A. Why Compute the Condition Number?

There are many summary statistics one could compute from the spectrum of the Jacobian. It is not obvious a priori that it makes sense to focus on the ratio of the maximum eigenvalue to the minimum eigenvalue, so here we make some attempt to justify that decision.

If you were to just glance at the spectra figures provided in the main text, using the log-determinant might seem like a reasonable thing to do. However, we note that (at least for the MNIST experiments) the largest singular values for the 'well behaved' runs are distinctly lower than those for the 'poorly behaved' ones. This suggests that the conditioning might be more pertinent than the determinant.

Even given that the conditioning is what's relevant, one could imagine other measures of Jacobian conditioning that less strongly emphasize the extreme singular values. Indeed, computing such quantities would be a useful exercise, and we expect that they would also correlate with GAN performance, but we have kept the condition number because it is simple and well-understood. We also feel that the condition number most closely relates to what is being optimized by the Jacobian Clamping procedure.

B. Additional Experimental Results

This section contains results that we have included for the purpose of completeness but which were not necessary for following the narrative of the paper. References to this section can be found in the main text.

B.1. Misbehaving Generators can be Well-Conditioned

We have observed that intervening to improve generator conditioning improves generator performance during GAN training. We also might like to know whether this relationship holds for all possible generators. Here we provide a counterexample of a deliberately pathological generator (not trained with a GAN loss) which is nonetheless well-conditioned. This suggests that the causal relationship we explore in the main text may relate to the GAN training process, and may not be an absolute property of generators in isolation.

We train a generator using the DCGAN architecture with a latent space of 64 dimensions. Rather than an adversarial loss, we train with an L2 reconstruction loss - in effect, teaching the generator to memorize the training examples it has seen. We select 10,000 examples to memorize: half of them (5,000) are random MNIST digits, and the other half are identical copies of a single MNIST sample (in this case, a four). We then establish a consistent but arbitrary mapping from 10,000 random z values to the training examples. The generator is trained with an L2 reconstruction loss to map each memorized z value to its associated training example. The generator's behavior on non-memorized z values is not considered at training time. There is no discriminator involved in this training procedure. Figure 8 shows the generator's output when provided the z values it was trained to associate with specific samples, indicating that it succeeds at memorizing the half-random half-identical data it was trained on.

At evaluation time, we provide random latent vectors, rather than the latent vectors the generator has been trained to memorize. Figure 9 shows the samples that this generator produces at evaluation time. This generator is clearly not well-behaved: it suffers from mode collapse (i.e. it often reproduces the single four that made up half of its training data) and mode dropping (i.e. even when it produces a novel sample, it usually looks an indistinct four or nine, and seldom looks like any other class). Figure 10 shows the label distribution as measured by a pre-trained classifier, confirming that this generator has a severe missing mode problem. This generator's poor behavior is also confirmed by its scores. Its Classifier Score is 4.95 for memorized z values and 2.22 on non-memorized z values. Its Frechet Distance is 118 for memorized z values and 240 for non-memorized z values.

Figure 11 shows that this poorly-behaved generator nonetheless has a good condition number. Taken in isolation, the trajectory of this generator's condition number would suggest that it belongs in the "good cluster" of Figure 1.

In summary, we demonstrate a generator that is *not* trained with a GAN loss, with conspicuous mode collapse and mode dropping, which is nonetheless well-conditioned. This suggests that the relationship between generator conditioning and generator performance does not hold for all generators, and suggests that it may instead be a property of GAN training dynamics.

Figure 8. Samples from memorized zs. Half of the samples the generator was trained to memorize are identical copies of a single MNIST sample (in this case, a four) and the other half are random MNIST digits. The generator has successfully memorized the z-to-digit associations it was trained to reproduce.

Figure 9. Samples from random zs. The generator's behavior on these z values was not considered at training time. These samples often resemble the single four that made up half its training data, or other four- and nine-like digits. Occasionally, it produces indistinct digits that are not four-like, such as the 3 and the 5 in the bottom row, or indistinct samples that are not digit-like.

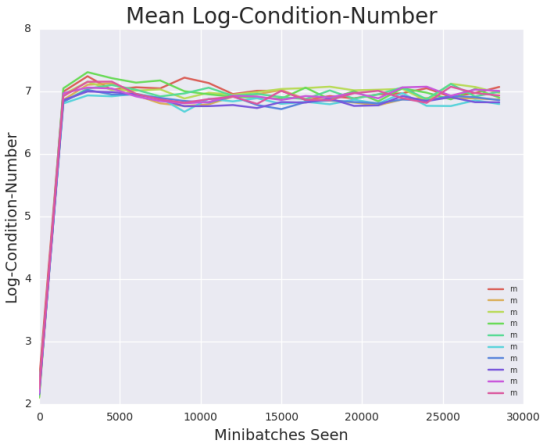

Figure 11. Mean log-condition number of misbehaving generator over 10 runs. Compare to Figure 1 in the main text: this misbehaving generator is better-conditioned than the "good cluster" of GAN generators.

