Demo: Accessible Deep Learning for the Scientific Community

Brian Broll
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA
brian.broll@vanderbilt.edu

***Abstract*—DeepForge is a gateway to deep learning for the scientific community. Utilizing a cloud-based infrastructure, DeepForge facilitates rapid development by promoting accessibility and reproducibility of experiments. In this demonstration, we present the core concepts of DeepForge and an image classification example.**

***Index Terms*—computer aided analysis, web services, artificial neural networks, open source software**

I. Introduction

Deep neural networks have shown to be very effective across a number of different domains including image classification [1] and speech recognition [2]. They have also been successfully applied to a number of scientific domains including chemistry, bioinformatics, and astronomy [3]–[8]. However, there are still significant challenges when using neural networks in scientific domains. DeepForge is a machine learning gateway developed to address these challenges and promote reproducibility and productivity. In this demonstration, we will be presenting the core concepts of DeepForge and an image classification example showcasing our work to make deep neural networks accessible to the scientific community. This includes designing neural network architectures, building machine learning pipelines, training machine learning models, model evaluation, and reproducibility.

II. Core Concepts

DeepForge presents four main concepts for testing and training machine learning models. These concepts are *Pipelines*, *operations*, *executions*, and *jobs*.

A *pipeline* is a directed acyclic graph, representing a given machine learning task, composed of atomic *operations*. Pipelines may have one or more inputs and outputs. These pipelines may perform many different tasks including data retrieval, data preprocessing, training a machine learning model, or model evaluation. Every project may contain multiple of these pipelines.

An *operation* is a function which accepts zero or more named inputs and outputs. Operations also can be parameterized by specifying *attributes* or *references* at design time. Attributes contain primitive values and references, as the name suggests, are a pointer to another existing component within the platform (such as a neural network architecture). Operations are implemented in Python and can be composed into pipelines.

Executing a pipeline, creates an *execution*. An execution, is a snapshot of the pipeline containing additional metadata about the current state of the given execution. Each operation in the originating pipeline is replaced with a *job* which contains metadata such as stdout and any plots generated from running the operation.

To support the creation and training of neural networks, DeepForge also contains concepts for *architectures* and *layers*. An architecture represents a neural network architecture and is composed of layers. A layer is a function with tensor inputs and outputs and optionally parameterized with attributes. When referenced from an operation, DeepForge will generate the corresponding code for an architecture and provide the resulting Python object to the operation's implementation for training.

III. Image Classification Example

Training a neural network effectively requires a couple main steps including defining the neural network architecture, retrieving the data, training the neural network with a set of hyperparameters, and evaluating the trained network. DeepForge provides a number of projects demonstrating this process on a number of different datasets. In this section, we will be exploring one such project which trains a neural network on the CIFAR10 dataset.

A. Defining the Architecture

DeepForge provides a simple visual interface for designing neural network architectures. This interface enforces the semantics of the domain and can provide useful feedback such as errors in the architecture and dimensionality feedback. Figure 1 shows a convolutional neural network being edited from within DeepForge.

B. Building Pipelines

Figure 2 shows an example pipeline in DeepForge. This pipeline contains four operations: **GetCifar10Data**, **Train**, **ScoreModel**, and **Output**. The **GetCifar10Data** operation is an operation which fetches and preprocesses the CIFAR10 dataset [9]. This includes partitioning the dataset into training and test sets.

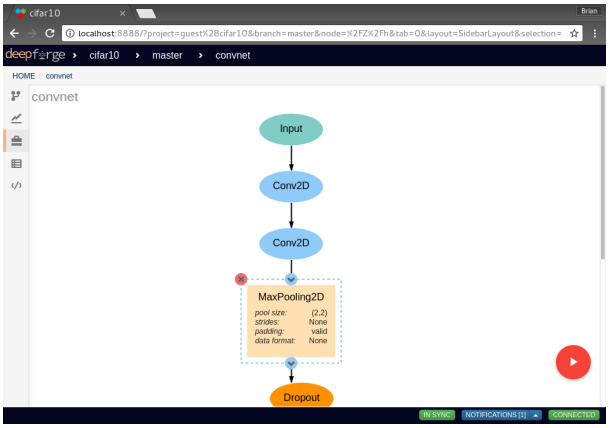

Fig. 1. Designing a Convolutional Neural Network

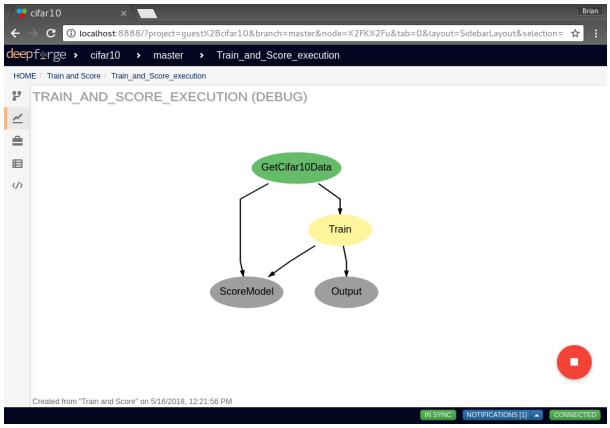

Fig. 3. Executing a Training Pipeline

After retrieving and preparing the data, the training data is passed as input to the **Train** operation. This operation has attributes for the training hyperparameters including learning rate and batch size as well as a reference to the neural network architecture to be used. Clicking on the operation will expand the operation showing its inputs, outputs, attributes, and references. An icon allowing the user to edit the operation interface and implementation is also shown when expanded.

The resulting trained model is passed to the **Output** and **ScoreModel** operations. **Output** records the produced artifact as an output and allows it to be downloaded or provided as input to other pipelines. **ScoreModel** accepts the trained model and testing data and evaluates the model on the given data.

