrix factorization to prune the networks.

Besides neural network pruning with matrix factorization many alternatives have been used in numerous ways to optimize neural network architecture. One of the latest study [11]used vector quantization methods for which they said have a clear gain over existing matrix factorization methods. Alternative approach [30]is application of singular value decomposition (SVD) on the weight matrices to decompose them and reconstruct the model based on the sparseness of the original matrices. There were also studies which used evolutionary pruning, more precisely, Genetic Algorithms [19] to examine potential redundancy in data and therefore prune the neural network. A simple solution to reduce the model size and preserve the generalization ability is to train models that have a constant number of simpler neurons which was presented in article [6].

Another examined strong method uses the significance of neurons by evaluating the information on weight variation and consequently prune the insignificant nodes. Removing all connections whose weight is lower than a threshold is introduced in [12]. There the first phase learns which connections are important and removes the unimportant ones using multiple iterations. Hashing is also an effective strategy for dimensionality reduction while preserving generalization performance [29, 27]. The strategy used on neural networks named HashedNets [5] uses a low-cost hash function to randomly group connection weights into hash buckets where all connection inside share a single and tuned parameter value.

Compressing the parameters to reduce model size brings the focus upon how to prune the dense connected layers since the vast majority of weights reside in these layers which results in significant savings and by replacing the fully connected layers of the network with an Adaptive Fastfood transform, introduced in article [31], and results in a deep fried convnet. The Fastfood transform allows for a theoretical reduction in computation also. However, the computation in convolutional neural networks is dominated by the convolutions, and hence the deep fried convnets are not necessarily faster in practice.

3 Method description

Deep neural network is a feed-forward, artificial neural network with more than one or two hidden layers between the input and output layer. Figure 1 shows the structure of a deep neural network. The number of nodes on the left layer, known as the input layer, is the same as the number of attributes in a dataset while the number of nodes in the most right layer, named as output layer is the same as the number of classes. Between them are the hidden layers as mentioned above. The number of nodes in every hidden layer is depended of the dataset we used. Our main goal was to use more nodes than necessary to prune the unnecessary ones later.

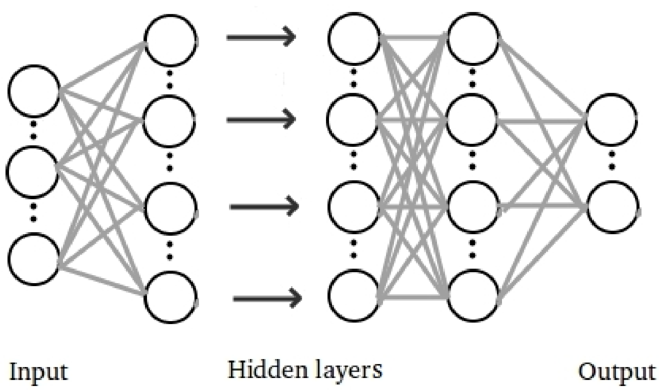

Figure 1: Deep neural network with densed connected layers.

3.1 Approximation of matrix with matrix factorization

Matrix factorization is a technique to search linear representation with factorizing. Approximation of matrix with matrix factorization is used to approximate the data in low-dimensional space in order to find latent features. The theorem of the method matrix factorization follows:

Izrek: Given is a matrix $A \in \Re^{m\times n}$ and a positive integer $k \ll min\{m, n\}$. Find matrices $W \in \Re^{m\times k}$ and $H \in \Re^{k\times n}$, such that $WH \approx A$. the product WH is named as a matrix factorization (MF) of a matrix A. WH is an approximated factorization if rang $r = k$.

We assume that factorized matrix W has less columns than the original matrix A has rows. Approximation is successful only when the latent structure is found and the dimensionality of data gets lower. Applications, such as text processing, image processing and data mining store valuable information in very big matrices. Not only that low rank matrix factorization reduces the consumption of memory but also offers more clear and more efficient representation of connections between data [22]. A low rank approximation finds the key components of data and discards the data that attribute to noise, error or inconsistency [16]. Because of the potential of matrix factorization, a lot of upgraded versions were created. One of them is a simultaneous matrix factorization [33].

