Generalization Bounds for Neural Networks via Sparsity-Inducing Priors

Noah MacAulay

January 15, 2019

1 Introduction

In recent years there has been an explosion of interest in deep neural networks. Neural networks have found applications in virtually every field of machine learning, including image recognition, natural language processing, speech recognition, object manipulation, and game playing. In many of these fields, neural-net-based programs have dramatically outperformed the state of the art.

In comparison, progress has been relatively slow in the theory of neural networks – understanding why, and under what circumstances, neural nets perform so well. The functions learned by neural networks are hugely complex, often requiring millions or even billions of parameters to specify. These highly over-parametrized functions have proven challenging for traditional machine learning theory.

This was illustrated by the work of [14]. They showed that the traditional bounds of statistical learning theory could not account for the ability of neural networks to generalize from training to the test set. The learning theory bounds depend on the number of parameters needed to specify a model; applied to modern neural networks they are vacuous, providing no bound at all. Despite this, neural networks have been found to generalize well across a wide range of tasks. Practice has outstripped theory.

In response, the past two years have seen the publication of the first non-vacuous generalization bounds for deep neural networks[2]. These works provided provable guarantees on the performance of neural networks on the test set in terms of their performance on the training set. This was achieved using

the PAC-Bayes theory[10], in which individual classifiers are replaced with probability distributions over a space of classifiers. Generalization bounds can then be given in terms of the KL-Divergence between these probability distributions and a fixed prior distribution. Dzuigaite and Roy[2] used PAC-Bayes bounds to obtain the first nonvacuous generalizaton bounds on neural nets with hundreds of thousands of parameters. Dzuigaite and Roy used for their prior a Normal distribution centred at the **initial** weights of the neural net, before training has taken place. This choice of prior can be seen as a hypothesis about the types of solutions learned by stochastic gradient descent(SGD) – namely, that they are often near the initial parameters, or can adjusted to be so without affecting performance.

Another interesting line of work has considered compression of neural networks. Researchers have found that the vast majority of parameters in some neural networks can be eliminated with no drop in performance[4][11]. The PAC-Bayesian framework shows that compressibility is linked to generalization; Zhou et al.[15] used PAC-Bayes bounds combined with compression techniques to obtain nonvacuous bounds on a neural network with millions of parameters trained on ImageNet, a challenging image-recognition task.

In this paper I will extend work on PAC-Bayes bounds and compression by considering a variational Bayesian framework with a broader class of priors. Previously in [2], independent normal distributions have been used for both the prior and posterior of the PAC-Bayes bounds. Here I will consider sparsity-inducing priors, distributions which favour posteriors in which many components are concentrated around zero. Such priors have been used in [8] to compress neural networks. These priors exploit the compressibility of neural networks, because nets in which most weights are near zero have low KL-Divergence. Since neural networks are often compressible in practice, it should be possible to obtain posteriors with low KL-Divergence and hence non-trivial generalization bounds. This choice of prior can also be thought of as a hypothesis about the types of solutions learned by SGD – that they are highly amenable to pruning, that is eliminating many weights from the model.

2 PAC-Bayes bounds

Here I will give a brief overview of the PAC-Bayes framework, following [9]. The PAC-Bayes framework can be applied in any context where a learning

algorithm is assigned a cost function based on its output. We have a hypothesis class H, and a distribution D over a set of 'situations' S. For each hypothesis $h \in H$ and situation $s \in S$, there is assigned as loss $L(h, s)$. In the context of image classification, a 'situation' would be an (image, label) pair, h would be a classifier, and $L(h, s)$ the loss of that classifier on the given pair. The overall loss of a given classifier, $L(h)$ is defined as the expected loss over the distribution D. Given a sample from D with N elements, we define the empirical risk $\hat{L}(h) = \frac{1}{N}\sum_{i=1}^{N} L(h, s_i)$ as the expected loss over the sample. To incorporate Bayesian methods, we now introduce distributions P and Q over the hypothesis class. P is a fixed prior distribution, while Q can be learned from the sample. The losses $L(Q)$, $\hat{L}(Q)$ are respectively the expected loss and empirical risk(loss on the training sample) of Q. Essentially, we can obtain valid bounds if the learned posterior Q can be specified compactly with P. To measure to what extent Q is close to P, we introduce the Kullback-Liebler divergence(KL Divergence) between two distributions P and Q:

$$KL(Q||P) = \int_H Q(h) log\left(\frac{Q(h)}{P(h)}\right) dh$$

Intuitively, this measures the number of bits needed to specify a random sample from Q in terms of P. In the case of a discrete hypothesis space, this measure becomes $\sum_h Q(h) log\left(\frac{Q(h)}{P(h)}\right)$.

