Pattern Recognition Letters 000 (2017) 1–10

Contents lists available at ScienceDirect

Pattern Recognition Letters

journal homepage: www.elsevier.com/locate/patrec

Winner takes all hashing for speeding up the training of neural networks in large class problems

Amir H. Bakhtiary*, Agata Lapedriza, David Masip

Universitat Oberta de Catalunya, Rambla del Poblenou 156, Barcelona 08018, Spain

ARTICLE INFO

Article history:
Available online xxx

MSC:
41A05
41A10
65D05
65D17

keywords:
Winner takes all hashing
Convolutional neural networks
Large scale classification

ABSTRACT

This paper proposes speeding up of convolutional neural networks using Winner Takes All (WTA) hashing. More specifically, WTA hash is used to identify relevant units and only these are computed, effectively ignoring the rest of the units. We show that the proposed method reduces the computational cost of forward and backward propagation for large fully connected layers. This allows us to train classification layers with a large number of units without the associated time penalty. We present different experiments on a dataset with 21K classes to gauge the effectiveness of this proposal. We concretely show that only a small amount of computation is required to train a classification layer. Then we measure and showcase the ability of WTA in identifying the required computation. Furthermore we compare this approach to the baseline and demonstrate a 6 fold speed up during training without compromising on the performance.

Introduction

Convolutional Neural Networks (CNN) have recently achieved state of the art performance in a number of computer vision tasks as demonstrated in Krizhevsky et al. [1] and Razavian et al. [2]. CNNs usually consist of a set of convolutional layers followed by several fully connected layers. The current trend is to train CNNs using stochastic gradient descent and backpropagation. Because these networks have a large number of parameters, large datasets are used during training to avoid overfitting.

One of the challenges when working with large networks is the computational cost during training. Many different efforts have emerged to increase the speed of these networks. This includes using special commands on CPUs [3], using GPUs (as in frameworks such as Torch, Caffe and Theano), and using FPGAs [4,5]. Although these techniques have had great success in reducing training and testing time for CNNs, the amount of computation required has remained constant. This is because they have focused on executing these operations faster and/or in parallel, instead of reducing the number of needed operations.

In this paper we propose a different approach for speeding up large CNN layers with sparse output activation. Instead of trying to speed up the computation itself we propose to reduce the computational complexity by cutting down on the amount of computation needed.

To this end we make use of 2 main observations: (1) the most computationally expensive operation for these fully connected layers is matrix multiplication which occurs both during forward propagation and backpropagation, and (2) the computations related to units with low activation can be avoided. This is because we are only interested in units that are “turned on” and which will lead to changes in the output. The same is true for backpropagation, units with extremely low values will have very low slope and can be ignored or aggregated. This reasoning holds true for any layer with a non-linearity function that applies a cut off on the output values but the extra computation is more pronounced in layers that are trained to have a sparse activation pattern.

Although methods for more efficient matrix multiplication exist [6], these have not been applied to neural networks and, in most cases, current implementations only lead to a modest saving in computational complexity of $O(n^{2.807})$ vs $O(n^3)$. As a consequence, the current implementations of neural networks have a computational cost that grows linearly with the number of units.

To avoid the costs associated with multiplication of matrices we propose to use Winner Takes All (WTA) hashing to identify the units that will have sufficiently high amplitude before performing the expensive matrix computations. Then we compute the exact output of only these units, and use a default value for the rest of the units. This way only a small number of units in the output of the layer needs to be computed.

* Corresponding author.
E-mail addresses: abakhtiary@uoc.edu, amir.h.bakhtiary@gmail.com (A.H. Bakhtiary), alapedriza@uoc.edu (A. Lapedriza), dmasipr@uoc.edu (D. Masip).

http://dx.doi.org/10.1016/j.patrec.2017.01.001
0167-8655/© 2017 Published by Elsevier B.V.

Furthermore, during the backpropagation phase, we only backpropagate through units that were activated and also units that were not activated but should have been. We can do this because the gradient of the none active units is zero or near zero depending on the non-linearity used. Therefore, not computing the dot products where these zero values are present saves computation without changing the final computed derivatives.

The WTA hashing technique has been already used in vision tasks. In particular, it was successfully applied on a HOG-based object detector [7] in order to detect 100,000 object classes. This was done by first training a part based model using a support vector approach [8], then WTA hashing was used to speed up the detection phase of the algorithm.

To contrast our work with [7], we are using the WTA hash to speed up layers of a deep neural network instead of a support vector machine. Furthermore we use WTA to prune the active units during both forward and backward propagation passes. This results in speedups both during training and testing.

To show the viability of our approach, we have applied this technique to the final layer of a CNN that performs classification on a very large number of classes. The first seven layers of our network is the same as the AlexNet from Krizhevsky et al. [1] which is the basis of many of the CNNs used for computer vision. Furthermore, during training we have used pretrained weights for these layers and kept them fixed during training.

The difference between this work and AlexNet is that we have replaced the last fully connected layer, the classification layer, with our WTA hashing layer. Also to showcase the strength of our approach, we have opted to use the network to perform classification on a dataset that contains 21K classes as opposed to the 1K classes.