3.2 Simultaneous matrix tri-factorization

Izrek: Simultaneous tri-factorization of multiple matrices simultaneously factorize all available relation matrices R_{ij} v $G_i\Re^{n_i\times k_i}$, $G_jE\Re^{n_j\times k_j}$ in $S\Re^{k_i\times k_j}$ and regulize their approximations through constrained matrices θ_i and θ_j, such that $R_{ij} \approx G_i S_{ij} G_j^T$ [33].

Simultaneous matrix tri-factorization focuses on a specific target relation and exploits directly connected data. It contains a set of restrictions of the connections that should connect and of those which should not. In that way it contains the relations between the objects of the same type [33]. It finds latent associations between matrices with simultaneous factorization of matrices, therefore directly considers every dataset that can be represented as a matrix [33].

The data fusion algorithm considers r object types $\xi_1, , \xi_r$ and a collection of data sources, each relating a pair of object types (ξ_i, ξ_j). If there are n_i objects of type ξ_i and n_j objects of type ξ_j, it represents the observations from the data source that relates (ξ_i, ξ_j) or $i \neq j$ in a sparse matrix $R_{ij}\Re^{n_i\times n_j}$. In general matrices R_{ij} and R_{ji} need not be symmetric. A data source that provides relations between objects of the same type ξ_i is represented by a constrant matrix $\theta_i \in \Re^{n_i\times n_i}$ [33]. Even if we are missing relations between some pairs, the algorithm still integrates all available data if an underlying graph of relations between object types is connected. Figure 2 shows an example of a block configuration of the fusion system with four object types [33].

To retain the block structure of a system, it is proposed the simultaneous matrix tri-factorization of all relation matrices R_{ij} constrained by θ_i. The resulting system contains factors that are specific to

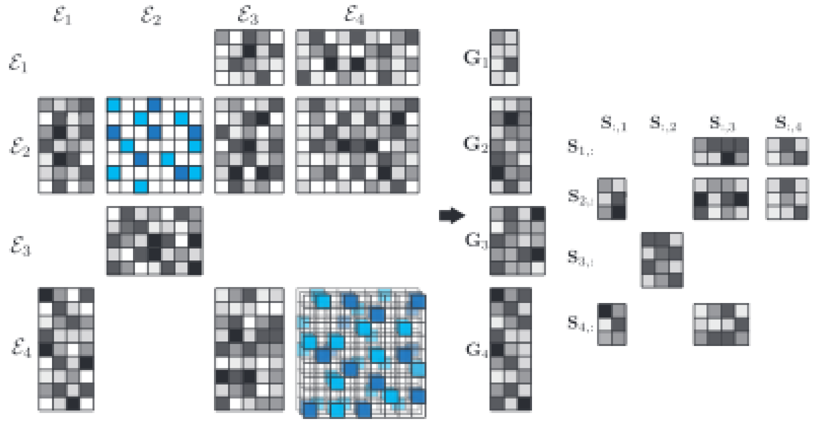

Figure 2: An example of simultaneous matrix tri-factorization. On the left side are all available relation matrices R. They are factorized to matrices G and S (on the right). Source [33].

each data source and factors that are specific to each object type. Through factor sharing it fuses the data but also identifies source-specific patterns [33].

3.3 Matrix factorization-based brain pruning

With ordinary artificial neural network, we have only one hidden layer and therefore two weight matrices with the same object type (sharing dimension). Because of this property, we are able to concatenate the matrices through sharing dimension and apply a matrix factorization. With deep neural networks we have a multi-layer architecture where neighbor weight matrices share the same object type (same dimension). We can apply co-dependency between neighbor weight matrices but we can apply dependency between, for example, first and third weight matrix. With simultaneous matrix tri-factorization we can solve the mentioned problem and pruning neural networks with simultaneous matrix tri-factorization named matrix factorization-based brain pruning (MFBP).

