[NTS15] B. Neyshabur, R. Tomioka, and N. Srebro. "Norm-based capacity control in neural networks". In: *Proceedings of the Eigth Annual Conference on Learning Theory*. COLT 2016. 2015, pp. 1376–1401. arXiv: 1503.00036v2 [cs.LG].

[Ris83] J. Rissanen. "A Universal Prior for Integers and Estimation by Minimum Description Length". *Ann. Statist.* 11.2 (June 1983), pp. 416–431.

[TF] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015.

[Zha+17] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. "Understanding deep learning requires rethinking generalization". In: *International Conference on Representation Learning (ICLR)*. 2017. arXiv: 1611.03530v2 [cs.LG].

A APPROXIMATING $\mathrm{KL}^{-1}(Q|C)$

There is no simple formula for $\mathrm{KL}^{-1}(q|c)$, but we can approximate it via root-finding techniques. For all $q \in (0,1)$ and $c \geq 0$, define $h_{q,c}(p) = \mathrm{KL}(q||p) - c$. Then $h'_{q,c}(p) = \frac{1-q}{1-p} - \frac{q}{p}$. Given a sufficiently good initial estimate p_0 of a root of $h_{q,c}(\cdot)$, we can obtain improved estimates of a root via Newton's method:

$$p_{n+1} = \mathrm{N}(p_n; q, c) \text{ where } \mathrm{N}(p; q, c) = p - \frac{h_{q,c}(c)}{h'_{q,c}(p)}.$$

This suggests the following approximation to $\mathrm{KL}^{-1}(q|c)$:

1. Let $\tilde{b} = q + \sqrt{\frac{c}{2}}$.
2. If $\tilde{b} \geq 1$, then return 1.
3. Otherwise, return $\mathrm{N}^k(\tilde{b})$, for some integer $k > 0$.

Our reported results use five steps of Newton's method.

B NETWORK SYMMETRIES

In an ideal world, we would account for all the network symmetries when computing the KL divergence in the PAC-Bayes bound. However, it does not seem to be computationally feasible to account for the symmetries, as we discuss below. Given this, it makes sense to try to break the symmetries somehow. Indeed, one consequence of randomly initializing a neural network's weights is that some symmetries are broken. If we do not expect SGD to reverse (many of) these symmetries, then the initial weight configuration, w_0, will be a better mean for the PAC-Bayes prior P than the origin. In fact, breaking symmetries in this way lead to much better bounds than setting the means to zero.

B.1 BOUNDS FROM MIXTURES

Fix a neural network architecture $H : \mathbb{R}^d \times \mathbb{R}^k \to \{-1, 1\}$ and write h_w for $H(w, \cdot)$. It has long been appreciated that distinct parametrizations $w, w' \in \mathbb{R}^d$ can lead to the same *functions* $h_w = h_{w'}$, and so the set $\mathcal{H} = \{h_w : w \in \mathbb{R}^d\}$ of classifiers defined by a neural network architecture is a quotient space of $\mathbb{R}^d$.

For the purposes of understanding the generalization error of neural networks, we would ideally work directly with $\mathcal{H}$. Let P, Q be a distributions on $\mathbb{R}^d$, i.e., stochastic neural networks. Then P and Q induce distributions on $\mathcal{H}$, which we will denote by $\bar{P}$ and $\bar{Q}$, respectively. For the purposes of the PAC-Bayes bound, it is the KL divergence $\mathrm{KL}(\bar{Q}||\bar{P})$ that upper bounds the performance of the stochastic neural network Q. In general, $\mathrm{KL}(\bar{Q}||\bar{P}) \leq \mathrm{KL}(Q||P)$, but it is difficult in practice to approximate the former because the quotient space is extremely complex.

