UCLA
UCLA Electronic Theses and Dissertations

Title
Building Trustworthy Machine Learning Models

Permalink
https://escholarship.org/uc/item/1w7948xc

Author
Liu, Xuanqing

Publication Date
2021

Peer reviewed|Thesis/dissertation

UNIVERSITY OF CALIFORNIA

Los Angeles

Building Trustworthy Machine Learning Models

A dissertation submitted in partial satisfaction

of the requirements for the degree

Doctor of Philosophy in Computer Science

by

Xuanqing Liu

2021

ABSTRACT OF THE DISSERTATION

Building Trustworthy Machine Learning Models

by

Xuanqing Liu

Doctor of Philosophy in Computer Science

University of California, Los Angeles, 2021

Professor Cho-Jui Hsieh, Chair

How and when can we depend on machine learning systems to make decisions for human-being? This is probably the question everybody may (and should) ask before deploying machine learning models in their own fields. Failure to do so can suffer from unexpected consequences: the text recognition systems in the mail distribution center may send the package to the wrong addresses; the self-driving cars may recognize a stop sign as a speed sign; or even worse, the AI-based medical imaging system may mislead the doctors into wrong diagnostics. We attribute a trustworthy machine learning model to three properties: robustness, interpretation, and precise uncertainty estimation. Robustness concerns how the model withstands unexpected inputs, also called out-of-distribution (OOD) data. Depending on whether the data is maneuvered in purpose, the OOD data comprises adversarial examples or unadversarial examples. Interpretation is a set of algorithms that uncover the black-box model inference process, trying to help humans understand why or why not the model generates the desired results. Finally, we seek the uncertainty estimation tools to locate the ground-truth value relative to the estimated value. It also protects the model users by holding the machine predictions for human inspections once the uncertainties rise above some

threshold.

In this thesis, I will walk through robustness, interpretation, and uncertainty estimation in three parts. In the first part, I will introduce the backgrounds of robust machine learning models with an example in graph-based semi-supervised learning, followed by a series of methods to train robust neural networks. In the next part, we will move to model interpretation tools, and we relate this part to the last part by discussing our work called Greedy-AS. In the final part, I will discuss my robust uncertainty estimation and confidence calibration. This part contains the algorithms, software packages, and a demo on how uncertainty estimation helps biological scientists to do quality control of stem cells more efficiently.

The dissertation of Xuanqing Liu is approved.

Yen-Chih Lin

Quanquan Gu

Stanley J. Osher

Wei Wang

Cho-Jui Hsieh, Committee Chair

University of California, Los Angeles

2021

To my parents

TABLE OF CONTENTS

LIST OF TABLES

ACKNOWLEDGMENTS

Many people helped me during my Ph.D. study. But first of all, I want to express my deepest gratitude to my Ph.D. advisor, Dr. Cho-Jui Hsieh, professor of the computer science department at UCLA. His dedication and keen intellect in this field and his incredible patience to help graduate students have greatly influenced my research in the past few years. I am also grateful for the valuable opportunities connecting me to some of the most talented researchers in the machine learning community. I am sure the experiences working with him are going to become the greatest wealth in my life.

I want to express the deepest sense of gratitude to my former collaborators and mentors in internships, including Dr. Jason Lee, professor at Princeton University, Dr. Yuekai Sun, professor at the University of Michigan, Ann Arbor, Dr. Sathiya S. Keerthi and Dr. Zhengming Xing, now a research scientist at LinkedIn, Dr. Jerry Zhu, professor at the University of Wisconsin-Madison, Dr. Neil Lin, professor at UCLA, Dr. Inderjit S. Dhillon, professor at UT Austin, Dr. Si Si, a research scientist at Google, and Dr. Hsiang-Fu Yu, a research scientist at Amazon. Without your insightful ideas and timely suggestions with kindness, I wouldn't be able to complete these interesting projects!

I am proud to have my friends who are also my student collaborators, Dr. Huan Zhang, now Postdoc at CMU, Dr. Minhao Cheng, now professor at HKUST, Dr. Yao Li, now professor at the University of Carolina Chapel Hill, Dr. Lu Wang at JD research, Dr. Wei-Cheng Chang at Amazon; as well as Patrick Chen at UCLA, Wei-Lin Chiang at UC Berkeley, Chongruo Wu at UC Davis. It is a pleasure working with all of you!

Finally, it is my privilege to thank Yanqiu Zhou at Cornell University and my parents, for patiently dealing with my frustration in thesis writting, for never giving up in supporting and encouraging me, for filling my life with joy and happiness.

VITA

2011–2016	Bachelor of Science in Physics, Peking University. Beijing, China.
2016–2018	Research Assistant in Computer Science Department, UC Davis, Davis, CA.
2016–2018	Teaching Assistant in Computer Science Department, UC Davis, Davis, CA.
2018–2021	Research Assistant in Computer Science Department, UCLA, Los Angeles, CA.
2018–2021	Teaching Assistant in Computer Science Department, UCLA, Los Angeles, CA.

Part I

Robustness of Deep Learning Models

CHAPTER 1

Adversarial and Unadversarial Robustness

1.1 Introduction

Deep neural networks have been proven to be successful for making decisions out of complex, multimodality data; such as image recognition [KSH12], semantic segmentation [LSD15], instance segmentation [HGD17], object detection [LDG17], text understanding [WSM18], etc. However, a series of researches have revealed that the good prediction abilities quickly disappeared under evolving conditions. In literature, there are multiple lines of research to study the robustness issue. Here we mainly focus on two cases: **adversarial inputs** and **out of distribution data**, which are two of the most representative instances of adversarial robustness and unadversarial robustness.

Adversarial input   In this case, there is a malicious algorithm sitting between the input data and the prediction models. The input data can be both training data and the test data – if the training data is changed by the algorithm, it is often called data poisoning attack; and if the test data is modified, it is called adversarial attack. Notice that depending on the problem settings, the attacking algorithm can have a full spectrum of capabilities regarding to the prediction models, which is not always achievable in practice.

Unadversarial robustness   This scenario is more frequently encountered in practice, because the data modification mechanism is not dedicatedly designed to ruin the models, but some natural but unpredictable noises. Examples include foggy or raining days for self-driving

car systems, unexpectedly long sentences for language translators, or even spelling errors in the user inputs.

In the remaining sections of this part, we will first study a concrete case of data poisoning attack in graph based SSL. Then we will move to some techniques to promote the adversarial robustness in deep neural networks.

1.2 Case study: Data Poisoning Attack in Graph-based SSL

1.2.1 Problem setting

We consider the graph-based semi-supervised learning (G-SSL) problem. The input include labeled data $\boldsymbol{X}_l \in \mathbb{R}^{n_l \times d}$ and unlabeled data $\boldsymbol{X}_u \in \mathbb{R}^{n_u \times d}$, we define the whole features $\boldsymbol{X} = [\boldsymbol{X}_l; \boldsymbol{X}_u]$. Denoting the labels of $\boldsymbol{X}_l$ as $\boldsymbol{y}_l$, our goal is to predict the labels of test data $\boldsymbol{y}_u$. The learner applies algorithm $\mathcal{A}$ to predict $\boldsymbol{y}_u$ from available data $\{\boldsymbol{X}_l, \boldsymbol{y}_l, \boldsymbol{X}_u\}$. Here we restrict $\mathcal{A}$ to label propagation method, where we first generate a graph with adjacency matrix $\boldsymbol{S}$ from Gaussian kernel: $\boldsymbol{S}_{ij} = \exp(-\gamma \|\boldsymbol{x}_i - \boldsymbol{x}_j\|^2)$, where the subscripts $\boldsymbol{x}_{i(j)}$ represents the $i(j)$-th row of $\boldsymbol{X}$. Then the graph Laplacian is calculated by $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{S}$, where $\boldsymbol{D} = \mathrm{diag}\{\sum_{k=1}^n \boldsymbol{S}_{ik}\}$ is the degree matrix. The unlabeled data is then predicted through energy minimization principle [ZGL03]

$$\min_{\hat{\boldsymbol{y}}} \frac{1}{2} \sum_{i,j} \boldsymbol{S}_{ij}(\hat{\boldsymbol{y}}_i - \hat{\boldsymbol{y}}_j)^2 = \hat{\boldsymbol{y}}^\mathsf{T} \boldsymbol{L} \hat{\boldsymbol{y}}, \quad \text{s.t.} \quad \hat{\boldsymbol{y}}_{:l} = \boldsymbol{y}_l. \tag{1.1}$$

The problem has a simple closed form solution $\hat{\boldsymbol{y}}_u = (\boldsymbol{D}_{uu} - \boldsymbol{S}_{uu})^{-1}\boldsymbol{S}_{ul}\boldsymbol{y}_l$, where we define $\boldsymbol{D}_{uu} = \boldsymbol{D}_{[0:u,0:u]}$, $\boldsymbol{S}_{uu} = \boldsymbol{S}_{[0:u,0:u]}$ and $\boldsymbol{S}_{ul} = \boldsymbol{S}_{[0:u,0:l]}$. Now we consider the attacker who wants to greatly change the prediction result $\boldsymbol{y}_u$ by perturbing the *training data* $\{\boldsymbol{X}_l, \boldsymbol{y}_l\}$ by small amounts $\{\Delta_x, \boldsymbol{\delta}_y\}$ respectively, where $\Delta_x \in \mathbb{R}^{n_l \times d}$ is the perturbation matrix , and $\boldsymbol{\delta}_y \in \mathbb{R}^{n_l}$ is a vector. This seems to be a simple problem at the first glance, however, we will show that the problem of finding optimal perturbation is often intractable, and therefore provable and effective algorithms are needed. To sum up, the problem have several degrees of freedom:

- **Learning algorithm**: Among all graph-based semi-supervised learning algorithms, we primarily focus on the label propagation method; however, we also discuss manifold regularization method in Appendix 4.2.

- **Task**: We should treat the regression task and classification task differently because the former is inherently a continuous optimization problem while the latter can be transformed into integer programming.

- **Knowledge of attacker**: Ideally, the attacker knows every aspect of the victim, including training data, testing data, and training algorithms. However, we will also discuss incomplete knowledge scenario; for example, the attacker may not know the exact value of hyper-parameters.

- **What to perturb**: We assume the attacker can perturb the label or the feature, but not both. We made this assumption to simplify our discussion and should not affect our findings.

- **Constraints**: We also assume the attacker has limited capability, so that (s)he can only make small perturbations. It could be measured ℓ_2-norm or sparsity.

1.2.2   Toy example

We show a toy example in Figure 1.1 to motivate the data poisoning attack to graph semi-supervised learning (let us focus on label propagation in this toy example). In this example, the shaded region is very close to node-i and yet quite far from other labeled nodes. After running label propagation, all nodes inside the shaded area will be predicted to be the same label as node-i. That gives the attacker a chance to manipulate the decision of all unlabeled nodes in the shaded area at the cost of flipping just one node. For example, in Figure 1.1, if we change node-i's label from positive to negative, the predictions in the shaded area containing three nodes will also change from positive to negative.
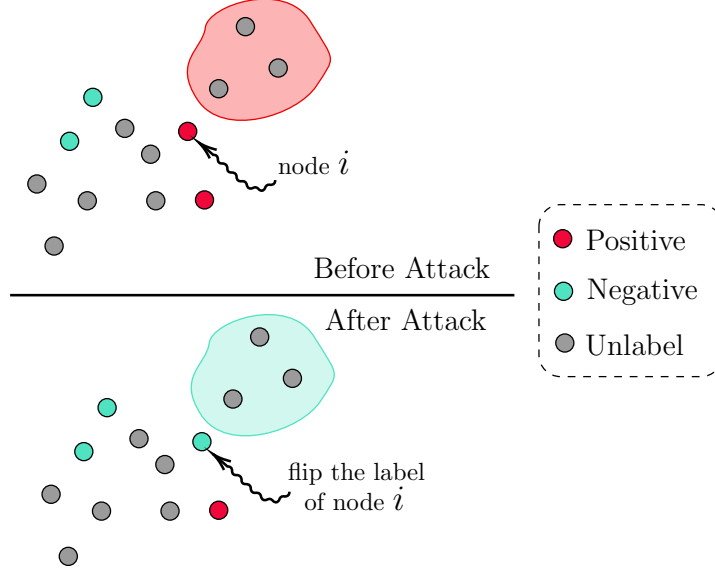
Figure 1.1: We show a toy example that illustrates the main idea of the poisoning attack against SSL. By flipping just one training data from positive to negative, the prediction of the whole shaded area will be changed.

Besides changing the labels, another way to attack is to perturb the features $\boldsymbol{X}$ so that the graph structure $\boldsymbol{S}$ changes subtly (recall the graph structure is constructed based on pair-wise distances). For instance, we can change the features so that node i is moved away from the shaded region, while more negative label points are moved towards the shaded area. Then with label propagation, the labels of the shaded region will be changed from positive to negative as well. We will examine both cases in the following sections.

1.2.3 A unified framework

The goal of poisoning attack is to modify the data points to maximize the error rate (for classification) or RMSE score (for regression); thus we write the objective as

$$\min_{\substack{\boldsymbol{\delta}_y \in \mathcal{R}_1 \\ \Delta_x \in \mathcal{R}_2}} -\frac{1}{2}\left\| g\Big((\boldsymbol{D}'_{uu} - \boldsymbol{S}'_{uu})^{-1} \boldsymbol{S}'_{ul}(\boldsymbol{y}_l + \boldsymbol{\delta}_y) \Big) - h(\boldsymbol{y}_u) \right\|_2^2 \quad \text{s.t. } \{\boldsymbol{D}', \boldsymbol{S}'\} = \mathrm{Ker}_\gamma(\boldsymbol{X}_l + \Delta_x). \quad (1.2)$$

To see the flexibility of Eq. (1.2) in modeling different tasks, different knowledge levels of attackers or different budgets, we decompose it into following parts that are changeable in real applications:

- $\mathcal{R}_1/\mathcal{R}_2$ are the constraints on $\boldsymbol{\delta}_y$ and Δ_x. For example, $\mathcal{R}_1 = \{\|\boldsymbol{\delta}_y\|_2 \leq d_{\max}\}$ restricts the perturbation $\boldsymbol{\delta}_y$ to be no larger than $d_{\max}$; while $\mathcal{R}_1 = \{\|\boldsymbol{\delta}_y\|_0 \leq c_{\max}\}$ makes the solution to have at most $c_{\max}$ non-zeros. As to the choices of $\mathcal{R}_2$, besides ℓ_2 regularization, we can also enforce group sparsity structure, where each row of Δ_x could be all zeros.

- $g(\cdot)$ is the task dependent squeeze function, for classification task we set $g(x) = \text{sign}(x)$ since the labels are discrete and we evaluate the accuracy; for regression task it is identity function $g(x) = x$, and ℓ_2-loss is used.

- $h(\cdot)$ controls the knowledge of unlabeled data. If the adversary knows the ground truth very well, then we simply put $h(\boldsymbol{y}_u) = \boldsymbol{y}_u$; otherwise one has to estimate it from Eq. (1.1), in other words, $h(\boldsymbol{y}_u) = \hat{\boldsymbol{y}}_u = g\big((\boldsymbol{D}_{uu} - \boldsymbol{S}_{uu})^{-1}\boldsymbol{S}_{ul}\boldsymbol{y}_l\big)$.

- Ker_γ is the kernel function parameterized by γ, we choose Gaussian kernel throughout.

- Similar to $\boldsymbol{S}$, the new similarity matrix $\boldsymbol{S}'$ is generated by Gaussian kernel with parameter γ, except that it is now calculated upon poisoned data $\boldsymbol{X}_l + \Delta_x$.

- Although not included here, we can also formulate targeted poisoning attack problem by changing min to max and let $h(\boldsymbol{y}_u)$ be the target.

There are two obstacles to solving Eq. 1.2, that make our algorithms non-trivial. First, the problem is naturally *non-convex*, making it hard to determine whether a specific solution is globally optimal; secondly, in classification tasks where our goal is to maximize the testing time error rate, the objective is *non-differentiable* under *discrete* domain. Besides, even with hundreds of labeled data, the domain space can be unbearably big for brute force search and yet the greedy search is too myopic to find a good solution (as we will see in experiments).

In the next parts, we show how to tackle these two problems separately. Specifically, in the first part, we propose an efficient solver designed for data poisoning attack to the regression problem under various constraints. Then we proceed to solve the discrete, non-differentiable poisoning attack to the classification problem.

1.2.4 Regression task, (un)known label

Algorithm 1 Trust region problem solver

1: **Input:** Vector $\boldsymbol{g}$, symmetric indefinite matrix $\boldsymbol{H}$ for problem $\min_{\|\boldsymbol{z}\| \leq 1} \frac{1}{2} \boldsymbol{z}^\mathsf{T} \boldsymbol{H} \boldsymbol{z} + \boldsymbol{g}^\mathsf{T} \boldsymbol{z}$.

2: **Output:** Approximate solution $\boldsymbol{z}^*$

3: Initialize $\boldsymbol{z}_0 = -0.5 \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}$ and step size η

4: `// Phase I: iterate inside sphere` $\|\boldsymbol{z}_t\| < 1$

5: **while** $\|\boldsymbol{z}_t\| < 1$ **do**

6: $\quad \boldsymbol{z}_{t+1} = \boldsymbol{z}_t - \eta(\boldsymbol{H}\boldsymbol{z}_t + \boldsymbol{g})$

7: **end while**

8: `// Phase II: iterate on the sphere` $\|\boldsymbol{z}_t\| = 1$

9: $\boldsymbol{z}_{t'} = \boldsymbol{z}_t$

10: **while** $t < \text{max\_iter}$ **do**

11: $\quad$ Choose $\alpha_{t'}$ by line search and do the following projected gradient descent on sphere

12: $\quad \boldsymbol{z}_{t'+1} = \frac{\boldsymbol{z}_{t'} - \alpha_{t'}(\boldsymbol{I}_d - \boldsymbol{z}_{t'}\boldsymbol{z}_{t'}^\mathsf{T})(\boldsymbol{H}\boldsymbol{z}_{t'} + \boldsymbol{g})}{\|\boldsymbol{z}_{t'} - \alpha_{t'}(\boldsymbol{I}_d - \boldsymbol{z}_{t'}\boldsymbol{z}_{t'}^\mathsf{T})(\boldsymbol{H}\boldsymbol{z}_{t'} + \boldsymbol{g})\|}$

13: **end while**

14: **Return** $\boldsymbol{z}_{\text{max\_iter}}$

We first consider the regression task where only label poisoning is allowed. This simplifies Eq. (1.2) as

$$\min_{\|\boldsymbol{\delta}_y\|_2 \leq d_{\max}} \begin{cases} -\dfrac{1}{2} \left\| (\boldsymbol{D}_{uu} - \boldsymbol{S}_{uu})^{-1} \boldsymbol{S}_{ul} \boldsymbol{\delta}_y \right\|_2^2 & \text{(estimated label)} \quad\quad (1.3\text{a}) \\[2ex] -\dfrac{1}{2} \left\| (\boldsymbol{D}_{uu} - \boldsymbol{S}_{uu})^{-1} \boldsymbol{S}_{ul} (\boldsymbol{y}_l + \boldsymbol{\delta}_y) - \boldsymbol{y}_u \right\|_2^2 & \text{(true label)} \quad\quad (1.3\text{b}) \end{cases}$$

Here we used the fact that $\hat{\boldsymbol{y}}_u = \boldsymbol{K}\boldsymbol{y}_l$, where we define $\boldsymbol{K} = (\boldsymbol{D}_{uu} - \boldsymbol{S}_{uu})^{-1}\boldsymbol{S}_{ul}$. We can solve

Eq. (1.3a) by SVD; it's easy to see that the optimal solution should be $\boldsymbol{\delta}_y = \pm d_{\max} \boldsymbol{v}_1$ and $\boldsymbol{v}_1$ is the top right sigular vector if we decompose $(\boldsymbol{D}_{uu} - \boldsymbol{S}_{uu})^{-1} \boldsymbol{S}_{ul} = \boldsymbol{U} \boldsymbol{\Sigma} \boldsymbol{V}^\mathsf{T}$. However, (1.3b) is less straightforward, in fact it is a non-convex trust region problem, which can be generally formulated as

$$\min_{\|\boldsymbol{z}\|_2 \leq d_{\max}} f(\boldsymbol{z}) = \frac{1}{2} \boldsymbol{z}^\mathsf{T} \boldsymbol{H} \boldsymbol{z} + \boldsymbol{g}^\mathsf{T} \boldsymbol{z}, \quad \boldsymbol{H} \text{ is indefinite.} \tag{1.4}$$

Our case (1.3b) can thus be described as $\boldsymbol{H} = -\boldsymbol{K}^\mathsf{T} \boldsymbol{K} \preceq \boldsymbol{0}$ and $\boldsymbol{g} = \boldsymbol{K}^\mathsf{T}(\boldsymbol{y}_u - \hat{\boldsymbol{y}}_u)$. Recently [HK16] proposed a sublinear time solver that is able to find a global minimum in $\mathcal{O}(M/\sqrt{\epsilon})$ time. Here we propose an asymptotic linear algorithm based purely on gradient information, which is stated in Algorithm 1 and Theorem 2. In Algorithm 1 there are two phases, in the following theorems, we show that the phase I ends within finite iterations, and phase II converges with an asymptotic linear rate. We postpone the proof to Appendix 1.

Theorem 1 (Convergent). *Suppose the operator norm $\|\boldsymbol{H}\|_{\mathrm{op}} = \beta$, by choosing a step size $\eta < 1/\beta$ with initialization $\boldsymbol{z}_0 = -\alpha \frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}$, $0 < \alpha < \min(1, \frac{\|\boldsymbol{g}\|^3}{|\boldsymbol{g}^\mathsf{T} \boldsymbol{H} \boldsymbol{g}|})$. Then iterates $\{\boldsymbol{z}_t\}$ generated from Algorithm 1 converge to the global minimum.*

Lemma 1 (Finite phase I). *Since $\boldsymbol{H}$ is indefinite, $\lambda_1 = \lambda_{\min}(\boldsymbol{H}) < 0$, and $\boldsymbol{v}_1$ is the corresponding eigenvector. Denote $\boldsymbol{a}^{(1)} = \boldsymbol{a}^\mathsf{T} \boldsymbol{v}_1$ is the projection of any $\boldsymbol{a}$ onto $\boldsymbol{v}_1$, let T_1 be number of iterations in phase I of Algorithm 1, then:*

$$T_1 \leq \log(1 - \eta \lambda_1)^{-1} \left[\log \left(\frac{1}{\eta |\boldsymbol{g}^{(1)}|} - \frac{1}{\eta \lambda_1} \right) - \log \left(\frac{-\boldsymbol{z}_0^{(1)}}{\eta \boldsymbol{g}^{(1)}} - \frac{1}{\eta \lambda_1} \right) \right]. \tag{1.5}$$

Theorem 2 (Asymptotic linear rate). *Let $\{\boldsymbol{z}_t\}$ be an infinite sequence of iterates generated by Algorithm 1, suppose it converges to $\boldsymbol{z}^*$ (guaranteed by Theorem 1), let $\lambda_{\boldsymbol{H},\min}$ and $\lambda_{\boldsymbol{H},\max}$ be the smallest and largest eigenvalues of $\boldsymbol{H}$. Assume that $\boldsymbol{z}^*$ is a local minimizer then $\lambda_{\boldsymbol{H},\min} > 0$ and given r in the interval $(r_*, 1)$ with $r_* = 1 - \min \left(2\sigma \bar{\alpha} \lambda_{\boldsymbol{H},\min}, 4\sigma(1 - \sigma)\beta \frac{\lambda_{\boldsymbol{H},\min}}{\lambda_{\boldsymbol{H},\max}} \right)$, $\bar{\alpha}$, σ are line search parameters. There exists an integer K such that:*

$$f(\boldsymbol{z}_{t+1}) - f(\boldsymbol{z}^*) \leq r \Big(f(\boldsymbol{z}_t) - f(\boldsymbol{z}^*) \Big)$$

for all $t \geq K$.

1.2.5 Classification task

As we have mentioned, data poisoning attack to classification problem is more challenging, as we can only *flip* an unnoticeable fraction of training labels. This is inherently a combinatorial optimization problem. For simplicity, we restrict the scope to binary classification so that $\boldsymbol{y}_l \in \{-1, +1\}^{n_l}$, and the labels are perturbed as $\tilde{\boldsymbol{y}}_l = \boldsymbol{y}_l \odot \boldsymbol{\delta}_y$, where $\odot$ denotes Hadamard product and $\boldsymbol{\delta}_y = [\pm 1, \pm 1, \ldots, \pm 1]$. For restricting the amount of perturbation, we replace the norm constraint in Eq. (1.3a) with integer constraint $\sum_{i=1}^{n_l} \mathbb{I}_{\{\boldsymbol{\delta}_y[i]=-1\}} \le c_{\max}$, where $c_{\max}$ is a user pre-defined constant. In summary, the final objective function has the following form

$$\min_{\boldsymbol{\delta}_y \in \{+1, -1\}^{n_l}} \quad -\frac{1}{2} \left\| g\Big(\boldsymbol{K}(\boldsymbol{y}_l \odot \boldsymbol{\delta}_y)\Big) - (\boldsymbol{y}_u \text{ or } \hat{\boldsymbol{y}}_u) \right\|^2, \quad \text{s.t.} \quad \sum_{i=1}^{n_l} \mathbb{I}_{\{\boldsymbol{\delta}_y[i]=-1\}} \le c_{\max}, \quad (1.6)$$

where we define $\boldsymbol{K} = (\boldsymbol{D}_{uu} - \boldsymbol{S}_{uu})^{-1} \boldsymbol{S}_{ul}$ and $g(x) = \text{sign}(x)$, so the objective function directly relates to error rate. Notice that the feasible set contains around $\sum_{k=0}^{c_{\max}} \binom{n_l}{k}$ solutions, making it almost impossible to do an exhaustive search. A simple alternative is greedy search: first initialize $\boldsymbol{\delta}_y = [+1, +1, \ldots, +1]$, then at each time we select index $i \in [n_l]$ and try flip $\boldsymbol{\delta}_y[i] = +1 \to -1$, such that the objective function (1.6) decreases the most. Next, we set $\boldsymbol{\delta}_y[i] = -1$. We repeat this process multiple times until the constraint in (1.6) is met.

Doubtlessly, the greedy solver is myopic. The main reason is that the greedy method cannot explore other flipping actions that appear to be sub-optimal within the current context, despite that some sub-optimal actions might be better in the long run. Inspired by the bandit model, we can imagine this problem as a multi-arm bandit, with n_l arms in total. And we apply a strategy similar to ϵ-greedy: each time we assign a high probability to the best action but still leave non-zero probabilities to other "actions". The new strategy can be called *probabilistic method*, specifically, we model each action $\boldsymbol{\delta}_y = \pm 1$ as a Bernoulli distribution, the probability of "flipping" is $P[\boldsymbol{\delta}_y = -1] = \boldsymbol{\alpha}$. The new loss function is just an expectation over Bernoulli variables

$$\min_{\boldsymbol{\alpha}} \quad \left\{ \mathcal{L}(\boldsymbol{\alpha}) := -\frac{1}{2} \mathop{\mathbb{E}}_{\boldsymbol{z} \sim \mathcal{B}(\boldsymbol{1}, \boldsymbol{\alpha})} \left[\left\| g\Big(\boldsymbol{K}(\boldsymbol{y}_l \odot \boldsymbol{z})\Big) - (\boldsymbol{y}_u \text{ or } \hat{\boldsymbol{y}}_u) \right\|^2 \right] + \frac{\lambda}{2} \cdot \|\boldsymbol{\alpha}\|_2^2 \right\}. \quad (1.7)$$

Here we replace the integer constraint in Eq. 1.6 with a regularizer $\frac{\lambda}{2}\|\boldsymbol{\alpha}\|_2^2$, the original constraint is reached by selecting a proper λ. Once problem (1.7) is solved, we craft the actual perturbation $\boldsymbol{\delta}_y$ by setting $\boldsymbol{\delta}_y[i] = -1$ if $\boldsymbol{\alpha}[i]$ is among the top-$c_{\max}$ largest elements.

To solve Eq. (1.7), we need to find a good gradient estimator. Before that, we replace $g(x) = \text{sign}(x)$ with $\tanh(x)$ to get a continuously differentiable objective. We borrow the idea of "reparameterization trick" [FMM18, TMM17] to approximate $\mathcal{B}(\mathbf{1}, \boldsymbol{\alpha})$ by a continuous random vector

$$\boldsymbol{z} \triangleq \boldsymbol{z}(\boldsymbol{\alpha}, \Delta_G) = \frac{2}{1 + \exp\left(\frac{1}{\tau}\left(\log\frac{\boldsymbol{\alpha}}{1-\boldsymbol{\alpha}} + \Delta_G\right)\right)} - 1 \in (-1, 1), \qquad (1.8)$$

where $\Delta_G \sim \boldsymbol{g}_1 - \boldsymbol{g}_2$ and $\boldsymbol{g}_{1,2} \overset{\text{iid}}{\sim} \text{Gumbel}(0, 1)$ are two Gumbel distributions. τ is the temperature controlling the steepness of sigmoid function: as $\tau \to 0$, the sigmoid function point-wise converges to a stair function. Plugging (1.8) into (1.7), the new loss function becomes

$$\mathcal{L}(\boldsymbol{\alpha}) := -\frac{1}{2} \underset{\Delta_G}{\mathbb{E}} \left[\left\| g\left(\boldsymbol{K}(\boldsymbol{y}_l \odot \boldsymbol{z}(\boldsymbol{\alpha}, \Delta_G))\right) - (\boldsymbol{y}_u \text{ or } \hat{\boldsymbol{y}}_u) \right\|^2 \right] + \frac{\lambda}{2} \cdot \|\boldsymbol{\alpha}\|_2^2. \qquad (1.9)$$

Therefore, we can easily obtain an unbiased, low variance gradient estimator via Monte Carlo sampling from $\Delta_G = \boldsymbol{g}_1 - \boldsymbol{g}_2$, specifically

$$\frac{\partial \mathcal{L}(\boldsymbol{\alpha})}{\partial \boldsymbol{\alpha}} \approx -\frac{1}{2} \frac{\partial}{\partial \boldsymbol{\alpha}} \left\| g\left(\boldsymbol{K}(\boldsymbol{y}_l \odot \boldsymbol{z}(\boldsymbol{\alpha}, \Delta_G))\right) - (\boldsymbol{y}_u \text{ or } \hat{\boldsymbol{y}}_u) \right\|^2 + \lambda\boldsymbol{\alpha}. \qquad (1.10)$$

Based on that, we can apply many stochastic optimization methods, including SGD and Adam [KB14], to finalize the process. In the experimental section, we will compare the greedy search with our probabilistic approach on real data.

1.2.6 Experiments

In this section, we will show the effectiveness of our proposed data poisoning attack algorithms for regression and classification tasks on graph-based SSL.

Table 1.1: Dataset statistics. Here n is the total number of samples, d is the dimension of feature vector and γ^* is the optimal γ in validation. `mnist17` is created by extracting images for digits '1' and '7' from standard `mnist` dataset.

Name	Task	n	d	γ^*
`cadata`	Regression	8,000	8	1.0
`E2006`	Regression	19,227	150,360	1.0
`mnist17`	Classification	26,014	780	0.6
`rcv1`	Classification	20,242	47,236	0.1

Experimental settings and baselines

We conduct experiments on two regression and two binary classification datasets[1]. The meta-information can be found in Table 1.1. We use a Gaussian kernel with width γ to construct the graph. For each data, we randomly choose n_l samples as the labeled set, and the rest are unlabeled. We normalize the feature vectors by $x' \leftarrow (x - \mu)/\sigma$, where μ is the sample mean, and σ is the sample variance. For regression data, we also scale the output by $y' \leftarrow (y - y_{\min})/(y_{\max} - y_{\min})$ so that $y' \in [0, 1]$. To evaluate the performance of label propagation models, for regression task we use RMSE metric defined as RMSE $= \sqrt{\frac{1}{n_u} \sum_{i=1}^{n_u} (y_i - \hat{y}_i)^2}$, while for classification tasks we use error rate metric. For comparison with other methods, since **this is the first work on data poisoning attack to G-SSL**, we proposed several baselines according to graph centrality measures. The first baseline is random perturbation, where we randomly add Gaussian noise (for regression) or Bernoulli noise (for regression) to labels. The other two baselines based on graph centrality scores are more challenging, they are widely used to find the "important" nodes in the graph. Intuitively, we need to perturb "important" nodes to attack the model, and we decide the

---

[1]Publicly available at `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`

importance by node degree or PageRank. We explain the baselines with more details in the appendix.

Effectiveness of data poisoning to G-SSL

In this experiment, we consider the white-box setting where the attacker knows not only the ground truth labels $\boldsymbol{y}_u$ but also the correct hyper-parameter γ^*. We thus apply our proposed label poisoning algorithms in Section 1.2.4 and 1.2.5 to attack regression and classification tasks, respectively. In particular, we apply ℓ_2 constraint for perturbation $\boldsymbol{\delta}_y$ in the regression task and use the greedy method in the classification task. The results are shown in Figure 1.2, as we can see in this figure, for both regression and classification problems, small perturbations can lead to vast differences: for instance, on `cadata`, the RMSE increases from 0.2 to 0.3 when applied a carefully designed perturbation $\|\boldsymbol{\delta}_y\| = 3$ (this is very small compared with the norm of label $\|\boldsymbol{y}_l\| \approx 37.36$); More surprisingly, on `mnist17`, the accuracy can drop from 98.46% to 50% by flipping just 3 nodes. This phenomenon indicates that **current graph-based SSL, especially the label propagation method, can be very fragile to data poisoning attacks**. On the other hand, using different baselines (shown in Figure 1.2, bottom row), the accuracy does not decline much, this indicates that our proposed attack algorithms are more effective than centrality based algorithms.

Moreover, the robustness of label propagation is strongly related to the number of labeled data n_l: for all datasets shown in Figure 1.2, we notice that the models with larger n_l tend to be more resistant to poisoning attacks. This phenomenon arises because, during the learning process, the label information propagates from labeled nodes to unlabeled ones. Therefore even if a few nodes are "contaminated" during poisoning attacks, it is still possible to recover the label information from other labeled nodes. Hence this experiment can be regarded as another instance of "no free lunch" theory in adversarial learning [TSE18].
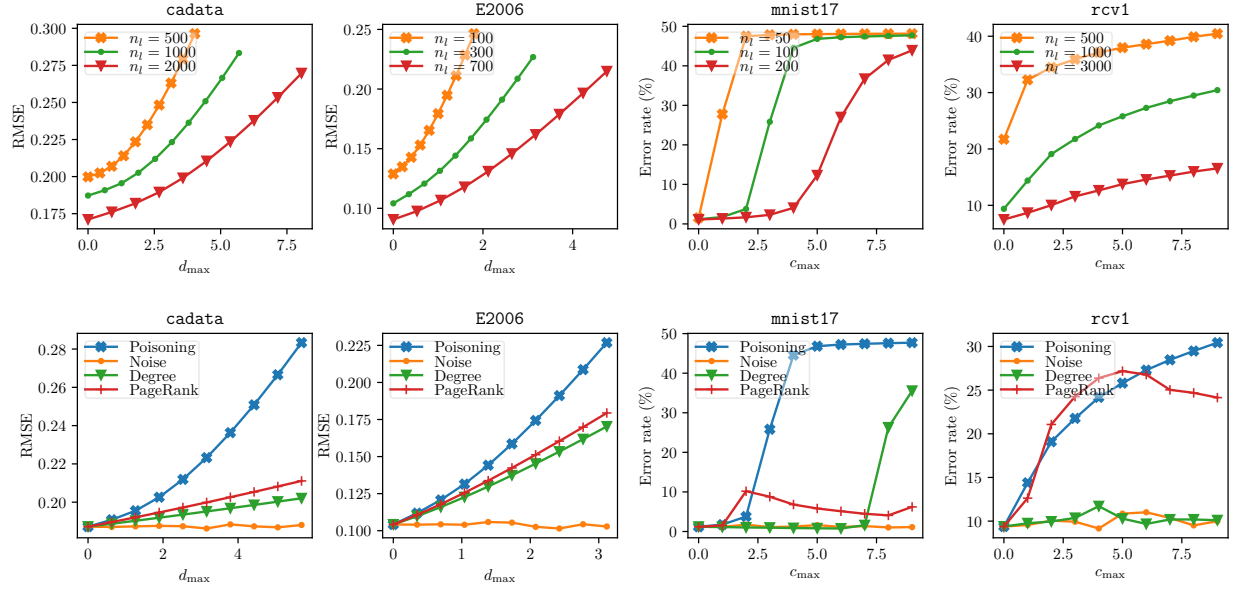
Figure 1.2: *Top row:* testing the effectiveness of poisoning algorithms on four datasets shown in Table (1.1). The left two datasets are regression tasks, and we report the RMSE measure. The right two datasets are classification tasks in which we report the error rate. For each dataset, we repeat the same attacking algorithm w.r.t. different n_l's. *Bottom row:* compare our poisoning algorithm with three baselines (random noise, degree-based attack, PageRank based attack). We follow our convention that d_{max} is the maximal ℓ_2-norm distortion, and c_{max} is the maximal ℓ_0-norm perturbation.

Comparing poisoning with and without truth labels

We compare the effectiveness of poisoning attacks with and without ground truth labels $\boldsymbol{y}_u$. Recall that if an attacker does not hold $\boldsymbol{y}_u$, (s)he will need to replace it with the estimated values $\hat{\boldsymbol{y}}_u$. Thus we expect a degradation of effectiveness due to the replacement of $\boldsymbol{y}_u$, especially when $\hat{\boldsymbol{y}}_u$ is not a good estimation of $\boldsymbol{y}_u$. The result is shown in Figure 1.3. Surprisingly, we did not observe such phenomenon: for regression tasks on cadata and E2006, two curves are closely aligned despite that attacks without ground truth labels $\boldsymbol{y}_u$ are only slightly worse. For classification tasks on mnist17 and rcv1, we cannot observe any difference,

13
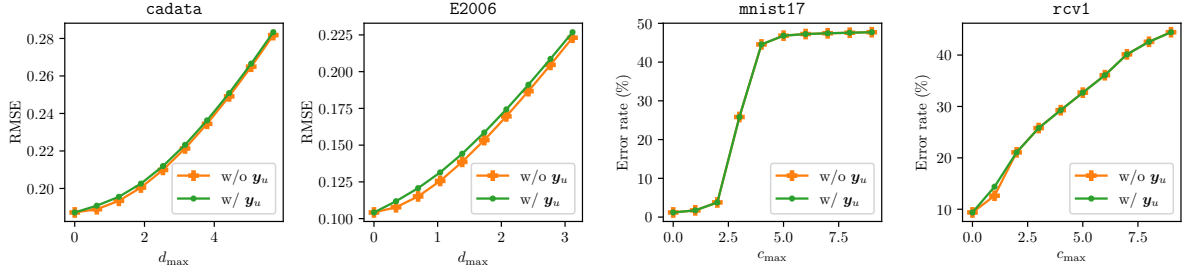
Figure 1.3: Comparing the effectiveness of label poisoning attack with and without knowing the ground truth labels of unlabeled nodes $\boldsymbol{y}_u$. Interestingly, even if the attacker is using the estimated labels $\hat{\boldsymbol{y}}_u$, the effectiveness of the poisoning attack does not degrade significantly.

the choices of which nodes to flip are exactly the same (except the $c_{\max} = 1$ case in `rcv1`). This experiment provides a valuable implication that hiding the ground truth labels cannot protect the SSL models, because the attackers can alternatively use the estimated ground truth $\hat{\boldsymbol{y}}_u$.

Comparing greedy and probabilistic method
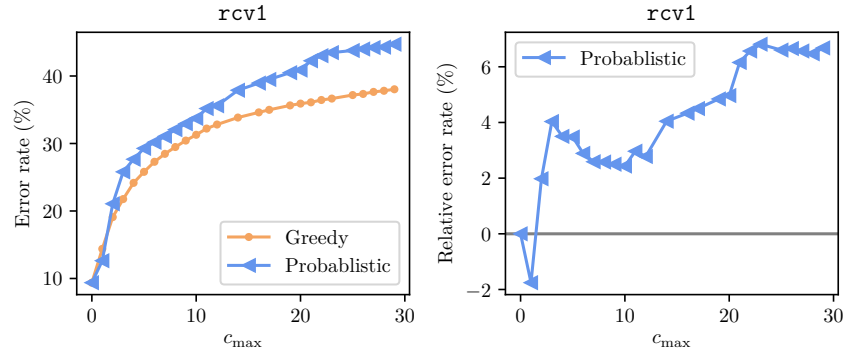
Figure 1.4: Comparing the relative performance of three approximate solvers to discrete optimization problem (1.6). For clarity, we also show the relative performance on the right (probabilistic − greedy).

In this experiment, we compare the performance of three approximate solvers for prob-

14

lem (1.6) in Section 1.2.5, namely greedy and probabilistic methods. We choose `rcv1` data as oppose to `mnist17` data, because `rcv1` is much harder for poisoning algorithm: when $n_l = 1000$, we need $c_{\max} \approx 30$ to make error rate $\approx 50\%$, whilst `mnist17` only takes $c_{\max} = 5$. For hyperparameters, we set $c_{\max} = \{0, 1, \ldots, 29\}$, $n_l = 1000$, $\gamma^* = 0.1$. The results are shown in Figure 1.4, we can see that for larger $c_{\max}$, greedy method can easily stuck into local optima and inferior than our probabilistic based algorithms.

Sensitivity analysis of hyper-parameter
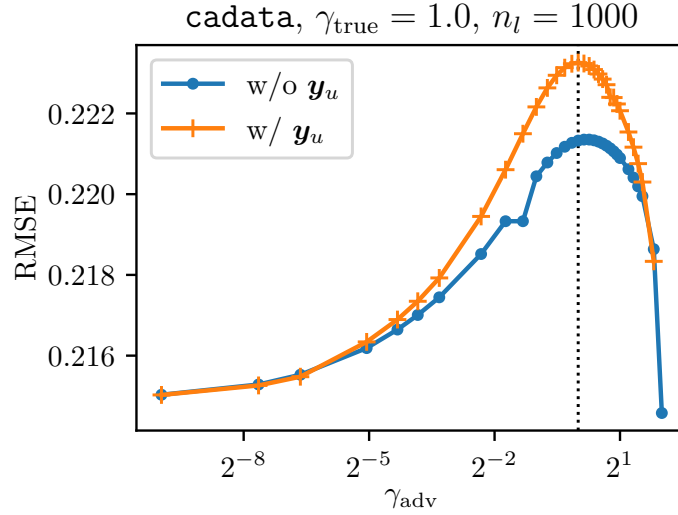
Figure 1.5: Experiment result on imperfect estimations of γ^*.

Since we use the Gaussian kernel to construct the graph, there is an important hyper-parameter γ (kernel width) that controls the structure of the graph defined in (1.1), which is often chosen empirically by the victim through validation. Given the flexibility of γ, it is thus interesting to see how the effectiveness of the poisoning attack degrades with the attacker's imperfect estimation of γ. To this end, we suppose the victim runs the model at the optimal hyperparameter $\gamma = \gamma^*$, determined by validation, while the attacker has a

very rough estimation $\gamma_{\mathrm{adv}} \approx \gamma^*$. We conduct this experiment on `cadata` when the attacker knows or does not know the ground truth labels $\boldsymbol{y}_u$, the result is exhibited in Figure 1.5. It shows that when the adversary does not have exact information of γ, it will receive some penalties on the performance (in RMSE or error rate). However, it is entirely safe to choose a smaller $\gamma_{\mathrm{adv}} < \gamma_{\mathrm{truth}}$ because the performance decaying rate is pretty low. Take Figure 1.5 for example, even though $\gamma_{\mathrm{adv}} = \frac{1}{8}\gamma_{\mathrm{truth}}$, the RMSE only drops from 0.223 to 0.218. On the other hand, if γ_{adv} is over large, the nodes become more isolated, and thus the perturbations are harder to propagate to neighbors.

CHAPTER 2

Randomization-based Adversarial Defense

2.1 Random Self-Ensemble (RSE)

Recent studies have revealed the vulnerability of deep neural networks: A small adversarial perturbation that is imperceptible to human can easily make a well-trained deep neural network misclassify. This makes it unsafe to apply neural networks in security-critical applications. We propose a new defense algorithm called Random Self-Ensemble (RSE) by combining two important concepts: **randomness** and **ensemble**. To protect a targeted model, RSE adds random noise layers to the neural network to prevent the strong gradient-based attacks, and ensembles the prediction over random noises to stabilize the performance.

2.1.1 Overview of (white-box) adversarial attack algorithms

In the white-box adversarial attack setting, attackers have all information about the targeted neural network, including network structure and network weights (denoted by w). Using this information, attackers can compute gradient with respect to input data $\nabla_x f(w, x)$ by back-propagation. Note that gradient is very informative for attackers since it characterizes the sensitivity of the prediction with respect to the input image.

To craft an adversarial example, [GSS15a] proposed a fast gradient sign method (FGSM), where the adversarial example is constructed by

$$x' = x_0 - \epsilon \cdot \text{sign}(\nabla_x f(w, x_0)) \tag{2.1}$$

with a small $\epsilon > 0$. Based on that, several followup works were made to improve the efficiency and availability, such as Rand-FGSM [TKP17] and I-FGSM [KGB17a]. Recently, Carlini & Wagner [CW17] showed that constructing an adversarial example can be formulated as solving the following optimization problem:

$$x' = \min_{x \in [0,1]^d} c \cdot g(x) + \|x - x_0\|_2^2, \tag{2.2}$$

where the first term is the loss function that characterizes the success of the attack and the second term is to enforce a small distortion. The parameter $c > 0$ is used to balance these two requirements. Several variants were proposed recently [CSZ18, MMS18a], but most of them can be categorized in the similar framework. The C&W attack has been recognized as a strong attacking algorithm to test defense methods.

For untargeted attack, where the goal is to find an adversarial example that is close to the original example but yields different class prediction, the loss function in (2.2) can be defined as

$$g(x) = \max\{\max_{i \neq t}(Z(x')_i) - Z(x')_t, -\kappa\}, \tag{2.3}$$

where t is the correct label, $Z(x)$ is the network's output before softmax layer (logits).

For targeted attack, the loss function can be designed to force the classifier to return the target label. For attackers, targeted attack is strictly harder than untargeted attack (since once the targeted attack succeeds, the same adversarial image can be used to perform untargeted attack without any modification). On the contrary, for defenders, untargeted attacks are strictly harder to defense than targeted attack. Therefore, we focus on defending the untargeted attacks in our experiments.

Another commonly used adversarial attack algorithm is called PGD attack. The goal of PGD attack is to find adversarial examples in a γ-ball, which can be naturally formulated as the following objective function:

$$\max_{\|\boldsymbol{\xi}\|_\infty \leq \gamma} \ell(f(\boldsymbol{x}_o + \boldsymbol{\xi}; \boldsymbol{w}), y_o). \tag{2.4}$$

Starting from $\boldsymbol{x}^0 = \boldsymbol{x}_o$, PGD attack conducts projected gradient descent iteratively to update the adversarial example:

$$\boldsymbol{x}^{t+1} = \Pi_\gamma \left\{ \boldsymbol{x}^t + \alpha \cdot \operatorname{sign}\left(\nabla_{\boldsymbol{x}}\ell\big(f(\boldsymbol{x}^t; \boldsymbol{w}), y_o\big)\right) \right\}, \qquad (2.5)$$

where Π_γ is the projection to the set $\{\boldsymbol{x}| \; \|\boldsymbol{x} - \boldsymbol{x}_o\|_\infty \leq \gamma\}$. Although multi-step PGD iterations may not necessarily return the optimal adversarial examples, we decided to apply it in our experiments, following the previous work of [MMS18a]. An advantage of PGD attack over C&W attack is that it gives us a direct control of distortion by changing γ, while in C&W attack we can only do this indirectly via tuning the regularizer.

2.1.2 Overview of adversarial defense algorithms

Because of the vulnerability of adversarial examples [SZS13], several methods have been proposed to improve the network's robustness against adversarial examples. [PMW16] proposed *defensive distillation*, which uses a modified softmax layer controlled by temperature to train the "teacher" network, and then use the prediction probability (soft-labels) of teacher network to train the student network (it has the same structure as the teacher network). However, as stated in [CW17], this method does not work properly when dealing with the C&W attack. Moreover, [ZNR17] showed that by using a modified ReLU activation layer (called BReLU) and adding noise into origin images to augment the training dataset, the learned model will gain some stability to adversarial images. Another popular defense approach is *adversarial training* [KGB17a, HXS15]. It generates and appends adversarial examples found by an attack algorithm to the training set, which helps the network to learn how to distinguish adversarial examples. Through combining adversarial training with enlarged model capacity, [MMS18a] is able to create an MNIST model that is robust to the first order attacks, but this approach does not work very well on more difficult datasets such as CIFAR-10.

It is worth mentioning that there are many defense algorithms (*r.f.* [BRR18, MLW18,

GRC18, DAB18, XWZ18, SKN18, SKC18]) against white box attacks in literature. Unfortunately, as [ACW18, AC18] pointed out, these algorithms are not truly effective to white box attacks. Recall the "white box" means that the attackers know *everything* concerning how models make decisions, these include the potential defense mechanisms. In this condition, the white box attacks can walk around all defense algorithms listed above and the accuracy under attack can still be nearly zero. In addition to changing the network structure, there are other methods [XEQ18, MC17, FCS17, GMP17] "detecting" the adversarial examples, which are beyond the scope of our discussion.

There is another highly correlated work (*r.f.* [LAG18a]) which also adopts very similar idea, except that they view this problem from the angle of differential privacy, while we believe that the adversarial robustness is more correlated with regularization and ensemble learning. Furthermore, our work is public available earlier than this similar work on Arxiv.

2.1.3 Proposed algorithm: random self-ensemble

In this section, we propose our self-ensemble algorithm to improve the robustness of neural networks. We will first motivate and introduce our algorithm and then discuss several theoretical reasons behind it.