We will also need the Bernoulli distribution in what follows. The Bernoulli distribution is defined as a distribution over a binary outcome. $Ber(p)$ will denote a Bernoulli distribution with probabilities p and $1 - p$. Given this setup, we have the following theorem(from [9]):

Theorem 1 *For fixed $\lambda > \frac{1}{2}$ selected before the draw of the sample, we have that, with probability at least $1 - \delta$, the following holds simultaneously for all distributions Q on H.*

$$KL(\hat{L}(Q)||L(Q)) \leq \frac{KL(Q||P) + log(\frac{N}{\delta})}{N - 1} \tag{1}$$

In a slight abuse of notation, we here define $KL(a||b) = KL(Ber(a)||Ber(b)) = alog(a/b) + (1 - a)log((1 - a)/(1 - b))$ for scalar a, b. This theorem bounds $KL(\hat{L}(Q)||L(Q))$ by a constant c, but we actually want a bound on $L(Q)$. So we want to find the maximum value p such that $KL(\hat{L}(Q)||p) \leq c$. There is

no simple formula for finding such a p, but, following [2], we can use Newton's method to find an approximately optimal p.

For the purposes of gradient-based optimization, we need a differentiable proxy for the upper bound on $L(Q)$. To do this we can use the fact that $KL(p||q) \geq (p-q)^2$ and a little algebra to obtain the following upper bound:

$$L(Q) \leq \hat{L}(Q) + \sqrt{\frac{1}{2}\frac{KL(Q||P) + log(\frac{N}{\delta})}{N-1}} \tag{2}$$

This is the bound we will optimize using gradient descent. To allow for gradient-based optimization we must replace the empirical risk $\hat{L}(Q)$ with a differentiable proxy. We will use the cross-entropy loss, which is $-log(p_{true})$, where p_{true} is the probability our model assigns to the true class of an instance.

3 Bayesian Neural Networks

In Bayesian neural networks, our model incorporates uncertainty about the values of the weights of the network. One of the simplest models of such uncertainty is to simply introduce an independent distribution for each parameter in the network. These distributions can themselves be parameterized, and these parameters can be updated using gradient descent to optimize the performance of the randomized neural network on a given task. One of the most commonly used models assigns a random Gaussian to each weight in the network: $q(w_{i,j}) = N(u_{i,j}, \sigma^2_{i,j})$. This distribution can be easily sampled from, and it can also be efficiently used in batch processing as shown in the next section.

3.1 Local Reparameterization

The local reparameterization trick[7] is a method for speeding the evaluation and reducing the variance of neural networks with Gaussian posteriors on their weights. It uses the fact that a linear combination of Gaussians also has a Gaussian distribution.

Suppose we're processing a mini-batch of size M in a neural network with Gaussian posteriors on our weights. If we have a $W \times H$ layer, then naively we need to take $N \times W \times H$ samples from our Gaussian posteriors. But

this can be avoided, as the sum of Gaussians is itself a Gaussian. We can instead take samples from the implied Gaussian distribution on the output of the layer, needing only $N \times W$ samples. Representing the activations of the inputs of the layer by $a_{m,i}$, the pre-activations of the outputs by $b_{m,j}$, and the weights by $w_{i,j}$, we have:

$$q(w_{i,j}) = N(u_{i,j}, \sigma_{i,j}^2) \rightarrow q(b_{m,j}) = N(\eta_{m,j}, \delta_{m,j})$$

where

$$\eta_{m,j} = \sum a_{m,i} u_{i,j}$$

$$\delta_{m,j} = \sum a_{m,i}^2 \sigma_{i,j}^2$$

This allows us to evaluate the layer more quickly, as we can take advantage of fast GPU matrix multiplication. It also reduces the variance of our gradient estimator.