We train this network to classify the ImageNet-21K dataset and reach a top-1 accuracy of 20.3%. To put this into perspective, [9] achieves an accuracy of 10.5% on the ImageNet-21K dataset. They use manually defined features and a fine tuned linear classifier. In [10] an accuracy of 15.8% is achieved. They first train a deep stack of autoencoders in an unsupervised fashion and then use the resulting network to derive features. A logistic classifier is then used to perform classification on these features.

In [11] it is shown how a CNN can achieve an accuracy of 29.8% on the ImageNet-21K dataset. This work uses an architecture similar to the AlexNet, but it differs from our approach in that it trains the complete network as opposed to using fixed pretrained weights. It needs to be pointed out that although this method achieves a higher accuracy, it requires training for 10 days on 62 machines. This is in contrast to our method where the training is performed on one machine in under 10 h.

In our work we will demonstrate a 6 fold speedup during training of the classification layer. But if the classification layer is large enough, the WTA hashing layer can speed up the training of the whole network.

The green bars in Fig. 1 show the distribution of the computational cost of the AlexNet architecture as we change the number of classes. In the original AlexNet configuration, with 1000 classes, the computational cost of the classification layer is small compared to the rest of the network. As we increase the number of classes from 1000 in Fig. 1a to 10,000 in Fig. 1b or 100,000 in Fig. 1c the cost of the output layer quickly dominates the computational cost of the network. This cost becomes more pronounced when the output size of layer 7 is increased because the computation complexity of a layer is linear in both its input size and its output size.

The red bars in Fig. 1 depict the computational cost of the output layer when we use WTA to prune the extra neurons. Our approach requires much less computation and this improvement allows using the same network architecture on very large numbers of classes without significantly increasing the overall computational cost.

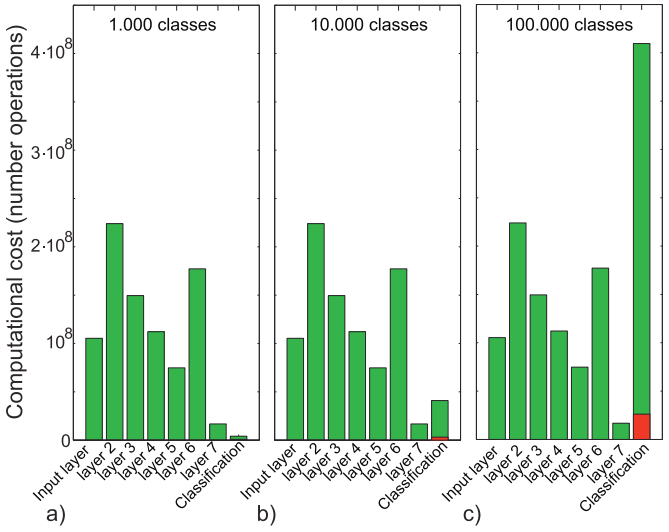

Fig. 1. Distribution of computational cost on AlexNet [1] as we vary the number of output classes from (a) 1,000, (b) 10,000 and (c) 100,000. In red we show the computational cost with our proposed method. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

In the next section we will give a brief overview of WTA hashing. We will follow with a section on how WTA can be used to speed up layers with sparse activation patterns. Section 3 discusses the theoretical computational complexity of the WTA hashing layer. Section 4 presents our experiments. Here we will show how much computational savings are ideally possible, and how WTA is able to realize these saving. Also we present an experiment showing how WTA can be used to achieve a 6 fold speedup when training and testing a classification layer on 21K classes. The final Section 5, contains the conclusions of this paper.

1. Winner takes all (WTA) hashing

The Winner Takes All, WTA, hashing is a generalization of the minhash approach. Minhash was designed for finding similar documents in a large set of documents and has been shown to be effective for data mining in large datasets [12].

The WTA hash itself is a part of the Locality Sensitive Hashing (LSH) family. LSH schemes assign similar values to vectors that are close to each other and were originally designed to address the nearest neighbor problem [13].

WTA hashing works by transforming the input feature space into binary codes. In the resulting space, the Hamming distance closely correlates with rank similarity measure [14]. Furthermore, the obtained binary descriptors show invariance given slight perturbations of the original data, which makes the method a suitable basis for retrieval algorithms. Also, the WTA is based on random permutations on the data components and does not require any data-driven learning.

Fig. 2 summarizes the computation of one WTA hash. Each hash is composed of N_s sections, each one being characterized by a different permutation. That is, each section α belonging to each hash i, has its own permutation denoted by the function $permute_{i,\alpha}$.

To compute a hash, we permute the input vector $[x_1, x_2, \ldots, x_K]$ by indexing the incoming data through the hash's permutation array. Then for each section, the N_e first elements of the permuted vector are selected:$[x'_1, x'_2, \ldots, x'_{N_e}]$. We compare these elements and the index of the largest element is recorded, $\check{h}_\alpha$. This process is repeated N_s times to arrive at N_s hash sections. These sections are concatenated to form a single hash h_i.

Please cite this article as: A.H. Bakhtiary et al., Winner takes all hashing for speeding up the training of neural networks in large class problems, Pattern Recognition Letters (2017), http://dx.doi.org/10.1016/j.patrec.2017.01.001