It is known that ensemble of several different models can improve the robustness. However, an ensemble of finite k models is not very practical because it will increase the model size by k folds. For example, AlexNet model on ImageNet requires 240MB storage, and storing 100 of them will require 24GB memory. Moreover, it is hard to find many heterogeneous models with similar accuracy. To improve the robustness of practical systems, we propose the following self-ensemble algorithm that can generate an infinite number of models on-the-fly without any additional memory cost.

Our main idea is to add randomness into the network structure. More specifically, we introduce a new "noise layer" that fuses input vector with a randomly generated noise,
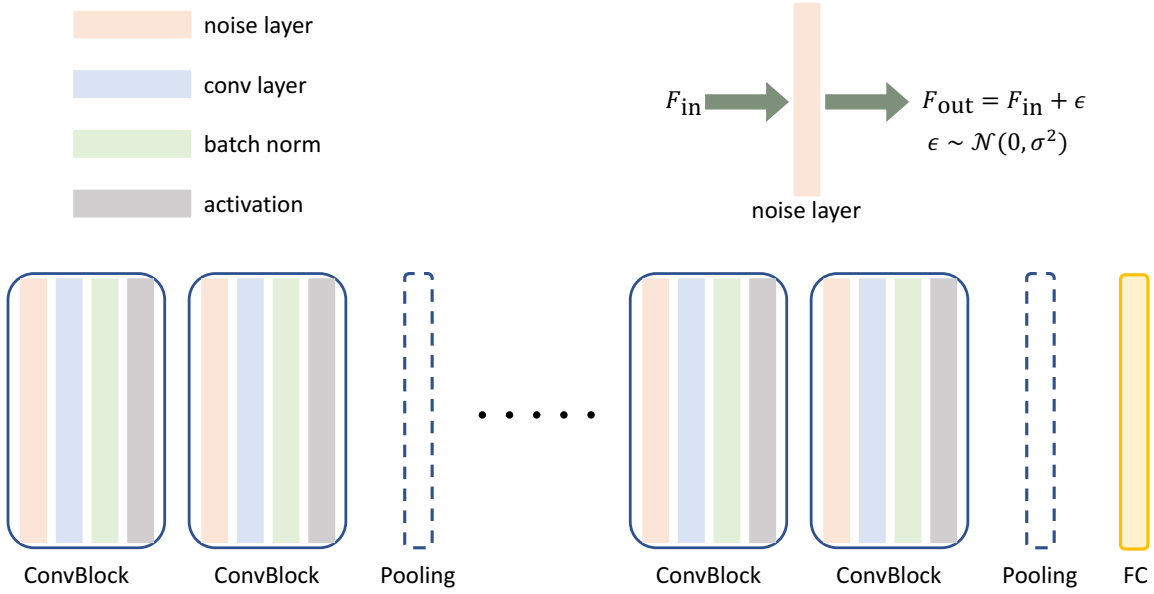
Figure 2.1: Our proposed noisy VGG style network, we add a noise layer before each convolution layer. For simplicity, we call the noise layer before the first convolution layer the "init-noise", and all other noise layer "inner-noise". For these two kinds of layers we adopt different variances of Gaussian noise. Note that similar design can be transplanted to other architectures such as ResNet.

i.e. $x \rightarrow x + \epsilon$ when passing through the noise layer. Then we add this layer before each convolution layer as shown in Fig. 2.1. Since most attacks require computing or estimating gradient, the noise level in our model will control the success rate of those attacking algorithms. In fact, we can integrate this layer into any other neural network.

If we denote the original neural network as $f(w, x)$ where $w \in \mathbb{R}^{d_w}$ is the weights and $x \in \mathbb{R}^{d_x}$ is the input image, then considering the random noise layer, the network can be denoted as $f_\epsilon(w, x)$ with random $\epsilon \in \mathbb{R}^{d_e}$. Therefore we have an infinite number of models in the pocket (with different ϵ) without having any memory overhead. However, adding randomness will also affect the prediction accuracy of the model. How can we make sure that the ensemble of these random models have enough accuracy?

Algorithm 2 Training and Testing of Random Self-Ensemble (RSE)

   Training phase:

for iter $= 1, 2, \ldots$ **do**

   Randomly sample (x_i, y_i) in dataset

   Randomly generate $\epsilon \sim \mathcal{N}(0, \sigma^2)$ for each noise layer.

   Compute $\Delta w = \nabla_w \ell(f_\epsilon(w, x_i), y_i)$ (Noisy gradient)

   Update weights: $w \leftarrow w - \Delta w$.

end for

   Testing phase:

Given testing image x, initialize $p = (0, 0, \ldots, 0)$

for $j = 1, 2, \ldots, \#$Ensemble **do**

   Randomly generate $\epsilon \sim \mathcal{N}(0, \sigma^2)$ for each noise layer.

   Forward propagation to calculate probability output

$$p^j = f_\epsilon(w, x)$$

   Update p: $p \leftarrow p + p^j$.

end for

Predict the class with maximum score $\hat{y} = \arg\max_k p_k$

A critical observation is that we need to add this random layer in both training and testing phases. The training and testing algorithms are listed in Algorithm 3. In the training phase, gradient is computed as $\nabla_w f_\epsilon(w, x_i)$ which includes the noise layer, and the noise is generated randomly for each stochastic gradient descent update. In the testing phase, we construct n random noises and ensemble their probability outputs by

$$p = \sum_{j=1}^{n} f_{\epsilon_j}(w, x), \text{ and predict } \hat{y} = \arg\max_k p_k. \tag{2.6}$$

If we do not care about the prediction time, n can be very large, but in practice we found it saturates at $n \approx 10$ (see Fig. 2.4).

This approach is different from Gaussian data augmentation in [ZNR17]: they only add Gaussian noise to images during the training time, while we add noise before each convolution layer at both training and inference time. When training, the noise helps optimization algorithm to find a stable convolution filter that is robust to perturbed input, while when testing, the roles of noise are two-folded: one is to perturb the gradient to fool gradient-based attacks.The other is it gives different outputs by doing multiple forward operations and a simple ensemble method can improve the testing accuracy.

2.1.4 Mathematical explanations

Training and testing of RSE. Here we explain our training and testing procedure. In the training phase, our algorithm is solving the following optimization problem:

$$w^* = \arg\min_w \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x_i, y_i) \in \mathcal{D}_{\text{train}}} \mathbb{E}_{\epsilon \sim \mathcal{N}(0, \sigma^2)} \ell\Big(f_\epsilon(w, x_i), y_i\Big), \tag{2.7}$$

where $\ell(\cdot, \cdot)$ is the loss function and $\mathcal{D}_{\text{train}}$ is the training dataset. Note that for simplicity we assume ϵ follows a zero-mean Gaussian, but in general our algorithm can work for a large variety of noise distribution such as Bernoulli-Gaussian: $\epsilon_i = b_i e_i$, where $e_i \overset{\text{iid}}{\sim} \mathcal{N}(0, \sigma^2)$ and $b_i \overset{\text{iid}}{\sim} \mathcal{B}(1, p)$.

At testing time, we ensemble the outputs through several forward propagation, specifically:

$$\hat{y}_i = \arg\max \mathbb{E}_{\epsilon \sim \mathcal{N}(0, \sigma^2)} f_\epsilon(w, x_i). \tag{2.8}$$

Here $\arg\max$ means the index of maximum element in a vector. The reason that our RSE algorithm achieves the similar prediction accuracy with original network is because (2.7) is minimizing an upper bound of the loss of (2.8) – Similar to the idea of [NYM17], if we choose

negative log-likelihood loss, then $\forall w \in \mathbb{R}^{d_w}$:

$$\frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x_i,y_i)\in\mathcal{D}_{\text{train}}} \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)}\ell\Big(f_\epsilon(w,x_i),y_i\Big)$$

$$\overset{(a)}{\approx} \mathbb{E}_{(x_i,y_i)\sim\mathcal{P}_{\text{data}}}\Big\{ -\mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)}\log f_\epsilon(w,x_i)[y_i]\Big\}$$

$$\overset{(b)}{\geq} \mathbb{E}_{(x_i,y_i)\sim\mathcal{P}_{\text{data}}}\Big\{ -\log \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)} f_\epsilon(w,x_i)[y_i]\Big\} \qquad (2.9)$$

$$\overset{(c)}{\geq} \mathbb{E}_{(x_i,y_i)\sim\mathcal{P}_{\text{data}}}\Big\{ -\log \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)} f_\epsilon(w,x_i)[\hat{y}_i]\Big\}$$

$$\overset{(a)}{\approx} \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{x_i\in\mathcal{D}_{\text{test}}} -\log \mathbb{E}_{\epsilon\sim\mathcal{N}(0,\sigma^2)} f_\epsilon(w,x_i)[\hat{y}_i].$$

Where $\mathcal{P}_{\text{data}}$ is the data distribution, $\mathcal{D}_{\text{train/test}}$ is the training set and test set, respectively. And (a) follows from generalization bound (see [SKL17] or appendix for details), (b) comes from Jensen's inequality and (c) is by the inference rule (2.8). So by minimizing (2.7) we are actually minimizing the upper bound of inference confidence $-\log f_\epsilon(w,x_i)[\hat{y}_i]$, this validates our ensemble inference procedure.

RSE is equivalent to Lipschitz regularization. Another point of view is that perturbed training is equivalent to Lipschitz regularization, which further helps defensing gradient based attack. If we fix the output label y then the loss function $\ell(f_\epsilon(w,x),y)$ can be simply denoted as $\ell\circ f_\epsilon$. Lipchitz of the function $\ell\circ f_\epsilon$ is a constant $L_{\ell\circ f_\epsilon}$ such that

$$|\ell(f_\epsilon(w,x),y) - \ell(f_\epsilon(w,\tilde{x}),y)| \leq L_{\ell\circ f_\epsilon}\|x - \tilde{x}\| \qquad (2.10)$$

for all $x, \tilde{x}$. In fact, it has been proved recently that Lipschitz constant can be used to measure the robustness of machine learning model [HA17, WZC18b]. If $L_{\ell\circ f_\epsilon}$ is large enough, even a tiny change of input $x - \tilde{x}$ can significantly change the loss and eventually get an incorrect prediction. On the contrary, by controlling $L_{\ell\circ f}$ to be small, we will have a more robust network.

Next we show that our noisy network indeed controls the Lipschitz constant. Following

the notation of (2.7), we can see that

$$\mathbb{E}_{\epsilon \sim \mathcal{N}(0,\sigma^2)} \ell\Big(f_\epsilon(w, x_i), y_i\Big) \stackrel{(a)}{\approx} \mathbb{E}_{\epsilon \sim \mathcal{N}(0,\sigma^2)} \Big[\ell\Big(f_0(w, x_i), y_i\Big) + \epsilon^\intercal \nabla_\epsilon \ell\Big(f_0(w, x_i), y_i\Big)$$

$$+ \frac{1}{2} \epsilon^\intercal \nabla_\epsilon^2 \ell\Big(f_0(w, x_i), y_i\Big) \epsilon\Big] \qquad (2.11)$$

$$\stackrel{(b)}{=} \ell\Big(f_0(w, x_i), y_i\Big) + \frac{\sigma^2}{2} \mathrm{Tr}\Big\{\nabla_\epsilon^2 \ell\Big(f_0(w, x_i), y_i\Big)\Big\}.$$

For (a), we do Taylor expansion at $\epsilon = 0$. Since we set the variance of noise σ^2 very small, we only keep the second order term. For (b), we notice that the Gaussian vector ϵ is i.i.d. with zero mean. So the linear term of ϵ has zero expectation, and the quadratic term is directly dependent on variance of noise and the trace of Hessian. As a convex relaxation, if we assume $\ell \circ f_0$ is convex, then we have that $d \cdot \|A\|_{\max} \geq \mathrm{Tr}(A) \geq \|A\|_{\max}$ for $A \in \mathbb{S}_+^{d \times d}$, we can rewrite (2.11) as

$$\mathrm{Loss}(f_\epsilon, \{x_i\}, \{y_i\}) \simeq \mathrm{Loss}(f_0, \{x_i\}, \{y_i\}) + \frac{\sigma^2}{2} L_{\ell \circ f_0}, \qquad (2.12)$$

which means the training of noisy networks is equivalent to training the original model with an extra regularization of Lipschitz constant, and by controlling the variance of noise we can balance the robustness of network with training loss.

2.1.5   Discussions

Here we show both *randomness* and *ensemble* are important in our algorithm. Indeed, if we remove any component, the performance will significantly drop.

First, as mentioned before, the main idea of our model is to have infinite number of models f_ϵ, each with a different ϵ value, and then ensemble the result. A naive way to achieve this goal is to fix a pre-trained model f_0 and then generate many f_ϵ in the testing phase by adding different small noise to f_0. However, Fig. 2.2 shows this approach (denoted as Test noise only) will result in much worse performance (20% without any attack). Therefore it is non-trivial to guarantee the model to be good after adding small random noise. In our random self-ensemble algorithm, in addition to adding noise in the testing phase, we also **add**

noise layer in the training phase, and this is important for getting good performance.

Second, we found adding noise in the testing phase and then ensemble the predictions is important. In Fig. 2.2, we compare the performance of RSE with the version that only adds the noise layer in the training phase but not in the testing phase (so the prediction is $f_\epsilon(w, x)$ instead of $\mathbb{E}_\epsilon f_\epsilon(w, x)$). The results clearly show that the performance drop under smaller attacks. This proves **ensemble in the testing phase is crucial**.
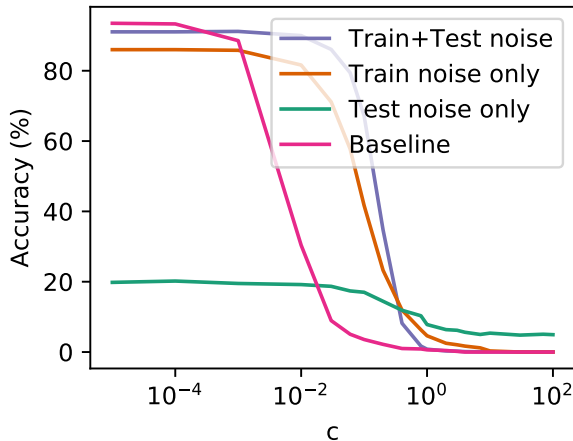
Figure 2.2: We test three models on CIFAR10 and VGG16 network: In the first model noise is added both at training and testing time, in the second model noise is added only at training time, in the last model we only add noise at testing time. As a comparison we also plot baseline model which is trained conventionally. For all models that are noisy at testing time, we automatically enable self-ensemble.

2.1.6 Experiments

Datasets and network structure We test our method on two datasets—CIFAR10 and STL10. We do not compare the results on MNIST since it is a much easier dataset and existing defense methods such as [PMG16b, ZNR17, KGB17a, HXS15] can effectively increase image distortion under adversarial attacks. On CIFAR10 data, we evaluate the performance on

both VGG-16 [SZ15] and ResNeXt [XGD17]; on STL10 data we copy and slightly modify a simple model[1] which we name it as "Model A".

Defense algorithms. We include the following defense algorithms into comparison (their parameter settings can be found in Tab. 2.1):

- Random Self-Ensemble (RSE): our proposed method.

- Defensive distillation [PMW16]: first train a teacher network at temperature T, then use the teacher network to train a student network of the same architecture and same temperature. The student network is called the distilled network.

- Robust optimization combined with BReLU activation [ZNR17]: first we replace all ReLU activation with BReLU activation. And then at the training phase, we randomly perturb training data by Gaussian noise with $\sigma = 0.05$ as suggested.

- Adversarial retraining by FGSM attacks [KGB17a, HXS15]: we first pre-train a neural network without adversarial retraining. After that, we either select an original data batch or an adversarial data batch with probability 1/2. We continue training it until convergence.

Attack models. We consider the white-box setting and choose the state-of-the-art C&W attack [CW17] to evaluate the above-mentioned defense methods. Moreover, we test our algorithm under untargeted attack, since untargeted attack is strictly harder to defense than targeted attack. In fact, C&W untargeted attack is the most challenging attack for a defense algorithm.

Moreover, we assume C&W attack knows the randomization procedure of RSE, so the C&W objective function will change accordingly (as proposed in [AC18] for attacking an ensemble model). The details can be found in the appendix.

---

[1]Publicly available at `https://github.com/aaron-xichen/pytorch-playground`

Measure. Unlike attacking models that only need to operate on correctly classified images, a competitive defense model not only protects the model when attackers exist, but also keeps a good performance on clean datasets. Based on this thought, we compare the accuracy of guarded models under different strengths of C&W attack, the strength can be measured by L_2-norm of image distortion and further controlled by parameter c in (2.2). Note that an adversarial image is correctly predicted under C&W attack if and only if the original image is correctly classified and C&W attack cannot find an adversarial example within a certain distortion level.
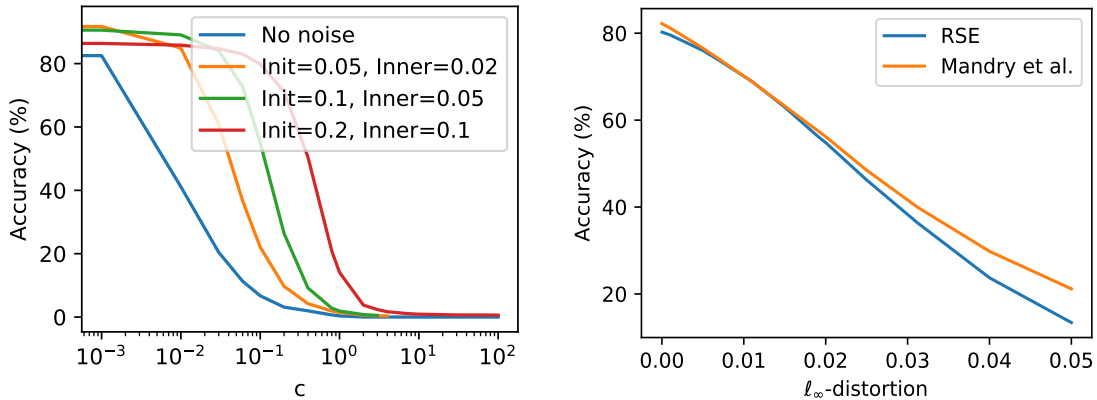
Figure 2.3: *Left*: the effect of noise level on robustness and generalization ability. Clearly random noise can improve the robustness of the model. *Right*: comparing RSE with adversarial defense method [MMS18a].

The effect of noise level

We first test the performance of RSE under different noise levels. We use Gaussian noise for all the noise layers in our network and the standard deviation σ of Gaussian controls the noise level. Note that we call the noise layer before the first convolution layer the "init-noise", and all other noise layers the "inner-noise".

In this experiment, we apply different noise levels in both training and testing phases to
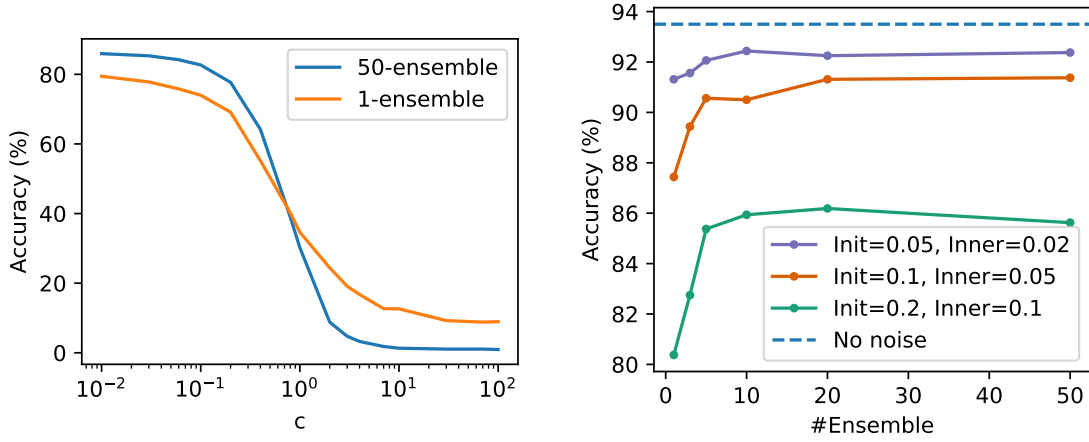
Figure 2.4: *Left*: Comparing the accuracy under different levels of attack, here we choose VGG16+CIFAR10 combination. We can see that the ensemble model achieves better accuracy under weak attacks. *Right*: Testing accuracy (without attack) of different n (number of random models used for ensemble).

see how different variances change the robustness as well as generalization ability of networks. As an example, we choose

$$(\sigma_{\text{init}}, \sigma_{\text{inner}}) = \{(0,0), (0.05, 0.02), (0.1, 0.05), (0.2, 0.1)\} \tag{2.13}$$

on VGG16+CIFAR10. The result is shown in Fig. 2.3 (*left*).

As we can see, both "init-noise" and "inner-noise" are beneficial to the robustness of neural network, but at the same time, one can see higher noise reduces the accuracy for weak attacks ($c \lesssim 0.01$). From Fig. 2.3, we observe that if the input image is normalized to $[0, 1]$, then choosing $\sigma_{\text{init}} = 0.2$ and $\sigma_{\text{inner}} = 0.1$ is good. Thus we fix this parameter for all the experiments.

Self-ensemble

Next we show self-ensemble helps to improve the test accuracy of our noisy mode. As an example, we choose VGG16+CIFAR10 combination and the standard deviation of initial

Table 2.1: Experiment setting for defense methods

Methods	Settings
No defense	Baseline model
RSE(for CIFAR10 + VGG16)	Initial noise: 0.2, inner noise: 0.1, 50-ensemble
RSE(for CIFAR10 + ResNeXt)	Initial noise: 0.1, inner noise 0.1, 50-ensemble
RSE(for STL10 + Model A)	Initial noise: 0.2, inner noise: 0.1, 50-ensemble
Defensive distill	Temperature = 40
Adversarial training (I)	FGSM adversarial examples, $\epsilon \sim \mathcal{U}(0.1, 0.3)$
Adversarial training (II)	Following [MMS18a], PGD adversary with $\epsilon_\infty = \frac{8.0}{256}$
Robust Opt. + BReLU	Following [ZNR17]

noise layer is $\sigma = 0.2$, other noise layers is $\sigma = 0.1$. We compare 50-ensemble with 1-ensemble (i.e. single model), and the result can be found in Fig. 2.4.

We find the 50-ensemble method outperform the 1-ensemble method by $\sim 8\%$ accuracy when $c < 0.4$. This is because when the attack is weak enough, the majority choice of networks has lower variance and higher accuracy. On the other hand, we can see if $c > 1.0$ or equivalently the average distortion greater than 0.93, the ensemble model is worse. We conjecture that this is because when the attack is strong enough then the majority of random sub-models make wrong prediction, but when looking at any individual model, the random effect might be superior than group decision. In this situation, self-ensemble may have a negative effect on accuracy.

Practically, if running time is the primary concern, it is not necessary to calculate many ensemble models. In fact, we find the accuracy saturates rapidly with respect to number of models, moreover, if we inject smaller noise then ensemble benefit would be weaker and the accuracy gets saturated earlier. Therefore, we find 10-ensemble is good enough for testing accuracy, see Fig. 2.4.

Table 2.2: Prediction accuracy of defense methods under C&W attack with different c. We can clearly observe that RSE is the most robust model. Our accuracy level remains at above 75% when other methods are below 30%.

	$c = 0.01$	$c = 0.03$	$c = 0.06$	$c = 0.1$	$c = 0.2$
RSE(ours)	**90.00%**	**86.06%**	**79.44%**	**67.19%**	**34.75%**
Adv retraining	27.00%	9.81%	4.13%	3.69%	1.44%
Robust Opt+BReLU	75.06%	47.93%	30.94%	20.69%	13.50%
Distill	49.88%	17.69%	4.56%	3.13%	1.44%
No defense	30.38%	8.93%	5.06%	3.56%	2.19%

Comparing defense methods

Finally, we compare our RSE method with other existing defense algorithms. Note that we test all of them using C&W untargeted attack, which is the most difficult setting for defenders.

The comparison across different datasets and networks can be found in Tab. 2.2 and Fig. 2.5. As we can see, previous defense methods have little effect on C&W attacks. For

	bird	car	cat	deer	dog	frog	horse	plane	truck
No defense	1.94	0.31	0.74	4.72	7.99	3.66	9.22	0.75	1.32
Defensive distill	6.55	0.70	**13.78**	2.54	13.90	2.56	**11.36**	0.66	3.54
Adv. retraining	2.58	0.31	0.75	6.08	0.75	9.01	6.06	0.31	4.08
Robust Opt. + BReLU	**17.11**	1.02	4.07	13.50	7.09	15.34	7.15	2.08	17.57
RSE(ours)	12.87	**2.61**	12.47	**21.47**	**31.90**	**19.09**	9.45	**10.21**	**22.15**

Table 2.3: Image distortion required for targeted attacks.
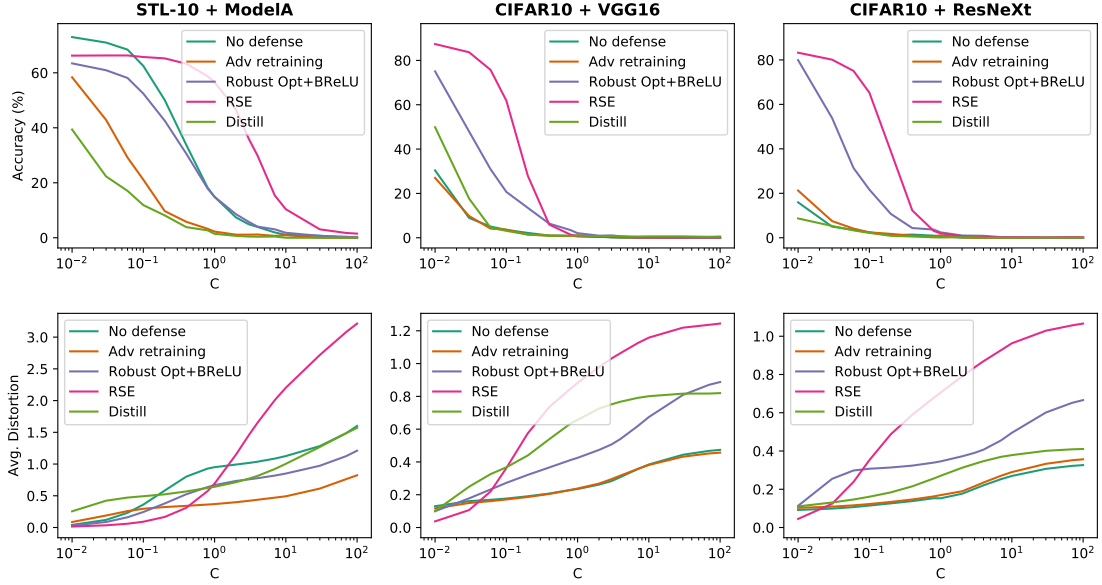
Figure 2.5: Comparing the accuracy of CIFAR10+{VGG16, ResNeXt} and STL10+Model A. We show both the change of accuracy and average distortion w.r.t. attacking strength parameter c (the parameter in the C&W attack). Our model (RSE) clearly outperforms all the existing methods under strong attacks in both accuracy and average distortion.

example, Robust Opt+BReLU [ZNR17] is useful for CIFAR10+ResNeXt, but the accuracy is even worse than no defense model for STL10+Model A. In contrast, our RSE method acts as a good defence across all cases. Specifically, RSE method enforces the attacker to find much more distorted adversarial images in order to start a successful attack. As showed in Fig. 2.5, when we allow an average distortion of 0.21 on CIFAR10+VGG16, C&W attack is able to conduct untargeted attacks with success rate > 99%. On the contrary, by defending the networks via RSE, C&W attack only yields a success rate of ∼20%. Recently, another version of adversarial training is proposed [MMS18a]. Different from "Adversarial training (I)" shown in Tab. 2.1, it trains the network with adversaries generated by multiple steps of gradient descent (therefore we call it "Adversarial training (II)" in Tab. 2.1). Compared with our method, the major weakness is that it takes ∼10 times longer to train a robust network despite that the result is only slightly better than our RSE, see Fig. 2.3 (*right*).
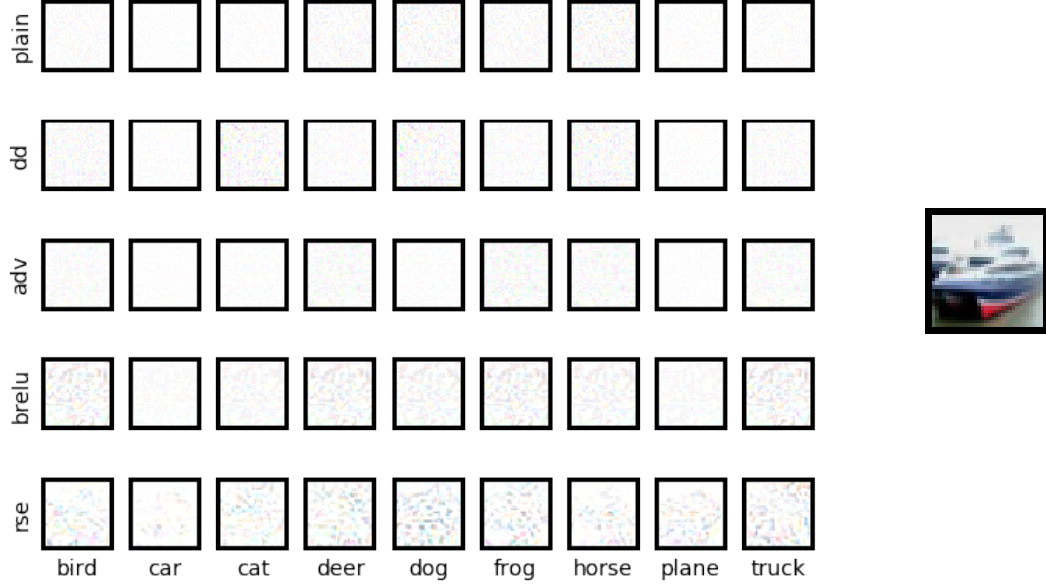
Figure 2.6: Targeted adversarial image distortion, each column indicates a defense algorithm and each row is the adversarial target (the original image is in "ship" class, shown in the right side). Here we choose $c = 1$ for targetd C&W attack. Visually, color spot means the distortion of images, thus a successful defending method should lead to more spots.

Apart from the accuracy under C&W attack, we find the distortion of adversarial images also increases significantly, this can be seen in Fig. 2.2(2nd row), as c is large enough (so that all defense algorithms no longer works) our RSE method achieves the largest distortion.

Although all above experiments are concerning untargeted attack, it does not mean targeted attack is not covered, as we said, targeted attack is harder for attacking methods and easier to defense. As an example, we test all the defense algorithms on CIFAR-10 dataset under targeted attacks. We randomly pick an image from CIFAR10 and plot the perturbation $x_{\text{adv}} - x$ in Fig. 2.6 (the exact number is in Tab. 2.3), to make it easier to print out, we subtract RGB channels from 255 (so the majority of pixels are white and distortions can be

noticed). One can easily find RSE method makes the adversarial images more distorted.

Lastly, apart from CIFAR-10, we also design an experiment on a much larger data to support the effectiveness of our method even on large data. Due to space limit, the result is postponed to appendix.

2.2 Adv-BNN: Adversarial Robustness with Bayesian Neural Network

This method can regarded as an extension of RSE method discussed in the previous section. This algorithm is motivated by the following two ideas. First, although recent work has demonstrated that fusing randomness can improve the robustness of neural networks [LCZ17], we noticed that adding noise blindly to all the layers is not the optimal way to incorporate randomness. Instead, we model randomness under the framework of Bayesian Neural Network (BNN) to formally learn the posterior distribution of models in a scalable way. Second, we formulate the mini-max problem in BNN to learn the best model distribution under adversarial attacks, leading to an adversarial-trained Bayesian neural network. Experiment results demonstrate that the proposed algorithm achieves state-of-the-art performance under strong attacks. On CIFAR-10 with VGG network, our model leads to 14% accuracy improvement compared with adversarial training [MMS18a] and random self-ensemble [LCZ17] under PGD attack with 0.035 distortion, and the gap becomes even larger on a subset of ImageNet[2].

2.2.1   Bayesian neural networks (BNN)

The idea of BNN is illustrated in Fig. 2.7. Given the observable random variables $(\boldsymbol{x}, y)$, we aim to estimate the distributions of hidden variables $\boldsymbol{w}$. In our case, the observable random variables correspond to the features $\boldsymbol{x}$ and labels y, and we are interested in the posterior

---

[2]Code for reproduction has been made available online at `https://github.com/xuanqing94/BayesianDefense`

over the weights $p(\boldsymbol{w}|\boldsymbol{x}, y)$ given the prior $p(\boldsymbol{w})$. However, the exact solution of posterior is often intractable: notice that $p(\boldsymbol{w}|\boldsymbol{x}, y) = \frac{p(\boldsymbol{x},y|\boldsymbol{w})p(\boldsymbol{w})}{p(\boldsymbol{x},y)}$ but the denominator involves a high dimensional integral [BKM17], hence the conditional probabilities are hard to compute. To speedup inference, we generally have two approaches—we can either sample $\boldsymbol{w} \sim p(\boldsymbol{w}|\boldsymbol{x}, y)$ efficiently without knowing the closed-form formula through, for example, Stochastic Gradient Langevin Dynamics (SGLD) [WT11], or we can approximate the true posterior $p(\boldsymbol{w}|\boldsymbol{x}, y)$ by a parametric distribution $q_{\boldsymbol{\theta}}(\boldsymbol{w})$, where the unknown parameter $\boldsymbol{\theta}$ is estimated by minimizing $\mathsf{KL}\big(q_{\boldsymbol{\theta}}(\boldsymbol{w}) \parallel p(\boldsymbol{w}|\boldsymbol{x}, y)\big)$ over $\boldsymbol{\theta}$. For neural network, the exact form of KL-divergence can be unobtainable, but we can easily find an unbiased gradient estimator of it using backward propagation, namely *Bayes by Backprop* [BCK15].
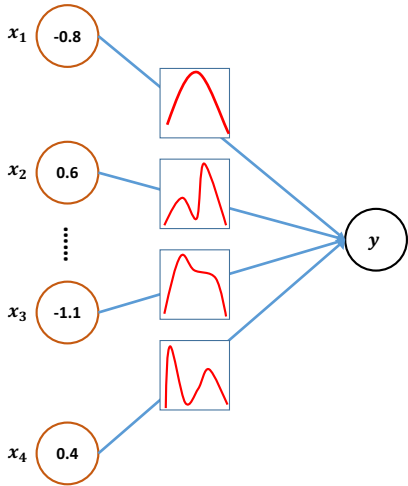
Figure 2.7: Illustration of Bayesian neural networks.

Despite that both methods are widely used and analyzed in-depth, they have some obvious shortcomings, making high dimensional Bayesian inference remain to be an open problem. For SGLD and its extension (e.g. [LCC16]), since the algorithms are essentially SGD updates with extra Gaussian noise, they are very easy to implement. However, they can only get one sample $\boldsymbol{w} \sim p(\boldsymbol{w}|\boldsymbol{x}, y)$ in each minibatch iteration at the cost of one forward-backward propagation, thus not efficient enough for fast inference. In addition, as the step size η_t in SGLD decreases, the samples become more and more correlated so that one needs to generate many samples in order to control the variance. Conversely, the variational inference method i s efficient to generate samples since we know the approximated posterior $q_{\boldsymbol{\theta}}(\boldsymbol{w})$ once we minimized the KL-divergence. The problem is that for simplicity we often assume the approximation $q_{\boldsymbol{\theta}}$

to be a fully factorized Gaussian distribution:

$$q_{\boldsymbol{\theta}}(\boldsymbol{w}) = \prod_{i=1}^{d} q_{\boldsymbol{\theta}_i}(\boldsymbol{w}_i), \text{ and } q_{\boldsymbol{\theta}_i}(\boldsymbol{w}_i) = \mathcal{N}(\boldsymbol{w}_i; \boldsymbol{\mu}_i, \boldsymbol{\sigma}_i^2). \tag{2.14}$$

Although our assumption (2.14) has a simple form, it inherits the main drawback from mean-field approximation. When the ground truth posterior has significant correlation between variables, the approximation in (2.14) will have a large deviation from true posterior $p(\boldsymbol{w}|\boldsymbol{x}, y)$. This is especially true for convolutional neural networks, where the values in the same convolutional kernel seem to be highly correlated. However, we still choose this family of distribution in our design as the simplicity and efficiency are mostly concerned.

In fact, there are many techniques in deep learning area borrowing the idea of Bayesian inference without mentioning explicitly. For example, Dropout [SHK14] is regarded as a powerful regularization tool for deep neural networks, which applies an element-wise product of the feature maps and i.i.d. Bernoulli or Gaussian r.v. $\mathcal{B}(1, \alpha)$ (or $\mathcal{N}(1, \alpha)$). If we allow each dimension to have an independent dropout rate and take them as model parameters to be learned, then we can extend it to the variational dropout method [KSW15]. Notably, learning the optimal dropout rates for data relieves us from manually tuning hyper-parameter on hold-out data. Similar idea is also used in RSE [LCZ17], except that it was used to improve the robustness under adversarial attacks. As we discussed in the previous section, RSE incorporates Gaussian noise $\boldsymbol{\epsilon} \sim \mathcal{N}(0, \sigma^2)$ in an additive manner, where the variance σ^2 is user predefined in order to maximize the performance. Different from RSE, our Adv-BNN has two degrees of freedom (mean and variance) and the network is trained on adversarial examples.

2.2.2 Method

In our method, we combine the idea of adversarial training [MMS18a] with Bayesian neural network, hoping that the randomness in the weights $\boldsymbol{w}$ provides stronger protection for our model.

To build our Bayesian neural network, we assume the joint distribution $q_{\boldsymbol{\mu},\boldsymbol{s}}(\boldsymbol{w})$ is fully factorizable (see (2.14)), and each posterior $q_{\boldsymbol{\mu}_i,\boldsymbol{s}_i}(\boldsymbol{w}_i)$ follows normal distribution with mean $\boldsymbol{\mu}_i$ and standard deviation $\exp(\boldsymbol{s}_i) > 0$. The prior distribution is simply isometric Gaussian $\mathcal{N}(\mathbf{0}_d, s_0^2 \boldsymbol{I}_{d\times d})$. We choose the Gaussian prior and posterior for its simplicity and closed-form KL-divergence, that is, for any two Gaussian distributions s and t,

$$\mathsf{KL}(s \parallel t) = \log \frac{\sigma_t}{\sigma_s} + \frac{\sigma_s^2 + (\mu_s - \mu_t)^2}{2\sigma_t^2} - 0.5, \qquad s \text{ or } t \sim \mathcal{N}(\mu_{s \text{ or } t}, \sigma_{s \text{ or } t}^2). \tag{2.15}$$

Note that it is also possible to choose more complex priors such as "spike-and-slab" [IR05] or Gaussian mixture, although in these cases the KL-divergence of prior and posterior is hard to compute and practically we replace it with the Monte-Carlo estimator, which has higher variance, resulting in slower convergence rate [Kin17].

Following the recipe of variational inference, we adapt the robust optimization to the evidence lower bound (ELBO) *w.r.t.* the variational parameters during training. First of all, recall the ELBO on the original dataset (the unperturbed data) can be written as

$$-\mathsf{KL}\Big(q_{\boldsymbol{\mu},\boldsymbol{s}}(\boldsymbol{w}) \parallel p(\boldsymbol{w})\Big) + \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_{\text{tr}}} \mathbb{E}_{\boldsymbol{w}\sim q_{\boldsymbol{\mu},\boldsymbol{s}}} \log p(y_i|\boldsymbol{x}_i, \boldsymbol{w}), \tag{2.16}$$

rather than directly maximizing the ELBO in (2.16), we consider the following alternative objective,

$$\mathcal{L}(\boldsymbol{\mu},\boldsymbol{s}) \triangleq -\mathsf{KL}\Big(q_{\boldsymbol{\mu},\boldsymbol{s}}(\boldsymbol{w}) \parallel p(\boldsymbol{w})\Big) + \sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_{\text{tr}}} \min_{\|\boldsymbol{x}_i^{\text{adv}}-\boldsymbol{x}_i\|\leq\gamma} \mathbb{E}_{\boldsymbol{w}\sim q_{\boldsymbol{\mu},\boldsymbol{s}}} \log p(y_i|\boldsymbol{x}_i^{\text{adv}}, \boldsymbol{w}). \tag{2.17}$$

This is essentially finding the minima for each data point $(\boldsymbol{x}_i, y_i) \in \mathcal{D}_{\text{tr}}$ inside the γ-norm ball, we can also interpret (2.17) as an even looser lower bound of evidence. So the robust optimization procedure is to maximize (2.17), i.e.

$$\boldsymbol{\mu}^*, \boldsymbol{s}^* = \underset{\boldsymbol{\mu},\boldsymbol{s}}{\arg\max} \, \mathcal{L}(\boldsymbol{\mu},\boldsymbol{s}). \tag{2.18}$$

To make the objective more specific, we combine (2.17) with (2.18) and get

$$\underset{\boldsymbol{\mu},\boldsymbol{s}}{\arg\max} \left\{ \Big[\sum_{(\boldsymbol{x}_i,y_i)\in\mathcal{D}_{\text{tr}}} \min_{\|\boldsymbol{x}_i^{\text{adv}}-\boldsymbol{x}_i\|\leq\gamma} \mathbb{E}_{\boldsymbol{w}\sim q_{\boldsymbol{\mu},\boldsymbol{s}}} \log p(y_i|\boldsymbol{x}_i^{\text{adv}}, \boldsymbol{w}) \Big] - \mathsf{KL}\Big(q_{\boldsymbol{\mu},\boldsymbol{s}}(\boldsymbol{w}) \parallel p(\boldsymbol{w})\Big) \right\} \tag{2.19}$$

In our case, $p(y|\boldsymbol{x}^{\mathrm{adv}}, \boldsymbol{w}) = \mathrm{Softmax}\big(f(\boldsymbol{x}_i^{\mathrm{adv}}; \boldsymbol{w})\big)[y_i]$ is the network output on the adversarial sample $(\boldsymbol{x}_i^{\mathrm{adv}}, y_i)$. More generally, we can reformulate our model as $y = f(\boldsymbol{x}; \boldsymbol{w}) + \zeta$ and assume the residual ζ follows either $\mathrm{Logistic}(0, 1)$ or Gaussian distribution depending on the specific problem, so that our framework includes both classification and regression tasks. We can see that the only difference between our Adv-BNN and the standard BNN training is that the expectation is now taken over the adversarial examples $(\boldsymbol{x}^{\mathrm{adv}}, y)$, rather than natural examples $(\boldsymbol{x}, y)$. Therefore, at each iteration we first apply a randomized PGD attack for T iterations to find $\boldsymbol{x}^{\mathrm{adv}}$, and then fix the $\boldsymbol{x}^{\mathrm{adv}}$ to update $\boldsymbol{\mu}, \boldsymbol{s}$.

When updating $\boldsymbol{\mu}$ and $\boldsymbol{s}$, the KL term in (2.17) can be calculated exactly by (2.15), whereas the second term is very complex (for neural networks) and can only be approximated by sampling. Besides, in order to fit into the back-propagation framework, we adopt the *Bayes by Backprop* algorithm [BCK15]. Notice that we can reparameterize $\boldsymbol{w} = \boldsymbol{\mu} + \exp(\boldsymbol{s}) \odot \boldsymbol{\epsilon}$, where $\boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}_d, \boldsymbol{I}_{d \times d})$ is a parameter free random vector, then for any differentiable function $h(\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s})$, we can show that

$$
\begin{aligned}
\frac{\partial}{\partial \boldsymbol{\mu}} \mathop{\mathbb{E}}_{\boldsymbol{w}}[h(\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s})] &= \mathop{\mathbb{E}}_{\boldsymbol{\epsilon}} \left[\frac{\partial}{\partial \boldsymbol{w}} h(\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s}) + \frac{\partial}{\partial \boldsymbol{\mu}} h(\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s}) \right] \\
\frac{\partial}{\partial \boldsymbol{s}} \mathop{\mathbb{E}}_{\boldsymbol{w}}[h(\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s})] &= \mathop{\mathbb{E}}_{\boldsymbol{\epsilon}} \left[\exp(\boldsymbol{s}) \odot \boldsymbol{\epsilon} \odot \frac{\partial}{\partial \boldsymbol{w}} h(\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s}) + \frac{\partial}{\partial \boldsymbol{s}} h(\boldsymbol{w}, \boldsymbol{\mu}, \boldsymbol{s}) \right].
\end{aligned}
\tag{2.20}
$$

Now the randomness is decoupled from model parameters, and thus we can generate multiple $\boldsymbol{\epsilon}$ to form a unbiased gradient estimator. To integrate into deep learning framework more easily, we also designed a new layer called `RandLayer`, which is summarized in appendix.

It is worth noting that once we assume the simple form of variational distribution (2.14), we can also adopt the *local reparameterization trick* [KSW15]. That is, rather than sampling the weights $\boldsymbol{w}$, we directly sample the activations and enjoy the lower variance during the sampling process. Although in our experiments we find the simple *Bayes by Backprop* method efficient enough.

For ease of doing SGD iterations, we rewrite (2.18) into a finite sum problem by dividing

both sides by the number of training samples N_{tr}

$$\boldsymbol{\mu}^*, \boldsymbol{s}^* = \arg\min_{\boldsymbol{\mu}, \boldsymbol{s}} \underbrace{-\frac{1}{N_{\text{tr}}} \sum_{i=1}^{N_{\text{tr}}} \log p(y_i | \boldsymbol{x}_i^{\text{adv}}, \boldsymbol{w})}_{\text{classification loss}} + \underbrace{\frac{1}{N_{\text{tr}}} g(\boldsymbol{\mu}, \boldsymbol{s})}_{\text{regularization}}, \qquad (2.21)$$

here we define $g(\boldsymbol{\mu}, \boldsymbol{s}) \triangleq \mathsf{KL}(q_{\boldsymbol{\mu}, \boldsymbol{s}}(\boldsymbol{w}) \parallel p(\boldsymbol{w}))$ by the closed form solution (2.15), so there is no randomness in it. We sample new weights by $\boldsymbol{w} = \mu + \exp(\boldsymbol{s}) \odot \boldsymbol{\epsilon}$ in each forward propagation, so that the stochastic gradient is unbiased. In practice, however, we need a weaker regularization for small dataset or large model, since the original regularization in (2.21) can be too large. We fix this problem by adding a factor $0 < \alpha \leq 1$ to the regularization term, so the new loss becomes

$$-\frac{1}{N_{\text{tr}}} \sum_{i=1}^{N_{\text{tr}}} \log p(y_i | \boldsymbol{x}_i^{\text{adv}}, \boldsymbol{w}) + \frac{\alpha}{N_{\text{tr}}} g(\boldsymbol{\mu}, \boldsymbol{s}), \quad 0 < \alpha \leq 1. \qquad (2.22)$$

In our experiments, we found little to no performance degradation compared with the same network without randomness, if we choose a suitable hyper-parameter α, as well as the prior distribution $\mathcal{N}(\boldsymbol{0}, s_0^2 \boldsymbol{I})$.

The overall training algorithm is shown in Alg. 3. To sum up, our Adv-BNN method trains an arbitrary Bayesian neural network with the min-max robust optimization, which is similar to [MMS18a]. As we mentioned earlier, even though our model contains noise and eventually the gradient information is also noisy, by doing multiple forward-backward iterations, the noise will be cancelled out due to the law of large numbers. This is also the suggested way to bypass some stochastic defenses in [ACW18].

