The Journal of Open Source Software

Delve: Neural Network Feature Variance Analysis

Justin Shenk[*][1,2], **Mats L. Richter**[†][2], **and Wolf Byttner**[3]

1 VisioLab, Berlin, Germany **2** Institute of Cognitive Science, University of Osnabrueck, Osnabrueck, Germany **3** Rapid Health, London, England, United Kingdom

DOI: 10.21105/joss.03992

Software

- Review
- Repository
- Archive

Editor: Rachel Kurchin

Reviewers:

- @sambitdash
- @dionhaefner

Submitted: 08 October 2021
Published: 29 January 2022

Summary

Designing neural networks is a complex task. Deep neural networks are often referred to as "black box" models - little insight in the function they approximate is gained from looking at the structure of layer outputs. `Delve` is a tool for looking at how a neural network represents data, and how these representations, or features, change throughout training. This tool enables deep learning researchers to understand the limitations and suggest improvements for the design of their networks, such as removing or adding layers.

Several tools exist which allow analyzing neural networks after and during training. These techniques can be characterized by their focus on either data or model as well as their level of abstractness. Examples for abstract model-oriented techniques are tools for analyzing the sharpness of local optima (Keskar et al., 2016; Novak et al., 2018), which can be an indicator for the generalizing capabilities of the trained models. In these scenarios the complexity of the dataset and model is reduced to the error surface, allowing for insights into the differences between different setups. A less abstract data-centric technique GradCam by Selvaraju et al. (Chattopadhay et al., 2018; Selvaraju et al., 2019), reduces the model to a set of class-activation maps that can be overlayed over individual data points to get an intuitive understanding of the inference process. SVCCA (Morcos et al., 2018; Raghu et al., 2017) can be considered model-centric and a middle ground in terms of abstractness, since it allows the comparative analysis of the features extracted by specific layers. SVCCA is also relevant from a functional perspective for this work, since it uses singular value decomposition as a core technique to obtain the analysis results. Another model-centric tool that allows for a layer-by-layer analysis is logistic regression probes (Alain & Bengio, 2016), which utilize logistic regressions trained on the output of a hidden layer to measure the linear separability of the data and thus the quality of the intermediate solution quality of a classifier model.

The latter is of great importance for this work since logistic regression probes are often used to compare models and identify the contribution of layers to overall performance (Richter, Shenk, et al., 2021; Richter, Byttner, et al., 2021; Richter, Schöning, et al., 2021) and to demonstrate that the saturation metric is capable of showing parameter-inefficiencies in neural network architectures.

However, the aforementioned tools have significant limitations in terms of their usefulness in practical application scenarios, where these tools are to be used to improve the performance of a given model. In the case of data-centric tools like GradCam, the solution propagates back to the data, which makes it hard to derive decisions regarding the neural architecture. However, the biggest concern in all aforementioned tools is the cost of computational resources and the integration of the analysis into the workflow of a deep learning practitioner. Tools like SVCCA and logistic regression probes require complex and computationally expensive procedures that need to be conducted after training. This naturally limits these techniques to

[*]co-first author
[†]co-first author

small benchmarks and primarily academic datasets like Cifar10 (Richter, Shenk, et al., 2021). An analysis tool that is to be used during the development of a deep learning-based model needs to be able to be used with as little computational and workflow overhead as possible. Ideally, the analysis can be done live while the training is in progress, allowing the researcher to interrupt potentially long-running training sessions to improve the model. Saturation was proposed in 2018 (Shenk, 2018) and later refined (Richter, Shenk, et al., 2021) and is the only analysis technique known to the authors that has this capability while allowing to identify parameter-inefficiencies in the setup (Richter, Shenk, et al., 2021; Richter, Byttner, et al., 2021; Richter, Schöning, et al., 2021). To make saturation usable in an application scenario, it is necessary to provide an easy-to-use framework that allows for an integration of the tool into the normal training and inference code with only minimally invasive changes. It is also necessary that the computation and analysis can be done online as part of the regular forward pass of the model, to make the integration as seamless as possible. A numerical comparison of these various methods is a promising avenue for future research into model introspection.

`Delve` is a tool for extracting information based on the covariance matrix of the data like saturation and the intrinsic dimensionality from neural network layers. To emphasize practical usability, special attention is placed on a low overhead and minimally invasive integration of `Delve` into existing training and inference setups: `Delve` hooks directly into PyTorch (Paszke et al., 2019) models to extract necessary information with little computational and memory overhead, thanks to an efficient covariance approximation algorithm. We enable the user to store and analyze the extracted statistics without changing their current experiment workflow, by making `Delve` easy to integrate into monitoring systems and making this interface easy to expand. This allows the user to utilize their preferred way of monitoring experiments, from simple CSV-Files and folder structures to more sophisticated solutions like TensorBoard (Abadi et al., 2015). A comprehensive source of documentation is provided on the homepage (http://delve-docs.readthedocs.io).

