# DIRECT PRODUCT BASED DEEP BELIEF NETWORKS FOR AUTOMATIC SPEECH RECOGNITION

*Petr Fousek, Steven Rennie, Pierre Dognin, Vaibhava Goel*

IBM Thomas J. Watson Research Center

petr_fousek@cz.ibm.com, {sjrennie, pdognin, vgoel}@us.ibm.com

## ABSTRACT

In this paper, we present new methods for parameterizing the connections of neural networks using sums of direct products. We show that low rank parameterizations of weight matrices are a subset of this set, and explore the theoretical and practical benefits of representing weight matrices using sums of Kronecker products. ASR results on a 50 hr subset of the English Broadcast News corpus indicate that the approach is promising. In particular, we show that a factorial network with more than *150 times* less parameters in its bottom layer than its standard unconstrained counterpart suffers minimal WER degradation, and that by using sums of Kronecker products, we can close the gap in WER performance while maintaining very significant parameter savings. In addition, direct product DBNs consistently outperform standard DBNs with the same number of parameters. These results have important implications for research on deep belief networks (DBNs). They imply that we should be able to train neural networks with thousands of neurons and minimal restrictions much more rapidly than is currently possible, and that by using sums of direct products, it will be possible to train neural networks with literally millions of neurons tractably–an exciting prospect.

***Index Terms***— Kronecker Product, Deep Belief Networks, Multi-Layer Perceptron, Back-Propagation, Matrix Factorization.

## 1. INTRODUCTION

Recently there has been a resurgence of interest in utilizing deep belief networks (DBNs) for machine learning tasks, including automatic speech recognition (ASR) [1, 2]. This resurgence has been fueled by new algorithms for DBN training [3], new results demonstrating significant performance improvements on practical tasks including ASR [1, 2], and the availability of large amounts of training data for these tasks [4]. However, despite this recent success, which has been largely facilitated by our ever-increasing computing capabilities, DBNs remain extremely time consuming to train–even "small" networks with just thousands of neurons (nodes) can take months to learn [4]. In practice, this severely restricts both the number of layers that can be utilized, and the number of neurons per DBN layer, which in turn, presumably limits their potential performance.

Currently most DBNs used in practice are trained layer by layer, and are restricted to have connections only between nodes in adjacent layers. The set of connections between any two adjacent layers (and their strength) can be and usually is represented by a "weight matrix", $W$. If layer $i$ of the network has $m$ nodes, and layer $j$ has $n$ nodes, then the connections between these two (adjacent) layers can be expressed as a $m \times n$ matrix, $W$. This weight matrix is typically either unconstrained ($m \times n$ connections) or highly constrained ahead of time (weight-tying, weight zeroing, etc.). More recently, several researchers have attempted to automatically learn the structure of the weight matrix, by imposing constraints such as sparsity [5] or low rank [2, 6].

In this paper, we present new methods for parameterizing the connections of neural networks using sums of direct products, and explore the theoretical and practical benefits of representing weight matrices using sums of Kronecker products. Preliminary ASR results on a 50 hr subset of the English Broadcast News corpus indicate that the approach is extremely promising, and has important implications for future research on DBNs. In particular, these new results imply that we can train neural networks with thousands of neurons and minimal restrictions much more rapidly than is currently possible, and that by using sums of direct products, it will be possible to train neural networks with millions of neurons.

## 2. MODEL

We are interested in modeling the connectivity between two sets of binary random variables having $M$ and $N$ nodes, respectively, via the matrix $W \in R^{M \times N}$, which is constrained to have the following form:

$$W = \sum_i A_i \odot B_i, \tag{1}$$

where $\odot$ is a cartesian product-based *direct matrix product* that maps suitably defined matrices $A_i \in R^{M_i \times N_i}$ and $B_i \in R^{O_i \times P_i}$ to $R^{M \times N}$. As discussed in [7], the set of possible direct products that may be formed by any two matrices can be fully specified via the outer, Kronecker, and recently introduced box product.

