TensorFlow as a Feature Engineering DSL

Pablo Duboue
Textualization Software Ltd.
Vancouver, BC, CANADA

Abstract

Deep Neural Networks have fulfilled the promise to reduce the need of feature engineering by leveraging deep networks and training data. An interesting side effect, maybe unexpected, is that frameworks for implementing Deep Learning networks can also be used as frameworks for feature engineering. We sketch here some ideas for a DSL implemented on top of TensorFlow, hoping to attract interest from both the deep learning computational graph and programming languages communities.

1 Introduction

Feature engineering (FE), defined as [1]

> the process of representing a problem domain to make it amenable for learning techniques. This process involves the initial discovery of features and their stepwise improvement based on domain knowledge and the observed performance of a given ML algorithm over specific training data.

Quality FE is crucial for the success of non-DL ML projects, as highlighted by Andrew Ng: [2]

> Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically Feature Engineering.

In this work, I follow other authors [3] in considering FE as an encompassing term that include feature generation, transformation, and selection. For many people FE is synonym of feature selection. You are welcomed to disagree with my use of the term, just keep in mind I use it to mean the three things in this document.

Deep Neural networks (DL) fulfilled promise has been to reduce the need of FE by leveraging deep networks and training data [4]. An interesting side effect, maybe unexpected, is that frameworks for implementing DL networks can also be used as frameworks for FE, possibly with some expansions.

This realization came about when writing my textbook "The Art of Feature Engineering" [1], where I discuss how gradient descent, weight-tying and loosely differentiable updates define a new computing paradigm. As brought to the forefront by the Automatic Differentiation workshop [5], any graph of units such that the path from input to output is loosely differentiable can thus be trained. This type of graphs is sometimes made explicit in some NN frameworks, like TensorFlow (TF) [6]. Programming in this new paradigm takes the form of master equations or flow diagrams. While this enables a new age of computing, in the immediate case of FE, frameworks like TF present themselves as a great opportunity to use them as a Domain Specific Language (DSL) for FE (independent of DL).

Based on my experience designing an earlier DSL for FE, heavily used at the IBM Jeopardy! Challenge [7], I sketch some constructs for the DSL in Sec. 3, expressing FE as a DSL implemented on top of TF primitives. The primities are also informed by the seven case studies discussed in my upcoming textbook. I will now discuss some of the advantages of having a DSL for FE on top of

TF, as it is advantageous for non-DL ML, programming languages and translators (PLT) and DL communities.

1.1 Advantages for ML in general

Using the same framework for both ML and featurization neatly ties the trained model with the different steps needed to execute it from raw data. This ensures a trained model will always be correctly fed the features used at training time. It also simplifies testing and, potentially, cross-validation. Moreover, it has the advantage of symplifying trainable FE components ("data-driven feature engineer" [8]) and to standardize the delivery and application of FE transformations. Implementing the feature transformations on top a DL framework fixes an issue with most ML (and AutoML) frameworks which focus only in handling "tidy" feature sets [9]. This restriction misses out the real-world task of transforming raw data into features [10].

1.2 Advantages for DL

Besides the advantages for ML in general, expanding the operators in TF with operators derived from FE operations might open the door for gradient-based learning of their parameters. The operations missing in TF might very well be missing due to the lack of continuous derivatives. The need for a particular operation has been fruitful in the past as a means to devise derivable alternatives, for example, SoftMax as a derivable alternative to the max function [11].

1.3 Advantages for PLT

The programming languages and translators community can also contribute to expanding a language designed to capture data modification but also execution of data modifications over different settings. This is a part where existing DSLs, discussed next, seem to be lacking as they either focus on the data manipulations or in the training operations.

2 Related Work

DSLs as ways to encapsulate problems and solve them more efficiently have been studied for quite some time. In general, they can be embedding within a host language or kept external and compile to a lower-level language [12].

Other DSLs for FE include Salesforce's TransmogrifAI [13] and Spotify's Featran [14], which focus on the feature manipulations. Both are embedded DSLs using Scala as the underlining language, a popular compiled language for embedding DSLs [15]. A similar approach is Uber's Michelangelo platform [16], which contains a feature store for re-using across the whole company, plus the ability to train models from the within the framework.. Another industrial framework is the Pinterest Linchpin DSL [17], an external DSL focusing in feature transformations, currently being evolved into a general framework similar to TF.

Other DSLs include [18] Pig Latin [19], a DSL for data manipulation on top of the Hadoop Map Reduce framework, and GraphLab [20], a DSL for graphical models.

Finally, there is some momentum within the academic community to express featurization as ETL (extract, transform and load) operations from data warehousing [21] or as SQL stored procedures [22] or UDFs (user defined functions) [23]. These are data-centric operations that do not take into account the possibility of some of the operators to be trained from data.

3 The DSL

The proposed DSL uses the TF computation graph to store the data operations plus non-DL estimators to set their parameters using data. The parameters for the featurizer are set first, ideally using held-out data. Once set, they are frozen and they are not trained by the DL estimators.

