A Guide to Probably Approximately Correct Bounds for Neural Networks

Thomas Walker

Supervised by Professor Alessio Lomuscio

Summer 2023

Contents

1 Introduction

A great resource for introducing the field of Probably Approximately Correct (PAC) learning theory is given in [15]. It details the progression of results in the field and motivates the various research avenues. PAC learning theory is a general framework for studying learning algorithms, and my aim here is to illustrate how this theory is being contextualized within machine learning, with a specific focus on neural networks. With this report, I want to introduce the theory and detail some applications, as well as provide some recent extensions. The main product of PAC learning theory is bounds on the performance of the output of learning algorithms, termed PAC bounds. This report will not provide an exhaustive list of the various PAC bounds being applied to neural networks. I will instead provide some well-known results in the literature and how some of them manifest in applications. For a comprehensive introduction to the field of PAC, the reader is recommended to refer to [15]. Nevertheless, this report will be mostly self-contained, with proofs for the major results and elaboration on the implementations of PAC theory.

2 PAC Bounds

2.1 Introducing PAC Bounds

2.1.1 Notation

We will first introduce some basic notation that is for the most part consistent with [15] and will remain constant throughout the report. Along the way, we will need to introduce some more specialized notation for the different sections. The problems we will concern ourselves most with will be supervised classification tasks. This means, we have a feature space $\mathcal{X}$ and a label space $\mathcal{Y}$ which combine to form the data space $\mathcal{Z} = \mathcal{X} \times \mathcal{Y}$ for which some unknown $\mathcal{D}$ is defined on. The challenge now is to learn a classifier $h : \mathcal{X} \to \mathcal{Y}$ that correctly labels samples from $\mathcal{X}$ according to $\mathcal{D}$. The training data $S = \{(x_i, y_i)\}_{i=1}^m$ consists of m $\mathrm{i.i.d}$ samples from $\mathcal{D}$. As we are considering neural networks, a classifier will be parameterized by a weight vector $\mathbf{w}$ which we will denote $h_{\mathbf{w}}$. Let $\mathcal{W}$ denote the set of possible weights for a classifier and the set of all possible classifiers $\mathcal{H}$ will sometimes be referred to as the hypothesis set. We will often denote the set of probability distributions over $\mathcal{W}$ as $\mathcal{M}(\mathcal{W})$. To assess the quality of a classifier we define a measurable function $l : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$ called the loss function and we will assume that $0 \leq l \leq C$. As our training data is just a sample from the underlying (unknown) distribution $\mathcal{D}$ there is the possibility that our classifier performs well on the training data, but performs poorly on the true distribution. Let the risk of our classifier be defined as

$$R(h_{\mathbf{w}}) = \mathbb{E}_{(x,y)\sim\mathcal{D}} \left(l(h(x), y) \right).$$

As our classifier is parameterized $\mathbf{w}$ we will instead write $R(\mathbf{w})$ for the risk of our classifier. Similarly, we define the empirical risk of our classifier to be

$$\hat{R}(\mathbf{w}) = \frac{1}{m} \sum_{i=1}^{m} l(h_{\mathbf{w}}(x_i), y_i).$$

Note that $\mathbb{E}_{S\sim\mathcal{D}^m} \left(\hat{R}(\mathbf{w}) \right) = R(\mathbf{w})$.

2.1.2 PAC Bounds

PAC bounds refer to a general class of bounds on the performance of a learned classifier. They aim to determine with high probability what the performance of a classifier will be like on the distribution $\mathcal{D}$ when trained on some training data taken from this distribution.

Theorem 2.1 ([15]). *Let $|\mathcal{W}| = M < \infty$, $\delta \in (0, 1)$, and $\mathbf{w} \in \mathcal{W}$ then it follows that*

$$\mathbb{P}_{S\sim\mathcal{D}^m} \left(R(\mathbf{w}) \leq \hat{R}(\mathbf{w}) + C\sqrt{\frac{\log\left(\frac{M}{\epsilon}\right)}{2n}} \right) \geq 1 - \delta.$$

Theorem 2.1 says that with arbitrarily high probability we can bound the performance of our trained classifier on the unknown distribution $\mathcal{D}$. However, there is nothing to guarantee that the bound is useful in practice. Note that requiring bounds to hold for greater precision involves sending ϵ to 0 which increases the bound. If the bound exceeds C then it is no longer useful as we know already that $R(\mathbf{w}) \leq C$. It is important to note at this stage that are two ways in which PAC bounds can hold. One set of bounds holds in expectation whilst the other hold in probability. Risk is a concept that will develop bounds in expectation. In 2.3 we will introduce definitions that will let us work with bounds that hold in probability. There are two general forms of PAC bounds, we have uniform convergence bounds and algorithmic-dependent bounds [13]. Uniform convergence bounds have the general form

$$\mathbb{P}_{\mathcal{S}\sim\mathcal{D}^m} \left(\sup_{\mathbf{w}\in\mathcal{W}} \left| R(\mathbf{w}) - \hat{R}(\mathbf{w}) \right| \leq \epsilon\left(\frac{1}{\delta}, \frac{1}{m}, \mathcal{W} \right) \right) \geq 1 - \delta.$$

This can be considered as a worst-case analysis of hypothesis generalization, and so in practice will lead to vacuous bounds. Algorithmic-dependent bounds involve the details of a learning algorithm A and take the form

$$\mathbb{P}_{\mathcal{S}\sim\mathcal{D}^m}\left(\left|R\left(A(S)\right)-\hat{R}\left(A(S)\right)\right|\leq\epsilon\left(\frac{1}{\delta},\frac{1}{m},A\right)\right)\geq 1-\delta.$$

These bounds can be seen as a refinement of the uniform convergence bounds as they are only required to hold for the output of the learning algorithm. It will be the subject of Section 5.1 to explore such bounds further.

2.1.3 Occam Bounds

Occam bounds are derived under the assumption that $\mathcal{H}$ is countable and that we have some bias π defined on the hypothesis space. Note that in our setup this does not necessarily mean that $\mathcal{W}$ is countable, as multiple weights may correspond to the same classifier. However, as the Occam bounds hold true for all $h\in\mathcal{H}$ it must also be the case that they hold for all classifiers corresponding to the weight $\mathbf{w}\in\mathcal{W}$. Using this we will instead assume that π is defined over $\mathcal{W}$.

Theorem 2.2 ([6]). *Simultaneously for all $\mathbf{w}\in\mathcal{W}$ and $\delta\in(0,1)$ the following holds,*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(R(\mathbf{w})\leq\inf_{\lambda>\frac{1}{2}}\frac{1}{1-\frac{1}{2\lambda}}\left(\hat{R}(\mathbf{w})+\frac{\lambda C}{m}\left(\log\left(\frac{1}{\pi(\mathbf{w})}\right)+\log\left(\frac{1}{\delta}\right)\right)\right)\right)\geq 1-\delta.$$

2.2 Expected Risk Minimization

In light of Theorem 2.1 it may seem reasonable to want to identify the parameter value $\hat{\mathbf{w}}_{\mathrm{ERM}}$ that minimizes $\hat{R}(\cdot)$. This optimization process is known as Empirical Risk Minimization (ERM) and is formally defined as

$$\hat{\mathbf{w}}_{\mathrm{ERM}}=\inf_{\mathbf{W}\in\mathcal{W}}\hat{R}(\mathbf{w}).$$

2.3 Compression

We now show how PAC bounds can be used to bound the performance of a compressed neural network. In classical statistical theory only as many parameters as training samples are required to overfit. So in practice, neural networks would be able to overfit the training data as they have many more parameters than training samples. Although overfitting to the training sample will yield a low empirical risk, in practice neural networks do not overfit to the data. This suggests that there is some capacity of the network that is redundant in expressing the learned function. In [9] compression frameworks are constructed that aim to reduce the effective number of parameters required to express the function of a trained network whilst maintaining its performance. To do this [9] capitalize on how a neural network responds to noise added to its weights. We first introduce the compression techniques for linear classifiers and then proceed to work with fully connected ReLU neural networks.

2.3.1 Establishing the Notion of Compression

We are in a scenario where we have a learned classifier h that achieves low empirical loss but is complex. In this case, we are considering $\mathcal{Y}=\mathbb{R}^k$ so that the output of h in the i^{th} can be thought of as a relative probability that the input belongs to class i. With this, we define the classification margin loss for $\gamma\geq 0$ to be

$$L_\gamma(h)=\mathbb{P}_{(x,y)\sim\mathcal{D}}\left(h(x)[y]\leq\gamma+\max_{j\neq y}h(x)[j]\right).$$

Similarly, we have the empirical classification margin loss defined as

$$\hat{L}_\gamma(h)=\frac{1}{m}\left|\left\{x_i\in S:h(x_i)[y_i]\leq\gamma+\max_{j\neq y_i}f(x_i)[j]\right\}\right|.$$

We will sometimes use $L(\cdot)$ to denote $L_0(\cdot)$ and refer to it as the classification loss. Suppose that our neural network has d fully connected layers and let x^i be the vector before the activation at layer $i = 0, \ldots, d$ and as x^0 is the input denote it x. Let A^i be the weight matrix of layer i and let layer i have n_i hidden layers with $n = \max_{i=1}^{d} n_i$. The classifier calculated by the network will be denoted $h_{\mathbf{w}}(x)$, where $\mathbf{w}$ can be thought of as a vector containing the weights of the network. For layers $i \leq j$ the operator for composition of the layers will be denoted $M^{i,j}$, the Jacobian of the input x will be denoted $J_x^{i,j}$ and $\phi(\cdot)$ will denote the component-wise ReLU. With these the following hold,

$$x^i = A^i \phi\left(x^{i-1}\right), \; x^j = M^{i,j}\left(x^i\right), \text{ and } M^{i,j}\left(x^i\right) = J_{x^i}^{i,j} x^i.$$

For a matrix B, $\|B\|_F$ will be its Frobenius norm, $\|B\|_2$ its spectral norm and $\frac{\|B\|_F^2}{\|B\|_2^2}$ its stable rank.

Definition 2.3. *Let h be a classifier and $G_{\mathcal{W}} = \{g_{\mathbf{w}} : \mathbf{w} \in \mathcal{W}\}$ be a class of classifiers. We say that h is (γ, S)-compressible via $G_{\mathcal{W}}$ if there exists $\mathbf{w} \in \mathcal{W}$ such that for any $x \in \mathcal{X}$,*

$$|h(x)[y] - g_{\mathbf{w}}(x)[y]| \leq \gamma$$

for all $y \in \{1, \ldots, k\}$.

Definition 2.4. *Suppose $G_{\mathcal{W},s} = \{g_{\mathbf{w},s} : \mathbf{w} \in \mathcal{W}\}$ is a class of classifiers indexed by trainable parameters $\mathbf{w}$ and fixed string s. A classifier h is (γ, S)-compressible with respect to $G_{\mathcal{W},s}$ using helper string s if there exists $\mathbf{w} \in \mathcal{W}$ such that for any $x \in \mathcal{X}$,*

$$|h(x)[y] - g_{\mathbf{w},s}(x)[y]| \leq \gamma$$

for all $y \in \{1, \ldots, k\}$.