In this paper, for simplicity and concreteness, we restrict our attention to a slightly simpler form of $W$:

$$W = \sum_i A_i \otimes B_i, \tag{2}$$

where $\otimes$ is the Kronecker product, keeping in mind that the results that follow apply more or less directly to the full representation (1) [1]. This restriction immediately implies that $M_i O_i = M$ and $N_i P_i = N$. Kronecker graphs for modeling network connectivity by composed successive Kronecker products of matrices (as opposed to sums of Kronecker products), were studied in [8]. Representing the weight matrix of a neural net layer by a single Kronecker product we believe has been studied in many application contexts, but we are unaware of any studies on utilizing sums of Kronecker products to approximate weight matrices.

The principle utility of Kronecker-factorized matrices is that matrix

[1] outer products are Kronecker products of vectors, and the multiplication of a box product by a vector can be achieved by multiplying by a suitably defined Kronecker product and then transposing the result.

multiplication is efficient. It is well known that for a $M \times N$ matrix $W = A_i \otimes B_i$:

$$(A_i \otimes B_i)\text{vec}(Z) = \text{vec}(B_i Z A_i^T) \quad (3)$$

where the $\text{vec}()$ operator converts a matrix to a vector by stacking its *columns*. The complexity of matrix-vector multiplication is therefore reduced from $O(M_i O_i N_i P_i)$ to $O(M_i N_i P_i + N_i P_i O_i)$. For $M_i = N_i = O_i = P_i = sqrt(M) = sqrt(N) = K$, this implies that the complexity goes from $O(K^4)$ to $O(K^3)$.

## 3. FORWARD PROPAGATION

Propagating the input to a given DBN layer up the network involves only matrix multiplication, the addition of a vector of biases, and the application of a non-linearity:

$$z_j \equiv p(h_j|h_{j-1}) = \sigma(x_j) = \sigma(W z_{j-1} + b_j) \quad (4)$$

where $\sigma$ is typically taken to be the sigmoid or hyperbolic tangent function, and is applied elementwise to the vector $x_j$.

If $W$ is structured according to (2), (3) immediately implies that:

$$\begin{aligned} x_j &= \sum_i (A_i \otimes B_i) z_{j-1} + b_j \\ &= \sum_i \text{vec}(B_i Z_i^{j-1} A_i^T) + b_j \end{aligned} \quad (5)$$

where $Z_i^{j-1} \equiv mat_{A_i,B_i}(z_{j-1})$, denotes the matrix that results from (column-major) reshaping of the vector $x_i$ so that it can be left and right multiplied by $B_i$ and $A_i^T$, respectively. As discussed previously, the factorization of the weight matrix into sums of Kronecker products, depending on the number and dimensions of the $\{A_i, B_i\}$, can lead to substantial savings in computation and storage associated with a given network layer.

Each Kronecker product (5) implicitly makes an independence assumptions about the input (left multiplication by a matrix acts independently on rows and right multiplication independently on columns). Such a factorization is natural in many circumstances, such as for the bottom layer of multi-frame (spliced) input features, where the rows of $B_i$ and columns of $A_i$ correspond to learned frame-level and temporal patterns in the input features, respectively. When sums of Kronecker products are utilized to approximate $W$, the model strengthens, and the independence assumptions correspondingly diminish. Indeed, the problem of minimizing the Frobenius norm of a sum of a given number of Kronecker products representation with the size of $\{A_i \, \forall i \in R^{m \times n}\}$ and $\{B_i \, \forall i \in R^{o \times p}\}$ can be transformed into an singular value decomposition (SVD) problem and easily solved, as shown in [9].

## 4. BACKWARD PROPAGATION OF ERRORS

Computing the gradient of the loss function $E$ w.r.t. the parameters of a DBN involves propagating what are often called the *errors*, $\delta_j \equiv \frac{\partial E}{\partial x_j}$, backward through the network:

$$\delta_{j-1} = \sigma'(x_{j-1}) \cdot W^T \delta_j \quad (6)$$

where $\cdot$ denotes elementwise multiplication. Since $(A \otimes B)^T = A^T \otimes B^T$ and $(A + B)^T = A^T + B^T$, when $W$ is represented using a sum of Kronecker products the result immediately follows:

$$\delta_{j-1} = \sigma'(x_{j-1}) \cdot \sum_i \text{vec}(B_i^T \Delta_i^j A_i) \quad (7)$$

where $\Delta_i^j \equiv mat_{A_i^T,B_i^T}(\delta_j)$. This notation was described in detail for $Z_i^{j-1}$ in (5) in the previous section.

