AMLA: an AutoML frAmework for Neural Network Design

Purushotham Kamath pukamath@cisco.com
Abhishek Singh abhishs8@cisco.com
Debo Dutta dedutta@cisco.com
Cisco Systems, San Jose, CA, USA

Abstract

AMLA is an Automatic Machine Learning frAmework for implementing and deploying neural architecture search algorithms. Neural architecture search algorithms are AutoML algorithms whose goal is to generate optimal neural network structures for a given task. AMLA is designed to deploy these algorithms at scale and allow comparison of the performance of the networks generated by different AutoML algorithms. Its key architectural features are the decoupling of the network generation from the network evaluation, support for network instrumentation, open model specification and a microservices based architecture for deployment at scale. In AMLA, AutoML algorithms and training/evaluation code are written as containerized microservices that can be deployed at scale on a public or private infrastructure. The microservices communicate via well defined interfaces and models are persisted using standard model definition formats, allowing the plug and play of the AutoML algorithms as well as the AI/ML libraries. This makes it easy to prototype, compare, benchmark and deploy autoML algorithms in production. AMLA is currently being used to deploy an AutoML algorithm that generates Convolutional Neural Networks (CNNs) used for image classification.

Keywords: AutoML, Neural Networks, Framework, CNN,

1. Introduction

AutoML techniques for neural networks have evolved significantly in the last few years with a number of new techniques proposed as well as advances made in existing ones. Examples include architecture search with recurrent networks and reinforcement learning (Zoph and Le (2017)), evolutionary networks (Real et al. (2018), Floreano et al. (2008)) and several others. These AutoML algorithms generally operate by iterating through two phases shown in Figure 1.

- Network generation: In this phase a network is generated based on search, evolution or other techniques.
- Network assessment: In this phase the network is trained and performance is evaluated. Training and evaluation are generally implemented using an ML/AI library/framework such as TensorFlow (Abadi and others. (2015)), PyTorch(Paszke et al. (2017)), or others. This step also involves *network instrumentation* to collect metrics during training or evaluation.

AutoML algorithms for neural networks generally iterate through these phases, continuously refining the network structure as the objective function improves. Results from the

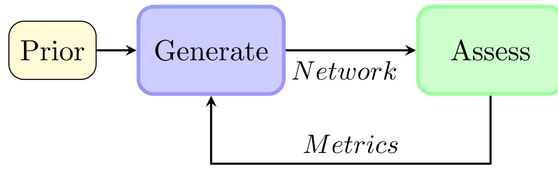

Figure 1: Abstract view of AutoML for neural network design

evaluation phase are collected and passed as input to the generation phase. The accuracy of the network is generally used as the objective function although some techniques use other metrics such as weights(Brock et al. (2017)) or featuremap statistics.

AMLA was created during the development of an AutoML algorithm (Kamath et al. (2018)) with the goals of being able to easily update and replace the algorithm, being independent of the AI/ML library and being able to deploy at scale.

2. Related Work

There has been considerable work in AutoML tools for machine learning algorithms. These tools select an algorithm from a fixed set of known models (*e.g.* SVM, Naive Bayes, Random Forest) and search for the optimal hyperparameters. Examples of machine learning AutoML tools include TPOT((Olson et al., 2016)), AutoWeka (Kotthoff et al. (2017)) and auto-sklearn(Feurer et al. (2015)). TPOT is a machine learning algorithm search tool that searches across multiple models. AutoWeka is an AutoML tool that uses Bayesian optimization to identify the best machine learning algorithm. It is built on top of Weka, a popular machine learning library. Auto-sklearn uses Bayesian optimization, meta-learning and ensemble construction to find the appropriate algorithm and hyperparameters. It is built on top of sklearn, a popular Python machine learning library. These tools differs from neural network search AutoML (the problem that AMLA addresses) which is a search for a neural network structure starting from nothing (or from a prior architecture).

In the area of neural networks there has been significant work in neural network hyperparameter optimization. Examples include Spearmint(Spearmint (2016)), MOE(MOE (2015)). These tools use Bayesian optimization and other techniques to optimize the hyperparameters of a neural network such as learning rate. However Bayesian optimization and other optimization strategies are hard to apply to optimize network structure because of the non differentiable nature of network structure.

Several algorithms have been proposed for neural architecture search, but to the best of our knowledge there does not exist openly available software to deploy or benchmark these algorithms.

3. Motivation

The design of AMLA was driven by two needs: The first was the need to be able to run a neural network AutoML system at scale. Recent work in AutoML has shown state of the art results on image classification, but the algorithms are resource intensive, some needing thousands of nodes(Zoph and Le (2017)) for the search task. This motivates the need for a scalable infrastructure to deploy and run the AutoML system.