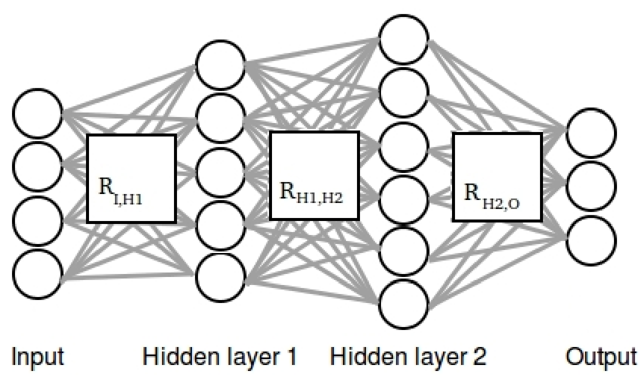

Figure 3: Artificial two-layered neural network with weight matrices (relation matrices R). The size od each matrix is dependent by number of neurons on surrounding layers. For example $R_{I,H1}$ has four (I) rows and five ($H1$) columns.

In a figure 3 is shown a neural network with two hidden layers and relation weight matrices R between them. The weight matrices are collected from neural network and configured in a graph of relations as shown in figure 4. The graph of relations is then used in data fusion. The result are approximations of the weight matrices. With approximations we determined which weights are

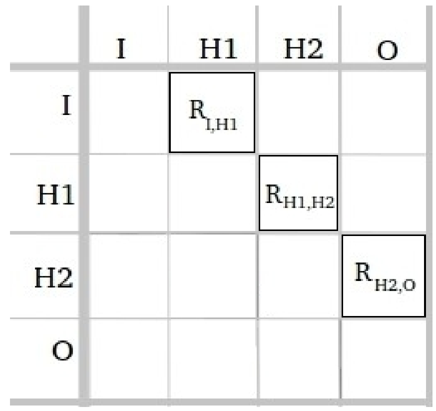

Figure 4: Configuration of relation matrices R from figure 3 for simultaneous matrix tri-factorization. In our case, the relation matrices R present weight matrices of neural network. Configuration is set on diagonal because the neighbour weight matrices share the dimension from shared hidden layer

better to prune. In theory it is said that a weight is pruned, when its value equals zero. But because in practice the weights are almost always non-zero, we have to estimate which weights to prune.

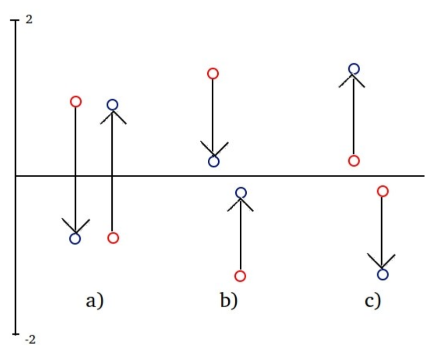

Figure 5: Possible changes of values of weights. Values before pruning (red circle) and after pruning (blue circle). In a) case, the weight value changes sign, b) moves closer to zero and c) moves away from zero.

In figure 5 are shown possible changes that can happen after data fusion. A weight can change its sign, can move closer to zero or move away from it. We chose the weights where absolute value move closer to zero for enough amount (for 0.2 in our case) for pruning. We set the chosen weights to zero and keep them at that value. Below is shown a pseudocode of our algorithm.

Pseudocode

4 Experimental setup

We evaluated matrix factorization-based brain pruning on MNIST (Mixed National Institute of Standards and Technology dataset) dataset. The MNIST database of handwritten digits 0-9, available in [17], has a training set of 60,000 instances and a test set of 10,000 instances. The digits have been size-normalized and centered in a fixed-size 28x28 images.

We used a modern neural network, presented in [21]. There are two main contributions to a modern neural network. One is changing of activation function. Instead of sigmoid function it uses a rectifier (Rectified linear unit (ReLU) $f(x) = max(0, x)$, where x is the input to a neuron. What the rectifier does is that if the input to a neuron is below zero the activation function does nothing. If the input is above zero it does activate. This activation function has been argued to be more biologically plausible [9]. It induces the sparsity in the hidden neurons. Another advantage of rectifier is that it does not face gradient vanishing problem as with sigmoid or tanh function. It has ben also shown that can be deep neural networks trained efficiently using rectifier even without pre-training. The other contribution is regularizing the model with dropout [28]. Dropout is one of the biggest improvements in the field of neural networks in recent years as it addresses the main problem in deep learning that is overfitting. The purpose of dropout is to add some noise by dropping out a random number of some neuron activations in a given layer. By dropping them is meant to set them to zero or as in our case to prune them. With every iteration a different random set of neurons are chosen to drop, therefore it prevents co-adaptation of neurons. There was also a change at update rule. Instead of a standard stochastic gradient descent (SGD) backpropagation method we used RMSprop (A mini-batch version of rpop). The idea behind SGD is to approximate the real update step by taking the average of the all given instances or as in our case mini batches. The problem of SGD is that it is sensitive to outliers which can destroy all the gradient information collected before [10]. On the other hand, the RMSprop keeps a running average of its recent gradient magnitudes and divides the next gradient by this average so that loosely gradient values are normalized [14]. RMSprop follows: $MeanSquare(w,t) = 0.9MeanSquare(w,t-1) + 0.1(\delta E/\delta w^{(t)})^2$ [14].

