DeepForge: A Scientific Gateway for Deep Learning

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

Miklós Maróti
Bolyai Institute
University of Szeged
Szeged, Hungary
mmaroti@math.u-szeged.hu

Péter Völgyesi
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA
peter.volgyesi@vanderbilt.edu

Ákos Lédeczi
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA
akos.ledeczi@vanderbilt.edu

***Abstract*—We introduce DeepForge, a gateway to deep learning for the scientific community. DeepForge is designed to lower the barrier to entry and facilitate the rapid development of deep learning models. Utilizing a cloud-based infrastructure, DeepForge facilitates rapid development by promoting reproducibility, collaboration and remote execution of machine learning pipelines. This represents an interdisciplinary approach to facilitating deep learning development as it leverages the strengths of Model Integrated Computing to provide a powerful hybrid textual-visual programming platform for the scientific community.**

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

I. Introduction

Deep learning has proven to be a very powerful machine learning approach in a variety of domains from image classification [1] to audio speech recognition [2]. A significant contributor to the success of deep neural networks is their ability to model very complex functions; this enables neural network architectures to be applied to various domains and the models to be confirmed empirically. Neural networks have also shown to be very effective when applied to scientific domains including bioinformatics, chemistry, and astronomy [3]–[8].

Despite this effectiveness, there are still significant barriers to deep learning for the scientific community. Programming is a necessary prerequisite for developing and evaluating deep learning models and mistakes are often only discovered at runtime. Due to the emphasis on empirical validation, it is also very important that researchers and practitioners are equipped with the appropriate tooling to allow them to iterate quickly, work together, and easily integrate the latest advancements in research into their own projects. To this end, we have developed DeepForge, a novel gateway to deep learning for the scientific community based on two main computing technologies, TensorFlow (via Keras) and Model Integrated Computing using WebGME.

TensorFlow is a high performance computing framework supporting machine learning including the development of deep learning models [9]. It supports a large number of different deployment platforms including CPU, GPU, and TPU and can run on a wide variety of devices from clusters to mobile devices. This broad support for deployment platforms makes TensorFlow an ideal backend for the training of deep learning models and promote the reuse of the existing computational resources across scientific domains and organizations.

Model Integrated Computing (MIC) is the technique of using models, or domain specific abstractions, for developing systems or applications [10] and was developed to aid in the rapid design and implementation of complex applications and systems. The Generic Modeling Environment (GME) is an open source MIC tool developed for creating domain specific modeling environments and has been effectively applied to a number of domains including embedded systems and mechatronics [11]–[17].

WebGME, the successor to GME, leverages a number of modern features such as a cloud-based infrastructure, integrated version control and real-time collaborative editing [18]. WebGME also introduces a number of powerful modeling abstractions, such as prototypal inheritance and mixins, to improve its ability to model complex systems [19]. WebGME has been used to improve development in a variety of domains from medical capsule robotics to space radiation [20]–[23].

DeepForge is a development environment with deeply integrated domain specific modeling aspects created with WebGME. This enables DeepForge to leverage the strengths of Model Integrated Computing to facilitate the rapid development of deep learning models and accessibility of deep learning while also improving experiment reproducibility. In doing so, we provide a powerful open source platform for deep learning, with aims to build a community around the platform in which users can contribute custom DeepForge extensions as well as seamlessly share and collaborate with one another.

The structure of the paper is as follows. In Section II, we will present the conceptual framework for creating and executing machine learning tasks. Section III presents the design of the DeepForge platform.

II. Core Concepts

DeepForge presents four main concepts for testing and training machine learning models: *Pipelines*, *Operations*, *Executions* and *Jobs*.

Operations are atomic functions which accept one or more named inputs and return one or more named outputs. Operation *attributes* can be defined for an operation at design time, providing adjustable parameters for the operation. At runtime, an operation's attributes are provided as constants to the operation. Operations can also define *references* which, like attributes, are specified at design time. Unlike attributes, a reference is a pointer to another artifact, such as a neural network architecture.

A *Pipeline* represents a machine learning task, such as model training, testing or data preprocessing composed of operations. Operations are composed by directing an output of a single operation, represented as a port, into an input of one or more other operations. That is, these operations are composed into acyclic data flow graphs. Pipelines can also contain *Input* and *Output* operations for representing input data to the pipeline and output data from the pipeline. That is, the *Input* operation is an initial node in a pipeline and *Output* operation is a terminal node.

