Portable, layer-wise task performance monitoring for NLP models

Thomas Lippincott
Johns Hopkins University
`tom@cs.jhu.edu`

1 Introduction

There is a long-standing interest in understanding the internal behavior of neural networks (Touretzky and Pomerleau, 1989; Zhou et al., 2017; Raghu et al., 2017; Alishahi et al., 2017). Deep neural architectures for natural language processing (NLP) are often accompanied by explanations for their effectiveness, from general observations (e.g. RNNs can represent unbounded dependencies in a sequence) to specific arguments about linguistic phenomena (early layers encode lexical information, deeper layers syntactic). The recent ascendancy of DNNs is fueling efforts in the NLP community to explore these claims (Belinkov et al., 2017; Dalvi et al., 2017; Karpathy et al., 2015; Kadar et al., 2016; Kohn, 2015; Qian et al., 2016a). Previous work has tended to focus on easily-accessible representations like word or sentence embeddings (Kohn, 2015; Qian et al., 2016b; Adi et al., 2016), with deeper structure requiring more ad hoc methods to extract and examine (Belinkov and Glass, 2017; Poliak et al., 2018). In this work, we introduce **Vivisect**, a toolkit that aims at a general solution for broad and fine-grained monitoring in the major DNN frameworks, with minimal change to research patterns. Vivisect is general enough to serve as a less-polished version of the widely-used TensorBoard tool, but has several priorities that set it apart:

- Minimal invasiveness (e.g. no SummaryOps)
- Low resource use (only keep final metrics)
- Uniform support for major DNN frameworks
- Monitor performance on auxiliary tasks

The first three points are largely ergonomic, though we hope that feature parity between the major DNN research frameworks will yield answers to previously-daunting questions, such as why seemingly-identical implementations of a deep architecture perform differently. The fourth point is the most important: when made aware of task labels from various linguistic modalities, Vivisect will train lightweight linear classifiers and clusterers using each of the model's internal representations as features. A lightweight web server aggregates and plots these scores as a function of other variables (e.g. training epoch) to give insight into what linguistic information is captured by different parts of the model, and how they evolve over time.

Vivisect has evolved out of a focus on neural machine translation models, but is designed with generalization as a fundamental principle. Therefore, it includes mechanisms for deciding what and when calculations are made, and on which parts of a model. There are simple APIs for registering additional metrics and entire DNN frameworks. Vivisect is provided as a code repository and optional prebuilt Docker image.

2 Client usage

To use Vivisect with a PyTorch Module, Tensorflow Session, or MXNet Block, one can minimally add three lines to existing code:

```
from vivisect import probe, flush
probe(model, vivisect_host, vivisect_port)
flush(vivisect_host, vivisect_port)
```

This will monitor all operations in the computation graph, using a Vivisect server at the given host and port. This is accomplished by traversing the computation graph, and at each operation, overriding the forward method (similarly for operation backward methods and parameter update methods). **probe** accepts two optional arguments: a callback **which(model, operation)** that determines whether to attach to each operation, and

when(model, operation) that determines whether the operation should be monitored at the current state.

3 Server architecture

Vivisect has a client and server arrangement so that the computational aspects of monitoring can be off-loaded to other servers, and without writing data to disk. Figure 1 shows the data flow, where all dashed edges are transmitting JSON objects with fields *values* and *metadata*.

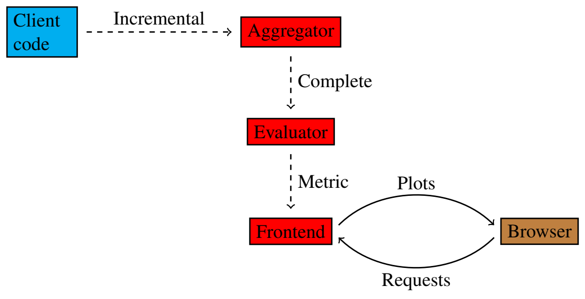

Figure 1: System diagram of Vivisect

(1) The client code, where the model is being trained or applied, sends *incremental* data points to the aggregator whenever a monitored parameter, activation, or gradient is used. These typically don't represent a full input for computing a metric (e.g. we want to monitor full epochs, but are getting activations for each mini-batch). (2) The aggregator keeps track of the metadata fields until it satisfies some condition (e.g. the *epoch* metadata field increases for the given model), constructs a *complete* data point by combining the appropriate arrays and metadata, and sends this to the *evaluator*. (3) The evaluator uses each complete data point to calculate arbitrary scalar-valued *metrics*, which it ships (again, with appropriate metadata) to the *frontend*. (4) The frontend is a simple web server on top of a sqlite database, into which it inserts each metric value, along with corresponding metadata like the model name and epoch. It uses this database to dynamically serve visualizations of the metrics along various axes, with the canonical use-case of how different activations perform as features for a classification task, as a function of epoch.

