6. APPENDIX

6.1. Learning Algorithm

Consider a learning problem specified by a function/hypothesis class $\mathcal{H}$, an instance set $\mathcal{X}$ with diameter at most B, and a evaluation function $o : \mathcal{H} \times \mathcal{X} \to \mathbb{R}$ which is bounded by a constant C. Given a distribution P_x defined on $\mathcal{X}$, the quality of a function $h(x)$ is measured by its expected evaluation statistic $F(P_x, h) = \mathbb{E}_{x\sim P_x}[o(h, x)]$. Since P_x is unknown, we need to rely on a finite training sample $S = \{x_1, \ldots, x_m\} \subset \mathcal{X}$ and often work with the empirical evaluation $F\left(\widehat{P}_x, h\right) = \mathbb{E}_{x\sim \widehat{P}_x}[o(h, x)] = \frac{1}{m}\sum_{x\in S} o(h, x)$, where $\widehat{P}_x$ is the empirical distribution defined on S. A learning algorithm $\mathcal{A}$ will pick a function $h_m \in \mathcal{H}$ based on input S, i.e., $h_m = \mathcal{A}(\mathcal{H}, S)$. Let $\mathcal{X} = \bigcup_{i=1}^{N} \mathcal{X}_i$ be a partition of $\mathcal{X}$ into N disjoint subsets. We use the following definition about robustness of an algorithm.

Definition 4 (Robustness)**.** An algorithm $\mathcal{A}$ is (N, ϵ)-robust, for $\epsilon : \mathcal{X}^m \to \mathbb{R}$, if the following holds for all $S \in \mathcal{X}^m$: $\forall s \in S, \forall x \in \mathcal{X}, \forall i \in 1, \ldots, N$, if $s, x \in \mathcal{X}_i$ then $|o(\mathcal{A}(\mathcal{H}, S), s) - o(\mathcal{A}(\mathcal{H}, S), x)| \leq \epsilon(S)$.

Basically, a robust algorithm will be a hypothesis which ensures that the evaluation difference of two similar data instances should be the same. A small change in the input leads to a small change in the output of the given hypothesis. In other words, the robustness ensures that each testing sample which is close to the training dataset will have a similar evaluation with that of the closest training samples. Therefore, the hypothesis $\mathcal{A}(S)$ will generalise well over the areas around S.

Theorem 4 ([24])**.** *If a learning algorithm $\mathcal{A}$ is (N, ϵ)-robust and the training data S is an i.i.d. sample from distribution P_x, then for any $\delta \in (0, 1]$ we have the following with probability at least $1-\delta$: $\left|F\left(P_x, \mathcal{A}(\mathcal{H}, S)\right) - F\left(\widehat{P}_x, \mathcal{A}(\mathcal{H}, S)\right)\right| \leq \epsilon(S) + C\sqrt{(N \log 4 - 2\log\delta)/m}$.*

This theorem formally makes the important connection between robustness of an algorithm and generalisation. If an algorithm is robust, then its resulting hypotheses can generalise. One important implication of this result is that we should ensure the robustness of a learning algorithm. However, it is nontrivial to do so in general.

Let us have a closer look at robustness. $\epsilon(S)$ in fact bounds the amount of change in the loss with respect to a change in the input given a fixed hypothesis. This observation suggests that robustness closely resembles the concept of Lipschitz continuity.

Definition 5 (Lipschitz Continuity)**.** Given two metric spaces (X, dX) and (Y, dY), where dX and dY are the metrics on the sets X and Y respectively, a function $f : X \to Y$ is called Lipschitz continuous if there exists a real constant $K \geq 0$ such that, for all $x_1, x_2 \in X$:

$$d_Y\left(f\left(x_1\right), f\left(x_2\right)\right) \leq K d_X\left(x_1, x_2\right) \tag{6}$$

K is called the Lipschitz constant for the function f.

Therefore, we establish the following connection between robustness and Lipschitz continuity.

