Visualizing Deep Network Training Trajectories with PCA

Eliana Lorch ELIANALORCH@GMAIL.COM
Thiel Fellowship

Abstract

Neural network training is a form of numerical optimization in a high-dimensional parameter space. Such optimization processes are rarely visualized because of the difficulty of representing high-dimensional dynamics in a visually intelligible way. We present a simple method for rendering the optimization trajectories of deep networks with low-dimensional plots, using linear projections obtained by principal component analysis. We show that such plots reveal visually distinctive properties of the training processes, and outline opportunities for future investigation.

1. Introduction

Deep neural networks can be trained to perform highly complex tasks beyond the abilities of classical algorithms, but the training process itself—the path that stochastic optimizers take through parameter space, from an initial randomized network to an effectively trained one—is not very well understood. In convex optimization, there is a strong theoretical understanding of the path that different optimization methods take and how fast they converge to the global optimum (Bertsekas, 2011), but deep neural network loss functions are very non-convex. There are some theoretical results (e.g. about the nature of obstacles in the loss landscape; Dauphin et al., 2014), but we don't yet have a thorough understanding.

Geometric intuition might reveal aspects of optimization in practice that could help improve our training techniques or inspire theoretical investigations of particular phenomena. To visualize parameter space, with hundreds of thousands or millions of dimensions, one approach is to reduce the dimensions down to just two or three. We are interested in choosing a linear subspace that maximally preserves the optimization trajectory's shape, in order to observe patterns that are meaningful in the full parameter space.

Proceedings of the 33^{rd} International Conference on Machine Learning, New York, NY, USA, 2016. JMLR: W&CP volume 48.

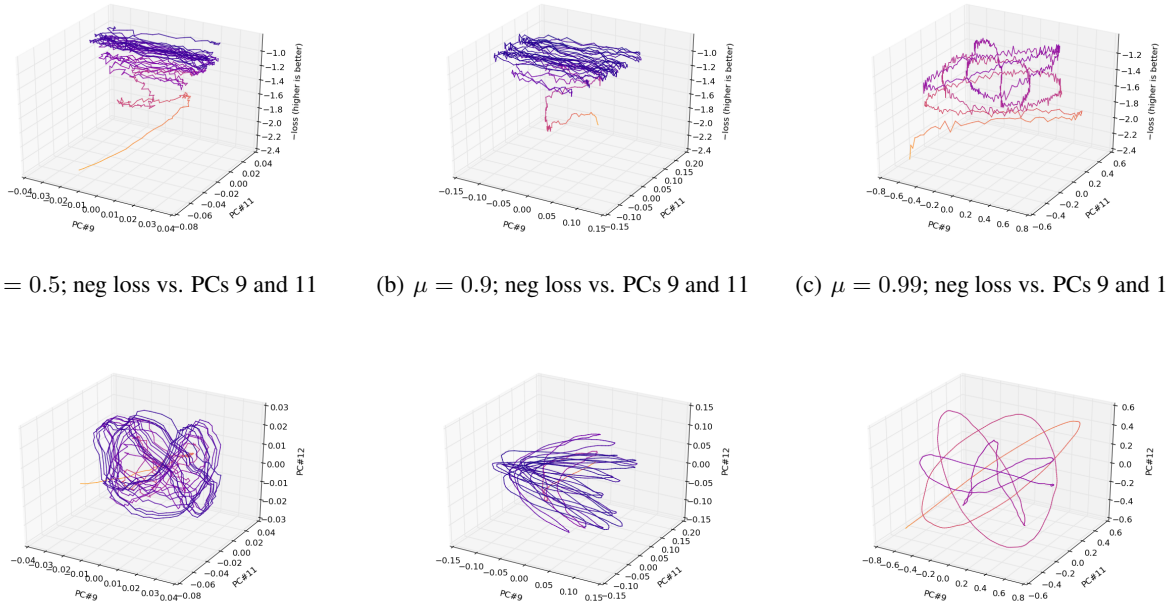
2. Related Work

Most visualizations of deep neural networks illustrate aspects of the network itself; there is not much work on visualizing the training trajectory. The primary work in this area (Goodfellow et al., 2014) renders the loss surface along two dimensions: the line between the initial and final parameter vectors, and for each point along that line, the unique perpendicular that passes through the corresponding point on the training path. By straightening out this path, it's easier to see what (if any) structures in the cost surface the training process encounters along the way (e.g. plateaus, valleys, local minima), but the trade-off here is that deviations of the same magnitude from this straight line look identical, regardless of the particular shape the path takes instead. This visualization reveals interesting and surprising properties of the loss surface; a different set of properties may be visualized by rendering the training trajectory in a way that preserves its shape as much as possible.