4 Beyond intrinsic measurements

Tracking and visualizing intrinsic properties of a model's internal state is useful, but well-covered by existing tools like Tensorboard and its variants. Vivisect's goal is to employ user-specified information about the model's input and output (in the latter case, during training or dev/eval) to test intuitions about how linguistic information is organized internally. The user can register such information with the server:

```
from vivisect import register_targets
register_targets(vivisect_host, vivisect_port,
                 name="Training classes",
                 targets=y_train,
                 model_pattern="Gluon MLP")
```

In this case, *y_train* are just the N-length sequence of classes that the model is being trained to identify, but since it is now registered, whenever the evaluator sees a CDP of appropriate dimension from a matching model, it trains a linear classifier and a k-means clustering using the CDP and calculates macro f-score and mutual information, respectively. These values are passed along in the same fashion as the intrinsic metrics, producing figures that compare how well the hidden layers are encoding this information:

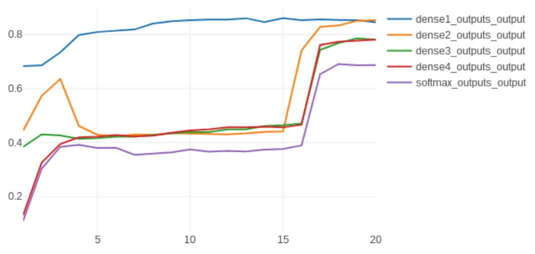

Figure 2: An example figure from the Vivisect frontend showing mutual information between clustering based on the given layer and reference labels

Figure 2 tells a simple story: for this small data set, information about the class is already captured on the surface in the shallow layers, and the model learns to preserve it as training progresses.

5 Ongoing work

Our immediate goal prior to the workshop is to employ Vivisect in training a large machine translation model with targets from several linguistic modalities not explicit in the model, at a minimum, part-of-speech and NER tagging at the word level, and topic ID at the sentence level, and present visualization and analysis.

References

Yossi Adi, Einat Kermany, Yonatan Belinkov, Ofer Lavi, and Yoav Goldberg. 2016. Fine-grained analysis of sentence embeddings using auxiliary prediction tasks. *CoRR*, abs/1608.04207.

Afra Alishahi, Marie Barking, and Grzegorz Chrupaa. 2017. Encoding of phonology in a recurrent neural model of grounded speech. *In Proceedings of the SIGNLL Conference on Computational Natural Language Learning*.

Yonatan Belinkov, Nadir Durrani, Fahim Dalvi, Hassan Sajjad, and James R. Glass. 2017. What do neural machine translation models learn about morphology? *CoRR*, abs/1704.03471.

Yonatan Belinkov and James R. Glass. 2017. Analyzing hidden representations in end-to-end automatic speech recognition systems. *CoRR*, abs/1709.04482.

Fahim Dalvi, Nadir Durrani, Hassan Sajjad, Yonatan Belinkov, and Stephan Vogel. 2017. Understanding and improving morphological learning in the neural machine translation decoder. *In Proceedings of the 8th International Joint Conference on Natural Language Processing*.

Akos Kadar, Grzegorz Chrupaa, and Afra Alishahi. 2016. Representation of linguistic form and function in recurrent neural networks. *arXiv preprint*.

Andrej Karpathy, Justin Johnson, and Fei-Fei Li. 2015. Visualizing and understanding recurrent networks. *arXiv preprint*.

Arne Kohn. 2015. Whats in an embedding? analyzing word embeddings through multilingual evaluation. *In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*.

Adam Poliak, Yonatan Belinkov, James R. Glass, and Benjamin Van Durme. 2018. On the evaluation of semantic phenomena in neural machine translation using natural language inference. *CoRR*, abs/1804.09779.

Peng Qian, Xipeng Qiu, , and Xuanjing Huang. 2016a. Analyzing linguistic knowledge in sequential model of sentence. *In Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*.

Peng Qian, Xipeng Qiu, and Xuanjing Huang. 2016b. Investigating language universal and specific properties in word embeddings. *In Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*.

M. Raghu, J. Gilmer, J. Yosinski, and J. Sohl-Dickstein. 2017. SVCCA: Singular Vector Canonical Correlation Analysis for Deep Learning Dynamics and Interpretability. *ArXiv e-prints*.

TensorBoard. 2017. https://github.com/tensorflow/tensorboard.

David S. Touretzky and Dean A. Pomerleau. 1989. What's hidden in the hidden layers? *BYTE*, 14(8):227–233.

Bolei Zhou, David Bau, Aude Oliva, and Antonio Torralba. 2017. Interpreting deep visual representations via network dissection. *CoRR*, abs/1711.05611.