The (matrix formatted) gradient of $E$ w.r.t. $A_i$ is given by:

$$\frac{\partial E}{\partial A_i} = {\Delta_i^{(j)}}^T B_i Z_i^{j-1} \quad (8)$$

The (matrix formatted) gradient of $E$ w.r.t. $B_i$ is given by:

$$\frac{\partial E}{\partial B_i} = \Delta_i^j A_i {Z_i^{(j-1)}}^T \quad (9)$$

The first result is derived below:

$$\begin{aligned} \frac{\partial E}{\partial (A_i)_{mn}} &= \sum_o \frac{\partial E}{\partial (X_i^j)_{om}} \frac{\partial (X_i^j)_{om}}{\partial (A_i)_{mn}} \quad (10) \\ &= \sum_o (\Delta_i^j)_{om} \frac{\partial}{\partial (A_i)_{mn}} [\sum_{np} (A_i)_{mn} (B_i)_{op} (Z_i^{j-1})_{pn}] \\ &= \sum_o (\Delta_i^j)_{om} \sum_p (B_i)_{op} (Z_i^{j-1})_{pn} \\ &= ({\Delta_i^{(j)}}^T B_i Z_i^{(j-1)})_{mn} \quad (11) \end{aligned}$$

The latter result follows similarly.

## 5. EXPERIMENTS

Experiments were carried out on a 50-hours English Broadcast News task [10]. The training set contains 50 hours of data from the 1996 and 1997 English Broadcast News Speech Corpora. The test set is 3 hours of EARS dev-04f set. The DBN was used in a *hybrid* mode, i.e. acting as an acoustic model, replacing the conventional GMMs. It was trained to estimate posterior probabilities of 2220 sub-phoneme acoustic classes, which were in turn used as emission probabilities in a Hidden Markov Model. The DBN input features were derived from 13-dimensional Perceptual Linear Prediction (PLP) cepstral features normalized by Vocal Tract Length Normalization (VTLN) and Ceptral Mean Substraction (CMS), which were then spliced to form 9*13=117 input features ($\pm 4$ frames of context). The DBN topology was [117, 1024, 1024, 2220] units, meaning three fully connected layers (L1, L2, L3) with Bernoulli output units in hidden layers L1, L2 and a softmax activation at the output layer L3.

The DBN was initialized randomly with no pretraining and trained using standard error back-propagation and the cross-entropy criterion. The training set was split into *train* and *dev* sets of 45 hrs and 5 hrs, respectively. After each iteration over the *train* set, the per-frame classification accuracy (FAcc) was evaluated on the *dev* set and once the performance ceased to improve, the learning rate was halved. The training was stopped as soon as the learning rate fell under a threshold. The training targets were obtained from a forced-alignment of transcripts with an existing model.

The baseline performance of 40-dimensional LDA features obtained from the above PLP features trained speaker-adaptively is 23.2% WER and the performance of the baseline DBN is 23.0% WER. Note, however, that this baseline is far from state-of-the-art. State-of-the-art DBN systems for ASR utilize more network layers, more

elaborate DBN training recipes such as sequence-level training, and more discriminative feature representations such as feature-level Minimum phone error (fMPE), which are themselves discriminately trained at the sequence level [11]. Our goal here is to begin exploring the new ideas that are presented in this paper, using systems that can be trained and optimized relatively quickly, and with minimal dependency and interaction with other system components.

### 5.1. Factoring an existing DBN into Kronecker products

We initiated our investigation into factoring weight matrices into sums of Kronecker products by investigating how much Kronecker product structure was naturally present in the weight matrix L1, since its input is actually a matrix (i.e. spliced features), and the features are quasi-stationary. We took the weight matrix of the above DBN's L1 layer and decomposed it into a sum of Kronecker products via the SVD method proposed in [9]. Reconstructing the weights matrix using only the first N singular values of SVD can be directly interpreted as decomposing the original matrix into a sum of N Kronecker products. Here we factored the input dimension only: $W[1024, 117] \approx \sum_i A_i[1024, 9] \otimes B_i[1, 13]$; layers L2 and L3 were left intact.