4 Sparsity-Inducing Priors

4.1 Horseshoe Prior

The horseshoe prior was introduced in [1]. It has the following hierarchical form:

$$\tau \sim C^+(0, \sigma)$$

$$\lambda_i | \tau \sim C^+(0, \tau)$$

$$\theta_i | \lambda_i \sim N(0, \lambda_i^2)$$

Here, $C^+(0, \sigma)$ is the Half-Cauchy distribution $C^+(0, \sigma) = 2(\sigma\pi(1 + (z/s)^2))^{-1}$. $X \sim Y$ means X is a random variable distributed according to Y. The variables modelling the data we are interested in are the θ_i's.

The horseshoe prior is useful for a wide variety of sparse inference problems. This is mainly due to its dual nature: it has heavy Cauchy-like tails decaying as θ^{-2}, yet near the origin it is very tightly concentrated around 0, and in fact the density becomes unbounded near the origin. An interesting feature of the horseshoe prior is that the variables λ_i are independently distributed for each variable in the distribution, but τ is shared among all variables. τ provides a 'global scale' for the distribution.

The horseshoe prior has been used for variational inference and compression in neural networks. In [8], it was used for group sparsity, that is, sparsity at the level of neurons instead of weights. This allows the compressed network to be run more efficiently. Compression rates of 800× were obtained, and evaluation speed improved by a factor of 2. It is interesting to note that the amount of compression is much greater than the obtained speedup – the reason for this is that GPU kernels for matrix multiplication are optimized for dense matrices, so obtaining a much smaller sparse matrix does not greatly improve evaluation speed.

4.2 Log-uniform prior

The log-uniform 'prior' is not really a prior at all, as its integral over the real line diverges. Nevertheless, it can serve in the role of a prior in variational inference and inducing sparsity. It has the following form:

$$p(\theta) = \frac{1}{|\theta|}$$

or equivalently

$$p(log(\theta)) = c$$

Here c is some constant(the constant chosen does not matter, as it does not affect the relative probability of different values of θ). The log-uniform prior has been used for variational inference and compression in neural nets. In [7] it was used as a prior for a neural net with Gaussian weights. They showed that this prior is the only one consistent with dropout, a commonly used technique for regularizing neural nets. (That is, optimizing a Gaussian neural net with this prior imitates the effects of dropout). Their work was extended in [11], who showed that an extension of this technique naturally sparsifies neural networks. They were able to reduce the number of parameters in neural networks by a factor of 280 using this method.

4.3 Lower Bounds for Mixture Priors

While the preceding priors have been successfully been used for neural network compression in the past, they are problematic for the purposes of optimizing PAC-Bayes bounds. The reason for this is that deep neural networks

contain hundreds of thousands of parameters. To obtain non-trivial bounds, we need the KL Divergence to be much smaller than the number of training examples(e.g. MNIST has 50000 training examples). In a factorized variational posterior, the KL Divergence is the sum of the KL Divergence of the posterior of each parameter, so the KL Divergence of most parameters must be very close to zero.

Unfortunately, many sparsity-inducing priors cannot have low KL-Divergence with a Gaussian posterior. No matter what parameters of the Gaussian we choose, the KL Divergence will be large, simply because the priors don't resemble a Gaussian at any point in their range. This applies to the two priors we discussed above(in fact the Log-uniform prior is not even normalized so it can't be used for KL-Divergences in any case), as well as to other sparsity-inducing priors such as the double-exponential prior.

To deal with this, we'll consider a restricted class of priors: scale mixtures of Gaussians. These are mixtures of Gaussian distributions with mean zero and variance λ distributed according to a fixed prior. We can ensure that such distributions will have low KL-Divergence with at least some Normal posteriors by concentrating the density of λ near a single value. Such distributions can be written as:

$$p(x) = \int_\lambda h(\lambda) d\lambda N[0, \lambda^2](x)$$

where $N[0, \lambda^2]$ is the density function of a normal distribution with mean 0 and variance λ^2, and $h(\lambda)$ is the probability density function over the scale.