The second was the need to compare different AutoML algorithms. This is a task that often arises when proposing a new algorithm (Real et al. (2018)). It is often difficult to compare different algorithms because of differences unrelated to the core network structure (e.g. CNNs may use different stem cells or classification blocks or may use different hyperparameters or different search algorithms may start with different priors, or training/evaluation data may undergo preprocessing). It is also difficult to compare performance improvements or generation time improvements if two algorithms are executed on different infrastructures - the results need to be normalized before being compared. Independently built tools rarely execute in a comparable way on the same infrastructure unless specifically designed to do so. Benchmarking of neural network models is a complex task (MLPerf (2018)) - while models have traditionally been compared using accuracy, in recent times, training time, inference time and model sizes have becoming increasingly important, necessitating the comparison on a common infrastructure.

The need for scalability lead to the decision to separate the generation function from the assessment function by designing them as microservices with well defined interfaces. This decoupling allows each to be deployed independently and to be scaled independently. Each function is implemented as a microservice with standard interfaces, capable of being deployed in containerized environment using container deployment tools.

The need for making it easy to compare AutoML algorithms has lead to two design decisions: decoupling the AutoML algorithm from the AI/ML libraries and from the model specification. Neural architecture search algorithms that are currently available as open source are often built as monolithic software with the two functions tightly integrated. This makes it easy to reproduce a single piece of work, but also makes it harder to do a fair comparison across tools, which is often needed when comparing different algorithms.

When building an AutoML framework, a reasonable question to ask is whether the AutoML framework should be embedded in the AI/ML library. Several AI/ML libraries have emerged in the last few years each with its own high level abstractions and low level design. Some of the design decisions (such as reverse mode auto differentiation in PyTorch) may make it easier to implement AutoML within the AI/ML library, but not all libraries support them. Other libraries are easy to instrument to collect metrics. A "good" AutoML framework should remain agnostic to the AI/ML library, allowing the user to pick the library that is best suited to implement the network.

An independent and open model specification standard is particularly important for the framework as the networks generated need to be compared across the AutoML generation algorithms, both visually as well as through other methods such as parameter and operations counts. AMLA subscribes to this philosophy and attempts to separate the specification of the generated model from the generator, by using portable model standards such as ONNX(ONNX (2018)).

4. System Architecture

Most AutoML systems for neural networks are feedback control systems. Figure 1 shows an abstract view of an AutoML system for neural architecture search. A network is generated (initially from a prior, which may be the empty set), and then its performance is assessed. Measurements from the assessment are fed back to the generation phase which generates a new network. As the algorithm iterates, the generated networks improve in performance. AMLA's design was influenced by the need for scalability as well as the ability to compare different AutoML algorithms.

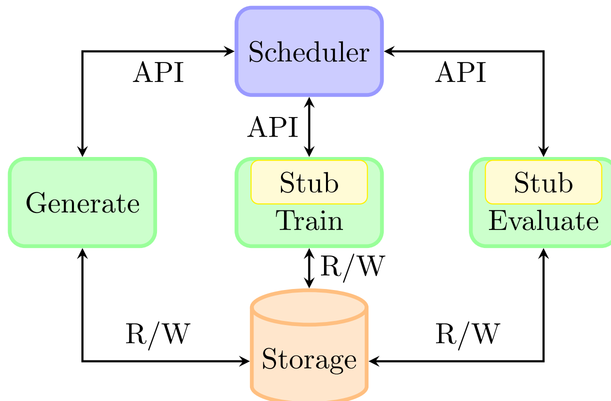

Figure 2: AMLA framework

Figure 2 shows the architecture of the AMLA framework. It consists of 4 main services along with a distributed key-value store used for persistent storage. The services and their responsibilities are:

- Generate: Generates the networks using an AutoML algorithm
- Train: Trains the generated networks
- Evaluate: Evaluates the trained networks
- Scheduler: Coordinates and schedules tasks on the other services

The services communicate via REST APIs. The scheduler spawns the generate/train/evaluate process and they make a callback to the scheduler once they complete their run. The models and control parameters generated by the AutoML algorithm are stored in a model specification format. The current implementation uses a JSON format. The Train and Evaluate services have stubs which convert between AMLA's model specification format and the model specification format understood by the AI/ML library. All persistent data (models, control parameters) are stored in the persistent key-value store. Data stored in the persistent store is referred to in the API calls using its key in the key-value store.

The Scheduler and the library stubs for the Train and Evaluate services are written in Python while the Generate service (the autoML algorithm) may be written in any language.

4.1. Scheduler

The Scheduler service coordinates execution of the Generate, Train and Evaluate services by calling their interface functions to trigger functions and collect status and results. It also manages a deployer that controls the deployment of the other microservices. As shown in the Figure 2, running one cycle of AutoML algorithm involves spawning three different workloads which can vary in their runtime as well as resource consumption. Our current deployment is on a bare metal server with work underway to support containerized deployments using Kubernetes/Kubeflow(Kubeflow (2018)). In a containerized environment, the framework containerizes the three workloads (Generate/Train/Evaluate) and the Scheduler spawns them on Kubeflow which takes care of workload placement based on resources(*e.g.* GPU, RAM) available.