3. Method

Our approach is simple: we reduce the dimensionality of the parameter trajectory by applying principal component analysis (PCA), then plot two or three of the principal coordinates (PCs) as a visualization. PCA identifies dimensions which capture the most variance across the training processthat is, directions in which there is a lot of movement. It's intuitively plausible that such directions would be interesting to examine visually. Yet although we are identifying "interesting" directions in a data-dependent way, a PCA rendering is just a particular linear projection of the full parameter space, which means it's a form of dimensionality reduction that is likely to preserve some geometric intuition.

The method in detail is to take snapshots of the parameters throughout training, and then after training is complete, treat the collection of snapshots as a set of n-dimensional vectors (where n is the number of parameters in the network). We could consider these vectors to be samples from a hypothetical distribution of parameter settings we would be likely to see at any point during training. Running PCA



(a) $\mu = 0.5$; neg loss vs. PCs 9 and 11 (b) $\mu = 0.9$; neg loss vs. PCs 9 and 11 (c) $\mu = 0.99$; neg loss vs. PCs 9 and 11

(d) $\mu = 0.5$; PCs 9, 11, 12 (e) $\mu = 0.9$; PCs 9, 11, 12 (f) $\mu = 0.99$; PCs 9, 11, 12

Figure 1. The top row shows a 3D parametric plot of the PCA-projected training trajectory, with negative test loss on the z axis (higher is better) and principal components 9 and 11 on the x and y axes, for each of three separate training runs with different values for momentum. The trajectory is colored from yellow to indigo, where yellow indicates high loss and indigo indicates low loss; all the plots in this paper use the same color scale (each color always represents the same absolute test loss). The corresponding plots in the bottom row show the same training runs, but replace negative loss on the z axis with principal component 12.

on this sample set corresponds to finding a linear projection that explains the most variance in this distribution, or equivalently, one that minimizes the reconstruction error of backprojection.

4. Experiments

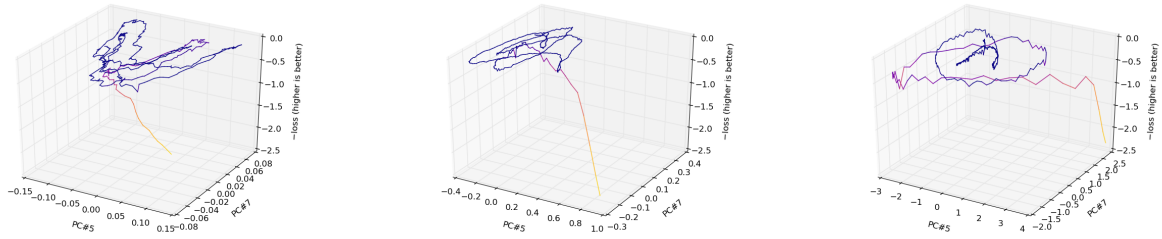
We performed a number of exploratory experiments on simple CIFAR-10 and MNIST models trained with SGD (with momentum) and ADAM (Kingma & Ba, 2015), with various hyperparameters, and observed a number of consistent patterns. We used the `cifar10_quick` and MNIST `lenet` models included in the deep learning framework Caffe (Jia et al., 2014) with various optimizer configurations. In each experiment, we ran the optimizer on CIFAR-10 for 5,000 iterations, capturing a snapshot every 5 iterations, (for a total of 1,000 snapshots); on MNIST, we ran 10,000 iterations and captured 2,000 snapshots.