Will it be beneficial to have randomness in adversarial training? After all, both randomized network and adversarial training can be viewed as different ways for controlling local Lipschitz constants of the loss surface around the image manifold, and thus it is non-trivial to see whether combining those two techniques can lead to better robustness. The connection between randomized network (in particular, RSE) and local Lipschitz regularization has been derived in [LCZ17]. Adversarial training can also be connected to local Lipschitz regularization

Algorithm 3 Code snippet for training Adv-BNN

1: **procedure pgd_attack($\boldsymbol{x}$, y, $\boldsymbol{w}$)**

2:     // Perform the PGD-attack (2.5), omitted for brevity

3: **end procedure**

4: **procedure train(data, $\boldsymbol{w}$)**

5:     // Input: dataset and network weights $\boldsymbol{w}$

6:     **for** $(\boldsymbol{x}, y)$ **in** data **do**

7:         $\boldsymbol{x}^{\mathrm{adv}} \leftarrow$ **pgd_attack**$(\boldsymbol{x}, y, \boldsymbol{w})$ ▷ *Generate adversarial images*

8:         $\boldsymbol{w} \leftarrow \boldsymbol{\mu} + \exp(\boldsymbol{s}) \odot \boldsymbol{\epsilon}, \ \boldsymbol{\epsilon} \sim \mathcal{N}(\boldsymbol{0}_d, \boldsymbol{I}_{d \times d})$ ▷ *Sample new model parameters*

9:         $\hat{y} \leftarrow$ **forward**$(\boldsymbol{w}, \boldsymbol{x}^{\mathrm{adv}})$ ▷ *Forward propagation*

10:         loss_ce $\leftarrow$ **cross_entropy**$(\hat{y}, y)$ ▷ *Cross-entropy loss*

11:         loss_kl $\leftarrow$ **kl_divergence**$(\boldsymbol{w})$ ▷ *KL-divergence* following (2.15)

12:         $\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{s}) \leftarrow$ loss_ce $+ \frac{\alpha}{N_{\mathrm{tr}}} \cdot$ loss_kl   ▷ *Total loss following* (2.22)

13:         $\frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}}, \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}} \leftarrow$ **backward**$\big(\mathcal{L}(\boldsymbol{\mu}, \boldsymbol{s})\big)$ ▷ *Backward propagation to get gradients*

14:         $\boldsymbol{\mu}, \boldsymbol{s} \leftarrow \boldsymbol{\mu} - \eta_t \frac{\partial \mathcal{L}}{\partial \boldsymbol{\mu}}, \boldsymbol{s} - \eta_t \frac{\partial \mathcal{L}}{\partial \boldsymbol{s}}$ ▷ SGD update, omitting momentum and weight decay

15:     **end for**

16:     **return** net

17: **end procedure**

---

with the following arguments. Recall that the loss function given data $(\boldsymbol{x}_i, y_i)$ is denoted as $\ell\big(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i\big)$, and similarly the loss on perturbed data $(\boldsymbol{x}_i + \boldsymbol{\xi}, y_i)$ is $\ell\big(f(\boldsymbol{x}_i + \boldsymbol{\xi}; \boldsymbol{w}), y_i\big)$. Then if we expand the loss to the first order

$$\Delta\ell \triangleq \ell\big(f(\boldsymbol{x}_i + \boldsymbol{\xi}; \boldsymbol{w}), y_i\big) - \ell\big(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i\big) = \boldsymbol{\xi}^\mathsf{T} \nabla_{\boldsymbol{x}_i} \ell\big(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i\big) + \mathcal{O}(\|\boldsymbol{\xi}\|^2), \quad (2.23)$$

we can see that the robustness of a deep model is closely related to the gradient of the loss over the input, i.e. $\nabla_{\boldsymbol{x}_i} \ell\big(f(\boldsymbol{x}_i), y_i\big)$. If $\|\nabla_{\boldsymbol{x}_i} \ell\big(f(\boldsymbol{x}_i), y_i\big)\|$ is large, then we can find a suitable $\boldsymbol{\xi}$ such that $\Delta\ell$ is large. Under such condition, the perturbed image $\boldsymbol{x}_i + \boldsymbol{\xi}$ is very likely to

be an adversarial example. It turns out that adversarial training directly controls the local Lipschitz value on the **training set**,

$$\min_{\boldsymbol{w}} \ell(f(\boldsymbol{x}_i^{\text{adv}}; \boldsymbol{w}), y_i) = \min_{\boldsymbol{w}} \max_{\|\boldsymbol{\xi}\| \le \gamma} \ell(f(\boldsymbol{x}_i + \boldsymbol{\xi}; \boldsymbol{w})$$

$$= \min_{\boldsymbol{w}} \max_{\|\boldsymbol{\xi}\| \le \gamma} \ell(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \boldsymbol{\xi}^\mathsf{T} \nabla_{\boldsymbol{x}_i} \ell(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \mathcal{O}(\|\boldsymbol{\xi}\|^2). \tag{2.24}$$

Moreover, if we ignore the higher order term $\mathcal{O}(\|\boldsymbol{\xi}\|^2)$ then (2.24) becomes

$$\min_{\boldsymbol{w}} \ell(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i) + \gamma \cdot \|\nabla_{\boldsymbol{x}_i} \ell(f(\boldsymbol{x}_i; \boldsymbol{w}), y_i)\|. \tag{2.25}$$

In other words, the adversarial training can be simplified to Lipschitz regularization, and if the model generalizes, the local Lipschitz value will also be small on the **test set**. Yet, as [LH18] indicates, for complex dataset like CIFAR-10, the local Lipschitz is still very large on **test set**, even though it is controlled on **training set**. The drawback of adversarial training motivates us to combine the randomness model with adversarial training, and we observe a significant improvement over adversarial training or `RSE` alone (see the experiment section below).

2.2.3 Experimental results

In this section, we test the performance of our robust Bayesian neural networks (`Adv-BNN`) with strong baselines on a wide variety of datasets. In essence, our method is inspired by adversarial training [MMS18a] and BNN [BCK15], so these two methods are natural baselines. If we see a significant improvement in adversarial robustness, then it means that randomness and robust optimization have independent contributions to defense. Additionally, we would like to compare our method with `RSE` [LCZ17], another strong defense algorithm relying on randomization. Lastly, we include the models without any defense as references. For ease of reproduction, we list the hyper-parameters in the appendix. Readers can also refer to the source code on github.

It is known that adversarial training becomes increasingly hard for high dimensional data [SST18]. In addition to standard low dimensional dataset such as CIFAR-10, we also did

experiments on two more challenging datasets: 1) STL-10 [CNL11], which has 5,000 training images and 8,000 testing images. Both of them are 96×96 pixels; 2) ImageNet-143, which is a subset of ImageNet [DDS09b], and widely used in conditional GAN training [MK18]. The dataset has 18,073 training and 7,105 testing images, and all images are 64×64 pixels. It is a good benchmark because it has much more classes than CIFAR-10, but is still manageable for adversarial training.

Evaluating models under white box ℓ_∞-PGD attack

In the first experiment, we compare the accuracy under the white box ℓ_∞-PGD attack. We set the maximum ℓ_∞ distortion to $\gamma \in$ `[0:0.07:0.005]` and report the accuracy on test set. The results are shown in Fig. 2.8. Note that when attacking models with stochastic components, we adjust PGD accordingly. To demonstrate the relative performance more clearly, we show some numerical results in Tab. 2.4.
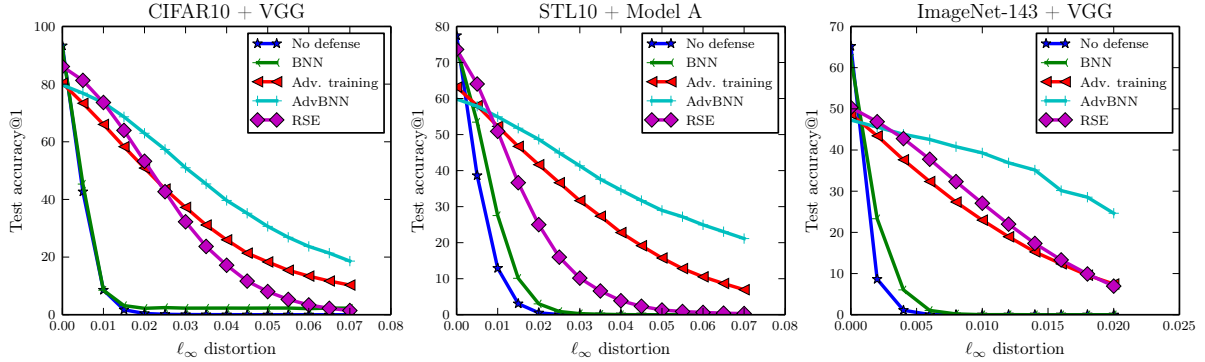
Figure 2.8: Accuracy under ℓ_∞-PGD attack on three different datasets: CIFAR-10, STL-10 and ImageNet-143. In particular, we adopt a smaller network for STL-10 namely "Model A"[1], while the other two datasets are trained on VGG.

From Fig. 2.8 and Tab. 2.4 we can observe that although BNN itself does not increase the robustness of the model, when combined with the adversarial training method, it dramatically

Data	Defense	0	0.015	0.035	0.055	0.07
CIFAR10	Adv. Training	**80.3**	58.3	31.1	15.5	10.3
	Adv-BNN	79.7	**68.7**	**45.4**	**26.9**	**18.6**
STL10	Adv. Training	**63.2**	46.7	27.4	12.8	7.0
	Adv-BNN	59.9	**51.8**	**37.6**	**27.2**	**21.1**

Data	Defense	0	0.004	0.01	0.016	0.02
ImageNet-143	Adv. Training	**48.7**	37.6	23.0	12.4	7.5
	Adv-BNN	47.3	**43.8**	**39.3**	**30.2**	**24.6**

Table 2.4: Comparing the testing accuracy under different levels of PGD attacks. We include our method, `Adv-BNN`, and the state of the art defense method, the multi-step adversarial training proposed in [MMS18a]. The better accuracy is marked in **bold**. Notice that although our `Adv-BNN` incurs larger accuracy drop in the original test set (where $\|\boldsymbol{\xi}\|_\infty = 0$), we can choose a smaller α in (2.22) so that the regularization effect is weakened, in order to match the accuracy.

increase the testing accuracy for $\sim$10% on a variety of datasets. Moreover, the overhead of Adv-BNN over adversarial training is small: it will only double the parameter space (for storing mean and variance), and the total training time does not increase much. Finally, similar to `RSE`, modifying existing network architectures into BNN is fairly simple, we only need to replace Conv/BatchNorm/Linear layers by their variational version. Hence we can easily build robust models based on existing ones.

---

[2]Publicly available at `https://github.com/aaron-xichen/pytorch-playground/tree/master/stl10`, repository has no affiliation with us.

Black-box transfer attack

Is our Adv-BNN model susceptible to transfer attack? we answer this question by studying the affinity between models, because if two models are similar (e.g. in loss landscape) then we can easily attack one model using the adversarial examples crafted through the other. In this section, we measure the adversarial sample transferability between different models namely `None` (no defense), `BNN`, `Adv.Train`, `RSE` and `Adv-BNN`. This is done by the method called "transfer attack" [LCL17]. Initially it was proposed as a black box attack algorithm: when the attacker has no access to the *target model*, one can instead train a similar model from scratch (called *source model*), and then generate adversarial samples with *source model*. As we can imagine, the success rate of transfer attack is directly linked with how similar the source/target models are. In this experiment, we are interested in the following question: how easily can we transfer the adversarial examples between these five models? We study the affinity between those models, where the affinity is defined by

$$\rho_{A \mapsto B} = \frac{\text{Acc}[B] - \text{Acc}[B|A]}{\text{Acc}[B] - \text{Acc}[B|B]}, \tag{2.26}$$

where $\rho_{A \mapsto B}$ measures the success rate using source model A and target model B, $\text{Acc}[B]$ denotes the accuracy of model B without attack, $\text{Acc}[B|A(\text{or } B)]$ means the accuracy under adversarial samples generated by model $A(\text{or } B)$. Most of the time, it is easier to find adversarial examples through the target model itself, so we have $\text{Acc}[B|A] \geq \text{Acc}[B|B]$ and thus $0 \leq \rho_{A \mapsto B} \leq 1$. However, $\rho_{A \mapsto B} = \rho_{B \mapsto A}$ is **not** necessarily true, so the affinity matrix is not likely to be symmetric. We illustrate the result in Fig. 2.9.

We can observe that $\{$`None`, `BNN`$\}$ are similar models, their affinity is strong ($\rho \approx 0.85$) for both direction: $\rho_{\text{BNN} \mapsto \text{None}}$ and $\rho_{\text{None} \mapsto \text{BNN}}$. Likewise, $\{$`RSE`, `Adv-BNN`, `Adv.Train`$\}$ constitute the other group, yet the affinity is not very strong ($\rho \approx 0.5 \sim 0.6$), meaning these three methods are all robust to the black box attack to some extent.
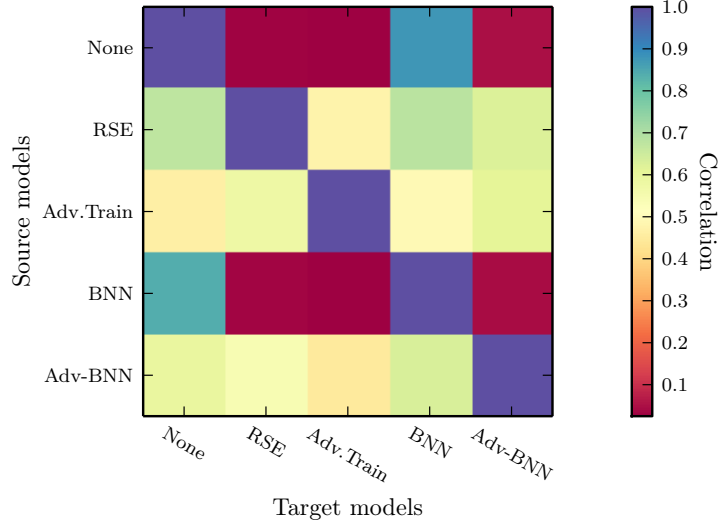
Figure 2.9: Black box, transfer attack experiment results. We select all combinations of source and target models trained from 5 defense methods and calculate the affinity according to (2.26).

Miscellaneous experiments

Following experiments are not crucial in showing the success of our method, however, we still include them to help clarifying some doubts of careful readers.

The first question is about sample efficiency, recall in prediction stage we sample weights from the approximated posterior and generate the label by

$$\hat{y} = \arg\max_y \frac{1}{m} \sum_{k=1}^{m} p(y|\boldsymbol{x}, \boldsymbol{w}_k), \quad \boldsymbol{w}_k \sim q_{\boldsymbol{\mu},\boldsymbol{s}}. \tag{2.27}$$

In practice, we do not want to average over lots of forward propagation to control the variance, which will be much slower than other models during the prediction stage. Here we take ImageNet-143 data + VGG network as an example, to show that only 10∼20 forward operations are sufficient for robust and accurate prediction. Furthermore, the number seems to be independent on the adversarial distortion, as we can see in Fig. 2.10(*left*). So our algorithm is especially suitable to large scale scenario.

One might also be concerned about whether 20 steps of PGD iterations are sufficient to find adversarial examples. It has been known that for certain adversarial defense method, the effectiveness appears to be worse than claimed [EIA18], if we increase the PGD-steps from 20 to 100. In Fig. 2.10(*right*), we show that even if we increase the number of iteration to 1000, the accuracy does not change very much. This means that even the adversary invests more resources to attack our model, its marginal benefit is negligible.
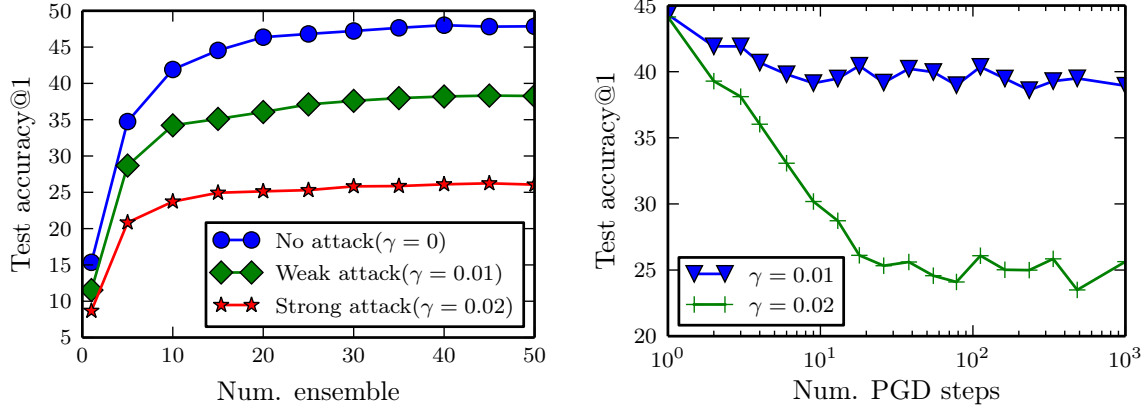
Figure 2.10: *Left*: we tried different number of forward propagation and averaged the results to make prediction (2.27). We see that for different scales of perturbation $\gamma \in \{0, 0.01, 0.02\}$, choosing number of ensemble $n = 10 \sim 20$ is good enough. *Right*: testing accuracy stabilizes quickly as #PGD-steps goes greater than 20, so there is no necessity to further increase the number of PGD steps.

2.3 Neural SDE: Explanation and Exploration of Noise-induced Robustness

Previous methods are all based on noise injection scheme in between convolutional blocks. In this section, we take this idea one step further to the continuous limit, based on the seminal work of Neural-ODE [CRB18]. Neural Ordinary Differential Equation (Neural

ODE) has been proposed as a continuous approximation to the ResNet architecture. Some commonly used regularization mechanisms in discrete neural networks (e.g., dropout, Gaussian noise) are missing in current Neural ODE networks. We introduce a new continuous neural network framework called Neural Stochastic Differential Equation (Neural SDE), which naturally incorporates various commonly used regularization mechanisms based on random noise injection. For regularization purposes, our framework includes multiple types of noise patterns, such as dropout, additive, and multiplicative noise, which are common in plain neural networks. We provide some theoretical analyses explaining the improved robustness of our models against input perturbations. Furthermore, we demonstrate that the Neural SDE network can achieve better generalization than the Neural ODE and is more resistant to adversarial and non-adversarial input perturbations.

Traditional neural networks are usually stacked with multiple layers; recent work [CRB18] shows that we can model it in the continuous limit. This means that there is no notion of discrete layers, and hidden features are changed smoothly. Mathematically it has the following form

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau, \tau; \boldsymbol{w}) \, \mathrm{d}\tau, \tag{2.28}$$

where $t > s$ are two different "depths"; $\boldsymbol{h}_t$ is the hidden features at depth t; $\boldsymbol{f}$ is the residual block parameterized by $\boldsymbol{w}$. This formula is exactly the continuous limit of the original ResNet [HZR16] structure

$$\boldsymbol{h}_{n+1} = \boldsymbol{h}_n + \boldsymbol{f}(\boldsymbol{h}_n; \boldsymbol{w}_n), \tag{2.29}$$

here the layer index $n = 1, 2, \ldots, N$ is discrete. Notice that the original Neural ODE model does not contain any randomness in hidden features $\boldsymbol{h}_t$. Thus it is not ready to model a variety of random neural networks (such as dropout). To address this limitation, we augment the original Neural ODE model (2.28) with two kinds of stochastic terms: one is the diffusion term (to model Gaussian noise), and the other is jump term (to model Bernoulli noise),

formally

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \underbrace{\int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau, \tau; \boldsymbol{w})\, \mathrm{d}\tau}_{\text{drift term}} + \underbrace{\int_s^t \boldsymbol{G}(\boldsymbol{h}_\tau, \tau)\, \mathrm{d}\mathbf{B}_\tau}_{\text{diffusion term}}$$
$$+ \underbrace{\int_s^t \boldsymbol{J}(\boldsymbol{h}_\tau, \tau) \odot \mathbf{Z}_{N_\tau}\, \mathrm{d}N_\tau}_{\text{jump term}}.$$

(2.30)

Compared with Neural ODE model in (2.28) that only contains deterministic component (drift term), we add two extra terms in (2.30) to model different nature of randomness: diffusion term and jump term. The diffusion term consists of Brownian motion $\mathbf{B}_t$ and its coefficient $\boldsymbol{G}$ (optionally) parameterized by unknown variables $\boldsymbol{v}$. Inside the jump term, deterministic function $\boldsymbol{J}(\boldsymbol{h}_\tau, \tau)$ controls the jump size; Random variables $\mathbf{Z}_{N_\tau} \sim \text{Bernoulli}(\pm 1, p)$ controls the direction; and $N_\tau \sim \text{Poisson}(\lambda \tau)$ is a Poisson counting process controlling the "frequency" of jumps. For completeness, we include some key properties of Brownian motion to appendix, and for more systematic discussion we refer readers to related sections in [Oks03, KS98], informally we can regard the random variable $\mathrm{d}\mathbf{B}_t$ as i.i.d. Gaussian random variables with distribution $\mathcal{N}(0, \mathrm{d}t)$. Next we will explain these two terms in details.

Diffusion term. This part is an Itô integral and we know it follows Gaussian distribution. To see it more clearly, we can simply set $\boldsymbol{G}(\boldsymbol{h}_\tau, \tau) = \sigma$ and so the result of integration will be $\mathbf{B}_t - \mathbf{B}_s \sim \mathcal{N}(0, (t-s)\sigma^2)$, which is consistent with adding Gaussian noise to each residual block. For general $\boldsymbol{G}$, it does allow closed-form solution but the result is still Gaussian, only the variance is now dependent on hidden features $\boldsymbol{h}_\tau$.

Jump term. The key feature of jump term is that the integral is calculated over Poisson process N_t, that is, the total number of jumps in interval $[s, t]$ follows Poisson distribution

$$P(N_{s \to t} = n) = \frac{[\lambda(t-s)]^n}{n!} e^{-\lambda(t-s)}, \quad n \in \mathbb{Z}_+.$$

We can imagine that by inserting the jump term to our hidden state transition formula (2.30), we are effectively adding n dropout layers to the network, where n is drawn from

some Poisson distribution; and for each dropout layer, it is randomly placed to any network depth (see Fig. 2.11 to get a better picture). Additionally, for each dropout layer, the drop
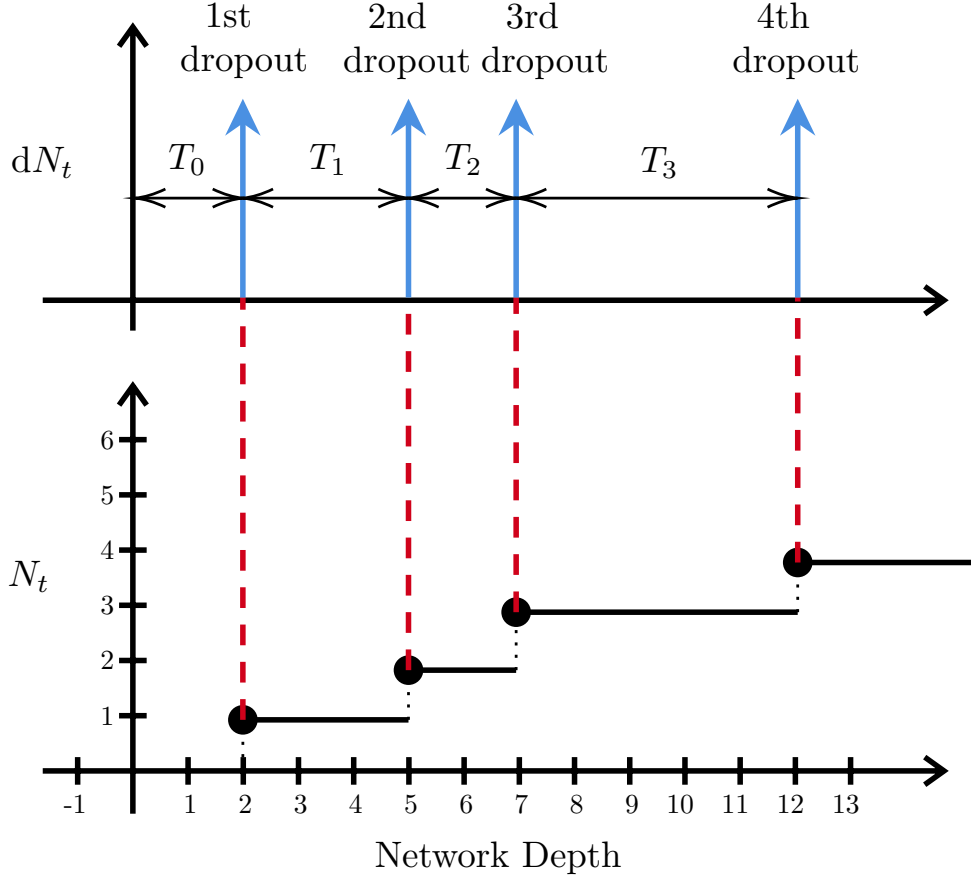
Figure 2.11: Illustrateion of dropout layers under (2.30).

probability is determined by Bernoulli random variables **Z**.

Rationale of (2.30)**.** It might not be straightforward to see why our model (2.30) is a proper replacement for the discrete version of a residual neural network equipped with Gaussian smoothing and Dropout layers. Here we make more justifications about it. We first notice that the noise pattern coming from dropout layers is very different from the noise generated by Gaussian smoothing. By definition, Dropout randomly sets some features to

zero, so the noise here is inherently Bernoulli distributed. On the other hand, the diffusion term is a Gaussian process (Itô integral). Therefore, it is not reasonable to model a dropout layer with diffusion term, nor is it suitable to model Gaussian noise with jump term. That is why we use two separate terms in our continuous framework.

2.3.1 Some concrete examples

We proposed a new framework in (2.30) for encoding the randomness into Neural ODE using diffusion and jump terms. Next we will give some concrete examples for these two terms.

Dropout. Dropout layer randomly disables some connections in neural network, here we consider a common situation where dropout layer is placed after convolutional block and before residual connection (Fig. 2.12). Mathematically we can formulate it as $\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \boldsymbol{\gamma}$,
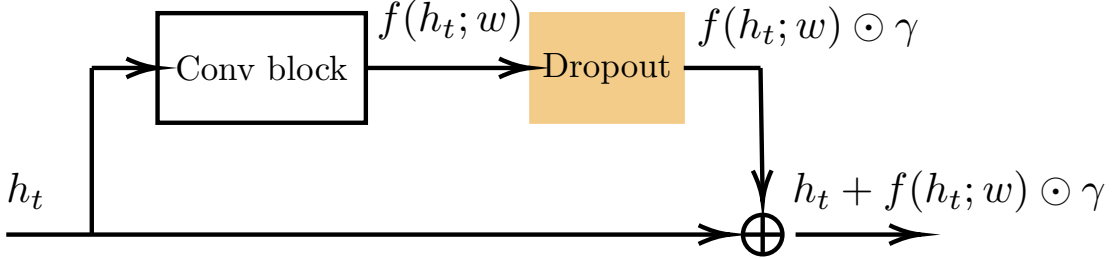
Figure 2.12: Our dropout layer configuration.

where $\boldsymbol{h}_t$ is input features at depth τ, and $P(\boldsymbol{\gamma}_i = 0) = p_{\text{drop}}$ determines the drop rate. To be compatible with (2.30), we rewrite it as

$$
\begin{aligned}
\boldsymbol{h}_{t+1} &= \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \boldsymbol{\gamma} \\
&= \boldsymbol{h}_t + \frac{1}{2}\boldsymbol{f}(\boldsymbol{h}_t; w) + 2\boldsymbol{f}(\boldsymbol{h}_t; w) \odot \frac{\boldsymbol{\gamma} - 0.5}{2}, \\
&= \boldsymbol{h}_t + \frac{1}{2}\boldsymbol{f}(\boldsymbol{h}_t; w) + 2\boldsymbol{f}(\boldsymbol{h}_t; w) \odot \mathbf{Z}
\end{aligned}
\tag{2.31}
$$

where we are essentially shifting Bernoulli random variables from $\boldsymbol{\gamma}$ (with values $\{0, 1\}$) to $\mathbf{Z}$ (with values $\{-1, +1\}$). Comparing (2.31) with (2.30), we arrive at the following continuous version of (2.31)

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \frac{1}{2} \boldsymbol{f}(\boldsymbol{h}_\tau; w) \, \mathrm{d}\tau + \int_s^t 2\boldsymbol{f}(\boldsymbol{h}_\tau; w) \odot \mathbf{Z}_{N_\tau} \, \mathrm{d}N_\tau.$$

Stochastic depth network. This is very similar to the previous dropout setting, the only difference is that for stochastic depth network, random vector $\boldsymbol{\gamma}$ is no longer i.i.d. Bernoulli distributed but "bonded" together. More formally, $\boldsymbol{\gamma} = \gamma \cdot \boldsymbol{J}$ where $\boldsymbol{J}_{i,j} = 1$ is an all-ones matrix and γ is a (scalar) Bernoulli random variable (of values $\{0, 1\}$). Beyond that, there is no difference with previous dropout layer, and the continuous form of it is

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \frac{1}{2} \boldsymbol{f}(\boldsymbol{h}_\tau; w) \, \mathrm{d}\tau + \int_s^t 2\boldsymbol{f}(\boldsymbol{h}_\tau; w) Z_{N_\tau} \, \mathrm{d}N_\tau, \tag{2.32}$$

here Z_{N_τ} is just a scalar random variable.

Gaussian-dropout. We can create another kind of dropout noise that does not involve Bernoulli random variables. We first scale the original dropout (2.31) by $1 - p_{\text{drop}}$, which becomes

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \frac{\gamma}{1 - p_{\text{drop}}}. \tag{2.33}$$

The reason we add an $1 - p_{\text{drop}}$ scaling factor is that now the output expectation $\mathbb{E}[\boldsymbol{h}_{t+1}] = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w)$ looks as if no dropout is used, due to the fact that $\mathbb{E}[\gamma] = 1 - p_{\text{drop}}$. Disentangling the mean from variance, we have

$$\begin{aligned}
\boldsymbol{h}_{t+1} &= \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) + \boldsymbol{f}(\boldsymbol{h}_t; w) \odot (\frac{\gamma}{1 - p_{\text{drop}}} - \boldsymbol{I}) \\
&= \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) + \sqrt{\frac{p_{\text{drop}}}{1 - p_{\text{drop}}}} \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \boldsymbol{z}_t,
\end{aligned} \tag{2.34}$$

where $\boldsymbol{I}$ is the identity matrix, and $\boldsymbol{z}_t \triangleq \sqrt{\frac{1 - p_{\text{drop}}}{p_{\text{drop}}}} \left(\frac{\gamma}{1 - p_{\text{drop}}} - \boldsymbol{I} \right)$. We can verify that $\boldsymbol{z}_t$ as a two-point distribution, has the same mean and variance as standard Gaussian distribution.

So as an approximation, we directly replace $\boldsymbol{z}_t$ with $\mathcal{N}(0,1)$. After that, the continuous version becomes

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau; w)\, \mathrm{d}\tau + \int_s^t \sqrt{\frac{p_{\mathrm{drop}}}{1 - p_{\mathrm{drop}}}} \boldsymbol{f}(\boldsymbol{h}_t; w) \odot \mathrm{d}\mathbf{B}_\tau.$$

Gaussian smoothing. As we have mentioned before, Gaussian noise is better modeled by diffusion term. The traditional way of applying Gaussian smoothing is adding small, uncorrelated noise to each hidden layer [LCZ18] (or just the input layer [LAG18b, CRK19]), mathematically

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) + \mathbf{W}_t, \quad \mathbf{W}_t \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}). \tag{2.35}$$

But through experiments we found that multiplicative noise of following form also works

$$\boldsymbol{h}_{t+1} = \boldsymbol{h}_t + \boldsymbol{f}(\boldsymbol{h}_t; w) + \boldsymbol{f}(\boldsymbol{h}_t; w)\mathbf{W}_t, \quad \mathbf{W}_t \sim \mathcal{N}(\mathbf{0}, \sigma^2 \boldsymbol{I}). \tag{2.36}$$

Unlike (2.35), the noise scale in (2.36) grows with the scale of output from convolution block $\boldsymbol{f}(\boldsymbol{h}_t; w)$, thus the noise variance is self-adjustable and sometimes it can be advantageous. For both additive and multiplicative Gaussian noise, the integral form is as straightforward as follows

$$\boldsymbol{h}_t = \boldsymbol{h}_s + \int_s^t \boldsymbol{f}(\boldsymbol{h}_\tau; w)\, \mathrm{d}\tau + \begin{cases} \int_s^t \boldsymbol{G}(\tau)\, \mathrm{d}\mathbf{B}_\tau, \\[2mm] \int_s^t \boldsymbol{G}(\boldsymbol{h}_t, \tau)\, \mathrm{d}\mathbf{B}_\tau. \end{cases} \tag{2.37}$$

As we can see, for additive noise, diffusion coefficient $\boldsymbol{G}$ is independent on $\boldsymbol{h}_t$; while for multiplicative noise, $\boldsymbol{G}$ can change along with hidden features $\boldsymbol{h}_t$.

2.3.2 Training algorithm and complexity

The implementation of the stochastic, continuous neural network training algorithm is similar to Neural ODE [CRB18] and it is stated in Algorithm 4. In fact, we can view (2.30) as a SDE problem, and standard SDE solvers can be applied here for training. We can see from the algorithm that for forward propagation we pick a standard SDE solvers such as

Euler-Maruyama [KP13], Milstein [Mil75] or higher order Runge-Kutta method, but for backward propagation we simply rely on the automatic gradient provided by major deep learning frameworks. Although it is possible to derive an adjoint algorithm as in [CRB18]

Algorithm 4 Forward and backward propagation

1: **procedure** TRAINING-PROCESS   ▷ Do forward & backward propagation

2:     Given initial state $\boldsymbol{h}_0$, integral range $[0, T]$.

3:     $\boldsymbol{h}_T = \text{SDE\_Solve}\Big(\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w}), \boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v}), [0, T]\Big)$.   ▷ Call a black-box SDE solver

4:     Calculate loss $L = \ell(\boldsymbol{h}_T)$.

5:     Calculate gradient $\frac{\partial L}{\partial \boldsymbol{w}}$ and $\frac{\partial L}{\partial \boldsymbol{v}}$ with autograd.

6:     Update network parameters $\boldsymbol{w}$ and $\boldsymbol{v}$.

7: **end procedure**

---

to save memory consumption, in practice, we find that the most straightforward autograd method works efficiently in all our experiments, see Fig. 2.13(left). Furthermore, we observe the error caused by discretization is small enough for end tasks even when using a large grid size in SDE solver (Line 3 in Algorithm 4) as shown in Fig 2.13(right). More details are in the appendix.

2.3.3   Connection between jump-diffusion and robustness

In this section, we build a new explanation to shed some light on answering how noise (inside jump-diffusion term (2.30)) helps training the robust neural network. It is worth noting that our analysis is very different from the traditional belief that noise acts as a regularizer during training. We focus on the role of randomness in testing time, and in this sense, our idea is complementary to the former. In the following parts, we first deliver a toy example to have a closer view of this phenomenon, and then we present some theories to understand the principles in behind.
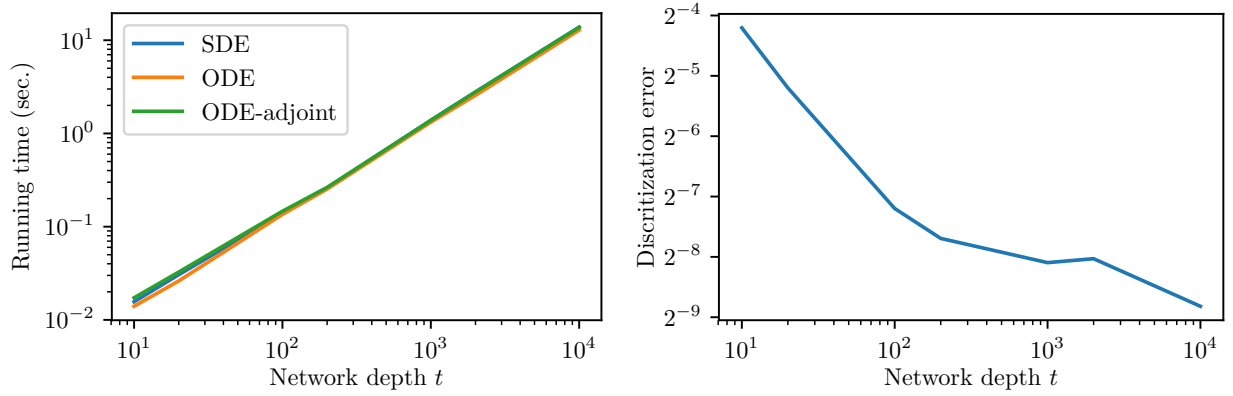
Figure 2.13: *Left*: we compare the propagation time between Neural ODE, ODE with adjoint, and SDE. We can see that the running time increases proportionally with network depth and there is no significant overhead in our model. *Right*: We compute the error of SDE solver caused by discretization in Euler schemes, measured by the relative error in h_t, i.e. $\varepsilon = \frac{\|h_T - \hat{h}_T\|}{\|h_T\|}$ and h_T is the ground-truth (computed with a very fine grid), $\hat{h}_T$ is computed with coarse grid $\Delta t \in [1.0 \times 10^{-4}, 1.0 \times 10^{-1}]$ (note that network depth $t = T/\Delta T$).

A toy example

Let's look at a 1-dimensional toy example where randomness stabilizes the system. Suppose we have a simple SDE

$$\mathrm{d}x_t = x_t \, \mathrm{d}t + \sigma x_t \, \mathrm{d}\mathbf{B}_t, \tag{2.38}$$

with $\mathbf{B}_t$ being the standard Brownian motion. When we remove the diffusion term by setting $\sigma = 0$, (2.38) becomes an ODE: $\mathrm{d}x_t = x_t \, \mathrm{d}t$ with solution $x_t = x_0 e^t$, where x_0 is the initialization of x_t. If $x_0 \neq 0$, we can see that $x_t \to \pm\infty$ as $t \to \infty$. In other words, any small perturbation at initialization $x_0 = \epsilon$ will be amplified through the ODE-system at future time t. In contrast, if we add the diffusion back $\sigma \neq 0$, we then have the classic geometric Brownian motion with solution $x_t = x_0 \exp\left((1 - \sigma^2/2)t + \sigma B_t\right)$. Once the variance of noise is large enough (e.g. $\sigma > \sqrt{2}$), then we know that $x_t \overset{\text{a.s.}}{\to} 0$.

To visualize the difference, we run several numerical simulations in Fig. 2.14 for x_t with

different variances σ. The experiments in Fig. 2.14 clearly show that the behavior of solution
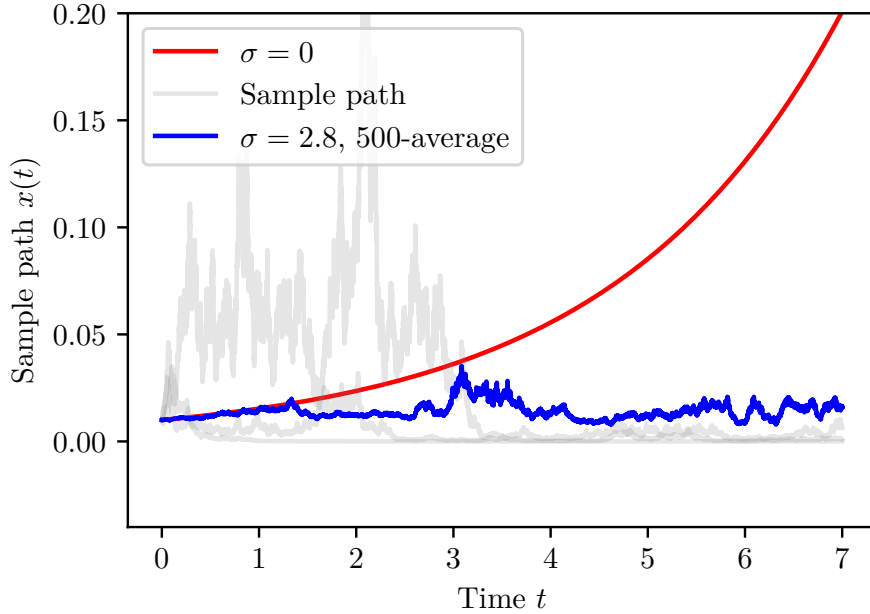
Figure 2.14: Toy example. By comparing the simulations under $\sigma = 0$ and $\sigma = 2.8$, we see adding noise to the system can be an effective way to control x_t. Average over multiple runs is used to cancel out the volatility during the early stage. It is noteworthy that here we employ the multiplicative noise, where the deviation term scales proportionally to x_t.

paths can change significantly after adding a diffusion term. This example is inspiring because we can control the impact of perturbations on the output by adding a stochastic term to our networks.

Theoretical explanation

Inspired by the toy example above, we theoretically analyze the stability of Neural jump-diffusion equation (2.30). Our analytical results show that the jump-diffusion terms can indeed improve the robustness of the model against small, arbitrary input perturbations. This finding also explains why noise injection can improve both generalizability and robustness in discrete networks, which has been observed in current literature [LCZ18, LAG18b]. To

simplify our symbols, we ignore the jump term temporarily and focus on the diffusion term. The following assumptions on drift $\boldsymbol{f}$ and diffusion $\boldsymbol{G}$ guarantee the existence and uniqueness of solution.

Assumption 1. $\boldsymbol{f}$ *and* $\boldsymbol{G}$ *are at most linear, i.e.* $\|\boldsymbol{f}(\boldsymbol{x},t)\| + \|\boldsymbol{G}(\boldsymbol{x},t)\| \leq c_1(1 + \|\boldsymbol{x}\|)$ *for* $c_1 > 0$, $\forall \boldsymbol{x} \in \mathbb{R}^n$ *and* $t \in \mathbb{R}^+$.

Assumption 2. $\boldsymbol{f}$ *and* $\boldsymbol{G}$ *are* c_2-*Lipschitz:* $\|\boldsymbol{f}(\boldsymbol{x},t) - \boldsymbol{f}(\boldsymbol{y},t)\| + \|\boldsymbol{G}(\boldsymbol{x},t) - \boldsymbol{G}(\boldsymbol{y},t)\| \leq c_2\|\boldsymbol{x} - \boldsymbol{y}\|$ *for* $c_2 > 0$, $\forall \boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$ *and* $t \in \mathbb{R}^+$.

Based on the above assumptions, we can show that the SDE (2.30) has a unique solution [Oks03]. We remark that these assumptions on $\boldsymbol{f}$ are quite natural and are also enforced on the original Neural ODE model (see Sec. 6 of [CRB18]). As to the diffusion matrix $\boldsymbol{G}$, we have seen that at least for additive Gaussian noise (where $\boldsymbol{G}$ is a constant matrix) and multiplicative Gaussian noise (where $\boldsymbol{G}$ is proportional to $\boldsymbol{f}$), both assumptions are automatically satisfied as long as $\boldsymbol{f}$ possesses the same regularities.

We analyze the dynamics of perturbation. Our analysis applies not only to the Neural SDE model but also to the Neural ODE model, by setting the diffusion term $\boldsymbol{G}$ and jump term $\boldsymbol{J}$ to zero. Our idea is illustrated in Fig. 2.15. First of all, we consider initializing our differential equation (2.30) at two slightly different values $\boldsymbol{h}_0$ and $\boldsymbol{h}_0^e = \boldsymbol{h}_0 + \boldsymbol{\varepsilon}_0$, where $\boldsymbol{h}_0$ is the original (clean) input, $\boldsymbol{\varepsilon}_0$ is the perturbation (also called "error") on $\boldsymbol{h}_0$. In many real problems, the perturbation at input is bounded, i.e. $\|\boldsymbol{\varepsilon}_0\| \leq \delta$. Therefore, under the perturbed initialization $\boldsymbol{h}_0^e$, the hidden states at time t follow the same rule in (2.30), recalling the jump term is ignored for simplicity,

$$\mathrm{d}\boldsymbol{h}_t^e = \boldsymbol{f}(\boldsymbol{h}_t^e, t; \boldsymbol{w})\,\mathrm{d}t + \boldsymbol{G}(\boldsymbol{h}_t^e, t)\,\mathrm{d}\mathbf{B}_t', \text{with } \boldsymbol{h}_0^e = \boldsymbol{h}_0 + \boldsymbol{\varepsilon}_0, \tag{2.39}$$

where $\mathbf{B}_t'$ is Brownian motions for the SDE associated with initialization $\boldsymbol{h}_0^e$. Then it is natural to analyze how the perturbation $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ evolves in the long run. Subtracting
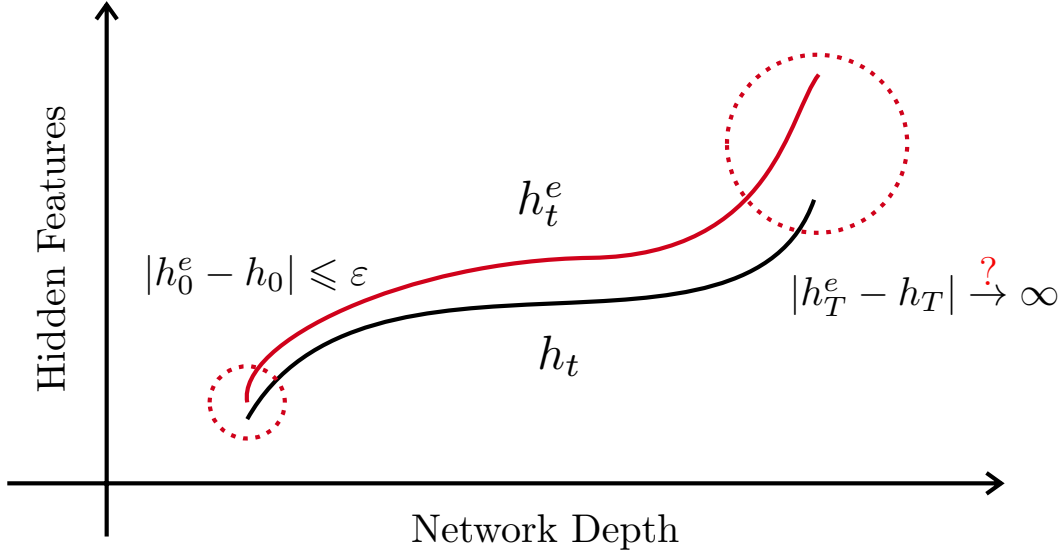
Figure 2.15: Illustration of our analysis. Given a smaller perturbation ε at the input, how does the error propagation through a deep neural network? If the error is controllable, then we can make sure that the final prediction result is also controllable. In our analysis, we do not need to care about how $\boldsymbol{h}_t$ or $\boldsymbol{h}_t^e$ evolves, only the difference $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ matters; and this is depicted in (2.40).

(2.30) from (2.39), we have

$$
\begin{aligned}
\mathrm{d}\boldsymbol{\varepsilon}_t &= \Big[\boldsymbol{f}(\boldsymbol{h}_t^e, t; \boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\Big]\, \mathrm{d}t \\
&\quad + \Big[\boldsymbol{G}(\boldsymbol{h}_t^e, t) - \boldsymbol{G}(\boldsymbol{h}_t, t)\Big]\, \mathrm{d}\mathbf{B}_t \\
&= \boldsymbol{f}_\Delta(\boldsymbol{\varepsilon}_t, t; \boldsymbol{w})\, \mathrm{d}t + \boldsymbol{G}_\Delta(\boldsymbol{\varepsilon}_t, t)\, \mathrm{d}\mathbf{B}_t.
\end{aligned}
\tag{2.40}
$$

Here we made an implicit assumption that the Brownian motions $\mathbf{B}_t$ and $\mathbf{B}_t'$ have the *same* sample path for both initialization $\boldsymbol{h}_0$ and $\boldsymbol{h}_0^e$, i.e. $\mathbf{B}_t = \mathbf{B}_t'$ w.p.1. In other words, we focus on the difference of two random processes $\boldsymbol{h}_t$ and $\boldsymbol{h}_t^e$ driven by the same underlying Brownian motion. So it is valid to subtract the diffusion terms.

An important property of (2.40) is that it admits a trivial solution $\boldsymbol{\varepsilon}_t \equiv \boldsymbol{0}$, $\forall t \in \mathbb{R}^+$ and $\boldsymbol{w} \in \mathbb{R}^d$. To verify that, we only need to show that both the drift ($\boldsymbol{f}$) and diffusion ($\boldsymbol{G}$) are

Table 2.5: Evaluating the model generalization under different choices of diffusion matrix $\boldsymbol{G}(\boldsymbol{h}_t, t; \boldsymbol{v})$ introduced above. For the three noise types, we search a suitable parameter σ_t for each of them so that the diffusion matrix $\boldsymbol{G}$ properly regularizes the model. TTN means testing time noise. We observe adding noises can improve the test accuracy over Neural ODE, and furthermore, noise at testing time is beneficial.

Data	Accuracy@1 — w/o TTN				Accuracy@1 — w/ TTN			
	ODE	Additive	Multiplicative	Dropout	ODE	Additive	Multiplicative	Dropout
CIFAR-10	87.95	88.69	89.06	88.23	–	88.73	**89.77**	88.44
CIFAR-10.1	70.00	70.80	71.50	71.85	–	71.70	72.05	**73.60**
STL-10	58.03	61.23	60.54	61.26	–	62.11	**62.58**	62.13
Tiny-ImageNet	45.19	45.25	46.94	47.04	–	45.39	46.65	**47.81**

zero under $\boldsymbol{\varepsilon}_t = 0$:

$$\boldsymbol{f}_\Delta(\boldsymbol{0}, t; \boldsymbol{w}) = \boldsymbol{f}(\boldsymbol{h}_t + \boldsymbol{0}, t; \boldsymbol{w}) - \boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w}) = 0,$$
$$\boldsymbol{G}_\Delta(\boldsymbol{0}, t) = \boldsymbol{G}(\boldsymbol{h}_t + \boldsymbol{0}, t) - \boldsymbol{G}(\boldsymbol{h}_t, t) = 0.$$

(2.41)

The implication of zero solution is clear: *for a neural network, if we do not perturb the input data, then the output will never change.* However, the solution $\boldsymbol{\varepsilon}_t = \boldsymbol{0}$ can be **highly unstable**, in the sense that for an arbitrarily small perturbation $\boldsymbol{\varepsilon}_0 \neq \boldsymbol{0}$ at initialization, the change of output $\boldsymbol{\varepsilon}_T$ can be arbitrarily large. Luckily, as we will show below, by choosing the diffusion term $\boldsymbol{G}$ properly, we can always control $\boldsymbol{\varepsilon}_t$ within a small range.

In general, we cannot get the closed-form solution to a multidimensional SDE, but we can still analyze the asymptotic stability through the dynamics $\boldsymbol{f}$ and $\boldsymbol{G}$. This is an extension of the Lyapunov stability theory to a stochastic system. First, we define the notion of stability in the stochastic case. Let $(\Omega, \mathcal{F}, P)$ be a complete probability space with filtration $\{\mathcal{F}_t\}_{t \geq 0}$ and $\mathbf{B}_t$ be an m-dimensional Brownian motion defined in the probability space, we consider

the SDE in (2.40) with initial value ε_0

$$\mathrm{d}\varepsilon_t = \boldsymbol{f}_\Delta(\varepsilon_t, t)\,\mathrm{d}t + \boldsymbol{G}_\Delta(\varepsilon_t, t)\,\mathrm{d}\mathbf{B}_t, \tag{2.42}$$

where for simplicity we dropped the dependency on parameters $\boldsymbol{w}$ and $\boldsymbol{v}$. We further assume $\boldsymbol{f}_\Delta : \mathbb{R}^n \times \mathbb{R}_+ \mapsto \mathbb{R}^n$ and $\boldsymbol{G}_\Delta : \mathbb{R}^n \times \mathbb{R}_+ \mapsto \mathbb{R}^{n \times m}$ are both Borel measurable. We can show that if assumptions (1) and (2) hold for $\boldsymbol{f}$ and $\boldsymbol{G}$, then they hold for $\boldsymbol{f}_\Delta$ and $\boldsymbol{G}_\Delta$ as well (see appendix), and we know the SDE (2.42) allows a unique solution ε_t. We have the following Lynapunov stability results from [Mao07].

