June 04, 2018

Weight compression for deep networks using Kronecker products

Yair Movshovitz-Attias

Elad Eban

Weight compression for deep networks using Kronecker products

ABSTRACT

Deep networks have shown success in many challenging applications, e.g., image understanding, natural language processing, etc. The success of deep networks is traced to the large numbers of neurons deployed, each with weighted interconnections to other neurons. The large numbers of weights result in classification accuracy, but also use significant memory.

This disclosure describes techniques to reduce the number of weights used in deep networks by representing the matrices of deep network weights as the Kronecker product of two or more smaller matrices. The reduction in weights is made possible by the observation that deep networks do not always use a majority of their weights. Training procedures are described for the resulting compressed network. The techniques of this disclosure enable deep networks to be deployed in small footprint applications, e.g., mobile or wearable devices. Applications with no immediate memory constraint, e.g., servers, also benefit by the greater speed of deployment enabled by the techniques herein.

KEYWORDS

- deep neural networks
- DNN
- Kronecker product
- neural network compression
- weight compression
- lightweight neural nets
- machine learning

BACKGROUND

Deep networks are a type of machine learning model that uses multiple layers of interconnected processing units (neurons) for feature extraction and generalization. Deep networks have shown success in many challenging applications of machine intelligence, e.g., image understanding, natural language processing, etc. The success of deep networks is traced to the large numbers of neurons deployed, each with several weighted interconnections to other neurons. The success of deep models has encouraged researchers to design models with increasing size. This results in improvements in accuracy but has the cost of increased memory usage. Memory is a limiting factor in deploying deep networks in certain contexts, e.g., on mobile and wearable devices. Memory constraints often force deep-network developers to compromise in various ways e.g.,

- deploy a smaller network that has a lower performance;
- store the network on a server and communicate data and network predictions back and forth; etc.

These approaches are sub-optimal in terms of latency and performance.

Even in settings where single-model memory is not an acute constraint (e.g., data centers) model size has an important role. For example, new models are pushed to production on cloud servers on a regular basis. The size of a model has a direct effect on the time it takes for a new model configuration to be deployed. Therefore, model size is a factor that limits rapid deployment. Additionally, many systems, e.g., driverless cars, employ multiple deep models and in the aggregate memory usage of the multiple models taken together becomes prohibitive.

DESCRIPTION

While deep neural networks (DNNs) have many layers (depth) of parameters, it has been shown that DNNs do not always need all the individual weights in each layer. The techniques of this disclosure leverage this observation by storing in memory a few small weight matrices, and assigning values to network parameters by judiciously extrapolating weights within these matrices using the Kronecker-product operator.

The Kronecker product $\mathbf{A}\otimes\mathbf{B}$ of two matrices **A** and **B** is a well-known mathematical operation.[1] Briefly, the Kronecker product of two matrices **A** and **B** is a matrix whose elements comprise the scalar product of every possible pair of elements of **A** and **B**. Specifically, If $\mathbf{A}=(a_{rs})$ is an $m\times n$ matrix and $\mathbf{B}=(b_{vw})$ is a $p\times q$ matrix, then the Kronecker product $\mathbf{A}\otimes\mathbf{B}$ is the $mp\times nq$ matrix given by:

$$(\mathbf{A}\otimes\mathbf{B})_{p(r-1)+v,\ q(s-1)+w} = a_{rs}b_{vw} \qquad (1)$$

The Kronecker product, also known as tensor product, is a generalization of the vector outer product operator to matrices.

For an original deep network model, denoted M_o has a set of weights $\mathbf{W}=(w_1, w_2, \ldots, w_N)$, where N is the total number of weights in the model, the techniques of this disclosure generate a compressed model M_c comprising less than N weights, as follows.