Figure 1 shows an example of a pipeline with four operations. The first operation downloads and prepares training and testing sets from the mnist [24] dataset. The training data is then provided to the "Train" operation which trains a neural network on the given input. The "Train" operation is parameterized by the batch size, number of epochs, and the neural network architecture (a reference). A trained model is output from the "Train" operation and then can be passed to the "Output" and "ScoreModel" operations. The "Output" operation saves the model back to DeepForge and the "ScoreModel" operation evaluates the trained model on the testing data from the first operation.

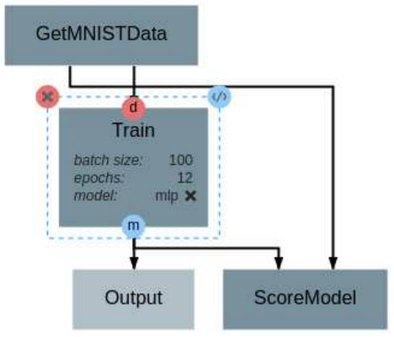


Fig. 1. Train and Evaluate Pipeline

Executing pipelines results in the creation of *Executions*. A pipeline's execution is an acyclic data flow graph that is isomorphic with the graph of the originating pipeline. This graph is created by converting each of the pipeline's operations into a *Job* that corresponds to the original operation combined with run status and running metadata (such as images or plots generated during the execution).

As DeepForge focuses on training (deep) neural networks, it also utilizes two additional concepts: *Architectures* and *Layers*. As expected, an architecture represents a neural network architecture while a layer represents a neural network layer. The layers are connected in a directed acyclic graph forming the architecture. Similar to operations, layers can have attributes which are set at design time[1]. Unlike operation attributes, layer attributes can also be set to another layer or sequence of layers.

III. Platform

Leveraging the concepts presented in Section II, DeepForge provides a web-based platform for deep learning with three principle design goals: accessibility, facilitating rapid development and reproducibility of experiments. DeepForge utilizes techniques in MIC along with intuitive, domain specific interfaces to improve the accessibility of deep learning. Rapid development is facilitated through promoting collaboration and supporting the entire development cycle from the first iteration to the last execution. Finally, DeepForge leverages WebGME's version control system to provide deeply integrated versioning of both the code and the data throughout the entire development process. In this section we will discuss the DeepForge platform in more detail with an emphasis on how the design supports these goals.

A. Accessibility

DeepForge uses WebGME, a framework for creating domain specific modeling environments, to create a domain specific modeling language for neural networks and the concepts described in Section II. This enables DeepForge to leverage the strengths of MIC, such as enforcing semantics of the domain and easily generating artifacts of different formats from the models, while also facilitating the development of other modern features including real-time collaborative editing and a deeply integrated version control system. DeepForge provides a hybrid textual-visual development environment for developing deep learning models. This allows users to leverage the strengths of a domain specific modeling environment when working at high levels of abstraction, such as designing pipelines, and utilize the precision of a textual programming environment when working at low levels of abstractions, such as implementing custom operations. This visual editor is not simply a diagram generated from the code but is an executable domain model and is always guaranteed to provide an up-to-date visual representation of the given pipeline or architecture.

Providing a visual interface for designing pipelines and architectures not only lowers the barrier to entry for developing deep learning models but also provides a more easily understandable diagram of the given structure. Lowering the barrier to entry reduces the amount of time it takes for a new user to be able to start developing and training neural network models. Designing pipelines and architectures with a visual interface can simplify complex machine learning tasks

[1]Layer attributes are analogous to constructor arguments in textual, object-oriented programming.

as the interface provides a diagram of the given pipeline or architecture. Textual interfaces are provided for some of the more advanced features of DeepForge such as implementing custom operations.

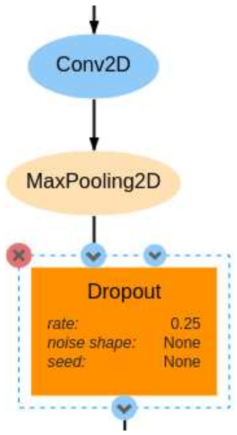


Fig. 2. Editing a Neural Network Architecture

Figure 2 shows the visual editor for creating neural network architectures. This example shows a segment of a convolutional neural network containing a two-dimensional convolution, max pooling layer, and a dropout layer. The dropout layer is currently selected and expanded to show the configurable attributes for the given layer. In this case, the dropout layer is configured to filter out 25% of the values from the previous max pooling layer (given by the value of "rate"). Values can be editing in place; inputs and outputs to the layer can be added using the light blue connection icons. This editor enforces many of the constraints of the domain including the directed acyclic graph structure of neural network architectures. In the future, this editor will also provide more precise feedback including dimensionality information and validation of layer attributes.