FE can be seen as a series of operations on the raw data to obtain features. These operations can be divided into three groups:

Dataset

F	T
A	+
B	**-**
C	+
A	+
B	**+**
C	+
A	+

F	T
A	+
B	**-**
C	-
A	+
B	**-**
C	-
B	**+**

$\rightarrow$

F	TRE
A	1.0
B	**0.4**
C	*0.5*

$\rightarrow$

TRE dataset

F_{TRE}	T
1.0	+
0.4	**-**
0.5	+
1.0	+
0.4	**+**
0.5	+
1.0	+

F_{TRE}	T
1.0	+
0.4	**-**
0.5	-
1.0	+
0.4	**-**
0.5	-
0.4	**+**

Figure 1: Example of Target Rate Encoding. The categorical feature F has three categories, {A, B, C}. The category 'A' always appears with the target class '+' so its target rate encoding is 1.0 (a 100%).

- (Expansion) Operations that expand the feature set, for example, by computing new features from existing ones.
- (Reduction) Operations that contract the feature set, for example, by removing uninformative features.
- (Normalization) Operations that "normalize" the feature set, for example, by replacing feature values with the number of standard deviations from the feature mean.

Most of these operations can be expressed in TF, although some might need to be expressed as custom regularizers or non-gradient operations. Some of the FE tasks showcased in the case studies in my book that could be accomplished using this approach include:

Expansion centering, histograms (including bag-of-words), delta features, different types of dampening (including logarithmic), computable features (for this the TF computational graph will excel), change of coordinates, one-hot encoding, target-rate encoding, imputing missing features (including missing feature indicator), thresholding features, scaling, binning, expanding datatime strings, dealing with n-grams (including skip-grams);

Normalization decorrelation, coalescing categories;

Reduction feature selection (for example, using mutual information), blacklisting features, hashing features, representing sets, and others.

As the case studies are released as open source code over open data, it provides data and code that can be used as unit tests for building a DSL on top of TF.

The intention is to have less code written in Python and more in the form of higher-level constructs that get rendered in TF computational graph. The benefit is similar to using Keras instead of TF Core. The proposed DSL is at an equal level of abstraction with Keras (or could be implemented within Keras, if there is interest by the Keras developers). Special FE operators could be implemented using Swift, following the TF push for Swift as the next generation language [24].

3.1 Case Study: Target Rate Encoding

As a case study, let us discuss how to express a Target Rate Encoding feature transformation. We can think that a binary feature is in some respect a very "blunt" feature. It is either at 100% or at 0%. To make them more informative, it is possible to compute their rate of agreement with the target class [25]. See Fig. 1 for an example. This has to be done on held-out data different from the training data (what is usually described as "Out-of-Fold" when doing cross-validation). The interesting aspect of this approach is how it will conflate categories that may be indistinguishable when it comes to predicting the target class. This engineered feature might have a much stronger signal than the original feature, particularly if the number of categories was large.

The envisioned DSL is sketched in Listing 1. In this example, the single categorical feature has three possible values, therefore, after line 7, features.output has a dimension of three. The number of categories in this example is inferred from the training data. Alternatively, the OneHot constructor in line 5 can be called with the number of categories. If line 5 were to be replaced by TargetRateEncoded instead of OneHot, then the output vector size will be of dimension one. More raw data can be added and more feature transformations can be encoded, making for a larger feature vector.

```
 1  held_out_X = [ ('A',),('B',),('C',),('A',),('B',),('C',),
 2    ('A',),('A',),('B',),('C',),('A',),('B',),('C',),('B',) ]
 3  held_out_y = [ 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1 ]
 4  raw = Input(shape=(1,))
 5  onehot = OneHot()(raw)
 6  features = Vector(inputs=[raw], outputs=[onehot])
 7  features.fit(held_out_X, held_out_y)
 8
 9  dense1 = Dense(10, activation='relu')(features.output)
10  output = Dense(1, activation='sigmoid')(dense)
11  model = Model(inputs=features.input, outputs=output)
12  model.compile(optimizer='adam', metrics=['accuracy'])
13  model.fit( raw_data, y )
```

Listing 1: DSL example for one-hot encoding. Changing line 5 with TargetRateEncoded will render it an example of TRE.

In a similar manner, feature selection, normalization, bag-of-words, etc. can be expressed. Note that Keras has an "utilities" package with Python functions for some of these operations. But it is done in pure Python, outside of the TF Graph. Here, the TF graph is expanded with new operators and the featurization operations are left as part of the model that can then be serialized, shared, reused ensuring the correct featurization operations are performed.

4 Conclusions

Building a DSL for feature engineering on top of an existing gradient-based machine learning software framework like TensorFlow or Torch can prove useful to users, framework developers and programming languages researchers.

Solution involving feature engineering have matured with a number of textbooks available [8, 3, 1]. Some include open source code that could be used to test new feature engineering DSLs built on top of gradient-based machine learning software frameworks.

References

[1] Pablo Duboue. *The Art of Feature Engineering: Essentials for Machine Learning*. Cambridge University Press, 2020. ISBN 1108709389.

