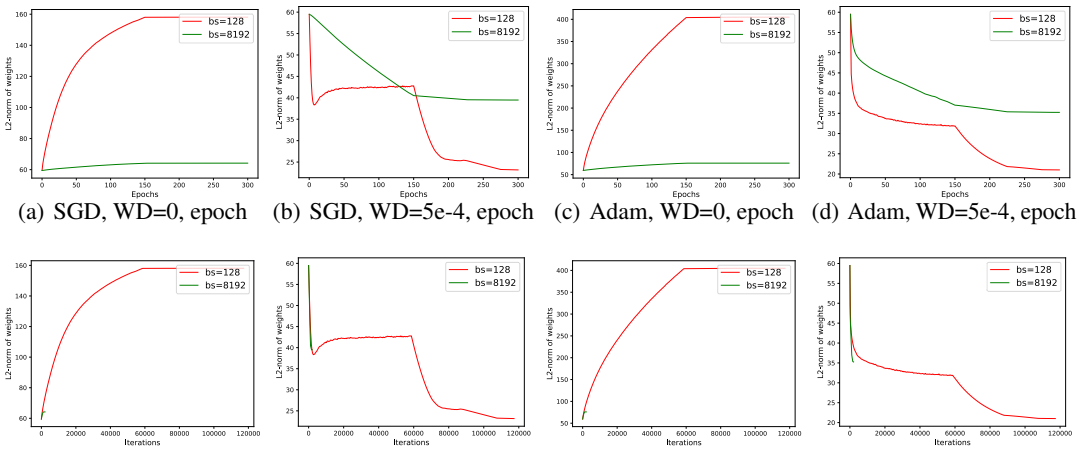
Visualizing the Loss Landscape of Neural Nets

A Comparison of Loss Surfaces

A.1 The Change of Weights Norm during Training

Figure 8 shows the change of weights norm during training in terms of epochs and iterations.



(a) SGD, WD=0, epoch (b) SGD, WD=5e-4, epoch (c) Adam, WD=0, epoch (d) Adam, WD=5e-4, epoch

(e) SGD, WD=0, iter (f) SGD, WD=5e-4, iter (g) Adam, WD=0, iter (h) Adam, WD=5e-4, iter

Figure 8: The change of weights norm during training for VGG-9. When weight decay is disabled, the weight norm grows steadily during training without constraints. When nonzero weight decay is adopted, the weight norm decreases rapidly at the beginning and becomes stable until the learning rate is decayed. Since we use a fixed number of epochs for different batch sizes, the difference in weight norm change between large-batch and small-batch training is mainly caused by the larger number of updates when a small batch is used. As shown in the second row, the changes of weight norm are at the same pace for both small and large batch training in terms of iterations.

A.2 Comparision of Normalization Methods

Here we compare several normalization methods for a given random normal direction d. Let θ_i denote the weights of layer i and $\theta_{i,j}$ represent the j-th filter in the i-th layer.

- **No Normalization** In this case, the direction d is added to the weights directly without processing.
- **Filter Normalization** The direction d is normalized so that the direction for each filter has the same norm as the corresponding filter in θ,

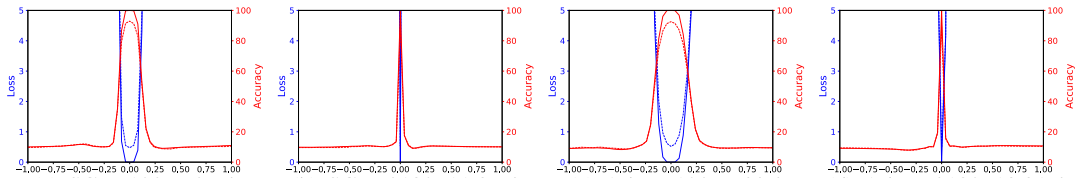
$$d_{i,j} \leftarrow \frac{d_{i,j}}{\|d_{i,j}\|}\|\theta_{i,j}\|.$$

  This is the approach advocated in this article, and is used extensively for plotting loss surfaces.

