Tudor Paraschivescu
Churchill
tp423

Computer Science Tripos Part II Project Proposal

MIRACLE Neural Network Exchange Format

12^{th} *of October, 2018*

Project Originator and Supervisor: Márton Havasi

Director of Studies: John Fawcett

Overseers: Anuj Dawar, Simon Moore, and Andrew Moore

Introduction and Description of the Work

Machine Learning is concerned with making predictions on unseen data using probabilistic models. Deep Neural Networks (DNN) are an example model class that are successful due to their high flexibility. They represent the state of the art in many image recognition/NLP tasks.

The problem with DNNs is their sizes (eg. an imagenet classifier is around 500 MB) as it puts a high strain on energy consumption, bandwidth, and storage space. Fortunately, there has been a lot of research done on compressing DNNs. Current algorithms are able to achieve a compression rate bigger than 500x.

A state-of-the-art algorithm for compression is MIRACLE (described in [1]). The approach is based on relaxing weight determinism; i.e treat each weight as a univariate Gaussian distribution for which we train the mean and variance. We condition that the trained model will not be too "different"[1] from a chosen multivariate distribution that both parties have access to. The compressed model will be a seed which, when fed in the aforementioned shared multivariate distribution, will recreate the model. The goal of this project is to develop a neural network exchange format based on the MIRACLE algorithm that allows practitioners to drastically compress the size of their DNNs.

[1] Condition that the KL distance does not exceed a set limit

Starting Point

I have some experience with the open-source software library TensorFlow.

The algorithm is described in the paper "Minimal Random Code Learning: Getting Bits Back from Compressed Model Parameters" [1].

The research code can be found at https://github.com/cambridge-mlg/miracle. This code is limited as it hardcodes the architecture of the LeNet5 and VGG-16 models, it does not output files (only used for outputting the graphs in the paper [1]), and it does not handle decompression.

The paper and the code will be used as a starting point in order to get a deeper understanding of the MIRACLE algorithm.

Substance and Structure of the Project

The aim of this project is to develop an exchange format for neural networks based on the MIRACLE algorithm. This will be done by developing a library which can be used for creating Neural Network Architectures. It should allow users to:

- Define an architecture containing feed-forward layers, convolutional layers, and pooling layers.
- Define the data in the computation graph by using placeholders.
- Compress a trained model. This outputs a '.mrcl' file.
- Load a model from a '.mrcl' file.

The project has the following main parts:

1. Studying and understanding the MIRACLE algorithm. This is a very technical endeavor as it also involves studying and understanding deep learning, image recognition, Bayesian Inference, variational inference, coding theory, and the bits-back argument.
2. Learn how to use low-level TensorFlow primitives and how to use cloud computing to develop software.
3. Implementing the compression for a toy architecture (e.g. linear regression). Implement the MNIST LeNet5 architecture (without compression).

4. Developing and testing the basic implementation of the compression algorithm. The basic MIRACLE algorithm has an exponential complexity in the number of bits of the compressed file, thereby making it unusable on any real-world architecture.

5. Developing and testing the practical implementation of the algorithm. This uses the following tricks to speed up:

    - Condition blocks of weights not to be too "different" from the respective multivariate Gaussian distribution instead of conditioning the whole model. Compress the individual blocks into seeds instead of the whole model.
    - Force randomly selected weights to be the same.

6. Evaluation using a toy architecture trained on a toy dataset (e.g. randomly sampled from a normal distribution) and on the LeNet5 architecture trained on MNIST.

7. Writing the Dissertation.

The implementation language will be Python. The TensorFlow library will be used extensively.

Evaluation Procedure and Success Criteria

The project will be evaluated on two architecture–dataset pairs:

- A toy architecture and dataset. This will be a small architecture used to evaluate the basic implementation in feasible time.
- The LeNet5 architecture and the MNIST dataset. Only the practical implementation will be evaluated on this pair.

The evaluation will be done by choosing different compression rates, compressing the architecture, and measuring the performance of the decompressed model. The performance will be given by the error rate on the test data. We will compare the performance with the performance of The Uncompressed model, Deep Compression ([2]), Weightless Compression ([3]), and Bayesian Compression ([4]). To get the performance of the references I will either look online or use a library that implements these algorithms.

The project will be a success if the final library can be used to define the LeNet5 MNIST architecture and if the implemented compression method is evaluated against the reference compression methods using the compression rate and performance metrics.

Extensions and Fallback

Extensions

The following may be implemented as extensions of the primary core project depending on the intermediary result obtained and the timeliness of its completion:

1. Evaluate on the VGG-16–CIFAR-10 architecture–dataset pair.
2. Make the library work seamlessly behind TensorFlow. This would allow the user to define the computation graph in TensorFlow and use the library only for training and compression.
3. Prepare a demo for the NIPS 2018 workshop on Compact Deep Neural Networks with industrial applications (acceptance notification on 29^{th} of October).
4. Experiment with the tricks used to make the algorithm practical and log the outcomes.