4.2. Generate

The Generate service is responsible for the generation of network model using an AutoML algorithm. In addition to generating the network, it also generates a number of *control parameters* that control the training and evaluation phases (such as training epochs, evaluation epochs, frequency of evaluation etc.) The network model and the control parameters are written in a model specification file (a JSON file) stored in persistent storage. The training and evaluation stubs handle converting the network model in this format to a network that is instantiated by the ML/AI library (support for network specification using the ONNX format is underway). New neural architecture search algorithms can be implemented by adding new Generate services, implemented in any language as long as they supports the Generate interface (described in Section 4.4.) and the model specification format. *E.g.* AMLA currently provides a generate service that implements a recently proposed neural construction method (Kamath et al. (2018)).

4.3. Train/Evaluate

The Train and Evaluate services convert the interface calls from the Scheduler to function calls that use the ML/AI libraries. The conversion is performed by the stubs, while the training/evaluation are handled by the AI/ML libraries. The stubs are written in a language supported by the ML/AI library (Python for Tensorflow and PyTorch). AMLA supports network instrumentation *i.e.* the collection of various metrics during training/evaluation (as long as the AI/ML library supports it), not just the objective function *i.e.* accuracy. Network instrumentation is implemented in the stubs. The metrics are stored in the persistent store. This allows the Generate service to implement algorithms, such as SMASH(Brock et al. (2017)), that generate structures based on metrics other than accuracy .

4.4. Interfaces

All communication is between the scheduler and the other services is through non blocking REST API calls with support for JSON data encoding. Future releases may support gRPC interfaces with Protobufs. The other services (Generate, Train, Evaluate) do not communicate among themselves. The Scheduler calls the service API and each request generates an id for the request which can be used by the Scheduler in subsequent calls to retrieve state/results.

5. Applications

AMLA was built to enable the evaluation of an AutoML algorithm for neural architecture search. It has been used for reliable and repeatable ablation experiments. The framework provides a suite of generators for standard networks (*e.g.* randomized networks, algorithmic). The Scheduler spins off multiple networks with each instance based on a specific generation strategy allowing easy comparison of autoML generated networks with standard repeatable .

In addition, recent work in the field of neural architecture search (Real et al. (2018); Pham et al. (2018)) has emphasized the importance of constraint based neural architecture search which minimizes the model's resource consumption. AMLA has been used to construct networks that meet resource constraints. Network instrumentation in the AI/ML library was used to estimate resource consumption (by estimating the parameters in every layer) and it was fed back to an AutoML algorithm that used that information to limit parameters at each stage of a neural network.

6. Conclusions

This work describes AMLA, a framework for implementing and deploying AutoML neural network generation algorithms. Its key architectural design choices are the decoupling of generation and evaluation, an independent model definition and a microservices architecture that supports deployment at scale. In contrast to previous monolithic designs, AMLA uses modern large scale distributed system design techniques such as loosely coupled microservices with standard interfaces and model specification languages to provide a pluggable framework for AutoML algorithms.

Preliminary deployments with AutoML to generate CNNs for image classification have been promising (Kamath et al. (2018)). The current version supports model specification using JSON files, loose coupling through microservices and a REST API based CLI is available here:
https://github.com/ciscoai/amla.

References

Martín Abadi and others. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Andrew Brock, Theodore Lim, James M. Ritchie, and Nick Weston. SMASH: one-shot model architecture search through hypernetworks. *CoRR*, abs/1708.05344, 2017.

Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015. URL http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine-learning.pdf.

D. Floreano, P. Drr, and C. Mattiussi. Neuroevolution: from architectures to learning. Evolutionary Intelligence, 2008. URL https://doi.org/10.1007/s12065-007-0002-4.

Purushotham Kamath, Abhishek Singh, and Debo Dutta. Neural architecture construction using envelopenets. *CoRR*, abs/1803.06744, 2018. URL http://arxiv.org/abs/1803.06744.

Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18(25):1–5, 2017. URL http://jmlr.org/papers/v18/16-261.html.

Kubeflow. Kubeflow:https://github.com/kubeflow/kubeflow. 2018.

MLPerf. Mlperf:https://mlperf.org/. 2018.

MOE. Metric optimization engine:https://github.com/yelp/moe. 2015.

Randal S. Olson, Nathan Bartley, Ryan J. Urbanowicz, and Jason H. Moore. Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 485–492, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4206-3. doi: 10.1145/2908812.2908918. URL http://doi.acm.org/10.1145/2908812.2908918.

ONNX. Open neural network exchange format (onnx):https://onnx.ai/. 2018.

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.

Hieu Pham, Melody Y. Guan, Barret Zoph, Quoc V. Le, and Jeff Dean. Efficient neural architecture search via parameter sharing. *CoRR*, abs/1802.03268, 2018. URL http://arxiv.org/abs/1802.03268.

Esteban Real, Alok Aggarwal, Yanping Huang, and Quoc V Le. Regularized evolution for image classifier architecture search. *CoRR*, abs/1802.01548, 2018. URL http://arxiv.org/abs/1802.01548.

Spearmint. Spearmint:https://github.com/hips/spearmint. 2016.

Barret Zoph and Quoc V. Le. Neural architecture search with reinforcement learning. In *ICLR 2017*, 2017. URL https://arxiv.org/abs/1611.01578.