Exploiting Linear Structure Within Convolutional Networks for Efficient Evaluation

Abstract

1. Introduction

Large neural networks have recently demonstrated impressive performance on a range of speech and vision tasks. However the size of these models can make their deployment at test time problematic. For example, mobile computing platforms are limited in their CPU speed, memory and battery life. At the other end of the spectrum, Internet-scale deployment of these models requires thousands of servers to process the 100's of millions of images per day. The electrical and cooling costs of these servers required is significant.

Training large neural networks (NN) can take weeks, or even months. This hinders research and consequently there have been extensive efforts devoted to speeding up training procedure. However, there are relatively few efforts are improving the *test-time* performance of the models.

In this paper we focus on speeding up the evaluation of *trained* networks, without compromising performance. We consider convolutional neural networks used for computer vision tasks, since they are large and widely used in commercial applications. Within these models, most of the time ($\sim 90\%$) is spent in the convolution operations in the lower layers of the model. The remaining operations: pooling, contrast normalization and the upper fully-connected layers collectively take up the remaning 10%.

We present two novel methods for speeding up the convolution operations. One involves projecting the input image into a set of 1-D color sub-spaces. This allows the filters in the first of layer of the model to be monochromatic (i.e. reducing the color channels from three to one), thereby saving a factor of 3 in computation. The second approach, applied to subsequent convolution layers, involves clustering the filters into a set of low-dimensional linear sub-spaces, each of which is represented by a set of tensor outer-products. Collectively, our techniques speed up execution by factor of $2-4$ while keeping prediction accuracy within 1% of the original model. These gains allow the use of larger, higher performance models than would otherwise be practical.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

2. Related Work

Vanhoucke et. al. (Vanhoucke et al., 2011) explored the properties of CPUs to speed up execution. They present many solutions specific to Intel and AMD CPUs, however some of their techniques are general enough to be used for any type of processor. They describe how to align memory, and use SIMD operations (vectorized operations on CPU) to boost the efficiency of matrix multiplication. Additionally, they propose the linear quantization of the network weights and input. This involves representing weights as 8-bit integers (range $[-128, 127]$), rather than 32-bit floats. This approximation is similar in spirit to our approach, but differs in that it is applied to each weight element independently. By contrast, our approximation approach models the structure within each filter. Potentially, the two approaches could be used in conjunction.

The most expensive operations in convolutional networks are the convolutions in the first few layers. The complexity of this operation is linear in the area of the receptive field of the filters, which is relatively large for these layers. However, Mathieu et. al. (Mathieu et al., 2013) has shown that convolution can be efficiently computed in Fourier domain, where it becomes element-wise multiplication (and there is no cost associated with size of receptive field). They report a forward-pass speed up of around $10x$ (depending on the kernel size, number of features etc.). Importantly, this method can be used jointly with most of techniques presented in this paper.

The use of low-rank approximations in our approach is inspired by work of Denil et. al (Denil et al., 2013) who demonstrate the redundancies in neural network parameters. They show that the weights within a layer can be accurately predicted from a small (e.g. $\sim 5\%$) subset of them. This indicates that neural networks are heavily over-parametrized. All the methods presented here focus on exploiting the linear structure of this over-parametrization.

3. Low Rank Approximations

In this section, we give theoretical background on low rank approximations. First, we discuss simplest setting, which is for matrices (two dimensional tensors). Further, we move to approximation of 4-dimensional tensors with 2 convolutional (spacial) dimensions.

3.1. Matrix Low Rank Approximation

Let's consider input $X \in \mathbb{R}^{n \times m}$, and matrix of weights $W \in \mathbb{R}^{m \times k}$. Matrix multiplication , which is the main operation for fully connected layers costs $O(nmk)$. However, potentially W might have a low-rank (many eigenvalues close to zero), and operation XW can be computed much faster.