Lemma 5. *Given any constant $\lambda > 0$, consider a function $f : \mathcal{H} \times \mathcal{X} \to \mathbb{R}$, where $\mathcal{X} \subset \mathbb{R}^n$ is compact, $B = \operatorname{diam}(\mathcal{X}) = \max_{x,x'\in\mathcal{X}} \|x - x'\|_\infty$, $N = \lceil B^{n_x}\lambda^{-n_x}\rceil$. If for any $h \in \mathcal{H}$, $o(h, x)$ is K-Lipschitz continuous w.r.t input x, then any algorithm $\mathcal{A}$ that maps $\mathcal{X}^m$ to $\mathcal{H}$ is $(N, L\lambda)$-robust.*

Combining Theorem 4 and Lemma 5, we make the following connection between Lipschitz continuity and generalization.

Theorem 6. *If a loss $o(h, x)$ is K-Lipschitz continuous w.r.t input x in a compact set $\mathcal{X} \subset \mathbb{R}^n$, for any $h \in \mathcal{H}$, and $\widehat{P}_x$ is the empirical distribution defined from m i.i.d. samples from distribution P_x, then $\sup_{h\in\mathcal{H}} \left|F\left(P_x, h\right) - F\left(\widehat{P}_x, h\right)\right|$ is upper-bounded by*

1. *$L\lambda + C\sqrt{(\lceil B^{n_x}\lambda^{-n_x}\rceil \log 4 - 2\log\delta)/m}$ with probability at least $1-\delta$, for any constants $\delta \in (0, 1]$ and $\lambda \in (0, B]$.*
2. *$(LB + 2C)m^{-\alpha/n}$ with probability at least $1 - 2\exp\left(-0.5m^{\alpha}\right)$, for any $\alpha \leq n/(2+n)$.*

6.2. Generative Models

From a statistical standpoint, let x denote the observable variable and let y represent the corresponding label. The learning objective for a generative model is to model the conditional probability distribution $P(x|y)$.

Among all generative models, generative adversarial networks (GANs) have garnered considerable attention in recent years due to their ability to generate realistic and indistinguishable high-quality images. Fundamentally, GANs consist of two neural networks—the discriminator and the generator G—and are trained to enable G to generate elements that mimic a target true data distribution P_d, even in the simplest case without sample labels. Given a training dataset of real-world data samples, the generator aims to capture the true data distribution, while the discriminator strives to discern whether the data samples originate from the generator or real data. The objective of the generator is to minimize the divergence between the discriminator's outputs for true versus generated samples, while the objective of the discriminator is to accurately classify the true versus fake samples. With certain enhancements, GANs may also be capable of reconstruction—finding a latent representation z for a given original vector $x \in \mathcal{X}$ such that $G(z)$ closely approximates x (e.g., according to some norm in the original space).

Furthermore, an autoencoder ($\mathcal{N}^E$, $\mathcal{N}^D$), where $\mathcal{N}^E$ and $\mathcal{N}^D$ are feed-forward neural networks denoted as the encoder and the decoder respectively, is another notable generative model. Its goal is to compress (encode) its inputs $x \in \mathcal{X}$ into low-dimensional latent vectors z, such that approximate decompression (decoding, reconstruction) can be achieved: $\mathcal{N}^D(z)$ closely approximates x. A generative autoencoder, such as those in [25, 26], is an autoencoder whose decoder is additionally trained to sample from the original distribution —effectively, a generative autoencoder performs both the tasks of an autoencoder and a GAN.

In addition to GANs and autoencoders, diffusion models (DMs) are also gaining traction. DMs consist of two stages: the forward diffusion process and the reverse diffusion process. In the forward process, the input data is gradually perturbed by Gaussian noises until it eventually resembles an isotropic Gaussian distribution. In the reverse process, DMs reverse the forward process, implementing a sampling process from Gaussian noises to reconstruct the true samples.

In summary, generative models are capable of data generation from low-dimensional vectors. Data generation is achieved by applying G to a low-dimensional vector $z \in \mathbb{R}^d$ sampled from the latent code distribution $\mathcal{Z}$ (typically, $\mathcal{N}(0, I)$). If $z \sim \mathcal{Z}$, then for a well-trained generator G, we may assume that $G(z) \sim \mathcal{X}$. Often, the dimension of $\mathcal{Z}$ is made smaller than the dimension of $\mathcal{X}$.