Accelerating Machine Learning Research with MI-Prometheus

Tomasz Kornuta **Vincent Marois** **Ryan L. McAvoy** **Younes Bouhadjar**
Alexis Asseman **Vincent Albouy** **T.S. Jayram** **Ahmet S. Ozcan**
IBM Research AI, Almaden Research Center, San Jose, USA
{tkornut, vmarois, mcavoy, byounes, jayram, asozcan}@us.ibm.com
{alexis.asseman, vincent.albouy}@ibm.com

Abstract

The paper introduces *MI-Prometheus* (Machine Intelligence - Prometheus), an open-source framework aiming at *accelerating Machine Learning Research*, by fostering the rapid development of diverse neural network-based models and facilitating their comparison. In its core, to *accelerate the computations* on their own, *MI-Prometheus* relies on PyTorch and extensively uses its mechanisms for the distribution of computations on CPUs/GPUs. The paper discusses the motivation of our work, the formulation of the requirements and presents the core concepts. We also briefly describe some of the problems and models currently available in the framework.

1 Motivation

The AI community has developed a plethora of tools and libraries facilitating building and training models, e.g. Torch [CBM02], Theano [BLP+12], Chainer [TOHC15] or TensorFlow [ABC+16]. While those tools significantly *accelerate computations* (e.g. parallelization during data loading, utilization of GPUs for matrix algebra and distribution of computational graphs on many GPUs), they are still not sufficient, considering the evolution of the Machine Learning domain. Three partially unsolved issues worth mentioning are as follows. First, when developing a new model for a new problem, one should compare its performance against some baselines, i.e. other, well established models, ideally using the same experiment setup (e.g. standardization of the object detection API in TensorFlow [HRS+17]). Second, the recent trends in building models which are trained jointly on problems from very different domains (e.g. as presented in [KGS+17]) indicate that the standardization of models specific to a single problem is not sufficient anymore: the same standardization should be done for problems as well. This was one of the motivations for the development of Tensor2Tensor [VBB+18], built on top of TensorFlow. However, a third issue still remains: How can one efficiently manage running batches of experiments, such as training a number of different models on a number of different problems? An ideal tool should provide a way of not only automating running those kinds of experiments, but also facilitate changing the (hyper)parameters of the problems, models, training procedures etc. on-the-fly, without the need for changing the actual problem or model code implementation. Doing so while also ensuring a minimum of varying factors is key to a valid comparison of models on the same set of problems.

To address these issues, we designed *MI-Prometheus*[1], a framework based on PyTorch [PGC+17]. The framework aims at *Research acceleration* and enables running many experiments in parallel, facilitates comparing different models etc. The key features of *MI-Prometheus* are as follows:

- A configuration management relying on (optionally nested) human-readable YAML files,

[1] https://github.com/ibm/mi-prometheus/

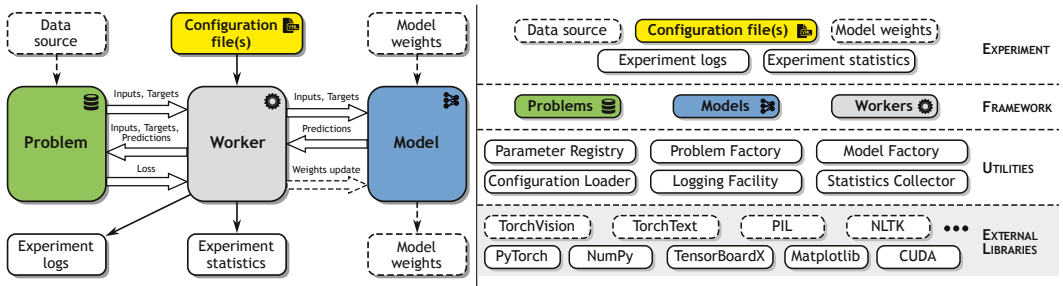
Submitted to 32nd Conference on Neural Information Processing Systems (NIPS 2018). Do not distribute.

- Standardization of the interfaces of components needed in a typical deep learning system: problems, models architectures, training/test procedures etc.,
- Automated tools ensuring that a given model and problem are compatible,
- Reusable scripts unifying the training & test procedures, enabling reproducible experiments,
- Automated tools for collecting statistics and logging the results of the experiments,
- A set of scripts for performing batches of experiments on collections of CPUs and/or GPUs,
- A collection of diverse problems, currently covering most of the actively explored domains,
- A collection of (often state-of-the-art) models,
- A set of tools to analyze the models during training and test (displaying model statistics and graph, dynamic visualizations, export of data to TensorBoard).

