End-to-end Training of Differentiable Pipelines Across Machine Learning Frameworks

Mitar Milutinovic
Computer Science Division
University of California, Berkeley
mitar@cs.berkeley.edu

Atılım Güneş Baydin
Department of Engineering Science
University of Oxford
gunes@robots.ox.ac.uk

Robert Zinkov
Department of Engineering Science
University of Oxford
zinkov@robots.ox.ac.uk

William Harvey
Department of Engineering Science
University of Oxford
willh@robots.ox.ac.uk

Dawn Song
Computer Science Division
University of California, Berkeley
dawnsong@cs.berkeley.edu

Frank Wood
Department of Engineering Science
University of Oxford
fwood@robots.ox.ac.uk

Wade Shen
Information Innovation Office (I2O)
DARPA
wade.shen@darpa.mil

Abstract

In this work we present a unified interface and methodology for performing end-to-end gradient-based refinement of pipelines of differentiable machine-learning primitives. This is distinguished from recent interoperability efforts such as the Open Neural Network Exchange (ONNX) format and other language-centric cross-compilation approaches in that the final pipeline does not need to be implemented nor trained in the same language nor cross-compiled into any single language; in other words, primitives may be written and pre-trained in PyTorch, TensorFlow, Caffe, scikit-learn or any of the other popular machine learning frameworks and fine-tuned end-to-end while being executed directly in their host frameworks. Provided primitives expose our proposed interface, it is possible to automatically compose all such primitives and refine them based on an end-to-end loss.

1 Motivation

The landscape of machine learning frameworks is very active and evolving. The availability of open-source machine learning frameworks has been one of the key driving forces behind the success of deep learning (Goodfellow et al., 2016) and its widespread adoption, by allowing modularization, code reuse, and sharing of state-of-the-art results. Mainstream frameworks such as TensorFlow (Abadi et al., 2016), Theano (Bastien et al., 2012), Caffe (Jia et al., 2014), and PyTorch (PyTorch core team, 2017) provide, as core components, a generalized linear-algebra system working on multi-dimensional arrays ("tensors") with high-performance hardware bindings such as BLAS, CUDA, and CuDNN, and a facility to compute derivatives using backpropagation (Rumelhart et al., 1988;

Griewank, 2012). Using these two components, one can construct large-scale gradient-based learning algorithms (Bottou, 2010) and higher-level machine-learning abstractions ranging in complexity from the basic building blocks of multi-layer Perceptrons, convolutional neural networks (CNNs) and recurrent neural network (RNNs) to neural programmer-interpreters (Reed and De Freitas, 2015) or the neural Turing machine (Graves et al., 2014).

Despite the availability of many high-quality mainstream machine learning frameworks, interoperability between these have been largely non-existent. For instance, conventionally a machine learning practitioner would have no expectation of being able to take a gradient across multiple frameworks. Efforts in this direction have been mostly restricted to transfer-learning settings and cases where a pre-trained model from one framework is reused in another, such as the captioning of images by an RNN-based language model connected to the output of a pre-trained CNN (Karpathy and Fei-Fei, 2015; Vinyals et al., 2015).

Our ongoing work within the Data-Driven Discovery of Models (D3M) program has the objective of automating the construction of task-specific machine learning pipelines and leveraging the rich ecosystem of model implementations and specialized workflows across different machine learning frameworks. To this end, an important task is to design an API that, among other things, allows the propagation of gradients across multiple frameworks and models. In the following, we present a universal framework for expressing end-to-end pipelines of differentiable, potentially probabilistic, machine learning primitives, and also provide a reference implementation. We demonstrate one such pipeline involving components implemented in Caffe, scikit-learn, and PyTorch.

2 End-to-End Optimization of Machine Learning Pipelines

We consider a "primitive" to be a unit of computation that produces outputs sampled from a conditional probability distribution, which may be dependent on an input and on trainable parameters. If the primitive is deterministic, the mentioned probability distribution will be expressed as a Dirac delta function. A machine learning pipeline consists of one or more primitives chained together to transform some input x, to some output y. The pipeline can be trained end-to-end using training pairs (x, y); or individual primitives (or subsections of the pipeline) can be pre-trained using data pertaining to a sub-part of the task. Note that primitives can encapsulate computations of varying levels of granularity, ranging from simple dimensionality-reduction transformations to full-scale image- or speech-recognition models.

A differentiable primitive is one which can compute (1) the gradient of its outputs with respect to its inputs as a minimum, and (2) the gradient of its output with respect to its internal parameters for allowing the gradient-based optimization of its parameters. A differentiable pipeline is one which can be differentiated end-to-end, requiring that there is at least one path from x to y through which every primitive can compute at least the gradient of its outputs with respect to its inputs, allowing backpropagation from y to x.