Definition 2.3.1 (Lyapunov stability of SDE). *The solution $\varepsilon_t = \boldsymbol{0}$ of* (2.42):

  A. *is <u>stochastically stable</u> if for any $\alpha \in (0,1)$ and $r > 0$, there exists a $\delta = \delta(\alpha, r) > 0$ such that $\Pr\{\|\varepsilon_t\| < r \text{ for all } t \geq 0\} \geq 1 - \alpha$ whenever $\|\varepsilon_0\| \leq \delta$. Moreover, if for any $\alpha \in (0,1)$, there exists a $\delta = \delta(\alpha) > 0$ such that $\Pr\{\lim_{t \to \infty} \|\varepsilon_t\| = 0\} \geq 1 - \alpha$ whenever $\|\varepsilon_0\| \leq \delta$, it is said to be <u>stochastically asymptotically stable</u>;*

  B. *is <u>almost surely exponentially stable</u> if for all $\varepsilon_0 \in \mathbb{R}^n$, $\limsup\limits_{t \to \infty} \frac{1}{t} \log \|\varepsilon_t\| < 0$ a.s.[3]*

Note that for part A in Definition 2.3.1, it is hard to quantify how well the stability is and how fast the solution reaches equilibrium. In addition, under assumptions (1, 2), we have a straightforward result $\Pr\{\varepsilon_t \neq \boldsymbol{0} \text{ for all } t \geq 0\} = 1$ whenever $\varepsilon_0 \neq \boldsymbol{0}$ as shown in appendix. That is, almost all the sample paths starting from a non-zero initialization can never reach zero due to Brownian motion. On the contrary, the almost sure exponentially stability result implies that almost all the sample paths of the solution will be close to zero exponentially fast. One important result regarding to stability of this system is [Mao07], deferred to appendix. We now consider a special case, when the noise is multiplicative $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma \cdot \boldsymbol{h}_t$ and $m = 1$. The corresponding SDE of perturbation $\varepsilon_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ becomes

$$\mathrm{d}\varepsilon_t = \boldsymbol{f}_\Delta(\varepsilon_t, t; \boldsymbol{w})\,\mathrm{d}t + \sigma \cdot \varepsilon_t\,\mathrm{d}\mathbf{B}_t. \tag{2.43}$$

---

[3] *"a.s." is the abbreviation for "almost surely".*

Table 2.6: Testing accuracy results under different levels of non-adversarial perturbations.

Data	Noise type	mild corrupt ← Accuracy → severe corrupt				
		Level 1	Level 2	Level 3	Level 4	Level 5
CIFAR10-C[†]	ODE	75.89	70.59	66.52	60.91	53.02
	Dropout	77.02	71.58	67.21	61.61	53.81
	Dropout+TTN	**79.07**	**73.98**	**69.74**	**64.19**	**55.99**
TinyImageNet-C[†]	ODE	23.01	19.18	15.20	**12.20**	**9.88**
	Dropout	22.85	18.94	14.64	11.54	9.09
	Dropout+TTN	**23.84**	**19.89**	**15.28**	12.08	9.44

[†] Downloaded from `https://github.com/hendrycks/robustness`

Note that for the deterministic case of (2.43) by setting $\sigma \equiv 0$, the solution may not be stable in certain cases (see Figure 2.14). Whereas for general cases when $\sigma > 0$, following corollary claims that by setting σ properly, we will achieve an (almost surely) exponentially stable system.

Corollary 2.1. *For* (2.43), *if* $\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})$ *is L-Lipschtiz continuous w.r.t.* $\boldsymbol{h}_t$, *then* (2.43) *has a unique solution with the property* $\limsup\limits_{t \to \infty} \frac{1}{t} \log \|\boldsymbol{\varepsilon}_t\| \le -(\frac{\sigma^2}{2} - L)$ *almost surely for any* $\boldsymbol{\varepsilon}_0 \in \mathbb{R}^n$. *In particular, if* $\sigma^2 > 2L$, *the solution* $\boldsymbol{\varepsilon}_t = \boldsymbol{0}$ *is almost surely exponentially stable.*

2.3.4 Experiment

In this section, we show the effectiveness of our framework in terms of generalization, non-adversarial robustness, and adversarial robustness. Throughout our experiments, we set $\boldsymbol{f}(\cdot)$ to be a neural network with several convolution blocks. As to $\boldsymbol{G}(\cdot)$ we have the following choices:
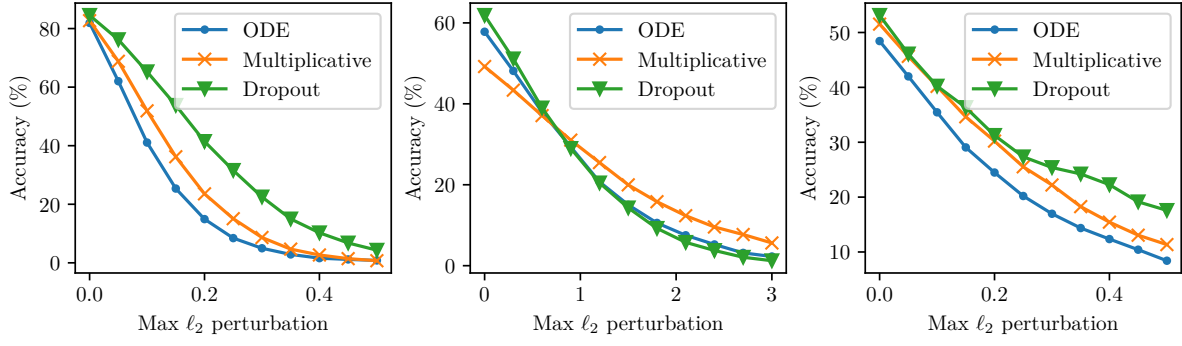
Figure 2.16: Comparing the robustness against ℓ_2-norm constrained adversarial perturbations, on CIFAR-10 (left), STL-10 (middle) and Tiny-ImageNet (right) data. We observe that jump-diffusion model with either multiplicative noise or dropout noise is more resistant to adversarial attack than Neural ODE.

- **Neural ODE**, this can be done by dropping the diffusion term $\boldsymbol{G}(\boldsymbol{h}_t, t) = \boldsymbol{0}$.

- **Additive noise**, when the diffusion term is independent of $\boldsymbol{h}_t$, here we simply set it to be diagonal $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma_t \boldsymbol{I}$.

- **Multiplicative noise**, when the diffusion term is proportional to $\boldsymbol{h}_t$, or $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma_t \boldsymbol{h}_t$.

- **Gaussian-dropout noise**, when the diffusion term is proportional to the drift term $\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})$, i.e. $\boldsymbol{G}(\boldsymbol{h}_t, t) = \sigma_t \mathrm{diag}\{\boldsymbol{f}(\boldsymbol{h}_t, t; \boldsymbol{w})\}$.

Note the last three are our proposed model with different types of randomness, as explained in Section 3.1. For more experimental details, the architecture of $\boldsymbol{f}(\cdot)$ and the numerical solver for SDE, please refer to our appendix. Note that we use the same architecture for all the methods mentioned above, so the comparisons are fair.

Generalization Performance

In the first experiment, we show a small noise helps generalization. However, note that our noise injection is different from the randomness layer in the discrete case. For instance,

dropout layers add Bernoulli noise at training time but not testing time, whereas our model keeps randomness at the testing time and takes the averaged prediction of multiple forward propagations.

As for datasets, we choose CIFAR-10, STL-10 and Tiny-ImageNet[4] to include various sizes and number of classes. The experimental results are shown in Table 2.5. We observe that for all datasets, noisy versions consistently outperform ODE, and the reason is that adding moderate noise to the models at training time can act as a regularizer and thus improves testing accuracy. Based upon that, if we further keep testing time noise and ensemble the outputs, we will obtain even better results.

Improved non-adversarial robustness

We evaluate the robustness of models under non-adversarial corruption following the idea of [HD19b]. The corrupted datasets contain tens of defects in photography, including motion blur, Gaussian noise, fog, etc. For each noise type, we run Neural ODE and our model with dropout noise and gather the testing accuracy. The final results are reported by mean accuracy (mAcc) in Table 2.6 by changing the level of corruption. Both models are trained on clean data, which means the corrupted images are not visible to them during the training stage, nor could they augment the training set with the same types of corruption. From the table, we can see that our model performs better than Neural ODE in 8 out of 10 cases. For the rest two, both ODE and SDE are performing very close. This shows that our proposed neural jump-diffusion improves the robustness of Neural ODE under non-adversarial corrupted data.

---

[4]Downloaded from `https://tiny-imagenet.herokuapp.com/`

Improved adversarial robustness

Next, we consider the performance of our models under adversarial perturbation. This scenario is strictly harder than the previous case – perturbations are crafted through constrained loss maximization procedure, so it represents the worst-case performance. In our experiment, we adopt ℓ_2-PGD attack with 20 steps [MMS18a]. The experimental results are shown in Figure 2.16. As we can see, jump-diffusion model with either multiplicative noise or dropout noise is more resistant to adversarial attack than Neural ODE. We also observe dropout noise outperforms multiplicative noise.

Visualizing the perturbations of hidden states
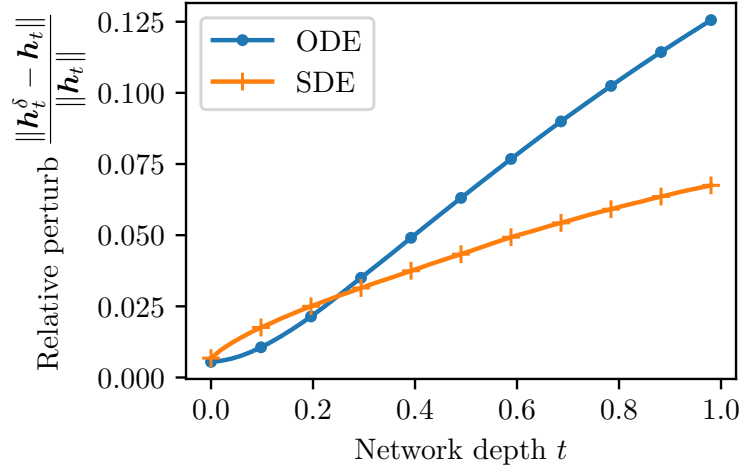
Figure 2.17: Comparing the perturbations of hidden states, $\boldsymbol{\varepsilon}_t$, on both ODE and SDE (we choose dropout-style noise).

In this experiment, we take a look at the perturbation $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ at any time t. Recall that in the 1-d toy example in Figure 2.14, we observe that the perturbation at time t can be well suppressed by adding a strong diffusion term, which is also confirmed by our theorem. However, it is still questionable whether the same phenomenon also exists in deep neural networks since we cannot add very large noise to the network during training or testing time.

If the noise is too large, it will also remove all useful features. Thus it becomes important to make sure that this will not happen to our models. To this end, we first sample an input $\boldsymbol{x}$ from CIFAR-10 and gather all the hidden states $\boldsymbol{h}_t$ at time $t = [0, \Delta t, 2\Delta t, \ldots, N\Delta t]$. Then we perform regular PGD attack [MMS18a] to find the perturbation $\boldsymbol{\delta}_x$ such that $\boldsymbol{x}_{\mathrm{adv}} = \boldsymbol{x} + \boldsymbol{\delta}_x$ is an adversarial image, and feed the new data $\boldsymbol{x}_{\mathrm{adv}}$ into network again so we get $\boldsymbol{h}_t^e$ at the same time stamps as $\boldsymbol{h}_t$. Finally we plot the error $\boldsymbol{\varepsilon}_t = \boldsymbol{h}_t^e - \boldsymbol{h}_t$ $w.r.t.$ time t (also called "network depth"), shown in Figure 2.17. We can observe that by adding a diffusion term (dropout-style noise), the error accumulates much slower than the ordinary Neural ODE model.

CHAPTER 3

Verifiable Adversarial Defense

3.1 Provably Robust Nearest-neighbor Classifiers

In the previous chapters we went through some randomization based adversarial defense algorithms. A clear drawback of these algorithms is that they are all attack-dependent, meaning that we never know the actual performance under other attacks even if it works well under the PGD attack. Verifiable adversarial defense, on the other hand, provides a worse case accuracy regardless of what adversarial attack algorithm is chosen.

We study the problem of evaluating the robustness of Nearest Neighbor (NN) classifiers. NN models are simple but effective, and are widely-used in many domains [BSI08, WS09, XKS06]. Furthermore, NN models have been used as a defense mechanism recently since they are believed to be more difficult to attack than neural networks [DMY19, PM18]. We aim to answer the following questions:

1) Is NN really more robust or it's just because previous methods fail to compute the minimum adversarial perturbation? 2) Is the robustness of NN verifiable?

As a non-continuous step function, NN classifiers are quite different from neural networks. As a result, the methods used for neural network attack and verification cannot be directly applied to NN classifiers. Previous attempts on attacking nearest neighbor models either resort to some simple heuristics [SW19] or apply gradient-based attacks to some continuous substitute models of NN [SW19, PMG16c, DMY19]. Unfortunately, these methods are far from optimal and have no theoretical guarantee. To the best of our knowledge, there is

no existing approach on exactly computing the minimum adversarial perturbation that can change an NN classifier's output, and there is even no existing verification method that can compute a meaningful lower bound for the range of the safe region of an NN classifier.

To address the above issues, we first study the 1-NN classifier and show that finding the minimum adversarial perturbation can be formulated as a set of convex quadratic programming (QP) problems, of which the optimal solution can be computed in **polynomial time** [KTK80]. This is quite different from neural networks or tree-based models for which finding the minimum perturbation has proven to be NP-hard [KBD17, KTJ16]. In particular, our formulation provides a clear and novel perspective on attack and verification for nearest neighbor classifiers: any feasible solution of any primal QP problem will be a successful attack, and the minimum of any set of feasible solutions of the dual problems are guaranteed lower bounds of the minimum adversarial perturbation. Moreover, the primal minimum and the dual maximum will exactly match at the value of the minimum adversarial perturbation. We show that the QP problems can be efficiently solved by greedy coordinate ascent, which exploits the sparsity of the optimal solutions. Based on this primal-dual perspective, we further provide several screening rules to speed up the process for solving the set of QP problems.

For general K-NN models with $K > 1$, the number of QP problems will grow exponentially with K. Therefore, computing the exact minimum adversarial perturbation is prohibitively time consuming when K is large. On the one hand, we address this issue by approximately solving a subset of the primal problems, leading to an efficient attack method. On the other hand, we propose an efficient method to deal with the dual problems for providing a tight lower bound of the minimum adversarial perturbation. This method does not need to exactly solve any dual problems, leading to an efficient K-NN verification algorithm with time complexity independent to K.

In summary, our main contributions of this work are as follows:

- For 1-NN models, our proposed algorithm can efficiently compute the minimum adversarial perturbation. Our algorithm is provably optimal, achieves much smaller values, and is more efficient than previous attack methods. Besides, this is the first robustness verification method for NN models.

- For K-NN models with $K > 1$, our formulation provides an efficient attack algorithm, which outperforms previous attack methods. More importantly, our dual problems lead to an efficient verification algorithm to compute the lower bound of the minimum adversarial perturbation and have time complexity independent to K. Experiments show that the bounds are reasonably tight.

- Equipped with our algorithms, we accurately compute the certified robust errors of the 1-NN model on the MNIST and Fashion-MNIST datasets. We find that a simple 1-NN model can achieve better robust errors than randomly smoothed [CRK19] neural networks on these data.

3.1.1 Background and motivation

First, we set up notations for the Nearest Neighbor (NN) classifiers. Assume there are C labels in total. We use $\{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ to denote the database where each $\boldsymbol{x}_i \in \mathbb{R}^d$ is a d-dimensional vector and $y_i \in [C]$ is the corresponding label. A K-NN classifier $f : \mathbb{R}^d \to [C]$ maps a test instance to a predicted label. Given a test instance $\boldsymbol{z} \in \mathbb{R}^d$, the classifier will first identify the K-nearest neighbors $\{\boldsymbol{x}_{\pi(1)}, \ldots, \boldsymbol{x}_{\pi(K)}\}$ based on the Euclidean distance $\|\boldsymbol{x}_i - \boldsymbol{z}\|$ and then predict the final label by majority voting among $\{y_{\pi(1)}, \ldots, y_{\pi(K)}\}$.

Next we define the notions of adversarial robustness, attack, and verification. Given a correctly-classified test instance $\boldsymbol{z}$, an adversarial perturbation is defined as $\boldsymbol{\delta} \in \mathbb{R}^d$ such that $f(\boldsymbol{z} + \boldsymbol{\delta}) \neq f(\boldsymbol{z})$. An attack algorithm aims to find the minimum-norm adversarial perturbation, and its norm is formulated as

$$\epsilon^* = \left\{ \min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\| \quad \text{s.t.} \ f(\boldsymbol{z} + \boldsymbol{\delta}) \neq f(\boldsymbol{z}) \right\}. \tag{3.1}$$

A verification algorithm seeks to find a lower bound of ϵ^*, denoted as r, which defines a safe region such that

$$f(\boldsymbol{z} + \boldsymbol{\delta}) = f(\boldsymbol{z}), \ \forall \|\boldsymbol{\delta}\| \leq r. \tag{3.2}$$

Clearly, the maximum lower bound r^* will exactly match with the minimum perturbation norm ϵ^* if both attack and verification are optimal. We will focus on ℓ_2 norm but will briefly talk about how to extend to ℓ_∞ and ℓ_1 norms. Moreover, we will focus on 1-NN first and then generalize to K-NN with $K > 1$.

Failure cases of previous attack methods. At first glance, the minimum adversarial perturbation seems to be easy to compute for the 1-NN model. For instance, an approach is proposed [SW19] to find the best adversarial example on the straight line connecting $\boldsymbol{z}$ and one of the training instances belonging to a different class. Unfortunately, in Figure 3.1, we show that the optimal perturbation may not be on the lines connecting these two points; furthermore, only checking the line segments can find an arbitrary bad solution. Other previous approaches try to form a continuous approximation of NN classifiers and craft adversarial perturbations for this substitute via gradient-based attack methods [SW19, PMG16c, DMY19]; clearly, they cannot provably find the optimal perturbation.

Connection to Voronoi diagrams and a solution for low-dimensional cases In fact, the decision boundary of a 1-NN model can be captured by the Voronoi diagram (see Figure 3.1(c)). In the Voronoi diagram, each training instance $\boldsymbol{x}_i$ forms a cell, and the decision boundary of the cell is captured by the convex boundary formed by bisecting hyperplanes between $\boldsymbol{x}_i$ and its neighbors. One can thus obtain the minimum adversarial perturbation by computing the distances from $\boldsymbol{z}$ to all the cells with $y_i \neq f(\boldsymbol{z})$. However, to compute the distance, we need to check all the faces (captured by one bisecting hyperplane) and angles (intersections of more than one bisecting hyperplanes) of the cell. For 2-dimensional space ($d = 2$), each cell can only have finite faces and angles [AK99], and there exists a
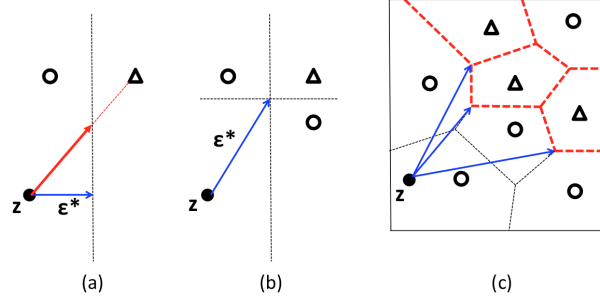
Figure 3.1: Illustration of the minimum adversarial perturbation for 1-NN model. The goal is to perturb z to be classified as a triangle. In (a), the red arrow is the perturbation computed by a naive approach used in a previous paper while the optimal solution (blue perturbation with the norm ϵ^*) could be much better, and the ratio can be arbitrary large by moving z downward. (b) shows that projection to the bisecting hyperplanes may not be optimal; one also needs to consider intersections of several bisectors which can be exponentially many. (c) shows that the optimal perturbation can be computed by evaluating the distance from z to each Voronoi cell of triangle instances.

polynomial-time algorithm for computing a Voronoi diagram. In general, for d-dimensional problems with n points, Voronoi diagram computation requires $O(n \log n + n^{\lceil \frac{d}{2} \rceil})$ time, which only works for low-dimensional problems — since the time complexity grows exponentially with the dimension d, it is prohibitively hard to use this algorithm unless d is very small.

3.1.2 Primal-dual quadratic programming

Finding the minimum adversarial perturbation is usually difficult. For example, computing minimum adversarial perturbations for ReLU networks and tree ensembles are both NP-hard [KBD17, KTJ16]. As discussed in the previous section, we can connect the 1-NN case to the Voronoi diagram computation, but the solver will require exponential time with respect to dimensionality. Therefore, a natural question is "Is it also NP-hard to compute the minimum adversarial perturbation for 1-NN models?". Surprisingly, it is not the case as we will demonstrate in this section below.

(Primal) Quadratic Programming Problems for Minimum Adversarial Perturbation

We consider the 1-NN classifier first. For a given correctly classified instance $\boldsymbol{z}$, we partition the training dataset into two disjoint sets $\mathbb{S}^+ = \{\boldsymbol{x}_1^+, \ldots, \boldsymbol{x}_{n^+}^+\}$ and $\mathbb{S}^- = \{\boldsymbol{x}_1^-, \ldots, \boldsymbol{x}_{n^-}^-\}$ such that all instances in $\mathbb{S}^+$ have the same label with $\boldsymbol{z}$ and $\mathbb{S}^-$ contains all the other instances; accordingly, we have $n = n^+ + n^-$. If we aim to make $\boldsymbol{z} + \boldsymbol{\delta}$ be closer to a particular $\boldsymbol{x}_j^-$ than all instances in $\mathbb{S}^+$, then this problem of finding the minimum perturbation can be formulated as an optimization problem:

$$\epsilon^{(j)} = \min_{\boldsymbol{\delta}} \frac{1}{2}\boldsymbol{\delta}^T\boldsymbol{\delta}$$
$$\text{s.t. } \|\boldsymbol{z} + \boldsymbol{\delta} - \boldsymbol{x}_j^-\|^2 \leq \|\boldsymbol{z} + \boldsymbol{\delta} - \boldsymbol{x}_i^+\|^2, \ \forall i \in [n^+]. \tag{3.3}$$

Each constraint can be rewritten as

$$\boldsymbol{\delta}^T(\boldsymbol{x}_j^- - \boldsymbol{x}_i^+) + \frac{\|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2}{2} \geq 0. \tag{3.4}$$

Therefore (3.3) becomes

$$\epsilon^{(j)} = \min_{\boldsymbol{\delta}:\boldsymbol{A}\boldsymbol{\delta}+\boldsymbol{b}\geq 0} \left\{\frac{1}{2}\boldsymbol{\delta}^T\boldsymbol{\delta}\right\} := P^{(j)}(\boldsymbol{\delta}), \tag{3.5}$$

where $\boldsymbol{A} \in \mathbb{R}^{n^+ \times d}$ is a matrix, $\boldsymbol{b} \in \mathbb{R}^{n^+}$ is a vector, the transpose of the ith row of $\boldsymbol{A}$ is $\boldsymbol{a}_i = (\boldsymbol{x}_j^- - \boldsymbol{x}_i^+)$, and the ith element of $\boldsymbol{b}$ is $b_i = (\|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2)/2$. Here for convenience we omit dependence of $\boldsymbol{A}$ and $\boldsymbol{b}$ on the index j. By solving the quadratic programming (QP) problem (3.5) for each $j \in [n^-]$, the final minimum adversarial perturbation norm is

$$\epsilon^* = \min_{j\in[n^-]} \sqrt{2\epsilon^{(j)}}. \tag{3.6}$$

It has been shown that convex quadratic programming problems can be solved in polynomial time [KTK80], so our formulation leads to a **polynomial-time algorithm** for finding the minimum adversarial perturbation norm ϵ^*.

Dual Quadratic Programming Problems

We also introduce the dual form of each QP, which is more efficient to solve in practice and will lead to a family of verification algorithms for adversarial robustness. The dual problem of (3.5) can be written as

$$\max_{\boldsymbol{\lambda} \geq 0} \left\{ -\frac{1}{2} \boldsymbol{\lambda}^T \boldsymbol{A} \boldsymbol{A}^T \boldsymbol{\lambda} - \boldsymbol{\lambda}^T \boldsymbol{b} \right\} := D^{(j)}(\boldsymbol{\lambda}), \tag{3.7}$$

where $\boldsymbol{\lambda} \in \mathbb{R}^{n^+}$ are the corresponding dual variables. The derivation is included in Appendix 3.1.6 for completeness. The primal-dual relationship connects primal and dual variables as $\boldsymbol{\delta} = \boldsymbol{A}^T \boldsymbol{\lambda}$. Based on weak duality, we have $D^{(j)}(\boldsymbol{\lambda}) \leq P^{(j)}(\boldsymbol{\delta})$ for any dual feasible solution $\boldsymbol{\lambda}$ and primal feasible solution $\boldsymbol{\delta}$. Furthermore, based on Slater's condition it can be shown that strong duality obtains, i.e., $D^{(j)}(\boldsymbol{\lambda}^*) = P^{(j)}(\boldsymbol{\delta}^*)$, where $\boldsymbol{\lambda}^*$ and $\boldsymbol{\delta}^*$ are optimal solutions for the primal and dual problems respectively, if $\boldsymbol{x}_j^- \neq \boldsymbol{x}_i^+$ holds for all $i \in [n^+]$.

[1] Based on strong duality, we have

$$\frac{1}{2}(\epsilon^*)^2 = \min_{j \in [n^-]} \{ P^{(j)}(\boldsymbol{\delta}^*) \} \tag{3.8}$$

$$= \min_{j \in [n^-]} \{ \max_{\boldsymbol{\lambda} \geq 0} D^{(j)}(\boldsymbol{\lambda}) \} \tag{3.9}$$

$$\geq \min_{j \in [n^-]} \{ D^{(j)}(\boldsymbol{\lambda}^{(j)}) \} \text{ with feasible } \boldsymbol{\lambda}^{(j)}, \tag{3.10}$$

so a set of feasible solutions $\{ \boldsymbol{\lambda}^{(j)} \}_{j \in [n^-]}$ leads to a lower bound of the minimum adversarial perturbation. In summary, we conclude the primal-dual relationship between 1-NN attack and verification:

- A primal feasible solution of $P^{(j)}$ is a successful attack and gives us an upper bound of ϵ^*. Therefore, one can solve a subset of QPs and select the minimum. Usually a $\boldsymbol{x}_j^-$ closer to $\boldsymbol{z}$ will lead to a smaller adversarial perturbation, so in practice we can sort $\boldsymbol{x}_j^-$ by the distance to $\boldsymbol{z}$, solve the subproblems one by one, and stop at any time. It

---

[1]If the precondition holds, then any point in a small ball around the center $\boldsymbol{\delta} = \boldsymbol{x}_j^- - \boldsymbol{z}$ will be a feasible solution, which implies strong duality by Slater's condition.

will give a valid adversarial perturbation. After solving all the subproblems, the result will reach the minimum, i.e., the minimum adversarial perturbation norm ϵ^*.

- A set of dual feasible solutions $\{\boldsymbol{\lambda}^{(j)}\}_{j\in[n^-]}$ will lead to a lower bound of ϵ^* according to (3.10). Thus any heuristic method for setting up a set of dual feasible solutions will give us a lower bound, which can be used for robustness verification. If all the dual problems are solved exactly, we will derive the tightest (maximum) lower bound, which is also ϵ^*.

Verification for 1-NN   Here we give an example of how to quickly set up dual variables to give a non-trivial lower bound of the minimum adversarial perturbation without solving any subproblem exactly. For a dual problem $D^{(j)}$, considering only having one variable $\lambda_i^{(j)}$ to be the optimization variable while fixing all the rest variables zero, then the optimal closed-form solution for this simplified dual QP problem will be

$$
\lambda_i^{(j)} = \max\left(0, -\frac{b_i}{\|\boldsymbol{a}_i\|^2}\right),
$$

$$
D^{(j)}([0,\ldots,0,\lambda_i^{(j)},0,\ldots,0]) = \frac{\max(-b_i,0)^2}{2\|\boldsymbol{a}_i\|^2}.
$$

(3.11)

Note that both $b_i = (\|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2)/2$ and $\boldsymbol{a}_i = \boldsymbol{x}_j^- - \boldsymbol{x}_i^+$ can be computed easily, thus according to (3.10) a guaranteed lower bound of ϵ^* can be computed:

$$
\min_{j\in[n^-]}\max_{i\in[n^+]}\frac{\max(\|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2, 0)}{2\|\boldsymbol{x}_j^- - \boldsymbol{x}_i^+\|}.
$$

(3.12)

This value has an interesting geometrical meaning. The inner value is the distance between $\boldsymbol{z}$ to the bisection between $\boldsymbol{x}_i^+$ and $\boldsymbol{x}_j^-$, which means if we want to perturb $\boldsymbol{z}$ to make it closer to $\boldsymbol{x}_j^-$ than $\boldsymbol{x}_i^+$, the perturbation must be larger than the inner value. Then, if we want to perturb $\boldsymbol{z}$ such that the nearest neighbor is $\boldsymbol{x}_j^-$, we need to by-pass all the bisections so we need to take the max operation among all the distances to bisections. Finally a lower bound of ϵ^* can be computed by taking minimum over all the $\boldsymbol{x}_j^-$.

In general, we can also get improved lower bounds by optimizing more coordinates rather than one variable for each subproblem.

Solving the QP Problems Efficiently

Now we discuss how to efficiently solve a series of QP problems $\{D^{(j)}\}_{j \in [n^-]}$ in practice. Although we can do this in polynomial time, in practice a naive algorithm is still too slow. Note that we have $O(n)$ quadratic problems and each has $O(n)$ dual variables, so roughly more than $O(n^3)$ time is required naively. In the following, we propose an efficient algorithm for this issue. First, we show that a greedy coordinate ascent algorithm can be efficiently applied to solve the dual QP problem (3.7) with time complexity much less than $O(n^2)$. The main idea is to exploit the sparsity of the solution — if $\boldsymbol{\lambda}^*$ is the dual optimal, then a nonzero λ_i^* means the primal constraint $\|\boldsymbol{z} + \boldsymbol{\delta} - \boldsymbol{x}_j^-\|^2 = \|\boldsymbol{z} + \boldsymbol{\delta} - \boldsymbol{x}_i^+\|^2$ is active, so the optimal $\boldsymbol{z} + \boldsymbol{\delta}$ will be on the bisecting hyperplane between $\boldsymbol{x}_i^+$ and $\boldsymbol{x}_j^-$. Therefore, if $\|\boldsymbol{\lambda}^*\|_0 = q$ then the optimal solution is the intersection of q bisecting hyperplanes, which means q is usually small. For instance, on the MNIST dataset when we test on 100 subproblems, the average value of $\|\boldsymbol{\lambda}^*\|_0$ is only 50.06 while the average number of constraints is 7,613. Sparsity of the optimal solution motivates the use of the greedy coordinate ascent algorithm. Starting from $\boldsymbol{\lambda} = \boldsymbol{0}$, we maintain the gradient vector with $\boldsymbol{g} = -\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\lambda} - \boldsymbol{b}$ and every time we pick the variable with the largest projected gradient

$$i^* = \arg\max_i |(\max(\boldsymbol{\lambda} + \boldsymbol{g}/\operatorname{diag}(\boldsymbol{A}\boldsymbol{A}^T), \boldsymbol{0}) - \boldsymbol{\lambda})_i| \tag{3.13}$$

and then update a single variable

$$\lambda_{i^*} \leftarrow \max(\lambda_{i^*} + g_{i^*}/\|\boldsymbol{a}_{i^*}\|^2, 0). \tag{3.14}$$

This is similar to the SMO method proposed for training kernel SVM [Pla98, CL11]; but due to no extra equality constraint, we only need to pick one variable at a time. Since there are only a few nonzero values in $\boldsymbol{\lambda}^*$, the algorithm usually converges much more quickly than standard quadratic programming solvers (see Figure 3.5 in the experiment section).

Second, we propose a screening rule to remove variables in each dual QP problem (3.7). There are only a few nonzero variables, and our screening rule will reduce the size of optimization variables *before solving the problem.* We introduce the following lemma:

Lemma 2. *For a specific quadratic problem $P^{(j)}(\boldsymbol{\delta})$, the optimal dual solution has $\lambda_i^* = 0$ if*

$$-(\|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2)/2 + \|\boldsymbol{x}_j^- - \boldsymbol{x}_i^+\|\|\boldsymbol{\delta}^*\| < 0, \tag{3.15}$$

where $\boldsymbol{\delta}^*$ is the optimal solution.

The proof is in Appendix 3.1.7. Note that checking (3.15) does not need to solve the QP problem. To conduct the screening rule, we need to have an estimation of $\|\boldsymbol{\delta}^*\|$ or its upper bound. A naive upper bound $\|\boldsymbol{\delta}^*\| \leq \|\boldsymbol{x}_j^- - \boldsymbol{z}\|$ can be used for running the screening rule.

With the methods mentioned above, each dual QP can be solved efficiently. However, there are $O(n)$ QPs in total, and solving all of them is still expensive. So *can we safely remove most of the irrelevant QPs?*

We can use the primal-dual relationship for removing most of the QP problems before solving them. Assume we have a primal solution $\bar{\boldsymbol{\delta}}$, then the minimum adversarial perturbation norm $\epsilon^* \leq \|\bar{\boldsymbol{\delta}}\|$, so every dual problem with $D^{(j)}(\boldsymbol{\lambda}) \geq \bar{\boldsymbol{\delta}}^T\bar{\boldsymbol{\delta}}/2$ for some $\boldsymbol{\lambda}$ can be removed. For a subproblem with respect to $\boldsymbol{x}_j^-$, based on (3.11) we know the subproblem can be removed if

$$\frac{1}{2}\bar{\boldsymbol{\delta}}^T\bar{\boldsymbol{\delta}} \leq \frac{\max(-b_i, 0)^2}{2\|\boldsymbol{a}_i\|^2} \tag{3.16}$$

for some i, thus we can use (3.16) to remove redundant subproblems. In practice, we sort the subproblems in ascending order of $\|\boldsymbol{z} - \boldsymbol{x}_j^-\|$ and iteratively run the screening rule after solving one more subproblem. As a result, most of the subproblems can be safely removed without need of solving them (see Figure 3.4), and hence we achieve significant speedup. Our overall algorithm is illustrated in Algorithm 5.

Algorithm 5 Computing minimum adversarial perturbation (QP-exact)

1: **Input:** Test instance $\boldsymbol{z}$, database $\mathbb{S}^+ = \{\boldsymbol{x}_1^+, \ldots, \boldsymbol{x}_{n^+}^+\}$ and $\mathbb{S}^- = \{\boldsymbol{x}_1^-, \ldots, \boldsymbol{x}_{n^-}^-\}$.

2: **Output:** Perturbation norm ϵ.

3: Initial $\epsilon = \infty$.

4: Sort subproblems $\{D^{(j)}\}_{j \in [n^-]}$ by ascending distances of $\|\boldsymbol{z} - \boldsymbol{x}_j^-\|$.

5: **for** each j (according to the sorted order) **do**

6:     **if** not screenable via (3.16) **then**

7:         Solve the subproblem via greedy coordinate ascent with screening rule (3.15).

8:         Update ϵ if we get a smaller value.

9:     **end if**

10: **end for**

3.1.3 Extending beyond 1-NN

We can extend our approach to K-NN models with $K > 1$. Remember for a given correctly classified instance $\boldsymbol{z}$, we partition the training dataset into two disjoint sets $\mathbb{S}^+ = \{\boldsymbol{x}_1^+, \ldots, \boldsymbol{x}_{n^+}^+\}$ and $\mathbb{S}^- = \{\boldsymbol{x}_1^-, \ldots, \boldsymbol{x}_{n^-}^-\}$ such that all instances in $\mathbb{S}^+$ have the same label with $\boldsymbol{z}$ and $\mathbb{S}^-$ contains the other instances. It is supposed that K is an odd number, $n^+ \geq (K-1)/2$ and $n^- \geq (K+1)/2$ hold to avoid trivial results. Taking the binary classification as an example, we can list all the possible combinations of $(\mathbb{I}, \mathbb{J})$ with $\mathbb{I} \subseteq [n^+]$, $\mathbb{J} \subseteq [n^-]$, $|\mathbb{I}| = (K-1)/2$, $|\mathbb{J}| = (K+1)/2$, and then solve the following QP problem to force $\boldsymbol{z} + \boldsymbol{\delta}$ to be closer to $\boldsymbol{x}_j^-$ for all $j \in [\mathbb{J}]$ than to all instances in $\mathbb{S}^+$ except $\boldsymbol{x}_i^+$ for all $i \in [\mathbb{I}]$:

$$\epsilon^{(\mathbb{I}, \mathbb{J})} = \min_{\boldsymbol{\delta}} \frac{1}{2} \boldsymbol{\delta}^T \boldsymbol{\delta}$$

$$\text{s.t. } \|\boldsymbol{z} + \boldsymbol{\delta} - \boldsymbol{x}_j^-\|^2 \leq \|\boldsymbol{z} + \boldsymbol{\delta} - \boldsymbol{x}_i^+\|^2, \qquad (3.17)$$

$$\forall i \in [n^+] - \mathbb{I}, \ \forall j \in \mathbb{J}.$$

Then, the norm of the minimum adversarial perturbation is $\epsilon^* = \min_{(\mathbb{I}, \mathbb{J})} \sqrt{2\epsilon^{(\mathbb{I}, \mathbb{J})}}$. There will be $O(nK)$ constraints and $O((n/k)^k)$ number of QPs in total. Therefore, in general it is not applicable to compute the minimum adversarial perturbation by iterating all QPs. However,

we can still obtain an upper and lower bound, corresponding to attack and verification.

Attack for K-NN For an upper bound (attack), we just need to heuristically choose a small number of $(\mathbb{I}, \mathbb{J})$ tuples and only solve these QPs. The overall algorithm, QP-greedy, is illustrated in Algorithm 6. The basic idea of QP-greedy is to iteratively choose $\mathbb{I}$ and $\mathbb{J}$ and then compute $\epsilon^{(\mathbb{I},\mathbb{J})}$ by solving (3.17) until $\epsilon^{(\mathbb{I},\mathbb{J})}$ is a feasible solution. Specifically, $\mathbb{I}$ is the indices of instances in $\mathbb{S}^+$ that are nearest to the test instance $\boldsymbol{z}$. At each iteration, a neighbor of $\boldsymbol{z}$ from $\mathbb{S}^-$ is selected, denoted as $\boldsymbol{x}^-$. $\mathbb{J}$ is the indices of instances in $\mathbb{S}^-$ that are nearest to $\boldsymbol{x}^-$ including $\boldsymbol{x}^-$ itself. Then, QP-greedy tries to solve (3.17) to get an optimal $\epsilon^{(\mathbb{I},\mathbb{J})}$.

Verification for K-NN For a lower bound (verification), by extending (3.12) to the $K > 1$ case, we have the following lemma:

Lemma 3. *For an K-NN model with any odd K, we have a lower bound for the minimum adversarial perturbation as*

$$\epsilon^* \geq k\text{th} \min s_{j\in[n^-]} \left(k\text{th} \max s_{i\in[n^+]} \sqrt{2\epsilon^{(i,j)}}\right), \tag{3.18}$$

where $s = (K+1)/2$ is a positive integer, "$k\text{th} \min s$" and "$k\text{th} \max s$" select the s-th minimum value and the s-th maximum value respectively, and

$$\epsilon^{(i,j)} = \frac{\max(\|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2, 0)^2}{8\|\boldsymbol{x}_j^- - \boldsymbol{x}_i^+\|^2}. \tag{3.19}$$

The proof is in Appendix 3.1.8. Note that the lower bound (3.18) can be computed efficiently. We need $O(n^2 d)$ time to compute all $\epsilon^{(i,j)}$ and then the top-s selection can be done in linear time, so the overall complexity is just $O(n^2 d)$ which is independent to K.

Attack and verification in the multi-class and $K > 1$ case Although the deviation above is for the binary case, we can easily extend it to the multi-class case by simply

---

Algorithm 6 QP-greedy

1: **Input:** Target instance $\boldsymbol{z}$, neighbor parameter K, minimum number of iterations t,

database $\mathbb{S}^+ = \{\boldsymbol{x}_1^+, \ldots, \boldsymbol{x}_{n^+}^+\}$ and $\mathbb{S}^- = \{\boldsymbol{x}_1^-, \ldots, \boldsymbol{x}_{n^-}^-\}$.

2: **Output:** Perturbation norm ϵ.

3: Initialize $\epsilon = \infty$

4: Select $\mathbb{I}$ with $\|\mathbb{I}\| = (K-1)/2$ and it is the indices of instances in $\mathbb{S}^+$ that are nearest to

the test instance $\boldsymbol{z}$

5: Sort $\mathbb{S}^-$ by ascending distances of $\|\boldsymbol{z} - \boldsymbol{x}_j^-\|$

6: **for** each j (according to the sorted order) **do**

7:     Select $\mathbb{J}$ with $\|\mathbb{J}\| = (K+1)/2$ and it is the indices of instances in $\mathbb{S}^-$ that are nearest

to $\boldsymbol{x}_j^-$ including $\boldsymbol{x}_j^-$ itself

8:     Compute $\epsilon^{(\mathbb{I}, \mathbb{J})}$ by solving (3.17)

9:     **if** $\epsilon^{(\mathbb{I}, \mathbb{J})}$ is a feasible solution **then**

10:         Update ϵ if we get a smaller value

11:         **if** iteration number is greater than t **then**

12:             **break**

13:         **end if**

14:     **end if**

15: **end for**

---

partitioning the training dataset in a different way. For an upper bound (attack), the training dataset needs to be partitioned such that all instances in $\mathbb{S}^-$ are with a same label but different from $\boldsymbol{z}$, and $\mathbb{S}^+$ contains the others including the label of $\boldsymbol{z}$. Then we just need to choose a small number of $(\mathbb{I}, \mathbb{J})$ tuples and solve these QPs (see (3.17)). For a lower bound (verification), the training dataset needs to be partitioned as usual such that all instances in $\mathbb{S}^+$ have the same label with $\boldsymbol{z}$ and $\mathbb{S}^-$ contains the other instances. Then (3.18) will be a lower bound.

3.1.4 Extending to ℓ_∞ and ℓ_1 norms

Sometimes people are interested in finding the minimum ℓ_∞ or ℓ_1 norm adversarial perturbation (replacing the ℓ_2 norm in (3.1)). Those can be solved similarly using our framework but will require linear programming instead of quadratic programming. The minimum ℓ_∞-norm adversarial perturbation can be formulated as

$$\epsilon^{(j)} = \min_{\boldsymbol{\delta}} \; v$$
$$\text{s.t. } \boldsymbol{A\delta} + \boldsymbol{b} \geq 0, \; v \geq \delta_i \geq -v, \; \forall i \in [d]. \tag{3.20}$$

The minimum ℓ_1-norm adversarial perturbation can be formulated as

$$\epsilon^{(j)} = \min_{\boldsymbol{\delta}} \; \mathbf{1}^\top \boldsymbol{v}$$
$$\text{s.t. } \boldsymbol{A\delta} + \boldsymbol{b} \geq 0, \; v_i \geq \delta_i \geq -v_i, \; \forall i \in [d]. \tag{3.21}$$

This can also be solved efficiently by linear programming solvers and the primal-dual relationship also holds.

3.1.5 Experiments

We show main results in Section 3.1.5, and analyze efficiency of our algorithm in Section 3.1.5.

Main Results

We show that our formulation leads to better attack and verification algorithms. Our primal-dual QP framework leads to the following proposed algorithms for exact computation, verification, and attack:

- QP-exact: computes the *exact* minimum adversarial perturbation for 1-NN via Algorithm 5.

- QP-verifiy: computes lower bounds (verification) for 1-NN via (3.12) and for K-NN via (3.18).

Table 3.1: Average ℓ_2 norms of adversarial perturbations of 100 random-sampled correctly-classified test instances for 1-NN. The attack success rates of Mean on Letter and Pendigits are 86% and 99% respectively, and other attack algorithms (upper bounds) on these instances have success rates 100%.

		Letter	Pendigits	USPS	Satimage	MNIST	Fashion-MNIST
Exact value	QP-exact	0.100	0.278	0.961	0.200	1.530	1.077
Lower bound	QP-verify	0.098	0.264	0.927	0.183	1.451	1.029
Upper bound	QP-top-1	0.104	0.284	0.973	0.210	1.560	1.090
	QP-top-10	0.100	0.279	0.961	0.200	1.530	1.077
	Naive-1	0.116	0.318	1.144	0.247	1.923	1.410
	Naive-10	0.110	0.306	1.114	0.235	1.848	1.377
	Mean	0.264	0.588	2.263	0.507	4.560	4.066
	Substitute	0.267	0.678	1.211	0.535	1.791	1.267

- QP-top-1 and QP-top-10: compute upper bounds (attack) for 1-NN via Algorithm 5 but only iterate over the top-1 and top-10 QP problems respectively.

- QP-greedy: computes an upper bound for K-NN by greedily selecting QP problems as Algorithm 6.

Note that there is no existing algorithm for computing the exact minimum adversarial perturbation and no existing verification method for K-NN that can compute a lower bound. Therefore we are only able to compare with the following attack methods:

- Naive-1 and Naive-10 [ABB17, SW19]: compute upper bounds for 1-NN by moving towards a nearby training instance from $\mathbb{S}^-$. Naive-10 repeats the process for 10 times

Table 3.2: Average ℓ_2 norms of adversarial perturbations of 100 random-sampled correctly-classified test instances for 15-NN. ASR stands for Attack Success Rate, and only makes sense for attack algorithms.

		Letter		Pendigits		USPS		Satimage		MNIST		Fashion-MNIST	
		ASR	Avg ℓ_2	ASR	Avg ℓ_2	ASR	Avg ℓ_2	ASR	Avg ℓ_2	ASR	Avg ℓ_2	ASR	Avg ℓ_2
Lower bound	QP-verify	–	0.082	–	0.287	–	0.929	–	0.192	–	1.395	–	1.051
	QP-greedy	1.00	0.156	1.00	0.396	1.00	1.706	1.00	0.339	1.00	2.842	1.00	2.546
	Naive-1	1.00	0.152	0.98	0.415	0.90	1.885	0.56	0.389	0.96	3.451	0.59	2.462
Upper bound	Naive-10	1.00	0.127	1.00	0.366	1.00	1.615	0.93	0.337	1.00	2.862	0.89	2.394
	Mean	0.90	0.265	0.99	0.597	1.00	2.216	1.00	0.487	1.00	4.560	0.85	3.738

and chooses the best perturbation. For $K > 1$, they move towards the mean of a nearby cluster from $\mathbb{S}^-$ with the size $(K+1)/2$ and all the instances in the cluster belong to the same class.

- Mean [SW19]: computes an upper bound for K-NN by moving towards a class mean. The class with the smallest class-mean distance to the test instance is chosen to move towards.

- Substitute [PMG16c]: computes an upper bound for 1-NN by attacking a differentiable substitute model via normalized gradient ascent.

We evaluate our proposed algorithms on six public datasets including four small or medium sized datasets [CL11], on which nearest neighbor classifiers perform well, and two image datasets, MNIST [LBB98] and Fashion-MNIST [XRV17], which are commonly used for evaluating adversarial robustness of neural networks.

Perturbations for 1-NN Table 3.1 shows average ℓ_2 norms of adversarial perturbations of 100 random-sampled correctly-classified test instances. The results suggest that
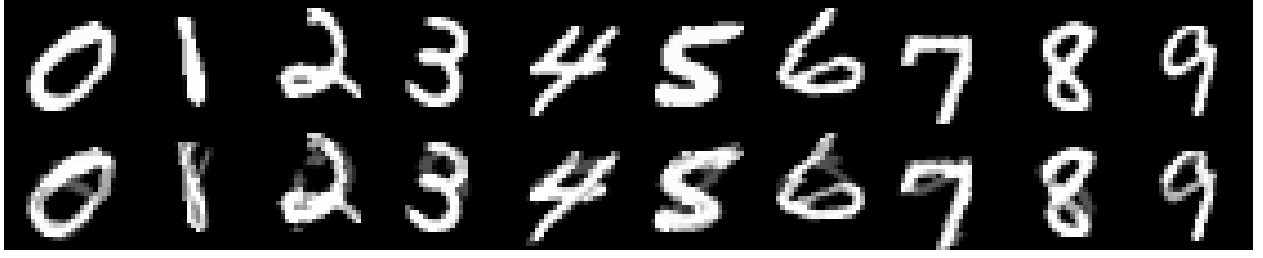
(a) MNIST
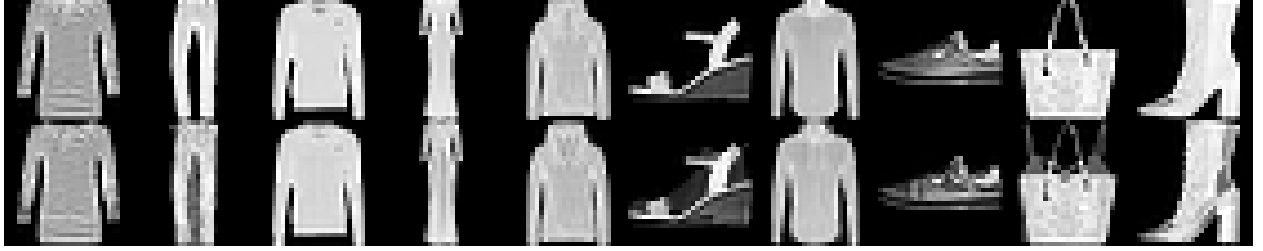
(b) Fashion-MNIST

Figure 3.2: Adversarial examples crapted by QP-exact for MNIST and Fashion-MNIST. The first row for every subfigure is the original images and the second row is the corresponding adversarial images.

- QP-verify can provide a lower bound of the minimum adversarial perturbation and the lower bound is quite tight.

- QP-top-1 and QP-top-10 achieve better attack performance than previous attack algorithms.

- QP-exact can successfully compute the minimum adversarial perturbation.

Some adversarial examples crafted by QP-exact for MNIST and Fashion-MNIST are visualized in Figure 3.2.