The KL divergence between such a distribution and an arbitrary posterior can be bounded as follows:

$$KL(q||p) = \int_x dx q(x) log\left(\frac{q(x)}{p(x)}\right)$$

$$= \int_x dx q(x) log(q((x)) - \int_x dx q(x) log\left(\int_\lambda h(\lambda) d\lambda N[0, \lambda^2](x)\right)$$

$$\leq \int_x dx q(x) log(q((x)) - \int_x dx q(x) \int_\lambda h(\lambda) d\lambda log(N[0, \lambda^2](x))$$

$$= \int_\lambda d\lambda h(\lambda) \int_x dx q(x) log\left(\frac{q(x)}{N[0, \lambda^2](x)}\right)$$

$$= \mathop{\mathbb{E}}_{h(\lambda)} KL(q||N[0, \lambda^2])$$

This last expression is the expected KL Divergence of our posterior with a random Gaussian chosen from the mixture. We will use this to upper-bound the divergence in a tractable way.

4.4 Variational Upper Bound for Discrete Mixture Prior

Here we will consider a particularly simple mixture prior: a discrete mixture over two values of λ. This is similar to a 'spike-and-slab' distribution.

We have two values of λ, λ_1 and λ_2 with prior probabilities p_1 and p_2, respectively. To better fit posteriors with different means, we will employ a variational posterior, that is we will introduce auxilliary probabilities q_1, q_2. Our total upper bound will then be

$$KL_{upper} = \mathop{\mathbb{E}}_{Ber(q_1,q_2)} KL(q||N[0,\lambda_i^2)) + KL(Ber(q_1,q_2)||Ber(p_1,p_2))$$

Here $Ber(p_1, p_2)$ denotes a Bernoulli distribution with probabilities p_1 and p_2. Expanding this out, this gives:

$$KL_{upper} = q_1 KL(q||N[0,\lambda_1^2]) + q_2 KL(q||N[0,\lambda_2^2]) + q_1 log(\frac{q_1}{p_1}) + q_2 log(\frac{q_2}{p_2})$$

$$= q_1(log(\frac{q_1}{p_1}) + KL(q||N[0,\lambda_1^2])) + (1-q_1)(log(\frac{(1-q_1)}{p_2}) + KL(q||N[0,\lambda_2^2]))$$

The last line follows as $q_1 + q_2 = 1$. For fixed values of p and λ, this function is convex in q_1, so its minimum can be found when the derivative is zero:

$$KL'_{upper}(q_1) = (log(\frac{q_1}{p_1}) + KL(q||N[0,\lambda_1^2])) - (log(\frac{1-q_1}{p_2}) + KL(q||N[0,\lambda_2^2])$$

When this is zero, we have:

$$log(\frac{q_1}{p_1}) + KL(q||N[0,\lambda_1^2]) = log(\frac{1-q_1}{p_2}) + KL(q||N[0,\lambda_2^2])$$

$$log(\frac{q_1}{1-q_1}) = KL(q||N[0,\lambda_2^2]) - KL(q||N[0,\lambda_1^2]) + log(\frac{p_1}{p_2})$$

$$\frac{1}{q_1^{-1}-1} = exp(KL([q||N[0,\lambda_2^2]) - KL(q||N[0,\lambda_1^2]) + log(\frac{p_1}{p_2}))$$

Letting $y = KL(q||N[0,\lambda_2^2]) - KL(q||N[0,\lambda_1^2]) + log(\frac{p_1}{p_2})$, we have that $q_1 = \frac{1}{1+exp(-y)}$, i.e. q_1 is a sigmoid of y. Since the sigmoid operation is differentiable, we can set q_1 to its optimal value during training and optimize using gradient descent.

