Neural Network Compression via Additive Combination of Reshaped, Low-rank Matrices

Yerlan Idelbayev and Miguel Á. Carreira-Perpiñán
Dept. CSE, University of California, Merced
http://eecs.ucmerced.edu

The code is available at:
https://github.com/UCMerced-ML/LC-model-compression
A $K$-layer neural net is a computational graph that computes output $f$ for an input $x$:

$$f(x; w) = \sigma(W^K \cdots \sigma(W^2 \sigma(W^1 x)))$$

The weights $W = \{W^1, \ldots, W^K\}$ are trained on a dataset of input-output pairs $(x, y)$ to make the network output $f(x; w)$ closer to the true output $y$:

**regression:** $\min_w L(w) = \sum_{x,y} ||y - f(x; w)||^2$

**classification:** $\min_w L(w) = \sum_{x,y} \text{CrossEntropy}(y, f(x; w))$
We replace a matrix $W^i$ with some rank-$r$ matrix

- Such matrix can be written as the product $UV^T$, i.e., $W = UV^T$
  - For small values of $r$ this reduces FLOPs and storage
  - Can achieve speed-up on any hardware (uses standard matrix-vector products)
- If ranks are known, training is not hard: simply decompose and then use SGD
- If ranks are not known, selection algorithms exist including many heuristics ones
Proposed scheme: Using additive combinations

Instead of single low-rank constraint we propose to use an additive combination

\[ W = \Theta_1 + \Theta_2 \]  \hspace{1cm} (1)

where \( \Theta_1 \) and \( \Theta_2 \) are some low-rank matrices.

Unfortunately such a naive scheme does not enrich the compression. Sum of low-rank matrices can be always expressed a single low-rank matrix.
Proposed scheme: Using additive combinations and reshapes

To circumvent the limitation of the naive approach we introduce a reshaping function $\mathcal{R}$ which is similar to MATLAB’s reshape function.

Formally, assume we have a weight matrix $\Theta$ of shape $m \times n$ (same as $W$). Let us define a reshape $\mathcal{R}(\Theta)$ to be a re-ordering of the $mn$ elements of $\Theta$ into some $p \times q$ sized matrix (with $pq = mn$), such that $\mathcal{R}(\Theta)$ contains the same set of elements as $\Theta$, but in different order.

For example, a reshape can be as follows:

$$
\mathcal{R} \left( \begin{bmatrix}
1 & 2 & 3 & 4 \\
5 & 6 & 7 & 8 \\
9 & 10 & 11 & 12
\end{bmatrix} \right) \rightarrow \begin{bmatrix}
1 & 3 & 5 & 2 & 4 & 6 \\
7 & 9 & 11 & 8 & 10 & 12
\end{bmatrix}
$$

And we will impose low-rank on reshaped matrices!
Using reshaping functions, our final compression scheme is as follows:

\[ W = \Theta_1 + \Theta_2, \quad \text{rank} (R_1(\Theta_1)) = r_1, \quad \text{rank} (R_2(\Theta_2)) = r_2. \] (2)

- Includes regular low-rank as a special case: set \( R_1 \) to be identity reshape and set \( r_2 = 0 \).
- Forward pass \((Wx)\) can be efficiently parallelized: compute \( \Theta_1x \) and \( \Theta_2x \) independently in parallel and then sum.
How to train networks with the proposed additive low-rank scheme?

To efficiently compress a network, we need to use the best configuration of ranks over the layers. But there are combinatorial number of possible configurations!