2 System description

When training a model, people write programs which typically follow a similar pattern. The analysis of that pattern led us to the formalization of the core concepts of the framework:

- Problem: a dataset or a data generator, returning a batch of inputs and ground truth labels used for a model training/validation/test,
- Model: a trainable model (i.e. a neural network),
- Worker: a specialized application that instantiates the Problem & Model objects and controls the interactions between them.



(a) Core concepts

(b) Architecture

Figure 1: Core concepts and architecture of the MI-Prometheus framework.

Given the plurality of the models and training parameters (so called hyper-parameters), which might result in varying model performance, practitioners in the field usually perform hyper-parameter search. Additionally, problems often need a run-time parameterization, the most common values being the paths and filenames of the input data. Those considerations led us to two additional definitions:

- Configuration file(s): YAML file(s) containing the parameters of the Problem, Model and training procedure (e.g. terminal conditions, curriculum learning parameters), divided into several sections,
- Experiment: a single run (training or test) of a given Model on a given Problem, using a specific Worker and Configuration file(s).

Fig. 1a displays four of the core concepts and emphasizes the most important interactions between them. The dotted elements indicate optional inputs/outputs/dataflows, e.g. the test worker (called Tester) typically loads an existing model but does not update its weights, thus has no need for saving it back to a file.

From an architectural point of view, *MI-Prometheus* can be seen as four stacked layers of interconnected modules (Fig. 1b). The lowest layer is formed by the external libraries that *MI-Prometheus* relies on, primarily PyTorch, NumPy and CUDA, which are extensively used (the latter one indirectly, to be precise) by the models, problems and workers.

The second layer includes all the utilities that we have developed internally, such as the Parameter Registry (a singleton offering access to the registry of parameters), the Application State (another singleton representing the current state of application, e.g. whether the computations should be done on GPUs or not), factories used by the workers for instantiating the problem and model classes (indicated by the configuration file and loaded from the corresponding file), logging facilities etc.

Next, the Framework layer contains the models, problems and workers, i.e. the three major components required for the execution of an experiment. The problem and model classes are organized following specific hierarchies, using inheritance to facilitate their further extensions.

The framework offers a set of diverse problems, starting from classical image classification (such as MNIST [LBBH98] and CIFAR10 [KH09]), machine translation (TranslationAnki) or Visual Question Answering (e.g. CLEVR [JHvdM+17] and SortOfCLEVR [SRB+17]). It also contains several algorithmic problems such as SerialRecall and ReverseRecall used e.g. in [GWD14].

MI-Prometheus contains a suite of models that can be categorized as follows. The first category includes several classic models, such as AlexNet [KH09] or the LSTM (Long-Short Term Memory) [HS97]. The second category includes Memory-Augmented Neural Networks [Sam17], e.g. NTM (Neural Turing Machine) [GWD14] or the DNC (Differentiable Neural Computer) [GWR+16] and ThalNet [HIDH17]. The third category includes models designed for the VQA (Visual Question Answering) [AAL+15] problem domain. Here, *MI-Prometheus* currently offers a few baselines and several state-of-the-art models such as SANs (Stacked Attention Networks) [YHG+16] RNs (Relational Networks) [SRB+17] and MAC (Memory Attention Composition) [HM18].

Our framework currently offer five types of workers. The Basic Workers are scripts that execute a certain task given a Model and a Problem. They are related to either the training (Trainer) or the testing procedure (Tester) and support CPU and GPU working modes. Additionally, there are five Grid Workers, i.e. scripts that do not run experiments on their own but manage sets of experiments on grids of CPUs/GPUs. Those are of three types: two Grid Trainers (separate versions for collections of CPUs and GPUs), two Grid Testers (similarly) and a single Grid Analyzer.

Finally, the Experiment layer includes the configuration files, along with all the required inputs (such as the files containing the dataset, the files containing the saved model checkpoints with the weights to be loaded etc.) and outputs (logs from the experiment run, CSV files gathering the collected statistics, files containing the checkpoints of the best obtained models etc.).

