Distributed Neural Network Training

Wojciech Dziwulski
National University of Singapore, University of Oxford

***Abstract*—This work proposes a novel method of distributing CNN training and processing between several computational units. Our approach utilises the Alternating Direction Method of Multipliers (ADMM) optimization. It minimizes the loss function similarly to the augmented Lagrangian method, but breaks down the optimized variable into two "dual" ones whose minimization is sequential. This allows for cutting down the communication overhead and hence makes the overall method much more memory efficient.**

The algorithm is validated, and its performance is evaluated, using the CIFAR10 and MNIST datasets. It is proven that the novel setup is able to accommodate a much larger batch size (by factor of 2). This is of large importance because it prospectively allows for training of much deeper network topologies with an architecture distributed between several computational units. The classification accuracy achieved by the dual setup is found to be within the 6% (CIFAR10) and 0.5% (MNIST) bound lower than the one achieved by a singular, traditional architecture. Prospectively, this allows for training much deeper network topologies in an architecture distributed between several computational units.

I. Introduction

Understanding data was universally humanity's greatest feat and the need for it is even more compelling nowadays, when the size of datasets is becoming unimaginable. Novel machine learning methods have made great advancements in inference and classification tasks, but the hunger for more robust algorithms and setups grows even quicker. A lot of recent progress is owed to an aggressive increase in the available computation power, allowing us to implement setups that were previously simply unfeasible. More power also means more complicated models, and hence the ability to represent complex functions. As we near the technical limits of single computational units though, we have to turn our research efforts to using distributed approaches allowing us to harness an ever increasing number of clock cycles. This paper presents a novel approach to distributing the computation developed by using a convex optimization algorithm known as ADMM, Alternating Direction Method of Multipliers.

II. Background

A. Machine intelligence

Deep learning is a term describing neural setups consisting of several processing layers, building up hierarchical features automatically, and hence removing the need for hand-designing them. The networks automatically learn the parameters of convolutional filters that should be applied to the raw input images in order to obtain the desired label. As with traditional neural networks, the weights are learned by computing their gradient with respect to the overall classification error, defined as the 2-norm of the predicted and expected image labels. This is done by running two complimentary algorithms - **forward** (computing the activations) and **backward** (computing the updated parameters) **propagation**.

Wojciech Dziwulski is with the Department of Electrical and Computer Engineering,National University of Singapore, 4 Enginnering Drive 3, Sinagpore (phone:65-9446-7274; email: wojciech.dziwulski@gmail.com).

B. ADMM

ADMM is a convex optimization algorithm - it tries to find a global minimum of a given loss function. It utilizes the traditional Lagrange multipliers method: $\min_x L(x) = f(x) + \boldsymbol{\lambda^T g(x)}$, but breaks it down into two parts, introducing another, "dual", variable $\boldsymbol{y}$. The loss function then becomes: $\min_{x,y} \max_\lambda L(x,y) + \lambda^T(x-y) + \beta||x-y||^2$. Hence one of our objectives is to minimize the difference between the dual variables, in addition to minimizing the overall loss function. We then proceed with the optimization sequentially, first moving ahead the optimization of x with all the other parameters fixed and then turning to the optimization of y.

III. Previous work

The two most notable works devoted to the field of distributed neural network training are those by Dean et al [1] and Chilimbi et al [2]. The first work proposes a framework called DistBelief which parallelizes the training within and between the machines as well as two suitable optimization algorithms - Downpour SGD and Sandblaster L-BFGS. Both offer an inherently parallel optimization method through asynchrounous updates of network parameters and distributing their storage. DistBelief provides model parallelism through distributing the responsibility for node parameter computation between different machines. The node parameters are stored centrally though, on a dedicated parameter server. The second paper proposes a similar approach, where the parameter and computation servers are disjoint, but where the parameter updates are carried in an asynchronous, lock-free fashion. The model is partitioned vertically, with the convolutional layers separate from the fully connected ones. The forward and back propagation are executed on concurrent threads serving different images. Interestingly, even though this introduces races and usage of not-updated parameter values the overall setup does converge which nicely demonstrates neural nets' resilience to such noise. The approach extends to even 120 machines and trains the networks to an unprecedented accuracy on the MNIST dataset.

