Optimizing Neural Networks for Mobile Devices

Radboud University, Nijmegen

Erdi Çallı

March 27, 2017

Abstract

Recent developments in Neural Networks(or Deep Learning) are promising. Some models are capable of accomplishing tasks as good as humans, or better. But we still lack the applications that are available to the public. The general opinion is, Neural Network models need expensive equipment. But that only applies to the training process, where the network learns from data. But using the trained model for inference is easier. There are also methods to reduce the computational complexity of a model. With these methods, we may be able to use cheap compute devices(i.e. Mobile Phones) for inference. Thus making some models available to public.

Contents

Chapter 1

Introduction

Recent state of the art Deep Learning models are surpassing previous methods. Fields such as; computer vision, automatic speech recognition, natural language processing, speech recognition, and bioinformatics make use of these models. They use deep models consisting of many layers (e.g. 152 layers in [HZRS15]), many parameters in each layer, and as a result, a lot of Floating Point Operations to run Inference (e.g. 11.3×10^9 in [HZRS15]). In contrast, mobile devices have limited processing power and memory. Also the best practice is to provide a fluent user experience with low response time. Thus, we should change these models to provide a good user experience. There is research on methods to define optimized models or optimize a given model. These methods consist; pruning unimportant parameters, using less bits to represent parameters, or using less parameters by using more optimized structures.

In this research we are going run experiments to answer;

1. Which models are running slow in Mobile Devices?
2. Why these models are running slow?
3. Which methods can we use to optimize these models?
4. What is the trade off of using these methods?
5. Why an optimization technique is working or not on a model?
6. Can we define a more optimized model for the same task?
7. How can we combine different optimization techniques?

8. Are these optimized models efficient enough to run in Mobile Devices?

things to be explained: FC layer, activation, bias, training dataset, (neuron or node)

1.1 Recent Studies

Artificial Neural Networks (ANN) have several parameters such as number of hidden layers, number of neurons in a layer, or the structure of a layer. Until now, we have seen different combinations for these parameters. For example, [SZ14] introduces a model called VGGNet. VGGNet introduces more layers (16 to 19) than the previous models. They show how this parameter effects the accuracy. [HZRS15] introduces the residual connections. This new connection between layers is capable of stacking more layers than before. Training up to 152 layers, they show superior accuracy. [ZK16] compares having higher number of neurons in each layer to having more layers. Each combination resulting in a unique model with a different accuracy level. In contrast to all these, [SLJ+14] suggests something different. Having a good harmony within the network works better than having more parameters. Supporting that, [CPC16] does a detailed comparison of different models. They show that, increasing the number of hidden layers or the number of neurons in a layer does not necessarily increase the accuracy.

Following these, we think that, some models are over-parameterized. Meaning they contain parameters that they are not making use of. Therefore, they are making unnecessary computations with them.

Chapter 2

Methods

2.1 Dependencies

2.1.1 Tensorflow

In our research, we will strictly use Tensorflow [AAB+16].

What is tensorflow, why we chose it, what are the advantages of using it, what are the limitations that come with it

2.2 Pruning

Pruning aims to reduce the number of operations by deleting the parameters that has low or no impact in the result. Studies show that applying this method in an ANN is effective in reducing the model complexity, improving generalization, and they are effective in reducing the required training cycles. In our experiments we will try to reproduce these effects. To visualize these methods, let's think of two fully connected layers, $\mathbf{l_1}$ and $\mathbf{l_2}$. $\mathbf{l_1}$ is the input of this operation and it consists of N values, $\mathbf{l_1} = (l_{11}, l_{12}, ..., l_{1N})$. $\mathbf{l_2}$ is the output of this operation consists of M values, $\mathbf{l_2} = (l_{21}, l_{22}, ..., l_{2M})$. Between these two layers, there is a weight matrix W with size $N \times M$. The operation, that we want to optimize is, $\mathbf{l_2} = \mathbf{l_1}W$.

maybe explain in more detail and give examples of pruning algorithms here. (e.g. Optimal Brain Damage, Second order derivatives for network pruning: Optimal Brain Surgeon, Optimal Brain Surgeon and general network pruning, SEE Pruning Algorithms-a survey from R. Reed)

2.2.1 Pruning Individual Weights

With this subcategory of pruning algorithms we want to optimize the number of floating point operations by removing some values from W. Theoretically, it makes sense to remove individual scalars from W, and exclude operations related to them. This would ideally reduce the required number of floating point operations. But in our library of our choice, Tensorflow, matrix multiplication implementation `tf.matmul` do not consider such a change. It takes two fixed size matrices, and does the computations using all of their values. Tensorflow also has another matrix multiplication operation, `tf.sparse_tensor_dense_matmul`. This operation takes a sparse matrix and a dense matrix as inputs and outputs a dense matrix. To implement this method, we could convert W to a sparse tensor after pruning the weights. But, Tensorflow documentations about this method state;

- Will the SparseTensor A fit in memory if densified?
- Is the column count of the product large ($>> 1$)?
- Is the density of A larger than approximately 15%?

"If the answer to several of these questions is yes, consider converting the SparseTensor to a dense one.". In our terms, SparseTensor A is corresponding to the pruned version of W.

Since W was already dense before, we can assume that the answer to the first question is yes. The column count of our product is M which is much larger than 1 in some cases. Also we don't know anything about the density of pruned version of W. Looking at these facts, we are assuming that implementing this operation will be problematic. Instead of delving deeper into these problems to evaluate this method, we will move on to other methods.

2.2.2 Activation Based Pruning

Activation based pruning, works by looking at individual values in layers, and prunes the layer and corresponding weight row/columns completely. To visualize this, we will assume that the fully connected layers we have defined are, trained to some extent, and activated using ReLU activations. With this definition, if we apply our dataset and count the number of activations in $\mathbf{l_1}$ and $\mathbf{l_2}$, we may realize that there are some neurons that are not being activated at all. By removing these neurons from the layers, we can reduce the number of operations to some extent. This removal operation is done by removing the non-activated neurons from $\mathbf{l_1}$ and $\mathbf{l_2}$, followed by removing corresponding columns and rows from W.

Chapter 3

Results

Chapter 4

Discussion

Chapter 5

Conclusion

Bibliography

[AAB+16] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

[CPC16] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. An analysis of deep neural network models for practical applications. 05 2016.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. 12 2015.

[SLJ+14] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. 09 2014.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 09 2014.

[ZK16] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. 05 2016.