Let the number of weights N in the original DNN M_o be factorized as $N=m\times p\times n\times q$. An order-2 Kronecker-compressed network M_c is a network of the same shape and number of parameters, but with only $m\times n + p\times q$ unique trainable weights arranged in two matrices **A** and **B**,

[1] See https://en.wikipedia.org/wiki/Kronecker_product

according to equation (1). More generally an order-k compression, defined by the k-wise tensor product of matrices $A_1, A_2, \ldots, A_k$ respectively of dimensions $n_1 \times m_1, \ldots, n_k \times m_k$, is given by

$$\mathbf{W} = \boldsymbol{A}_1 \otimes \boldsymbol{A}_2 \otimes \cdots \otimes \boldsymbol{A}_k$$

where the number of parameters in $\mathbf{W}$ is $N = \Pi_{i=1}^{k} n_i m_i$.

The compressed model M_c is derived using the following steps:

- compression strength selection, e.g., Kronecker shape selection; and
- re-adjustment of deep network model, e.g., training of compressed model.

Each of the above is explained in greater detail below.

Compression strength selection

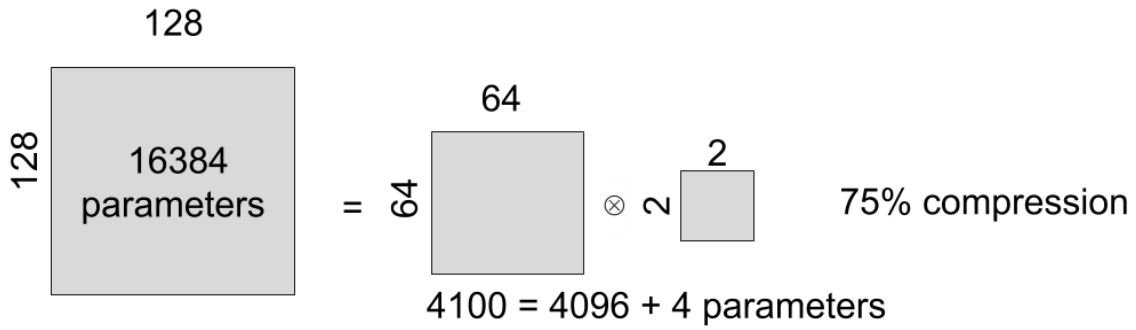

Fig. 1: Parameter reduction using Kronecker product

Fig. 1 illustrates an example of parameter reduction using the Kronecker product. In Fig. 1, an original DNN comprising 16,384 weights is decomposed into a Kronecker product of two matrices comprising 4,096 and 4 weights respectively, representing a parametric reduction of 75%, or a compression strength of 100−75=25%.

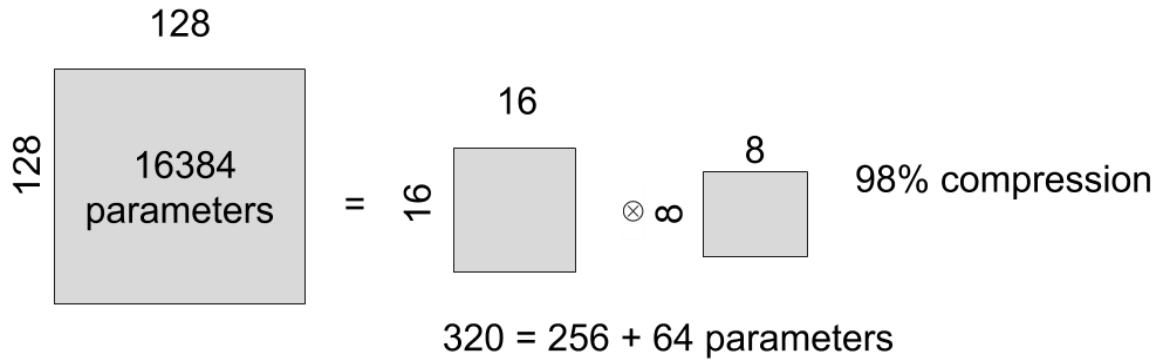

Fig. 2: Parameter reduction using Kronecker product

Fig. 2 illustrates another example of parameter reduction using the Kronecker product. In Fig. 2, an original DNN comprising 16,384 weights is decomposed into a Kronecker product of two matrices comprising 256 and 64 weights respectively, for parametric reduction of 98%, or a compression strength of 100−98=2%.