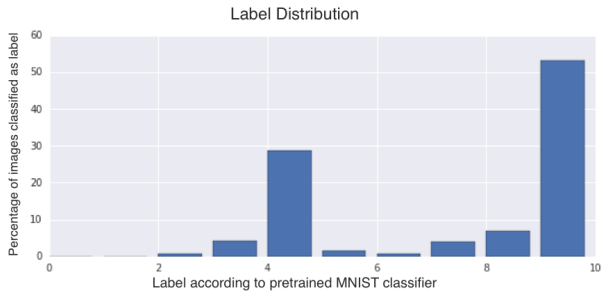

Figure 10. Label distribution of samples from random zs

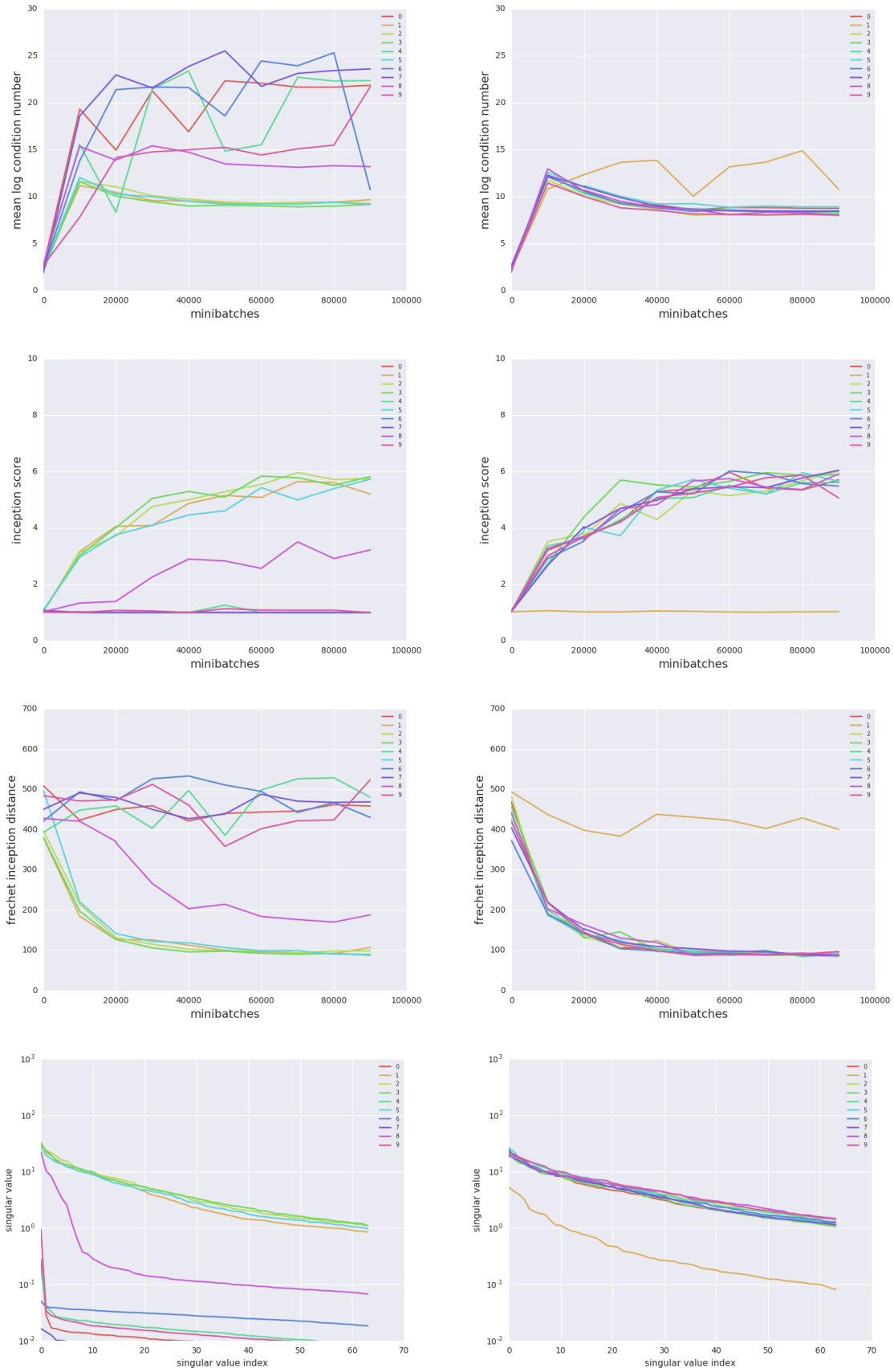

Figure 12. CIFAR10 Experimental results. Left and right columns correspond to 10 runs without and with Jacobian Clamping, respectively. Within each column, each run has a unique color.