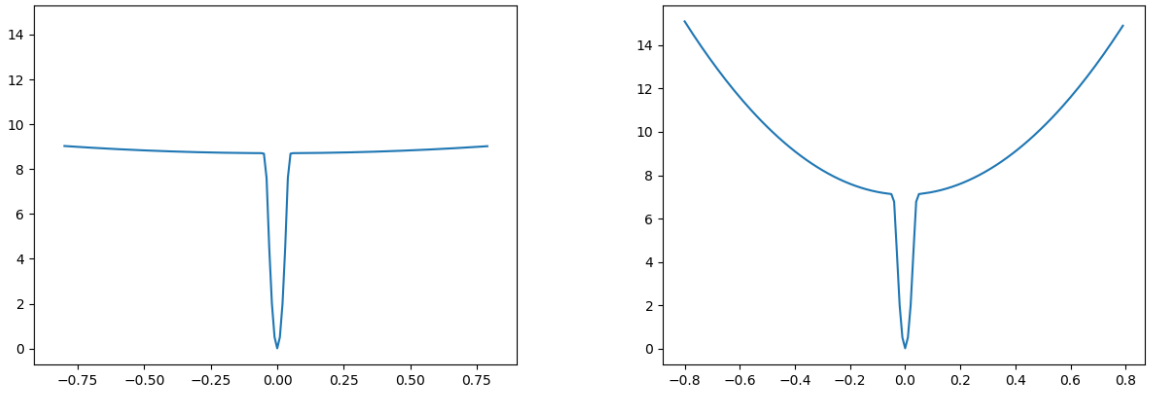

Figure 1: The KL Divergence of posteriors with different means, for two priors. x-axis tracks the mean posterior, and the y shows our variational upper bound on the KL Divergence. Both priors have $\lambda_1 = 0.01$ and $p_1 = 0.99$, the left has $\lambda_2 = 0.2$ and the right has $\lambda_2 = 1$. The posterior variance is equal to λ_1

4.5 Gradients of Discrete Mixture Prior

The discrete mixture prior assigns very low KL Divergence to Gaussian posteriors near zero, while not penalizing the posteriors that are away from zero by too much. However, there is a trade-off involving the size of λ_2, the 'slab' proportion of the mix. If we increase λ_2, we can decrease the KL Divergence of parameters far from zero – however, this comes at the cost of reducing the gradients far from zero, making it less likely for a parameter to be pushed to zero by gradient descent. The problem essentially arises because the KL Divergence is highly non-convex. We illustrate this in figure 1, which shows the effect of varying the posterior mean on the KL Divergence. In the right panel, $\lambda_2 = 0.2$, while in the left panel $\lambda_2 = 1$. As can be seen, setting $\lambda_2 = 1$ decreases the value of the KL Divergence for large values of the mean, but suffers from small gradients. This will result in suboptimal levels of sparsity.

To address this, we'll perform optimization in two stages. First we'll optimize with a small value of λ_2 to encourage sparsity. Then we'll switch to a larger value of λ_2, decreasing the KL Divergence for posteriors far from zero while leaving the posteriors near zero unaffected.

5 Experiments

5.1 Two-Class MNIST

We ran our discrete mixture model on a modified version of MNIST. Each of the classes 0, 1, 2, 3, 4 are assigned class 0, and otherwise class 1. We calculated the PAC-Bayes error bound (2), using the cross-entropy loss as a proxy for the empirical risk. Training was done using Adam[6].

The network architecture was fully-connected with two hidden layers with dimensions 300 and 100. ReLU nonlinearities were used. The network was pre-trained by SGD(Stochastic Gradient Descent) for 20 epochs to a good optimum on the original task, then the PAC-Bayes bound was optimized using Adam with a learning rate of 10^{-3}. To avoid numerical instability, we decomposed the KL divergence upper bound as $KL_{upper} = q_1(log(\sigma(y)) - log(p_1)) + q_2(log(\sigma(-y)) - log(p_2)) + q_1 KL(q||N[0, \lambda_1^2]) + q_2 KL(q||N[0, \lambda_2^2])$ and used the logsigmoid function from the pytorch[12] library, which numerically stabilizes the gradient of $\log(\sigma(y))$.