Fallback

In case of unforeseen difficulties that make the core project no longer able to be completed in the remaining timeframe, the fallback project will be to implement a pruning quantization based compression algorithm such as the Deep Compression algorithm described in [2]. Deep Compression is a straightforward algorithm that relies on a three-stage pipeline: pruning, trained quantization, and Huffman coding.

Timetable and Milestones

The following table shows the timeline for the project, specifying the proposed work and the deliverables in each work package. Shaded regions correspond to periods outside term.

Work package	**Proposed work**	**Deliverable(s)**
19/10 – 01/11	Study the MIRACLE paper and the other algorithms required to understand it. Get familiar with TensorFlow and Python.	Document detailing my initial understanding of the algorithm.

02/11 – 15/11	Initial experiments with the MIRACLE compression algorithm.	Set up the developing environment. Implementation of the LeNet5 architecture without compression. Partial implementation of the compression algorithm on linear regression. Linear Regression (or other toy architecture).
16/11 – 29/11	Start working on the core library. Implement the ability for users to define a computation graph that can use the basic compression algorithm.	A library that allows the user to define a computation graph and train the model on data.
30/11 – 13/12	Implement and test the basic compression algorithm. Create a demo for the NIPS 2018 workshop.	Demo for the workshop. The library now allows the users to compress their model to a '.mrcl' file and to load a '.mrcl' file.
14/12 – 27/12	Two weeks Christmas break. Time to catch up if fallen behind, get ahead to work on extensions, or rest.	Slack period.
28/12 – 10/01	Start the implementation of the practical compression algorithm by working on the first trick used to make it practical.	The user can now specify a block size when training. This should highly speed up compression.
11/01 – 24/01	Continue the implementation of the practical compression algorithm by implementing the second trick and write the progress report and presentation.	The user can now specify a number of weights that will be identical when training. This should highly increase the compression size. Submitted the progress report.
25/01 – 07/02	Put the final touches to the core library. Evaluate on the toy dataset and on LeNet5. If finished early, start working on the extensions.	The core deliverables are now finished. Compared the performance of the compressed model when using the library vs when using the reference compressions.
08/02 – 21/02	Write the "introduction" and "preparation" sections of the dissertation.	Draft versions of the "introduction" and "preparation" sections sent to supervisor for comments.
22/02 – 07/03	Write the "implementation" and "evaluation" sections of the dissertation.	Draft versions of the "implementation" and "evaluation" sections sent to supervisor for comments.

08/03 – 21/03	Integrate supervisor comments into body of dissertation. Complete the "conclusions" section and prepare the other sections and attachments.	Conclusion section and second draft sent to supervisor.
22/03 – 04/04	Two weeks Easter Break. Used for exam revision and awaiting supervisor feedback.	Slack period.
05/04 – 18/04	Make final alterations to the dissertation and exam revision.	Submitted dissertation.
19/04 – 02/05	Slack (in case of falling behind), exam revision	Slack (in case of falling behind), exam revision
03/05 – 16/05	Slack (in case of falling behind), exam revision	Slack (in case of falling behind), exam revision

Resource Declaration

- **My own laptop**

  Purpose: Writing the project code, using the GPU for local development when the workload is small, and writing the dissertation.

  Specifications: NVIDIA GEFORCE GTX 960M, Intel I7 4750-HQ, 16GB RAM, Windows 10.

  Failure Plan:

  – Commit code to GitHub, copy the code to my Google Drive daily, and to an external USB stick at least once a week. In case of an irreparable hardware fault, I will use an MCS machine provided by DCST.

  – The dissertation will be written in Overleaf and saved to my computer and to an external USB stick at least once a week.

  Version control plan: I will use GitHub for version control.

  I accept full responsibility for this machine and I have made contingency plans to protect myself against hardware and/or software failure.

- **Google Cloud**

  Purpose: Using the GPUs on the cloud for development. I use Google Cloud because it has better integration with a local development environment than the HPC Wilkes2 cluster.

  Specifications: I have 300$ worth of credit.

Failure Plan: In case the service can no longer be accessed I can use Amazon AWS, Microsoft Azure, or the HPC Wilkes2 cluster.

- **HPC Wilkes2 Cluster**

  Purpose: Evaluating the model.

  Specifications: I currently have a COMPUTERLAB-SL2-GPU account with HPC.

  Failure Plan: Evaluate on Google Cloud, Amazon AWS, or a MCS machine with a GPU.

References

[1] Marton Havasi, Robert Peharz, and José Miguel Hernández-Lobato. Minimal random code learning: Getting bits back from compressed model parameters. *arXiv preprint arXiv:1810.00440*, 2018.

[2] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[3] Brandon Reagen, Udit Gupta, Robert Adolf, Michael M Mitzenmacher, Alexander M Rush, Gu-Yeon Wei, and David Brooks. Weightless: Lossy weight encoding for deep neural network compression. *arXiv preprint arXiv:1711.04686*, 2017.

[4] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.