Literature Review:

Visualizing the Loss Landscape of Neural Nets

Joshua C. Cubero
SEAS8510-DA3
George Washington University

Background

Many years ago, one might have learned the very foundation of machine learning: $y = mx + b$, which one could use to predict a value for *y*. The foundation of machine learning, however, is to find Θ, such that $\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y_i})^2$ is minimized, which is just one of many loss functions. According to Teven, et. al, loss functions are implemented to measure the distance between the predicted value y and $\hat{y}$. Additionally, loss functions have a property, convexity, that if present make optimization possible via gradient decent. The inexperienced practitioner might explore the model's loss via a simple line plot, however, there exists a loss landscape of a neural network that provides valuable insights into the robust model's trainability and generalization (Ruigi, et. al, 2023). This literature review will delve into Li, et. al research paper: *Visualizing the Loss Landscape of Neural Nets.*

In the machine learning realm, one needs no convincing of the impact of network architecture and parameter selection on the model's trainability and generalization. However, what is not understood is why some neural network architectures and parameters train and generalize better than others. Additionally, the machine learning community has little understanding of the underlying loss surface (Li, et. al, 2017). The research team sought to uncover and understand loss function visualizations and how the loss landscape's geometry impacts the loss function and trainability.

The research team discussed two less effect methods of visualizing the loss function. First, the team discussed 1-D linear interpolation, which uses two parameter vectors Θ and Θ' which are used to analyze the flatness and sharpness of the minima (Li, et. al, 2017). This method of visualization suffers for two reasons: 1). The 1-D visualization lacks the ability to visualize convex surfaces; 2). 1-D linear interpolation does not account for batch normalization or invariance. Thus, sharpness comparisons tend to be misleading. Next, Li, et. al, surveyed contour plots and random directions which can be used to understand the different trajectories of various minimization methods. Contour plots are computationally expensive and produce low-resolution images and fail to capture complex non-convex loss surfaces (Li, et. al, 2017). Considering these limited loss landscape visualizations, Li, et. al proposed the implementing visualization of loss landscape via filter-wise normalization.

Proposed Solution

Filter-Wise Normalization supersedes simple random directions approach, which are limited by scale invariance of network weights (Li, et. al, 2017). Scale invariance limits the researcher's ability to meaningfully understand side by side comparison of plots. Li, et. al state that a neural network with larger weights may appear smooth and unfettered by a one-unit perturbation, whereas smaller weights could be devastated. The research team removes scaling invariance by implementing filter-wise normalization, which normalizes filter *d* to match the norm of Θ. Now that the team addressed scaling, they sought to understand the Sharp vs. Flat Dilemma.

Li, et. al test the widely debated postulate that small-batch stochastic gradient descent produces models that generalize well, and large-batches produce models with sharp minima and poor generalization. The research team trained a 9-layer VGG network using the CIFAR-10 image classification dataset, implemented 1-D linear interpolation and filter normalized plots. The researchers found that the 1-D linear interpolation indeed captured sharpness resulting from differences in weight scale. These differences occurred because small batches update more frequently and increase weight decay, which does enable the depiction of sharpness. Visualization, however, does not depict minimizer sharpness, but is indeed capturing the effect of weight scaling.

The researchers then implemented the same plots with filter normalized plots and found that this method eliminates contrast in geometry that is caused by scaling. Normalized plots displayed the same minimizer sharpness as 1-D interpolation, but the differences are much more subtle in the normalized plots. Furthermore, the research team implemented contour plots and random directions with greater success and readability with normalized filters.

Results

Li, et. al, confirmed that trainability and generalization are indeed correlated to network architectures and parameter initialization. The team also found that the depth of a neural network has a profound impact on loss landscape, particularly when skip connections are not implemented. Li, et. al also implemented shortcut connections that prevent model chaos as the neural network depth increases. Next, the team increased filters $k = \{2, 4, 8\}$, and found that this increased network width, and thus reduced chaos with wide and flat minima. Furthermore, the team showed that landscape geometry is a strong indicator of the model's ability to generalize. Lastly, the team visualizes convexity by mapping the ratio of the absolute value of the min and maximum eigenvalues of the Hessian matrix. This enables the team to visualize convexity in the Resnet-56 and DenseNet-121 models.

Conclusion

The team of researchers led by Li provided insight into the architectural dilemma faced by practitioners at the time of writing. The team acknowledged that much of the neural network research of the time was based on anecdotal research and theories based on complex assumptions. More recently, however, Xin-Chun Li, Lan Li, and De-Chuan Zhan penned new loss landscape research titled: *Visualizing, Rethinking, and Mining the Loss Landscape of Deep Neural Networks.* Zhan et. al go on to conduct loss landscape research on models of greater complexity by visualizing the loss landscape of mainstream deep neural networks (Zhan et. al, 2024). Zhan et. al conclude, like Li, et. al, that proving the complexity of 1-D loss curves remains an open question in the deep learning space, and thus requires further exploration.

Li, H., Xu, Z., Taylor, G., Studer, C., & Goldstein, T. (n.d.). *Visualizing the Loss Landscape of Neural Nets*. https://arxiv.org/pdf/1712.09913

Li, X.-C., Li, L., & Zhan, D.-C. (2024). Visualizing, Rethinking, and Mining the Loss Landscape of Deep Neural Networks. *ArXiv (Cornell University)*. https://doi.org/10.48550/arxiv.2405.12493

Terven, J., Cordova-Esparza, D., Ramirez-Pedraza, A., & Chavez-Urbiola, E. (2023). *LOSS FUNCTIONS AND METRICS IN DEEP LEARNING A PREPRINT*. https://arxiv.org/pdf/2307.02694

Ding, R., Li, T., Huang, X., Yanıkoglu, B., & Buntine, W. (n.d.). Better Loss Landscape Visualization for Deep Neural Networks with Trajectory Information. *Proceedings of Machine Learning Research*, *222*, 2023–2023. Retrieved November 24, 2024, from https://proceedings.mlr.press/v222/ding24a/ding24a.pdf

Singular Value Decomposition: GitHub – SVD Paper Compare & Contrast

Joshua C. Cubero
SEAS8510-DA3
George Washington University

GitHub SVD – SVD Comprehensive Guide

We are presented with the task of analyzing the SVD comprehensive guide and comparing the SVD implementation in the course GitHub repository. The SVD guide provides a comprehensive explanation of how one calculates SVD, displaying the inner workings of $A = U\Sigma V^T$. The SVD guide demonstrates calculations from $A^TA \rightarrow \det(A^TA - \lambda I) \rightarrow u_i = 1/\sigma_i Av_i \rightarrow U\Sigma V^T$. Thus, the comparison to the course GitHub implementation would seek to ensure that code followed this same pattern.

Findings

The course GitHub notebook contained two SVD implementations: SVD from scratch with NumPy and an implementation using the NumPy's SVD method. The SVD sans NumPy SVD function occurs in the first block of code from lines 20 to 42. One can see that $U\Sigma V^T$ are calculated from A^TA to completion of SVD. Lines 46 to 80 reconstruct the image from $U\Sigma V^T$, displays the images, then displays Approximation Error vs. Rank k analysis.

Next, SVD is implemented using the NumPy SVD method, which returns a tuple of values
$U\ \Sigma\ V^T$. From here, the code follows the same pattern as the from scratch implementation by reconstructing, displaying, and analyzing the images as presented earlier. In conclusion, the GitHub notebook appears to be a genuine implementation of SVD as seen in the SVD comprehensive guide.