One potential way to approach $\mathcal{H}$ is to account for symmetries in the parameterization. A *network symmetry* is a map $\sigma : \mathbb{R}^d \to \mathbb{R}^d$ such that, for all $w \in \mathbb{R}^d$, we have $h_w = h_{\sigma(w)}$. As an example of such a symmetry, in a fully connected network with identical activation functions at every unit, the function computed by the network is invariant to permuting the nodes with a hidden layer. Let $\mathbb{S}$ be any finite set of symmetries possessed by the architecture. For every distribution Q on $\mathbb{R}^d$ and network symmetry σ, we may define $Q_\sigma = Q \circ \sigma^{-1}$ to be the distribution over networks obtained by first sampling network parameters from Q and then applying the map σ to obtain a network that computes the same function.

Define $Q^{\mathbb{S}} = \frac{1}{|\mathbb{S}|} \sum_{\sigma \in \mathbb{S}} Q_\sigma$. Informally, Q and $Q^{\mathbb{S}}$ are identical when viewed as distributions on functions, yet $Q^{\mathbb{S}}$ spreads its mass evenly over equivalent parametrizations. In particular, for any data set S, we have $\hat{e}(Q, S) = \hat{e}(Q^{\mathbb{S}}, S)$. We call $Q^{\mathbb{S}}$ a symmetrized version of Q. The following lemma states that symmetrized versions always have smaller KL divergence with respect to distributions that are invariant to symmetrization: Before stating the lemma, recall that the differential entropy of an absolutely continuous distribution Q on $\mathbb{R}^d$ with density q is $\int q(x) \log q(x) \mathrm{d}x \in \mathbb{R} \cup \{-\infty, \infty\}$.

Lemma B.1. *Let $\mathbb{S}$ be a finite set of network symmetries, let P be an absolutely continuous distribution such that $P = P_\sigma$ for all $\sigma \in \mathbb{S}$, and define $Q^{\mathbb{S}}$ as above for some arbitrary absolutely continuous distribution Q on $\mathbb{R}^d$ with finite differential entropy. Then* $\mathrm{KL}(Q^{\mathbb{S}}||P) = \mathrm{KL}(Q||P) - \mathrm{KL}(Q||Q^{\mathbb{S}}) \leq \mathrm{KL}(Q||P)$.

The above lemma can be generalized to distributions over (potentially infinite) sets of network symmetries.

It follows from this lemma that one can do no worse by accounting for symmetries using mixtures, provided that one is comparing to a distribution P that is invariant to those symmetries. In light of the PAC-Bayes theorem, this means that a generalization bound based upon a KL divergence that does not account for symmetries can likely be improved. However, for a finite set $\mathbb{S}$ of symmetries, it is easy to show that the improvement is bounded by $\log |\mathbb{S}|$, which suggests that, in order to obtain appreciable improvements in a numerical bound, one would need to account for an exponential number of symmetries. Unfortunately, exploiting this many symmetries seems intractable. It is hard to obtain useful lower bounds to $\mathrm{KL}(Q||Q^{\mathbb{S}})$, while upper bounds from Jensen's inequality lead to negative (hence vacuous) lower bounds on $\mathrm{KL}(Q^{\mathbb{S}}||P)$.

In this work, we therefore take a different approach to dealing with symmetries. Neural networks are randomly initialized in order to *break* symmetries. Combined with the idea that the learned parameters will reflect these broken symmetries, we choose our prior P to be located at the random initialization, rather than at zero.

C COMPARING WEIGHTS BEFORE AND AFTER PAC-BAYES OPTIMIZATION

In the course of optimizing the PAC-Bayes bound, we allow the mean w to deviate from the SGD solution w_{SGD} that serves as the starting point. This is necessary to obtain bounds as tight as those that we computed. Do the weights change much during optimization of the bound? How would we measure this change?

To answer these questions, we calculated the p-value of the SGD solution under the distribution of the stochastic neural network.