Every matrix can be expressed using singular value decomposition:

$$W = USV^T, \text{ where } U \in \mathbb{R}^{m \times m}, S \in \mathbb{R}^{m \times k}, V \in \mathbb{R}^{k \times k}$$

S is has eigenvalues on the diagonal, and apart from it has zeros. W can be approximated by using t most significant eigenvalues from S. We can write approximation as

$$\hat{W} = \hat{U}\hat{S}\hat{V}^T, \text{ where } \hat{U} \in \mathbb{R}^{m \times t}, \hat{S} \in \mathbb{R}^{t \times t}, \hat{V} \in \mathbb{R}^{t \times k}$$

Cost of multiplication X with $\hat{W}$ is $O(nmt+nt^2+ntk)$, which can be significantly smaller than $O(nmk)$ (for sufficiently small t). E.g. for $n = 1000, m = 1000, k = 1000$ and $t = 100$, we have that exact computation takes 10^9 operations, while the approximated one takes $2 * 10^8 + 10^7$ operations. This would give $\times 5$ theoretical speed up. However, often major cost is in memory management, and theoretical speed up might be infeasible or very difficult to achieve.

3.2. Tensor Low Rank Approximations

In typical object recognition architectures, the convolutional tensors resulting from the training exhibit strong redundancy and regularity across all its dimensions. A particularly simple way to exploit such regularity is to linearly compress the tensors, which amounts to finding low-rank approximations.

Convolution kernels can be described as a 4-dimensional tensors. Let $W \in \mathbb{R}^{C \times X \times Y \times F}$ be convolutional kernel. C is input number of feature maps (or colors), X, Y are special dimensions over which we compute convolution, and F is the target number of feature maps. Let $I \in \mathbb{R}^{C \times N \times M}$ denote an input signal. A generic convolutional layer is defined as

$$I * W(f, x, y) = \sum_{c=1}^{C} \sum_{x'=-X/2}^{X/2} \sum_{y'=-Y/2}^{Y/2} I(c, x-x', y-y') W(c, x', y', f)$$

Low-rank approximation of tensors is very similar to low-rank approximation of matrices. However, first we have to choose 2 dimensions with respect to which we are approximating, and treat rest of tensor as it would be a matrix. Moreover, we show how to perform approximation in the way to exploit computation on GPU (e.g. enforce memory alignment).

3.2.1. Monochromatic filters

For generic convolution there are connections between all output feature maps, and all input feature maps. However, one could imagine that this connections in trained network in some basis could become sparse. Sparsity of this connections would allow to save number of multiplications. Concretely, we look for such a linear transformation of input $I \in \mathbb{R}$

We denote by $W_C \in \mathbb{R}^{C \times XYF}$ reshaped version of kernel $W \in \mathbb{R}^{C \times X \times Y \times F}$

3.3. Linear Compression of Convolutional Filter bank

Given a 4-tensor W of dimensions (C, X, Y, F), we search for decompositions that minimize

$$\|W - \sum_{k \leq K} \alpha_k \otimes \beta_k \otimes \gamma_k \otimes \delta_k\|_F \ , \qquad (1)$$

where α_k, β_k, γ_k and δ_k are rank 1 vectors of dimensions C, X, Y and F respectively, and $\|X\|_F$ denotes the Frobenius norm. Generalization of the SVD.

The rank K approximation (3) can be obtained using a greedy algorithm, which computes for a given tensor X its best rank-1 approximation:

$$\min_{\alpha,\beta,\gamma,\delta} \|X - \alpha \otimes \beta \otimes \gamma \otimes \delta\|_F \ . \qquad (2)$$

This problem is solved by iteratively minimizing one of the monoids while keeping the rest fixed. Each of the step consists in solving a least squares problem. (todo expand).

Figures ? and ?? show low-rank approximations of the first two convolutional layers of the Imagenet architecture.

3.4. Analysis of Complexity

A good low-rank approximation allows a computational speed-up.

Let us assume a fixed stride of Δ in each spatial dimension.

Table **??** shows the number of multiplications required to perform the convolution. In order to optimize the complexity, it is not always a good idea to decompose the full tensor W. Indeed, depending on its dimensions, the approximation cost might be superior than the original. We might consider instead low-rank approximations of W which partition the coordinate space in the most efficient manner.