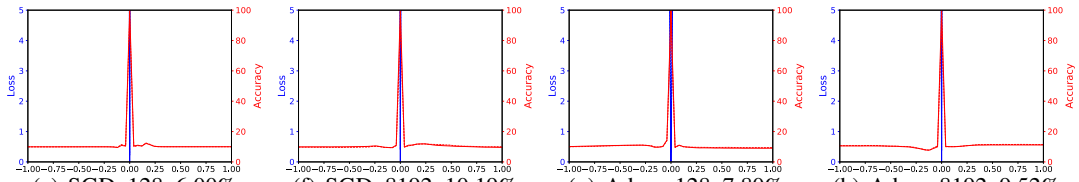
- **Layer Normalization** The direction d is normalized in the layer level so that the direction for each layer has the same norm as the corresponding layer of θ,

$$d_i \leftarrow \frac{d_i}{\|d_i\|}\|\theta_i\|.$$

Figure 9 shows the 1D plots without normalization. One issue with the non-normalized plots is that the x-axis range must be chosen carefully. Figure 10 shows enlarged plots with $[-0.2, 0.2]$ as the range for the x-axis. Without normalization, the plots fail to show consistency between flatness and generalization error. Here we compare filter normalization with layer normalization. We find filter normalization is more accurate than layer normalization. One failing case for layer normalization is shown in Figure 11, where Figure 11(g) is flatter than Figure 11(c), but with worse generalization error.

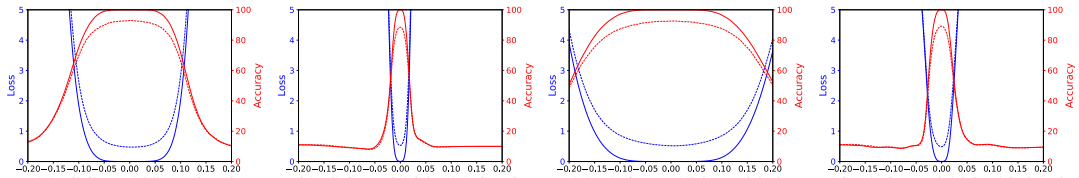


(a) SGD, 128, 7.37% (b) SGD, 8192, 11.07% (c) Adam, 128, 7.44% (d) Adam, 8192, 10.91%

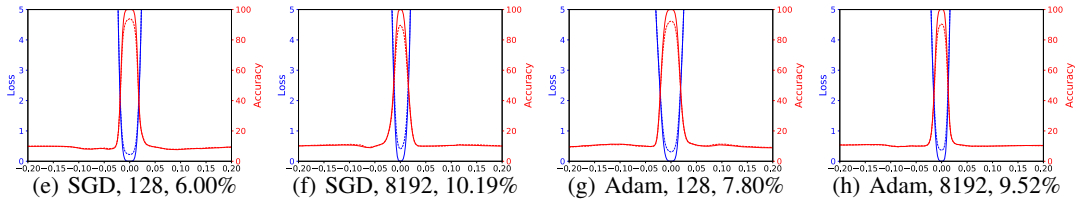


(e) SGD, 128, 6.00% (f) SGD, 8192, 10.19% (g) Adam, 128, 7.80% (h) Adam, 8192, 9.52%

Figure 9: 1D loss plots for VGG-9 without normalization. The first row has no weight decay and the second row uses weight decay 0.0005.

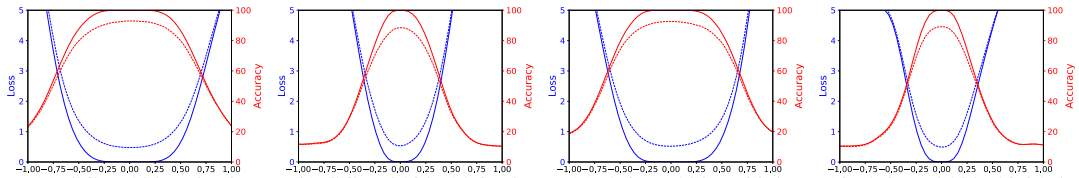


(a) SGD, 128, 7.37% (b) SGD, 8192, 11.07% (c) Adam, 128, 7.44% (d) Adam, 8192, 10.91%