Let Q_{SNN} denote the distribution obtained by optimizing the PAC-Bayes bound, write w_{SNN} and Σ_{SNN} for its mean and covariance, and let $\|w\|_{\Sigma_{\text{SNN}}} = w^T \Sigma_{\text{SNN}}^{-1} w$ denote the induced norm. Using 10000 samples, we estimated

$$\mathop{\mathbb{P}}_{w \sim Q_{\text{SNN}}} \Big(\|w - w_{\text{SNN}}\|_{\Sigma_{\text{SNN}}} < \|w_{\text{SGD}} - w_{\text{SNN}}\|_{\Sigma_{\text{SNN}}} \Big).$$

The estimate was 0 for all true label experiments, i.e., w_{SGD} is less extreme of a perturbation of w_{SNN} than a typical perturbation. For the random-label experiments, w_{SNN} and w_{SGD} differ significantly, which is consistent with the bound being optimized in the face of random labels.

D EVALUATING RADEMACHER ERROR BOUNDS

Fix a class $\mathcal{F}$ of measurable functions from $\mathbb{R}^D$ to $\mathbb{R}$ and let $\mathcal{R}_m(\mathcal{F})$ denote the Rademacher complexity of $\mathcal{F}$ associated with m i.i.d. samples. For $h \in \mathcal{F}$, we will obtain binary classifications (and measure error and empirical error) by computing the sign of its output, i.e., by thresholding. The following error bound is a straightforward adaptation of [BM02, Thm. 7], which is itself an adaptation of [KP02, Thm. 2].

Theorem D.1. *For every $L > 0$, with probability at least $1 - \delta$ over the choice of $S_m \sim \mu^m$, for all $h \in \mathcal{F}$,*

$$e(h) \le \hat{e}(h, S_m, L) + 2L\mathcal{R}_m(\mathcal{F}) + \sqrt{\frac{\log(\frac{2}{\delta})}{2m}}, \qquad (10)$$

where

$$\hat{e}(h, S_m, L) = \frac{1}{m} \sum_{i=1}^{m} \max(\min(1 - L y_i h(x_i), 1), 0).$$

In order to compute these bounds, we must compute (bounds on) the Rademacher complexity of appropriate function classes. To that end, we will use results by Neyshabur, Tomioka, and Srebro [NTS15] for ReLU networks (i.e., multilayer perceptrons with ReLU activations).

Let w be the weights of a ReLU network and let $w_{i,j}^{(k)}$ denote the weight associated with the edge from neuron i in layer $k-1$ to neuron j in layer k. Neyshabur, Tomioka, and Srebro [NTS15] define the ℓ_1 path norm

$$\phi_1(w) = \sum_j \Big[|w_{j,1}^{(2)}| \sum_i |w_{i,j}^{(1)}| \Big], \qquad (11)$$

stated here in the special case of a 2-layer network with 1 output neuron. For any number of layers, the path norm can be computed easily in a forward pass, requiring only a matrix–vector product at each layer.

Neyshabur, Tomioka, and Srebro also provide the follow Rademacher bound in terms of the path norm:

Theorem D.2 ([NTS15, Cor. 7])**.** *Given m datapoints $x_1, \dots, x_m \in \mathbb{R}^D$, the Rademacher complexity of the class of depth-d ReLU networks, whose ℓ_1 path norms are bounded by ϕ, is no greater than*

$$2^d \phi \sqrt{\frac{\log(2D)}{m}} \max_i \|x_i\|_\infty. \qquad (12)$$

Let $w_j^{(k)}$ for the jth column of $w^{(j)}$, i.e., the vector of weights for edges from layer $k-1$ to neuron j in layer k. The ℓ_1 path norm is closely related to the norm

$$\gamma_{1,\infty}(w) = \prod_{i=1}^{d} \max_j \big\|w_j^{(k)}\big\|_1.$$

If the upper bound ϕ appearing in the bound of Theorem D.2 is instead taken to be a bound on $\gamma_{1,\infty}(w)$, then one essentially obtains the Gaussian complexity bounds for neural networks established by Bartlett and Mendelson [BM02] and Koltchinskii and Panchenko [KP02]. However, their bounds apply only to networks with bounded activation functions, ruling out ReLU networks.