Perturbations for K-NN with $K > 1$ Table 3.6 shows attack success rates of attack algorithms, and average ℓ_2 norms of adversarial perturbations of verification and attack algorithms for K-NN ($K = 15$). In general, it is hard to say that K-NN is definitely more

robust than 1-NN. It is due to the fact that the exact perturbation of 1-NN is often between the lower bound and the upper bound of K-NN. However, our experiments show that the lower bound of K-NN is close to the exact value of 1-NN, while the upper bound of K-NN is much larger than the exact value of 1-NN, which leads to the conjecture that K-NN tends to be more robust than 1-NN.

Comparing certified robust errors under ℓ_2-norm with neural networks We compare *certified robust errors*, defined as the fraction with lower bounds less than the given threshold (if an instance is wrongly classified, the lower bound is 0), of 1-NN, a simple convolutional network (ConvNet), and a strong ℓ_2 certified defense network (RandSmooth) [CRK19] in Figure 3.3. The results show that 1-NN can achieve better certified robust errors than neural networks on these two datasets. This is partially because the proposed verification algorithm provides very tight certified regions for NN classifiers. Note that we are not claiming that K-NN is more robust than neural networks on more complex datasets such as CIFAR and ImageNet. Instead, we show that, due to a more accurate verification method, NN classifiers can achieve better certified robust errors on some simpler tasks.

Efficiency of Our Algorithm

We show runtime of the proposed algorithms for 1-NN on MNIST and Fashion-MNIST in Table 3.3. The results show that efficiency of our proposed algorithms is comparable with other attack algorithms. In particular, **QP-exact can compute the exact minimum adversarial perturbation while having similar runtime compared with other attack algorithms.** Note that we have three main contributions in speeding up QP-exact: sorting, screening rules for reducing the number of QPs, and the greedy coordinate ascent solver for solving each QP.
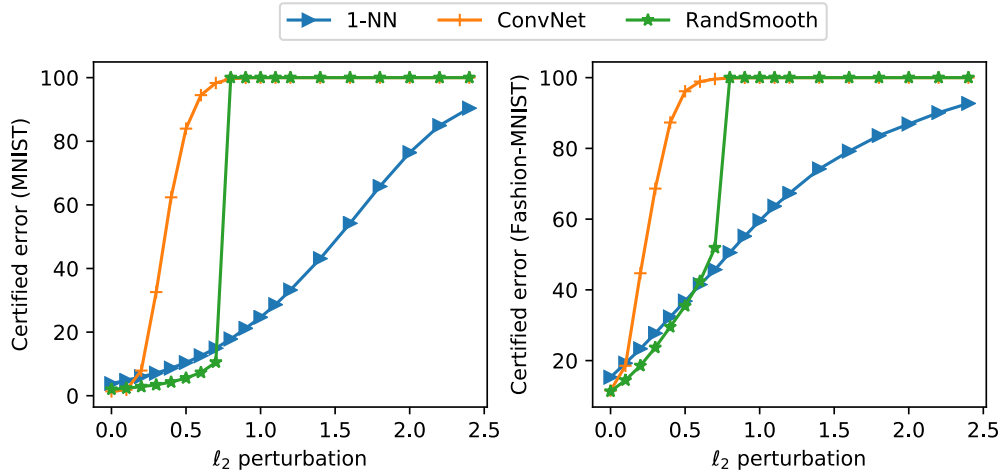
Figure 3.3: Comparing certified (robust) errors of 1-NN, ConvNet (a simple convolutional network), and RandSmooth (a defense neural network via random smoothing proposed by [CRK19]). For RandSmooth, we choose the noise standard deviation $\sigma = 0.2$. 1-NN is more robust on these two datasets.

Table 3.3: Average real runtime (s) when performing on 100 randomly-sampled correctly-classified test instances.

	QP-exact	QP-verify	QP-top-1	QP-top-10	Naive-1	Naive-10	Mean	Substitute
MNIST	4.862	3.931	1.010	1.254	1.719	12.803	5.090	8.762
Fashion-MNIST	4.828	3.807	0.924	1.064	1.765	13.448	5.156	4.756

Sorting and screening In the MNIST case, for every test instance, we have to solve about $54,000$ (the size of $\mathbb{S}^-$) QP problems without screening. When running on 100 correctly-classified test instances, the average number of QPs left to solve for QP-exact *with* sorting is 2.53, while the average number of QPs left to solve for QP-exact *without* sorting is 25.49. Therefore, sorting and screening significantly reduces the number of QPs required to solve. The screening parameter n_{scr}, number of *i*s chosen for each j in (3.16), is important for controlling the trade-off between the number of screened subproblems and the screening overheads. We plot the trade-off on MNIST in Figure 3.4. This shows a very small n_{scr}
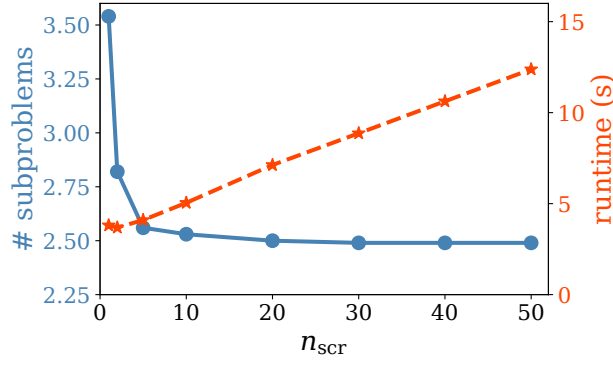
Figure 3.4: Screening trade-off. Nearly all subproblems are removed without need of solving them.
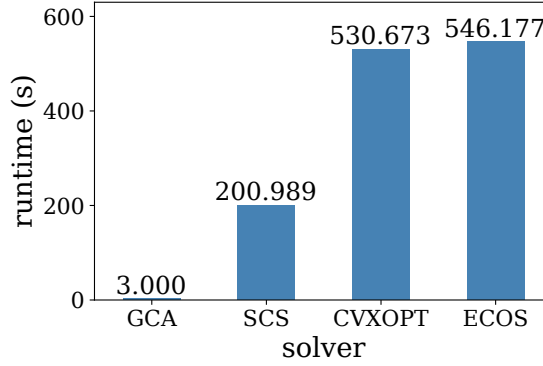
Figure 3.5: Runtime of QP solvers. Average real runtime (s) when performing 100 randomly-sampled correctly-classified test instances on 1/10 MNIST.

suffices for screening, and we choose 8 for our experiments where 2.53 lies.

Greedy coordinate ascent Due to sparsity of the optimal solution for each QP problem, the greedy coordinate ascent solver is much more efficient than other standard QP solvers. To verify this, we compare greedy coordinate ascent with SCS [OCP16], CVXOPT and ECOS [DCB13] for solving these QP problems. We fix everything the same (with the same screening rule and sorting technique) while only change the QP solver. Since it is difficult for standard QP solvers to deal with high dimensional problems, 1/10 training samples of

84

MNIST are used. The average runtime of 100 correctly-classified test instances is presented in Figure 3.5. Greedy coordinate ascent is faster than other solvers by more than 60 times for computing K-NN robustness.

3.1.6 Derivation of the dual form

Consider the primal problem in (3.5). The Lagrangian can be written as

$$L(\boldsymbol{\delta}, \boldsymbol{\lambda}) = \frac{1}{2}\boldsymbol{\delta}^T\boldsymbol{\delta} - \boldsymbol{\lambda}^T(\boldsymbol{A}\boldsymbol{\delta} + \boldsymbol{b}). \tag{3.22}$$

The dual problem is then

$$\max_{\boldsymbol{\lambda} \geq 0} \ \min_{\boldsymbol{\delta}} \ L(\boldsymbol{\delta}, \boldsymbol{\lambda}). \tag{3.23}$$

Taking derivative of Lagrangian we get

$$\frac{\partial}{\partial \boldsymbol{\delta}}L = \boldsymbol{\delta} - \boldsymbol{A}^T\boldsymbol{\lambda} = 0, \tag{3.24}$$

which gives us the primal-dual relationship $\boldsymbol{\delta} = A^T\boldsymbol{\lambda}$. Substitute this back to the dual problem we get

$$\max_{\boldsymbol{\lambda} \geq 0} \ -\frac{1}{2}\boldsymbol{\lambda}^T\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\lambda} - \boldsymbol{\lambda}^T\boldsymbol{b}. \tag{3.25}$$

3.1.7 Proof of Lemma 2

Proof. By definition we have $\nabla D^{(j)}(\boldsymbol{\lambda}) = -\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\lambda} - \boldsymbol{b}$. By the primal-dual relationship, we have

$$\nabla D^{(j)}(\boldsymbol{\lambda}^*) = -\boldsymbol{A}\boldsymbol{A}^T\boldsymbol{\lambda}^* - \boldsymbol{b} \tag{3.26}$$

$$= -\boldsymbol{A}\boldsymbol{\delta}^* - \boldsymbol{b}. \tag{3.27}$$

Consider the i-th element of $D^{(j)}(\boldsymbol{\lambda})$, then we have

$$\nabla D_i^{(j)}(\boldsymbol{\lambda}^*)$$

$$= -\boldsymbol{a}_i^T \boldsymbol{\delta}^* - b_i \tag{3.28}$$

$$= (\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-)^T \boldsymbol{\delta}^* - \frac{\|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2}{2} \tag{3.29}$$

$$\leq \|\boldsymbol{x}_i^+ - \boldsymbol{x}_j^-\| \|\boldsymbol{\delta}^*\| - \frac{\|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2}{2}. \tag{3.30}$$

Therefore, when (3.15) holds, by KKT conditions of the dual problem we know $\lambda_i^* = 0$. $\square$

3.1.8 Proof of Lemma 3

Proof. The problem (3.17) can be formulated as

$$\epsilon^{(\mathbb{I},\mathbb{J})} \geq \max_{i \in [n^+] - \mathbb{I}, j \in \mathbb{J}} \epsilon^{(i,j)}, \tag{3.31}$$

where

$$\epsilon^{(i,j)} = \left\{ \min_{\boldsymbol{\delta}} \frac{1}{2} \boldsymbol{\delta}^T \boldsymbol{\delta} \ \text{ s.t. } \|\boldsymbol{z} + \boldsymbol{\delta} - \boldsymbol{x}_j^-\|^2 \leq \|\boldsymbol{z} + \boldsymbol{\delta} - \boldsymbol{x}_i^+\|^2 \right\}. \tag{3.32}$$

Based on (3.11), we have

$$\epsilon^{(i,j)} = \frac{\max(\|\boldsymbol{z} - \boldsymbol{x}_j^-\|^2 - \|\boldsymbol{z} - \boldsymbol{x}_i^+\|^2, 0)^2}{8\|\boldsymbol{x}_j^- - \boldsymbol{x}_i^+\|^2}. \tag{3.33}$$

Therefore, we have a lower bound for the minimum adversarial perturbation,

$$\epsilon^* = \min_{(\mathbb{I},\mathbb{J})} \sqrt{2\epsilon^{(\mathbb{I},\mathbb{J})}} \geq \min_{(\mathbb{I},\mathbb{J})} \max_{\forall i \in [n^+] - \mathbb{I}, \forall j \in \mathbb{J}} \sqrt{2\epsilon^{(i,j)}} \tag{3.34}$$

$$= \min_{(\mathbb{I},\mathbb{J})} \max_{\forall j \in \mathbb{J}} \max_{\forall i \in [n^+] - \mathbb{I}} \sqrt{2\epsilon^{(i,j)}} \tag{3.35}$$

$$\geq k\text{th} \min s_{j \in [n^-]} \left(k\text{th} \max s_{i \in [n^+]} \sqrt{2\epsilon^{(i,j)}} \right). \tag{3.36}$$

$\square$

3.2 Provably Robust Metric Learning

Metric learning has been an important family of machine learning algorithms and has achieved successes on several problems, including computer vision [KJG09, GVS09, HBW07], text analysis [Leb06], meta learning [VBL16, SSZ17] and others [SWW08, XC06, YZS16]. Given a set of training samples, metric learning aims to learn a good distance measurement such that items in the same class are closer to each other in the learned metric space, which is crucial for classification and similarity search. Since this objective is directly related to the assumption of nearest neighbor classifiers, most of the metric learning algorithms can be naturally and successfully combined with K-Nearest Neighbor (K-NN) classifiers.

Adversarial robustness of machine learning algorithms has been studied extensively in recent years due to the need of robustness guarantees in real world systems. It has been demonstrated that neural networks can be easily attacked by adversarial perturbations in the input space [SZS14, GSS15b, BR18], and such perturbations can be computed efficiently in both white-box [CW17, MMS18b] and black-box settings [CZS17, IEM19, CSC20, WZY20]. To tackle this issue, many defense algorithms have been proposed to improve the robustness of neural networks [KGB17b, MMS18b]. Although these algorithms can successfully defend from standard attacks, it has been shown that many of them are vulnerable under stronger attacks when the attacker knows the defense mechanisms [CW17]. Therefore, recent research in adversarial defense of neural networks has shifted to the concept of "certified defense", where the defender needs to provide a certification that no adversarial examples exist within a certain input region [WK18, CRK19, ZCX20].

In this paper, we consider the problem of learning a metric that is robust against adversarial input perturbations. It has been shown that nearest neighbor classifiers are not as robust as expected [PMG16a, WLY19, SW20], where a small and human imperceptible perturbation in the input space can fool a K-NN classifier, thus it is natural to investigate how to obtain a metric that improves the adversarial robustness. Despite being an important and interesting

research problem to tackle, to the best of our knowledge it has not been studied in the literature. There are several caveats that make this a hard problem: 1) attack and defense algorithms for neural networks often rely on the smoothness of the corresponding functions, while K-NN is a discrete step function where the gradient does not exist. 2) Even evaluating the robustness of K-NN with the Euclidean distance is harder than neural networks — attack and verification for K-NN are nontrivial and time consuming [WLY19]. Furthermore, none of the existing work have considered general Mahalanobis distances. 3) Existing algorithms for evaluating the robustness of K-NN, including attack [YRW20] and verification [WLY19], are often non-differentiable, while training a robust metric will require a differentiable measurement of robustness.

To develop a provably robust metric learning algorithm, we formulate an objective function to learn a Mahalanobis distance, parameterized by a positive semi-definite matrix $\boldsymbol{M}$, that maximizes the minimal adversarial perturbation on each sample. However, computing the minimal adversarial perturbation is intractable for K-NN, so to make the problem solvable, we propose an efficient formulation for lower-bounding the minimal adversarial perturbation, and this lower bound can be represented as an explicit function of $\boldsymbol{M}$ to enable the gradient computation. We further develop several tricks to improve the efficiency of the overall procedure. Similar to certified defense algorithms in neural networks, the proposed algorithm can provide a certified robustness improvement on the resulting K-NN model with the learned metric. Decision boundaries of 1-NN with different Mahalanobis distances for a toy dataset (with only four orange triangles and three blue squares in a two-dimensional space) are visualized in Figure 3.6. It can be observed that the proposed Adversarial Robust Metric Learning (ARML) method can obtain a more "robust" metric on this example.

We conduct extensive experiments on six real world datasets and show that the proposed algorithm can improve both certified robust errors and the empirical robust errors (errors under adversarial attacks) over existing metric learning algorithms.

(a) Euclidean	(b) NCA [GHR04]	(c) ARML (Ours)
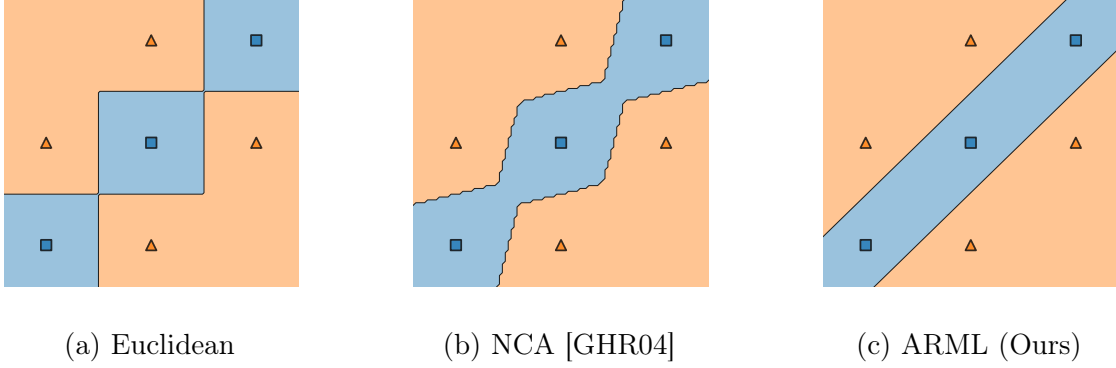
Figure 3.6: Decision boundaries of 1-NN with different Mahalanobis distances.

3.2.1 Background

Metric learning for nearest neighbor classifiers A nearest-neighbor classifier based on a Mahalanobis distance could be characterized by a training dataset and a positive semi-definite matrix. Let $\mathbb{X} = \mathbb{R}^D$ be the instance space, $\mathbb{Y} = [C]$ the label space where C is the number of classes. $\mathbb{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$ is the training set with $(\boldsymbol{x}_i, y_i) \in \mathbb{X} \times \mathbb{Y}$ for every $i \in [N]$. $\boldsymbol{M} \in \mathbb{R}^{D \times D}$ is a positive semi-definite matrix. The Mahalanobis distance for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{X}$ is defined as

$$d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^\top \boldsymbol{M} (\boldsymbol{x} - \boldsymbol{x}'), \tag{3.37}$$

and a Mahalanobis K-NN classifier $f : \mathbb{X} \to \mathbb{Y}$ will find the K nearest neighbors of the test instance in $\mathbb{S}$ based on the Mahalanobis distance, and then predicts the label based on majority voting of these neighbors.

Many metric learning approaches aim to learn a good Mahalanobis distance $\boldsymbol{M}$ based on training data [GHR04, DKJ07, WS09, JKD10, Sug07]. However, none of these previous methods are trying to find a metric that is robust to small input perturbations.

Adversarial robustness and minimal adversarial perturbation There are two important concepts in adversarial robustness: adversarial attack and adversarial verification

(or robustness verification). Adversarial attack aims to find a perturbation to change the prediction, and adversarial verification aims to find a radius within which no perturbation could change the prediction. Both of them can be reduced to the problem of finding the minimal adversarial perturbation. For a classifier f on an instance $(\boldsymbol{x}, y)$, the *minimal adversarial perturbation* can be defined as

$$\arg\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\| \quad \text{s.t.} \quad f(\boldsymbol{x} + \boldsymbol{\delta}) \neq y, \tag{3.38}$$

which is the smallest perturbation that could lead to "misclassification". Note that if $(\boldsymbol{x}, y)$ is not correctly classified, the minimal adversarial perturbation is $\boldsymbol{0}$, i.e., the zero vector. Let $\boldsymbol{\delta}^*(\boldsymbol{x}, y)$ denote the optimal solution and $\epsilon^*(\boldsymbol{x}, y) = \|\boldsymbol{\delta}^*(\boldsymbol{x}, y)\|$ the optimal value. Obviously, $\boldsymbol{\delta}^*(\boldsymbol{x}, y)$ is also the solution of the optimal adversarial attack, and $\epsilon^*(\boldsymbol{x}, y)$ is the solution of the optimal adversarial verification. For neural networks, it is often NP-complete to solve (3.38) exactly [KBD17], so many efficient algorithms have been proposed for attack [GSS15b, CW17, BRB18, CSC20] and verification [WK18, WZC18a, MGV18], corresponding to computing upper and lower bounds of the minimal adversarially perturbation respectively. However, these methods do not work for discrete models such as nearest neighbor classifiers.

In this paper our algorithm will be based on a novel derivation of a lower bound of the minimal adversarial perturbation for Mahalanobis K-NN classifiers. To the best of our knowledge, there has been no previous work tackling this problem. Since the Mahalanobis K-NN classifier is parameterized by a positive semi-definite matrix $\boldsymbol{M}$ and the training set $\mathbb{S}$, we further let the optimal solution $\boldsymbol{\delta}^*_{\mathbb{S}}(\boldsymbol{x}, y; \boldsymbol{M})$ and the optimal value $\epsilon^*_{\mathbb{S}}(\boldsymbol{x}, y; \boldsymbol{M})$ explicitly indicate their dependence on $\boldsymbol{M}$ and $\mathbb{S}$. In this paper we will consider ℓ_2 norm in (3.38) for simplicity.

Certified and empirical robust errors   Let $\underline{\epsilon}^*(\boldsymbol{x}, y)$ be a lower bound of the norm of the minimal adversarial perturbation $\epsilon^*(\boldsymbol{x}, y)$, possibly computed by a robustness verification algorithm. For a distribution $\mathcal{D}$ over $\mathbb{X} \times \mathbb{Y}$, the **certified robust error** with respect to the

given radius $\epsilon \geq 0$ is defined as the probability that $\underline{\epsilon}^*(\boldsymbol{x}, y)$ is not greater than ϵ, namely

$$\mathrm{cre}(\epsilon) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[\mathbf{1}\{\underline{\epsilon}^*(\boldsymbol{x}, y) \leq \epsilon\}]. \tag{3.39}$$

Note that in the case with $\underline{\epsilon}^*(\boldsymbol{x}, y) = \epsilon^*(\boldsymbol{x}, y)$, the certified robust error at $\epsilon = 0$ is reduced to the *clean error* (the normal classification error). In this paper we will investigate how to compute the certified robust error for Mahalanobis K-NN classifiers.

On the other hand, adversarial attack algorithms are trying to find a feasible solution of (3.38), denoted as $\hat{\boldsymbol{\delta}}(\boldsymbol{x}, y)$, which will give an upper bound, i.e., $\|\hat{\boldsymbol{\delta}}(\boldsymbol{x}, y)\| \geq \epsilon^*(\boldsymbol{x}, y)$. Based on the upper bound, we can measure the **empirical robust error** of a model by

$$\mathrm{ere}(\epsilon) = \mathbb{E}_{(\boldsymbol{x},y)\sim\mathcal{D}}[\mathbf{1}\{\|\hat{\boldsymbol{\delta}}(\boldsymbol{x}, y)\| \leq \epsilon\}]. \tag{3.40}$$

Since $\hat{\boldsymbol{\delta}}(\boldsymbol{x}, y)$ is computed by an attack method, the empirical robust error is also called the *attack error* or the *attack success rate*. A family of decision-based attack methods, which view the victim model as a black-box, can be used to attack Mahalanobis K-NN classifiers [BRB18, CLC19, CSC20].

3.2.2 Adversarially robust metric learning

The objective of *adversarially robust metric learning* (ARML) is to learn the matrix $\boldsymbol{M}$ via the training data $\mathbb{S}$ such that the resulting Mahalanobis K-NN classifier has small certified and empirical robust errors.

Basic formulation

The goal is to learn a positive semi-definite matrix $\boldsymbol{M}$ to minimize the certified robust training error. Since the certified robust error defined in (3.39) is non-smooth, we replace the indicator function by a loss function. The resulting objective can be formulated as

$$\min_{\boldsymbol{G}\in\mathbb{R}^{D\times D}} \frac{1}{N}\sum_{i=1}^{N} \ell\left(\epsilon^*_{\mathbb{S}-\{(\boldsymbol{x}_i,y_i)\}}(\boldsymbol{x}_i, y_i; \boldsymbol{M})\right) \quad \text{s.t. } \boldsymbol{M} = \boldsymbol{G}^\top\boldsymbol{G}, \tag{3.41}$$

where $\ell : \mathbb{R} \to \mathbb{R}$ is a monotonically non-increasing function, e.g., the hinge loss $[1 - \epsilon]_+$, exponential loss $\exp(-\epsilon)$, logistic loss $\log(1 + \exp(-\epsilon))$, or "negative" loss $-\epsilon$. We also employ the matrix $\boldsymbol{G}$ to enforce $\boldsymbol{M}$ to be positive semi-definite, and it is possible to derive a low-rank $\boldsymbol{M}$ by constraining the shape of $\boldsymbol{G}$. Note that the minimal adversarial perturbation is defined on the training set excluding $(\boldsymbol{x}_i, y_i)$ itself, since otherwise a 1-nearest neighbor classifier with any distance measurement will have 100% accuracy. In this way, we minimize the "leave-one-out" certified robust error. The remaining problem is how to exactly compute or approximate $\epsilon_\mathbb{S}^*(\boldsymbol{x}, y; \boldsymbol{M})$ in our training objective.

Bounding minimal adversarial perturbation for Mahalanobis K-NN

For convenience, suppose K is an odd number and denote $k = (K+1)/2$. In the binary classification case for simplicity, i.e., $C = 2$, the computation of $\epsilon_\mathbb{S}^*(\boldsymbol{x}_\text{test}, y_\text{test}; \boldsymbol{M})$ for Mahalanobis K-NN could be formulated as

$$
\min_{\substack{\mathbb{J} \subseteq \{j : y_j \neq y_\text{test}\}, |\mathbb{J}| = k \\ \mathbb{I} \subseteq \{i : y_i = y_\text{test}\}, |\mathbb{I}| = k-1}} \min_{\boldsymbol{\delta}_{\mathbb{I}, \mathbb{J}}} \ \|\boldsymbol{\delta}_{\mathbb{I}, \mathbb{J}}\|
$$
$$
\text{s.t.} \ \ d_{\boldsymbol{M}}(\boldsymbol{x}_\text{test} + \boldsymbol{\delta}_{\mathbb{I}, \mathbb{J}}, \boldsymbol{x}_j) \leq d_{\boldsymbol{M}}(\boldsymbol{x}_\text{test} + \boldsymbol{\delta}_{\mathbb{I}, \mathbb{J}}, \boldsymbol{x}_i),
$$
$$
\forall j \in \mathbb{J}, \ \forall i \in \{i : y_i = y_\text{test}\} - \mathbb{I}. \tag{3.42}
$$

This minimization formulation enumerates all the K-size nearest neighbor set containing at most $k-1$ instances in the same class with the test instance, computes the minimal perturbation resulting in each K-nearest neighbor set, and takes the minimum of them.

Obviously, solving (3.42) exactly (enumerating all $(\mathbb{I}, \mathbb{J})$ pairs) has time complexity growing exponentially with K, and furthermore, a numerical solution cannot be incorporated into the training objective (3.41) since we need to write ϵ^* as a function of $\boldsymbol{M}$ for back-propagation. To address these issues, we resort to a lower bound of the optimal value of (3.42) rather than solving it exactly.

First, we consider a simple triplet problem: given vectors $\boldsymbol{x}^+, \boldsymbol{x}^-, \boldsymbol{x} \in \mathbb{R}^D$ and a positive semi-definite matrix $\boldsymbol{M} \in \mathbb{R}^{D \times D}$, find the minimum perturbation $\boldsymbol{\delta} \in \mathbb{R}^D$ on $\boldsymbol{x}$ such that

$d_{\boldsymbol{M}}(\boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{x}^-) \leq d_{\boldsymbol{M}}(\boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{x}^+)$ holds. It could be formulated as the following optimization problem

$$\min_{\boldsymbol{\delta}} \ \|\boldsymbol{\delta}\| \ \text{ s.t. } d_{\boldsymbol{M}}(\boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{x}^-) \leq d_{\boldsymbol{M}}(\boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{x}^+). \tag{3.43}$$

Note that the constraint in (3.43) can be written as a linear form, so this is a convex quadratic programming problem with a linear constraint. We show that the optimal value of (3.43) can be expressed in closed form:

$$\frac{\left[d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}^-) - d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}^+)\right]_+}{2\sqrt{(\boldsymbol{x}^+ - \boldsymbol{x}^-)^\top \boldsymbol{M}^\top \boldsymbol{M}(\boldsymbol{x}^+ - \boldsymbol{x}^-)}}, \tag{3.44}$$

where $[\cdot]$ denotes $\max(\cdot, 0)$. The derivation for the optimal value is deferred to Appendix 3.2.4. Note that if $\boldsymbol{M}$ is the identity matrix and $d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}^-) > d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}^+)$ strictly holds, the optimal value has a clear geometric meaning: it is the Euclidean distance from $\boldsymbol{x}$ to the bisection between $\boldsymbol{x}^+$ and $\boldsymbol{x}^-$.

For convenience, we define the function $\tilde{\epsilon} : \mathbb{R}^D \times \mathbb{R}^D \times \mathbb{R}^D \to \mathbb{R}$ as

$$\tilde{\epsilon}(\boldsymbol{x}^+, \boldsymbol{x}^-, \boldsymbol{x}; \boldsymbol{M}) = \frac{d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}^-) - d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}^+)}{2\sqrt{(\boldsymbol{x}^+ - \boldsymbol{x}^-)^\top \boldsymbol{M}^\top \boldsymbol{M}(\boldsymbol{x}^+ - \boldsymbol{x}^-)}}. \tag{3.45}$$

Then we could relax (3.42) further and have the following theorem:

Theorem 3 (Robustness verification for Mahalanobis K-NN). *Given a Mahalanobis K-NN classifier parameterized by a neighbor parameter K, a training dataset $\mathbb{S}$ and a positive semi-definite matrix $\boldsymbol{M}$, for any instance $(\boldsymbol{x}_{test}, y_{test})$ we have*

$$\epsilon^*(\boldsymbol{x}_{test}, y_{test}; \boldsymbol{M}) \geq k\text{th}\min_{j:y_j \neq y_{test}} k\text{th}\max_{i:y_i = y_{test}} \tilde{\epsilon}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_{test}; \boldsymbol{M}), \tag{3.46}$$

where $k\text{th}\max$ and $k\text{th}\min$ select the k-th maximum and k-th minimum respectively with $k = (K+1)/2$.

The proof is deferred to Appendix 3.2.5. In this way, we only need to compute $\tilde{\epsilon}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_{\text{test}})$ for each i and j in order to derive a lower bound of the minimal adversarial perturbation of Mahalanobis K-NN. It leads to an efficient algorithm to verify the

robustness of Mahalanobis K-NN. The time complexity is $O(N^2)$ and independent of K. Note that any subset of $\{i : y_i = y_{\text{test}}\}$ also leads to a feasible lower bound of the minimal adversarial perturbation and could improve computational efficiency, but the resulting lower bound is not necessarily as tight as (3.46). Therefore, in the experimental section, to evaluate certified robust errors as accurately as possible, we do not employ this strategy.

In the general multi-class case, the constraint of (3.42) is the necessary condition for successful attacks, rather than the necessary and sufficient condition. As a result, the optimal value of (3.42) is a lower bound of the minimal adversarial perturbation. Therefore, Theorem 3 also holds for the multi-class case. Based on this lower bound of ϵ^*, we will derive the proposed ARML algorithm.

Training algorithm of adversarially robust metric learning

By replacing the ϵ^* in (3.41) with the lower bound derived in Theorem 3, we get a trainable objective function for adversarially robust metric learning:

$$\min_{\boldsymbol{G} \in \mathbb{R}^{D \times D}} \quad \frac{1}{N} \sum_{t=1}^{N} \ell \left(\underset{j: y_j \neq y_t}{k\text{th} \min} \; \underset{i: i \neq t, y_i = y_t}{k\text{th} \max} \tilde{\epsilon}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_t; \boldsymbol{M}) \right) \quad \text{s.t. } \boldsymbol{M} = \boldsymbol{G}^\top \boldsymbol{G}. \qquad (3.47)$$

Although (3.47) is trainable since $\tilde{\epsilon}$ is a function of $\boldsymbol{M}$, for large datasets it is time-consuming to run the inner min-max procedure. Furthermore, since what we really care is the generalization performance of the learned metric instead of the leave-one-out robust training error, it is unnecessary to compute the exact solution. Therefore, instead of computing the kth max and kth min exactly, we propose to sample positive and negative instances from the neighborhood of each training instance, which leads to the following formulation:

$$\min_{\boldsymbol{G} \in \mathbb{R}^{D \times D}} \quad \frac{1}{N} \sum_{i=1}^{N} \ell \left(\tilde{\epsilon} \left(\text{randnear}_{\boldsymbol{M}}^{+}(\boldsymbol{x}_i), \text{randnear}_{\boldsymbol{M}}^{-}(\boldsymbol{x}_i), \boldsymbol{x}_i; \boldsymbol{M} \right) \right) \quad \text{s.t. } \boldsymbol{M} = \boldsymbol{G}^\top \boldsymbol{G}, \qquad (3.48)$$

where $\text{randnear}_{\boldsymbol{M}}^{+}(\cdot)$ denotes a sampling procedure for an instance in the same class within $\boldsymbol{x}_i$'s neighborhood, and $\text{randnear}_{\boldsymbol{M}}^{-}(\cdot)$ denotes a sampling procedure for an instance in a different

class, also within $\boldsymbol{x}_i$'s neighborhood, and the distances are measured by the Mahalanobis distance $d_{\boldsymbol{M}}$. In our implementation, we sample instances from a fixed number of nearest instances. As a result, the optimization formulation (3.48) approximately minimizes the certified robust error and improves computational efficient significantly.

Our *adversarially robust metric learning* (ARML) algorithm is shown in Algorithm 7. At every iteration, $\boldsymbol{G}$ is updated with the gradient of the loss function, while the calculations of $\text{randnear}_{\boldsymbol{M}}^+(\cdot)$ and $\text{randnear}_{\boldsymbol{M}}^-(\cdot)$ do not contribute to the gradient for the sake of efficient and stable computation.

Algorithm 7 Adversarially robust metric learning (ARML)

1: **Input:** Training data $\mathbb{S}$, number of epochs T.

2: **Output:** Positive semi-definite matrix $\boldsymbol{M}$.

3: Initialize $\boldsymbol{G}$ and $\boldsymbol{M}$ as identity matrices.

4: **for** $t = 0 \dots T - 1$ **do**

5:      Update $\boldsymbol{G}$ with the gradient $\mathbb{E}_{(\boldsymbol{x},y)\in\mathbb{S}} \nabla_{\boldsymbol{G}} \ell \left(\tilde{\epsilon} \left(\text{randnear}_{\boldsymbol{M}}^+(\boldsymbol{x}), \text{randnear}_{\boldsymbol{M}}^-(\boldsymbol{x}), \boldsymbol{x}; \boldsymbol{G}^\top \boldsymbol{G} \right) \right)$

6:      Update $\boldsymbol{M}$ with the constraint $\boldsymbol{M} = \boldsymbol{G}^\top \boldsymbol{G}$

7: **end for**

Exact minimal adversarial perturbation of Mahalanobis 1-NN

In the special Mahalanobis 1-NN case, we will show a method to compute the exact minimal adversarial perturbation in a similar formulation to (3.42). However, this algorithm can only compute a numerical value of the minimal adversarial perturbation $\boldsymbol{\delta}^*$, so it cannot be used in training time. We will use this method to evaluate the robust error for the Mahalanobis 1-NN case in the experiments.

Computing the minimal adversarial perturbation $\epsilon_{\mathbb{S}}^*(\boldsymbol{x}_{\text{test}}, y_{\text{test}}; \boldsymbol{M})$ for Mahalanobis 1-NN classifier can be formulated as the following optimization problem:

$$\min_{j:y_j\neq y_{\text{test}}} \min_{\boldsymbol{\delta}_j} \|\boldsymbol{\delta}_j\| \ \text{ s.t. } d_{\boldsymbol{M}}(\boldsymbol{x}_{\text{test}} + \boldsymbol{\delta}_j, \boldsymbol{x}_j) \leq d_{\boldsymbol{M}}(\boldsymbol{x}_{\text{test}} + \boldsymbol{\delta}_j, \boldsymbol{x}_i), \ \forall i : y_i = y_{\text{test}}. \qquad (3.49)$$

Interestingly and not surprisingly, it is a special case of (3.42) where we have $K = 1$ and $k = (K+1)/2 = 1$, and hence $\mathbb{I}$ is an empty set, and $\mathbb{J}$ has only one element. The formulation of (3.49) is equivalent to considering each $\boldsymbol{x}_j$ in a different class from y_{test} and computing the minimum perturbation needed for making $\boldsymbol{x}_{\text{test}}$ closer to $\boldsymbol{x}_j$ than all the training instances in the same class with y_{test}, i.e., $\boldsymbol{x}_i$s,. It is noteworthy that the constraint of (3.49) could be equivalently written as

$$(\boldsymbol{x}_i - \boldsymbol{x}_j)^\top \boldsymbol{M} \boldsymbol{\delta} \leq \frac{1}{2} \left(d_{\boldsymbol{M}}(\boldsymbol{x}_{\text{test}}, \boldsymbol{x}_i) - d_{\boldsymbol{M}}(\boldsymbol{x}_{\text{test}}, \boldsymbol{x}_j) \right), \ \forall i : y_i = y_{\text{test}}, \tag{3.50}$$

which are all affine functions. Therefore, the inner minimization is a convex quadratic programming problem and could be solved in polynomial time [KTK80]. As a result, it leads to a naive polynomial-time algorithm for finding the minimal adversarial perturbation of Mahalanobis 1-NN: solve all the inner convex quadratic programming problems and then select the minimum of them.

Instead, we propose a much more efficient method to solve (3.49). The main idea is to compute a lower bound for each inner minimization problem first, and with these lower bounds, we could screen most of the inner minimization problems safely without the need of solving them exactly. This method is an extension of our previous work [WLY19], where only the Euclidean distance is taken into consideration. See Algorithm 8 in Appendix 3.2.6 for details and this algorithm is used for computing certified robust errors of Mahalanobis 1-NN in the experimental section.

3.2.3 Experiments

We compare the proposed ARML (Adversarial Robust Metric Learning) method with the following baselines:

- Euclidean: uses the Euclidean distance directly without learning any metric;

- Neighbourhood components analysis (NCA) [GHR04]: maximizes a stochastic variant

of the leave-one-out nearest neighbors score on the training set.

- Large margin nearest neighbor (LMNN) [WS09]: keeps close nearest neighbors from the same class, while keeps instances from different classes separated by a large margin.

- Information Theoretic Metric Learning (ITML) [DKJ07]: minimizes the log-determinant divergence with similarity and dissimilarity constraints.

- Local Fisher Discriminant Analysis (LFDA) [Sug07]: a modified version of linear discriminant analysis by rewriting scatter matrices in a pairwise manner.

For evaluation, we use six public datasets on which metric learning methods perform favorably in terms of clean errors, including four small or medium-sized datasets [CL11]: Splice, Pendigits, Satimage and USPS, and two image datasets MNIST [LBB98] and Fashion-MNIST [XRV17], which are wildly used for robustness verification for neural networks. For the proposed method, we use the same hyperparameters for all the datasets.

Mahalanobis 1-NN

Certified robust errors of Mahalanobis 1-NN with respect to different perturbation radii are shown in Table 3.4. It should be noted that these radii are only used to show the experimental results, and they are not hyperparameters. In this Mahalanobis 1-NN case, the proposed algorithm in Algorithm 8, which solves (3.49), can compute the exact minimal adversarial perturbation for each instance, so the values we get in Table 3.4 are both (optimal) certified robust errors and (optimal) empirical robust errors (attack errors). Also, note that when the radius is 0, the resulting certified robust error is equivalent to the clean error on the unperturbed test set.

We have three main observations from the experimental results. First, although NCA and LMNN achieve better clean errors (at the radius 0) than Euclidean in most datasets, they are less robust to adversarial perturbations than Euclidean (except the Splice dataset, on which

Euclidean performs overly poorly in terms of clean errors and then has a large robust errors accordingly). Both NCA and LMNN suffer from the trade-off between the clean error and the certified robust error. Second, ARML performs competitively with NCA and LMNN in terms of clean errors (achieves the best on 4/6 of the datasets). Third and the most importantly, ARML is much more robust than all the other methods in terms of certified robust errors for all perturbation radii.

Mahalanobis K-NN

For K-NN models, it is intractable to compute the exact minimal adversarial perturbation, so we report both certified robust errors and empirical robust errors (attack errors). We set $K = 11$ for all the experiments. The certified robust error can be computed by Theorem 3, which works for any Mahalanobis distance. On the other hand, we also conduct adversarial attacks to these models to derive the empirical robust error — the lower bounds of the certified robust errors — via a hard-label black-box attack method, i.e., the Boundary Attack [BRB18]. Different from the 1-NN case, since both adversarial attack and robustness verification are not optimal, there will be a gap between the two kinds of robust errors. These results are shown in Table 3.6. Note that these empirical robust errors at the radius 0 are also the clean errors.

The three observations of Mahalanobis 1-NN also hold for the K-NN case: NCA and LMNN have improved clean errors (empirical robust errors at the radius 0) but this often comes with degraded robust errors compared with the Euclidean distance, while ARML achieves good robust errors as well as clean errors. The results suggest that ARML is more robust both *provably* (in terms of the certified robust error) and empirically (in terms of the empirical robust error).

Comparison with neural networks

We compare Mahalanobis 1-NN classifiers with neural networks, including ordinary neural networks certified by the robustness verification method CROWN [ZCX20] and randomized-smoothing neural networks (with smoothness parameters 0.2 and 1) [CRK19]. The results are shown in Figure 3.7. It is shown that randomized smoothing encounters a trade-off between clean and robust errors, whereas ARML does not sacrifice clean errors compared with previous metric learning methods.

(a) MNIST. (b) Fashion-MNIST.
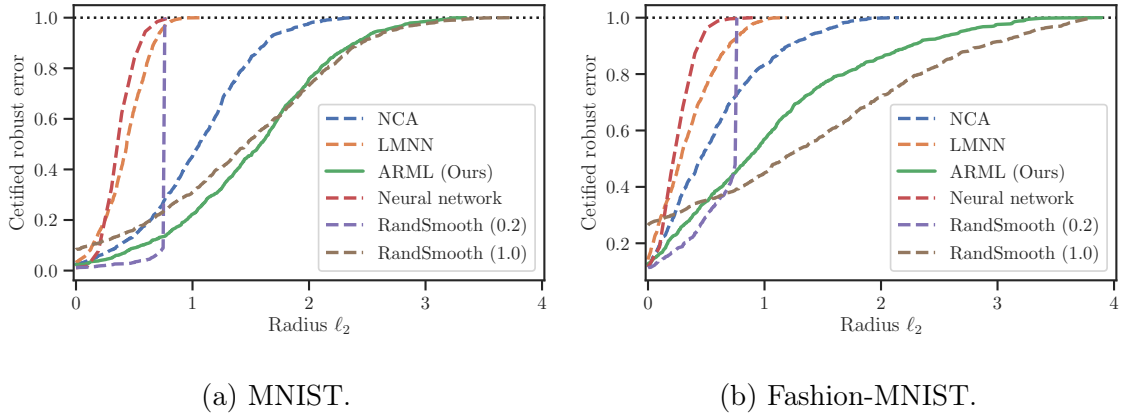
Figure 3.7: Certified robust errors comparing neural networks.

3.2.4 Optimal value of triplet problem

The triplet problem is formalized as below:

$$\min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\| \quad \text{s.t. } d_{\boldsymbol{M}}(\boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{x}^-) \leq d_{\boldsymbol{M}}(\boldsymbol{x} + \boldsymbol{\delta}, \boldsymbol{x}^+). \tag{3.51}$$

It is equivalent to the optimization

$$\min_{\boldsymbol{\delta}} \boldsymbol{\delta}^\top \boldsymbol{\delta} \quad \text{s.t. } \boldsymbol{a}^\top \boldsymbol{\delta} \leq b, \tag{3.52}$$

where we have

$$\boldsymbol{a} = \boldsymbol{M} \left(\boldsymbol{x}^+ - \boldsymbol{x}^- \right), \tag{3.53}$$

$$b = \frac{1}{2} \left(d_M(\boldsymbol{x}, \boldsymbol{x}^+) - d_M(\boldsymbol{x}, \boldsymbol{x}^-) \right). \tag{3.54}$$

The dual function is

$$g(\lambda) = \inf_{\boldsymbol{\delta}} \ \boldsymbol{\delta}^\top \boldsymbol{\delta} + \lambda(\boldsymbol{a}^\top \boldsymbol{\delta} - b) \tag{3.55}$$

$$= -\frac{1}{4} \boldsymbol{a}^\top \boldsymbol{a} \lambda^2 - b\lambda, \tag{3.56}$$

where inf holds for $\boldsymbol{\delta} = -\lambda \boldsymbol{a}/2$. Then the dual problem is

$$\max_{\lambda \geq 0} \ -\frac{1}{4} \boldsymbol{a}^\top \boldsymbol{a} \lambda^2 - b\lambda. \tag{3.57}$$

The optimal point is

$$\left[-\frac{2b}{\boldsymbol{a}^\top \boldsymbol{a}} \right]_+, \tag{3.58}$$

and the optimal value is

$$\begin{cases} 0 & \text{if } b \geq 0 \\ \frac{b^2}{\boldsymbol{a}^\top \boldsymbol{a}} & \text{otherwise.} \end{cases} \tag{3.59}$$

By the Slater's condition, if $\boldsymbol{x}^+ \neq \boldsymbol{x}^-$ holds, we have the strong duality. Therefore, the optimal value of (3.51) is

$$\left[\frac{-b}{\sqrt{\boldsymbol{a}^\top \boldsymbol{a}}} \right]_+ = \frac{\left[d_M(\boldsymbol{x}, \boldsymbol{x}^-) - d_M(\boldsymbol{x}, \boldsymbol{x}^+) \right]_+}{2\sqrt{(\boldsymbol{x}^+ - \boldsymbol{x}^-)^\top \boldsymbol{M}^\top \boldsymbol{M}(\boldsymbol{x}^+ - \boldsymbol{x}^-)}}. \tag{3.60}$$

In fact, it is easy to verify that even if $\boldsymbol{x}^+ = \boldsymbol{x}^-$ obtains, the optimal value also holds.

3.2.5   Proof of Theorem 3

Proof. Let $\epsilon^{(\mathbb{I},\mathbb{J})}$ denote the optimal value of the inner minimization problem of (3.42). By relaxing the constraint via replacing the universal quantifier, we have

$$\epsilon^{(\mathbb{I},\mathbb{J})} \geq \max_{i \in \{i:y_i = y_{\text{test}}\} - \mathbb{I}, j \in \mathbb{J}} \tilde{\epsilon}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_{\text{test}}; \boldsymbol{M}). \tag{3.61}$$

Substitute it in (3.42) and then we have

$$\epsilon^* \geq \min_{\mathbb{I},\mathbb{J}} \epsilon^{(\mathbb{I},\mathbb{J})} \tag{3.62}$$

$$\geq \min_{\mathbb{I},\mathbb{J}} \max_{i \in \{i:y_i=y_{\text{test}}\}-\mathbb{I},\ j \in \mathbb{J}} \tilde{\epsilon}(\boldsymbol{x}_i^+, \boldsymbol{x}_j^-, \boldsymbol{x}_{\text{test}}) \tag{3.63}$$

$$\geq \min_{\mathbb{I},\mathbb{J}} \max_{j \in \mathbb{J}} \max_{i \in [\{i:y_i=y_{\text{test}}\}-\mathbb{I}} \tilde{\epsilon}(\boldsymbol{x}_i^+, \boldsymbol{x}_j^-, \boldsymbol{x}_{\text{test}}) \tag{3.64}$$

$$\geq \min_{\mathbb{I},\mathbb{J}} \max_{j \in \mathbb{J}} \ k\text{th}\max_{i \in \{i:y_i=y_{\text{test}}\}} \tilde{\epsilon}(\boldsymbol{x}_i^+, \boldsymbol{x}_j^-, \boldsymbol{x}_{\text{test}}) \tag{3.65}$$

$$\geq \min_{\mathbb{I},\mathbb{J}} \ k\text{th}\min_{j \in \{j:y_j \neq y_{\text{test}}\}} \ k\text{th}\max_{i \in \{i:y_i=y_{\text{test}}\}} \tilde{\epsilon}(\boldsymbol{x}_i^+, \boldsymbol{x}_j^-, \boldsymbol{x}_{\text{test}}) \tag{3.66}$$

$$= \ k\text{th}\min_{j \in \{j:y_j \neq y_{\text{test}}\}} \ k\text{th}\max_{i \in \{i:y_i=y_{\text{test}}\}} \tilde{\epsilon}(\boldsymbol{x}_i^+, \boldsymbol{x}_j^-, \boldsymbol{x}_{\text{test}}) \tag{3.67}$$

$$\square$$

3.2.6 Details of computing exact minimal adversarial perturbation of Mahalanobis 1-NN

The overall algorithm is displayed in Algorithm 8. We denote $\epsilon^{(j)}$ as the optimal value of the inner minimization problem with respect to j, and denote $\underline{\epsilon}^{(j)}$ as its lower bound. We first sort the subproblems according to the ascending order of $\|\boldsymbol{x}_j - \boldsymbol{x}_{\text{test}}\|$ for $\{j : y_j \neq y_{\text{test}}\}$. For every subproblem, we compute the lower bound of its optimal value. If the optimal value is too large, we just screen the subproblem safely without solving it exactly.

Greedy coordinate ascent (descent)

For the subproblem we have to solve exactly, we employ the greedy coordinate ascent method. Note that the inner minimization problem of (3.49) is a convex quadratic programming problem. We solve the problem by dealing with its dual formulation. The greedy coordinate ascent method is used because the optimal dual variables are very sparse. The algorithm is shown in Algorithm 9. At every iteration, only one dual variable is updated.

Algorithm 8 Computing the minimal adversarial perturbation for Mahalanobis 1-NN
___
1: **Input:** Test instance $(\boldsymbol{x}_{\text{test}}, y_{\text{test}})$, dataset $\mathbb{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^N$.

2: **Output:** Perturbation norm ϵ.

3: Initialize $\epsilon = \infty$.

4: Sort $\{j : y_j \neq y_{\text{test}}\}$ by the ascending order of $d_{\boldsymbol{M}}(\boldsymbol{x}_{\text{test}}, \boldsymbol{x}_j)$

5: **for** $j : y_j \neq y_{\text{test}}$ according to the ascending order **do**

6:　　Compute a lower bound $\underline{\epsilon}^{(j)}$ of the inner minimization corresponding to j

7:　　**if** $\underline{\epsilon} < \epsilon$ **then**

8:　　　　Solve the inner minimization problem exactly via the greedy coordinate ascent method and derive the optimal value $\epsilon^{(j)}$

9:　　　　**if** $\epsilon^{(j)} < \epsilon$ **then**

10:　　　　　$\epsilon = \epsilon^{(j)}$

11:　　　　**end if**

12:　　**end if**

13: **end for**
___

Lower bound of inner minimization problem

The following theorem is dependent on the solution of the triplet problem.