We then performed PCA using `scikit-learn` (Pedregosa et al., 2011), extracting the top 12 principal components. Approximations to PCA (Szlam et al., 2014) result in the same visual features and run an order of magnitude faster (on our data), which is useful for large networks or long training trajectories. For our CIFAR-10 trajectories (1,000 parameter snapshots $\times$ 145,578 parameters), exact PCA takes only 1–2 minutes. Initial runs on our MNIST trajectories (2,000 parameter snapshots $\times$ 431,080 parameters) took approximately 20 minutes, so we used `scikit-learn`'s implementation of randomized PCA (Szlam et al., 2014) for our MNIST figures.

We then projected each training trajectory onto every possible pair or triplet of its top 12 principal components (PCs). We also augmented the 2D plots (trajectories projected onto PC pairs) with a z-axis representing negative test loss. Included in this paper are samples of these plots, where the PCs were selected by hand to highlight visual features.

4.1. Momentum

We performed training runs with various values of the momentum hyperparameter μ, between 0 to 0.99. In all of these runs, the learning rate α was set to 0.001. For both CIFAR-10 and MNIST, training with low momentum produces dense, jagged paths (e.g., Figures 1(d), 2(d)); on the other hand, training with high momentum produces plots with sparse, arcing traces (e.g., Figures 1(f), 2(f)). The ex-



(a) $\mu = 0.5$; PCs 5, 7, 11 (b) $\mu = 0.9$; PCs 5, 7, 11 (c) $\mu = 0.93$; PCs 5, 7, 11

(d) $\mu = 0.5$; PCs 5, 7, 11 (e) $\mu = 0.9$; PCs 5, 7, 11 (f) $\mu = 0.93$; PCs 5, 7, 11

Figure 2. Examples of 2,500-iteration trajectories from the `lenet` MNIST model with varying values of momentum (μ). Note that higher momentum makes the traces increasingly smooth.

ample plots were chosen because the pattern is especially apparent in them, but we see this pattern generally across combinations of PCs on which the data is projected.

This pattern reinforces intuition about the role of momentum in stochastic gradient descent: it encourages the optimizer to maintain roughly the same direction across time steps, and discourages quick turns. As we can see in Figure 1(c), with too high of a momentum value, the training gradient itself is not given enough weight for the loss to be consistently reduced, and training does not converge in the given number of iterations (the color of the trajectory does not reach purple). On the other hand, when momentum is too low, training still succeeds, but takes longer to converge.

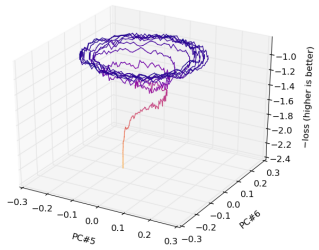
4.2. Oscillations

Another curious pattern this visualization reveals is that sometimes PC coordinates seem to be oscillating out of phase, tracing out circles (Figures 3(b), 3(a)) or Lissajous curves (Figures 3(c), 3(e)) in the plots. While multiple PCs appear to oscillate, not all do, and the ones that do vary depending on the training run. This means that a significant fraction, but not all, of the parameter variance during training is explained by oscillatory behavior. (Oscillations in all PCs would imply that the training process is dominated by oscillation, which would be even more surprising.)

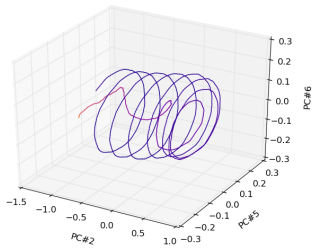
Such oscillations may be related to a learning dynamic theorized by (Qian, 1999), which shows (non-stochastic) gradient descent with momentum to be equivalent (in the limit, near a local optimum) to a system of coupled, damped harmonic oscillators (with masses determined by the momentum parameter). We see this pattern both in models trained with SGD and with ADAM, which suggests that harmonic oscillation may be a generally observable dynamic of gradient-based algorithms in practice (not just in the limit).