Regardless, the path-norm bound is tighter for ReLU networks. In order to establish the connection, let $\mathcal{W}(w)$ denote the set of all weights w' obtained from redistributing the weights w across layers, i.e., by multiplying the weights $w^{(k-1)}$ in a layer by a constant $c > 0$ and multiplying the weights in the subsequent layer $w^{(k)}$ by c^{-1}. Note that the function computed by a ReLU network is invariant to this transformation. This is the key insight of Neyshabur, Tomioka, and Srebro. Obviously, $\phi_1(w) = \phi_1(w')$ for all $w' \in \mathcal{W}(w)$. Neyshabur, Tomioka, and Srebro show that $\phi_1(w) = \inf_{w' \in \mathcal{W}(w)} \gamma_{1,\infty}(w')$, and so the path norm better captures the complexity of a ReLU network.

In our experiments, we will compute the bound obtained by combining Theorems D.1 and D.2.

Note that the constant L in Theorem D.1 must be chosen independently of the data S_m. As in the original result [KP02, Thm. 2], one can use a union bound to allow oneself to choose L based on the data in order to minimize the bound. Even though the effect of this change is usually (relatively) small, its magnitude depends on the particular weight function employed in the union bound. Instead, we will apply the bound with an optimized L, yielding an optimistic bound (formally, a lower bound on any upper bound obtained from a union bound). We optimize L over a grid of values, and handle the vacuous edge cases analytically. Nevertheless, we will see even the resulting (optimistic) bound is vacuous.

D.1 EXPERIMENT DETAILS

We use SGD to train a two-layer 600-hidden-unit ReLU network on the same binary class variant of MNIST used to evaluate our PAC-Bayes bounds. We set the global learning rate to 0.005. As in our PAC-Bayes experiments, we optimize the average logistic loss during training. The random initializations commonly used for ReLU networks lead to initial path norms that produce vacuous error bounds. In order to visualize the behavior of the path-norm bound under SGD, we reduce the standard deviation of the truncated-normal initialization from 0.04 to 0.0001. As before, we use mini-batches of 100 training examples, yielding 550 iterations per epoch.

For comparison, we also train the same network architecture while explicitly regularizing the path norm. (Neyshabur, Salakhutdinov, and Srebro [NSS15] propose training neural networks via steepest descent with respect to the path norm. We leave this comparison to future work.)

D.2 RESULTS

When the network is trained by optimizing the logistic cost function without regularization, the error bound becomes vacuous within a fraction of a single epoch. This occurs before the training error dips appreciable below chance. The bound's behavior is due to the path norm diverging. While the level sets $\hat{e}(h, S, \cdot)^{-1}$ of the empirical margin distribution are growing, they are not growing fast enough to counteract the growth of the path norm. (See the left column of Fig. 1.)

When the network is trained with explicit path-norm regularization, we obtain vacuous error bounds, unless we apply excessive amounts of regularization. We report results when the regularization parameter is 0.01 and 0.05. Both settings are clearly too large, as evidenced by the training error converging to ~20% and ~30%, respectively. A cursory study of overall ℓ_1 and ℓ_2 regularization produced qualitatively similar results. Further study is necessary.