Theorem 2.5. *Suppose $G_{\mathcal{W},s} = \{g_{\mathbf{w},s} : \mathbf{w} \in \mathcal{W}\}$ where $\mathbf{w}$ is a set of q parameters each of which has at most r discrete values and s is a helper string. Let S be a training set with m samples. If the trained classifier h is (γ, S)-compressible via $G_{\mathcal{W},s}$ with helper string s, then there exists $\mathbf{w} \in \mathcal{W}$ with high probably such that*

$$L_0(g_{\mathbf{w}}) \leq \hat{L}_\gamma(h) + O\left(\sqrt{\frac{q \log(r)}{m}}\right)$$

over the training set.

Remark 2.6. *Theorem 2.5 only gives a bound for $g_{\mathbf{w}}$ which is the compression of h. However, there are no requirements on the hypothesis class, assumptions are only made on h and its properties on a finite training set.*

Corollary 2.7. *If the compression works for $1 - \xi$ fraction of the training sample, then with a high probability*

$$L_0(g_{\mathbf{w}}) \leq \hat{L}_\gamma(h) + \xi + O\left(\sqrt{\frac{q \log r}{m}}\right).$$

2.3.2 Compression of a Linear Classifier

We now develop an algorithm to compress the decision vector of a linear classifier. We will use linear classifiers to conduct binary classification, where the members of one class have label 1 and the others have label -1. The linear classifiers will be parameterized by the weight vector $\mathbf{w} \in \mathbb{R}^d$ such that for $x \in \mathcal{X}$ we have $h_{\mathbf{w}}(x) = \operatorname{sgn}(\mathbf{w}^\top x)$. Define the margin, $\gamma > 0$, of $\mathbf{w}$ to be such that $y\left(\mathbf{w}^\top x\right) \geq \gamma$ for all (x, y) in the training set. In compressing $\mathbf{w}$, according to Algorithm 1, we end up with a linear classifier parameterized by the weight vector $\hat{\mathbf{w}}$ with some PAC bounds relating to its performance.

Theorem 2.8. *For any number of samples m, Algorithm 1 generates a compressed vector $\hat{\mathbf{w}}$, such that*

$$L(\hat{\mathbf{w}}) \leq \tilde{O}\left(\left(\frac{1}{\gamma^2 m}\right)^{\frac{1}{3}}\right).$$

Algorithm 1 $(\gamma, \mathbf{w})$

```
Require: vector $\mathbf{w}$ with $\|\mathbf{w}\| \leq 1$, $\eta$.
Ensure: vector $\hat{\mathbf{w}}$ such that for any fixed vector $\|u\| \leq 1$, with probability at least $1-\eta$, $\left|\mathbf{w}^\top u - \hat{\mathbf{w}}^\top u\right| \leq \gamma$.
  Vector $\hat{\mathbf{w}}$ has $O\left(\frac{\log d}{\eta\gamma^2}\right)$ non-zero entries.
  for $i = 1 \to d$ do
      Let $z_i = 1$ with probability $p_i = \frac{2w_i^2}{\eta\gamma^2}$ and $0$ otherwise.
      Let $\hat{w}_i = \frac{z_i w_i}{p_i}$.
  end for
  return $\hat{\mathbf{w}}$
```

Algorithm 2 $(\gamma, \mathbf{w})$

```
Require: vector $\mathbf{w}$ with $\|\mathbf{w}\| \leq 1$, $\eta$.
Ensure: vector $\hat{\mathbf{w}}$ such that for any fixed vector $\|u\| \leq 1$, with probability at least $1-\eta$, $|\mathbf{w}^\top u - \hat{\mathbf{w}}^\top u| \leq \gamma$.
  Let $k = \frac{16\log\left(\frac{1}{\eta}\right)}{\gamma^2}$.
  Sample the random vectors $v_1, \ldots, v_k \sim \mathcal{N}(0, I)$.
  Let $z_i = \langle v_i, \mathbf{w} \rangle$.
  (In Discrete Case) Round $z_i$ to closes multiple of $\frac{\gamma}{2\sqrt{dk}}$.
  return $\hat{\mathbf{w}} = \frac{1}{k}\sum_{i=1}^{k} z_i v_i$
```

Remark 2.9. *The rate is not optimal as it depends on $m^{1/3}$ and not $\sqrt{m}$. To resolve this we employ helper strings.*

Remark 2.10. *The vectors v_i of Algorithm 2 form the helper string.*

Theorem 2.11. *For any number of sample m, Algorithm 2 with the helper string generates a compressed vector $\hat{\mathbf{w}}$, such that*

$$L(\hat{\mathbf{w}}) \leq \tilde{O}\left(\sqrt{\frac{1}{\gamma^2 m}}\right).$$

2.3.3 Compression of a Fully Connected Network

In a similar way, the layer matrices of a fully connected network can be compressed in such a way as to maintain a reasonable level of performance. A similar compression algorithm on how to do this is detailed in Algorithm 3. Throughout we will let $\mathbf{w}$ parameterize our classifier. It can just be thought of as a list of layer matrices for our neural network.

Algorithm 3 (A, ϵ, η)

```
Require: Layer matrix $A \in \mathbb{R}^{n_1 \times n_2}$, error parameters $\epsilon, \eta$.
Ensure: Returns $\hat{A}$ such that for all vectors $u, v$ we have that
    $$\mathbb{P}\left(\left|u^\top \hat{A} v - u^\top A v\right| \geq \epsilon \|A\|_F \|u\| \|v\|\right) \leq \eta$$
  Sample $k = \frac{\log(\frac{1}{\eta})}{\epsilon^2}$ random matrices $M_1, \ldots, M_k$ with $\mathrm{i.i.d}$ entries $\pm 1$.
  for $k' = 1 \to k$ do
      Let $Z_l = \langle A, M_l \rangle M_l$
  end for
  return $\hat{A} = \frac{1}{k}\sum_{l=1}^{k} Z_l$
```

Definition 2.12. *If M is a mapping from real-valued vectors to real-valued vectors, and $\mathcal{N}$ is a noise distribution. Then the noise sensitivity of M at x with respect to $\mathcal{N}$ is*

$$\psi_{\mathcal{N}}(M,x) = \mathbb{E}\left(\frac{\|M(x+\eta\|x\|)-M(x)\|^2}{\|M(x)\|^2}\right),$$

and $\psi_{\mathcal{N},S}(M) = \max_{x \in S} \psi_{\mathcal{N}}(M,x)$.

Remark 2.13. *When $x \neq 0$ and the noise distribution is the Gaussian distribution $\mathcal{N}(0,I)$ then the noise sensitivity of matrix M is exactly $\frac{\|M\|_F^2}{\|Mx\|^2}$. Hence, it is at most the stable rank of M.*

Definition 2.14. *The layer cushion of layer i is defined as the largest μ_i such that for any $x \in \mathcal{X}$ we have*

$$\mu_i \left\|A^i\right\|_F \left\|\phi\left(x^{i-1}\right)\right\| \leq \left\|A^i \phi\left(x^{i-1}\right)\right\|.$$

Remark 2.15. *Note that $\frac{1}{\mu_i^2}$ is equal to the noise sensitivity of A^i at $\phi\left(x^{i-1}\right)$ with respect to noise $\eta \sim \mathcal{N}(0,I)$.*

Definition 2.16. *For layers $i \leq j$ the inter-layer cushion $\mu_{i,j}$ is the largest number such that*

$$\mu_{i,j} \left\|J^{i,j}_{x^i}\right\|_F \left\|x^i\right\| \leq \left\|J^{i,j}_{x^i} x^i\right\|$$

for any $x \in \mathcal{X}$. Furthermore, let $\mu_{i\rightarrow} = \min_{i \leq j \leq d} \mu_{i,j}$.

Remark 2.17. *Note that $J^{i,i}_{x^i} = I$ so that*

$$\frac{\left\|J^{i,i}_{x^i} x^i\right\|}{\left\|J^{i,j}_{x^i}\right\|_F \left\|x^i\right\|} = \frac{1}{\sqrt{h^i}}.$$

Furthermore, $\frac{1}{\mu_{i,j}^2}$ is the noise sensitivity of $J^{i,j}_x$ with respect to noise $\eta \sim \mathcal{N}(0,I)$.

Definition 2.18. *The activation contraction c is defined as the smallest number such that for any layer i*

$$\left\|\phi\left(x^i\right)\right\| \geq \frac{\left\|x^i\right\|}{c}$$

for any $x \in \mathcal{X}$.

Definition 2.19. *Let η be the noise generated as a result of applying Algorithm 3 to some of the layers before layer i. Define the inter-layer smoothness ρ_δ to be the smallest number such that with probability $1-\delta$ and for layers $i < j$ we have that*

$$\left\|M^{i,j}\left(x^i+\eta\right) - J^{i,j}_{x^i}\left(x^i+\eta\right)\right\| \leq \frac{\|\eta\| \left\|x^j\right\|}{\rho_\delta \left\|x^i\right\|}$$

for any $x \in \mathcal{X}$.

Remark 2.20. *For a neural network let x be the input, A be the layer matrix and U the Jacobian of the network output with respect to the layer input. Then the network output before compression is given by UAx and after compression the output is given by $U\hat{A}x$.*

Theorem 2.21. *For any fully connected network $h_{\mathbf{w}}$ with $\rho_\delta \geq 3d$, any probability $0 < \delta \leq 1$ and any margin γ. Algorithm 3 generates weights $\tilde{\mathbf{w}}$ such that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(L_0(h_{\tilde{\mathbf{w}}}) \leq \hat{L}_\gamma(h_{\mathbf{w}}) + \tilde{O}\left(\sqrt{\frac{c^2 d^2 \max_{x\in S}\|h_{\mathbf{w}}(x)\|_2^2 \sum_{i=1}^d \frac{1}{\mu_i^2 \mu_{i\rightarrow}^2}}{\gamma^2 m}}\right)\right) \geq 1-\delta.$$

3 Empirical PAC-Bayes Bounds

3.1 Introduction to PAC-Bayes Theory

3.1.1 Bayesian Machine Learning

Here we will outline an introduction to Bayesian machine learning given by [10]. This will provide some context to the framework under which PAC-Bayes bounds are derived. As before we suppose that our training data $S = \{(x_i, y_i)\}_{i=1}^m$ consists of samples from the distribution $\mathcal{D}$ defined on $\mathcal{Z}$. Bayesian machine learning is used to find a parameter $\hat{\mathbf{w}}$ that corresponds to a hypothesis $h_{\hat{\mathbf{w}}}$ with the property that $h_{\hat{\mathbf{w}}}(x) \approx y$. To do this a learning algorithm is employed, which is simply a map from the data space to the parameter space, $\mathcal{W}$. The learning algorithm requires some prior distribution, π, to be defined on $\mathcal{W}$. Then using the training data the posterior distribution, ρ, is formed from the prior distribution. From the posterior distribution, there are many methodologies to then determine the parameter $\hat{\mathbf{w}}$. For example, one could take $\hat{\mathbf{w}}$ to be the mean, median or a random realization of ρ.

3.1.2 Notations and Definitions

Bayesian machine learning is a way to manage randomness and uncertainty in the learning task. PAC-Bayes bounds are derived under this framework.