In this manner, the compression strength is controlled by judiciously selecting the factors m, n, p, and q of the number of weights $N=m\times p\times n\times q$ of the original DNN M_o.

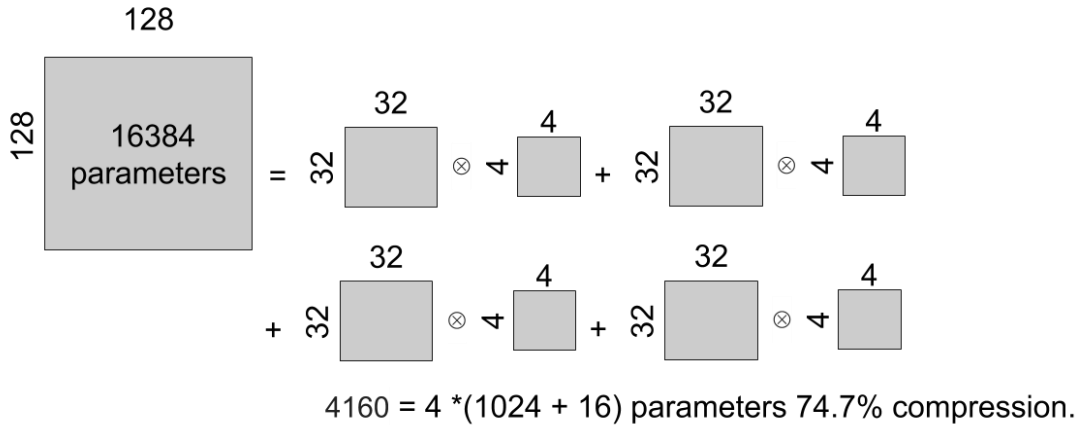

Fig. 3: Parameter reduction using a sum of matrices, each being a Kronecker product

It is possible to generalize the decomposition of the original DNN M_o such that a compressed network M_c comprises the sum of Kronecker products. This is shown in Fig. 3, where an original DNN comprising 16,384 weights is decomposed into the sum of four matrices,

each such constituent matrix being the Kronecker product of two matrices comprising respectively 32×32=1024 and 4×4=16 weights. The total parametric reduction is 74.7%.

Given an original DNN comprising *N* weights and a target compression strength α, there are thus many ways (candidate decompositions) to achieve the target compression strength. A python-language code-snippet that achieves the target compression strength is shown in Fig. 4. For ease of computation, the code-snippet of Fig. 4 works for the case when the number of weights *N* in the original DNN is a power of 4, but the code can easily be generalized to arbitrary values of *N*.