Statement of Need

Research on spectral properties of neural network representations has exploded in recent years (Alain & Bengio, 2016; Montavon et al., 2010; Morcos et al., 2018; Raghu et al., 2017; Selvaraju et al., 2019; Zhou et al., 2016). Publications like (Raghu et al., 2017) and (Richter, Shenk, et al., 2021) demonstrate that useful and interesting information can be extracted from the spectral analysis of these latent representations. It has also been shown that metrics like saturation (Shenk, 2018; Shenk et al., 2019) can be used to optimize neural network architectures by identifying pathological patterns hinting at inefficiencies of the neural network structure.

The main purpose of `Delve` is to provide easy and flexible access to these types of layer-based statistics. The combination of ease of usage and extensibility in `Delve` enables exciting scientific explorations for machine learning researchers and engineers. `Delve` has already been used in a number of scientific publications (Richter, Shenk, et al., 2021; Richter, Byttner, et al., 2021; Richter, Schöning, et al., 2021). The source code for `Delve` has been archived to Zenodo with the linked DOI: (Shenk et al., 2021)

Overview of the Library

The software is structured into several modules which distribute tasks. Full details are available at https://delve-docs.readthedocs.io/.

The TensorBoardX `SummaryWriter` (Abadi et al., 2015) is used to efficiently save artifacts like images or statistics during training with minimal interruption. A variety of layer feature statistics can be observed:

Statistic
intrinsic dimensionality
layer saturation (intrinsic dimensionality divided by feature space dimensionality
the covariance-matrix
the determinant of the covariance matrix (also known as generalized variance)
the trace of the covariance matrix, a measure of the variance of the data
the trace of the diagonal matrix, another way of measuring the dispersion of the data.
layer saturation (intrinsic dimensionality divided by feature space dimensionality)

Several layers are currently supported:

- Convolutional
- Linear
- LSTM

Additional layers such as PyTorch's ConvTranspose2D are planned for future development (see issue #43).

Eigendecomposition of the feature covariance matrix

The computation of saturation and other related metrics like the intrinsic dimensionality requires the covariance matrix of the layer's output. Computing the covariance matrix of a layer's output on the training or evaluation set is impractical to do naively, since it would require holding the entire dataset in memory. This would also contradict our goal of seamless integration in existing training loops, which commonly operate with mini-batches. Therefore, a batch-wise approximation algorithm is used to compute the covariance matrix online during training:

We can compute the covariance between two variables by using the covariance approximation algorithm for two random variables X and Y with n samples:

$$Q(X,Y) = \frac{\sum_{i=1}^{n} x_i y_i}{n} - \frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n^2}$$

Where x_i and y_i are individual observations of the respective random variables X and Y and n is the total number of samples. The advantage of this method is that only the number of seen samples, the sum of squares and the sum of the variables need to be stored, making the memory consumption per layer constant with respect to the size of the dataset. By computing Q(X, Y) for all possible combinations of features, we obtain the covariance matrix of the layer's output $Q(Z_l, Z_l)$, where Z_l is the layer's output over the entire dataset. We can parallelize the computations of all feature combinations by exploiting the shape of the layer output matrix A_l of the layer l: We can compute $\sum_{i=1}^{n} x_i y_i$ for all feature combinations in layer l by calculating the running squares $\sum_{b=0}^{B} A_{l,b}^T A_{l,b}$ of the batch output matrices $A_{l,b}$ where $b \in \{0, ..., B-1\}$ for B batches. We replace $\frac{(\sum_{i=1}^{n} x_i)(\sum_{i=1}^{n} y_i)}{n^2}$ by the outer product $\bar{A}_l \bigotimes \bar{A}_l$ of the sample mean $\bar{A}_l$. This is the running sum of all outputs $z_{l,k}$, where $k \in \{0, ..., n\}$ at training time, divided by the total number of training samples n. Our formula for a batch-wise approximated covariance matrix can now be written like this:

$$Q(Z_l, Z_l) = \frac{\sum_{b=0}^{B} A_{l,b}^T A_{l,b}}{n} - (\bar{A}_l \bigotimes \bar{A}_l)$$

The batch-wise updating algorithm allows us to integrate the approximation of the covariance matrix as part of the regular forward pass during training and evaluation. Our algorithm uses a thread-safe common value store on a single compute device or node, which furthermore allows updating the covariance matrix asynchronously when the network is trained in a distributed manner. To avoid problems that can be caused by rounding errors and numerical instability, our implementation of the algorithm converts by default all data into 64-bit floating-point values by default.