B. Rapid Development

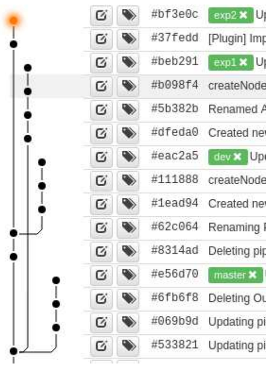
DeepForge includes a number of modern features to facilitate rapid development of deep learning models. This is done primarily through promoting collaboration, reproducibility and easy execution in a distributed environment.

DeepForge provides a number of features promoting collaboration. Google Docs-style real-time collaborative editing allows users to simultaneously work together on a project. Version control is deeply integrated into DeepForge and user actions are automatically committed. This allows users to not only see the recent changes made by collaborators but also enables them to leverage branches to manage collaboration in large projects.

Supporting the complete development cycle of creating machine learning pipelines was paramount in DeepForge; supporting both the creation and execution of the machine learning pipelines not only greatly simplifies the user experience but also allows the environment to support developer iteration, multi-tasking and record the results of the given task. DeepForge supports the execution of machine learning pipelines on a distributed environment. This includes not only the ability to execute pipelines in a distributed environment but also powerful utilities for monitoring and improving upon existing pipelines. DeepForge provides real-time feedback from running jobs, including creating plots, viewing images or simply viewing the job's console output. To promote rapid development, DeepForge also enables the user to restart individual jobs within an execution and caches intermediate job results in an execution; this allows the user to reuse the outputs of unchanged jobs and avoid redundant computation.

C. Reproducibility

The final design goal in DeepForge is to provide easy reproducibility of experiments. This is achieved through the use of automatic versioning during development and as well as versioning both the code and the binary artifacts (such as data and trained models) for the given experiments. Automatic version control ensures that all changes and edits will be recorded in the history of the project and are reproducible. Versioning both the code and all associated binary artifacts, DeepForge is able to ensure that not only the historical versions of the code are reproducible for a given experiment but also any associated data and models.



```
#bf3e0c  exp2 ✖ U
#37fedd  [Plugin] Imp
#beb291  exp1 ✖ U
#b098f4  createNode
#5b382b  Renamed A
#dfeda0  Created ne
#eac2a5  dev ✖ Up
#111888  createNode
#1ead94  Created ne
#62c064  Renaming
#8314ad  Deleting pi
#e56d70  master ✖
#6fb6f8  Deleting Ou
#069b9d  Updating pi
#533821  Updating pi
```

Fig. 3. Viewing commit history of a project in DeepForge

Leveraging the version control API's provided by the WebGME framework, DeepForge is able to provide an automatic version control system specific to the deep learning domain. This includes automatically tagging commits when experiments are run, creating commits automatically complete with meaningful commit messages and committing intermediate results from machine learning pipelines to promote reproducibility and rapid iteration.

IV. Conclusion

This paper presented DeepForge, a robust development platform for deep learning with a powerful underlying conceptual model. DeepForge utilizes the strengths of model integrated computing, including model analysis and manipulation, to develop a deep learning platform that both provides an intuitive visual interface enforcing domain semantics as well as providing portable models which can be exported to various deployment formats. Furthermore, DeepForge has been designed to facilitate collaboration, reproducibility and extensibility. This includes supporting real-time collaborative editing, integrated version control and the remote execution of machine learning pipelines.

Future work includes developing adapters for additional deployment platforms such as NERSC and DOE supercomputers, public cloud providers (eg, Amazon EC2 GPU instances), and private clusters. Integration of publicly available scientific datasets (OpenML, Kaggle, Open Science Data Cloud, etc) as well as creating a registry for hosting operation definitions, architectures, and trained models [25]–[27] is also planned. We also plan to extend the existing extension architecture to support custom data visualization capabilities. This will enable the development and incorporation of third party custom model and data visualization utilities and will facilitate the incorporation of the latest neural network introspection and visualization techniques [28], [29]. Developing an active community is particularly valuable as users can develop custom DeepForge extensions. Currently, extensions can be used to create custom export formats (simplifying deployment of machine learning pipelines); however, in the future, these will also include custom model introspection utilities as well as data visualization.