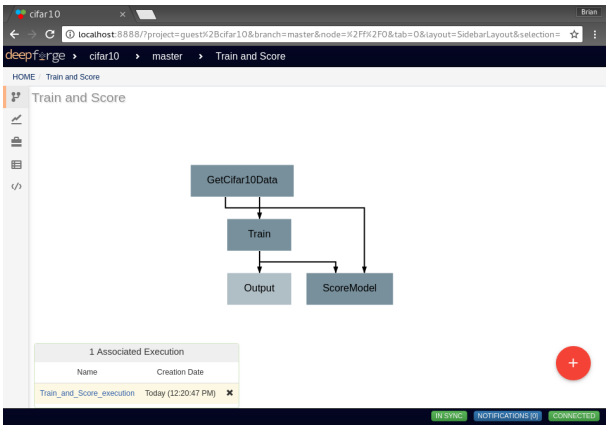

Fig. 2. Building Machine Learning Pipelines

C. Training the Model

After defining the architecture and building the training pipeline, the model can be trained by executing the given pipeline. Executing the pipeline will create an execution which contains additional metadata pertaining to the state of the training. An example of the execution is provided in Figure 3. In this example, we can see that the first job, **GetCifar10Data**, has completed and the platform is currently training the neural network architecture. The grey jobs are currently queued as they are awaiting the result of the **Train** job.

After the training has completed, the model will be evaluated by the **ScoreModel** job. A copy of the model is also provided as the output of the execution and will be stored as an artifact in DeepForge. This will allow the model to be provided as inputs to other pipelines or exported for use outside of the platform.

D. Reproducibility

Another important design consideration is the reproducibility of experiments. Version control is deeply integrated into the platform and both the project and the associated data is tracked. Every user action is automatically versioned ensuring that previous project states can be restored and pipelines can be executed with the exact code and input data. An example browsing the history of the project is shown in Figure 4.

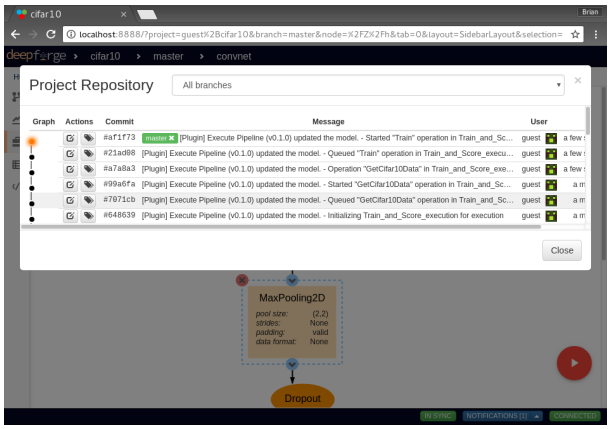

Fig. 4. Viewing Project History

IV. Conclusion

The presented image classification project provided a simple example demonstrating a few of the features of DeepForge that promote accessibility of deep learning and reproducibility of experiments. We are currently working to extend DeepForge to integrate with existing computational resources such as NERSC and DOE supercomputers as well as integrate with existing scientific data sources. Future work includes

developing visualization capabilities for model introspection and data visualization to provide more insight when training intelligent machine learning models. By designing accessible deep learning in a machine learning gateway, we hope to accelerate the impact of machine learning across scientific domains.

ACKNOWLEDGMENT

This material is supported by the National Science Foundation under Grant Number SI2-SSE-1740151. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[2] D. Amodei, R. Anubhai, E. Battenberg, C. Case, J. Casper, B. Catanzaro, J. Chen, M. Chrzanowski, A. Coates, G. Diamos, E. Elsen, J. Engel, L. Fan, C. Fougner, T. Han, A. Y. Hannun, B. Jun, P. LeGresley, L. Lin, S. Narang, A. Y. Ng, S. Ozair, R. Prenger, J. Raiman, S. Satheesh, D. Seetapun, S. Sengupta, Y. Wang, Z. Wang, C. Wang, B. Xiao, D. Yogatama, J. Zhan, and Z. Zhu, "Deep speech 2: End-to-end speech recognition in english and mandarin," *CoRR*, vol. abs/1512.02595, 2015. [Online]. Available: http://arxiv.org/abs/1512.02595

[3] O. Dor and Y. Zhou, "Real-spine: An integrated system of neural networks for real-value prediction of protein structural properties," *PROTEINS: Structure, Function, and Bioinformatics*, vol. 68, no. 1, pp. 76–81, 2007.

[4] E. Byvatov, U. Fechner, J. Sadowski, and G. Schneider, "Comparison of support vector machine and artificial neural network systems for drug/nondrug classification," *Journal of chemical information and computer sciences*, vol. 43, no. 6, pp. 1882–1889, 2003.

[5] J. Khan, J. S. Wei, M. Ringner, L. H. Saal, M. Ladanyi, F. Westermann, F. Berthold, M. Schwab, C. R. Antonescu, C. Peterson *et al.*, "Classification and diagnostic prediction of cancers using gene expression profiling and artificial neural networks," *Nature medicine*, vol. 7, no. 6, pp. 673–679, 2001.

[6] J. Lyons, A. Dehzangi, R. Heffernan, A. Sharma, K. Paliwal, A. Sattar, Y. Zhou, and Y. Yang, "Predicting backbone cα angles and dihedrals from protein sequences by stacked sparse auto-encoder deep neural network," *Journal of computational chemistry*, vol. 35, no. 28, pp. 2040–2046, 2014.

[7] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[8] M. Razzano and E. Cuoco, "Image-based deep learning for classification of noise transients in gravitational wave detectors," *Classical and Quantum Gravity*, vol. 35, no. 9, p. 095016, 2018.

[9] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," 2009.