To evaluate our experiments, we implemented algorithm on Python with the help of Theano [2, 3]. Theano is a Python library that is suitable for building an optimized neural network. We chose it as it gives a comprehensive control over neural network formation which is suitable for our problem. Another reason we used Theano is because the implementation of modern neural net described above is available online as open source. Data fusion algorithm which performs simultaneous matrix tri-factorization is available in a python library Scikit-fusion [33]. To measure our results, we used a machine learning library Scikit-learn [23].

To estimate and analyze our results, we trained and tested four neural networks: ordinary neural network with two hidden layers without pruning, ordinary neural network with two hidden layers with pruning, deep neural network with five hidden layers without pruning and another deep neural network with five hidden layers with pruning. Every neural network had 60 iterations available to learn. The non-pruned networks learned on 60 iterations where every iteration had 50,000 train instances packed in mini-batches of 128. The pruned network had 40 iterations to learn without pruning. The other 20 iterations consisted of every second iteration of pruning (overall 10 iterations of pruning) and another half of iterations for fine-tuning. Fine-tuning was used to adapt the non-pruned weights which have been affected with pruning, in other words, to recover the non-pruned weight values which have been biased by the pruned weights before pruning. In this case there were also all 50,000 train instances available at every iteration in mini-batches.

5 Results

The reported results are measured with area under ROC curve (AC) on test set and shown in figure 6. The model size compression rate in % results are coming...

6 Discussion and conclusion

To be written...

Acknowledgments

References

References

[1] M Gethsiyal Augasta and T Kathirvalavakumar. Pruning algorithms of neural networksa comparative study. *Central European Journal of Computer Science*, 3(3):105–115, 2013.

[2] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian J. Goodfellow, Arnaud Bergeron, Nicolas Bouchard, and Yoshua Bengio. Theano: new features and speed improvements. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

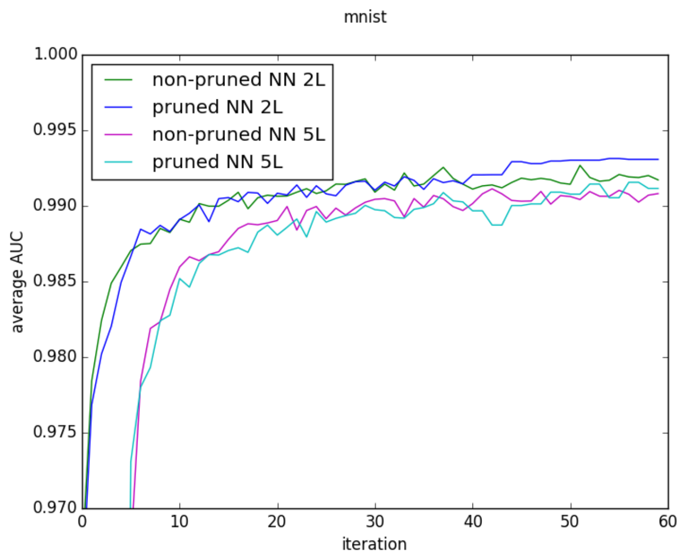

Figure 6: AUC results of neural networks. With pruned neural networks, the pruning starts at 40th iteration.

[3] James Bergstra, Olivier Breuleux, Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, Guillaume Desjardins, Joseph Turian, David Warde-Farley, and Yoshua Bengio. Theano: a CPU and GPU math expression compiler. In *Proceedings of the Python for Scientific Computing Conference (SciPy)*, June 2010. Oral Presentation.