Definition 3.1 ([15]). *Let $\mathcal{M}(\mathcal{W})$ be a set of probability distributions defined over $\mathcal{W}$. A data-dependent probability measure is a function*

$$\hat{\rho} : \bigcup_{n=1}^{\infty} (\mathcal{X} \times \mathcal{Y})^n \to \mathcal{M}(\mathcal{W}).$$

For ease of notation we will simple write $\hat{\rho}$ to mean $\hat{\rho}((X_1, Y_1), \ldots, (X_n, Y_n))$. The Kullback-Liebler (KL) divergence is a measure of similarity between probability measures defined on the same measurable space.

Definition 3.2 ([15]). *Given two probability measures Q and P defined on some sample space $\mathcal{X}$, the KL divergence between Q and P is*

$$\mathrm{KL}(Q, P) = \int \log\left(\frac{dQ(x)}{dP(x)}\right) Q(dx)$$

when Q is absolutely continuous with respect to P. Otherwise, $\mathrm{KL}(Q, P) = \infty$.

Remark 3.3 ([8]). *When Q, P are probability measures on Euclidean space $\mathbb{R}^d$ with densities q, p respectively. The KL divergence is*

$$\mathrm{KL}(Q, P) := \int \log\left(\frac{q(x)}{p(x)}\right) q(x)dx.$$

Note that KL divergence can take values in the range $[0, \infty]$. Also, note the asymmetry in the definition.

For the multivariate normal distributions [8] $N_q \sim \mathcal{N}(\mu_q, \Sigma_q)$ and $N_p \sim \mathcal{N}(\mu_p, \Sigma_p)$ defined on $\mathbb{R}^d$ we have that,

$$\mathrm{KL}(N_q, N_p) = \frac{1}{2}\left(\mathrm{tr}\left(\Sigma_p^{-1}\Sigma_q\right) - d + (\mu_p - \mu_q)^\top \Sigma_p^{-1}(\mu_p - \mu_q) + \log\left(\frac{\det \Sigma_p}{\det \Sigma_q}\right)\right).$$

Similarly, for Bernoulli distributions [8] $\mathcal{B}(q) \sim \mathrm{Bern}(q)$ and $\mathcal{B}(p) \sim \mathrm{Bern}(p)$ it follows that

$$\mathrm{kl}(q, p) := \mathrm{KL}(\mathcal{B}(q), \mathcal{B}(p)) = q \log\left(\frac{q}{p}\right) + (1 - q) \log\left(\frac{1 - q}{1 - p}\right),$$

For $p^* \in [0, 1]$ bounds of the form $\mathrm{kl}(q, p^*) \leq c$ for some $q \in [0, 1]$ and $c \geq 0$ are of interest. Hence, we introduce the notation

$$\mathrm{kl}^{-1}(q, c) := \sup\{p \in [0, 1] : \mathrm{kl}(q, p) \leq c\}.$$

For a distribution Q defined on $\mathcal{W}$ we will use the notation

$$\mathbb{E}_{\mathbf{w}\sim Q}(R(\mathbf{w})) = R(Q) \text{ and } \mathbb{E}_{\mathbf{w}\sim Q}\left(\hat{R}(\mathbf{w})\right) = \hat{R}(Q)$$

for convenience. The first PAC-Bayes bounds we will encounter is known as Catoni's bound. Recall, that under the Bayesian framework, we first fix a prior distribution, $\pi \in \mathcal{M}(\mathcal{W})$.

3.1.3 PAC-Bayes Bounds

Theorem 3.4 ([5]). *For all $\lambda > 0$, for all $\rho \in \mathcal{M}(\mathcal{W})$, and $\delta \in (0,1)$ it follows that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(R(\rho) \leq \hat{R}(\rho) + \frac{\lambda C^2}{8m} + \frac{\mathrm{KL}(\rho,\pi) + \log\left(\frac{1}{\delta}\right)}{\lambda}\right) \geq 1-\delta.$$

Theorem 3.4 motivates the study of the data-dependent probability measure

$$\hat{\rho}_\lambda = \mathrm{argmin}_{\rho\in\mathcal{M}(\mathcal{W})}\left(\hat{R}(\rho) + \frac{\mathrm{KL}(\rho,\pi)}{\lambda}\right). \tag{1}$$

Definition 3.5 ([15]). *The optimization problem defined by Equation (1) has the solution $\hat{\rho}_\lambda = \pi_{-\lambda\hat{R}}$ given by*

$$\hat{\rho}_\lambda(d\mathbf{w}) = \frac{\exp\left(-\lambda\hat{R}(\mathbf{w})\right)\pi(d\mathbf{w})}{\mathbb{E}\left(\exp\left(-\lambda\hat{R}(\pi)\right)\right)}.$$

This is distribution is known as the Gibbs posterior.

Corollary 3.6 ([15]). *For all $\lambda > 0$, and $\delta \in (0,1)$ it follows that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(R(\hat{\rho}_\lambda) \leq \inf_{\rho\in\mathcal{M}(\mathcal{W})}\left(\hat{R}(\rho) + \frac{\lambda C^2}{8m} + \frac{\mathrm{KL}(\rho,\pi) + \log\left(\frac{1}{\delta}\right)}{\lambda}\right)\right) \geq 1-\delta.$$

For a learning algorithm, we noted that there are different methodologies for how the learned classifier is sampled from the posterior. In the case where consider a single random realization of the posterior distribution, we have the following result.

Theorem 3.7. *[15] For all $\lambda > 0$, $\delta \in (0,1)$, and data-dependent probability measure $\tilde{\rho}$ we have that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\mathbb{P}_{\tilde{\mathbf{w}}\sim\tilde{\rho}}\left(R\left(\tilde{\mathbf{w}}\right) \leq \hat{R}\left(\tilde{\mathbf{w}}\right) + \frac{\lambda C^2}{8m} + \frac{\log\left(\frac{d\rho(\tilde{\mathbf{w}})}{d\pi(\tilde{\mathbf{w}})}\right) + \log\left(\frac{1}{\delta}\right)}{\lambda}\right) \geq 1-\delta.$$

Note that Theorem 3.4 is a bound in probability. We now state an equivalent bound that holds in expectation.

Theorem 3.8. *[15] For all $\lambda > 0$, and data-dependent probability measure $\tilde{\rho}$, we have that*

$$\mathbb{E}_{S\sim\mathcal{D}^m}(R(\tilde{\rho})) \leq \mathbb{E}_{S\sim\mathcal{D}^m}\left(\hat{R}(\tilde{\rho}) + \frac{\lambda C^2}{8m} + \frac{\mathrm{KL}(\tilde{\rho},\pi)}{\lambda}\right).$$

Corollary 3.9 ([15]). *For $\tilde{\rho} = \hat{\rho}_\lambda$, the following holds*

$$\mathbb{E}_{S\sim\mathcal{D}^m}(R(\tilde{\rho})) \leq \mathbb{E}_{S\sim\mathcal{D}^m}\left(\inf_{\rho\in\mathcal{M}(\mathcal{W})}\left(\hat{R}(\rho)\right) + \frac{\lambda C^2}{8m} + \frac{\mathrm{KL}(\rho,\pi)}{\lambda}\right).$$

In the results that follow we will consider the 0-1 loss. This is a measurable function $l : \mathcal{Y}\times\mathcal{Y} \to \{0,1\}$ defined by $l(y,y') = \mathbf{1}(y \neq y')$.

Theorem 3.10. *[1] For all $\rho \in \mathcal{M}(\mathcal{W})$ and $\delta > 0$ we have that*

$$\mathbb{P}_{S \sim \mathcal{D}^m}\left(R(\rho) \leq \hat{R}(\rho) + \sqrt{\frac{\mathrm{KL}(\rho, \pi) + \log\left(\frac{1}{\delta}\right) + \frac{5}{2}\log(m) + 8}{2m - 1}}\right) \geq 1 - \delta.$$

Theorem 3.11 ([4])**.** *For $a > 0$ and $p \in (0,1)$ let*

$$\Phi_a(p) = \frac{-\log\left(1 - p(1 - \exp(-a))\right)}{a}.$$

Then for any $\lambda > 0$, $\delta > 0$ and $\rho \in \mathcal{M}(\mathcal{W})$ we have that

$$\mathbb{P}_{S \sim \mathcal{D}^m}\left(R(\rho) \leq \Phi^{-1}_{\frac{\lambda}{m}}\left(\hat{R}(\rho) + \frac{\mathrm{KL}(\rho, \pi) + \log\left(\frac{1}{\delta}\right)}{\lambda}\right)\right) \geq 1 - \delta.$$

Theorem 3.12 ([3])**.** *For any $\delta > 0$ and $\rho \in \mathcal{M}(\mathcal{W})$ then we have that*

$$\mathbb{P}_{S \sim \mathcal{D}^m}\left(R(\rho) \leq \mathrm{kl}^{-1}\left(\hat{R}(\rho), \frac{\mathrm{KL}(\rho, \pi) + \log\left(\frac{2\sqrt{m}}{\delta}\right)}{m}\right)\right) \geq 1 - \delta.$$

3.2 Optimizing PAC-Bayes Bounds via SGD

In practice, it is often the case that these bounds are not useful. Despite providing insight into how generalization relates to each of the components of the learning process they do not have much utility in providing non-vacuous bounds on the performance of neural networks on the underlying distribution. The significance of the KL divergence between the posterior and the prior can be noted in each of the bounds of Section 3.1.2. This motivated the work of [8] who successfully minimized this term to provide non-vacuous results in practice. They considered a restricted problem that lends itself to efficient optimization. They use stochastic gradient descent to refine the prior, which is effective as SGD is known to find flat minima. This is important as around flat minima such as $\mathbf{w}^*$ we have that $\hat{R}(\mathbf{w}) \approx \hat{R}(\mathbf{w}^*)$ [15]. The setup considered by [8] is the same as the one we have considered throughout this report. With $\mathcal{X} \subset \mathbb{R}^k$ and labels being ± 1. That is, we are considering binary classification based on a set of features. We explicitly state our hypothesis set as

$$\mathcal{H} = \left\{h_{\mathbf{w}} : \mathbb{R}^k \to \mathbb{R} : \mathbf{w} \in \mathbb{R}^d\right\}.$$

We are still considering the 0-1, however, because our classifiers output real numbers we modify the loss slightly to account for this. That is, we let $l : \mathbb{R} \to \{\pm 1\}$ be defined as $l(y, y') = \mathbf{1}(\mathrm{sgn}(y') = y)$. For optimization purposes we use the convex surrogate loss function $\tilde{l} : \mathbb{R} \times \{\pm 1\} \to \mathbb{R}_+$

$$\tilde{l}(y, \hat{y}) = \frac{\log\left(1 + \exp\left(-\hat{y}y\right)\right)}{\log(2)}.$$

For the empirical risk under the convex surrogate loss we write

$$\tilde{R}(\mathbf{w}) = \frac{1}{m}\sum_{i=1}^{m} \tilde{l}(h_{\mathbf{w}}(x_i), y_i).$$

Recall, that this definition implicitly depends on the training sample S_m. As noted previously the work [8] looks to minimize the KL divergence between the prior and the posterior to achieve non-vacuous bounds. To do this they work under a restricted setting and construct a process to minimize the divergence between the prior and the posterior when the learning algorithm is stochastic gradient descent (SGD). To begin [8] utilize the following bound.