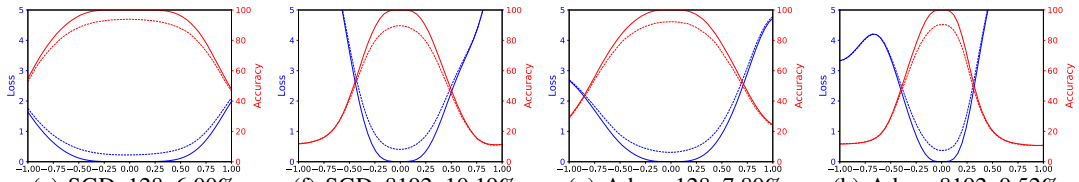


(e) SGD, 128, 6.00% (f) SGD, 8192, 10.19% (g) Adam, 128, 7.80% (h) Adam, 8192, 9.52%

Figure 10: Enlarged Figure 9. The range of the x-axis is [-0.2, 0.2] instead of [-1.0, 1.0]. The first row has no weight decay and the second row uses weight decay 0.0005. The pairs (a, e) and (c, g) show that sharpness of minima does not correlate well with test error.



(a) SGD, 128, 7.37% (b) SGD, 8192, 11.07% (c) Adam, 128, 7.44% (d) Adam, 8192, 10.91%



(e) SGD, 128, 6.00% (f) SGD, 8192, 10.19% (g) Adam, 128, 7.80% (h) Adam, 8192, 9.52%

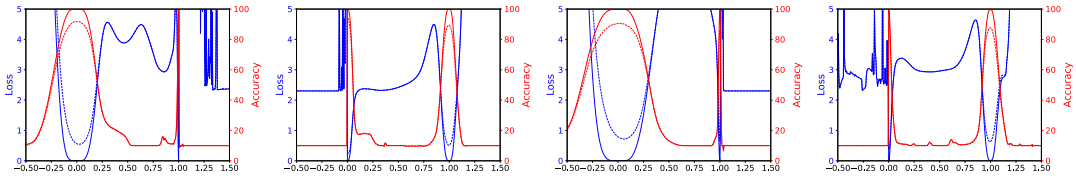
Figure 11: 1D loss plots for VGG-9 with layer normalization. The first row has no weight decay and the second row uses weight decay 5e-4.

A.3 Small-Batch vs Large-Batch for ResNet-56

Similar to the observations made in Section 5, the “sharp vs flat dilemma" also applies to ResNet-56 as shown in Figure 12. The generalization error for each solution is shown in Table 1. The 1D and 2D visualizations with filter normalized directions are shown in Figure 13.

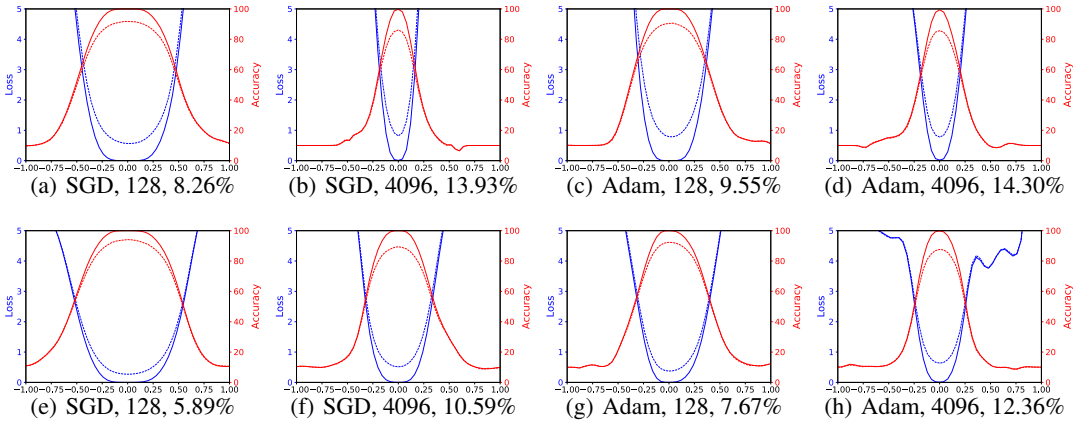
Table 1: Test errors for ResNet-56 with different optimizer, batch-size and weight-decay.