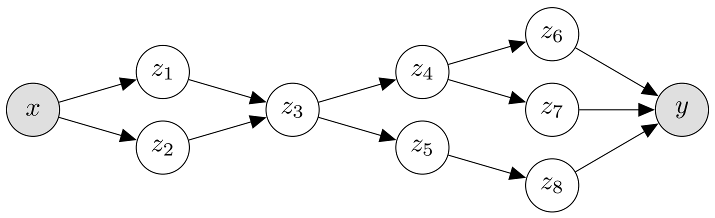

Figure 1: Pipeline. Each node is a (potentially) stochastic variable representing the output of a primitive.

We propose to learn the parameters, θ, of the primitives in a differentiable pipeline consisting of N primitives, by optimizing the probability of the training outputs and θ given the training inputs and the pipeline structure, s. This is given by the product of the likelihood of the end-to-end training outputs, y, given the training inputs, x, and θ, the likelihood of any local training outputs for each primitive, y_p, given the local training inputs, x_p, and θ, and the prior probability of θ:

$$p(y, \theta | x, s) = p(y|x, \theta, s) \prod_{p=1}^{P} p(y_p | x_p, \theta, s) \prod_{n=1}^{N} p(\theta_n | s) \tag{1}$$

This probability is also dependent on s, the structure of the pipeline. This provides a way to evaluate a given structure, allowing model selection to be performed, even with non-differentiable pipelines.

In order to train the parameters to maximize this probability in a differentiable pipeline, the likelihood of the end-to-end data is calculated and backpropagated through the pipeline. The contributions of local training data and parameter priors are added separately by each primitive during training.

When using non-deterministic primitives, estimating the likelihood of the end-to-end training data requires marginalizing out the stochastic intermediate outputs in the pipeline:

$$p(y|x, \theta, s) = \int \dots \int p(y, z_1, \dots, z_{N-1} | x, \theta, s) dz_1 \dots dz_{N-1} \tag{2}$$

In general, this integral is intractable, but can be approximated by using importance sampling (Doucet and Johansen, 2009) with L samples drawn from a proposal distribution, $q(z|y, x, \theta)$:

$$p(y|x, \theta, s) \approx \frac{1}{L} \sum_{l=1}^{L} \frac{p(y, z^l_{1:N-1} | x, \theta, s)}{q(z^l_{1:N-1} | y, x, \theta, s)} \quad z^l \sim q(z_{1:N-1} | y, x, \theta, s) \tag{3}$$

When the proposal distribution is taken as $q(z_{1:N-1}|y, x, \theta, s) = p(z_1|x, \theta, s)p(z_2|z_1, x, \theta, s) \dots p(z_{N-1}|z_{1:N-2}, x, \theta, s)$, it can be sampled from by simply running the pipeline. In this case, $p(y, z|x, \theta)$ is equal to $q(z|y, x, \theta)p(y|z, x, \theta)$. Therefore, the equation reduces to:

$$p(y|x, \theta, s) \approx \frac{1}{L} \sum_{l=1}^{L} p(y|z^l, x, \theta, s) \quad z^l \sim q(z|y, x, \theta, s) \tag{4}$$

This formulation allows the loss on end-to-end data to be calculated and backpropagated through the pipeline. We propose that primitives combine this with the contributions to the gradient from the parameter priors and local training data in their `backprop` method.

We propose an API for the primitives which will provide the minimum functionality to allow this form of optimization. This contains:

- `produce` or `sample` — These run the primitive forward, taking in its inputs and returning a its outputs.
- `log_likelihood` — This takes in pairs of inputs and outputs and returns the log likelihood of the outputs given the inputs. It can be substituted for another loss in deterministic primitives.
- `backprop` — If this is used in the final primitive in the pipeline, this uses the loss from `log_likelihood` as the loss. Otherwise, it requires the gradient of the loss with respect to its outputs to be passed as an argument. It then updates the gradient of the loss with respect to its parameters accordingly and returns the gradient of the loss with respect to its inputs.
- `set_training_data` — This provides local pre-training data to a single primitive.
- `fit` — This fits (trains) a primitive using local training data and any parameter priors.

3 Example Pipeline

Using a reference implementation of the described API, we demonstrate this training approach on a regression task with a dataset consisting of hand images, for which the task is to predict the wrist breadth (Miguel-Hurtado et al., 2016). There are 112 images in this corpus.