To choose the parameters for the variational prior, we used the following strategy. Letting p_1, p_2, λ_1, λ_2 denote the parameters mentioned in section 4.3, we always set $p_1 = 0.99$ and $\lambda_1 = 0.01$, to encourage most weights to cluster around zero. For λ_2, we want to choose a value which will allow us to efficiently encode the nonzero weights in each layer. A good prior for this purpose would match the distribution of weight magnitudes found in a typical layer. The most popular initialization schemes[3][5] for neural network weights use a Gaussian with variance proportional to $\frac{1}{\sqrt{\#inputs}}$(this was the initialization we used for pre-training). During the initial training we want to strongly encourage sparsity so we set $\lambda_2 = \frac{1}{2\sqrt{\#inputs}}$. After 10000 steps we switch λ_2 to $\frac{4}{\sqrt{\#inputs}}$.

The network was trained for 20000 steps with a mini-batch size of 100. After training was complete, we evaluated the final accuracy on the training set and KL Divergence, and calculated our final bound by using Newton's method to approximate the bound (1).

Network	Generalization Bound	Train Accuracy	Test Accuracy	KL/N
FC-300-100	0.768	0.952	0.961	0.129

Interestingly the test accuracy is sometimes greater than the training accuracy. Perhaps optimizing the PAC-Bayes bound acts as a strong form of

regularization.

This is a nonvacuous bound, although not as good as the bound reported in [2]. We think it is of independent interest, however, as it uses a prior centred at zero instead of the initial weights. It relies upon a different sort of regularity found in the training of neural networks, namely the prunability of the learned nets.

To examine the extent to which our prior encourages sparsity in the weights, figure 2 shows a histogram of the learned posterior means, before and after optimizing the upper bound.

6 Discussion

Different PAC-Bayes bounds can be thought of as different hypotheses about the types of solutions found by SGD. The bound in [2] relied upon the fact that the solutions found by SGD are often reasonably close to the initial pre-trained weights. The bound in [15] instead used the compressibility of the network, leveraging the fact that the weights neural networks can be pruned, compressed and stored highly efficiently. The bounds in this paper apply a similar strategy, relying on the prunability of learned networks. Compared to [15], we are working in a more purely Bayesian framework, explicitly using a prior over weights instead of the implicit prior implied by a compression scheme. Another difference is that the present paper does not exploit the fact that non-zero weights can be further compressed using quantization and Lempel-Ziv coding. It would be interesting to see if improved PAC-Bayes bounds could be obtained using a soft-weight-sharing type prior, which encourages quantization. [13]

References

[1] Carlos Carvalho, Nicholas Polson, and James Scott. The horseshoe estimator for sparse signals. *Biometrika*, 2010.

[2] Gintare Karolina Dziugaite and Daniel M Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