	SGD		Adam	
	bs=128	bs=4096	bs=128	bs=4096
WD = 0	8.26	13.93	9.55	14.30
WD = 5e-4	**5.89**	**10.59**	**7.67**	**12.36**



(a) SGD, WD=0 (b) SGD, WD=5e-4 (c) Adam, WD=0 (d) Adam, WD=5e-4

Figure 12: 1D linear interpolation of solutions obtained by small-batch and large-batch methods for ResNet-56. The blue lines are loss values and the red lines are error.



(a) SGD, 128, 8.26% (b) SGD, 4096, 13.93% (c) Adam, 128, 9.55% (d) Adam, 4096, 14.30%

(e) SGD, 128, 5.89% (f) SGD, 4096, 10.59% (g) Adam, 128, 7.67% (h) Adam, 4096, 12.36%

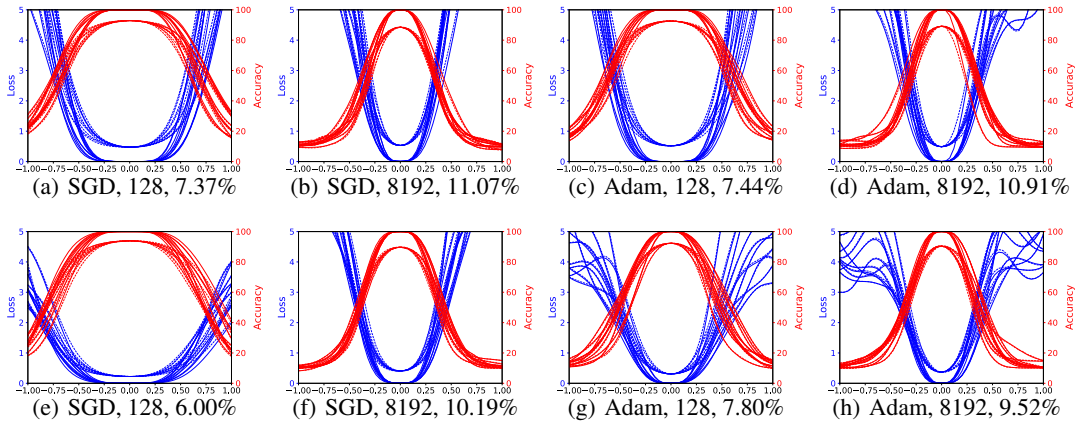
(i) SGD, 128, 8.26% (j) SGD, 4096, 13.93% (k) Adam, 128, 9.55% (l) Adam, 4096, 14.30%

(m) SGD, 128, 5.89% (n) SGD, 4096, 10.59% (o) Adam, 128, 7.67% (p) Adam, 4096, 12.36%

Figure 13: 1D and 2D visualization of ResNet-56 trained with different optimizer, batch size, and weight decay. The first and third row uses zero weight decay and the second and fourth row uses 5e-4 weight decay.

A.4 Repeatability of the Loss Surface Visualization

Do different random directions produce dramatically different plots? We plot the 1D loss surface of VGG-9 with 10 random filter-normalized directions. As shown in Figure 14, the plots are very close in shape. We also repeat the 2D loss surface plots multiple times for ResNet-56-noshort, which has worse generalization error. As shown in Figure 15, there are apparent changes in the loss surface for different plots, however, the qualitative choatic behaviour is quite consistent across plots.