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] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin *et al.*, "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," *arXiv preprint arXiv:1603.04467*, 2016.

[10] J. Sprinkle, "Model-integrated computing," *IEEE potentials*, vol. 23, no. 1, pp. 28–30, 2004.

[11] K. Czarnecki, T. Bednasch, P. Unger, and U. Eisenecker, "Generative programming for embedded software: An industrial experience report," in *International Conference on Generative Programming and Component Engineering*. Springer, 2002, pp. 156–172.

[12] J. A. Stankovic, R. Zhu, R. Poornalingam, C. Lu, Z. Yu, M. Humphrey, and B. Ellis, "Vest: An aspect-based composition tool for real-time systems," in *Real-Time and Embedded Technology and Applications Symposium, 2003. Proceedings. The 9th IEEE*. IEEE, 2003, pp. 58–69.

[13] C. Özgen, "Transforming mission space models to executable simulation models," Ph.D. dissertation, Middle East Technical University, 2011.

[14] K. Thramboulidis, "Model-integrated mechatronics-toward a new paradigm in the development of manufacturing systems," *IEEE Transactions on Industrial Informatics*, vol. 1, no. 1, pp. 54–61, 2005.

[15] A. Childs, J. Greenwald, G. Jung, M. Hoosier, and J. Hatcliff, "Calm and cadena: Metamodeling for component-based product-line development," *Computer*, vol. 39, no. 2, pp. 42–50, 2006.

[16] J. Sprinkle, "Generative components for hybrid systems tools." *Journal of Object Technology*, vol. 4, no. 3, pp. 33–38, 2004.

[17] X. Ke and K. Sierszecki, "Generative programming for a component-based framework of distributed embedded systems," in *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling (DSM06)*, 2006, pp. 113–122.

[18] M. Maróti, T. Kecskés, R. Kereskényi, B. Broll, P. Völgyesi, L. Jurácz, and Á. Lédeczi, "Next generation (meta) modeling: Web-and cloud-based collaborative tool infrastructure." 2014.

[19] Z. Lattmann, T. Kecskés, P. Meijer, G. Karsai, P. Völgyesi, and Á. Lédeczi, "Abstractions for modeling complex systems," in *International Symposium on Leveraging Applications of Formal Methods*. Springer, 2016, pp. 68–79.

[20] P. S. Kumar, W. Emfinger, G. Karsai, D. Watkins, B. Gasser, and A. Anilkumar, "Rosmod: a toolsuite for modeling, generating, deploying, and managing distributed real-time component-based software using ros," *Electronics*, vol. 5, no. 3, p. 53, 2016.

[21] M. Beccani, H. Tunc, A. Taddese, E. Susilo, P. Völgyesi, A. Lédeczi, and P. Valdastri, "Systematic design of medical capsule robots," *IEEE Design & Test*, vol. 32, no. 5, pp. 98–108, 2015.

[22] R. A. Austin, "A radiation-reliability assurance case using goal structuring notation for a cubesat experiment," Ph.D. dissertation, Vanderbilt University, 2016.

[23] H. Neema, J. Sztipanovits, M. Burns, and E. Griffor, "C2wt-te: A model-based open platform for integrated simulations of transactive smart grids," in *Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), 2016 Workshop on*. IEEE, 2016, pp. 1–6.

[24] L. Deng, "The MNIST database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[25] J. N. Van Rijn, B. Bischl, L. Torgo, B. Gao, V. Umaashankar, S. Fischer, P. Winter, B. Wiswedel, M. R. Berthold, and J. Vanschoren, "Openml: A collaborative science platform," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer, 2013, pp. 645–649.

[26] Kaggle, "Kaggle Datasets," https://www.kaggle.com/datasets, accessed: 2017-02-1.

[27] R. L. Grossman, Y. Gu, J. Mambretti, M. Sabala, A. Szalay, and K. White, "An overview of the open science data cloud," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*. ACM, 2010, pp. 377–384.

[28] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[29] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.