Theorem 3.13 ([8]). *For every $\delta > 0$,$m \in \mathbb{N}$, distribution $\mathcal{D}$ on $\mathbb{R}^k \times \{\pm 1\}$, distribution π on $\mathcal{W}$ and distribution $\rho \in \mathcal{M}(\mathcal{W})$, we have that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(\mathrm{kl}\left(\hat{R}(\rho), R(\rho)\right) \leq \frac{\mathrm{KL}(\rho,\pi) + \log\left(\frac{m}{\delta}\right)}{m-1}\right) \geq 1 - \delta.$$

Remark 3.14. *Note how this is a slightly weaker statement than Theorem 3.12. This is because [8] cited this Theorem from [2], however, since then [3] was able to tighten the result by providing Theorem 3.12. In the following we will update the work of [8] and use the tightened result provided by Theorem 3.12.*

This motivates the following PAC-Bayes learning algorithm.

- Fix a $\delta > 0$ and a distribution π on $\mathcal{W}$,
- Collect an $\mathrm{i.i.d}$ sample S_m of size m,
- Compute the optimal distribution ρ on $\mathcal{W}$ that minimizes

$$\mathrm{kl}^{-1}\left(\hat{R}(\rho), \frac{\mathrm{KL}(\rho,\pi) + \log\left(\frac{2\sqrt{m}}{\delta}\right)}{m}\right), \tag{2}$$

- Then return the randomized classifier given by ρ.

Implementing such an algorithm in this general form is intractable in practice. Recall, that we are considering neural networks and so $\mathbf{w}$ represents the weights and biases of our neural network. To make the algorithm more practical we therefore consider

$$\mathcal{M}(\mathcal{W}) = \left\{\mathcal{N}_{\mathbf{w},\mathbf{s}} = \mathcal{N}(\mathbf{w}, \mathrm{diag}(\mathbf{s})) : \mathbf{w} \in \mathbb{R}^d, \mathbf{s} \in \mathbb{R}^d_+\right\}.$$

Utilizing the bound $\mathrm{kl}^{-1}(q,c) \leq q + \sqrt{\frac{c}{2}}$ and replacing the loss with the convex surrogate loss in Equation (2) we obtain the updated optimization problem

$$\min_{\mathbf{w}\in\mathbb{R}^d, \mathbf{s}\in\mathbb{R}^d_+} \tilde{R}\left(\mathcal{N}_{\mathbf{w},\mathbf{s}}\right) + \sqrt{\frac{\mathrm{KL}(\mathcal{N}_{\mathbf{w},\mathbf{s}},\pi) + \log\left(\frac{2\sqrt{m}}{\delta}\right)}{2m}}. \tag{3}$$

We now suppose our prior π is of the form $\mathcal{N}(\mathbf{w}_0, \lambda I)$. As we will see the choice of $\mathbf{w}_0$ is not too impactful, as long as it is not $\mathbf{0}$. However, to efficiently choose a judicious value for λ we discretize the problem, with the side-effect of expanding the eventual generalization bound. We let λ have the for $c\exp\left(-\frac{j}{b}\right)$ for $j \in \mathbb{N}$, so that c is an upper bound and b controls precision. By ensuring that Theorem 3.12 holds with probability $1 - \frac{6\delta}{\pi^2 j^2}$ for each $j \in \mathbb{N}$ we can then apply a union bound argument to ensure that we get results that hold for probability $1 - \delta$. Treating λ as continuous during the optimization process and then discretized at the point of evaluating the bound yields the updated optimization problem

$$\min_{\mathbf{w}\in\mathbb{R}^d, \mathbf{s}\in\mathbb{R}^d_+, \lambda\in(0,c)} \tilde{R}(\mathcal{N}_{\mathbf{w},\mathbf{s}}) + \sqrt{\frac{1}{2}B_{\mathrm{RE}}(\mathbf{w},\mathbf{s},\lambda;\delta)} \tag{4}$$

where

$$B_{\mathrm{RE}}(\mathbf{w},\mathbf{s},\lambda;\delta) = \frac{\mathrm{KL}(\mathcal{N}_{w,s}, \mathcal{N}(\mathbf{w}_0, \lambda I)) + 2\log\left(b\log\left(\frac{c}{\lambda}\right)\right) + \log\left(\frac{\pi^2\sqrt{m}}{3\delta}\right)}{m}.$$

To optimize Equation (4) we would like to compute its gradient and apply SGD. However, this is not feasible in practice for $\tilde{R}(\mathcal{N}_{\mathbf{w},\mathbf{s}})$. Instead we compute the gradient of $\tilde{R}\left(\mathbf{w} + \xi \odot \sqrt{\mathbf{s}}\right)$ where $\xi \sim \mathcal{N}_{0,\mathbf{1}_d}$. Once good candidates for this optimization problem are found we return to (2) to calculate the final error bound. With the choice of λ it follows that with probability $1-\delta$, uniformly over all $\mathbf{w}\in\mathbb{R}^d, \mathbf{s}\in\mathbb{R}^d_+$ and λ (of the discrete form) the expected risk of $\rho=\mathcal{N}_{\mathbf{w},\mathbf{s}}$ is bounded by

$$\mathrm{kl}^{-1}\left(\hat{R}(\rho), B_{\mathrm{RE}}(\mathbf{w},\mathbf{s},\lambda;\delta)\right).$$

However, it is often not possible to compute $\hat{R}(\rho)$ due to the intractability of ρ. So instead an unbiased estimate is obtained by estimating ρ using a Monte Carlo approximation. Given n $\mathrm{i.i.d}$ samples $\mathbf{w}_1,\dots,\mathbf{w}_n$ from ρ we use the Monte Carlo approximation $\hat{\rho}_n=\sum_{i=1}^n\delta_{\mathbf{w}_i}$, to get the bound

$$\hat{R}(\rho)\leq\overline{\hat{R}_{n,\delta'}}(\rho):=\mathrm{kl}^{-1}\left(\hat{R}\left(\hat{\rho}_n\right),\frac{1}{n}\log\left(\frac{2}{\delta'}\right)\right),$$

which holds with probability $1-\delta'$. Finally, by Theorem 3.15

$$R(\rho)\leq\mathrm{kl}^{-1}\left(\overline{\hat{R}_{n,\delta'}}(\rho), B_{\mathrm{RE}}(\mathbf{w},\mathbf{s},\lambda;\delta)\right),$$

holds with probability $1-\delta-\delta'$. Now all that is left is to do is to determine optimal values for $\mathbf{w}$ and $\mathbf{s}$. To do this first train a neural network via SGD to get a value of $\mathbf{w}$. Then instantiate a stochastic neural network with the multivariate normal distribution $\rho=\mathcal{N}_{\mathbf{w},\mathbf{s}}$ over the weights, with $\mathbf{s}=|\mathbf{w}|$. Next apply Algorithm 4 to deduce values of $\mathbf{w},\mathrm{s}$ and λ that give a tighter bound.

Algorithm 4 Optimizing the PAC Bounds

```
Require:
  $\mathbf{w}_0\in\mathbb{R}^d$, the network parameters at initialization.
  $\mathbf{w}\in\mathbb{R}^d$, the network parameters after SGD.
  $S_m$, training examples.
  $\delta\in(0,1)$, confidence parameter.
  $b\in\mathbb{N}, c\in(0,1)$, precision and bound for $\lambda$.
  $\tau\in(0,1), T$, learning rate.
Ensure: Optimal $\mathbf{w},\mathrm{s},\lambda$.
  $\zeta=|\mathbf{w}|$                                              ▷ $\mathbf{s}(\zeta)=e^{2\zeta}$
  $\rho=-3$                                                         ▷ $\lambda(\rho)=e^{2\rho}$
  $B(\mathbf{w},\mathbf{s},\lambda,\mathbf{w}')=\tilde{R}(\mathbf{w})+\sqrt{\frac{1}{2}B_{\mathrm{RE}}(\mathbf{w},\mathbf{s},\lambda)}$
  for $t=1\to T$ do
    Sample $\xi\sim\mathcal{N}(0,I_d)$
    $\mathbf{w}'(\mathbf{w},\zeta)=\mathbf{w}+\xi\odot\sqrt{\mathbf{s}(\zeta)}$
    $\begin{pmatrix}\mathbf{w}\\ \zeta\\ \rho\end{pmatrix}=-\tau\begin{pmatrix}\nabla_{\mathbf{w}}B(\mathbf{w},\mathbf{s}(\zeta),\lambda(\rho),\mathbf{w}'(\mathbf{w},\zeta))\\ \nabla_{\zeta}B(\mathbf{w},\mathbf{s}(\zeta),\lambda(\rho),\mathbf{w}'(\mathbf{w},\zeta))\\ \nabla_{\rho}B(\mathbf{w},\mathbf{s}(\zeta),\lambda(\rho),\mathbf{w}'(\mathbf{w},\zeta))\end{pmatrix}$
  end for
  return $\mathbf{w},\mathrm{s}(\zeta),\lambda(\rho)$
```

Once the values of $\mathbf{w},\mathrm{s}$ and λ are found we then need to compute $\overline{\hat{R}_{n,\delta'}}(\rho):=\mathrm{kl}^{-1}\left(\hat{R}\left(\hat{\rho}_n\right),\frac{1}{n}\log\left(\frac{2}{\delta'}\right)\right)$ to get our bound. We note that

$$\hat{R}(\hat{\rho}_n)=\sum_{i=1}^n\delta_{\mathbf{w}_i}\left(\frac{1}{m}\sum_{j=1}^m l(h_{\mathbf{w}_i}(x_j),y_j)\right).$$

Then to invert the kl divergence we employ Newton's method, in the form of Algorithm 5, to get an approximation for our bound.

Algorithm 5 Newton's Method for Inverting kl Divergence

```
Require: $q, c$, initial estimate $p_0$ and $N \in \mathbb{N}$
Ensure: $p$ such that $p \approx \mathrm{kl}^{-1}(q, c)$
  for $n = 1 \to N$ do
    if $p \geq 1$ then
      return $1$
    else
      $p_0 = p_0 - \frac{q\log\left(\frac{q}{c}\right)+(1-q)\log\left(\frac{1-q}{1-c}\right)-c}{\frac{1-q}{1-p}-\frac{q}{p}}$
    end if
  end for
  return $p_0$
```

4 Oracle PAC-Bayes Bounds

4.1 Theory of Oracle PAC-Bayes Bounds

Oracle bounds are theoretical objects that are not suitable for practical applications. Their utility lies in their ability to highlight properties about the behavior of the bounds and they can take the form

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(R\left(\hat{\mathbf{w}}\right) \leq \inf_{\mathbf{w}\in\mathcal{W}} R(\mathbf{w}) + r_m(\delta)\right) \geq 1-\delta.$$

Where $r_m(\delta)$ is a remainder term that tends to 0 as m tends to ∞. Although this bound cannot be computed in practice it is illustrative of the behavior of the bound. Just like empirical bounds, there exist oracle bounds that hold in expectation and in probability.