IV. ADMM and Deep Learning

The idea above nicely extends to deep learning. Let's start with breaking down our deep neural network into two parts, duplicating one of the convolutional layers so as to be included in both of the resulting networks. The dual variables x and y will then correspond to the activations of the duplicated layers. Hence the overall optimization is going to consist of two sequential parts: 1) minimizing the difference between the layer activations, 2) minimizing the difference between the predicted and actual image labels.

V. Experimental setup

Based on the idea above, a deep neural setup loosely based on CaffeNet was produced and broken down in the fashion shown in figure 1.

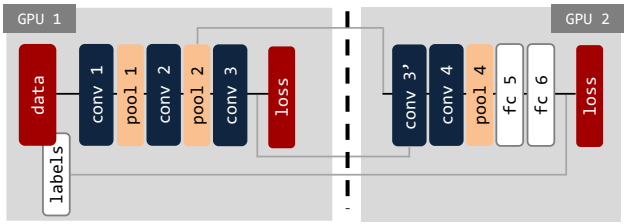

Fig. 1. The dual network deep setup.

The setup was implemented and tested using Caffe deep learning framework. The network parameters were stored on the same server, but managed separately and automatically by Caffe. The communication between the two networks was kept to minimum. The parameters that were mutually necessary for computation were the output of layer pool 2, which served as an input to the other part of the network, and the output of layer conv3' (the dual layer), which was necessary to compute the loss function for the first network.

Based on the above considerations the algorithm was finalized as:

GPU 1

```
for number_sequences:
  load conv3'.data
  for number_net1_epochs:
    net1.forward
    calculate loss=||conv3-conv3'||^2
    net1.backward
  save pool2.data
  ...
  ...
  ...
  ...
  ...
  ...
```

GPU 2

```
for number_sequences:
  ...
  ...
  ...
  ...
  ...
  ...
  load pool2.data
  for number_net2_epochs:
    net2.forward
    calculate loss = ||labels-fc6||^2
    net2.backward
  save conv3'.data
```

VI. Results

The setup above was tested using the cifar10 and MNIST datasets. The results can be measured against several performance metrics: 1) classification accuracy, 2) loss function value 3) training time 4) maximum batch size, 5) size of the model trained on each machine. Accordingly:

1) The classification accuracy achieved after convergence of the networks was within a 6% bound from the traditional setup in case of cifar10 dataset over three distinct trials. In MNIST, the accuracy was bound by 0.5%.
2) As seen in the figure 2, the loss function exhibited very similar behaviour and convergence compared in both single and dual network setups. The characteristic step pattern can be observed in the figure, corresponding to the sequential optimization.
3) The training time was much larger in case of the dual training, by a factor of around 4, depending on the trial. This is to be expected as the hard drive network communication is far from optimal. This can be easily improved, though, and was not one of the project objectives.
4) The maximum batch size that could have been fitted in the setup was increased by a factor of 2.
5) The size of the models individually trained on the GPUs was decreased as well confirming that the dual setup is much more memory efficient. Is is thus proved to be able to train much deeper models, and hence more complicated functions.

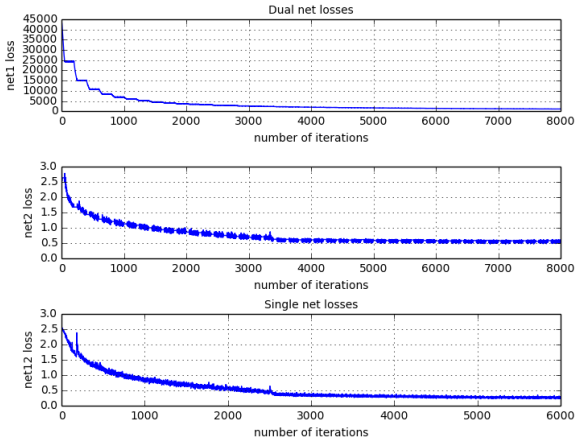

Fig. 2. The final loss function curves.

VII. Conclusions

The results above clearly prove the usefulness of the ADMM Deep Learning Distributed approach. It allows for a much more memory-efficient use of the available computational units, hence also permitting to train bigger networks and learning more complicated functions underlying the complex datasets.

The future implementations of the setup should break it down further between more computational units.

References

[1] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, A. Senior, P. Tucker, K. Yang, Q. V. Le, et al. Large scale distributed deep networks. In *Advances in neural information processing systems*, pages 1223–1231, 2012.

[2] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. Project adam: Building an efficient and scalable deep learning training system. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*, pages 571–582, 2014.