Another challenge is the dimensionality of the data in convolutional layers, where a simple flattening of the data vector would result in a very high dimensional vector and a computationally expensive singular value decomposition as a direct consequence. To address this issue, we treat every kernel position as an individual observation. This turns a 4th-degree output-tensor of shape (samples, height, width, filters) into a matrix of shape (samples · height · width, filters). The advantage of this strategy is that no information is lost, while keeping the dimensionality of Q at a manageable size. Optionally, to reduce the computations required further, the feature map can be automatically reduced in size using linear interpolation to a constant maximum height and width. Since information is lost during this process, this is disabled.

This approximation method was described alongside the saturation metric in the works of (Shenk, 2018; Shenk et al., 2019) and further refined by (Richter, Shenk, et al., 2021).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. https://www.tensorflow.org/

Alain, G., & Bengio, Y. (2016). Understanding intermediate layers using linear classifier probes. *ArXiv*, *abs/1610.01644*.

Chattopadhay, A., Sarkar, A., Howlader, P., & Balasubramanian, V. N. (2018). Grad-CAM++: Generalized gradient-based visual explanations for deep convolutional networks. *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*. https://doi.org/10.1109/wacv.2018.00097

Keskar, N. S., Mudigere, D., Nocedal, J., Smelyanskiy, M., & Tang, P. T. P. (2016). On large-batch training for deep learning: Generalization gap and sharp minima. *CoRR*, *abs/1609.04836*.

Montavon, G., Müller, K.-R., & Braun, M. L. (2010). Layer-wise analysis of deep networks with gaussian kernels. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, & A. Culotta (Eds.), *Advances in neural information processing systems 23* (pp. 1678–1686). Curran Associates, Inc. http://papers.nips.cc/paper/4061-layer-wise-analysis-of-deep-networks-with-gaussian-kernels.pdf

Morcos, A. S., Raghu, M., & Bengio, S. (2018). *Insights on representational similarity in neural networks with canonical correlation*. http://arxiv.org/abs/1806.05759

Novak, R., Bahri, Y., Abolafia, D. A., Pennington, J., & Sohl-Dickstein, J. (2018). Sensitivity and generalization in neural networks: An empirical study. *International Conference on Learning Representations*. https://openreview.net/forum?id=HJC2SzZCW

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in neural information* *processing systems 32* (pp. 8024–8035). Curran Associates, Inc. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Raghu, M., Gilmer, J., Yosinski, J., & Sohl-Dickstein, J. (2017). *SVCCA: Singular vector canonical correlation analysis for deep learning dynamics and interpretability*. http://arxiv.org/abs/1706.05806

Richter, M. L., Byttner, W., Krumnack, U., Schallner, L., & Shenk, J. (2021). Size matters. *CoRR*, *abs/2102.01582*. https://doi.org/10.1007/978-3-030-86340-1_11

Richter, M. L., Schöning, J., & Krumnack, U. (2021). Should you go deeper? Optimizing convolutional neural network architectures without training by receptive field analysis. *CoRR*, *abs/2106.12307*. https://arxiv.org/abs/2106.12307

Richter, M. L., Shenk, J., Byttner, W., Arpteg, A., & Huss, M. (2021). Feature Space Saturation during Training. *"32st British Machine Vision Conference 2021, BMVC 2021, Virtual Event, UK, November 22-25, 2021"*. https://www.bmvc2021-virtualconference.com/assets/papers/0108.pdf

Selvaraju, R. R., Cogswell, M., Das, A., Vedantam, R., Parikh, D., & Batra, D. (2019). Grad-CAM: Visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, *128*(2), 336–359. https://doi.org/10.1007/s11263-019-01228-7

Shenk, J. (2018). *Spectral Decomposition for Live Guidance of Neural Network Architecture Design* [Master's Thesis]. University of Osnabrück.

Shenk, J., Richter, M. L., Arpteg, A., & Huss, M. (2019). Spectral analysis of latent representations. *CoRR*, *abs/1907.08589*. http://arxiv.org/abs/1907.08589

Shenk, J., Richter, M. L., Byttner, W., & Marcinkiewicz, M. (2021). *Delve-team/delve: v0.1.45* (Version v0.1.45) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.5233860

Zhou, B., Khosla, A., Lapedriza, A., Oliva, A., & Torralba, A. (2016). Learning deep features for discriminative localization. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2921–2929. https://doi.org/10.1109/cvpr.2016.319