Theorem 4. *The optimal value $\epsilon^{(j)}$ of the inner minimization of* (3.49) *with respect to j is lower bounded as*

$$\epsilon^{(j)} \geq \max_{i:y_i=y_{test}} \left[\tilde{\epsilon}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_{test}; \boldsymbol{M}) \right]_+. \tag{3.68}$$

Proof. Relaxing the constraint of (3.49) by means of replacing the universal quantifier, we know $\epsilon^{(j)}$ is lower bounded by the optimal value of the following optimization problem

$$\max_{i:y_i=y_{\text{test}}} \min_{\boldsymbol{\delta}_{i,j}} \ \|\boldsymbol{\delta}_{i,j}\| \tag{3.69}$$

$$\text{s.t.} \ \ d_{\boldsymbol{M}}(\boldsymbol{x}_{\text{test}} + \boldsymbol{\delta}_{i,j}, \boldsymbol{x}_j) \leq d_{\boldsymbol{M}}(\boldsymbol{x}_{\text{test}} + \boldsymbol{\delta}_{i,j}, \boldsymbol{x}_i). \tag{3.70}$$

Algorithm 9 Greedy coordinate descent for QP: $\min_{\boldsymbol{x} \geq 0} \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{P} \boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x}$

1: **Input:** $\boldsymbol{P}$, $\boldsymbol{q}$, ϵ, T, $\boldsymbol{x} \leftarrow \boldsymbol{0}$, $\boldsymbol{g} \leftarrow \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}$.

2: **Output:** $\boldsymbol{x}$.

3: **for** $t = 0$ to $T - 1$ **do**

4:      $\forall i,\ y_i \leftarrow \max\left(x_i - \frac{g_i}{p_{i,i}}, 0\right) - x_i$

5:      // choose a coordinate

6:      $i^* \leftarrow \arg\max_i |y_i|$

7:      // update the solution

8:      $x_{i^*} \leftarrow x_{i^*} + y_{i^*}$

9:      // update the gradient

10:     $\boldsymbol{g} \leftarrow \boldsymbol{g} + y_{i^*}\boldsymbol{p}_{i^*}$

11: **end for**

Obviously, the optimal value of the inner problem is $\left[\tilde{\epsilon}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{x}_{\text{test}}; \boldsymbol{M})\right]_+$. $\square$

In this way, we could derive a lower bound of the optimal value in closed form.

Table 3.4: Certified robust errors of Mahalanobis 1-NN. The best (minimum) certified robust errors among all methods are in bold. Note that the certified robust errors of 1-NN are also the optimal empirical robust errors (attack errors), and these robust errors at the radius 0 are also the clean errors.

MNIST	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500	Pendigits	ℓ_2-radius	0.000	0.100	0.200	0.300	0.400	0.500
	Euclidean	0.033	0.112	0.274	0.521	0.788	0.945		Euclidean	0.032	0.119	0.347	0.606	0.829	0.969
	NCA	0.025	0.140	0.452	0.839	0.977	1.000		NCA	0.034	0.202	0.586	0.911	0.997	1.000
	LMNN	0.032	0.641	0.999	1.000	1.000	1.000		LMNN	0.029	0.183	0.570	0.912	0.995	0.999
	ITML	0.073	0.571	0.928	1.000	1.000	1.000		ITML	0.049	0.308	0.794	0.991	1.000	1.000
	LFDA	0.152	1.000	1.000	1.000	1.000	1.000		LFDA	0.042	0.236	0.603	0.912	0.998	1.000
	ARML (Ours)	**0.024**	**0.089**	**0.222**	**0.455**	**0.757**	**0.924**		ARML (Ours)	**0.028**	**0.115**	**0.344**	**0.598**	**0.823**	**0.967**

Fashion-MNIST	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500	Satimage	ℓ_2-radius	0.000	0.150	0.300	0.450	0.600	0.750
	Euclidean	0.145	0.381	0.606	0.790	0.879	0.943		Euclidean	0.108	0.642	0.864	0.905	0.928	0.951
	NCA	**0.116**	0.538	0.834	0.950	0.998	1.000		NCA	0.103	0.710	0.885	0.915	0.940	0.963
	LMNN	0.142	0.756	0.991	1.000	1.000	1.000		LMNN	**0.092**	0.665	0.871	0.912	0.944	0.969
	ITML	0.163	0.672	0.929	0.998	1.000	1.000		ITML	0.127	0.807	0.979	1.000	1.000	1.000
	LFDA	0.211	1.000	1.000	1.000	1.000	1.000		LFDA	0.125	0.836	0.919	0.956	0.992	1.000
	ARML (Ours)	0.127	**0.348**	**0.568**	**0.763**	**0.859**	**0.928**		ARML (Ours)	0.095	**0.605**	**0.839**	**0.899**	**0.920**	**0.946**

Splice	ℓ_2-radius	0.000	0.100	0.200	0.300	0.400	0.500	USPS	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500
	Euclidean	0.320	0.513	0.677	0.800	0.854	0.880		Euclidean	0.045	0.224	0.585	0.864	**0.970**	**0.999**
	NCA	**0.130**	0.252	0.404	0.584	0.733	0.836		NCA	0.056	0.384	0.888	0.987	1.000	1.000
	LMNN	0.190	0.345	0.533	0.697	0.814	0.874		LMNN	0.046	0.825	1.000	1.000	1.000	1.000
	ITML	0.306	0.488	0.679	0.809	0.862	0.882		ITML	0.060	0.720	0.999	1.000	1.000	1.000
	LFDA	0.264	0.434	0.605	0.760	0.845	0.872		LFDA	0.098	1.000	1.000	1.000	1.000	1.000
	ARML (Ours)	**0.130**	**0.233**	**0.370**	**0.526**	**0.652**	**0.758**		ARML (Ours)	**0.043**	**0.204**	**0.565**	**0.857**	**0.970**	**0.999**

Table 3.5: Certified robust errors (left) and empirical robust errors (right) of Mahalanobis K-NN. The best (minimum) robust errors among all methods are in bold. The empirical robust errors at the radius 0 are also the clean errors.

		Certified robust errors						Empirical robust errors					
	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500	0.000	0.500	1.000	1.500	2.000	2.500
MNIST	Euclidean	0.038	0.134	0.360	0.618	0.814	0.975	0.031	0.063	0.104	0.155	0.204	0.262
	NCA	**0.030**	0.175	0.528	0.870	0.986	1.000	**0.027**	0.063	0.120	0.216	0.330	0.535
	LMNN	0.040	0.669	1.000	1.000	1.000	1.000	0.036	0.121	0.336	0.775	0.972	1.000
	ITML	0.106	0.731	0.943	1.000	1.000	1.000	0.084	0.218	0.355	0.510	0.669	0.844
	LFDA	0.237	1.000	1.000	1.000	1.000	1.000	0.215	1.000	1.000	1.000	1.000	1.000
	ARML (Ours)	0.034	**0.101**	**0.276**	**0.537**	**0.760**	**0.951**	0.032	**0.055**	**0.077**	**0.109**	**0.160**	**0.213**
	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500	0.000	0.500	1.000	1.500	2.000	2.500
Fashion-MNIST	Euclidean	0.160	0.420	0.650	0.800	0.895	0.946	0.143	0.227	0.298	0.360	0.420	0.489
	NCA	**0.144**	0.557	0.832	0.946	1.000	1.000	**0.121**	0.232	0.343	0.483	0.624	0.780
	LMNN	0.158	0.792	0.991	1.000	1.000	1.000	0.140	0.364	0.572	0.846	0.983	0.999
	ITML	0.236	0.784	0.949	1.000	1.000	1.000	0.209	0.460	0.692	0.892	0.978	1.000
	LFDA	0.291	1.000	1.000	1.000	1.000	1.000	0.263	0.870	0.951	0.975	0.988	0.995
	ARML (Ours)	0.152	**0.371**	**0.589**	**0.755**	**0.856**	**0.924**	0.134	**0.202**	**0.274**	**0.344**	**0.403**	**0.487**
	ℓ_2-radius	0.000	0.100	0.200	0.300	0.400	0.500	0.000	0.100	0.200	0.300	0.400	0.500
Splice	Euclidean	0.333	0.558	0.826	0.965	0.988	0.996	0.306	0.431	0.526	0.608	0.676	0.743
	NCA	**0.103**	**0.209**	0.415	0.659	0.824	0.921	**0.103**	**0.173**	0.274	0.414	0.570	0.684
	LMNN	0.149	0.332	0.630	0.851	0.969	0.994	0.149	0.241	0.357	0.492	0.621	0.722
	ITML	0.279	0.571	0.843	0.974	0.995	0.997	0.279	0.423	0.525	0.603	0.675	0.751
	LFDA	0.242	0.471	0.705	0.906	0.987	0.997	0.242	0.371	0.466	0.553	0.637	0.737
	ARML (Ours)	0.128	0.221	**0.345**	**0.509**	**0.666**	**0.819**	0.128	0.196	**0.273**	**0.380**	**0.497**	**0.639**
	ℓ_2-radius	0.000	0.100	0.200	0.300	0.400	0.500	0.000	0.100	0.200	0.300	0.400	0.500
Pendigits	Euclidean	0.039	0.126	0.316	0.577	0.784	0.937	0.036	0.085	0.155	0.248	0.371	0.528
	NCA	0.038	0.196	0.607	0.884	0.997	1.000	0.038	0.103	0.246	0.428	0.637	0.804
	LMNN	0.034	0.180	0.568	0.898	0.993	0.999	**0.030**	0.096	0.246	0.462	0.681	0.862
	ITML	0.060	0.334	0.773	0.987	1.000	1.000	0.060	0.149	0.343	0.616	0.814	0.926
	LFDA	0.047	0.228	0.595	0.904	1.000	1.000	0.043	0.104	0.248	0.490	0.705	0.842
	ARML (Ours)	**0.035**	**0.114**	**0.308**	**0.568**	**0.780**	**0.937**	0.034	**0.078**	**0.138**	**0.235**	**0.368**	**0.516**

Table 3.6: **(Continue)** Certified robust errors (left) and empirical robust errors (right) of Mahalanobis K-NN. The best (minimum) robust errors among all methods are in bold. The empirical robust errors at the radius 0 are also the clean errors.

		Certified robust errors						Empirical robust errors					
	ℓ_2-radius	0.000	0.150	0.300	0.450	0.600	0.750	0.000	0.150	0.300	0.450	0.600	0.750
Satimage	Euclidean	**0.101**	0.579	0.842	0.899	0.927	0.948	0.091	0.237	0.482	0.682	**0.816**	0.897
	NCA	0.117	0.670	0.886	0.915	0.936	0.961	0.101	0.297	0.564	0.746	0.876	0.931
	LMNN	0.105	0.613	0.855	0.914	0.944	0.961	**0.090**	0.269	0.548	0.737	0.855	0.910
	ITML	0.130	0.768	0.959	1.000	1.000	1.000	0.109	0.411	0.757	0.939	0.990	1.000
	LFDA	0.128	0.779	0.904	0.958	0.995	1.000	0.112	0.389	0.673	0.860	0.950	0.986
	ARML (Ours)	0.103	**0.540**	**0.824**	**0.898**	**0.920**	**0.943**	0.092	**0.228**	**0.464**	**0.668**	0.817	**0.896**
	ℓ_2-radius	0.000	0.500	1.000	1.500	2.000	2.500	0.000	0.500	1.000	1.500	2.000	2.500
USPS	Euclidean	0.063	0.239	0.586	0.888	0.977	1.000	0.058	0.125	0.211	0.365	0.612	**0.751**
	NCA	0.072	0.367	0.903	0.986	1.000	1.000	0.063	0.158	0.365	0.686	0.899	0.980
	LMNN	0.062	0.856	1.000	1.000	1.000	1.000	0.055	0.359	0.890	0.999	1.000	1.000
	ITML	0.082	0.696	0.999	1.000	1.000	1.000	0.072	0.273	0.708	0.987	1.000	1.000
	LFDA	0.134	1.000	1.000	1.000	1.000	1.000	0.118	0.996	1.000	1.000	1.000	1.000
	ARML (Ours)	**0.057**	**0.203**	**0.527**	**0.867**	**0.971**	**0.997**	**0.053**	**0.118**	**0.209**	**0.344**	**0.572**	0.785

CHAPTER 4

Unadversarial Robustness

4.1 Case study: Learning to Encode Position for Transformer Models

We introduce a new way of learning to encode position information for non-recurrent models, such as Transformer models. Unlike RNN and LSTM, which contain inductive bias by loading the input tokens sequentially, non-recurrent models are less sensitive to position. The main reason is that position information among input units is not inherently encoded, i.e., the models are permutation equivalent; this problem justifies why all of the existing models are accompanied by a sinusoidal encoding/embedding layer at the input. However, this solution has clear limitations: the sinusoidal encoding is not flexible enough as it is manually designed and does not contain any learnable parameters, whereas the position embedding restricts the maximum length of input sequences. It is thus desirable to design a new position layer that contains learnable parameters to adjust to different datasets and different architectures. At the same time, we would also like the encodings to extrapolate in accordance with the variable length of inputs. In our proposed solution, we borrow from the recent Neural ODE approach, which may be viewed as a versatile continuous version of a ResNet. This model is capable of modeling many kinds of dynamical systems. We model the evolution of encoded results along position index by such a dynamical system, thereby overcoming the above limitations of existing methods. We evaluate our new position layers on a variety of neural machine translation and language understanding tasks, the experimental

results show consistent improvements over the baselines.

4.1.1 Introduction

Transformer based models [VSP17, DCL18a, YDY19, RWC19, LCG19, RSR19] have become one of the most effective approaches to model sequence data of variable lengths. Transformers have shown wide applicability to many natural language processing (NLP) tasks such as language modeling [RWC19], neural machine translation (NMT) [VSP17], and language understanding [DCL18a]. Unlike traditional recurrent-based models (e.g., RNN or LSTM), Transformer utilizes a non-recurrent but self-attentive neural architecture to model the dependency among elements at different positions in the sequence, which leads to better parallelization using modern hardware and alleviates the vanishing/exploding gradient problem in traditional recurrent models.

It is known that the self-attentive architecture corresponds to a family of permutation equivalence functions. Thus, for applications where the ordering of the elements matters, how to properly encode position information is crucial for Transformer based models. There have been many attempts to encode position information for the Transformer. In the original Transformer paper [VSP17], a family of pre-defined sinusoidal functions was adapted to construct a set of embeddings for each position. These fixed position embeddings are then added to the word embeddings of the input sequence accordingly. To further construct these position embeddings in a more data-driven way, many recent Transformer variants such as [DCL18a, LOG19] include these embeddings as learnable model parameters in the training stage. This data-driven approach comes at the cost of limiting the maximum length of sequence $L_{\max}$ and the computational/memory overhead from additional $L_{\max} \times d$ parameters, where $L_{\max}$ is usually set to 512 in many applications, and d is the dimension of the embeddings. [SUV18] propose a relative position representation to reduce the number of parameters to $(2K + 1)d$ by dropping the interactions between tokens with a distance greater than K. In addition to just the input layer, [DGV18] and [LCG19] suggest that the injection

108

of position information to every layer leads to even better performance for the Transformer. More recently, [WZL19] proposes to encode positions by complex numbers of different phases and wave-lengths. Despite being successful in several tasks, it increases the embedding matrix size by a factor of three.

An ideal position encoding approach should satisfy the following three properties:

1. **Inductive**: the ability to handle sequences longer than any sequence seen in the training time.

2. **Data-Driven**: the position encoding should be learnable from the data.

3. **Parameter Efficient**: number of trainable parameters introduced by the encoding should be limited to avoid increased model size, which could hurt generalization.

In Table 4.1, we summarize some of the existing position encoding approaches in terms of these three properties.

We propose a new method to encode position with minimum cost. The main idea is to model position encoding as a continuous dynamical system, so we only need to learn the system dynamics instead of learning the embeddings for each position independently. By doing so, our method enjoys the best of both worlds – we bring back the inductive bias, and the encoding method is freely trainable while being parameter efficient. To enable training of this dynamical system with backpropagation, we adopt the recent progress in continuous neural network [CRB18], officially called Neural ODE. In some generative modeling literature, it is also called the free-form flow model [GCB18], so we call our model **FLOw-bAsed TransformER (FLOATER)**. We highlight our contributions as follows:

- We propose FLOATER, a new position encoder for Transformer, which models the position information via a continuous dynamical model in a data-driven and parameter-efficient manner.

Table 4.1: Comparing position representation methods

Methods	Inductive	Data-Driven	Parameter Efficient
Sinusoidal [VSP17]	✓	✗	✓
Embedding [DCL18a]	✗	✓	✗
Relative [SUV18]	✗	✓	✓
This paper	✓	✓	✓

- Due to the use of a continuous dynamic model, FLOATER can handle sequences of any length. This property makes inference more flexible.

- With careful design, our position encoder is **compatible** with the original Transformer; *i.e.*, the original Transformer can be regarded as a special case of our proposed position encoding approach. As a result, we are not only able to train a Transformer model with FLOATER from scratch but also plug FLOATER into most existing pre-trained Transformer models such as BERT, RoBERTa, *etc.*

- We demonstrate that FLOATER consistent improvements over baseline models across a variety of NLP tasks ranging from machine translations, language understanding, and question answering.

4.1.2 Background and related work

Importance of position encoding for transformer

We use a simplified self-attentive sequence encoder to illustrate the importance of position encoding in the Transformer. Without position encoding, the Transformer architecture can be viewed as a stack of N blocks $B_n : n = 1, \dots, N$ containing a self-attentive A_n and a

feed-forward layer F_n. By dropping the residual connections and layer normalization, the architecture of a simplified Transformer encoder can be represented as follows.

$$\text{Encode}(\boldsymbol{x}) = B_N \circ B_{N-1} \circ \cdots \circ B_1(\boldsymbol{x}), \tag{4.1}$$

$$B_n(\boldsymbol{x}) = F_n \circ A_n(\boldsymbol{x}), \tag{4.2}$$

where $\boldsymbol{x} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_L]^\top \in \mathbb{R}^{L \times d}$, L is the length of the sequence and d is the dimension of the word embedding. $A_n(\cdot)$ and $F_n(\cdot)$ are the self-attentive and feed-forward layer in the n-th block $B_n(\cdot)$, respectively.

Each row of $A_1(\boldsymbol{x})$ can be regarded as a weighted sum of the value matrix $\boldsymbol{V} \in \mathbb{R}^{L \times d}$, with the weights determined by similarity scores between the key matrix $\boldsymbol{K} \in \mathbb{R}^{L \times d}$ and query matrix $\boldsymbol{Q} \in \mathbb{R}^{L \times d}$ as follows:

$$
\begin{aligned}
A_1(\boldsymbol{x}) &= \texttt{Softmax}\left(\frac{\boldsymbol{Q}\boldsymbol{K}^\top}{\sqrt{d}}\right)\boldsymbol{V}, \\
\boldsymbol{Q} &= [\boldsymbol{q}_1, \boldsymbol{q}_2, ..., \boldsymbol{q}_L]^\top, \qquad \boldsymbol{q}_i = \boldsymbol{W}_q \boldsymbol{x}_i + \boldsymbol{b}_q, \\
\boldsymbol{K} &= [\boldsymbol{k}_1, \boldsymbol{k}_2, ..., \boldsymbol{k}_L]^\top, \qquad \boldsymbol{k}_i = \boldsymbol{W}_k \boldsymbol{x}_i + \boldsymbol{b}_k, \\
\boldsymbol{V} &= [\boldsymbol{v}_1, \boldsymbol{v}_2, ..., \boldsymbol{v}_L]^\top, \qquad \boldsymbol{v}_i = \boldsymbol{W}_v \boldsymbol{x}_i + \boldsymbol{b}_v,
\end{aligned}
\tag{4.3}
$$

$\boldsymbol{W}_{q/k/v}$ and $\boldsymbol{b}_{q/k/v}$ are the weight and bias parameters introduced in the self-attentive function $A_1(\cdot)$. The output of the feed-forward function $F_1(\cdot)$ used in the Transformer is also a matrix with L rows. In particular, the i-th row is obtained as follows.

$$\text{the } i\text{-th row of } F_1(\boldsymbol{x}) = \boldsymbol{W}_2 \sigma(\boldsymbol{W}_1 \boldsymbol{x}_i + \boldsymbol{b}_1) + \boldsymbol{b}_2, \tag{4.4}$$

where $\boldsymbol{W}_{1/2}$ and $\boldsymbol{b}_{1/2}$ are the weights and biases of linear transforms, and $\sigma(\cdot)$ is the activation function. It is not hard to see from (4.3) and (4.4) that both $A_1(\cdot)$ and $F_1(\cdot)$ are permutation equivalent. Thus, we can conclude that the entire function defined in (4.1) is also permutation equivalent, i.e., $\Pi \times \text{Encode}(\boldsymbol{x}) = \text{Encode}(\Pi \times \boldsymbol{x})$ for any $L \times L$ permutation matrix Π. This permutation equivalence property restricts the Transformer without position information from modeling sequences where the ordering of elements matters.
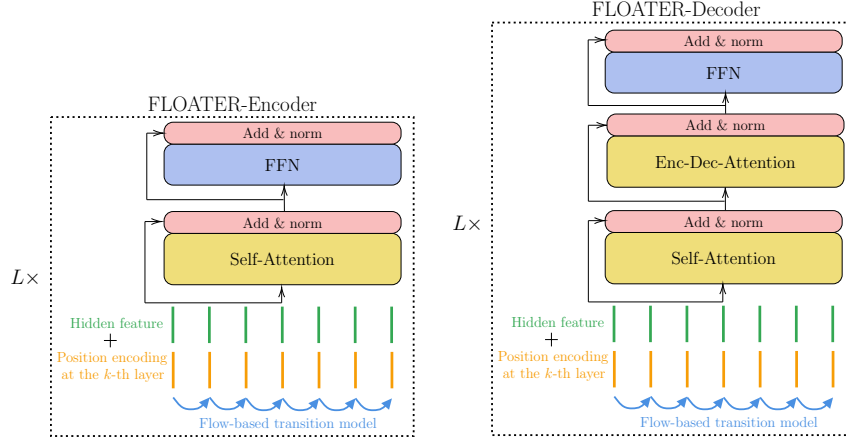
Figure 4.1: The architecture of our model (FLOATER). The main differences between FLOATER and the original Transformer model are: 1) the position representation is integrated into each block in the hierarchy (there are N blocks in total); and 2) there is a dynamical model (see (4.8)) that generates position encoding vectors for each block. The dynamics are solved with a black-box ODE solver detailed in the supplementary material.

Position encoding in transformer

As mentioned in Section 4.1.1, there are many attempts to inject position information in self-attentive components. Most of them can be described in the following form:

$$B_n(\boldsymbol{x}) = F_n \circ A_n \circ \Phi_n(\boldsymbol{x}), \ n \in \{1, ..., N\}, \tag{4.5}$$

where $\Phi_n(\boldsymbol{x})$ is a position encoding function.

[VSP17] propose to keep $\Phi_n(\boldsymbol{x}) = \boldsymbol{x}, \forall n \geq 2$ and inject position information only at the input block with a family of pre-defined sinusoidal functions: $\Phi_1(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(1)}$, where $\boldsymbol{p}^{(1)} = [\boldsymbol{p}_1^{(1)}, \boldsymbol{p}_2^{(1)}, ..., \boldsymbol{p}_L^{(1)}]$ is a position embedding matrix with the i-th row corresponding to the i-th position in the input sequence. In particular, the j-th dimension of the i-th row is

defined as follows.

$$
\boldsymbol{p}_i^{(1)}[j] = \begin{cases} \sin(i \cdot c^{\frac{j}{d}}) & \text{if } j \text{ is even,} \\ \cos(i \cdot c^{\frac{j-1}{d}}) & \text{if } j \text{ is odd,} \end{cases} \tag{4.6}
$$

where $c = 10^{-4}$. [DGV18] and [LCG19] observe better performance by further injecting the position information at each block, i.e., $\Phi_n(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(n)}$ as follows:

$$
\boldsymbol{p}_i^{(n)}[j] = \begin{cases} \sin(i \cdot c^{\frac{j}{d}}) + \sin(n \cdot c^{\frac{j}{d}}) & \text{if } j \text{ is even,} \\ \cos(i \cdot c^{\frac{j-1}{d}}) + \cos(n \cdot c^{\frac{j-1}{d}}) & \text{if } j \text{ is odd.} \end{cases} \tag{4.7}
$$

Note that for the above two approaches, position encoding functions $\Phi_n(\cdot)$ are fixed for all the applications. Although no additional parameters are introduced in the model, both approaches are inductive and can handle input sequences of variable length.

Many successful variants of pre-trained Transformer models, such as BERT [DCL18a] and RoBERTa [LOG19], include the entire embedding matrix $\boldsymbol{p}^{(1)} \in \mathbb{R}^{L \times d}$ in $\Phi_1(\boldsymbol{x})$ as training parameters. As the number of training parameters needs to be fixed, the maximum length of a sequence, $L_{\max}$, is required to be determined before the training. Although it lacks the inductive property, this data-driven approach is found to be effective for many NLP tasks. Note that, unlike the fixed sinusoidal position encoding, there is no attempt to inject a learnable position embedding matrix at each block for Transformer due to a large number of additional parameters ($NL_{\max}d$).

4.1.3   FLOATER: our proposed position encoder

We introduce our method in three steps. In the first step, we only look at one Transformer block, and describe how to learn the position representation driven by a dynamical system; in the second step, we show how to save parameters if we add position signals to every layer; lastly, we slightly change the architecture to save trainable parameters further and make FLOATER "compatible" with the original Transformer [VSP17]. The compatibility means

our model is a strict superset of the vanilla Transformer so that it can be initialized from the Transformer.

Position Encoding with Dynamical Systems

Position representations in Transformer models are a sequence of vectors $\{\boldsymbol{p}_i \in \mathbb{R}^d : i = 1, ..., L\}$ to be added to the sequence of the input representations $\{\boldsymbol{x}_i : i = 1, ..., L\}$. Existing position encoding approaches either apply a fixed sinusoidal function to obtain $\{\boldsymbol{p}_i\}$, or include them as uncorrelated learnable parameters. Both of them fail to capture the dependency or dynamics among these position representations $\{\boldsymbol{p}_i\}$. We propose to use a dynamical system to model these position representations; that is, there is a "latent force" denoted by $\boldsymbol{h}_i$ that drives the changes from $\boldsymbol{p}_i$ to $\boldsymbol{p}_{i+1}$. To encourage smoothness, we consider $\boldsymbol{p}(t) : \mathbb{R}_+ \mapsto \mathbb{R}^d$ as the continuous version of the discrete sequence $\{\boldsymbol{p}_i\}$. In particular, our proposed continuous dynamical system is characterized as follows:

$$\boldsymbol{p}(t) = \boldsymbol{p}(s) + \int_s^t \boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h) \, \mathrm{d}\tau, \ 0 \le s \le t < \infty, \tag{4.8}$$

together with an initial vector $\boldsymbol{p}(0)$, where $\boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h)$ is a neural network parameterized by $\boldsymbol{\theta}_h$ and takes the previous state $(\tau, \boldsymbol{p}(\tau))$. Notice that the domain of $\boldsymbol{p}(\cdot)$ is $\mathbb{R}_+$. The position sequence $\{\boldsymbol{p}_i\}$ can be obtained by taking $\boldsymbol{p}(\cdot)$ on a series of points $\{t_i : 0 \le t_1 < \cdots \le t_L\}$: $\boldsymbol{p}_i = \boldsymbol{p}(t_i)$. One simple strategy is to set $t_i = i \cdot \Delta t$ so that the points are equidistant, where Δ is a hyperparameter (e.g., $\Delta = 0.1$). With this strategy, we are implicitly assuming the position signals evolve steadily as we go through each token in a sentence. In general, $\{t_i\}$ can be any monotonically increasing series, which allows us to extend our work to more applications where the elements in the sequence are not always observed with the same interval. More discussions about the applicability for this general setting is included in the Supplementary material. For the NLP applications discussed here, we choose $t_i = i \cdot \Delta t$.

Eq. (4.8) is equivalent to an ODE problem $\frac{\mathrm{d}\boldsymbol{p}(t)}{\mathrm{d}t} = \boldsymbol{h}(t, \boldsymbol{p}(t); \boldsymbol{\theta}_h)$, which is guaranteed to have a unique solution under mild conditions [TP85]. We follow the efficient approach by

[CRB18] to calculate the gradients of $\boldsymbol{\theta}_h$ with respect to the overall training loss, which allows us to include this parameterized dynamical position encoder into the end-to-end training of Transformer models. More details can be found in the Supplementary material.

Our dynamical system (4.8) is quite flexible to admit the standard sinusoidal position encoding (4.6) as a special case:

$$
\boldsymbol{p}_{i+1}[j] - \boldsymbol{p}_i[j]
$$

$$
= \begin{cases} \sin\left((i+1) \cdot c^{\frac{j}{d}}\right) - \sin\left(i \cdot c^{\frac{j}{d}}\right) & \text{if } j \text{ is even} \\ \cos\left((i+1) \cdot c^{\frac{j-1}{d}}\right) - \cos\left(i \cdot c^{\frac{j-1}{d}}\right) & \text{if } j \text{ is odd} \end{cases} \tag{4.9}
$$

$$
= \begin{cases} \int_i^{i+1} c^{-\frac{j}{d}} \cos(\tau \cdot c^{\frac{j}{d}}) \, \mathrm{d}\tau & \text{if } j \text{ is even} \\ \int_i^{i+1} -c^{-\frac{j-1}{d}} \sin(\tau \cdot c^{\frac{j-1}{d}}) \, \mathrm{d}\tau & \text{if } j \text{ is odd,} \end{cases}
$$

This indicates that for simple sinusoidal encoding, there exists a dynamical system $\boldsymbol{h}(\cdot)$ which is also sinusoidal function.

Parameter Sharing among Blocks

As mentioned in Section 4.1.2, injecting position information to each block for Transformer leads to better performance [DGV18, LCG19] in some language understanding tasks. Our proposed position encoder FLOATER (4.8) can also be injected into each block. The idea is illustrated in Figure 4.1. Typically there are 6 blocks in sequence-to-sequence Transformer and 12 or 24 blocks in BERT. We add a superscript (n) to denote dynamics at n-th block:

$$
\boldsymbol{p}^{(n)}(t) = \boldsymbol{p}^{(n)}(s) + \int_s^t \boldsymbol{h}^{(n)}(\tau, \boldsymbol{p}^{(n)}(\tau); \boldsymbol{\theta}_h^{(n)}) \, \mathrm{d}\tau.
$$

As we can imagine, having N different dynamical models $\boldsymbol{h}^{(n)}(\cdot; \boldsymbol{\theta}_h^{(n)})$ for each block can introduce too many parameters and cause significant training overhead. Instead, we address this issue by sharing parameters across all the blocks, namely

$$
\boldsymbol{\theta}_h^{(1)} = \boldsymbol{\theta}_h^{(2)} = \cdots = \boldsymbol{\theta}_h^{(N)}. \tag{4.10}
$$

Note that (4.10) does not imply that all the $\boldsymbol{p}_t^{(n)}$ are the same, as we will assign different initial values for each block, that is $\boldsymbol{p}^{(n_1)}(0) \neq \boldsymbol{p}^{(n_2)}(0)$ for $n_1 \neq n_2$.

	Transformer-Base		*Transformer-Large*	
	En-De	En-Fr	En-De	En-Fr
Position encoders at all blocks				
FLOATER	**28.6**	**41.6**	**29.2**	**42.7**
Pre-defined Sinusoidal Position Encoder	28.2	40.6	28.4	42.0
Fixed-length Position Embedding	26.9	40.9	28.3	42.0
Position encoder only at input block				
FLOATER	28.3	41.1	29.1	42.4
Pre-defined Sinusoidal Position Encoder	27.9	40.4	28.4	41.8
Fixed-length Position Embedding	27.8	40.9	28.5	42.4

Table 4.2: Experimental results of various position encoders on the machine translation task.

Compatibility and Warm-start Training

In this section, we change the way to add position encoding so that our FLOATER can be directly initialized from Transformer. As an example, we use the standard Transformer model, which has a fixed sinusoidal encoding at the input block and no position encoding at deeper levels. Note that this technique can be extended to other variants of Transformers with different position encoding methods, such as embedding matrix. We first examine the standard Transformer model, the query matrix $\boldsymbol{Q}^{(n)}$ at block-n is

$$\widetilde{\boldsymbol{q}}_i^{(n)} = \boldsymbol{W}_q^{(n)}\left(\boldsymbol{x}_i + \widetilde{\boldsymbol{p}}_i^{(n)}\right) + \boldsymbol{b}_q^{(n)}, \tag{4.11}$$

116

where $\boldsymbol{W}_q^{(n)}$ and $\boldsymbol{b}_q^{(n)}$ are parameters in A_n (4.3); $\widetilde{\boldsymbol{p}}^{(n)}$ is the sinusoidal encoding; $\widetilde{\boldsymbol{q}}_i^{(n)}$ is the i-th row of $\boldsymbol{Q}^{(n)}$. Here we add a tilde sign to indicate the sinusoidal vectors. Formulas for $\widetilde{\boldsymbol{k}}_i^{(n)}$ and $\widetilde{\boldsymbol{v}}_i^{(n)}$ have a very similar form and are omitted for brevity.

Now we consider the case of FLOATER, where new position encodings $\boldsymbol{p}_i$ are added

$$
\begin{aligned}
\boldsymbol{q}_i^{(n)} &= \boldsymbol{W}_q^{(n)}\Big(\boldsymbol{x}_i + \boldsymbol{p}_i\Big) + \boldsymbol{b}_q^{(n)} \\
&= \underbrace{\boldsymbol{W}_q^{(n)}(\boldsymbol{x}_i + \widetilde{\boldsymbol{p}}_i^{(n)}) + \boldsymbol{b}_q^{(n)}}_{\text{Eq. (4.11)}} + \underbrace{\boldsymbol{W}_q^{(n)}(\boldsymbol{p}_i - \widetilde{\boldsymbol{p}}_i^{(n)})}_{\text{Extra bias term depends on } i} \\
&= \widetilde{\boldsymbol{q}}_i^{(n)} + \boldsymbol{b}_{q,i}^{(n)}.
\end{aligned}
\tag{4.12}
$$

It is easy to see that the changing the position embedding from $\{\widetilde{\boldsymbol{p}}_i^{(n)}\}$ to $\{\boldsymbol{p}_i^{(n)}\}$ is equivalent to adding a position-aware bias vector $\boldsymbol{b}_{q,i}^{(n)}$ into each self-attentive layers $\{A_n(\cdot)\}$. As a result, we can instead apply (4.8) to model the dynamics of $\boldsymbol{b}_q^{(n)}$. In particular, we have the following dynamical system:

$$
\boldsymbol{b}_q^{(n)}(t) = \boldsymbol{b}_q^{(n)}(0) + \int_0^t \boldsymbol{h}^{(n)}(\tau, \boldsymbol{b}_q^{(n)}(\tau); \boldsymbol{\theta}_h)\, \mathrm{d}\tau.
\tag{4.13}
$$

After that, we set $\boldsymbol{b}_{q,i}^{(n)} = \boldsymbol{b}_q^{(n)}(i \cdot \Delta t)$. We can see that if $\boldsymbol{h}(\cdot) = 0$ and $\boldsymbol{b}_q^{(n)}(0) = 0$, then $\boldsymbol{b}_q^{(n)} \equiv 0$. This implies (4.12) degenerates to (4.11). Note that (4.13) has the same form as (4.8), except that we are now modeling the bias terms $\boldsymbol{b}_{q,i}$ in (4.3). We will apply the same technique to $\boldsymbol{K}$ and $\boldsymbol{V}$.

To summarize, our model has a tight connection to the original Transformer: if we set all dynamical models to zero, which means $\boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h) \equiv 0$, then our FLOATER model will be equivalent to the original Transformer with the sinusoidal encoding. The same trick also works for Transformer with position embedding such as BERT [DCL18a].

We strive to make our model compatible with the original Transformer due to the following reasons. First of all, the original Transformer is faster to train as it does not contain any recurrent computation; this is in contrast to our dynamical model (4.8), where the next position $\boldsymbol{p}_{i+1}$ depends on the previous one $\boldsymbol{p}_i$. By leveraging the compatibility of model

architecture, we can directly initialize FLOATER model from a pre-trained Transformer model checkpoint and then fine-tune for the downstream task for a few more epochs. By doing so, we enjoy all the benefits of our FLOATER model but still maintain an acceptable training budget. Likewise, for models such as BERT or Transformer-XL, we already have well-organized checkpoints out of the box for downstream tasks. These models are costly to train from scratch, and since our goal is to examine whether our proposed position representation method can improve over the original one, we decided to copy the weights layer by layer for attention as well as FFN layers, and randomly initialize the dynamical model $\boldsymbol{h}(\tau, \boldsymbol{p}(\tau); \boldsymbol{\theta}_h)$.

4.1.4 Experimental results

In this section, we perform experiments to see if FLOATER can improve over the existing position encoding approaches for a given Transformer model on various NLP tasks. Thus, all the metrics reported here are computed from a single (not ensemble) Transformer model over each evaluation NLP task. Albeit lower than top scores on the leaderboard, these metrics are able to reveal more clear signal to judge the effectiveness of the proposed position encoder.

All our codes to perform experiments here are based on the Transformer implementations in the `fairseq` [OEB19] package. Implementation details can be found in the Supplementary material. Our experimental codes will be made publicly available.

Table 4.3: Experimental results on GLUE benchmark

Model	Single Sentence		Similarity and Paraphrase			Natural Language Inference		
	CoLA	SST-2	MRPC	QQP	STS-B	MNLI	QNLI	RTE
Base model								
RoBERTa	**63.6**	94.8	88.2	**91.9**	91.2	87.6	92.8	78.7
FLOATER	63.4	**95.1**	**89.0**	91.7	**91.5**	**87.7**	**93.1**	**80.5**
Large model								
RoBERTa	68.0	96.4	90.9	**92.2**	92.4	90.2	94.7	86.6
FLOATER	**69.0**	**96.7**	**91.4**	**92.2**	**92.5**	**90.4**	**94.8**	**87.0**

Table 4.4: Experiment results on RACE benchmark. "Middle" means middle school level English exams, "High" means high school exams. Other details can be found in [LXL17].

Model	Accuracy	Middle	High
Single model on test, large model			
RoBERTa	82.8	86.5	81.3
FLOATER	**83.3**	**87.1**	**81.7**

Neural Machine Translation

Neural Machine Translation (NMT) is the first application that demonstrates the superiority of a sequence-to-sequence Transformer model over conventional recurrent sequence models. We include the following three additive position encoders: $\Phi^{(n)}(\boldsymbol{x}) = \boldsymbol{x} + \boldsymbol{p}^{(n)}$.

- **Data-driven FLOATER**: $\boldsymbol{p}^{(n)}$ is generated by our proposed continuous dynamical models with data-driven parameters described in (4.8).
- **Pre-defined sinusoidal position encoder**: $\boldsymbol{p}^{(n)}$ is constructed by a pre-defined function described in (4.7), which is proposed by [VSP17] and extended by [DGV18].
- **Length-fixed position embedding**: $\boldsymbol{p}^{(n)}$ is included as learnable training parameters. This is first introduced by [VSP17] and adopted in many variants of Transformer [DCL18a, LOG19].

To better demonstrate the parameter efficiency brought by FLOATER, for each above encoder, we also include two experimental settings: position encoder at all blocks or only at the input block (i.e., $\boldsymbol{p}^{(n)} = 0, \forall n \geq 2$).

In Table 4.2, we present the BLEU scores on WMT14 Ee-De and En-Fr datasets with both *Transformer-base* and *Transformer-large* models described in [VSP17]. Among all the data/model combinations, our proposed FLOATER at all blocks outperforms two other position encoders.

On the other hand, we also observe that adding position encoders at all blocks yields better performance than only at the input block. While there is an exception in the fixed-length position embedding approach. We suspect that this phenomenon is due to over-fitting cased by $L_{\max}dN$ learnable parameters introduced by this approach. In contrast, our proposed FLOATER is parameter efficient (more discussions in Section 4.1.4), so the performance can be improved by injecting the position encoder at all the blocks of Transformer without much additional overhead.

Language Understanding and Question Answering

Table 4.5: Experiment results on SQuAD benchmark. All results are obtained from RoBERTa-large model.

Model	SQuAD 1.1		SQuAD 2.0	
	EM	F1	EM	F1
Single models on dev, w/o data augmentation				
RoBERTa	88.9	94.6	86.5	89.4
FLOATER	88.9	94.6	**86.6**	**89.5**

Pretrained Transformer models such as BERT and RoBERTa have become the key to achieving the state-of-the-art performance for various language understanding and question answering tasks. In this section, we want to evaluate the effectiveness of the proposed FLOATER on these tasks. In particular, we focus on three language understanding benchmark sets, GLUE [WSM18], RACE [LXL17] and SQuAD [RZL16]. As mentioned in Section 4.1.3, FLOATER is carefully designed to be compatible with the existing Transformer models. Thus, we can utilize pretrained Transformer models to warm-start a FLOATER model easily to be used to finetune on these NLP tasks. We download the *same* pre-trained RoBERTa model from the official repository as our pretrained Transformer model for all NLP tasks

discussed in this section.

GLUE Benchmark. This benchmark is commonly used to evaluate the language understanding skills of NLP models. Experimental results in Table 4.3 show that our FLOATER model outperforms RoBERTa in most datasets, even though the only difference is the choice of positional encoding.

RACE benchmark Similar to the GLUE benchmark, the RACE benchmark is another widely used test suit for language understanding. Compared with GLUE, each item in RACE contains a significantly longer context, which we believe requires more important to grasp the accurate position information. Like in GLUE benchmark, we finetune the model from the same pretrained RoBERTa checkpoint. We keep the hyperparameters, such as batch size and learning rate, to also be the same. Table 4.4 shows the experimental results. We again see consistent improvement of FLOATER across all subtasks.
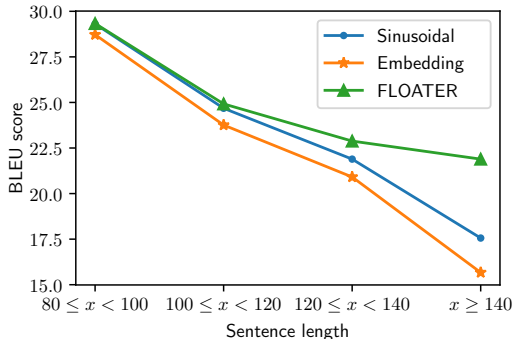
Figure 4.2: Comparing BLEU scores of different encoding methods.

SQuAD benchmark SQuAD benchmark [RZL16, RJL18] is another challenging task to evaluate the question answering skills of NLP models. In this dataset, each item contains a lengthy paragraph containing facts and several questions related to the paragraph. The model needs to predict the range of characters that answer the questions. In SQuAD-v2, the problem becomes more challenging that the questions might be unanswerable by the context. We follow the same data processing script as BERT/RoBERTa for fair comparison;

121

Table 4.6: Performance comparison on WMT14 En-De data and Transformer-base architecture. Both BLUE scores and the number of trainable parameters inside each position encoder are included.

	BLEU ($\uparrow$)	#Parameters ($\downarrow$)
FLOATER	**28.57**	526.3K
1-layer RNN + scalar	27.99	263.2K
2-layer RNN + scalar	28.16	526.3K
1-layer RNN + vector	27.99	1,050.0K
1-Layer LSTM + scalar	28.17	1050.0K
1-Layer GRU + scalar	28.11	789.5K
2-Layer LSTM + scalar	28.10	2100.0K
2-Layer GRU + scalar	26.21	1580.0K

more details about the training process are described in the Supplementary material. The experiment results are presented in Table 4.5. As we can see, the FLOATER model beats the baseline RoBERTa model consistently across most datasets. The improvement is significant, considering that both models are finetuned from the same pretrained checkpoint.

More Discussions and Analysis

How inductive is FLOATER? FLOATER is designed to be inductive by a data-driven dynamical model (4.8). To see how inductive FLOATER is when comparing to existing approaches, we design the following experiment. We first notice that in WMT14 En-De dataset, 98.6% of the training sentences are shorter than 80 tokens. Based on that, we make a new dataset called *En-De short to long* (or S2L for brevity): this dataset takes all the short sentences (< 80 tokens) as the training split and all the long sentences (≥ 80 tokens) as the

testing split. We further divide the testing split to four bins according to the source length fallen in $[80, 100)$, $[100, 120)$, $[120, 140)$, $[140, +\infty)$. BLEU scores are calculated in each bin, and the results are presented in Figure 4.2.

Our FLOATER model performs particularly well on long sentences, even though only short sentences are seen by the model during training. This empirical observation supports our conjecture that FLOATER model is inductive: the dynamics learned from shorter sequences can be appropriately generalized to longer sequences.

Is RNN a good alternative to model the dynamics?   Recurrent neural network (RNN) is commonly used to perform sequential modeling. RNN and our continuous dynamical model (4.8) indeed share some commonality. Computing the value at the i-th step relies on the results at the $(i-1)$-st step. Further, they all contain trainable parameters, allowing them to adapt to each particular task. Lastly, they can be extrapolated to any length as needed. To see if RNN works equally well, we model the sequence $\{\boldsymbol{p}_i\}_{i \in \{1,2,\dots\}}$ with RNN models:

$$\boldsymbol{p}_{i+1} = \mathrm{RNN}(\boldsymbol{z}_i, \boldsymbol{p}_i), \tag{4.14}$$

where $\boldsymbol{z}_i \in \mathbb{R}^{d_{\mathrm{in}}}$ is the input to the RNN model at index i. Recall in RNN language models, $\boldsymbol{z}_i$ is the word embedding or hidden feature of the i-th token. In our case, since we apply RNN to learn the encodings as opposed to hidden features, sensible inputs can be scalar value i or vectorized value Vectorize(i) by sinusoidal encoding. We tried both choices on WMT14 En-De data and found that vectorized value generally works better, though not as good as our FLOATER model. Detailed results can be found in Table 4.6.

What does each position encoding look like?   To better understand how different position encodings affect the sequence modeling, in Figure 4.3, we visualize the position embedding matrix $\boldsymbol{p}$ obtained from four different position encoding approaches for the Transformer-base backbone on WMT14 En-De dataset. We can see that sinusoidal encoding

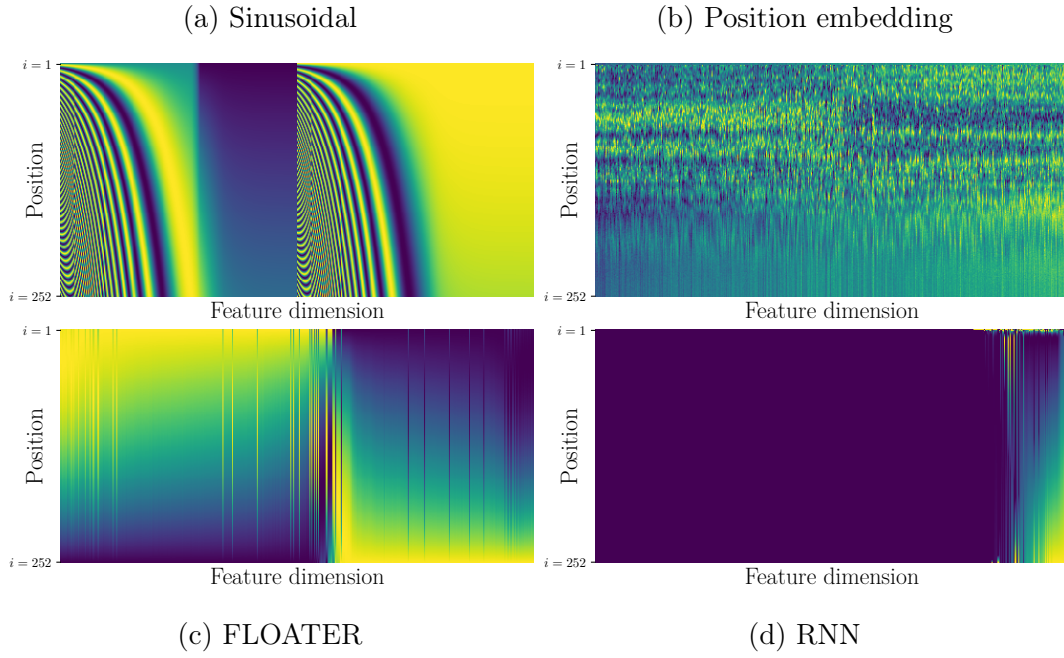(a) Sinusoidal   (b) Position embedding

(c) FLOATER   (d) RNN

Figure 4.3: Visualizing the four different position methods. All models are trained using the Transformer-base architecture and En-De dataset. For better visualization, dimension indices are permuted in Figure 4.3b-4.3d.

(4.3a) is the most structural, while position embedding (4.3b) is quite chaotic. Our FLOATER model learns position representation completely from data, but still exhibits some regularities (4.3c). Finally, the RNN model (4.3d) fails to extract sufficient positional information, probably due to the vanishing gradient problem. Another finding is that by looking at (4.3b), we observe that the vectors are nearly constant among different large positions (near the bottom of Figure 4.3b, we see patterns of vertical lines with the same color). This phenomenon is due to long sentences in the dataset being scarce, and so the positional information carried by lower indices cannot be extrapolated to higher indices. On the contrary, the dynamical model proposed here enjoys the best of both worlds – it is adaptive to dataset distribution, and it is inductive to handle sequences with lengths longer than the training split.