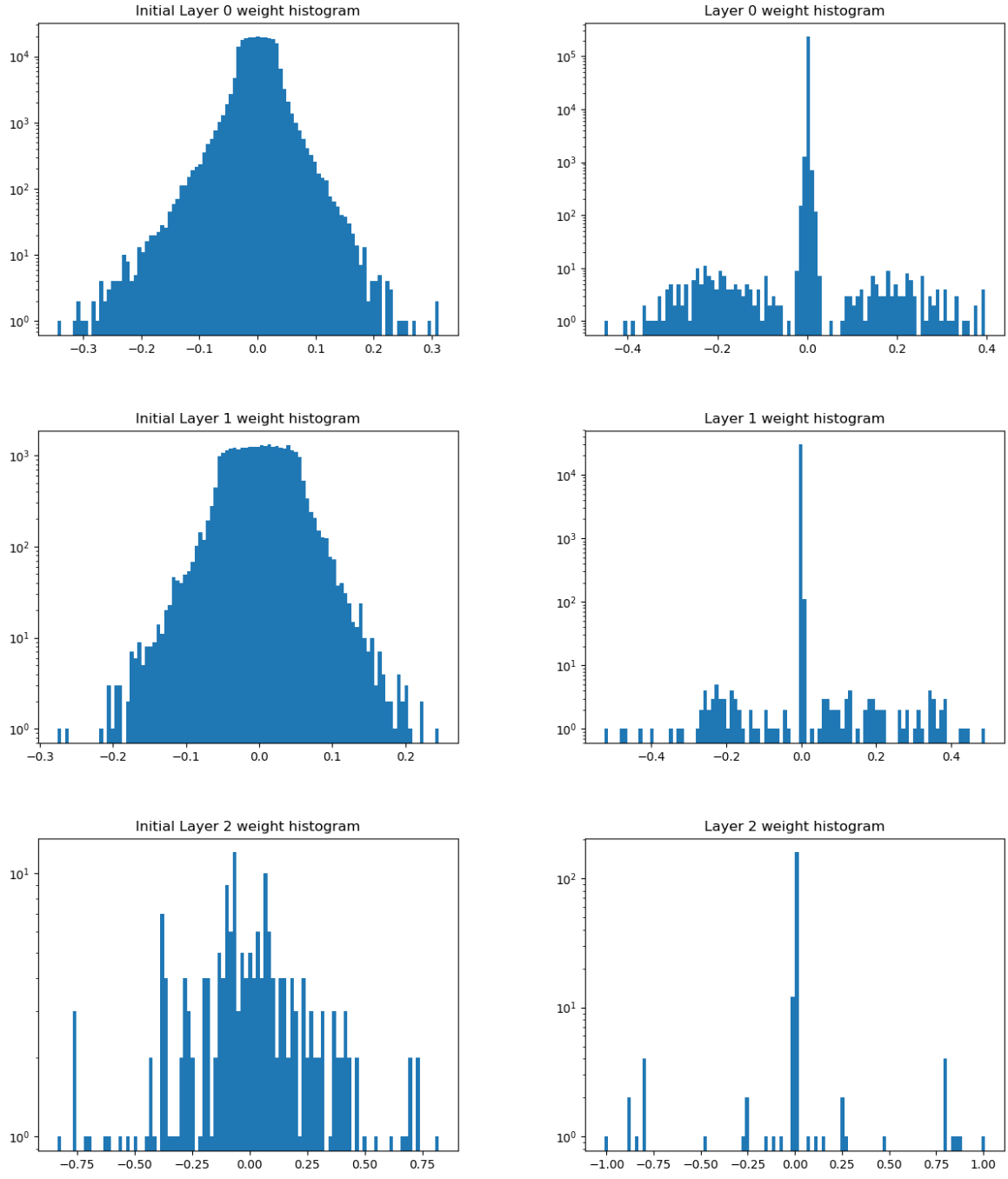

Figure 2: Distribution of weight means before and after optimizing PAC-Bayes bound

[3] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[4] Yiwen Guo, Anbang Yao, and Yurong Chen. Dynamic network surgery for efficient dnns. In *Advances In Neural Information Processing Systems*, pages 1379–1387, 2016.

[5] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[6] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[7] Diederik P Kingma, Tim Salimans, and Max Welling. Variational dropout and the local reparameterization trick. In *Advances in Neural Information Processing Systems*, pages 2575–2583, 2015.

[8] Christos Louizos, Karen Ullrich, and Max Welling. Bayesian compression for deep learning. In *Advances in Neural Information Processing Systems*, pages 3288–3298, 2017.

[9] David McAllester. Simplified pac-bayesian margin bounds. In *Learning theory and Kernel machines*, pages 203–215. Springer, 2003.

[10] David McAllester. A pac-bayesian tutorial with a dropout bound. *arXiv preprint arXiv:1307.2118*, 2013.

[11] Dmitry Molchanov, Arsenii Ashukha, and Dmitry Vetrov. Variational dropout sparsifies deep neural networks. *arXiv preprint arXiv:1701.05369*, 2017.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in pytorch. 2017.

[13] Karen Ullrich, Edward Meeds, and Max Welling. Soft weight-sharing for neural network compression. *arXiv preprint arXiv:1702.04008*, 2017.

[14] Chiyuan Zhang, Samy Bengio, Moritz Hardt, Benjamin Recht, and Oriol Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[15] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P Adams, and Peter Orbanz. Compressibility and generalization in large-scale deep learning. *arXiv preprint arXiv:1804.05862*, 2018.