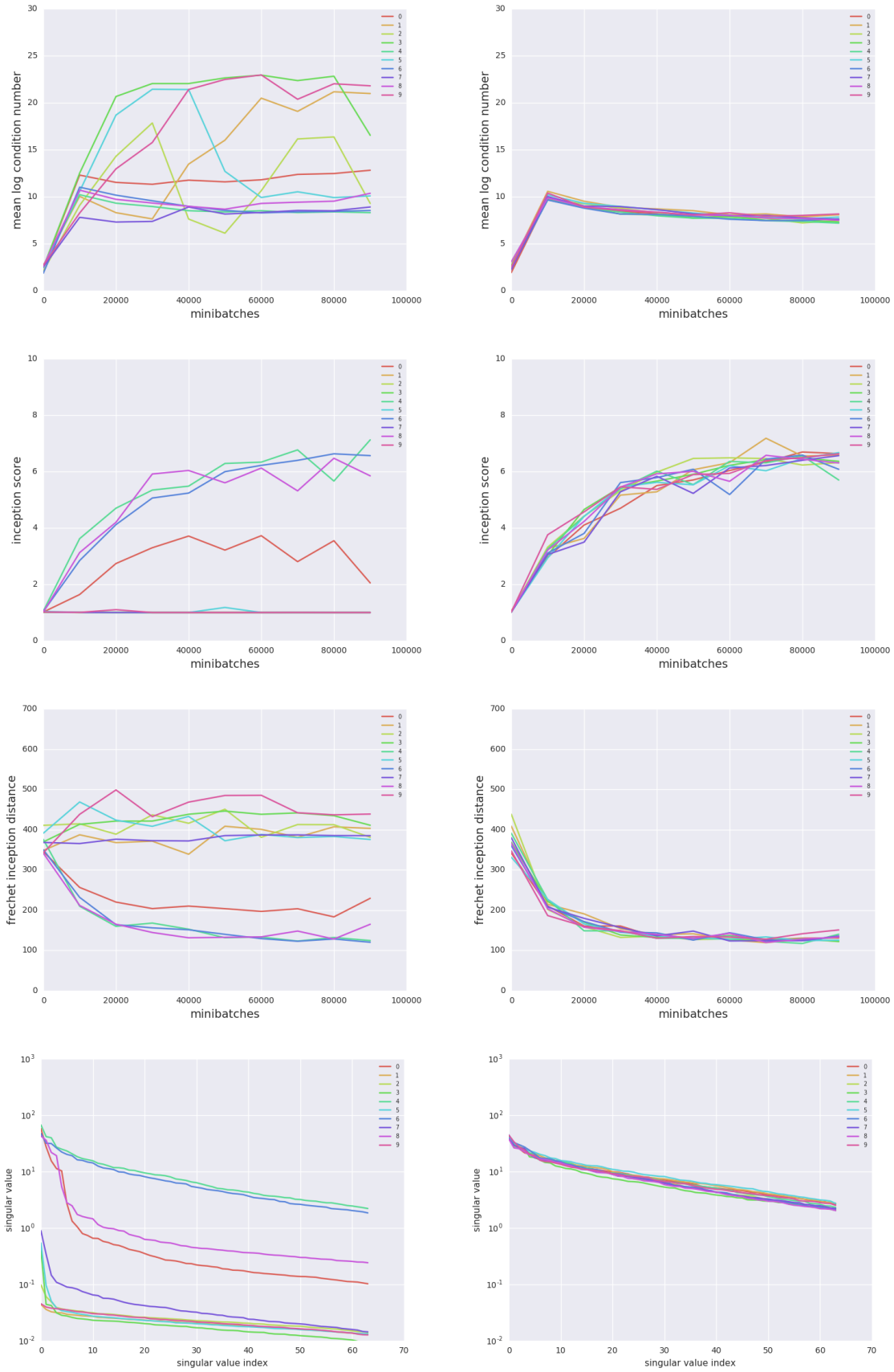

Figure 13. STL10 Experimental Results. Left and right columns correspond to 10 runs without and with Jacobian Clamping, respectively. Within each column, each run has a unique color.