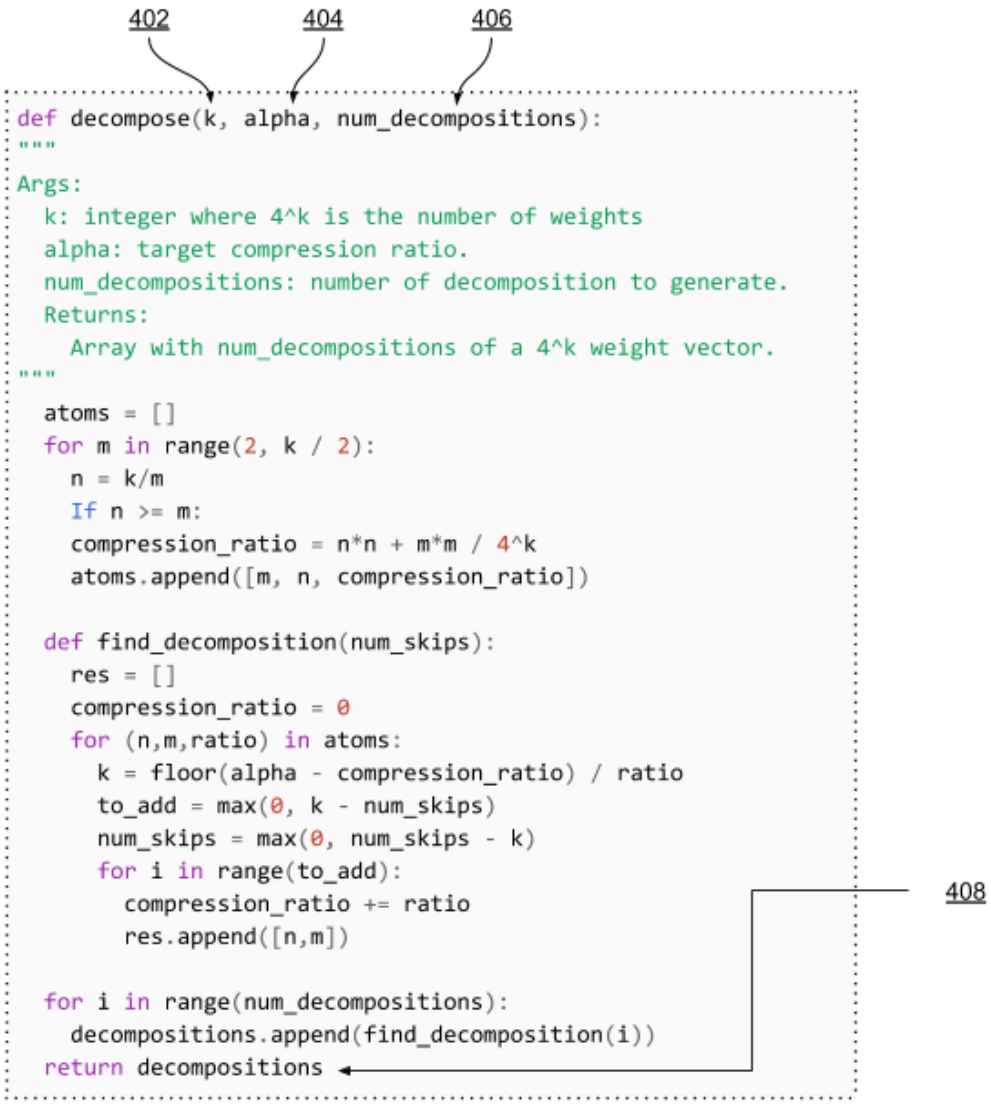

Fig. 4: Code-snippet that computes Kronecker product decompositions of a DNN comprising $N=2^{2k}$ weights

The program of Fig. 4 accepts as input parameters the following:

- the size *N* of original DNN in the form of a base-4 exponent `k` (402), such that $N=2^{2k}$;
- a requested compression strength (404) `alpha`; and
- a requested number of candidate decompositions, `num_decompositions` (406).

The program returns as output a number of candidate decompositions (408), labeled as the return value `decompositions`.

Example 1: The input parameters are `k=7`; `alpha=25%`, and `num_decompositions=1`. The output `decompositions` represents Kronecker product of two matrices of sizes 64×64 and 2×2, as illustrated in Fig. 1.

Example 2: The input parameters are `k=7`; `alpha=2%`, and `num_decompositions=1`. The output `decompositions` represents Kronecker product of two matrices of sizes 16×16 and 8×8, as illustrated in Fig. 2.

Example 3: The input parameters are `k=7`; `alpha=25%`, and `num_decompositions=2`. Two candidate decompositions are returned,

- A first Kronecker product of two matrices of sizes 64×64 and 2×2, as illustrated in Fig. 1; and
- A second Kronecker product being the sum of four matrices, each matrix being the Kronecker product of two matrices of size 32×32 and 4×4, as illustrated in Fig. 3.