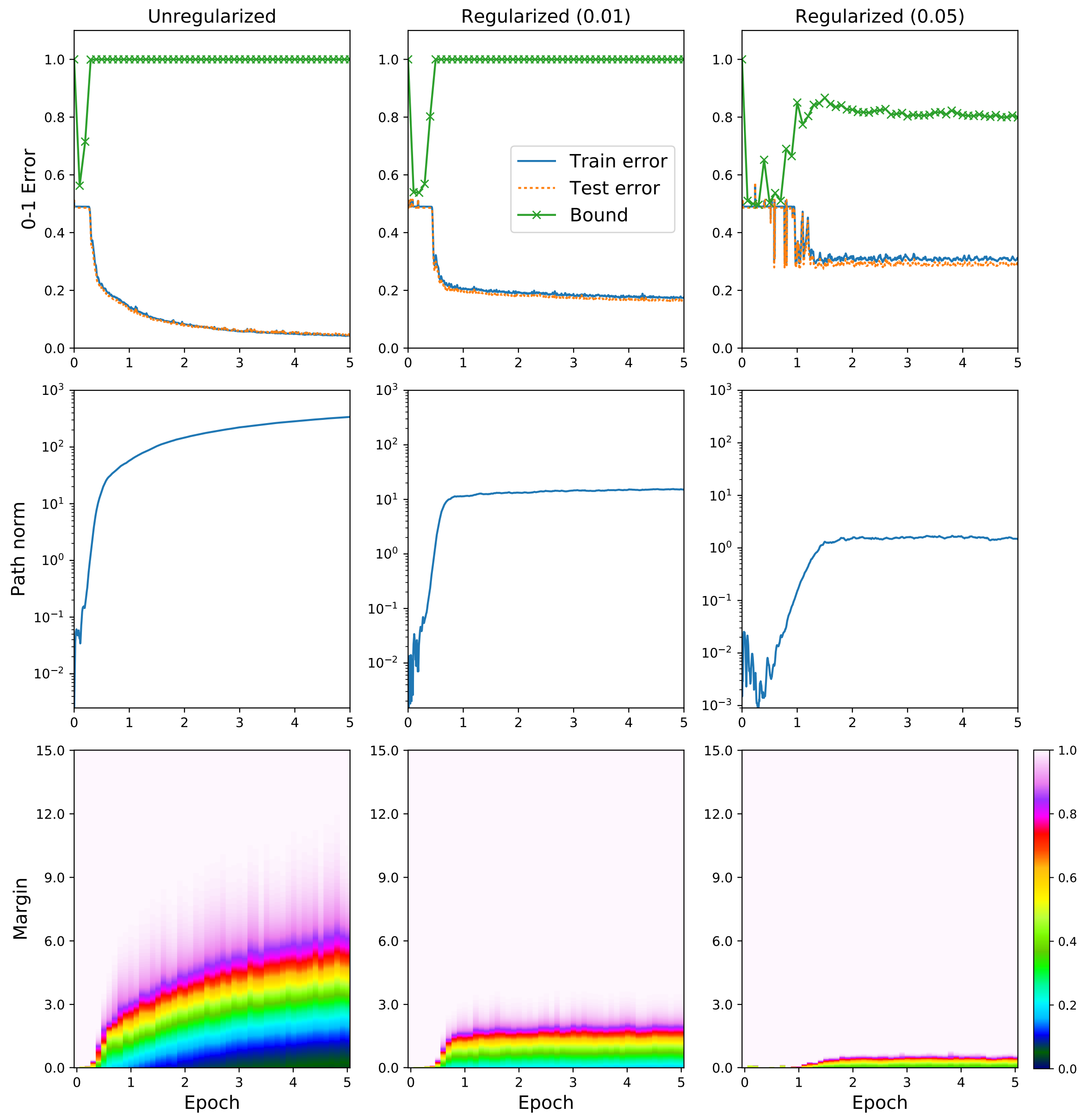

Figure 1: Unregularized (**left column**) and path-norm regularized (**center and right columns** with regularization parameter specified in parenthesis) optimization of two-layer 600-hidden-unit ReLU network by SGD for 5 epochs. (We ran 20 epochs and found no new patterns. Plots for longer experiment obscured the initial behavior.) (**top row**) Training error, testing error, and error bounds versus (iterations measured in) epochs. Without regularization, the bounds are immediately vacuous once the network performance deviates from chance, and this remains true under regularization unless the explicit regularization is very strong. In this case, the bound is nonvacuous, but trivial in the sense that the error rate of guessing is 50%. Note that training/testing error is also very large in this case. (**center row**) Log plot of path norm versus epochs. Without regularization, the path norm diverges quickly. (**bottom row**) Empirical margin distributions versus epochs. The margin that attains a fixed average loss is growing, but not rapidly enough to counteract the rapidly increasing path norm.