[2] Weizhong Yan. Feature Engineering for PHM applications. `https://www.phmsociety.org/sites/phmsociety.org/files/FeatureEngineeringTutorial_2015PHM_V2.pdf`, oct 2015. Accessed: 2018-05-01, linked from `https://www.phmsociety.org/events/conference/phm/15/tutorials`.

[3] Huan Liu Guozhu Dong, editor. *Feature Engineering for Machine Learning and Data Analytics*. Series: Chapman/Hall/CRC Data Mining and Knowledge Discovery Series. CRC Press, 1st edition edition, April 2018. ISBN 9781138744387.

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

[5] Autodiff workshop: Nips workshop on the future of gradient-based machine learning software. `https://autodiff-workshop.github.io/2016.html`, 2016.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete

Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `https://www.tensorflow.org/`. Software available from tensorflow.org.

[7] D. C. Gondek, A. Lally, A. Kalyanpur, J. W. Murdock, P. A. Duboue, L. Zhang, Y. Pan, Z. M. Qiu, and C. Welty. A framework for merging and ranking of answers in DeepQA. *IBM Journal of Research and Development*, 56(3.4):14:1 – 14:12, 2012. Digital Object Identifier: 10.1147/JRD.2012.2188760.

[8] Alice Zheng and Amanda Casari. *Feature engineering for machine learning: principles and techniques for data scientists*. O'Reilly Media, Inc., 2018.

[9] Hadley Wickham. Tidy data. *Journal of Statistical Software*, 59(1):1–23, 2014.

[10] Richard Lippmann, Swaroop Vattam, Pooya Khorrami, and Cagri Dagli. A new data corpus to promote more complete autonomous machine learning pipelines. Presentation at CiML 2018. `https://docs.google.com/a/chalearn.org/viewer?a=v&pid=sites&srcid=Y2hhbGVhcm4ub3JnfHdvcmtzaG9wfGd4OjNiODFkODQ2ODZjMGQyZjY`, 2018.

[11] John S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In D. S. Touretzky, editor, *Advances in Neural Information Processing Systems 2*, pages 211–217. Morgan-Kaufmann, 1990.

[12] Martin Fowler. *Domain-specific languages*. Pearson Education, 2010.

[13] Matthew Tovbin. Meet TransmogrifAI, open source AutoML that powers einstein predictions. SF Big Analytics Meetup. `https://www.youtube.com/watch?v=93vsqjfGPCw&feature=youtu.be&t=2800`, 2019. Accessed: 2019-7-13.

[14] Neville Li. Featran: Type safe and generic feature transformation in scala. `https://www.lyh.me/slides/featran.html`, November 2017.

[15] Debasish Ghosh. *DSLs in action*. Manning Publications Co., 2010.

[16] Jeremy Hermann and Mike Del Balso. Meet michelangelo: Uber's machine learning platform. `https://eng.uber.com/michelangelo/`, September 2017.

[17] Angela Sheu. The little engine that could: Linchpin dsl for pinterest ranking. `https://medium.com/pinterest-engineering/the-little-engine-that-could-linchpin-dsl-for-pinterest-ranking-17699add8e56`, August 2017.

[18] Ivens Portugal, Paulo Alencar, and Donald Cowan. A preliminary survey on domain-specific languages for machine learning in big data. In *2016 IEEE International Conference on Software Science, Technology and Engineering (SWSTE)*, pages 108–110. IEEE, 2016.

[19] Christopher Olston, Benjamin Reed, Utkarsh Srivastava, Ravi Kumar, and Andrew Tomkins. Pig latin. *Proceedings of the 2008 ACM SIGMOD international conference on Management of data - SIGMOD '08*, 2008. doi: 10.1145/1376616.1376726.

[20] Yucheng Low, Joseph E. Gonzalez, Aapo Kyrola, Danny Bickson, Carlos E. Guestrin, and Joseph Hellerstein. Graphlab: A new framework for parallel machine learning, 2014.

[21] Giri Iyengar. CS 5304: Data Science in the Wild. Cornell EECS, class Feature Engineering. `https://courses.cit.cornell.edu/cs5304/Lectures/lec5_FeatureEngineering.pdf`, March 2016. Accessed: 2016-10-01.

[22] Christopher Ré, Amir Abbas Sadeghian, Zifei Shan, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. Feature engineering for knowledge base construction. *arXiv preprint arXiv:1407.6439*, 2014.

[23] Michael R. Anderson and Michael J. Cafarella. Input selection for fast feature engineering. In *IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 577–588, 2016.

[24] Chris Lattner and Brennan Saeta. Swift for tensorflow: The next-generation machine learning framework. Presentation at TF Dev Summit '19, `https://youtu.be/s65BigoMV_I`, March 2019.

[25] Misha Bilenko. Big learning made easy – with counts! Microsoft Machine Learning Blog. `https://blogs.technet.microsoft.com/machinelearning/2015/02/17/big-learning-made-easy-with-counts/`, 2015. Accessed: 2019-07-11.