(a) SGD, 128, 7.37% (b) SGD, 8192, 11.07% (c) Adam, 128, 7.44% (d) Adam, 8192, 10.91%

(e) SGD, 128, 6.00% (f) SGD, 8192, 10.19% (g) Adam, 128, 7.80% (h) Adam, 8192, 9.52%

Figure 14: Repeatability of the surface plots for VGG-9 with filter normalization. The shape of minima obtained using 10 different random filter-normalized directions.

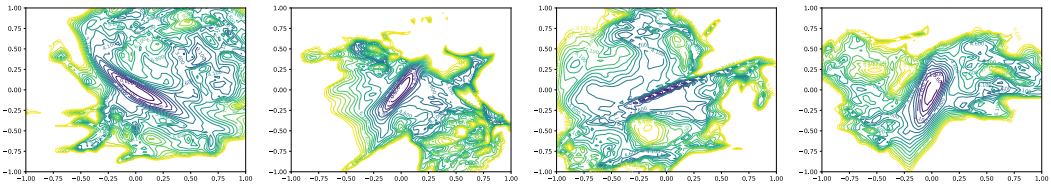


Figure 15: Repeatability of the 2D surface plots for ResNet-56-noshort. The model is trained with batch size 128, initial learning rate 0.1 and weight decay 5e-4. The final training loss is 0.192, the training error is 6.49 and the test error is 13.31.

A.5 Implementation Details

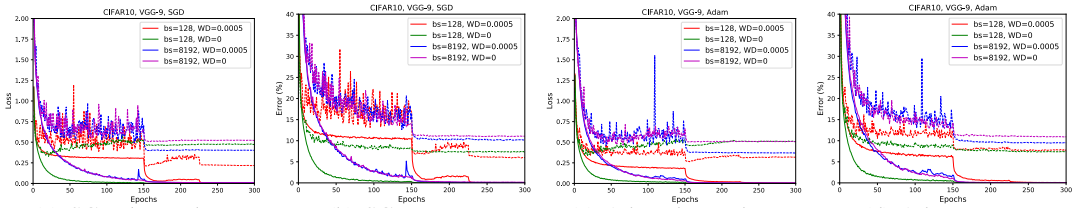
Computing resources for generating the figures Our PyTorch code can be executed in a multiple GPU workstation as well as an HPC with hundreds of GPUs using `mpi4py`. The computation time depends on the model's inference speed on the training set, the resolution of the plots, and the number of GPUs. The resolution for the 1D plots in Figure 3 is 401×401. The default resolutions used for the 2D contours in Figure 3 and Figure 5 is 51×51. We use higher resolutions (251×251) for the ResNet-56-noshort used in Figure 1 to show more details. For reference, a 2D contour plot of ResNet-56 with a (relatively low) resolution of 51×51 will take about 1 hour on a workstation with 4 GPUs (Titan X Pascal or 1080 Ti).

Batch Normalization parameters In the 1D linear interpolation methods, the Batch Normalization (BN) parameters including the "running mean" and "running variance" need to be considered as part of θ. If these parameters are not considered, then it is not possible to reproduce the exact loss values for both minimizers. In the filter-normalized visualization, the random direction perturbs all weights except batch norm parameters. Note that the filter normalization process removes the effect of weight scaling, and so the batch normalization can be ignored.

The VGG-9 architecture and parameters for Adam VGG-9 is a cropped version of VGG-16, which keeps the first 7 Conv layers in VGG-16 with 2 FC layers. A BN layer is added after each conv layer and the first FC layer. We find VGG-9 is an efficient network with better performance comparing to VGG-16 on CIFAR-10. We use the default values for β_1, β_2 and ϵ in Adam with the same learning rate schedule as used in SGD.