Table 1 gives the performance as a function of the number of factorized terms, measured in Word Error Rate and matrix reconstruction error – Mean Square Error Ratio, $MSER = \frac{\sum_{mn}((W)_{mn}-(\tilde{W})_{mn})^2}{\sum_{mn}(W)^2_{mn}}$. In our case, 9 terms was enough for a perfect reconstruction. Apparently, we can reduce the number of parameters in L1 by 1.8 times to only 5 terms with no impact on overall performance. This confirms that Kronecker structure is naturally present in L1. Using less than 5 terms rapidly degrades the performance and suggests to incorporate the factorization directly into the DBN training process so that all layers are trained jointly.

| terms | L1 savings factor | MSER | %WER |
|---|---|---|---|
| 1 | 8.9939 | 0.7395 | 95.8 |
| 2 | 4.4970 | 0.4954 | 95.7 |
| 3 | 2.9980 | 0.3137 | 65.7 |
| 4 | 2.2485 | 0.1689 | 61.5 |
| 5 | 1.7988 | 0.0850 | 23.0 |
| 6 | 1.4990 | 0.0333 | 22.9 |
| 7 | 1.2848 | 0.0148 | 22.9 |
| 8 | 1.1242 | 0.0059 | 23.0 |
| 9 | 0.9993 | 0.0000 | 23.0 |
| original | 1 | - | 23.0 |

**Table 1**. WER and MSER as a function of the number of terms in the factorization. Post-training factorization of L1 weight matrix, input (9*13) dimension only.

The DBN input of 9 frames times 13 coefficients determines a suitable factorization of the input L1 dimension. However, it is not clear if we can benefit from factoring the output dimension of 1024. Fig. 1 gives some insight. It shows MSER as a function of Savings Factor (SF is defined as a ratio of number of parameters in original and factored matrices). SF in turn depends on two things. On the number of terms used (the more terms, the less savings and the lower error) and also on the output dimension factorization which varies from $1024 = 1 * 1024 = 2 * 512 = ... = 32 * 32 = ... = 1024 * 1$ (biggest savings for the square 32*32 case), the first factor given in the legend of the image. Note that Tab. 1 operates on the dark blue curve of Fig. 1. The same curves for a random matrix (shown in grey) give a rough upper bound on the MSER. Clearly only very modest SFs are achieved for low MSERs. Note, however, that in deriving these results we have (so far) made no attempt to 'pool' output nodes into clusters that would make the assumed Kronecker structures more viable. This could improve the results substantially.

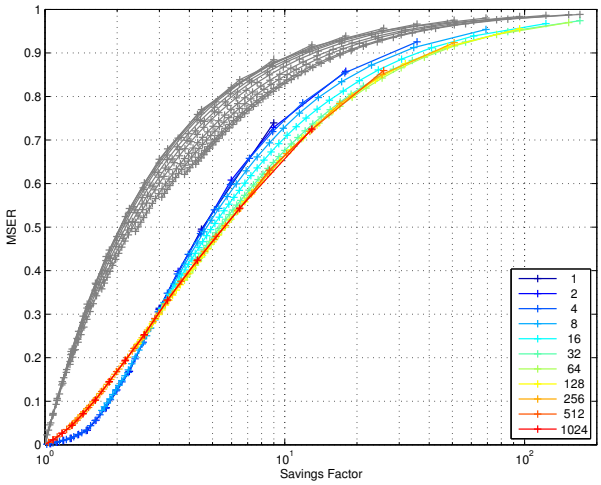


**Fig. 1**. MSER vs. savings factor for a fixed input factorization (9*13), variable output factorization (given in the legend), and varying number of sum terms.

### 5.2. Training Factored Weight Matrices: Simulation via periodic SVD

As a next step, we incorporated the factorization into the training process. We re-used the original training recipe but enforced the factored structure of the weight matrix in the first layer by replacing it by its SVD-derived sum of Kronecker products approximation every few updates. Since SVDs are expensive, we tried to minimize thier use. An empirical search suggested that factoring the matrix every 5 updates preserved the performance of factorization after every update (we do one weight update per one training utterance). We also factored the matrix at the end of the training. Here we factored both input and output dimensions of L1: $W[1024, 117] = \sum_i A_i[32, 9] \otimes B_i[32, 13]$; layers L2 and L3 were again left unfactored.