Although the compressions achieved, e.g., as shown in Fig. 1-3, are significant, in many DNN applications, there is only a slight loss in classification accuracy with use of the compressed

model. Also, there is no restriction on using a single Kronecker decomposition to represent the entire original DNN. For example, each layer of the DNN can have a different Kronecker decomposition targeted at different compression strengths.

If the size *N* of the original DNN is not a power of 4, then the original set of weights may be broken into subsets whose cardinality is a power of 4. For example, if *N*=3504, the code-snippet of Fig. 4 can be run with `k=5` ($N = 4^5 = 1024$), such that four subsets of weights are formed, each with 1024 weights. As 3504 weights are represented by four groups each with 45=1024 weights, 4×1024−3504=592 weights are surplus.

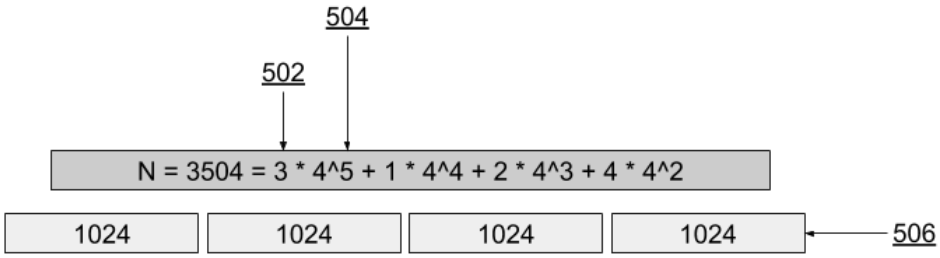

Fig. 5: Dividing weights into sets with power-of-4 cardinality

This is illustrated in Fig. 5, where the coefficient (502) of the highest power in the base-4 expansion of *N*=3504 is 3, while the highest power (504) itself is 5. Hence the set of weights is divided into 3+1=4 groups of $4^5 = 1024$ weights each (506).

Mathematically, *N* is represented in base-4 arithmetic as:

$$N = a_0 + a_1 4^1 + a_2 4^2 + a_3 4^3 + \ldots + a_{k-1} 4^{k-1} + a_k 4^k \text{ , where } a_i \in \{0, 1, 2, 3\}$$

The number of groups the *N* weights is divided into is a_k. This is achieved by rounding *N* to the first number N' such that $a_{k-1}' = a_{k-2}' = \ldots a_0' = 0$ in the base-4 expansion of N'. It holds that $N \leq N' \leq 2N$. Fig. 6 illustrates an example code snippet to round up *N* to the next-highest power of 4.

```
def round_up(N):
  k = floor(log2(N) / 2.0)
  a_k = floor(N / 4 ** k)
  return (a_k +1) *  4 ** k
```

Fig. 6: Rounding up to next-highest power of 4

Re-adjustment of deep network model

Having obtained a compressed deep neural network with a lower number of weights as explained above, there remains the task of training the compressed network. Training can be performed in at least two ways, e.g., training from scratch, training via network distillation, etc. In training from scratch, the training procedure, e.g., backpropagation, gradient descent, etc., is applied directly to the reduced set of weights to optimize the same loss function that is optimized by the original network. In network distillation, the original network acts as a teacher that trains the compressed network, which in turn acts as a student. The teacher network provides predictions (training examples) that are used as targets by the student network for training purposes.