4.1.1 Oracle PAC-Bayes Bounds in Expectation

Theorem 4.1 ([15]). *For $\lambda > 0$ we have that*

$$\mathbb{E}_{S\sim\mathcal{D}^m} R(\hat{\rho}_\lambda) \leq \inf_{\rho\in\mathcal{M}(\mathcal{W})}\left(R(\rho) + \frac{\lambda C^2}{8m} + \frac{\mathrm{KL}(\rho,\pi)}{\lambda}\right).$$

4.1.2 Oracle PAC-Bayes Bounds in Probability

Theorem 4.2 ([15]). *For any $\lambda > 0$, and $\delta \in (0,1)$ we have that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(R(\hat{\rho}_\lambda) \leq \inf_{\rho\in\mathcal{M}(\mathcal{W})}\left(R(\rho) + \frac{\lambda C^2}{4m} + \frac{2\mathrm{KL}(\rho,\pi) + \log\left(\frac{2}{\delta}\right)}{\lambda}\right)\right) \geq 1-\delta.$$

4.1.3 Bernstein's Assumption

Definition 4.3 ([15]). *Let $\mathbf{w}^*$ denote a minimizer of R when it exists,*

$$R(\mathbf{w}^*) = \min_{\mathbf{w}\in\mathcal{W}} R(\mathbf{w}).$$

When $\mathbf{w}^*$ exists and there is a constant K such that for any $\mathbf{w} \in \mathcal{W}$ we have that

$$\mathbb{E}_{S\sim\mathcal{D}^m}\left(\left(l(h_{\mathbf{w}}(x_i), y_i) - l(h_{\mathbf{w}^*}(x_i), y_i)\right)^2\right) \leq K\left(R(\mathbf{w}) - R(\mathbf{w}^*)\right)$$

we say that Bernstein's assumption is satisfied with constant K.

Theorem 4.4 ([15]). *Assume Bernstein's assumption is satisfied with some constant $K > 0$. Take $\lambda = \frac{m}{\max(2K,C)}$ then we have*

$$\mathbb{E}_{S\sim\mathcal{D}^m} R(\hat{\rho}_\lambda) - R\left(\mathbf{w}^*\right) \leq 2\inf_{\rho\in\mathcal{M}(\mathcal{W})}\left(R(\rho) - R\left(\mathbf{w}^*\right) + \frac{\max(2K,C)\mathrm{KL}(\rho,\pi)}{m}\right).$$

4.2 Data Driven PAC-Bayes Bounds

A lot of work to obtain non-vacuous PAC-Bayes bounds is to develop priors that reduce the size of the KL divergence between the prior and the posterior. The idea behind the work of [12] is to hold out some of the training data to obtain data-inspired priors. For this section, we use a PAC-Bayes bound that can be thought of as the Bayesian equivalent of Theorem 2.2, however, now we are dealing with potentially uncountable hypothesis sets.

Theorem 4.5 ([6]). *For $\lambda > \frac{1}{2}$ selected before drawing our training sample, then for all $\rho \in \mathcal{M}(\mathcal{W})$ and $\delta \in (0,1)$ we have that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(R(\rho) \leq \frac{1}{1-\frac{1}{2\lambda}}\left(\hat{R}(\rho) + \frac{\lambda C}{m}\left(\mathrm{KL}(\rho,\pi) + \log\left(\frac{1}{\delta}\right)\right)\right)\right) \geq 1-\delta.$$

Corollary 4.6 ([12]). *Let $\beta, \delta \in (0,1)$, $\mathcal{D}$ a probability distribution over $\mathcal{Z}$, and $\pi \in \mathcal{M}(\mathcal{W})$. Then for all $\rho \in \mathcal{M}(\mathcal{W})$ we have that*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(R(\rho) \leq \Psi_{\beta,\delta}(\rho,\pi;S)\right) \geq 1-\delta,$$

where $\Psi_{\beta,\delta}(\rho,\pi;S) = \frac{1}{\beta}\hat{R}(\rho) + \frac{\mathrm{KL}(\rho,\pi)+\log\left(\frac{1}{\delta}\right)}{2\beta(1-\beta)m}$.

As we have done previously, we can consider the optimization problem of minimizing the bound of Corollary 4.6.

Theorem 4.7 ([12]). *Let $m \in \mathbb{N}$ and fix a probability kernel $\rho : \mathcal{Z}^m \to \mathcal{M}(\mathcal{W})$. Then for all $\beta, \delta \in (0,1)$ and distributions $\mathcal{D}$ defined on $\mathcal{Z}$ we that $\mathbb{E}_{S\sim\mathcal{D}^m}\left(\Psi_{\beta,\delta}(\rho(S),\pi;S)\right)$ is minimized, in π, by the oracle prior $\pi^* = \mathbb{E}_{S\sim\mathcal{D}^m}(\rho(S))$.*

For a subset J of $\{1,\ldots,m\}$ of size n, we can use it to sample the training data and yield the subset S_J. We can then define the data-dependent oracle prior as

$$\pi^*(S_J) = \inf_{\pi\in\mathcal{Z}^n\to\mathcal{M}(\mathcal{W})} \mathbb{E}(\mathrm{KL}(\rho(s),\pi(S_J)))$$

which turns out to be $\pi^*(S_J) = \mathbb{E}(\rho(S)|S_J)$. It can be shown that the data-dependent oracle prior minimizes the bound of Corollary 4.6 in expectation. Therefore, despite being a theoretical quantity, as it cannot be computed in practice, it motivates the construction of practical data-dependent priors as a method to tighten the bounds.

4.2.1 Implementing Data-Dependent Priors

To implement data-dependent priors we restrict the optimization problem to make it tractable. We only consider the set of Gaussian priors $\mathcal{F}$ that generate Gaussian posteriors. Neural networks are trained via SGD, and hence there is some randomness to the learning algorithm. Let $(\Omega, \mathcal{F}, \nu)$ define a probability space and let us focus on the kernels

$$\rho : \Omega \times \mathcal{Z}^m \to \mathcal{M}(\mathcal{W}), \quad \rho(U,S) = \mathcal{N}(\mathbf{w}_S, \mathbf{s}),$$

where $\mathbf{w}_S$ are the learned weights via SGD on the full dataset S. The random variable U represents the randomness of the learning algorithm. As before we consider a non-negative integer $n \leq m$ and with $\alpha = \frac{n}{m}$ we define a subset S_α of size n containing the first n indices of S processed by SGD. Let $\mathbb{E}^{S_\alpha,U}[\cdot]$ denote the conditional expectation operator given S_α and U. Our aim now is to tighten the bound of Corollary 4.6 by minimizing $\mathbb{E}^{S_\alpha,U}(\mathrm{KL}(\rho(U,S),\pi))$. To do this we further restrict the priors of consideration to those of the form $\mathcal{N}(\mathbf{w}_\alpha, \sigma I)$ such that with σ fixed we are left with the minimization problem

$$\mathrm{argmin}_{\mathbf{w}_\alpha}\left(\mathbb{E}^{S_\alpha,U}\left(\|\mathbf{w}_S - \mathbf{w}_\alpha\|\right)\right),$$

which can be solved to yield $\mathbf{w}_\alpha = \mathbb{E}^{S_\alpha,U}(\mathbf{w}_S)$. This minimizer is unknown in practice so we attempt to approximate it. We first define a so-called ghost sample, S^G, which is an independent sample equal in distribution to S. We combine a $1-\alpha$ fraction of S^G with S_α to obtain the sample S^G_α. Let $\mathbf{w}^G_\alpha$ be the mean of $\rho(U, S^G_\alpha)$. By construction, SGD will first process S_α then the combined portion of S^G and hence $\mathbf{w}^G_\alpha$ and $\mathbf{w}_S$ are equal in distribution when conditioned on S_α and U. Therefore, $\mathbf{w}^G_\alpha$ is an unbiased estimator of $\mathbb{E}^{S_\alpha,U}(\mathbf{w}_S)$. Before formalizing this process algorithmically we clarify some notation.

- The SGD run on S is the base run.
- The SGD run on S_α is the α-prefix run.
- The SGD run on S^G_α is the α-prefix+ghost run and obtains the parameters $\mathbf{w}^G_\alpha$.

The resulting parameters of the α-prefix and α-prefix+ghost run can be used as the centres of the Gaussian priors to give the tightened generalization bounds. However, sometimes the ghost sample is not attainable in practice, and hence one simply relies upon α-prefix runs to obtain the mean of the prior. It is not clear whether α-prefix+ghost run will always obtain a parameter that leads to a tighter generalization bound. Recall, that σ is assumed to be fixed in the optimization process. Algorithm 7 is independent of this parameter and so it can be optimized afterwards without requiring a re-run of the optimization process.

Algorithm 6 Stochastic Gradient Descent

```
Require: Learning rate $\eta$
  function SGD($\mathbf{w}_0, S, b, t, \mathcal{E} = -\infty$)
      $\mathbf{w} \leftarrow \mathbf{w}_0$
      for $i \leftarrow 1$ to $t$ do
          Sample $S' \in S$ with $|S'| = b$
          $\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla l_{S'}(\mathbf{w})$
          if $l^{0\text{-}1}_S(\mathbf{w}) \leq \mathcal{E}$ then
              break
          end if
      end for
  end function
```

Algorithm 7 Obtaining Bound Using SGD Informed Prior

```
Require: Stopping criteria $\mathcal{E}$, Prefix fraction $\alpha$, Ghost Data $S^G$ (If available), Batch size $b$.
  function GETBOUND($\mathcal{E}, \alpha, T, \sigma_P$)
      $S_\alpha \leftarrow \{z_1, \ldots, z_{\alpha|S|} \subset S\}$
      $\mathbf{w}^0_\alpha \leftarrow$SGD$\Big(\mathbf{w}_0, S_\alpha, b, \frac{|S_\alpha|}{b}\Big)$
      $\mathbf{w}_S \leftarrow$SGD$\big(\mathbf{w}^0_\alpha, S, b, \infty, \mathcal{E}\big)$          ▷ Base Run
      $\mathbf{w}^G_\alpha \leftarrow$SGD$\big(\mathbf{w}^0_\alpha, S^G_\alpha, b, T, \cdot\big)$          ▷ Ghost run if data available, otherwise prefix run
      $\pi \leftarrow \mathcal{N}\left(\mathbf{w}^G_\alpha, \sigma I\right)$
      $\rho \leftarrow \mathcal{N}\left(\mathbf{w}_S, \sigma I\right)$
      Bound$\leftarrow \Psi^*_\delta(\rho, \pi; S \setminus S_\alpha)$
      return Bound
  end function
```

5 Extensions of PAC-Bayes Bounds

5.1 Disintegrated PAC-Bayes Bounds

The majority of the PAC-Bayes bounds we have discussed so far have been derived to hold for all posterior distributions. The intention of disintegrated PAC-Bayes bounds is to refine these results by only requiring them to hold for a single posterior distribution. We now study the work of [13] that sets out a general framework for deriving such bounds. The setup is the same as the one we have considered so far, with the added assumption that $C = 1$ and the additional consideration of a deterministic learning algorithm $A : \mathcal{Z}^m \to \mathcal{M}(\mathcal{W})$ that is applied to the training sample S.