Remarks on Training and Testing Efficiency

It is not surprising that during the training time, our flow-based method adds a non-negligible time and memory overhead; this is because solving the Neural ODE precisely involves ∼100 times forward and backward propagations of the *flow model*. Even though we deliberately designed a small flow model (consisting of only two FFN and one nonlinearity layers), stacking them together still increases training time substantially. To make it possible to train big models, we use the following optimizations:

- Initialize with pretrained models that do not contain flow-based dynamics, as discussed in Section 4.1.3.
- From (4.8), we know that if $h(\cdot)$ is close to zero, then the position information diminishes (derived in appendix). In this way, our model degenerates to the original Transformer. Inspired by this property, we can initialize the FLOATER with smaller weights. Combining with the previous trick, we obtain an informed initialization that incurs lower training loss at the beginning.
- We observed that weights in $h(\cdot)$ are more stable and easy to train. Thus, we can separate the weights of $h(\cdot)$ from the remaining parts of the Transformer model. Concretely, we can

1) cache the positional bias vectors for some iterations without re-computing, 2) update the weights of flow models less frequently than other parts of the Transformer, and 3) update the flow models with a larger learning rate to accelerate convergence.

- For the RoBERTa model, we adopt an even more straightforward strategy: we first download a pretrained RoBERTa model, plug in some flow-based encoding layers, and re-train the encoding layers on WikiText-103 dataset for one epoch. When finetuning on GLUE datasets, we can choose to freeze the encoding layers.

Combining those tricks, we successfully train our proposed models with only 20-30% overhead compared to traditional models, and virtually no overhead when finetuning RoBERTa model on GLUE benchmarks. Moreover, there is no overhead during the inference stage if we store the pre-calculated positional bias vectors in the checkpoints.

Part II

Robust Uncertainty Estimation

CHAPTER 5

Fast Ensemble Method for Robust Uncertainty Estimation

5.1   Motivation

As we deploy machine learning models to real-life production systems, an obstacle to many practitioners is to what extent we can trust the prediction results generated from millions or billions of parameters. To solve this problem, we need another layer of abstraction that takes in the model and data information and outputs the confidence interval (for regression task) or the expected error rate (for classification task). Ideally, such mechanism needs to: 1) handle both expected input (called *in-distribution* data) or unexpected input (called *out-of-distribution data*), 2) compared with the original model training time, operate efficiently enough, so that little overhead is posted, and 3) independent of modeling details, can work even for black-box models.

The algorithm that calibrates the model confidence to match the prediction accuracy is formally called *confidence calibration*. For instance, the original machine learning model may report 99.8%-confidence about its prediction, yet the actual accuracy is only 90% – far below the confidence. This disparity requires us to calibrate the confidence estimation from 99.8% to 90%. A related concept is called *uncertainty estimation*; it is meant to generate the confidence interval for model predictions, so we expect the true value falls into this interval with high probability. Predictive uncertainty also alarms the human-in-the-loop (HITL) machine learning paradigm, signaling that human intervention is needed once it raises above

a threshold.

This work is motivated by the importance of uncertainty estimation in biomedical applications. In the past few years, many machine learning models have been deployed in biomedical imaging, such as the cell type identification [CYA18], label-free organelle labeling [OSM18], histology virtual staining [RWW19], and noninvasive cell phenotyping [ILL21]. Among these applications, many involve image-to-image translation models. Unfortunately, uncertainty estimation in image translation has been under-explored – there exists no promising benchmarks nor systematical studies on how existing uncertainty estimation methods perform on image generation tasks. One of the main obstacles for this problem is the lack of benchmark and evaluation method – it is difficult to quantitatively evaluate uncertainty estimation methods for image generation. To evaluate an uncertainty estimation method for classification, one can easily choose a leave-one-out set, usually a new class that is not appeared in training, and calculate the disagreement between uncertainty estimation and prediction error rate. However, this cannot be easily done in image-to-image translation. As the output space is high dimensional, uncertainty estimation cannot be easily calculated in a per-sample manner, and there could be nonuniform uncertainty for different patchs of the image. For example, it may be the case that in the same image, some cells have been seen in training but others are not, leading to nonuniform uncertainty within an image.

We develop the first systematic benchmark and evaluation method for uncertainty evaluation in image-to-image translation. To build this evaluation benchmark, we collect a series of phase contrast (transmitted light microscopy) and immunofluorescent images of mesenchyaml stromal cells [ILL21] and prostatic cancer cells (LNCaP). With these microscopy data, we measure the quality of uncertainty estimation through out-of-distribution detection and distribution shift assessment. Equipped with this new benchmark, we evaluate six representative uncertainty estimation methods, including naive ensemble, snapshot ensemble, batch ensemble, SGLD, variational inference, and Monte-Carlo dropout. Our experimental results suggest that the naive ensemble consistently outperforms other more complicated algorithms

on our biomedical image-to-image translation benchmark.

While the native ensemble approach provides an accurate estimate on the prediction uncertainty, a major weakness of such an approach is its computational overhead for building independent models. So it is critical to ask *can we find a <u>near zero-cost</u> method for uncertainty estimation?* To this end, we present our solution called FastEnsemble. This solution is inspired by the recent findings on the connectivity of local minimum of deep neural networks [GIP18]. In particular, our goal is not to find the low-loss path connecting two distinct local minima but to find some independent low-loss paths starting from an initial solution. We search the path efficiently so that each path only takes 3% to 5% of the time to train one model from scratch. In total, gathering an ensemble of six models requires ∼20% extra computation. Our contributions can be summarized as follows:

- We develop a new benchmark to evaluate uncertainty estimation algorithms on biomedical image generation applications. Based on that, we try to find the best solution out of six popular uncertainty estimators. To our knowledge, this is the first to study uncertainty quantification on image generation tasks systematically.

- We propose a new method that generates many independent ensemble models with a small overhead. Experimental results demonstrate that our approach can significantly speed up the running time without sacrificing the uncertainty estimation quality.

5.2   Related work

We have seen the active development of new efficient methods for confidence calibration and uncertainty quantification. Similar work can be roughly divided into two groups. The first group falls into the category of Bayesian learning. Here, we include the following approximate Bayesian methods:

Monte-Carlo dropout (MC-Dropout): [GG16] showed that the dropout layers ap-

plied before every weight layer is mathematically equivalent to the deep Gaussian process. The most significant benefit of this solution is simplicity, meaning that the existing neural networks with dropout layer before (de)convolution layer or fully connected layer are naturally becoming a Bayesian neural network.

Stochastic gradient Langevin dynamics (SGLD): [WT11, LCC16] proposed a way to transform stochastic gradient optimizer to imitate the Langevin dynamics. Similar to MC-Dropout, this method does not change the architecture of neural networks as long as the stochastic gradient can be computed efficiently. Like SGD optimizer, a damping step size ϵ_t is required to guarantee that the injected Gaussian noise eventually dominates the stochastic gradient noise so that the parameter trajectory converges to the true posterior. In practice, we follow the previous implementations to turn off the noise injection at the burn-in phase, then turn on the noise injection in the sampling phase.

Stochastic variational inference (SVI) [WJ08, BKM17]: This method approximate the posterior by maximizing the ELBO. Unlike MC-Dropout and SGLD, to apply SVI we need to double the number of parameters to learn both mean and standard deviation (assuming factorized Gaussian is used).

The other group we will include in the experiments is the ensemble methods. Specifically, the ones featuring low training overhead, detailed as follows

Snapshot ensemble [HLP17]: It generates different model parameters with cyclic cosine learning rate, a checkpoint is stored whenever the learning rate drops to the minimum. Although there are other ensemble methods by the cyclic learning rate, such as the piece-wise linear rate in [GIP18], we only experiment with snapshot ensemble here for brevity.

Batch ensemble [WTB20]: The more recent advancement is batch ensemble. This method generates less correlated models by learning a series of rank-1 vectors $\boldsymbol{v}_i \in \mathbb{R}^d$ and $\boldsymbol{u}_i \in \mathbb{R}^d$, which are later element-wise multiplied by the weights $w_i \leftarrow w \odot (\boldsymbol{v}_i \boldsymbol{u}_i^\top)$.

Finally, we would like to address the differences between our work and [OFR19]. In our

work, we intend to dive deeper into the biomedical imaging domain (both image2image and image classification), where uncertainty estimation is critical and out-of-distribution data is abundant. In contrast, [OFR19] studies classification problems exclusively, including image classification, text classification and Ads-click binary classification problems. Most of the datasets studied here are originated from real applications. To our knowledge, this is the first systematical study concerning image2image. Our study is unique because in-distribution data and out-of-distribution data coexist in the same image, so the uncertainty values are directly comparable.

5.3 A new uncertainty estimation benchmark for image generation task

The predictive uncertainty originates from a lack of training data (namely *epistemic uncertainty*) or the inherent randomness in the data generation model (*aleatory uncertainty*) [KG17, HW21, APH21]. In machine learning applications, uncertainty arises from unpredictable changes in the environment. For example, researchers may hand-pick the biomedical images in the training set uniformly, so low-quality images are cleaned up. At the same time, the model is deployed at hospitals owning different brands of microscopes in suboptimal working conditions, or there may be impurities of various shapes/components that are impossible to enumerate beforehand.

In this section, we introduce a new benchmark for evaluating uncertainty estimation methods on the image-to-image translation task. As image-to-image translation is crucial to many biomedical applications, our datasets primarily consist of microscopy cell images. Following [OFR19], we investigate how different models behave under two Out of Distribution (OOD) settings: the first context is the local perturbation, meaning that the whole image is in-distribution except for some small patches. This scenario frequently happens in biomedical experiments, where impurities contaminate the cell culture. The other context is called

132

global perturbation. The whole image distribution is drifted away from the original data generation distribution in the training set. It happens when the cells are cultured under different conditions (e.g., drug treatment, growth media changes, or image acquisition at different time points).

Out of distribution detection

The first benchmark is a collection of pairwise image to image translation tasks closely related to biomedical imaging. This work mainly used the same dataset that was tested and published in our previous study (Imboden etal). In addition, in the current study we acquired new LNCaP images for mimicking non-trivial image contaminations. To benchmark the out-of-distortion detection of our model, we tested three conditions: MSC clean (control), MSC-impurities (non-cellular objects), and MSC-LNCaP (cellular objects).

MSC-Clean (quality control): The dataset used for training purposes contains pairs of phase contrast and the respective fluorescence (IF) images of mesenchymal stromal cells (MSC). The cells were immunofluorescently stained for CD105, a surface marker, widely used to define MSC subpopulations. All images were acquired with an inverted microscope (Etaluma LS720, Lumaview 720/600-Series software) with a 20x phase contrast objective (Olympus, LCACHN 20XIPC). This is called a cleaned dataset as a quality control was performed where blurry or corrupt images were excluded.

MSC-Impurities: This dataset includes images of the same cell type (MSCs) and surface marker (CD105) as the cleaned dataset. To evaluate the impact of image impurities on the training accuracy, the MSC-Impurities dataset contains images of which 25% show artifacts. We included three different types of image artifacts: microscope slide impurities (e.g. scratches, bubbles, slide dust), fluorescent speckles and non-specific binding of the antibody.

MSC-LNCaP: This dataset is artificially created by mixing the images of MSC cells

(majority) with LNCaP cells (read patch boxes in Figure 5.3). MSC cells and LNCaP cells are visually different, but for non-expert humans it is non-intuitive to tell them apart. So we expect this dataset to be much harder than **MSC-Impurities**.

Among those datasets, **MSC-Clean** is the one to train the U-Net [RFB15] model to be experimented later. At this moment, the model hasn't encountered the OOD patches in the subsequent two datasets. After that, we apply the model on **MSC-Impurities** and **MSC-LNCaP** to collect the uncertainty of each pixel. Finally, we examine whether the model assigns high uncertainties inside the bounding boxes and low uncertainties outside the bounding boxes. To this end, we encapsulate this problem by the ranking problem. Specifically, we leverage two commonly used metrics in information retrieval, Precision@k and Recall@k, to compare different methods. Here we treat pixels inside bounding boxes as positive instances $\mathcal{S}_1$ (and vice versa); we then rank the pixels by the uncertainty values in descending order. The top-k highest uncertainty pixels $\mathcal{S}_2$ are selected. Then we have

$$\text{TP@}k = |\mathcal{S}_1 \cap \mathcal{S}_2|, \quad \text{Precision@}k = \frac{\text{TP@}k}{k}, \quad \text{Recall@}k = \frac{\text{TP@}k}{|\mathcal{S}_1|}, \tag{5.1}$$

here TP means number of true positives. We illustrate this idea in Figure 5.1.

Moreover, by changing k, we can plot the ROC curve to compare different methods visually. The experimental results are displayed in Figure 5.2. In this comparison, we include six popular uncertainty estimation methods, including the naive ensemble, snapshot ensemble, batch ensemble, SGLD, SVI, and MC-Dropout. Details of these methods can be found in related work.

From this figure, we can observe that the naive ensemble method outperforms all other methods, sometimes with a significant margin (**MSC-LNCaP**). We remark that this finding supports a similar conclusion in [OFR19], where the authors found that the simple ensemble method outperforms other Bayesian methods in image recognition datasets. Our experiment further indicates that existing fast ensemble methods (BatchEnsemble, Snapshot Ensemble) cannot close the gap concerning OOD robustness.
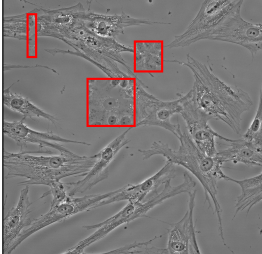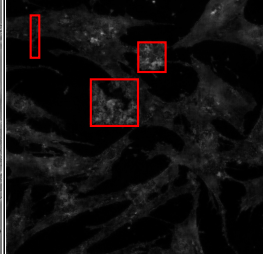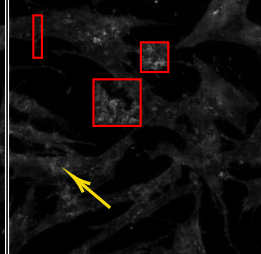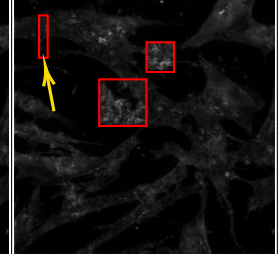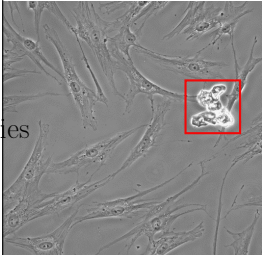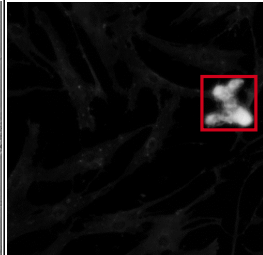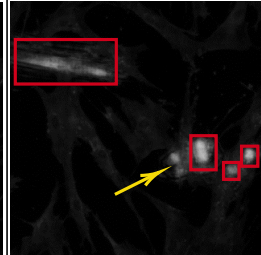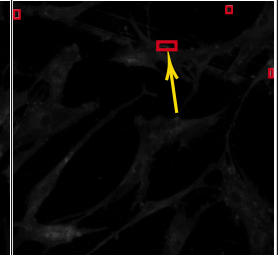
	Input with label	Prediction	Ex. of False Positive	Ex. of False Negative
MSC-LNCaP				
MSC-Impurities				

Figure 5.1: Image samples from the **MSC-LNCaP** and **MSC-Impurities** datasets and the corresponding uncertainty estimation generated by the ensemble method. The first row highlights bounding boxes (drawn to highlight the ground truth inaccessible to models) in an image from **MSC-LNCaP**. It is difficult even for humans to notice the out-of-distribution LNCaP cells surrounded by MSC cells without expertise. The second row is generated from **MSC-Impurities** data. This is an easier task because impurities usually are easily distinguishable from the cells.

Distribution shift assessment

In this experiment, we show that ensemble method is more robust even under large perturbations. Previous benchmark datasets are mostly in-distribution except for small patches labeled by bounding boxes, a more challenging case where the testing samples are different from the whole training set remains to be investigated. To evaluate different algorithms in this condition, we introduce a new dataset called **LNCaP-Density**.

LNCaP-Density: In contrast to **MSC-Clean** and **MSC-Impurities**, the images used for this dataset are of an LNCaP cell type. LNCaP cells are androgen-sensitive human prostate
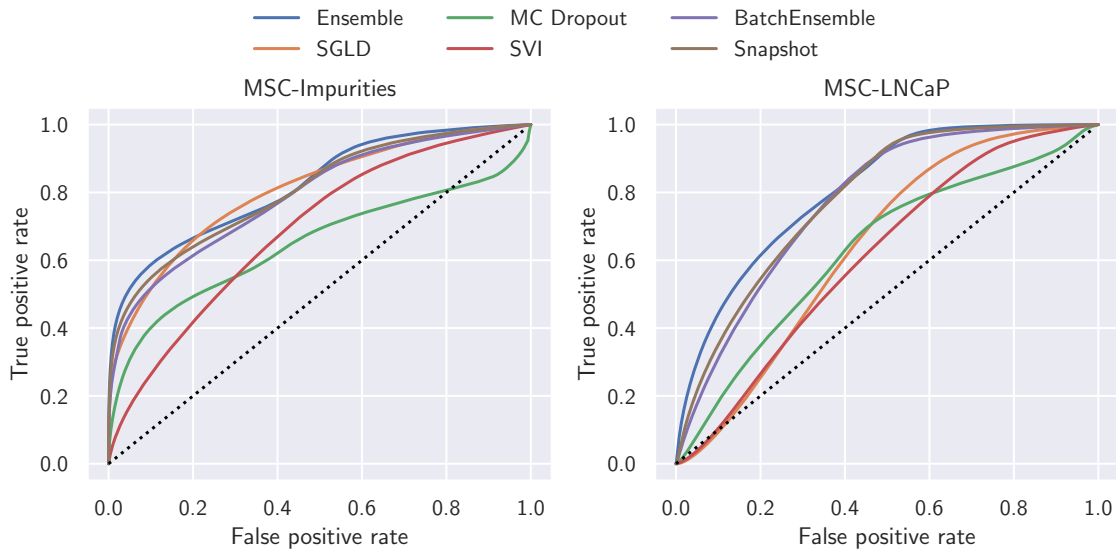
Figure 5.2: Comparison of some widely used ensemble methods and Bayesian inference algorithms. Notice the naive ensemble method performs similarly to batch ensemble or SGLD in **MSC-Impurities** data and significantly better in **MSC-LNCaP** data. In practice, we want to control the false positive rate to a small value, so we mainly look at the AUC when false positive ≤ 0.2.

adenocarcinoma cells. In this dataset, time-lapse phase contrast images of 12 different fields of view (FOVs) were acquired over a period of 72 hours. Cell density increases significantly over the time period due to cell division and growth. We did some manual sorting work to distribute all images into four subsets: namely VSparse ("very sparse"), Sparse, Dense, and VDense. Figure 5.3 gives some samples in each groups.

We train two models: model A is trained using the most sparsely populated cells (VSparse), and model B is trained with the most densely populated cells (VDense). After completing training, we run the predictions on all groups (VSparse, Sparse, Dense, VDense). The Pearson correlation between prediction and ground truth is calculated as the metric. We plot the histogram in Figure 5.4.

Similar to the previous local perturbation benchmark, from Figure 5.4, we can see naive
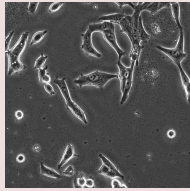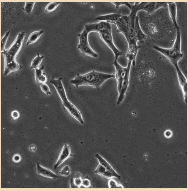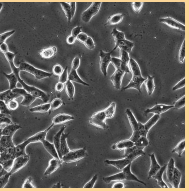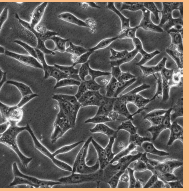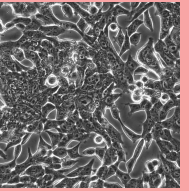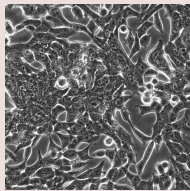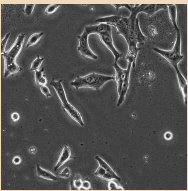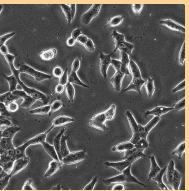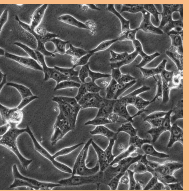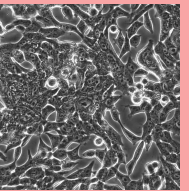
	Training	Testing			
	Sparse/Dense	Very sparse	Sparse	Dense	Very dense
Model A					
Model B					

Figure 5.3: Samples from the **LNCaP-Density** dataset and illustration of the distribution shift experiment. Model A is trained with the "very sparse" subset of **LNCaP-Density**, and Model B is trained with the "very dense" subset. Both models are then tested with all subsets of varying densities.

ensemble is still the best performing method in nearly every case. But the gap between batch ensemble / snapshot ensemble is small. Moreover, we generally find the ensemble-based methods more stable than Bayesian methods by comparing the error bar length. This finding aligns well again with [OFR19].

5.4 Accelerating ensemble method

In the previous section, we tested three Bayesian methods and three ensemble methods on two OOD benchmarks. Our investigation reveals that the most robust uncertainty estimator is the naive ensemble aggregation, despite the Bayesian methods being more theoretically principled.

include a problem statement We hypothesize that the power of the Bayesian method is restricted by choice of prior distributions and approximate inference. On the other hand, the
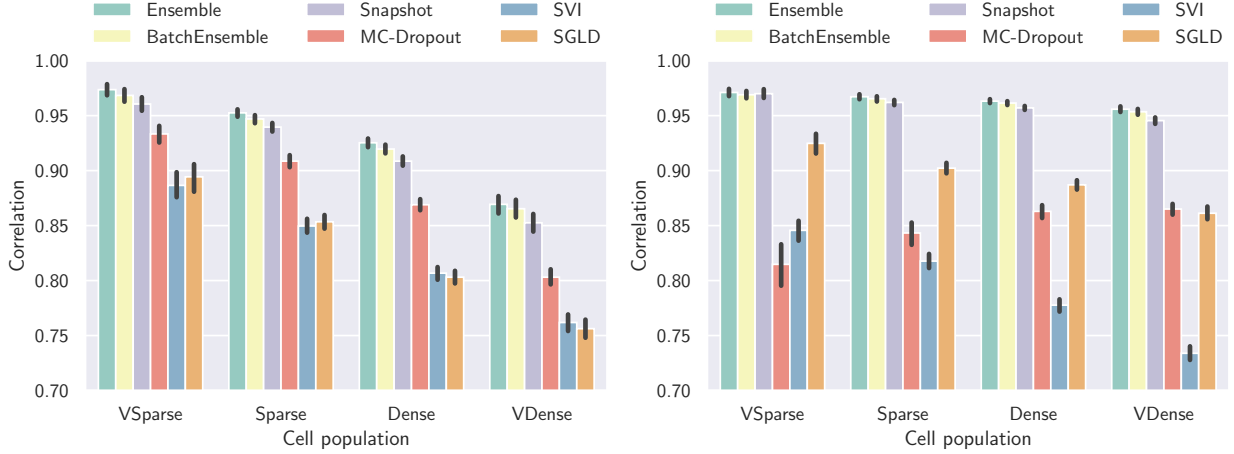
Figure 5.4: Comparing our method with other ensemble or Bayesian methods under a distribution shift setting. *Left*: Training on the "very sparse" subset and evaluation on each subset (Model A of Fig. 5.3). *Right*: Training on the "very dense" subset (Model B) and evaluation on each subset. The error bar is computed over all images. We can see the correlation drops more quickly for the "very sparse" training set (*Left*); this is because the "very sparse" subset contains mostly dark backgrounds and so less meaningful information can be extracted.

training cost of the naive ensemble method makes the deployment prohibitive to large-scale databases. Training an ensemble of K models will increase the computational cost by K times. As we have seen in the previous experiments, current fast ensemble methods are not meant for robust uncertainty estimation.

In the following sections, we introduce a simple yet effective ensemble method called FastEnsemble. Our approach is inspired by the recent findings mode connectivity of local minimum [GIP18]: we first find a seed model w_0, then explore along the "loss valley" by adding a bias term $\|w - w_0\|_1$ to the classification or regression loss. On convergence, we expect the new model w' to be as good as w_0, but show enough independence. Our idea contrasts to snapshot ensemble or batch ensemble, where the former is controlled by a cyclically climbing up and decaying learning rate. The latter takes no direct measure to achieve this.

FastEnsemble Algorithm

Denote the loss function of data pair (x_i, y_i) as ℓ; $f(\cdot; w)$ is the neural network parameterized by w. Our algorithm has two stages: in the initial stage, we train a "seed model" to convergence following the usual routine, the model is denoted as w_0. Then in the next stage, we augment the loss function ℓ by a series of ℓ_1 distances defined over model set $\mathcal{M}$.

$$\ell^+(w) = \ell\Big(f\big(x_i; w\big), y_i\Big) - \frac{\lambda}{|\mathcal{M}|} \cdot \sum_{w_{\text{anchor}} \in \mathcal{M}} \|w - w_{\text{anchor}}\|_1 . \tag{5.2}$$

Previous finding [GIP18] suggests that the low loss area (Figure 5.5) is connected. Once we train the seed model to a low loss, we can generate many good and independent models by simultaneously minimizing the training loss and maximizing the distance between the new model and all existing ones in $\mathcal{M}$. The algorithm in pseudo-code is shown in Algorithm 10.
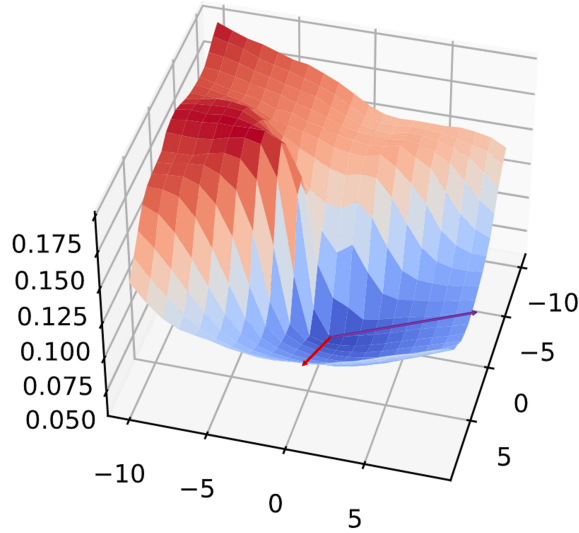
Figure 5.5: Loss landscape around a local minimum. There are multiple directions (in red arrows) we can choose to escape the local minimum while staying in the low loss "valley".

Notice in this algorithm, we choose the number of iterations $k_1 \gg k_2, k_3$ so that compared to the one-time seed model training, the rest $N-1$ ensemble members only takes $\frac{k_2 + k_3}{k_1} \approx 3{\sim}8\%$ overhead. That makes our new training overhead considerably cheaper than in snapshot

Algorithm 10 Algorithm of FastEnsemble

1: Initialize: N: number of ensemble models parameterized by w_i; $\ell(\hat{y}, y)$: the loss function; λ: the hyperparameter to be tuned. $k_1 \gg k_2, k_3$: number of iterations for training seeding model, training sub-models and fintuning sub-models.

2: // Train the seeding model

3: **for all** $i \in \{0 \ldots k_1 - 1\}$ **do**

4:     Run one step of optimizer and learning rate scheduler.

5: **end for**

6: Initial model list $\mathcal{M} = \{w_0\}$.

7: // Train the rest $N - 1$ models

8: **for all** $n \in \{1 \ldots N - 1\}$ **do**

9:     **for all** $i \in \{0 \ldots k_2 - 1\}$ **do**

10:         // Quick training

11:         Minimize $\ell^+(w)$ in Eq. (5.2).

12:     **end for**

13:     **for all** $i \in \{0 \ldots k_3 - 1\}$ **do**

14:         Minimize $\ell(\hat{y}, y)$.   ▷ Finetuning

15:     **end for**

16:     Append to model list $\mathcal{M} = \mathcal{M} + w_n$.

17: **end for**

---

ensemble. Our algorithm introduces a hyperparameter λ, which controls the trade-off between model accuracy and inter-model independence. In other words, a larger λ causes a lower model correlation (due to longer distances between $\mathcal{M}$), but the individual model performs worse than before.

Dataset	Naive	MC-Dropout	SGLD	SVI	BatchEnsemble	Snapshot	FastEnsemble
Measured by AUC (controlling FPR$\leq$ 0.2)							
MSC-Impurities	**0.112**	0.074	0.099	0.050	0.098	0.002	0.108
MSC-LNCaP	0.082	0.035	0.021	0.023	0.059	0.001	**0.090**
LNCaP-Desity(Model A), measured by mean Pearson correlation							
Very dense	**0.869**	0.803	0.756	0.762	0.865	0.853	0.865
Dense	**0.925**	0.869	0.803	0.807	0.919	0.909	0.923
Sparse	**0.952**	0.909	0.853	0.849	0.947	0.939	0.950
Very sparse	**0.974**	0.933	0.894	0.887	0.968	0.960	0.971
LNCaP-Density(Model B), measured by mean Pearson correlation							
Very dense	**0.956**	0.865	0.861	0.734	0.953	0.945	0.952
Dense	**0.963**	0.863	0.887	0.778	0.961	0.957	0.962
Sparse	**0.967**	0.843	0.902	0.818	0.965	0.962	0.966
Very sparse	0.971	0.815	0.924	0.846	0.969	0.970	**0.972**

Table 5.1: Experimental results in image generation benchmark. For clarity, the first place is marked in **bold font**, the second place is in red, the third place is in blue.

We repeat all the experiments in Section 5.3 again with our proposed method, then make some comparisons in AUROC or Pearson correlation measures. The results are displayed in Table 5.1. The naive ensemble method is still better than the others except MSC-LNCaP dataset; this indicates that current fast ensemble models are still sacrificing accuracy for the speed. Among all efficient methods, our FastEnsemblesurpasses all others in the MSC-LNCaP dataset and ranked second on all other datasets.

5.5 Classification benchmark and calibration robustness

In this section, we intend to show that the proposed FastEnsemble method can also work on regular classification tasks. In particular, we first run on CIFAR10 and CIFAR100 as two standard datasets, then we move to out-of-distribution robustness on CIFAR10-C [HD19a] and CIFAR100-C. Finally, we focus on biomedical imaging datasets, Camelyon17 [BGM18] and RxRx1 [TEM19], as two larger-scale real applications.

We measure three things: accuracy, log-likelihood, and confidence calibration. Confidence is defined as the probability in the model output (values after sigmoid or softmax function). As the size of the deep learning model grows, the model can easily fit the training set to a low NLL loss by generating probabilities closer to one-hot distribution, which implicitly hurts the confidence estimation [GPS17]. We quantify the miscalibration level by the expected calibration error (ECE) [NCH15]:

$$\text{ECE} = \int_0^1 w(p) \cdot \big|\text{Acc}(p) - p\big| \, \mathrm{d}p. \tag{5.3}$$

In this equation p is the confidence output from Softmax; $w(p)$ is percent of data having confidence p ; $\text{Acc}(p)$ is the accuracy as a function of confidence. In practice, the integration (5.3) is computed by confidence binning $\text{ECE} = \sum_{m=1}^{M} \frac{|B_m|}{n} \big|\text{Acc}(B_m) - \text{Conf}(B_m)\big|$, in which $B_m = ((m-1)/M, m/M]$ is the m-th bin between $[0, 1]$.

For the network architecture and training configurations, we mostly follow the previous literature. Specifically:

- CIFAR: This configuration applies to all CIFAR based datasets. We train with AdamW optimizer for 200 epochs, batch size is 128. We adopt the linear learning rate scheduler, the initial learning rate is 1.0×10^{-3}.

- Camelyon17: This is a collection of tissue slides under microscopy, in which training and testing distributions differ due to patient population or in slide staining and image acquisition. We follow the configuration in WILDS benchmark [KSX21]. The

model architecture is a ImageNet-1k pretrained DenseNet121 [HLV17], finetuned with momentum SGD and batch size = 32.

- RxRx1: Similar to Camelyon17, there is a distribution shift due to the batch effect. We choose ImageNet-1k pretrained ResNet50 [HZR16] to initialize the model, finetuned with Adam and batch size = 72.

More experiment details can be found in Appendix. First, we explore the accuracy and ECE under distribution shift. The results can be found in Figure 5.6.
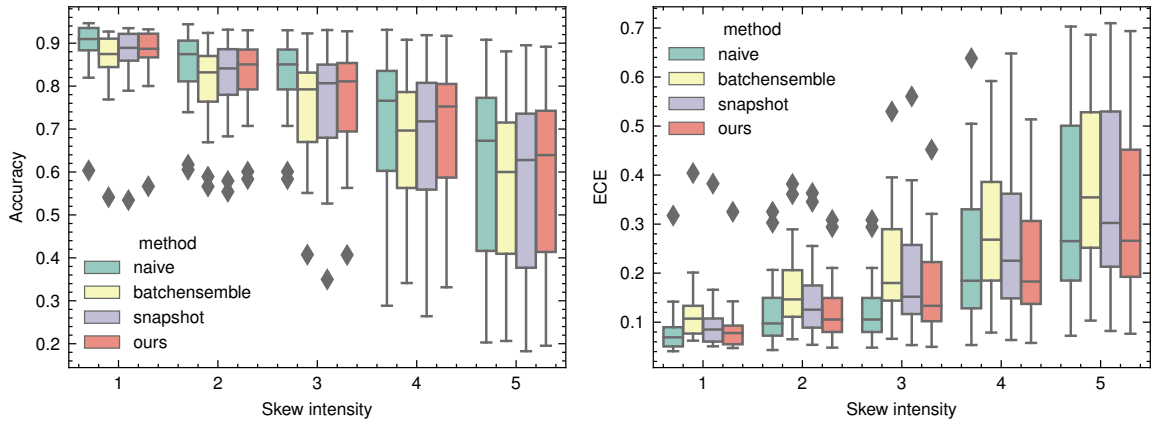
Figure 5.6: CIFAR10-C: accuracy and ECE (the lower the better) degrade as image skewness intensifies. The box plot is made by aggregating the measurements over 15 kinds of corruptions made by [HD19a].

The figure shows that the naive ensemble method is still the best choice considering the best accuracy and calibration in all cases. But our approach is on par with the naive ensemble; both are significantly better than batch ensemble and snapshot ensemble. Next, we repeat the same routine to all six datasets to compare accuracy, log-likelihood, as well as ECE. We repeated the experiments three times by changing random seeds. Finally, we report the mean measures and standard deviations. From Table 5.2, we can conclude that our method is the closest to naive ensemble in terms of accuracy, and often has the lowest calibration error on the datasets we tested.

	Single			Naive			Batch			Snapshot			Ours		
	ACC / NLL / ECE			ACC / NLL / ECE			ACC / NLL / ECE			ACC / NLL / ECE			ACC / NLL / ECE		
CIFAR10+VGG16:															
	92.89	0.492	0.058	94.64	0.306	0.041	92.79	0.566	0.061	93.62	0.375	0.049	93.24	0.308	0.047
	0.10	0.031	0.002	0.06	0.018	0.001	0.08	0.019	0.001	0.21	0.029	0.001	0.19	0.015	0.002
CIFAR100+VGG16:															
	68.65	2.496	0.236	75.15	1.516	0.173	68.44	2.954	0.252	70.52	1.775	0.198	71.14	1.326	0.149
	0.10	0.137	0.004	0.07	0.105	0.003	0.16	0.074	0.003	0.39	0.126	0.015	0.29	0.103	0.012
Camelyon17+DenseNet121:															
	84.99	0.397	0.083	85.96	0.347	0.066	84.39	0.399	0.081	Failure			87.71	0.305	0.048
	1.06	0.038	0.012	0.04	0.004	0.003	—						0.33	0.016	0.009
RxRx1+ResNet50:															
	25.82	7.908	0.469	34.80	5.638	0.370	30.31	7.409	0.450	19.36	5.556	0.265	31.08	6.490	0.407
	0.27	0.025	0.002	0.07	0.195	0.012	0.51	0.272	0.012	0.06	0.167	0.019	0.39	0.091	0.003
Test-only datasets using models trained from CIFAR10 and CIFAR100															
CIFAR10-C+VGG16:															
	86.84	0.980	0.110	89.21	0.671	0.084	85.91	1.180	0.121	87.00	0.810	0.102	87.38	0.623	0.089
	0.56	0.105	0.006	0.27	0.052	0.003	0.13	0.018	0.001	0.21	0.061	0.003	0.31	0.312	0.002
CIFAR100-C+VGG16:															
	55.81	4.117	0.335	63.45	2.670	0.256	55.15	5.054	0.359	58.36	3.035	0.282	58.96	2.249	0.220
	0.28	0.286	0.006	0.36	0.246	0.009	0.14	0.100	0.003	0.35	0.224	0.019	0.31	0.139	0.007

Table 5.2: Experiment results on distribution shifted or clean datasets. Mean values are in normal font. Standard deviations are computed over three independent runs, and we display them in gray color. The metrics are NLL/ACC/ECE. Notice that CIFAR10-C and CIFAR100-C are test-only datasets; we evaluate them using the same model checkpoint acquired from CIFAR10 and CIFAR100. In Camelyon17+DenseNet121 combination, we found the snapshot ensemble method failed to converge in all three trials. The reason is that when the learning rate spikes at the beginning of the second cycle, the optimizer makes an unnecessarily big step to drive the model out of the low loss area.

CHAPTER 6

Demo: An AI-based Biomedical Imaging Software

6.1 Goal of this software

We want to design a user-friendly software for the biological researchers to build their own models without writing any code. There will be several essential components as follows:

- **Model training and inference**. This is the core of our software. It is composed of a data loader, trainer, and U-Net backbone. The data loader reads pair-wise images from file system, preprocessing it with multiple data augmentation algorithms, then normalize it to $[0, 1]$. The trainer is a Bayesian ensemble trainer, we choose ensemble model to support uncertainty estimation at inference time.

- **Web-based user-interface**. This is the only place to collect information from users. Since we assume no programming ability is required, we added a few widgets to gather dataset URLs, user name, email address, etc. all by HTML pages. Apart from that, we also need another page to display all current and historical jobs. For instance, users may want to know the current status of the job they submitted a few hours ago, or the download link once the job is finished.

- **Control panel**. This panel belongs to the web interface but worth to be mentioned separately. Although our default hyperparameters are tested on many datasets (see the **Data repository** item below), some advanced users with deep learning backgrounds will be able to change the hyperparameters so fit their needs. Often they want to

increase the number of ensemble models to generate higher precision results, or better uncertainty estimations. Or, they want to decrease the number of training epochs because the dataset is large.

- **Data repository**. This repository contains some datasets that are ready to use out-of-the-box. There are MSC cells, Enza cells, etc. Users can try our software without collecting their own images, just download the zip files from this data repository and upload to their own Google drive.

6.2 Design

Our software is open sourced in `https://github.com/xuanqing94/fnet-web`. The core model training and inference is open sourced in `https://github.com/xuanqing94/BNNBench`. The website is implemented with Bootstrap and Flask, with a built-in database powered by sqlite. We use PyTorch to implement all the deep learning training and inference pipelines. The workflow can be shown in Figure 6.1.
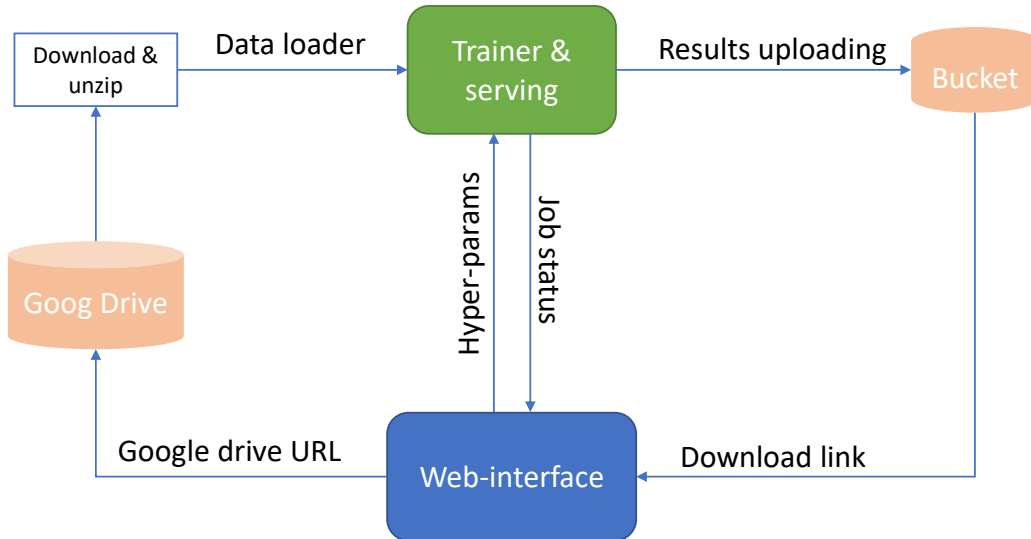
Figure 6.1: Workflow of the Web-UI training interface.

The submission page is shown in Figure 6.2. In the left panel, users will input their contact information such as name and email. The right panel contains some of the most important hyper-parameters that users might need to change.

Figure 6.2: The control panel of our system

The job status page is shown in Figure 6.3. There are currently five different status: SUBMIT, QUEUE, RUN, SUCCESS, FAILURE. When the job is completed in SUCCESS state, the downloadable link will be displayed in the last column.

The results are packaged in a zip file with following contents: the source images, ground truth target images, and the prediction images.

Job ID#	Timestamp ↓	Name	Email	Proj name	Status	Download
4ec16781-5658-419a-acdd-33723cec7846	2021-10-10 13:17:07	Xuanqing Liu	liuatpku@gmail.com	iefoegeor	SUCCESS	Download
19711b41-d49b-47e3-94be-84512817c3f6	2019-01-01 09:00:04	Xuanqing Liu	xqliu@cs.ucla.edu	Test job list	FAILURE	Check later
0faa8f5a-4d4c-4a29-bd17-e171535f6016	2019-01-01 09:00:03	Xuanqing Liu	xqliu@cs.ucla.edu	Test job list	SUCCESS	Download
6d02e6ce-396c-47f1-ba9a-08062a330c30	2019-01-01 09:00:02	Xuanqing Liu	xqliu@cs.ucla.edu	Test job list	RUN	Check later
9f371651-db42-4551-96be-594b5b6e3327	2019-01-01 09:00:01	Xuanqing Liu	xqliu@cs.ucla.edu	Test job list	QUEUE	Check later
ad972cf8-76a4-45a6-9f64-1249e83cbc8f	2019-01-01 09:00:00	Xuanqing Liu	xqliu@cs.ucla.edu	Test job list	SUBMIT	Check later

List of submitted jobs

Figure 6.3: Job queue table

Model Interpretation

CHAPTER 7

Model Interpretation by Robustness Analysis

7.1 Robustness Analysis for Evaluating Explanations

7.1.1 Problem notation

We consider the setting of a general K-way classification problem with input space $\mathcal{X} \subseteq \mathbb{R}^d$, output space $\mathcal{Y} = \{1, \ldots, K\}$, and a predictor function $f : \mathcal{X} \to \mathcal{Y}$ where $f(\boldsymbol{x})$ denotes the output class for some input example $\boldsymbol{x} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_d] \in \mathcal{X}$. Then, for a particular prediction $f(\boldsymbol{x}) = y$, a common goal of feature based explanations is to extract a compact set of relevant features with respect to the prediction. We denote the set of relevant features provided by an explanation as $S_r \subseteq U$ where $U = \{1, \ldots, d\}$ is the set of all features, and use $\overline{S_r} = U \setminus S_r$, the complementary set of S_r, to denote the set of irrelevant features. We further use $\boldsymbol{x}_S$ to denote the features within $\boldsymbol{x}$ that are restricted to the set S.

7.1.2 Evaluation through robustness analysis

A common thread underlying evaluations of feature based explanations [SBM16, PDS18], even ranging over axiomatic treatments [STY17, LL17], is that the importance of a set of features corresponds to the change in prediction of the model when the features are removed from the original input. Nevertheless, as we discussed in previous sections, operationalizing such a removal of features, for instance, by setting them to some reference value, introduces biases. To finesse this, we leverage adversarial robustness, but to do so in this context, we rely on two key intuitive assumptions that motivate our method:

Assumption 1: When the values of the important features are anchored (fixed), perturbations restricted to the complementary set of features has a weaker influence on the model prediction.

Assumption 2: When perturbations are restricted to the set of important features, fixing the values of the rest of the features, even small perturbations could easily change the model prediction.

Based on these two assumptions, we propose a new framework leveraging the notion of adversarial robustness on feature subsets, as defined below, to evaluate feature based explanations.

Definition 7.1.1. *Given a model f, an input $\boldsymbol{x}$, and a set of features $S \subseteq U$ where U is the set of all features, the minimum adversarial perturbation norm on $\boldsymbol{x}_S$, which we will also term Robustness-S of $\boldsymbol{x}$ is defined as:*

$$\epsilon^*_{\boldsymbol{x}_S} = g(f, \boldsymbol{x}, S) = \left\{ \min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p \ s.t. \ f(\boldsymbol{x} + \boldsymbol{\delta}) \neq y, \ \boldsymbol{\delta}_{\overline{S}} = 0 \right\}, \tag{7.1}$$

where $y = f(\boldsymbol{x})$, $\overline{S} = U \setminus S$ is the complementary set of features, and $\boldsymbol{\delta}_{\overline{S}} = 0$ means that the perturbation is constrained to be zero along features in $\overline{S}$.

Suppose that a feature based explanation partitions the input features of $\boldsymbol{x}$ into a relevant set S_r, and an irrelevant set $\overline{S_r}$, Assumption 1 implies that the quality of the relevant set can be measured by $\epsilon^*_{\boldsymbol{x}_{\overline{S_r}}}$ – by keeping the relevant set unchanged, and measuring the adversarial robustness norm by perturbing only the irrelevant set. Specifically, from Assumption 1, a larger coverage of pertinent features in set S_r entails a higher robustness value $\epsilon^*_{\boldsymbol{x}_{\overline{S_r}}}$. On the other hand, from Assumption 2, a larger coverage of pertinent features in set S_r would in turn entail a smaller robustness value $\epsilon^*_{\boldsymbol{x}_{S_r}}$, since only relevant features are perturbed. More formally, we propose the following twin criteria for evaluating the quality of S_r identified by any given feature based explanation.

Definition 7.1.2. *Given an input* $\boldsymbol{x}$ *and a relevant feature set* S_r, *we define Robustness-*$\overline{S_r}$ *and Robustness-*S_r *of the input* $\boldsymbol{x}$ *as the following:*

$$Robustness\text{-}\overline{S_r} = \epsilon^*_{\boldsymbol{x}_{\overline{S_r}}}. \qquad Robustness\text{-}S_r = \epsilon^*_{\boldsymbol{x}_{S_r}}.$$

Following our assumptions, a set S_r that has larger coverage of relevant features would yield *higher* Robustness-$\overline{S_r}$ and *lower* Robustness-S_r.

Evaluation for Feature Importance Explanations. While Robustness-$\overline{S_r}$ and Robustness-S_r are defined on sets, general feature attribution based explanations could also easily fit into the evaluation framework. Given any feature attribution method that assigns importance score to each feature, we can sort the features in descending order of importance weights, and provide the top-K features as the relevant set S_r. The size of K (or $|S_r|$), can be specified by the users based on the application. An alternative approach that we adopt in our experiments is to vary the size of set K and plot the corresponding values of Robustness-$\overline{S_r}$ and Robustness-S_r over different values of K. With a graph where the $X-$axis is the size of K and the $Y-$axis is Robustness-$\overline{S_r}$ or Robustness-S_r, we are then able to plot an evaluation curve for an explanation and in turn compute its the area under curve (AUC) to summarize its performance. A larger (smaller) AUC for Robustness-$\overline{S_r}$ (Robustness-S_r) indicates a better feature attribution ranking. Formally, given a curve represented by a set of points $\mathcal{C} = \{(x_0, y_0), \ldots, (x_n, y_n)\}$ where $x_{i-1} < x_i$, we calculate the AUC of the curve by: $AUC(\mathcal{C}) = \sum_{i=1}^{n} (y_i + y_{i-1})/2 * (x_i - x_{i-1})$.

Relation to Insertion and Deletion Criteria. We relate the proposed criteria to a set of commonly adopted evaluation metrics: the *Insertion* and *Deletion* criteria [SBM16, PDS18, SLL20]. The Insertion score measures the model's function value when only the top-relevant features, given by an explanation, are presented in the input while the others are removed (usually by setting them to some reference value representing feature missingness). The Deletion score, on the other hand, measures the model's function value when the most

relevant features are masked from the input. As in our evaluation framework, we could plot the evaluation curves for Insertion (Deletion) score by progressively increasing the number of top-relevant features. A larger (smaller) AUC under Insertion (Deletion) then indicates better explanation, as the identified relevant features could indeed greatly influence the model prediction. We note that optimizing the proposed Robustness-$\overline{S_r}$ and Robustness-S_r could roughly be seen as optimizing a lower bound for the Insertion and Deletion score respectively. This follows from the intuition: Robustness-$\overline{S_r}$ considers features that when anchored, would make the prediction most robust to "adversarial perturbation". Since adversarial perturbation is the worst case of "any arbitrary perturbations", the prediction will also be robust to different removal techniques (which essentially correspond to different perturbations) considered in the evaluation of Insertion score; The same applies to the connection between Robustness-S_r and Deletion score. We shall see in the experiment section that explanation optimizing our robustness measurements enjoys competitive performances on the Insertion / Deletion criteria.

Untargeted v.s. Targeted Explanation. Definition 7.1.1 corresponds to the untargeted adversarial robustness – a perturbation that changes the predicted class to any label other than y is considered as a successful attack. Our formulation can also be extended to **targeted adversarial robustness**, where we replace (7.1) by:

$$\epsilon^*_{\boldsymbol{x}_S, t} = \left\{ \min_{\boldsymbol{\delta}} \|\boldsymbol{\delta}\|_p \text{ s.t. } f(\boldsymbol{x} + \boldsymbol{\delta}) = t; \boldsymbol{\delta}_{\overline{S}} = 0 \right\}, \qquad (7.2)$$

where t is the targeted class. Using this definition, our approach will try to address the question "Why is this example classified as y instead of t" by highlighting the important features that contrast between class y and t. Further examples of the "targeted explanations" are in the experiment section.