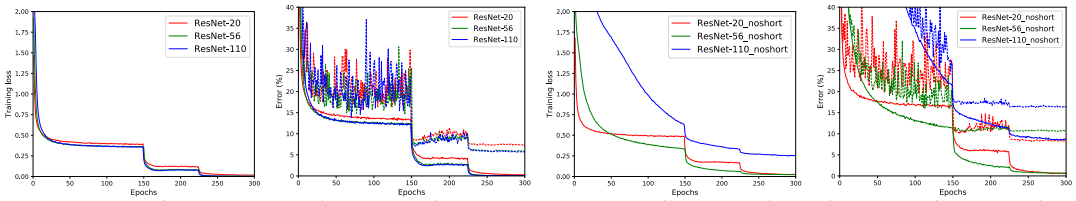
A.6 Training Curves for VGG-9 and ResNets

The loss curves for training VGG-9 used in Section 5 are shown in Figure 16. Figure 17 shows the loss curves and error curves of architectures used in Section 6 and Table 2 shows the final error and loss values. The default setting for training is using SGD with Nesterov momentum, batch-size 128, and 0.0005 weight decay for 300 epochs. The default learning rate was initialized at 0.1, and decreased by a factor of 10 at epochs 150, 225 and 275.



(a) SGD, loss values (b) SGD, errors (c) Adam, loss values (d) Adam, errors

Figure 16: Training loss/error curves for VGG-9 with different optimization methods. Dashed lines are for testing, solid for training.



(a) ResNet-CIFAR (b) ResNet-CIFAR (c) ResNet-CIFAR-noshort (d) ResNet-CIFAR-noshort

Figure 17: Convergence curves for different architectures.

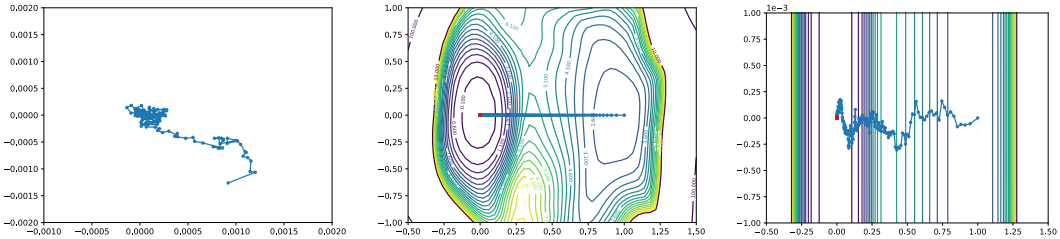
Table 2: Loss values and errors for different architectures trained on CIFAR-10.

	init LR	Training Loss	Training Error	Test Error
ResNet-20	0.1	0.017	0.286	7.37
ResNet-20-noshort	0.1	0.025	0.560	8.18
ResNet-56	0.1	0.004	0.052	5.89
ResNet-56-noshort	0.1	0.192	6.494	13.31
ResNet-56-noshort	0.01	0.024	0.704	10.83
ResNet-110	0.1	0.002	0.042	5.79
ResNet-110-noshort	0.01	0.258	8.732	16.44

B Visualizing Optimization Paths

Finally, we explore methods for visualizing the trajectories of different optimizers. For this application, random directions are ineffective. We will provide a theoretical explanation for why random directions fail, and explore methods for effectively plotting trajectories on top of loss function contours.

Several authors have observed that random direction fail to capture the variation in optimization trajectories, including [10, 29, 28, 27]. Several failed visualizations are depicted in Figure 18. In Figure 18(a), we see the iterates of SGD projected onto the plane defined by two random directions. Almost none of the motion is captured (notice the super-zoomed-in axes and the seemingly random walk). This problem was noticed by [13], who then visualized trajectories using one direction that points from initialization to solution, and one random direction. This approach is shown in Figure 18(b). As seen in Figure 18(c), the random axis captures almost no variation, leading to the (misleading) appearance of a straight line path.



(a) Two random directions (b) Random direction for y-axis (c) Enlarged version (b)

Figure 18: Ineffective visualizations of optimizer trajectories. These visualizations suffer from the orthogonality of random directions in high dimensions.

B.1 Why Random Directions Fail: Low Dimensional Optimization Trajectories

It is well-known that two random vectors in a high dimensional space will be nearly orthogonal with high probability. In fact, the expected cosine similarity between Gaussian random vectors in n dimensions is roughly $\sqrt{2/(\pi n)}$ ([12], Lemma 5).