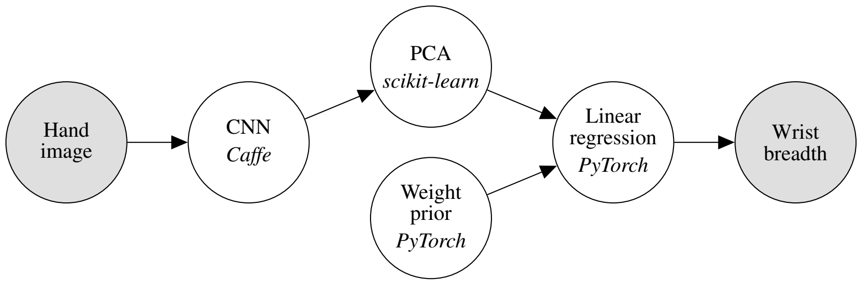

Figure 2: Example pipeline for predicting a physiological feature (wrist breadth) from images of hands using four primitives in three different frameworks.

Our pipeline (Figure 2) consists of a feature extractor using DeepHands, a convolutional neural network (CNN) pre-trained for classifying hand-poses (Koller et al., 2016). The output features in turn are passed through a dimensionality reduction primitive using probabilistic principal component analysis (PCA), and finally through a linear regression primitive regularized by a Gaussian weight prior. Both the linear regression and the probabilistic PCA sample their output from an inferred distribution, while the CNN is deterministic.

Every primitive of this pipeline is written in a different machine learning framework. The CNN loads a pre-trained model published in Model Zoo and uses Caffe. The PCA primitive wraps the PCA implementation in scikit-learn. The linear regression primitive is written in PyTorch. By each component providing a `backprop` method, we are able to jointly train the classifier, the dimensionality reduction, and the neural network, also fine-tuning the weights of the pre-trained CNN in the process.

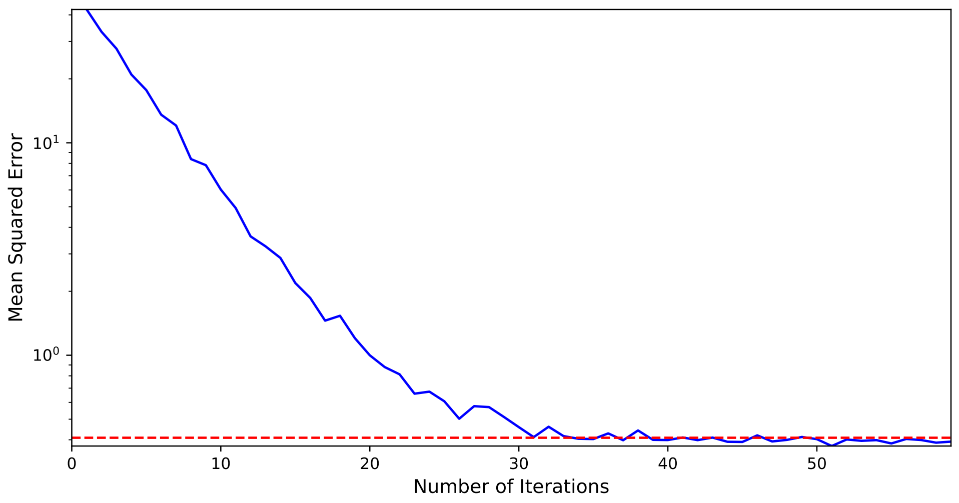

Figure 3: Mean-squared error of pipeline (solid blue) versus baseline (dashed red) throughout end-to-end refinement.

Code for the reference API implementation, the primitives, and the pipeline is available at:

`https://github.com/probprog/d3m-end-to-end-pipeline`

Acknowledgments

This work has been supported in part by the Defense Advanced Research Projects Agency (DARPA) Data-Driven Discovery of Models (D3M) Program.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016.

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. Deep Learning and Unsupervised Feature Learning NIPS 2012 Workshop, 2012.

Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

Arnaud Doucet and Adam M Johansen. A tutorial on particle filtering and smoothing: Fifteen years later. In *Handbook of Nonlinear Filtering*. 2009.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

Andreas Griewank. Who invented the reverse mode of differentiation? *Documenta Mathematica*, Extra Volume ISMP:389–400, 2012.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Andrej Karpathy and Li Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

Oscar Koller, Hermann Ney, and Richard Bowden. Deep hand: How to train a cnn on 1 million hand images when your data is continuous and weakly labelled. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3793–3802, Las Vegas, NV, USA, June 2016.

Oscar Miguel-Hurtado, Righard Guest, Sarah V. Stevenage, Greg J. Neil, and Sue Black. Comparing Machine Learning Classifiers and Linear/Logistic Regression to Explore the Relationship between Hand Dimensions and Demographic Characteristics, October 2016. URL `https://doi.org/10.5281/zenodo.17487`.

PyTorch core team. PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration, 2017. URL `http://pytorch.org/`.

Scott Reed and Nando De Freitas. Neural programmer-interpreters. *arXiv preprint arXiv:1511.06279*, 2015.

David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.