[4] Andrey Bondarenko and Arkady Borisov. Artificial neural network generalization and simplification via pruning. *Information Technology and Management Science*, 17(1):132–137, 2014.

[5] Wenlin Chen, James T Wilson, Stephen Tyree, Kilian Q Weinberger, and Yixin Chen. Compressing neural networks with the hashing trick. *arXiv preprint arXiv:1504.04788*, 2015.

[6] Maxwell D Collins and Pushmeet Kohli. Memory bounded deep convolutional networks. *arXiv preprint arXiv:1412.1442*, 2014.

[7] Misha Denil, Babak Shakibi, Laurent Dinh, Nando de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

[8] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

[9] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In Geoffrey J. Gordon and David B. Dunson, editors, *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics (AISTATS-11)*, volume 15, pages 315–323. Journal of Machine Learning Research - Workshop and Conference Proceedings, 2011.

[10] Eren Golge. Comparison: Sgd vs momentum vs rmsprop vd momentum+rmsprop vs adagrad, 2015. Avaliable: `http://www.erogol.com/` [Reached 17.1.2016].

[11] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir D. Bourdev. Compressing deep convolutional networks using vector quantization. *CoRR*, abs/1412.6115, 2014.

[12] Song Han, Jeff Pool, John Tran, and William J Dally. Learning both weights and connections for efficient neural networks. *arXiv preprint arXiv:1506.02626*, 2015.

[13] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *Neural Networks, 1993., IEEE International Conference on*, pages 293–299. IEEE, 1993.

[14] Geoffrey Hinton. Lecture 6a overview of mini-batch gradient descent, 2014. Avaliable: `www.cs.toronto.edu/` [Reached 17.1.2016].

[15] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[16] Amy N Langville, Carl D Meyer, Russell Albright, James Cox, and David Duling. Algorithms, initializations, and convergence for the nonnegative matrix factorization. *arXiv preprint arXiv:1407.7299*, 2014.

[17] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[18] Yann LeCun, John S Denker, Sara A Solla, Richard E Howard, and Lawrence D Jackel. Optimal brain damage. In *NIPs*, volume 89, 1989.

[19] Xiangmei Li. Tuning the structure and parameters of a neural network by a new network model based on genetic algorithms. *International Journal of Digital Content Technology & its Applications*, 6(11), 2012.

[20] Michael Mathieu, Mikael Henaff, and Yann LeCun. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*, 2013.

[21] Alec Radford (newmu). Theano-tutorials, 2015. Avaliable: `http://www.github.com` [Reached 27.1.2016].

[22] Behnam Neyshabur and Rina Panigrahy. Sparse matrix factorization. *arXiv preprint arXiv:1311.3315*, 2013.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] Roberto Rigamonti, Amos Sironi, Vincent Lepetit, and Pascal Fua. Learning separable filters. In *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on*, pages 2754–2761. IEEE, 2013.

[25] Tara N Sainath, Brian Kingsbury, Vikas Sindhwani, Ebru Arisoy, and Bhuvana Ramabhadran. Low-rank matrix factorization for deep neural network training with high-dimensional output targets. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 6655–6659. IEEE, 2013.

[26] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117, 2015.

[27] Qinfeng Shi, James Petterson, Gideon Dror, John Langford, Alex Smola, and SVN Vishwanathan. Hash kernels for structured data. *The Journal of Machine Learning Research*, 10:2615–2637, 2009.

[28] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[29] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. Feature hashing for large scale multitask learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, pages 1113–1120. ACM, 2009.

[30] Jian Xue, Jinyu Li, and Yifan Gong. Restructuring of deep neural network acoustic models with singular value decomposition. In *INTERSPEECH*, pages 2365–2369, 2013.

[31] Zichao Yang, Marcin Moczulski, Misha Denil, Nando de Freitas, Alex Smola, Le Song, and Ziyu Wang. Deep fried convnets. *arXiv preprint arXiv:1412.7149*, 2014.

[32] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *Computer Vision–ECCV 2014*, pages 818–833. Springer, 2014.

[33] Marinka Zitnik and Blaz Zupan. Data fusion by matrix factorization. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 37(1):41–53, 2015.