3 Summary

The paper introduced *MI-Prometheus*, a framework designed with the goal of accelerating Machine Learning research. In order not to reinvent the wheel, we decided to build the framework on top of PyTorch, which has recently been gaining more and more popularity. PyTorch on its own is a great tool for developing new models and problems and offers several mechanisms enabling the *acceleration of computations*. Our experiences led us to the formulation of a set of requirements enabling the *acceleration of research*. In particular, the most important ones are: flexible configuration management and spanning experiments over grids of CPUs and/or GPUs. *MI-Prometheus* fulfills these requirements and provides new solutions such as: nested configuration files, registry of parameters, dynamic handshaking of data definitions and grid workers.

The unification of training and test procedures enables reproducible experiments and makes it easy to run several models on the same problem (or a set of problems). We have proposed a simple taxonomy of problems, which follows a domain-based hierarchy. This results in an increase of code reuse and facilitates adding new ones. The diversity of problems and models, including examples from diverse research domains, proves both the flexibility of the *MI-Prometheus* framework and its usefulness.

Finally, one usually needs to train/test a given model to fully understand its limitations, which in turn may trigger ideas on improvements and new models. *MI-Prometheus* provides several useful tools for getting insights on the models, e.g. logging utilities, dynamic visualization of model operation, the collection of statistics, export of statistics to TensorBoard. From our own experience, we know that diverse ways of getting insights into the model offered by *MI-Prometheus* are complementary and extremely useful when building novel, trainable models.

References

[AAL+15] Stanislaw Antol, Aishwarya Agrawal, Jiasen Lu, Margaret Mitchell, Dhruv Batra, C Lawrence Zitnick, and Devi Parikh. Vqa: Visual question answering. In *Proceedings of the IEEE international conference on computer vision*, pages 2425–2433, 2015.

[ABC+16] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. TensorFlow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[BLP+12] Frédéric Bastien, Pascal Lamblin, Razvan Pascanu, James Bergstra, Ian Goodfellow, Arnaud Bergeron, Nicolas Bouchard, David Warde-Farley, and Yoshua Bengio. Theano: new features and speed improvements. *arXiv preprint arXiv:1211.5590*, 2012.

[CBM02] Ronan Collobert, Samy Bengio, and Johnny Mariéthoz. Torch: a modular machine learning software library. Technical report, Idiap, 2002.

[GWD14] Alex Graves, Greg Wayne, and Ivo Danihelka. Neural Turing Machines. *arXiv preprint arXiv:1410.5401*, 2014.

[GWR+16] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al. Hybrid computing using a neural network with dynamic external memory. *Nature*, 538(7626):471, 2016.

[HIDH17] Danijar Hafner, Alexander Irpan, James Davidson, and Nicolas Heess. Learning hierarchical information flow with recurrent neural modules. In *Advances in Neural Information Processing Systems*, pages 6724–6733, 2017.

[HM18] Drew A Hudson and Christopher D Manning. Compositional attention networks for machine reasoning. In *International Conference on Learning Representations (ICLR)*, 2018.

[HRS+17] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4, 2017.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[JHvdM+17] Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 1988–1997. IEEE, 2017.

[KGS+17] Lukasz Kaiser, Aidan N Gomez, Noam Shazeer, Ashish Vaswani, Niki Parmar, Llion Jones, and Jakob Uszkoreit. One model to learn them all. *arXiv preprint arXiv:1706.05137*, 2017.

[KH09] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[LBBH98] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[PGC+17] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[Sam17] Mostafa Samir. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*, chapter Memory Augmented Neural Networks. " O'Reilly Media, Inc.", 2017.

[SRB+17] Adam Santoro, David Raposo, David G Barrett, Mateusz Malinowski, Razvan Pascanu, Peter Battaglia, and Tim Lillicrap. A simple neural network module for relational reasoning. In *Advances in neural information processing systems*, pages 4967–4976, 2017.

[TOHC15] Seiya Tokui, Kenta Oono, Shohei Hido, and Justin Clayton. Chainer: a next-generation open source framework for deep learning. In *Proceedings of workshop on machine learning systems (LearningSys) in the twenty-ninth annual conference on neural information processing systems (NIPS)*, volume 5, pages 1–6, 2015.

[VBB+18] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, et al. Tensor2tensor for neural machine translation. *arXiv preprint arXiv:1803.07416*, 2018.

[YHG+16] Zichao Yang, Xiaodong He, Jianfeng Gao, Li Deng, and Alex Smola. Stacked attention networks for image question answering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 21–29, 2016.