Definition 5.1 ([13]). *The two distributions P and Q defined on the some sample space $\mathcal{X}$, then for any $\alpha > 1$ their Renyi divergence is defined to be*

$$D_\alpha(Q, P) = \frac{1}{\alpha - 1} \log \left(\mathbb{E}_{x \sim P} \left(\frac{Q(x)}{P(x)} \right)^\alpha \right).$$

Theorem 5.2 ([13]). *For any distribution $\mathcal{D}$ on $\mathcal{Z}$, for any parameter space $\mathcal{W}$, for any prior distribution π on $\mathcal{W}$, for any $\phi : \mathcal{W} \times \mathcal{Z}^m \to \mathbb{R}^+$, for any $\alpha > 1$, for any $\delta > 0$ and for any deterministic learning algorithm $A : \mathcal{Z}^m \to \mathcal{M}(\mathcal{W})$ the following holds*

$$\mathbb{P}_{S \sim \mathcal{D}^m, \mathbf{w} \sim \rho_S} \left(\frac{\alpha}{\alpha - 1} \log\left(\phi(\mathbf{w}, S)\right) \leq \frac{2\alpha - 1}{\alpha - 1} \log\left(\frac{2}{\delta}\right) + D_\alpha(\rho_S, \pi) + \log\left(\mathbb{E}_{S' \sim \mathcal{D}^m} \mathbb{E}_{\mathbf{w}' \sim \pi} \phi(\mathbf{w}', S')^{\frac{\alpha}{\alpha-1}} \right) \right) \geq 1 - \delta,$$

where $\rho_S := A(S)$.

5.1.1 Application to Neural Network Classifiers

We can contextualize this bound to over-parameterized neural networks. Suppose that $\mathbf{w} \in \mathbb{R}^d$ is a weight vector of a neural network, with $d \gg m$. Assume that the network is trained for T epochs and that these epochs are used to generate T priors $\mathbf{P} = \{\pi_t\}_{t=1}^T$. Let the priors be of the form $\pi_t = \mathcal{N}\left(\mathbf{w}_t, \sigma^2 \mathbf{I}_d\right)$ where $\mathbf{w}_t$ is the weight vector obtained after the t^{th} epoch. We assume that the priors are obtained from the learning algorithm being applied to the sample S_{prior} where $S_{\text{prior}} \cap S = \emptyset$.

Corollary 5.3. *For any distribution $\mathcal{D}$ on $\mathcal{Z}$, for any set $\mathcal{W}$, for any set $\mathbf{P}$ of T priors on $\mathcal{W}$, for any learning algorithm $A : \mathcal{Z}^m \to \mathcal{M}(\mathcal{W})$, for any loss $l : \mathcal{W} \times \mathcal{Z} \to [0, 1]$ and for any $\delta > 0$ then for any $\pi_t \in \mathbf{P}$ we have that*

$$\mathbb{P}_{\mathcal{S} \sim \mathcal{D}^m, \mathbf{w} \sim \rho_S} \left(\mathrm{kl}\left(\hat{R}(\mathbf{w}), R(\mathbf{w}) \right) \leq \frac{1}{m} \left(\frac{\|\mathbf{w} - \mathbf{w}_t\|_2^2}{\sigma^2} + \log\left(\frac{16T\sqrt{m}}{\delta^3} \right) \right) \right) \geq 1 - \delta.$$

Corollary 5.4 ([13]). *Under the assumptions of Corollary 5.3 with $\delta \in (0, 1)$ and for all $\pi_t \in \mathbf{P}$ we have that*

$$\mathbb{P}_{S \sim \mathcal{D}^m, \mathbf{w} \sim \rho_S} \left(\mathrm{kl}\left(\hat{R}(\mathbf{w}), R(\mathbf{w}) \right) \leq \frac{1}{m} \left(\frac{\|\mathbf{w} + \epsilon - \mathbf{w}_t\|_2^2 - \|\epsilon\|_2^2}{2\sigma^2} + \log\left(\frac{2T\sqrt{m}}{\delta} \right) \right) \right),$$

$$\mathbb{P}_{S \sim \mathcal{D}^m, \mathbf{w} \sim \rho_S} \left(\mathrm{kl}\left(\hat{R}(\mathbf{w}), R(\mathbf{w}) \right) \leq \frac{1}{m} \left(\frac{m+1}{m} \frac{\|\mathbf{w} + \epsilon - \mathbf{w}_t\|_2^2 - \|\epsilon\|_2^2}{2\sigma^2} + \log\left(\frac{T(m+1)}{\delta} \right) \right) \right),$$

and for all $c \in \mathbf{C}$

$$R(\mathbf{w}) \leq \frac{1 - \exp\left(-c\hat{R}(\mathbf{w}) - \frac{1}{m} \left(\frac{\|\mathbf{w} + \epsilon - \mathbf{w}_t\|_2^2 - \|\epsilon\|_2^2}{2\sigma^2} + \log\left(\frac{T|\mathbf{C}|}{\delta} \right) \right) \right)}{1 - \exp(-c)}.$$

Where $\epsilon \sim \mathcal{N}\left(\mathbf{0}, \sigma^2 \mathbf{I}_d\right)$ is Gaussian noise such that $\mathbf{w} + \epsilon$ acts as the weights sampled from $\mathcal{N}(\mathbf{w}, \sigma^2 \mathbf{I}_d)$, and $\mathbf{C}$ is a set of hyper-parameters fixed a priori.

5.2 PAC-Bayes Compression Bounds

We will now see how compression ideas can be capitalized to tighten PAC-Bayes bounds. The work of [11] evaluates generalization bounds by first measuring the effective compressed size of a neural network and then substituting this into the bounds. We have seen that compression techniques can efficiently reduce the effective size of a network, and so accounting for this can lead to tighter bounds. This also captures the intuition that we expect a model to overfit if it is more difficult to compress. Therefore, these updated bounds also incorporate a notion of model complexity. The work of [11] utilizes a refined version of Theorem 3.11.

Theorem 5.5 ([4]). *Let L be a 0-1 valued loss function. Let π be a probability measure on the parameter space, and let $\alpha > 1, \delta > 0$. Then,*

$$\mathbb{P}_{S\sim\mathcal{D}^m}\left(R(\rho) \leq \inf_{\lambda>1} \Phi^{-1}_{\lambda/m}\left(\hat{R}(\rho) + \frac{\alpha}{\lambda}\left(\mathrm{KL}(\rho,\pi) - \log(\delta) + 2\log\left(\frac{\log\left(\alpha^2\lambda\right)}{\log(\alpha)}\right)\right)\right)\right) \geq 1-\delta.$$

The intention now is to motivate the choice of π using ideas of compressibility such that $\mathrm{KL}(\rho,\pi)$ is kept small. To do this we will choose a prior π that assigns greater probability mass to models with a shorter code length.

Theorem 5.6 ([11]). *Let $|\mathbf{w}|_c$ denote the number of bits required to represent hypothesis $h_{\mathbf{w}}$ using some pre-specified coding c. Let ρ denote the point mass distribution at $\hat{\mathbf{w}}$ which is the compression of $\mathbf{w}$ and corresponds to the compressed model $h_{\hat{\mathbf{w}}}$. Let M denote any probability measure on the positive integers. Then there exists a prior π_c such that*

$$\mathrm{KL}(\rho,\pi_c) \leq |\hat{\mathbf{w}}|_c \log(2) - \log\left(M\left(|\hat{\mathbf{w}}|_c\right)\right).$$

Remark 5.7. *An example of a coding scheme c could be the Huffman encoding. However, such a compression scheme is agnostic to any structure of the hypotheses which is translated to the space $\mathcal{W}$. By exploiting structure in the hypothesis class the bound can be improved substantially.*

We now formalize compression schemes to allow us to refine Theorem 5.6. Denote a compression procedure by a triple (S, C, Q) where

- $S = \{s_1, \ldots, s_k\} \subseteq \{1, \ldots, d\}$ is the location of the non-zero weights,
- $C = \{c_1, \ldots, c_r\} \subseteq \mathbb{R}$, is a codebook, and
- $Q = (q_1, \ldots, q_k)$ for $q_i \in \{1, \ldots, r\}$ are the quantized values.

Define the corresponding weights $\mathbf{w}(S, Q, C) \in \mathbb{R}^d$ as,

$$w_i(S,Q,C) = \begin{cases} c_{q_j} & i = s_j \\ 0 & \text{otherwise.} \end{cases}$$

Training a neural network is a stochastic process due to the randomness of SGD. So to analyse the generalization error we try to capture randomness in the analysis by applying Gaussian noise to weights. For this we use $\rho \sim \mathcal{N}\left(\mathbf{w}, \sigma^2 J\right)$, with J being a diagonal matrix.

Theorem 5.8 ([11]). *Let (S, C, Q) be the output of a compression scheme, and let $\rho_{S,C,Q}$ be the stochastic estimator given by the weights decoded from the triplet and variance σ^2. Let c denote an arbitrary fixed coding scheme and let M denote an arbitrary distribution on the positive integers. Then for any $\tau > 0$, there is a prior π such that*

$$\begin{aligned}\mathrm{KL}(\rho_{S,C,Q},\pi) \leq &(k\lceil\log(r)\rceil + |S|_c + |C|_c)\log(2) - \log(M(k\lceil\log(r)\rceil + |S|_c + |C|_c)) \\ &+ \sum_{i=1}^{k} \mathrm{KL}\left(\mathcal{N}\left(c_{q_i},\sigma^2\right), \sum_{j=1}^{r}\mathcal{N}\left(c_j,\tau^2\right)\right).\end{aligned}$$

Choosing the prior alluded to by Theorem 5.8 and utilizing Theorem 5.5 one can obtain a PAC-Bayes generalization bound that exploits notions of compressibility.