To handle the combinatorial nature of the problem:

- we formulate a suitable optimization problem
- and give an efficient optimization algorithm based on Learning-Compression framework [1, 2, 3, 4, 5]
Problem formulation

Given a $K$-layer net with weights $\mathbf{W} = \{\mathbf{W}^1, \ldots, \mathbf{W}^K\}$ trained on the loss $L$ (e.g., cross-entropy), we formulate the following rank selection problem:

$$\begin{align*}
\min_{\mathbf{W}, \Theta, r} & \quad L(\mathbf{W}) + \lambda C(\Theta, r) \\
\text{s.t.} & \quad \mathbf{W}^k = \Theta_1^k + \Theta_2^k, \quad \text{for} \quad k = 1 \ldots K, \\
& \quad \text{rank} \left( R_1(\Theta_1^k) \right) = r_1^k, \quad \text{rank} \left( R_2(\Theta_2^k) \right) = r_2^k.
\end{align*}$$

(3)

Here, the term $\lambda C(\Theta, r)$ controls the amount of compression and can target a specific cost of interest like FLOPs or storage.
Problem formulation: Compression cost function

The cost $C$ is a function of the layers’ ranks:

$$C(\Theta, r) = C(r) = \sum_{k=1}^{K} (\alpha^k_1 r^k_1 + \alpha^k_2 r^k_2)$$  \hspace{1cm} (4)$$

for some constant values of $\alpha^k_1$ and $\alpha^k_2$.

This definition is quite general. Consider the inference FLOPs of a fully connected layer with weights $W$ of $m \times n$. The total FLOPs is:

$$\text{FLOPs}(Wx) = \text{FLOPs}(\Theta_1 x) + \text{FLOPs}(\Theta_2 x)$$

$$= \left( m + n \right) \times r_1 + \left( m + n \right) \times r_2$$

const. const.
Problem formulation

Given a $K$-layer net with weights $\mathbf{W} = \{\mathbf{W}^1, \ldots, \mathbf{W}^K\}$ trained on the loss $L$ (e.g., cross-entropy), we formulate the following rank selection problem:

$$\min_{\mathbf{W}, \Theta, r} \quad L(\mathbf{W}) + \lambda C(\Theta, r)$$

s.t. $\mathbf{W}^k = \Theta^k_1 + \Theta^k_2$, for $k = 1 \ldots K$,

$$\text{rank } (\mathcal{R}_1(\Theta^k_1)) = r^k_1, \quad \text{rank } (\mathcal{R}_2(\Theta^k_2)) = r^k_2.$$

This is a mixed-integer optimization problem. To solve it efficiently, we need an algorithm that can natively handle both integer and real-valued weights. To perform the optimization we use a learning-compression algorithm.
Let us bring the equality constraints into the objective using a penalty method and obtain an equivalent formulation (optimized as $\mu \to \infty$):

$$
\min_{W, \Theta, r} \quad L(W) + \lambda C(\Theta, r) + \frac{\mu}{2} \sum_{k=1}^{K} \left\| W^k - \Theta_1^k - \Theta_2^k \right\|^2
$$

s.t. $\quad \text{rank} \left( R_1(\Theta_1^k) \right) = r_1^k, \quad \text{rank} \left( R_2(\Theta_2^k) \right) = r_2^k, \quad k = 1 \ldots K.$

Under standard assumptions, the stationary point of (5) at $\mu \to \infty$ is the a feasible solution of the original problem (3).
Optimization algorithm: Deriving the L and C steps (cont.)

Let us now apply alternating optimization over variables $W$ and $\{\Theta, r\}$:

- The step over $W$, which we call a learning (L) step, has the form of:

$$
\min_{W} \quad L(W) + \frac{\mu}{2} \sum_{k=1}^{K} \left\| W^k - \Theta^k_1 - \Theta^k_2 \right\|^2.
$$

Regular NN training independent of compression, typically solved by SGD

- The step over $\{\Theta, r\}$, which we call a compression (C) step, has the form of:

$$
\min_{\Theta, r} \quad \lambda C(\Theta, r) + \frac{\mu}{2} \sum_{k=1}^{K} \left\| W^k - \Theta^k_1 - \Theta^k_2 \right\|^2
$$

s.t. $\quad \text{rank} \left( R_1(\Theta^k_1) \right) = r^k_1$, $\quad \text{rank} \left( R_2(\Theta^k_2) \right) = r^k_2$, $\quad k = 1 \ldots K$.