Table 2 shows the model performance as a function of the number of terms. We can see that with only one single term, one can reduce the number of L1 parameters by approximately 170 times with only 7.6% relative drop in performance.

### 5.3. Training Factored Weight Matrices Via Projected Gradient

The inconsistency and inefficiency of the periodic SVD method for constraining the weight matrix led us to pursue a more direct approach. We next considered projecting the gradient of the full weight matrix onto a sum of Kronecker-product restricted parameterization of it via the chain rule. More specifically, we utilized the following procedure:

- Compute gradient of $W$, $\frac{\partial E}{\partial W}$.

| terms | %FAcc | %WER |
|---|---|---|
| 1 | 34.1 | 24.3 |
| 2 | 33.7 | 25.0 |
| 3 | 34.0 | 24.5 |
| all (baseline) | 35.0 | 23.0 |

**Table 2**. FAcc (Frame Accuracy) and WER as a function of the number of terms in the factorization of the L1 weight matrix. The training recipe enforces the factorization of $W$ into sum of Kronecker products via SVD every 5 training utterances. The results do not show a positive trend w.r.t increasing the number of factors in the representation, despite the fact that the projection was verified to be frequent enough to ensure consistent selection of the top singular values.

| L1 Topology | FA | WER | L1 PR |
|---|---|---|---|
| [117,1024] | 35.0 | 23.0 | 1 |
| 1x[32*32,32*32] | 32.8 | 25.0 | 170 |
| 2x[32*32,32*32] | 32.9 | 24.9 | 85 |
| 3x[32*32,32*32] | 33.2 | 24.6 | 57 |

**Table 3**. Frame Accuracy (FA) and Word Error Rate (WER) as a function of the number of Kronecker terms in the trained representation of the L1 weight matrix (L2 and L3 have topology [1024,1024] and [1024,2048] in all cases). The ratio of the number of parameters of the baseline model relative to each model (parameter ratio, PR) is also shown for layer 1.

- Rearrange $\frac{\partial E}{\partial W}$ to form $\frac{\partial E}{\partial \tilde{W}}$ as if preparing $\tilde{W}$ to do an SVD [9].
- Compute the gradient of and update $U$ and $V$, assuming that $\tilde{W} = UV^T$, where the columns of U each correspond to a $B_i$, the columns of $V$ to a $A_i$. $\frac{\partial E}{\partial U} = \frac{\partial E}{\partial \tilde{W}} V$, $\frac{\partial E}{\partial V} = U \frac{\partial E}{\partial \tilde{W}}^T$.
- Update $\tilde{W}$, rearrange to obtain $W$.

This method has the advantage that it utilizes existing training code as a subroutine so that factorizations of $W$ can be investigated, but does not realize the computation and storage benefits that the parameterization of the matrices affords. This method was utilized to generate the results depicted in Table 3 and 4. These preliminary results suggest that the number of parameters in existing DBNs could potentially be greatly reduced, and that direct product DBNs outperform standard DBNs with the same number of parameters.

### 5.4. Training Factored Weight Matrices Natively

As discussed above, the next step is to utilize sum-of-Kronecker product parameterizations of $W$ natively within the core routines of a DBN trainer. It is our intention to do so and investigate how fast such networks are to train and utilize when practical aspects of the computation, in addition to storage and number of computational operations, such as the regularity and parallelizability of the computation, are important considerations. In traditional DBN trainers, a matrix of input data with stacked frames can be processed in each layer by one matrix-matrix product, and the elementwise application of non-linearities. The computation when utilizing Kronecker-structured weight matrices is greatly reduced (both directly, and because there are less parameters to train) but has more granularity.