6 Appendix

6.1 Extensions to Convolutional Neural Networks

In this section, we extend the ideas of Section 2.3 to convolutional neural networks (CNN) [9]. This extension is not trivial due to the parameter sharing that occurs in the CNN architecture. To investigate these ideas we update our notation from that of Section 2.3. In particular, we suppose that the i^{th} layer has an image dimension of $n_1^i \times n_2^i$, where each pixel has l^i channels, and the filter at layer i has size $\kappa_i \times \kappa_i$ with stride s_i. The convolutional filter has dimension $l^{i-1} \times l^i \times \kappa_i \times \kappa_i$. If we apply Algorithm 3 to each copy of the filter then the number of new parameters grows proportionally to $n_1^i n_2^i$, which is undesirable. On the other hand, compressing the filter once and re-using it for all patches removes the implicit assumption that the noise generated by the compression behaves similar to a Gaussian as the shared filters introduces correlations. To solve these issues Algorithm 8 generates p-wise independent compressed filters for each convolution location. This results in p more parameters than a single compression, but if p grows logarithmically with respect to the relevant parameters then the filters behave like fully independent filters. To proceed with this idea we need to introduce some operations. For $k' \leq k$ let Y be a k^{th} order tensor and Z a $(k')^{\text{th}}$ order tensor with a matching dimensionality to the last k'-dimensions of Y. The product operator $\times_{k'}$ when given tensors Y and Z returns a $(k-k')^{\text{th}}$ order tensor as follows

$$(Y \times_{k'} Z)_{i_1,\ldots,i_{k-k'}} = \left\langle Y_{i_1,\ldots,i_{k-k'}}, Z \right\rangle = \left\langle \operatorname{vec}\left(Y_{i_1,\ldots,i_{k-k'}}\right), \operatorname{vec}(Z) \right\rangle.$$

Let $X \in \mathbb{R}^{l \times n_1 \times n_2}$ be an $n \times n$ image where the pixels have l features. Denote the $\kappa \times \kappa$ sub-image starting from pixel (i,j) by $X_{(i,j),\kappa} \in \mathbb{R}^{l \times \kappa \times \kappa}$. Let $A \in \mathbb{R}^{l' \times l \times \kappa \times \kappa}$ be a convolutional weight tensor. The convolutional operator with stride s can then be defined as

$$(A *_s X)_{i,j} = A \times_3 X_{(s(i-1)+1, s(j+1)+1),\kappa}$$

for $1 \leq i \leq \left\lfloor \frac{n_1-\kappa}{s} \right\rfloor =: n_1'$ and $1 \leq j \leq \left\lfloor \frac{n_2-\kappa}{s} \right\rfloor =: n_2'$ so that $A *_s X \in \mathbb{R}^{l' \times n_1' \times n_2'}$. Algorithm 8 generates p-wise independent filters $\hat{A}_{(a,b)}$ for each convolution location $(a,b) \in [n_1'] \times [n_2']$ and so $\hat{A} *_s X$ will be used to denote the convolution operator

$$\left(\left(\hat{A} *_s X\right)_{i,j}\right) = \hat{A}_{(i,j)} \times_3 X_{(s(i-1)+1, s(j+1)+1),\kappa}$$

for $1 \leq i \leq n_1'$ and $1 \leq j \leq n_2'$. With this we see that for any $i > 1$ we have

$$x^{i+1} = \phi\left(A^i *_{s_i} x^i\right), \text{ and } x^j = M^{ij}\left(x^i\right) = J^{ij}_{x^i} \times_3 x^i.$$

Definition 6.1. *For any two layer $i \leq j$, we define the inter-layer cushion $\mu_{i,j}$ as the largest number such that for any $(x,y) \in S$ we have that*

$$\mu_{i,j} \frac{1}{\sqrt{n_1^i n_2^i}} \left\| J^{i,j}_{x^i} \right\|_F \left\| x^i \right\| \leq \left\| J^{i,j}_{x^i} x^i \right\|.$$

For any layer i let the minimal inter-layer cushion be $\mu_{i\rightarrow} = \min_{i \leq j \leq d} \mu_{i,j} = \min\left(\frac{1}{\sqrt{l^i}}, \min_{i<j\leq d} \mu_{i,j}\right)$.

Definition 6.2. *Let $J^{i,j}_x \in \mathbb{R}^{l^i \times n_1^i \times n_2^i \times l^j \times n_1^j \times n_2^j}$ be the Jacobian of $M^{i,j}$ at x. We say that the Jacobian is β well-distributed if for any $(x,y) \in S$, any i,j and any $(a,b) \in \left[n_1^i \times n_2^i\right]$ we have that*

$$\left\| \left[J^{i,j}_x\right]_{:,a,b,:,:,:} \right\|_F \leq \frac{\beta}{\sqrt{n_1^i n_2^i}} \left\| J^{i,j}_x \right\|_F.$$

For any $\delta > 0$, $\epsilon \leq 1$, let $G = \left\{\left(U^i, V^i\right)\right\}_{i=1}^m$ be a set of matrix/vector pairs where $U \in \mathbb{R}^{l' \times n_1' \times n_2' \times n_u}$ and $V \in \mathbb{R}^{l \times n_1 \times n_2}$, let $\hat{A}_{(i,j)} \in \mathbb{R}^{l \times l'}$ be the output of Algorithm 8 with $\eta = \frac{\delta}{\eta}$ and $\Delta_{(i,j)} = \hat{A}_{(i,j)} - A$. Suppose the U's are β-well-distributed. Then for any $(U,V) \in G$ we have that

$$\mathbb{P}\left(\| U \times_3 (\Delta *_s V) \| \leq \frac{\eta\beta}{\sqrt{l_1' l_2'}} \|A\|_F \|U\|_F \|V\|_F \right) \geq 1 - \delta \quad (\star). \tag{5}$$

```
**Algorithm 8** $(A, \epsilon, \eta, n'_1 \times n'_2))$
Require: Convolution Tensor $A \in \mathbb{R}^{l' \times l \times \kappa \times \kappa}$, error parameters $\epsilon, \eta$.
Ensure: Generate $n'_1 \times n'_2$ different tensors $\hat{A}_{(i,j)}$ $((i,j) \in [n'_1] \times [n'_2])$ that satisfy (5).
  Let $k = \frac{Q\lceil \frac{\kappa}{s} \rceil^2 (\log(\frac{1}{\eta}))^2}{\epsilon^2}$ for a large enough universal constant $Q$.
  Let $p = \log\left(\frac{1}{\eta}\right)$.
  Sample a uniformly random subspace $\mathcal{S}$ of $l' \times l \times \kappa \times \kappa$ of dimension $k \times p$.
  for $(i,j) \in [n'_1] \times [n'_2]$ do
      Sample $k$ matrices $M_1, \ldots, M_k \in \mathcal{N}(0,1)^{l' \times l \times \kappa \times \kappa}$ with random $\mathrm{i.i.d}$ entries.
      for $k' = 1 \to k$ do
          Let $M'_{k'} = \sqrt{\frac{ll'\kappa^2}{kp}} \mathrm{Proj}_{\mathcal{S}}(M_{k'})$.
          Let $Z_{k'} = \langle A, M'_{k'} \rangle M'_{k'}$.
      end for
      Let $\hat{A}_{(i,j)} = \frac{1}{k} \sum_{k'=1}^{k} Z_{k'}$.
  end for
```

Algorithm 8 is designed to generate different compressed filters $\hat{A}_{i,j}$ in a way that keeps the total number of parameters small, but also ensures that the $\hat{A}_{i,j}$'s behave similarly to the compressed filters that would generated if Algorithm 3 were applied to each location independently.

Theorem 6.3. *For any convolutional neural network $h_{\mathbf{w}}$ with $\rho_\delta \geq 3d$, any probability $0 < \delta \leq 1$ and any margin γ, then Algorithm 8 generates weights $\tilde{\mathbf{w}}$ for the network $h_{\tilde{\mathbf{w}}}$ such that*

$$\mathbb{P}_{S \sim \mathcal{D}^m}\left(L_0(h_{\tilde{\mathbf{w}}}) \leq \hat{L}_\gamma(h_{\mathbf{w}}) + \tilde{O}\left(\sqrt{\frac{c^2 d^2 \max_{(x,y)\in S} \|h_{\mathbf{w}}(x)\|_2^2 \sum_{i=1}^d \frac{\beta^2 \left(\lceil \frac{\kappa_i}{s_i} \rceil\right)^2}{\mu_i^2 \mu_{i\to}^2}}{\gamma^2 m}}\right)\right) \geq 1 - \delta,$$

where $\mu_i, \mu_{i\to}, c, \rho_\delta$ and β are layer cushion, inter-layer cushion, activation contraction, inter-layer smoothness and well-distributed Jacobian respectively. Furthermore, κ_i and s_i are the filter and stride in layer i.

Corollary 6.4. *For any convolutional neural network $h_{\mathbf{w}}$ with $\rho_\delta \geq 3d$, any probability $0 < \delta \leq 1$ and any margin γ, then Algorithm 8 generates weights $\tilde{\mathbf{w}}$ for the network $h_{\tilde{\mathbf{w}}}$ such that*

$$\mathbb{P}_{S \sim \mathcal{D}^m}\left(L_0(h_{\tilde{\mathbf{w}}}) \leq \hat{L}_\gamma(h_{\mathbf{w}}) + \zeta + \tilde{O}\left(\sqrt{\frac{c^2 d^2 \max_{(x,y)\in S} \|h_{\mathbf{w}}(x)\|_2^2 \sum_{i=1}^d \frac{\beta^2 \left(\lceil \frac{\kappa_i}{s_i} \rceil\right)^2}{\mu_i^2 \mu_{i\to}^2}}{\gamma^2 m}}\right)\right) \geq 1 - \delta,$$

where $\mu_i, \mu_{i\to}, c, \rho_\delta$ and β are layer cushion, inter-layer cushion, activation contraction, inter-layer smoothness and well-distributed Jacobian respectively measured on a $1 - \zeta$ fraction of the training set S. Furthermore, κ_i and s_i are the filter and stride in layer i.

6.2 Current State of the Art PAC-Bayes Bounds

We have seen that PAC-Bayes bounds provide a theoretical perspective on the learning process and the consequences it has on the performance of the learned classifier. In practice, we would ideally want these bounds to be meaningful. When implemented naively they produce vacuous bounds that provide no information. The first implementation of non-vacuous PAC-Bayes was discussed in Section 3.2 with the work [8] that focused on a particular setting to get the non-vacuous bounds. Since then there have been directed efforts to improve the tightness of these bounds and extend the success to different contexts. Currently, the tightest bounds seen in practice come from the work of [14]. In this section, we will discuss the work and see how it is a development of some previous work we have discussed. The work of [14] is an extension of the work of [11] and follows the same compression paradigm that was first considered by [9]. In [14] the tighter generalization bounds are achieved by first restricting to lower-dimensional settings using a notion called intrinsic dimensionality. Then they develop more aggressive quantization schemes that are adapted to the problem at hand.

6.2.1 The PAC-Bayes Foundations

Throughout this section, we will adopt the same notation as the rest of this report. Consider Theorem 2.1, the $\log(M)$ term counts the number of bits needed to specify any hypothesis $h_{\mathbf{w}}$ with $\mathbf{w} \in \mathcal{W}$, supposing that we assume each hypothesis is equally likely. If instead we have some prior belief on the likely hypotheses we can construct a variable length code that uses fewer bits to specify those hypotheses. For a prior distribution π, then for any $\mathbf{w} \in \mathcal{W}$ the number of bits required to represent hypothesis $h_{\mathbf{w}}$ is $\log_2\left(\frac{1}{\pi(\mathbf{w})}\right)$ when using an optimal compression code for π. Furthermore, if we consider a set of good distributions ρ and we do not care which element of Q we arrive at, we can gain some bits back. In particular, the average number of bits to code a sample from ρ using π is the cross entropy $H(\rho, \pi)$ and since we are agnostic to the sample from ρ we get back $H(\rho)$ bits. Therefore, the average number of bits is

$$H(\rho, \pi) - H(\rho) = \mathrm{KL}(\rho, \pi).$$

Definition 6.5. *For probability measures ρ and π on a state space $\mathcal{X}$ that are absolutely continuous with respect to some measure λ, then*

$$H(\rho, \pi) = \int_{\mathcal{X}} \rho(x) \log\left(\pi(x)\right) d\lambda(x),$$

where $H(\rho) := H(\rho, \rho)$.