3.5. Optimizing Cost with Subspace Clustering

We can decompose the 4-tensor W in a collection $W_{k,l}$ of 4-tensors, by considering a partition $G_1, ..G_k, .., G_N$ of the first coordinate space C and a partition $H_1, ...H_l, ...H_M$ of the last coordinate space F. If we assume a uniform partition with N groups of C/N coordinates and M groups of F/M coordinates respectively, and that each tensor $W_{k,l}$ is approximated with K rank-1 tensors, the resulting complexity is

$$K \cdot N \cdot M \cdot \left(\frac{C}{N} + X\Delta^{-1} + Y\Delta^{-2} + \frac{F}{M}\Delta^{-2} \right)$$

How to optimize the groupings on each of the variables? We perform a subspace clustering.

Sharing between blocks:

$$\widetilde{W} = \sum_{k \leq K} \alpha_{i(k)} \otimes \beta_{j(k)} \otimes \gamma_{h(k)} \otimes \delta_{m(k)} \ , \qquad (3)$$

If now each of the separable filters is taken out of a collection smaller than K, we can gain in computation. This can be for instance implemented with a K-Means on the tensor decompositions.

Examples: Monochromatic filtering

Spatially Separable

Memory access constraints

4. Numerical Experiments

4.1. Testing time

on GPU: Michael can help.

on CPU

img/eval_per_batch.png

Figure 1. CPU computational time per image for various batch sizes.

4.1.1. Monochromatic

4.1.2. Linear combination of filters

4.1.3. Separable filters

4.2. Denoising

5. Implications

5.1. Denoising Aspect

we can improve training by simple linear denoting.

5.2. Low-Rank training

Low-rank to avoid over-fitting.

6. Discussion

References

Denil, Misha, Shakibi, Babak, Dinh, Laurent, Ranzato, Marc'Aurelio, and de Freitas, Nando. Predicting parameters in deep learning. *arXiv preprint arXiv:1306.0543*, 2013.

Hinton, Geoffrey E, Srivastava, Nitish, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan R. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

Le, Q. V., Ngiam, J., Chen, Z., Chia, D., Koh, P. W.,

`img/eval_per_layer_per_batch_128_batch_size.png`

Figure 2. Per layer breakdown of execution time for mini batch of size 128. Such size of mini batch gives optimal per image CPU time.

and Ng, A. Y. Tiled convolutional neural networks. In *Advances in Neural Information Processing Systems*, 2010.

Le, Quoc V, Ranzato, Marc'Aurelio, Monga, Rajat, Devin, Matthieu, Chen, Kai, Corrado, Greg S, Dean, Jeff, and Ng, Andrew Y. Building high-level features using large scale unsupervised learning. *arXiv preprint arXiv:1112.6209*, 2011.

Lowe, David G. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pp. 1150–1157. Ieee, 1999.

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

Vanhoucke, Vincent, Senior, Andrew, and Mao, Mark Z. Improving the speed of neural networks on cpus. In *Proc. Deep Learning and Unsupervised Feature Learning NIPS Workshop*, 2011.

Zeiler, Matthew D, Taylor, Graham W, and Fergus, Rob. Adaptive deconvolutional networks for mid and high level feature learning. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2018–2025. IEEE, 2011.

`img/eval_per_layer_per_batch_1_batch_size.png`

Figure 3. Per layer breakdown of execution for mini batch of size 1. Use for real time applications.

`img/eval_per_batch_GPU.png`

Figure 4. GPU computational time per image for various batch sizes.

img/eval_per_layer_per_batch_GPU_128_batch_size.png

Figure 5. Per layer breakdown of execution time for mini batch of size 128. Such size of mini batch gives optimal per image GPU time.

img/first.png

img/first_appr.png

Figure 6. (Left) Original filters, (Right) approximated filters. (this pictures are too big, and should contain white separation).