This is problematic when optimization trajectories lie in extremely low dimensional spaces. In this case, a randomly chosen vector will lie orthogonal to the low-rank space containing the optimization path, and a projection onto a random direction will capture almost no variation. Figure 18(b) suggests that optimization trajectories are low dimensional because the random direction captures orders of magnitude less variation than the vector that points along the optimization path. Below, we use PCA directions to directly validate this low dimensionality, and also to produce effective visualizations.

B.2 Effective Trajectory Plotting using PCA Directions

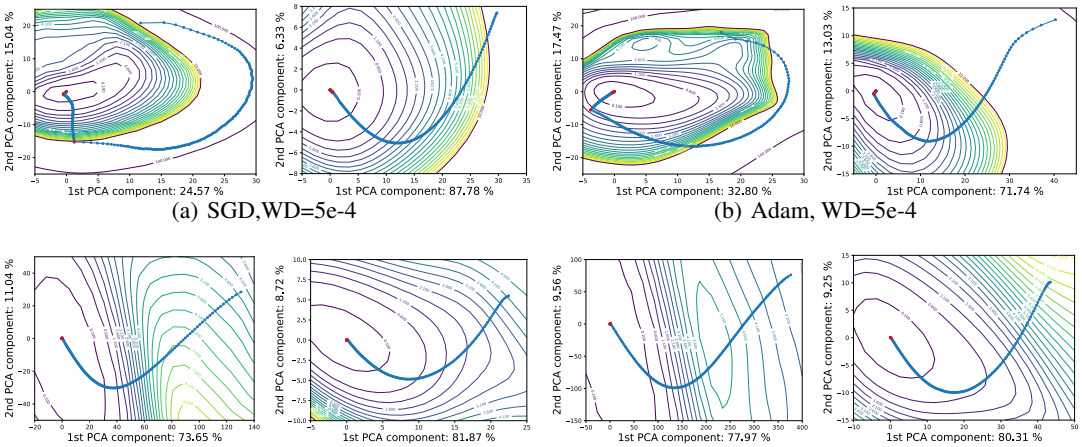
To capture variation in trajectories, we need to use non-random (and carefully chosen) directions. Here, we suggest an approach based on PCA that allows us to measure how much variation we've captured; we also provide plots of these trajectories along the contours of the loss surface.

Let θ_i denote model parameters at epoch i and the final estimate as θ_n. Given n training epochs, we can apply PCA to the matrix $M = [\theta_0 - \theta_n; \cdots; \theta_{n-1} - \theta_n]$, and then select the two most explanatory directions. Optimizer trajectories (blue dots) and loss surfaces along PCA directions are shown in Figure 19. Epochs where the learning rate was decreased are shown as red dots. On each axis, we measure the amount of variation in the descent path captured by that PCA direction.

We see some interesting behavior in these plots. At early stages of training, the paths tend to move perpendicular to the contours of the loss surface, i.e., along the gradient directions as one would expect from non-stochastic gradient descent. The stochasticity becomes fairly pronounced in several plots during the later stages of training. This is particularly true of the plots that use weight decay and small batches (which leads to more gradient noise, and a more radical departure from deterministic gradient directions). When weight decay and small batches are used, we see the path turn nearly parallel to the contours and "orbit" the solution when the stepsize is large. When the stepsize is dropped (at the red dot), the effective noise in the system decreases, and we see a kink in the path as the trajectory falls into the nearest local minimizer.

Finally, we can directly observe that the descent path is very low dimensional: between 40% and 90% of the variation in the descent paths lies in a space of only 2 dimensions. The optimization trajectories in Figure 19 appear to be dominated by movement in the direction of a nearby attractor. This low dimensionality is compatible

with the observations in Section 6, where we observed that non-chaotic landscapes are dominated by wide, nearly convex minimizers.



(a) SGD,WD=5e-4 (b) Adam, WD=5e-4

(c) SGD, WD=0 (d) Adam,WD=0

Figure 19: Projected learning trajectories use normalized PCA directions for VGG-9. The left plot in each subfigure uses batch size 128, and the right one uses batch size 8192.