With these improvements, we can get bounds such as Theorem 3.10. For this work, we will work with Theorem 5.5 to get the generalization bounds. The prior that we will use here is known as the universal prior and explicitly penalizes the minimum compressed length of the hypothesis,

$$\pi(\mathbf{w}) = \frac{2^{-K(\mathbf{w})}}{Z}.$$

Then using a point mass posterior on the single parameter $\mathbf{w}^*$ we get that

$$\mathrm{KL}\left(\mathbf{I}_{\{\mathbf{w}=\mathbf{w}^*\}}, \pi\right) = \log\left(\frac{1}{\pi(\mathbf{w}^*)}\right) \leq K\left(\mathbf{w}^*\right)\log(2) \leq l(\mathbf{w}^*)\log(2) + 2\log\left(l(\mathbf{w}^*)\right),$$

where $l(\mathbf{w})$ is the length of the program that reproduces $\mathbf{w}$. Improving the tightness of our bounds comes about by reducing the compressed length $l(\mathbf{w}^*)$ for the $\mathbf{w}^*$ achieved through training. For this work, the method for model compression consists of two components. One component is reducing the dimensionality of the problem, and the second is developing an aggressive quantization scheme.

6.2.2 Finding Random Subspaces

A neural network parameterized by the weight vector $\mathbf{w} \in \mathbb{R}^D$ is often optimized through gradient descent so that the updates occur in the D-dimensional loss landscape. Despite D being very large the optimization process is relatively stable and converges to simple solutions. However, we can work in a reduced dimension $d < D$ (referred to as the intrinsic dimension) by considering

$$\mathbf{w} = \mathbf{w}_0 + P\hat{\mathbf{w}},$$

where $\mathbf{w}_0 \in \mathbb{R}^D$ is the initialized weight, $P \in \mathbb{R}^{D \times d}$ is such that $P^\top P \approx I_{d\times d}$ and $\hat{\mathbf{w}} \in \mathbb{R}^d$. Now the vector $\hat{\mathbf{w}}$ is optimized so that the updates take place on a d-dimensional landscape. Finding the smallest value of d for which we still attain good performance on the problem at hand is the bottleneck to this approach. The work lies in finding projection P that is stable under training and finding optimal subspaces in which to optimize in. Imposing the condition $P^\top P \approx I_{d\times d}$ solves the first concern, for the next, we consider three possible methods for constructing P.

1. Kronecker Sum Projector: Construct the matrix

   $$P_\oplus = \frac{\mathbf{1}\otimes R_1 + R_2 \otimes \mathbf{1}}{\sqrt{2D}}$$

   where $\otimes$ is the Kronecker product, $R_1, R_2 \sim \mathcal{N}(0,1)^{\sqrt{D}\times d}$. Note that $P_\oplus^\top P_\oplus = I_{d\times d} + O\left(\frac{1}{\sqrt{D}}\right)$. Applying this to a vector $\hat{\mathbf{w}} \in \mathbb{R}^d$ takes $O\left(d\sqrt{D}\right)$ time.

2. Kronecker Product Projector: Construct the matrix

   $$P_\otimes = \frac{Q_1 \otimes Q_2}{\sqrt{D}}$$

   where $Q_1, Q_2 \sim \mathcal{N}(0,1)^{\sqrt{D}\times\sqrt{d}}$. Note that $P_\otimes^\top P_\otimes = I_{d\times d} + O\left(\frac{1}{D^{\frac{1}{4}}}\right)$. Applying this to a vector $\hat{\mathbf{w}} \in \mathbb{R}^d$ takes $O\left(\sqrt{dD}\right)$ time.

6.2.3 Quantization and Training

For the full precision weight vector $\mathbf{w} = (w_1, \dots, w_d) \in \mathbb{R}^d$ and vector $c = (c_1, \dots, c_L) \in \mathbb{R}^L$ of L quantization levels, construct the quantized vector $\tilde{\mathbf{w}} \in \mathbb{R}^d$ where $\tilde{w}_i = c_{q(i)}$ where $q(i) = \mathrm{argmin}_k |w_i - c_k|$. The vector c is learned alongside $\mathbf{w}$ where the gradients are defined as

$$\frac{\partial \tilde{w}_i}{\partial w_j} = \delta_{ij}, \text{ and } \frac{\partial \tilde{w}_i}{\partial c_k} = \mathbf{I}_{q(i)=k}.$$

c is initialized to have uniform spacing between the minimum and maximum values of $\mathbf{w}$, or using k-means. The latter approach refers to a quantization scheme proposed in [7] where for $k = L$ we partition the weights into clusters $\mathcal{C}_1, \dots, \mathcal{C}_k$ with $c_1, \dots, c_k$ such that

$$\mathrm{argmin}_{\mathcal{C}_1,\dots,\mathcal{C}_k} \left(\sum_{i=1}^{k} \sum_{w\in\mathcal{C}_i} |w_i - c_i|^2 \right), \quad \text{for } c_i = \frac{1}{|\mathcal{C}_i|} \sum_{w\in\mathcal{C}_i} w.$$

Next, we capitalize on the fact that certain quantization levels will be more likely than others to introduce a variable length coding scheme. For each level c_k associate the probability p_k and apply arithmetic coding. Each arithmetic coding of $\mathbf{w}$ takes at most $\lceil d \times H(p) \rceil$ bits, where p is the discrete distribution of the p_k's. Considering the total number of bits we see that

$$l(\mathbf{w}) \leq \lceil d \times H(p) \rceil + L \times (16 + \lceil \log_2(d) \rceil) + 2$$

as $L \times \lceil \log_2(d) \rceil$ bits are required for the probabilities p_k and $16L$ bites for the codebook c.

6.2.4 Optimization

Note that the smaller the intrinsic dimension d is the closer that our trained weight will be to the initialized weight $\mathbf{w}_0$. Therefore, $\mathbf{w}_0$ is more likely under our universal prior. Recall, that we must therefore condition on $\mathbf{w}_0$ to generate our prior. Similarly, if we optimize over different hyper-parameters such as d, L or the learning rate (η), we must encode these into the prior and pay a penalty for optimizing over them. To do this we simply redefine our weight vector to be $\mathbf{w}' = (\mathbf{w}, d, L, \eta)$ and so our prior becomes

$$\pi\left(\mathbf{w}'\right) = \frac{2^{-K\left(\mathbf{w}'\right)}}{Z},$$

where now we have that

$$K\left(\mathbf{w}'\right) \leq K(\mathbf{w}|d, L) + K(d) + K(L) + K(\eta).$$

Typically, we optimize these hyper-parameters over finite sets and so we can bound these terms by the ceiling of $\log_2$ of the size of these sets. The process we have discussed can be summarized in Algorithm 9.

Algorithm 9 Compute PAC-Bayes Compression Bound

```
Require: Neural network $h_{\mathbf{w}}$, training data set $S = \{(x_i, y_i)\}_{i=1}^{m}$, Clusters $L$, Intrinsic Dimension $d$, Confidence $1 - \delta$, Prior distribution $\pi$.
  function COMPUTEBOUND($h_{\mathbf{w}}, L, d, S, \delta, \pi$)
      $\mathbf{w} \leftarrow \mathrm{TRAINID}(h_{\mathbf{w}}, d, S)$
      $\tilde{\mathbf{w}} \leftarrow \mathrm{TRAINQUANTIZE}\left(h_{\mathbf{w}}, L, S\right).$
      Compute quantized empirical risk $\hat{R}(\tilde{\mathbf{w}})$.
      $\mathrm{KL}(\rho, \pi) \leftarrow \mathrm{GETKL}\left(\tilde{w}, \pi\right).$
      return $\mathrm{GETCATONIBOUND}\left(\hat{R}(\tilde{w}), \mathrm{KL}(\rho, \pi), \delta, m\right)$
  end function
  function TRAINQUANTIZE($\mathbf{w}, L, S$)
      Initialize $c \leftarrow \mathrm{GETCLUSTERS}\left(\mathbf{w}, L\right).$
      for $i = 1 \rightarrow \mathrm{quantepochs}$ do
          $\begin{pmatrix} c \\ \mathbf{w} \end{pmatrix} \leftarrow \begin{pmatrix} c - \eta \nabla_c \mathcal{L}\left(\mathbf{w}, c\right) \\ \mathbf{w} - \eta \nabla_{\mathbf{w}} \mathcal{L}(\mathbf{w}, c) \end{pmatrix}.$
      end for
      return $\tilde{w}$
  end function
  function GETKL($\tilde{\mathbf{w}}, \pi$)
      $c, \mathrm{count} \leftarrow \mathrm{GETUNIQUEVALSCOUNTS}\left(\tilde{\mathbf{w}}\right).$
      $\mathrm{messagesize} \leftarrow \mathrm{DOARITHMETICENCODING}\left(\tilde{w}, c, \mathrm{count}\right).$
      $\mathrm{messagesize} \leftarrow \mathrm{messagesize} + \mathrm{hyperparamsearch}$
      return $\mathrm{messagesize} + 2 \times \log\left(\mathrm{messagesize}\right).$
  end function
```

References

[1] David A. McAllester. “PAC-Bayesian model averaging”. In: *Annual Conference Computational Learning Theory*. 1999.

[2] John Langford and Matthias Seeger. “Bounds for Averaging Classifiers”. In: (Feb. 2001).

[3] Andreas Maurer. “A Note on the PAC Bayesian Theorem”. In: *CoRR* (2004).

[4] Olivier Catoni. “Pac-Bayesian Supervised Classification: The Thermodynamics of Statistical Learning”. In: *IMS Lecture Notes Monograph Series* 56 (2007), pp. 1–163.

[5] Olivier Catoni. “A PAC-Bayesian approach to adaptive classification”. In: (Jan. 2009).

[6] David A. McAllester. “A PAC-Bayesian Tutorial with A Dropout Bound”. In: *CoRR* (2013).

[7] Yoojin Choi, Mostafa El-Khamy, and Jungwon Lee. *Towards the Limit of Network Quantization*. 2017.

[8] Gintare Karolina Dziugaite and Daniel M. Roy. “Computing Nonvacuous Generalization Bounds for Deep (Stochastic) Neural Networks with Many More Parameters than Training Data”. In: *CoRR* (2017).

[9] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang. “Stronger generalization bounds for deep nets via a compression approach”. In: *CoRR* (2018).

[10] Benjamin Guedj. *A Primer on PAC-Bayesian Learning*. 2019.

[11] Wenda Zhou, Victor Veitch, Morgane Austern, Ryan P. Adams, and Peter Orbanz. *Non-Vacuous Generalization Bounds at the ImageNet Scale: A PAC-Bayesian Compression Approach*. 2019.

[12] Gintare Karolina Dziugaite, Kyle Hsu, Waseem Gharbieh, and Daniel M. Roy. “On the role of data in PAC-Bayes bounds”. In: *CoRR* (2020).

[13] Paul Viallard, Pascal Germain, Amaury Habrard, and Emilie Morvant. *A General Framework for the Disintegration of PAC-Bayesian Bounds*. 2021.

[14] Sanae Lotfi, Marc Finzi, Sanyam Kapoor, Andres Potapczynski, Micah Goldblum, and Andrew Gordon Wilson. *PAC-Bayes Compression Bounds So Tight That They Can Explain Generalization*. 2022.

[15] Pierre Alquier. *User-friendly introduction to PAC-Bayes bounds*. 2023.