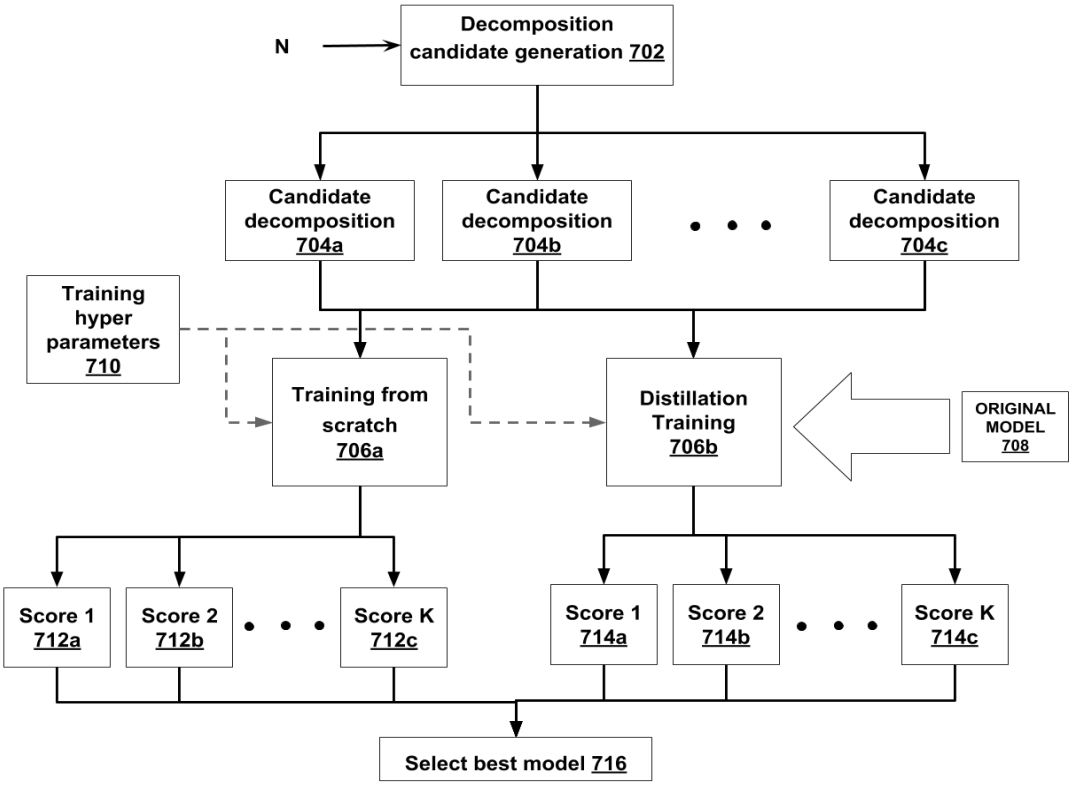

Fig. 7: Selecting an optimal compressed network model from several candidate decompositions

Since several candidate decompositions are available, each with at least two training procedures, each training procedure itself having tuning parameters, a final, optimal, choice is made of the compressed network. This is illustrated in Fig. 7. Given the number of weights N in an uncompressed network M_o , a candidate decomposition procedure (702) is run to obtain a number of candidate decompositions (704a-c).

Each candidate decomposition is run through training procedures, e.g., training from scratch (706a), distillation training (706b), etc. In case of distillation training, the original model (708) serves as a teacher network to the candidate compressed networks. In either training procedure, training hyper-parameters (710) are provided as necessary. At the end of training, each candidate decomposition and training procedure is scored against a standard test-set. Scores are generated for compressed networks trained from scratch (712a-c) and trained via distillation

(714a-c). An optimal score is selected (716) that results in an optimal selection of compressed network, training procedure, and training hyper-parameters.

CONCLUSION

This disclosure describes techniques to reduce the number of weights used in deep networks by representing the matrices of deep network weights as the Kronecker product of two or more smaller matrices. The reduction in weights is made possible by the observation that deep networks do not always use a majority of their weights. Training procedures are described for the resulting compressed network. The techniques of this disclosure enable deep networks to be deployed in small footprint applications, e.g., mobile or wearable devices. Applications with no immediate memory constraint, e.g., servers, also benefit by the greater speed of deployment enabled by the techniques herein.