4.3. Predictable first principal components

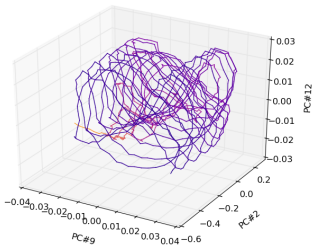
In every experiment we've run where the test loss followed the typical pattern of generally decreasing until its noticeably reduced (i.e. whenever training is "successful"), the first principal component (PC1) has always increased monotonically. PC2 always increases along the first principal component axis and then decreases again (usually around the halfway point). See Figure 4 for examples. We don't yet know what these components represent, but this pattern seems to be robust across the CIFAR-10 and MNIST models we trained with various optimization algorithms and hyperparameters.



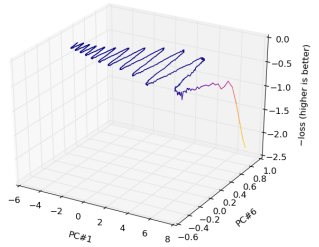
(a) $\mu = 0.9$; $-$loss vs. PCs 5, 6



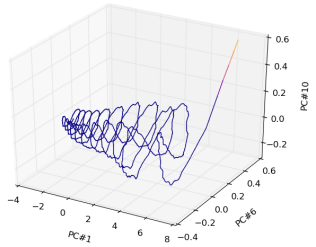
(b) $\mu = 0.9$; PCs 2, 5, 6



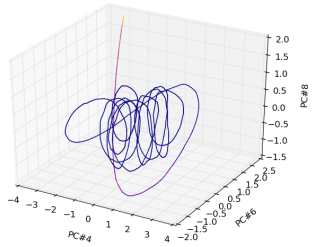
(c) $\mu = 0.5$; PCs 9, 2, 12



(d) MNIST; $\mu = 0.93$; $-$loss vs. PCs 1, 6

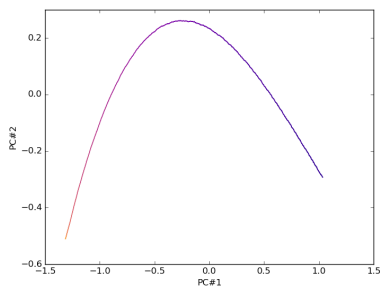


(e) MNIST; $\mu = 0.9$; PCs 1, 6, 10

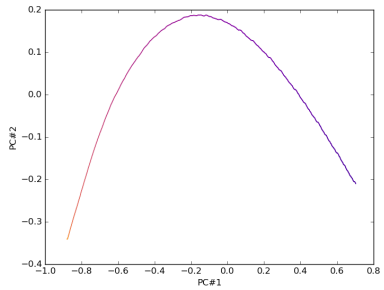


(f) MNIST; $\mu = 0.98$; PCs 4, 6, 8

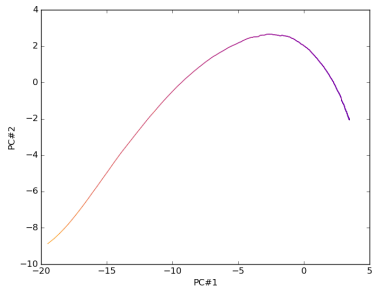
Figure 3. Examples of oscillatory behavior in various forms. Figures 3(a) and 3(b) are two different views of the same training run as Figures 1(b) and 1(e). Figure 3(c) is a different view of the same training run as Figure 1(a) and 1(d). Figures 3(d), 3(e), and 3(f) show similar behaviors on a different dataset, namely MNIST.