| Topology L1/L2/L3 | PR L1/L2/L3 | PR DBN | FA | WER |
|---|---|---|---|---|
| [117,1024] (baseline)<br>[1024,1024]<br>[1024,2220] | 1<br>1<br>1 | 1 | 35.0 | 23.0 |
| [117,740]<br>[740,740]<br>[740,2220] | 1.4<br>1.9<br>1.4 | 1.5 | 32.7 | 26.4 |
| [117,280]<br>[280,280]<br>[280, 2220] | 3.7<br>13.4<br>3.7 | 4.7 | 31.2 | 27.7 |
| [117,135]<br>[135,135]<br>[135 2220] | 7.6<br>57.3<br>7.6 | 10.3 | 28.8 | 31.2 |
| 5x[9*13,32*32]<br>10x[32*32,32*32]<br>[1024,2220] | 27<br>49<br>1 | 1.5 | 33.7 | 24.8 |
| 5x[9*13,32*32]<br>10x[32*32,32*32]<br>10x[32*32,1*2220] | 27<br>49<br>3.2 | 4.7 | 31.9 | 26.9 |
| 5x[9*13,32*32]<br>20x[32*32,32*32]<br>4x[32*32,1*2220] | 27<br>25<br>7.9 | 10.3 | 29.8 | 29.1 |

**Table 4**. Frame Accuracy (FA) and Word Error Rate (WER) as a function of DBN topology. The ratio of the number of parameters of the baseline model to each model (parameter ratio, PR) is also shown for each layer, and overall. Direct Product DBNs consistently outperform unstructured DBNs with the same number of parameters (same DBN PR).

## 6. CONCLUSION

In this paper we introduced the idea of representing the weight matrices used in DBNs as sums of direct matrix products. While the preliminary results are promising, there is clearly much that has yet to be explored. More experimentation will be required to fully assess the potential of this new framework. In addition, the experiments in this paper have focused on utilizing sums of Kronecker products that have factors $A_i, B_i$ that have the same dimension $\forall i$. This constraint has the advantage that estimating the weight matrix can be transformed into an SVD problem, but is not necessary. In any case, the prospect of being able to train conventional sized DBNs more quickly, and train much larger DBNs with direct-product-constrained weight matrices, is compelling.

## 7. REFERENCES


[1] A. Mohamed and G. E. Hinton, "Phone recognition using restricted boltzmann machines.," in *ICASSP*, 2010.

[2] G. Hinton, L. Deng, D. Yu, G. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. Sainath, et al., "Deep neural networks for acoustic modeling in speech recognition," *IEEE Signal Processing Magazine*, 2012.

[3] G. E. Hinton, S. Osindero, and Y.-W. Teh, "A fast learning algorithm for deep belief nets," *Neural Comput.*, vol. 18, pp. 1527–1554, July 2006.

[4] T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Improving training time of deep belief networks through hybrid pre-training and larger batch sizes," in *NIPS Workshop on Log-linear Models*, 2012.

[5] Y. MarcAurelio Ranzato, L. Boureau, and Y. LeCun, "Sparse feature learning for deep belief networks," *Advances in neural information processing systems*, vol. 20, pp. 1185–1192, 2007.

[6] Ruslan Salakhutdinov, Andriy Mnih, and Geoffrey Hinton, "Restricted boltzmann machines for collaborative filtering," in *ACM international conference proceeding series*, 2007, vol. 227, pp. 791–798.

[7] Jullian Chin, Peder Olsen, Jorge Nocedal, and Steven J. Rennie, "Second order methods for optimizing convex functions and sparse inverse acoustic model compression," *IEEE Transactions on Audio, Speech, and Language Processing (submitted)*, vol. 20, no. 6, 2013.

[8] J. Leskovec, D. Chakrabarti, J. Kleinberg, C. Faloutsos, and Z. Ghahramani, "Kronecker graphs: An approach to modeling networks," *The Journal of Machine Learning Research*, vol. 11, pp. 985–1042, 2010.

[9] C. Van Loan and N. Pitsianis, "Approximation with kronecker products," *Technical Report, Cornell University*, 1992.

[10] Brian Kingsbury, "Lattice-based optimization of sequence classification criteria for neural-network acoustic modeling," in *ICASSP*, 2009, pp. 3761–3764.

[11] Tara N Sainath, Brian Kingsbury, Bhuvana Ramabhadran, Petr Fousek, Petr Novak, and A-R Mohamed, "Making deep belief networks effective for large vocabulary continuous speech recognition," in *Automatic Speech Recognition and Understanding (ASRU), 2011 IEEE Workshop on*. IEEE, 2011, pp. 30–35.