Actual compression step independent of NN weights and dataset.
Optimization algorithm: Solution of the C step

Due to the layerwise separability of the cost function $C$, the C-step problem separates over the layers into $K$ smaller problems:

$$
\min_{\Theta_1^k, \Theta_2^k, r_1^k, r_2^k} \lambda C_k(r_1^k, r_2^k) + \frac{\mu}{2} \left\| W^k - \Theta_1^k - \Theta_2^k \right\|^2
$$

s.t. $\text{rank} \left( R_1(\Theta_1^k) \right) = r_1^k$, $\text{rank} \left( R_2(\Theta_2^k) \right) = r_2^k$.

(6)

We are not aware of any closed-form solution for this problem. However, we can again use alternating optimization, over the groups of $\{\Theta_1^k, r_1^k\}$ and $\{\Theta_2^k, r_2^k\}$. In such case each subproblem does have a closed-form solution, involving SVD and selection over the ranks [4].
Optimization algorithm: Pseudocode

**input** $K$-layer neural net with weights $\mathbf{W} = (\mathbf{W}^1, \ldots, \mathbf{W}^K)$, hyperparameter $\lambda$, cost function $C$, distinct reshaping functions $\mathcal{R}_1$ and $\mathcal{R}_2$

$\mathbf{W} = (\mathbf{W}^1, \ldots, \mathbf{W}^K) \leftarrow \arg\min_{\mathbf{W}} L(\mathbf{W})$  \hfill \text{reference net}

$r = \{r^k_1, r^k_2\}^{K}_{k=1} \leftarrow 0$  \hfill \text{ranks}

$\Theta = \{\Theta^k_1, \Theta^k_2\}^{K}_{k=1} \leftarrow 0$  \hfill \text{additive terms}

\textbf{for} $\mu = \mu_1 < \mu_2 < \cdots < \mu_T$

\textbf{L step}

$\mathbf{W} \leftarrow \arg\min_{\mathbf{W}} L(\mathbf{W}) + \frac{\mu}{2} \sum_{k=1}^{K} \|\mathbf{W}^k - \Theta^k_1 - \Theta^k_2\|^2$

\textbf{C step}

\textbf{repeat} for $S$ times

$\Theta^k_1, r^k_1 \leftarrow \arg\min_{\Theta^k_1, r^k_1} \lambda C_k(r^k_1, r^k_2) + \frac{\mu}{2} \|\mathbf{W}^k - \Theta^k_2 - \Theta^k_1\|^2 \quad \text{s.t.} \quad \text{rank} \left( \mathcal{R}_1(\Theta^k_1) \right) = r^k_1$

$\Theta^k_2, r^k_2 \leftarrow \arg\min_{\Theta^k_2, r^k_2} \lambda C_k(r^k_1, r^k_2) + \frac{\mu}{2} \|\mathbf{W}^k - \Theta^k_1 - \Theta^k_2\|^2 \quad \text{s.t.} \quad \text{rank} \left( \mathcal{R}_2(\Theta^k_2) \right) = r^k_2$

\textbf{return} $\mathbf{W}, \Theta, r$
Experiments

VGG16 on CIFAR10

Test error, %

MFLOPs

AlexNet on ImageNet

Test error (top-1), %

MFLOPs
Code is available online

Our code is written in Python using PyTorch, and we make it available as part of our extensible model compression framework (under BSD 3-clause license):

https://github.com/UCMerced-ML/LC-model-compression

Using the provided code, you will be able to:

▷ replicate all reported experiments
▷ compress your own models with our proposed scheme and many others.

But this library does much more than that. It is intended to support compression of an arbitrary model (not just neural nets) and an arbitrary compression technique. At the moment it offers the following:

▷ quantization (in various forms)
▷ pruning (in various forms)
▷ low-rank with automatic rank selection
▷ combinations of all the above
References