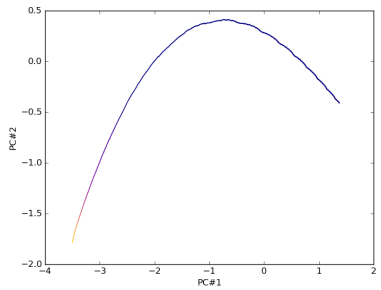
(a) $\mu = 0.5$; PCs 1, 2



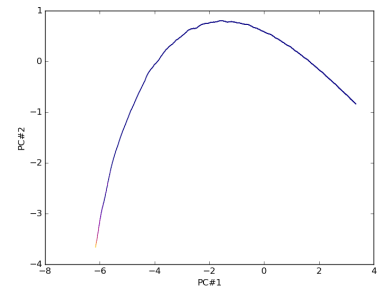
(b) $\alpha = 0.0001$; PCs 1, 2



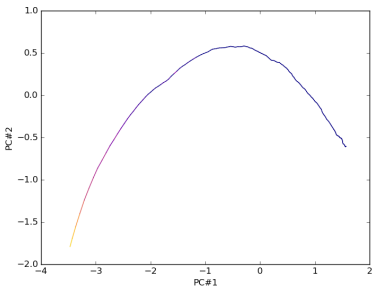
(c) ADAM; PCs 1, 2



(d) MNIST $\mu = 0.5$; PCs 1, 2



(e) MNIST $\mu = 0.9$; PCs 1, 2



(f) MNIST $\mu = 0.9$, 200; PCs 1, 2

Figure 4. PC1 and PC2 seem to always trace out similar shapes as long as training is successful. PC1 in particular is monotonic.

5. Future Work

We are interested in investigating several further questions:

- How robust are these patterns? We tested simple convolutional MNIST and CIFAR-10 models with various optimization algorithms and hyperparameters, but do other architectures (RNNs, LSTMs, etc) and larger datasets share these visual features? Are there distinguishing attributes between different optimization methods?
- Can we visualize the principal component axes themselves? Do they have any natural interpretations?
- Would it be useful to visualize subsets of the parameters (e.g., single layers or filters)?
- Would techniques that normalize the weights change the picture, such as batch normalization (Ioffe & Szegedy, 2015), path normalization (Neyshabur et al., 2016), natural gradient (Amari, 1998), or various kinds of data-dependent initialization (Kraehenbuehl et al., 2016; Mishkin & Matas, 2016)?
- Would it be useful to plot multiple training runs of the same model on the same axes (e.g. by applying a single PCA to all of them)?
- How do the principal component axes change depending on the segment of training over which the PCA is run? If the axes converge early in training, would it be possible to exploit them to accelerate training (e.g., by increasing PC1, if we know it will increase monotonically for the rest of training)?

6. Discussion

We select the highest-variance components of the parameter trajectories with PCA, and appear to pull out some "invariant" dynamics of the training process—the most dominant dynamics nearly always take the same form, and some oscillations are generally present—even with entirely different architectures and datasets. This visualization makes such structure visible where it would otherwise be obscured by the high dimensionality of the parameter space.

It's unclear whether this technique will lead to improvements in neural network optimization. However, it's one of very few tools so far that can give a visual representation of the process of training a deep network, exposing new patterns for further investigation.

References

Amari, S.-I. Natural gradient works efficiently in learning. *Neural Computation*, 10(2):251–276, 1998.

Bertsekas, D. P. Incremental gradient, subgradient, and proximal methods for convex optimization: A survey. In *Optimization for Machine Learning*, pp. 1–38. MIT Press, 2011.

Dauphin, Y., Pascanu, R., Gulcehre, C., Cho, K., Ganguli, S., and Bengio, Y. Identifying and attacking the saddle point problem in high-dimensional non-convex optimization. In *Advances in Neural Information Processing Systems 27*, pp. 2933–2941. 2014.

Goodfellow, Ian J, Vinyals, Oriol, and Saxe, Andrew M. Qualitatively characterizing neural network optimization problems. *arXiv preprint arXiv:1412.6544*, 2014.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, pp. 448–456, 2015.

Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

Kraehenbuehl, P., Doersch, C., Donahue, J., and Darrell, T. Data-dependent initialization of convolutional neural networks. In *International Conference on Learning Representations*, 2016.

Mishkin, D. and Matas, J. All you need is a good init. In *International Conference on Learning Representations*, 2016.

Neyshabur, B., Tomioka, R., Salakhutdinov, R., and Srebro, N. Data-dependent path normalization in neural networks. In *International Conference on Learning Representations*, 2016.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

Qian, N. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, 1999.

Szlam, A., Kluger, Y., and Tygert, M. An implementation of a randomized algorithm for principal component analysis. *arXiv preprint arXiv:1412.3510*, 2014.