Robustness Evaluation on Feature Subset. It is known that computing the exact minimum distortion distance in modern neural networks is intractable [KBD17], so many

different methods have been developed to estimate the value. Adversarial attacks, such as C&W [CW17] and projected gradient descent (PGD) attack [MMS18a], aim to find a feasible solution of (7.1), which leads to an upper bound of $\epsilon^*_{\boldsymbol{x}_S}$. They are based on gradient based optimizers which are usually efficient. On the other hand, neural network verification methods aim to provide a lower bound of $\epsilon^*_{\boldsymbol{x}_S}$ to ensure that the model prediction will not change within certain perturbation range [SGM18, WK18, WZC18a, GMD18, ZWC18, WPW18, ZZH19]. The proposed framework can be combined with any method that aims to approximately compute (7.1), including attack, verification, and some other statistical estimations. However, for simplicity we only choose to evaluate (7.1) by the state-of-the-art PGD attack [MMS18a], since the verification methods are too slow and often lead to much looser estimation as reported in some recent studies [SYZ19]. Our additional constraint restricting perturbation to only be on a subset of features specifies a set that is simple to project onto, where we set the corresponding coordinates to zero at each step of PGD.

7.2 Extracting Relevant Features through Robustness Analysis

Our adversarial robustness based evaluations allow us to evaluate any given feature based explanation. Here, we set out to design new explanations that explicitly optimize our evaluation measure. We focus on feature set based explanations, where we aim to provide an important subset of features S_r. Given our proposed evaluation measure, an optimal subset of feature S_r would aim to maximize (minimize) Robustness-$\overline{S_r}$ (Robustness-S_r), under a cardinality constraint on the feature set, leading to the following set of optimization problems:

$$\underset{S_r \subseteq U}{\text{maximize}} \quad g(f, \boldsymbol{x}, \overline{S_r}) \quad \text{s.t.} \quad |S_r| \leq K \tag{7.3}$$

$$\underset{S_r \subseteq U}{\text{minimize}} \quad g(f, \boldsymbol{x}, S_r) \quad \text{s.t.} \quad |S_r| \leq K \tag{7.4}$$

where K is a pre-defined size constraint on the set S_r, and $g(f, \boldsymbol{x}, S)$ computes the the minimum adversarial perturbation from (7.1), with set-restricted perturbations.

It can be seen that this sets up an adversarial game for (7.3) (or a co-operative game for (7.4)). In the adversarial game, the goal of the feature set explainer is to come up with a set S_r such that the minimal adversarial perturbation is as large as possible, while the adversarial attacker, given a set S_r, aims to design adversarial perturbations that are as small as possible. Conversely in the co-operative game, the explainer and attacker cooperate to minimize the perturbation norm. Directly solving these problems in (7.3) and (7.4) is thus challenging, which is exacerbated by the discrete input constraint that makes it intractable to find the optimal solution. We therefore propose a greedy algorithm in the next section to estimate the optimal explanation sets.

7.2.1 Greedy algorithm to compute optimal explanations

We first consider a greedy algorithm where, after initializing S_r to the empty set, we iteratively add to S_r the most promising feature that optimizes the objective at each local step until S_r reaches the size constraint. We thus sequentially solve the following sub-problem at every step t:

$$\arg\max_i \ g(f, \boldsymbol{x}, \overline{S_r^t \cup i}), \ \text{or} \ \arg\min_i \ g(f, \boldsymbol{x}, S_r^t \cup i), \ \forall i \in \overline{S_r^t} \tag{7.5}$$

where S_r^t is the relevant set at step t, and $S_r^0 = \emptyset$. We repeat this subprocedure until the size of set S_r^t reaches K. A straightforward approach for solving (7.5) is to exhaustively search over every single feature. We term this method **Greedy**. While the method eventually selects K features for the relevant set S_r, it might lose the sequence in which the features were selected. One approach to encode this order would be to output a feature explanation that assigns higher weights to those features selected earlier in the greedy iterations.

7.2.2 Greedy by set aggregation score

The main downside of using the greedy algorithm to optimize the objective function is that it ignores the interactions among features. Two features that may seem irrelevant when evaluated separately might nonetheless be relevant when added simultaneously. Therefore, in each greedy step, instead of considering how each individual feature will marginally contribute to the objective $g(\cdot)$, we propose to choose features based on their expected marginal contribution when added to the union of S_r and a random subset of unchosen features. To measure such an aggregated contribution score, we draw from cooperative game theory literature [DS79, HH92] to reduce this to a linear regression problem. Formally, let S_r^t and $\overline{S_r^t}$ be the ordered set of chosen and unchosen features at step t respectively, and $\mathcal{P}(\overline{S_r^t})$ be all possible subsets of $\overline{S_r^t}$. We measure the expected contribution that including each unchosen feature to the relevant set would have on the objective function by learning the following regression problem:

$$\boldsymbol{w}^t, c^t = \arg\min_{\boldsymbol{w}, c} \sum_{S \in \mathcal{P}(\overline{S_r^t})} ((\boldsymbol{w}^T b(S) + c) - v(S_r^t \cup S))^2, \tag{7.6}$$

where $b : \mathcal{P}(\overline{S_r^t}) \to \{0, 1\}^{|\overline{S_r^t}|}$ is a function that projects a set into its corresponding binary vector form: $b(S)[j] = \mathbb{I}(\overline{S_r^t}[j] \in S)$, and $v(\cdot)$ is set to be the objective function in (7.3) or (7.4): $v(S_r) = g(f, \boldsymbol{x}, \overline{S_r})$ for optimizing (7.3); $v(S_r) = g(f, \boldsymbol{x}, S_r)$ for optimizing (7.4). We note that $\boldsymbol{w}^t$ corresponds to the well-known Banzhaf value [Ban64] when $S_r^t = \emptyset$, which can be interpreted as the importance of each player by taking coalitions into account [DS79]. [HH92] show that the Banzhaf value is equivalent to the optimal solution of linear regression with pseudo-Boolean functions as targets, which corresponds to (7.6) with $S_r^t = \emptyset$. At each step t, we can thus treat the linear regression coefficients $\boldsymbol{w}^t$ in (7.6) as each corresponding feature's expected marginal contribution when added to the union of S_r and a random subset of unchosen features.

We thus consider the following set-aggregated variant of our greedy algorithm in the previous section, which we term **Greedy-AS**. In each greedy step t, we choose features

that are expected to contribute most to the objective function, i.e. features with highest (for (7.3)) or lowest (for (7.4)) aggregation score (Banzhaf value), rather than simply the highest marginal contribution to the objective function as in vanilla greedy. This allows us to additionally consider the interactions among the unchosen features when compared to vanilla greedy. The chosen features each step are then added to S_r^t and removed from $\overline{S_r^t}$. When S_r^t is not $\emptyset$, the solution of (7.6) can still be seen as the Banzhaf value where the players are the unchosen features in $\overline{S_r^t}$, and the value function computes the objective when a subset of players is added into the current set of chosen features S_r^t. We solve the linear regression problem in (7.6) by sub-sampling to lower the computational cost, and we validate the effectiveness of Greedy and Greedy-AS in the experiment section. [1]

7.3 Experiments

In this section, we first evaluate different model interpretability methods on the proposed criteria. We justify the effectiveness of the proposed Greedy-AS. We then move onto further validating the benefits of the explanations extracted by Greedy-AS through comparisons to various existing methods both quantitatively and qualitatively. Finally, we demonstrate the flexibility of our method with the ability to provide targeted explanations as mentioned in Section 7.1.2. We perform the experiments on two image datasets, MNIST [LCB10] and ImageNet [DDS09a], as well as a text classification dataset, Yahoo! Answers [ZZL15]. On MNIST, we train a convolutional neural network (CNN) with 99% testing accuracy. On ImageNet, we deploy a pre-trained ResNet model obtained from the Pytorch library. On Yahoo! Answers, we train a BiLSTM sentence classifier which attains testing accuracy of 71%.

---

[1] We found that concurrent to our work, greedy with choosing the players with the highest restricted Banzhaf was used in [EFL17].

Setup. In the experiments, we consider $p = 2$ for $\| \cdot \|_p$ in (7.1) and (7.2). We note that (7.1) is defined for a single data example. Given n multiple examples $\{\boldsymbol{x}_i\}_{i=1}^n$ with their corresponding relevant sets provided by some explanation $\{S_i\}_{i=1}^n$, we compute the overall Robustness-$\overline{S_r}$ (-S_r) by taking the average: for instance, $\frac{1}{n} \sum_{i=1}^n g(f, \boldsymbol{x}_i, \overline{S_i})$ for Robustness-$\overline{S_r}$. We then plot the evaluation curve as discussed in Section 7.1.2 and report the AUC for different explanation methods. For all quantitative results, we report the average over 100 random examples. For the baseline methods, we include vanilla gradient (Grad) [SGK17], integrated gradient (IG) [STY17], and expected gradient (EG) [EJS19, SLL20] from gradient-based approaches; leave-one-out (LOO) [ZF14, LMJ16], SHAP [LL17] and black-box meaningful perturbation (BBMP) (only for image examples) [FV17] from perturbation-based approaches [ACO18]; counterfactual explanation (CFX) proposed by [WMR17]; Anchor [RSG18] for text examples; and a Random baseline that ranks feature importance randomly. Following common setup [STY17, ACO18], we use zero as the reference value for all explanations that require baseline.

7.3.1 Robustness analysis on model interpretability methods

Comparisons between Different Explanations. From Table 7.1, we observe that the proposed Greedy-AS consistently outperforms other explanation methods on both criteria. On one hand, this suggests that the proposed algorithm indeed successfully optimizes towards the criteria; on the other hand, this might indicate the proposed criteria do capture different characteristics of explanations which most of the current explanations do not possess. Another somewhat interesting finding from the table is that while Grad has generally been viewed as a baseline method, it nonetheless performs competitively on the proposed criteria. We conjecture the phenomenon results from the fact that Grad does not assume any reference value as opposed to other baselines such as LOO which sets the reference value as zero to mask out the inputs. Indeed, it might not be surprising that Greedy-AS achieves the best performances on the proposed criteria since it is explicitly designed for so. To more

158

Table 7.1: AUC of Robustness-$\overline{S_r}$ and Robustness-S_r for various explanations on different datasets. The higher the better for Robustness-$\overline{S_r}$; the lower the better for Robustness-S_r.

Datasets	Explanations	Grad	IG	EG	SHAP	LOO	BBMP	CFX	Random	Greedy-AS
MNIST	Robustness-$\overline{S_r}$	88.00	85.98	93.24	75.48	74.14	78.58	69.88	64.44	**98.01**
	Robustness-S_r	91.72	91.97	91.05	101.49	104.38	176.61	102.81	193.75	**82.81**
ImageNet	Robustness-$\overline{S_r}$	27.13	26.01	26.88	18.25	22.29	21.56	27.12	17.98	**31.62**
	Robustness-S_r	45.53	46.28	48.82	60.02	58.46	158.01	46.10	56.11	**43.97**
Yahoo!Answer	Robustness-$\overline{S_r}$	1.97	1.86	1.96	1.81	1.74	-	1.95	1.71	**2.13**
	Robustness-S_r	2.91	3.14	2.99	3.34	4.04	-	2.96	7.64	**2.41**

objectively evaluate the usefulness of the proposed explanation, we demonstrate different advantages of our method by comparing Greedy-AS to other explanations quantitatively on existing commonly adopted measurements, and qualitatively through visualization in the following subsections.

7.3.2 Qualitative results

Image Classification. To complement the quantitative measurements, we show several visualization results on MNIST and ImageNet in Figure 7.1 and Figure 7.2. On MNIST, we observe that existing explanations tend to highlight mainly on the white pixels in the digits; among which SHAP and LOO show less noisy explanations comparing to Grad and IG. On the other hand, the proposed Greedy-AS focuses on both the "crucial positive" (important white pixels) as well as the "pertinent negative" (important black regions) that together support the prediction. For example, in the first row, a 7 might have been predicted as a 4 or 0 if the pixels highlighted by Greedy-AS are set to white. Similarly, a 1 may be turned to a 4 or a 7 given additional white pixels to its left, and a 9 may become a 7 if deleted the lower circular part of its head. From the results, we see that Greedy-AS focuses on *"the region where perturbation on its current value will lead to easier prediction change"*,
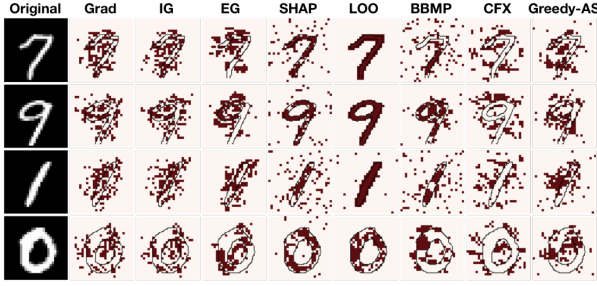
Figure 7.1: Visualization on top 20 percent relevant features provided by different explanations on MNIST. We see Greedy-AS highlights both crucial positive and pertinent negative features supporting the prediction.
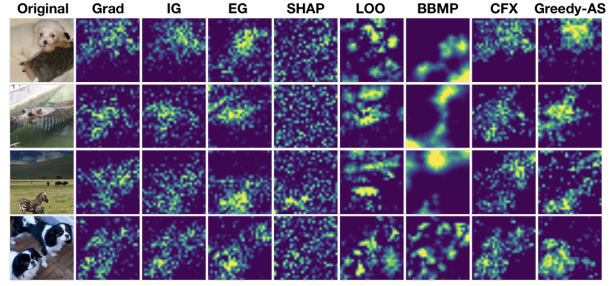
Figure 7.2: Visualization of different explanations on ImageNet, where the predicted class for each input is "Maltese", "hippopotamus", "zebra", and "Japanese Spaniel". Greedy-AS focuses more compactly on objects.

which includes both the crucial positive and pertinent negative pixels. Such capability of Greedy-AS is also validated by its superior performance on the proposed robustness criteria, on which methods like LOO that highlights only the white strokes of digits show relatively low performance. The capability of capturing pertinent negative features has also been observed in explanations proposed in some recent work [DCL18b, BBM15, OWT19]. From the visualized ImageNet examples shown in Figure 7.2, we observe that our method provides more compact explanations that focus mainly on the actual objects being classified. For instance, in the first image, our method focuses more on the face of the Maltese while others tend to have noisier results; in the last image, our method focuses on one of the Japanese Spaniel whereas others highlight both the dogs and some noisy regions.

Text Classification. Here we demonstrate our explanation method on a text classification model that classifies a given sentence into one of the ten classes (Society, Science, . . . , Health). We showcase an example in Figure 7.3. We see that the top-5 keywords highlighted by Greedy-AS are all relevant to the label "sport", and Greedy-AS is less likely to select stop words as compared to other methods.

Figure 7.3: Explanations on a text classification model which correctly predicts the label "sport". Unlike most other methods, the top-5 relevant keywords highlighted by Greedy-AS are all related to the concept "sport".
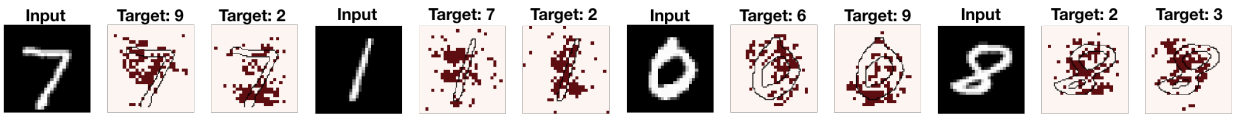
Figure 7.4: Visualization of targeted explanation. For each input, we highlight relevant regions explaining why the input is not predicted as the target class. We see the explanation changes in a semantically meaningful way as the target class changes.

Targeted Explanation Analysis. In section 7.1.2, we discussed about the possibility of using targeted adversarial perturbation to answer the question of *"why the input is predicted as A but not B"*. In Figure 7.4, for each input digit, we provide targeted explanation towards two different target classes. Interestingly, as the target class changes, the generated explanation varies in an interpretable way. For example, in the first image, we explain why the input digit 7 is not classified as a 9 (middle column) or a 2 (rightmost column). The resulting explanation against 9 highlights the upper-left part of the 7. Semantically, this region is indeed pertinent to the classification between 7 and 9, since turning on the highlighted pixel values in the region (currently black in the original image) will then make the 7 resemble a 9. However, the targeted explanation against 2 highlights a very different but also meaningful region, which is the lower-right part of the 7; since adding a horizontal stroke on the area would turn a 7 into a 2.

REFERENCES

[ABB17]   Laurent Amsaleg, James Bailey, Dominique Barbe, Sarah M. Erfani, Michael E. Houle, Vinh Nguyen, and Milos Radovanovic. "The vulnerability of learning to adversarial perturbation increases with intrinsic dimensionality." In *IEEE Workshop on Information Forensics and Security*, pp. 1–6, 2017.

[AC18]   Anish Athalye and Nicholas Carlini. "On the Robustness of the CVPR 2018 White-Box Adversarial Example Defenses." *arXiv preprint arXiv:1804.03286*, 2018.

[ACO18]   Marco Ancona, Enea Ceolini, Cengiz Oztireli, and Markus Gross. "A unified view of gradient-based attribution methods for Deep Neural Networks." *International Conference on Learning Representations*, 2018.

[ACW18]   Anish Athalye, Nicholas Carlini, and David Wagner. "Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples." *arXiv preprint arXiv:1802.00420*, 2018.

[AK99]   Franz Aurenhammer and Rolf Klein. "Voronoi diagrams." *Handbook of computational geometry*, **5**(10):201–290, 1999.

[APH21]   Moloud Abdar, Farhad Pourpanah, Sadiq Hussain, Dana Rezazadegan, Li Liu, Mohammad Ghavamzadeh, Paul Fieguth, Xiaochun Cao, Abbas Khosravi, U Rajendra Acharya, et al. "A review of uncertainty quantification in deep learning: Techniques, applications and challenges." *Information Fusion*, 2021.

[Ban64]   John F Banzhaf III. "Weighted voting doesn't work: A mathematical analysis." *Rutgers L. Rev.*, **19**:317, 1964.

[BBM15]   Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. "On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation." *PloS one*, **10**(7):e0130140, 2015.

[BCK15]   Charles Blundell, Julien Cornebise, Koray Kavukcuoglu, and Daan Wierstra. "Weight Uncertainty in Neural Network." In *International Conference on Machine Learning*, pp. 1613–1622, 2015.

[BGM18]   Peter Bandi, Oscar Geessink, Quirine Manson, Marcory Van Dijk, Maschenka Balkenhol, Meyke Hermsen, Babak Ehteshami Bejnordi, Byungjae Lee, Kyunghyun Paeng, Aoxiao Zhong, et al. "From detection of individual metastases to classification of lymph node status at the patient level: the camelyon17 challenge." *IEEE transactions on medical imaging*, **38**(2):550–560, 2018.

[BKM17]   David M Blei, Alp Kucukelbir, and Jon D McAuliffe. "Variational inference: A review for statisticians." *Journal of the American Statistical Association*, **112**(518):859–877, 2017.

[BR18]   Battista Biggio and Fabio Roli. "Wild patterns: Ten years after the rise of adversarial machine learning." *Pattern Recognition*, **84**:317–331, 2018.

[BRB18]   Wieland Brendel, Jonas Rauber, and Matthias Bethge. "Decision-based adversarial attacks: Reliable attacks against black-box machine learning models." In *ICLR*, 2018.

[BRR18]   Jacob Buckman, Aurko Roy, Colin Raffel, and Ian Goodfellow. "Thermometer Encoding: One Hot Way To Resist Adversarial Examples." In *International Conference on Learning Representations*, 2018.

[BSI08]   Oren Boiman, Eli Shechtman, and Michal Irani. "In defense of Nearest-Neighbor based image classification." In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2008.

[CL11]   Chih-Chung Chang and Chih-Jen Lin. "LIBSVM: A library for support vector machines." *ACM Transactions on Intelligent Systems and Technology*, **2**(3):27, 2011.

[CLC19]   Minhao Cheng, Thong Le, Pin-Yu Chen, Jinfeng Yi, Huan Zhang, and Cho-Jui Hsieh. "Query-efficient hard-label black-box attack: An optimization-based approach." In *ICLR*, 2019.

[CNL11]   Adam Coates, Andrew Ng, and Honglak Lee. "An analysis of single-layer networks in unsupervised feature learning." In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pp. 215–223, 2011.

[CRB18]   Tian Qi Chen, Yulia Rubanova, Jesse Bettencourt, and David K Duvenaud. "Neural ordinary differential equations." In *Advances in Neural Information Processing Systems*, pp. 6572–6583, 2018.

[CRK19]   Jeremy M Cohen, Elan Rosenfeld, and J Zico Kolter. "Certified adversarial robustness via randomized smoothing." *arXiv preprint arXiv:1902.02918*, 2019.

[CSC20]   Minhao Cheng, Simranjit Singh, Pin-Yu Chen, Sijia Liu, and Cho-Jui Hsieh. "Sign-OPT: A Query-Efficient Hard-label Adversarial Attack." In *International Conference on Learning Representations (ICLR)*, 2020.

[CSZ18]   Pin-Yu Chen, Yash Sharma, Huan Zhang, Jinfeng Yi, and Cho-Jui Hsieh. "EAD: Elastic-Net Attacks to Deep Neural Networks via Adversarial Examples." In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[CW17]   Nicholas Carlini and David Wagner. "Towards evaluating the robustness of neural networks." In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 39–57. IEEE, 2017.

[CYA18]   Eric M Christiansen, Samuel J Yang, D Michael Ando, Ashkan Javaherian, Gaia Skibinski, Scott Lipnick, Elliot Mount, Alison O'Neil, Kevan Shah, Alicia K Lee, et al. "In silico labeling: predicting fluorescent labels in unlabeled images." *Cell*, **173**(3):792–803, 2018.

[CZS17]   Pin-Yu Chen, Huan Zhang, Yash Sharma, Jinfeng Yi, and Cho-Jui Hsieh. "Zoo: Zeroth order optimization based black-box attacks to deep neural networks without training substitute models." In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 15–26. ACM, 2017.

[DAB18]   Guneet S. Dhillon, Kamyar Azizzadenesheli, Jeremy D. Bernstein, Jean Kossaifi, Aran Khanna, Zachary C. Lipton, and Animashree Anandkumar. "Stochastic activation pruning for robust adversarial defense." In *International Conference on Learning Representations*, 2018.

[DCB13]   Alexander Domahidi, Eric Chu, and Stephen Boyd. "ECOS: An SOCP solver for embedded systems." In *European Control Conference*, pp. 3071–3076, 2013.

[DCL18a] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. "BERT: Pre-training of deep bidirectional transformers for language understanding." *arXiv preprint arXiv:1810.04805*, 2018.

[DCL18b] Amit Dhurandhar, Pin-Yu Chen, Ronny Luss, Chun-Chen Tu, Paishun Ting, Karthikeyan Shanmugam, and Payel Das. "Explanations based on the missing: Towards contrastive explanations with pertinent negatives." In *Advances in Neural Information Processing Systems*, pp. 592–603, 2018.

[DDS09a] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. "ImageNet: A Large-Scale Hierarchical Image Database." In *CVPR09*, 2009.

[DDS09b] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 248–255. Ieee, 2009.

[DGV18]   Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. "Universal transformers." *arXiv preprint arXiv:1807.03819*, 2018.

[DKJ07]   Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. "Information-theoretic metric learning." In *International Conference on Machine learning (ICML)*, pp. 209–216, 2007.

[DMY19]   Abhimanyu Dubey, Laurens van der Maaten, Zeki Yalniz, Yixuan Li, and Dhruv Mahajan. "Defense Against Adversarial Images using Web-Scale Nearest-Neighbor Search." In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8767–8776, 2019.

[DS79]   Pradeep Dubey and Lloyd S Shapley. "Mathematical properties of the Banzhaf power index." *Mathematics of Operations Research*, **4**(2):99–131, 1979.

[EFL17]   Edith Elkind, Piotr Faliszewski, Martin Lackner, Dominik Peters, and Nimrod Talmon. "Committee scoring rules, banzhaf values, and approximation algorithms." In *4th workshop on exploring beyond the worst case in computational social choice (EXPLORE'17)*, 2017.

[EIA18]   Logan Engstrom, Andrew Ilyas, and Anish Athalye. "Evaluating and Understanding the Robustness of Adversarial Logit Pairing." *arXiv preprint arXiv:1807.10272*, 2018.

[EJS19]   Gabriel Erion, Joseph D Janizek, Pascal Sturmfels, Scott Lundberg, and Su-In Lee. "Learning explainable models using attribution priors." *arXiv preprint arXiv:1906.10670*, 2019.

[FCS17]   Reuben Feinman, Ryan R Curtin, Saurabh Shintre, and Andrew B Gardner. "Detecting Adversarial Samples from Artifacts." *arXiv preprint arXiv:1703.00410*, 2017.

[FMM18]   Michael Figurnov, Shakir Mohamed, and Andriy Mnih. "Implicit Reparameterization Gradients." *arXiv preprint arXiv:1805.08498*, 2018.

[FV17]   Ruth C. Fong and Andrea Vedaldi. "Interpretable Explanations of Black Boxes by Meaningful Perturbation." *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3449–3457, 2017.

[GCB18]   Will Grathwohl, Ricky TQ Chen, Jesse Betterncourt, Ilya Sutskever, and David Duvenaud. "Ffjord: Free-form continuous dynamics for scalable reversible generative models." *arXiv preprint arXiv:1810.01367*, 2018.

[GG16]   Yarin Gal and Zoubin Ghahramani. "Dropout as a bayesian approximation: Representing model uncertainty in deep learning." In *international conference on machine learning*, pp. 1050–1059. PMLR, 2016.

[GHR04]   Jacob Goldberger, Geoffrey E. Hinton, Sam T. Roweis, and Ruslan R Salakhutdinov. "Neighbourhood Components Analysis." In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 513–520, 2004.

[GIP18] Timur Garipov, Pavel Izmailov, Dmitrii Podoprikhin, Dmitry Vetrov, and Andrew Gordon Wilson. "Loss surfaces, mode connectivity, and fast ensembling of dnns." In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 8803–8812, 2018.

[GMD18] Timon Gehr, Matthew Mirman, Dana Drachsler-Cohen, Petar Tsankov, Swarat Chaudhuri, and Martin Vechev. "Ai2: Safety and robustness certification of neural networks with abstract interpretation." In *IEEE Symposium on Security and Privacy*, pp. 3–18, 2018.

[GMP17] Kathrin Grosse, Praveen Manoharan, Nicolas Papernot, Michael Backes, and Patrick McDaniel. "On the (statistical) detection of adversarial examples." *arXiv preprint arXiv:1702.06280*, 2017.

[GPS17] Chuan Guo, Geoff Pleiss, Yu Sun, and Kilian Q Weinberger. "On calibration of modern neural networks." In *International Conference on Machine Learning*, pp. 1321–1330. PMLR, 2017.

[GRC18] Chuan Guo, Mayank Rana, Moustapha Cisse, and Laurens van der Maaten. "Countering Adversarial Images using Input Transformations." In *International Conference on Learning Representations*, 2018.

[GSS15a] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and Harnessing Adversarial Examples." In *International Conference on Learning Representations*, 2015.

[GSS15b] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples." In *ICLR*, 2015.

[GVS09] Matthieu Guillaumin, Jakob Verbeek, and Cordelia Schmid. "Is that you? Metric learning approaches for face identification." In *International Conference on Computer Vision (ICCV)*, pp. 498–505, 2009.

[HA17] Matthias Hein and Maksym Andriushchenko. "Formal Guarantees on the Robustness of a Classifier against Adversarial Manipulation." In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 2263–2273, 2017.

[HBW07] Gang Hua, Matthew Brown, and Simon Winder. "Discriminant embedding for local image descriptors." In *International Conference on Computer Vision (ICCV)*, pp. 1–8, 2007.

[HD19a] Dan Hendrycks and Thomas Dietterich. "Benchmarking Neural Network Robustness to Common Corruptions and Perturbations." *Proceedings of the International Conference on Learning Representations*, 2019.

[HD19b]   Dan Hendrycks and Thomas Dietterich. "Benchmarking neural network robustness to common corruptions and perturbations." *arXiv preprint arXiv:1903.12261*, 2019.

[HGD17]   Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask r-cnn." In *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[HH92]   Peter L Hammer and Ron Holzman. "Approximations of pseudo-Boolean functions; applications to game theory." *Zeitschrift für Operations Research*, **36**(1):3–21, 1992.

[HK16]   Elad Hazan and Tomer Koren. "A linear-time algorithm for trust region problems." *Mathematical Programming*, **158**(1-2):363–381, 2016.

[HLP17]   Gao Huang, Yixuan Li, Geoff Pleiss, Zhuang Liu, John E Hopcroft, and Kilian Q Weinberger. "Snapshot ensembles: Train 1, get m for free." *arXiv preprint arXiv:1704.00109*, 2017.

[HLV17]   Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. "Densely connected convolutional networks." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[HW21]   Eyke Hüllermeier and Willem Waegeman. "Aleatoric and epistemic uncertainty in machine learning: An introduction to concepts and methods." *Machine Learning*, **110**(3):457–506, 2021.

[HXS15]   Ruitong Huang, Bing Xu, Dale Schuurmans, and Csaba Szepesvári. "Learning with a strong adversary." *arXiv preprint arXiv:1511.03034*, 2015.

[HZR16]   Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[IEM19]   Andrew Ilyas, Logan Engstrom, and Aleksander Madry. "Prior Convictions: Black-box Adversarial Attacks with Bandits and Priors." In *International Conference on Learning Representations (ICLR)*, 2019.

[ILL21]   Sara Imboden, Xuanqing Liu, Brandon S Lee, Marie C Payne, Cho-Jui Hsieh, and Neil YC Lin. "Investigating heterogeneities of live mesenchymal stromal cells using AI-based label-free imaging." *Scientific Reports*, **11**(1):1–11, 2021.

[IR05]   Hemant Ishwaran, J Sunil Rao, et al. "Spike and slab variable selection: frequentist and Bayesian strategies." *The Annals of Statistics*, **33**(2):730–773, 2005.

[JKD10] Prateek Jain, Brian Kulis, and Inderjit S. Dhillon. "Inductive Regularized Learning of Kernel Functions." In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 946–954, 2010.

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization." *arXiv preprint arXiv:1412.6980*, 2014.

[KBD17] Guy Katz, Clark Barrett, David L Dill, Kyle Julian, and Mykel J Kochenderfer. "Reluplex: An efficient SMT solver for verifying deep neural networks." In *International Conference on Computer Aided Verification*, pp. 97–117, 2017.

[KG17] Alex Kendall and Yarin Gal. "What uncertainties do we need in bayesian deep learning for computer vision?" *arXiv preprint arXiv:1703.04977*, 2017.

[KGB17a] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. "Adversarial machine learning at scale." In *International Conference on Learning Representations (ICLR)*, 2017.

[KGB17b] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. "Adversarial Machine Learning at Scale." In *International Conference on Learning Representations (ICLR)*, 2017.

[Kin17] Diederik P Kingma. "Variational inference & deep learning: A new synthesis." 2017.

[KJG09] Brian Kulis, Prateek Jain, and Kristen Grauman. "Fast similarity search for learned metrics." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **31**(12):2143–2157, 2009.

[KP13] Peter E Kloeden and Eckhard Platen. *Numerical solution of stochastic differential equations*, volume 23. Springer Science & Business Media, 2013.

[KS98] Ioannis Karatzas and Steven E Shreve. "Brownian motion." In *Brownian Motion and Stochastic Calculus*, pp. 47–127. Springer, 1998.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." *Advances in neural information processing systems*, **25**:1097–1105, 2012.

[KSW15] Diederik P Kingma, Tim Salimans, and Max Welling. "Variational dropout and the local reparameterization trick." In *Advances in Neural Information Processing Systems*, pp. 2575–2583, 2015.

[KSX21] Pang Wei Koh, Shiori Sagawa, Sang Michael Xie, Marvin Zhang, Akshay Balsubramani, Weihua Hu, Michihiro Yasunaga, Richard Lanas Phillips, Irena Gao, Tony Lee, et al. "Wilds: A benchmark of in-the-wild distribution shifts." In *International Conference on Machine Learning*, pp. 5637–5664. PMLR, 2021.

[KTJ16]   Alex Kantchelian, JD Tygar, and Anthony Joseph. "Evasion and hardening of tree ensemble classifiers." In *International Conference on Machine Learning*, pp. 2387–2396, 2016.

[KTK80]   Mikhail K Kozlov, Sergei P Tarasov, and Leonid G Khachiyan. "The polynomial solvability of convex quadratic programming." *USSR Computational Mathematics and Mathematical Physics*, **20**(5):223–228, 1980.

[LAG18a]   M. Lecuyer, V. Atlidakis, R. Geambasu, D. Hsu, and S. Jana. "Certified Robustness to Adversarial Examples with Differential Privacy." *ArXiv e-prints*, February 2018.

[LAG18b]   Mathias Lecuyer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, and Suman Jana. "Certified robustness to adversarial examples with differential privacy." *arXiv preprint arXiv:1802.03471*, 2018.

[LBB98]   Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, **86**(11):2278–2324, 1998.

[LCB10]   Yann LeCun, Corinna Cortes, and CJ Burges. "MNIST handwritten digit database." *ATT Labs [Online]. Available: http://yann. lecun. com/exdb/mnist*, **2**, 2010.

[LCC16]   Chunyuan Li, Changyou Chen, David E Carlson, and Lawrence Carin. "Preconditioned Stochastic Gradient Langevin Dynamics for Deep Neural Networks." In *AAAI*, volume 2, p. 4, 2016.

[LCG19]   Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. "Albert: A lite BERT for self-supervised learning of language representations." *arXiv preprint arXiv:1909.11942*, 2019.

[LCL17]   Yanpei Liu, Xinyun Chen, Chang Liu, and Dawn Song. "Delving into transferable adversarial examples and black-box attacks." In *International Conference on Learning Representation*, 2017.

[LCZ17]   Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. "Towards Robust Neural Networks via Random Self-ensemble." *arXiv preprint arXiv:1712.00673*, 2017.

[LCZ18]   Xuanqing Liu, Minhao Cheng, Huan Zhang, and Cho-Jui Hsieh. "Towards robust neural networks via random self-ensemble." In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 369–385, 2018.

[LDG17]   Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. "Feature pyramid networks for object detection." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[Leb06]   Guy Lebanon. "Metric learning for text documents." *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **28**(4):497–508, 2006.

[LH18]   Xuanqing Liu and Cho-Jui Hsieh. "From Adversarial Training to Generative Adversarial Networks." *arXiv preprint arXiv:1807.10454*, 2018.

[LL17]   Scott M Lundberg and Su-In Lee. "A unified approach to interpreting model predictions." In *Advances in Neural Information Processing Systems*, pp. 4765–4774, 2017.

[LMJ16]   Jiwei Li, Will Monroe, and Dan Jurafsky. "Understanding Neural Networks through Representation Erasure." *CoRR*, **abs/1612.08220**, 2016.

[LOG19]   Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. "RoBERTa: A robustly optimized BERT pretraining approach." *arXiv preprint arXiv:1907.11692*, 2019.

[LSD15]   Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation." In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[LXL17]   Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. "RACE: Large-scale reading comprehension dataset from examinations." *arXiv preprint arXiv:1704.04683*, 2017.

[Mao07]   Xuerong Mao. *Stochastic differential equations and applications*. Elsevier, 2007.

[MC17]   Dongyu Meng and Hao Chen. "MagNet: A Two-Pronged Defense Against Adversarial Examples." In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, pp. 135–147, New York, NY, USA, 2017. ACM.

[MGV18]   Matthew Mirman, Timon Gehr, and Martin Vechev. "Differentiable abstract interpretation for provably robust neural networks." In *International Conference on Machine Learning*, pp. 3575–3583, 2018.

[Mil75]   GN Mil'shtein. "Approximate integration of stochastic differential equations." *Theory of Probability & Its Applications*, **19**(3):557–562, 1975.

[MK18]   Takeru Miyato and Masanori Koyama. "cGANs with Projection Discriminator."
            2018.

[MLW18]   Xingjun Ma, Bo Li, Yisen Wang, Sarah M. Erfani, Sudanthi Wijewickrema, Grant
            Schoenebeck, Michael E. Houle, Dawn Song, and James Bailey. "Characterizing
            Adversarial Subspaces Using Local Intrinsic Dimensionality." In *International
            Conference on Learning Representations*, 2018.

[MMS18a]   Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and
            Adrian Vladu. "Towards deep learning models resistant to adversarial attacks."
            6-th International Conference on Learning Representations (ICLR), 2018.

[MMS18b]   Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and
            Adrian Vladu. "Towards Deep Learning Models Resistant to Adversarial Attacks."
            In *International Conference on Learning Representations (ICLR)*, 2018.

[NCH15]   Mahdi Pakdaman Naeini, Gregory Cooper, and Milos Hauskrecht. "Obtaining
            well calibrated probabilities using bayesian binning." In *Twenty-Ninth AAAI
            Conference on Artificial Intelligence*, 2015.

[NYM17]   Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. "Regulariz-
            ing Deep Neural Networks by Noise: Its Interpretation and Optimization." In
            Advances in Neural Information Processing Systems, pp. 5115–5124, 2017.

[OCP16]   Brendan O'Donoghue, Eric Chu, Neal Parikh, and Stephen Boyd. "Conic Opti-
            mization via Operator Splitting and Homogeneous Self-Dual Embedding." *Journal
            of Optimization Theory and Applications*, **169**(3):1042–1068, 2016.

[OEB19]   Myle Ott, Sergey Edunov, Alexei Baevski, Angela Fan, Sam Gross, Nathan Ng,
            David Grangier, and Michael Auli. "fairseq: A fast, extensible toolkit for sequence
            modeling." *arXiv preprint arXiv:1904.01038*, 2019.

[OFR19]   Yaniv Ovadia, Emily Fertig, Jie Ren, Zachary Nado, David Sculley, Sebastian
            Nowozin, Joshua V Dillon, Balaji Lakshminarayanan, and Jasper Snoek. "Can
            you trust your model's uncertainty? Evaluating predictive uncertainty under
            dataset shift." *arXiv preprint arXiv:1906.02530*, 2019.

[Oks03]   Bernt Øksendal. "Stochastic differential equations." In *Stochastic differential
            equations*, pp. 65–84. Springer, 2003.

[OSM18]   Chawin Ounkomol, Sharmishtaa Seshamani, Mary M Maleckar, Forrest Collman,
            and Gregory R Johnson. "Label-free prediction of three-dimensional fluorescence
            images from transmitted-light microscopy." *Nature methods*, **15**(11):917–920,
            2018.

[OWT19]   Jose Oramas, Kaili Wang, and Tinne Tuytelaars. "Visual Explanation by Interpretation: Improving Visual Feedback Capabilities of Deep Neural Networks." In *International Conference on Learning Representations*, 2019.

[PDS18]   Vitali Petsiuk, Abir Das, and Kate Saenko. "RISE: Randomized Input Sampling for Explanation of Black-box Models." *arXiv preprint arXiv:1806.07421*, 2018.

[Pla98]   John Platt. "Sequential minimal optimization: A fast algorithm for training support vector machines." *Microsoft Research Technical Report*, 1998.

[PM18]   Nicolas Papernot and Patrick McDaniel. "Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning." *CoRR*, **abs/1803.04765**, 2018.

[PMG16a]   Nicolas Papernot, Patrick McDaniel, and Ian Goodfellow. "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples." *arXiv preprint arXiv:1605.07277*, 2016.

[PMG16b]   Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. "Practical black-box attacks against deep learning systems using adversarial examples." *arXiv preprint*, 2016.

[PMG16c]   Nicolas Papernot, Patrick D. McDaniel, and Ian J. Goodfellow. "Transferability in Machine Learning: from Phenomena to Black-Box Attacks using Adversarial Samples." *CoRR*, **abs/1605.07277**, 2016.

[PMW16]   Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. "Distillation as a defense to adversarial perturbations against deep neural networks." In *Security and Privacy (SP), 2016 IEEE Symposium on*, pp. 582–597. IEEE, 2016.

[RFB15]   Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." In *International Conference on Medical image computing and computer-assisted intervention*, pp. 234–241. Springer, 2015.

[RJL18]   Pranav Rajpurkar, Robin Jia, and Percy Liang. "Know what you don't know: Unanswerable questions for SQuAD." *arXiv preprint arXiv:1806.03822*, 2018.

[RSG18]   Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. "Anchors: High-precision model-agnostic explanations." In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[RSR19]   Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. "Exploring the limits of transfer learning with a unified text-to-text transformer." *arXiv preprint arXiv:1910.10683*, 2019.

[RWC19]   Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. "Language Models are Unsupervised Multitask Learners." 2019.

[RWW19]   Yair Rivenson, Hongda Wang, Zhensong Wei, Kevin de Haan, Yibo Zhang, Yichen Wu, Harun Günaydın, Jonathan E Zuckerman, Thomas Chong, Anthony E Sisk, et al. "Virtual histological staining of unlabelled tissue-autofluorescence images via deep learning." *Nature biomedical engineering*, **3**(6):466–477, 2019.

[RZL16]   Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. "SQuAD: 100,000+ questions for machine comprehension of text." *arXiv preprint arXiv:1606.05250*, 2016.

[SBM16]   Wojciech Samek, Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, and Klaus-Robert Müller. "Evaluating the visualization of what a deep neural network has learned." *IEEE transactions on neural networks and learning systems*, **28**(11):2660–2673, 2016.

[SGK17]   Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. "Learning important features through propagating activation differences." *International Conference on Machine Learning*, 2017.

[SGM18]   Gagandeep Singh, Timon Gehr, Matthew Mirman, Markus Püschel, and Martin Vechev. "Fast and effective robustness certification." In *Advances in Neural Information Processing Systems*, pp. 10802–10813, 2018.

[SHK14]   Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research*, **15**(1):1929–1958, 2014.

[SKC18]   Pouya Samangouei, Maya Kabkab, and Rama Chellappa. "Defense-GAN: Protecting Classifiers Against Adversarial Attacks Using Generative Models." In *International Conference on Learning Representations*, 2018.

[SKL17]   Jacob Steinhardt, Pang Wei W Koh, and Percy S Liang. "Certified defenses for data poisoning attacks." In *Advances in Neural Information Processing Systems*, pp. 3520–3532, 2017.

[SKN18]   Yang Song, Taesup Kim, Sebastian Nowozin, Stefano Ermon, and Nate Kushman. "PixelDefend: Leveraging Generative Models to Understand and Defend against Adversarial Examples." In *International Conference on Learning Representations*, 2018.

[SLL20]   Pascal Sturmfels, Scott Lundberg, and Su-In Lee. "Visualizing the impact of feature attribution baselines." *Distill*, **5**(1):e22, 2020.

[SST18]   Ludwig Schmidt, Shibani Santurkar, Dimitris Tsipras, Kunal Talwar, and Alek-sander Madry. "Adversarially Robust Generalization Requires More Data." *arXiv preprint arXiv:1804.11285*, 2018.

[SSZ17]   Jake Snell, Kevin Swersky, and Richard Zemel. "Prototypical Networks for Few-shot Learning." In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 4077–4087, 2017.

[STY17]   Mukund Sundararajan, Ankur Taly, and Qiqi Yan. "Axiomatic attribution for deep networks." In *International Conference on Machine Learning*, 2017.

[Sug07]   Masashi Sugiyama. "Dimensionality Reduction of Multimodal Labeled Data by Local Fisher Discriminant Analysis." *Journal of Machine Learning Research (JMLR)*, **8**:1027–1061, 2007.

[SUV18]   Peter Shaw, Jakob Uszkoreit, and Ashish Vaswani. "Self-Attention with Relative Position Representations." In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 464–468, 2018.

[SW19]   Chawin Sitawarin and David Wagner. "On the Robustness of Deep K-Nearest Neighbors." *CoRR*, **abs/1903.08333**, 2019.

[SW20]   Chawin Sitawarin and David A. Wagner. "Minimum-Norm Adversarial Examples on KNN and KNN-Based Models." *CoRR*, **arXiv/2003.06559**, 2020.

[SWW08]   Malcolm Slaney, Kilian Q. Weinberger, and William White. "LEARNING A METRIC FOR MUSIC SIMILARITY." In *International Symposium/Conference on Music Information Retrieval*, pp. 313–318, 2008.

[SYZ19]   Hadi Salman, Greg Yang, Huan Zhang, Cho-Jui Hsieh, and Pengchuan Zhang. "A Convex Relaxation Barrier to Tight Robust Verification of Neural Networks." *CoRR*, **abs/1902.08722**, 2019.

[SZ15]   Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In *International Conference on Learning Representation*, 2015.

[SZS13]   Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. "Intriguing properties of neural networks." *arXiv preprint arXiv:1312.6199*, 2013.

[SZS14]   Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. "Intriguing properties of neural networks." In *International Conference on Learning Representations (ICLR)*, 2014.

[TEM19] J. Taylor, B. Earnshaw, B. Mabey, M. Victors, and J. Yosinski. "RxRx1: An Image Set for Cellular Morphological Variation Across Many Experimental Batches." In *International Conference on Learning Representations (ICLR)*, 2019.

[TKP17] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Dan Boneh, and Patrick McDaniel. "Ensemble Adversarial Training: Attacks and Defenses." *arXiv preprint arXiv:1705.07204*, 2017.

[TMM17] George Tucker, Andriy Mnih, Chris J Maddison, John Lawson, and Jascha Sohl-Dickstein. "Rebar: Low-variance, unbiased gradient estimates for discrete latent variable models." In *NIPS*, 2017.

[TP85] M. Tenenbaum and H. Pollard. *Ordinary Differential Equations: An Elementary Textbook for Students of Mathematics, Engineering, and the Sciences*. Dover Books on Mathematics. Dover Publications, 1985.

[TSE18] Dimitris Tsipras, Shibani Santurkar, Logan Engstrom, Alexander Turner, and Aleksander Madry. "Robustness may be at odds with accuracy." 2018.

[VBL16] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Koray Kavukcuoglu, and Daan Wierstra. "Matching networks for one shot learning." In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 3637–3645, 2016.

[VSP17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need." In *Advances in Neural Information Processing Systems*, pp. 5998–6008, 2017.

[WJ08] Martin J Wainwright and Michael I Jordan. "Introduction to variational methods for graphical models." *Foundations and Trends in Machine Learning*, **1**:1–103, 2008.

[WK18] Eric Wong and Zico Kolter. "Provable Defenses against Adversarial Examples via the Convex Outer Adversarial Polytope." In *International Conference on Machine Learning (ICML)*, pp. 5283–5292, 2018.

[WLY19] Lu Wang, Xuanqing Liu, Jinfeng Yi, Zhi-Hua Zhou, and Cho-Jui Hsieh. "Evaluating the Robustness of Nearest Neighbor Classifiers: A Primal-Dual Perspective." *CoRR*, **abs/1906.03972**, 2019.

[WMR17] S. Wachter, Brent D. Mittelstadt, and Chris Russell. "Counterfactual Explanations Without Opening the Black Box: Automated Decisions and the GDPR." *European Economics: Microeconomics & Industrial Organization eJournal*, 2017.

[WPW18] Shiqi Wang, Kexin Pei, Justin Whitehouse, Junfeng Yang, and Suman Jana. "Efficient formal safety analysis of neural networks." In *Advances in Neural Information Processing Systems*, pp. 6367–6377, 2018.

[WS09]   Kilian Q. Weinberger and Lawrence K. Saul. "Distance Metric Learning for Large
           Margin Nearest Neighbor Classification." *Journal of Machine Learning Research*,
           10:207–244, 2009.

[WSM18]   Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and
           Samuel R Bowman. "GLUE: A multi-task benchmark and analysis platform
           for natural language understanding." *arXiv preprint arXiv:1804.07461*, 2018.

[WT11]   Max Welling and Yee W Teh. "Bayesian learning via stochastic gradient Langevin
           dynamics." In *Proceedings of the 28th International Conference on Machine
           Learning (ICML-11)*, pp. 681–688, 2011.

[WTB20]   Yeming Wen, Dustin Tran, and Jimmy Ba. "Batchensemble: an alterna-
           tive approach to efficient ensemble and lifelong learning." *arXiv preprint
           arXiv:2002.06715*, 2020.

[WZC18a]   Tsui-Wei Weng, Huan Zhang, Hongge Chen, Zhao Song, Cho-Jui Hsieh, Luca
           Daniel, Duane Boning, and Inderjit Dhillon. "Towards Fast Computation of Cer-
           tified Robustness for ReLU Networks." In *International Conference on Machine
           Learning*, pp. 5273–5282, 2018.

[WZC18b]   Tsui-Wei Weng, Huan Zhang, Pin-Yu Chen, Jinfeng Yi, Dong Su, Yupeng Gao,
           Cho-Jui Hsieh, and Luca Daniel. "Evaluating the Robustness of Neural Networks:
           An Extreme Value Theory Approach." *6-th International Conference on Learning
           Representations (ICLR)*, 2018.

[WZL19]   Benyou Wang, Donghao Zhao, Christina Lioma, Qiuchi Li, Peng Zhang, and
           Jakob Grue Simonsen. "Encoding word order in complex embeddings." *arXiv
           preprint arXiv:1912.12333*, 2019.

[WZY20]   Lu Wang, Huan Zhang, Jinfeng Yi, Cho-Jui Hsieh, and Yuan Jiang. "Span-
           ning Attack: Reinforce Black-box Attacks with Unlabeled Data." *CoRR*,
           abs/2005.04871, 2020.

[XC06]   Huilin Xiong and Xue-wen Chen. "Kernel-based distance metric learning for
           microarray data classification." *BMC bioinformatics*, **7**(1):299, 2006.

[XEQ18]   Weilin Xu, David Evans, and Yanjun Qi. "Feature Squeezing: Detecting Adversar-
           ial Examples in Deep Neural Networks." *Network and Distributed System Security
           Symposium*, 2018.

[XGD17]   Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. "Aggre-
           gated residual transformations for deep neural networks." In *Computer Vision
           and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 5987–5995.
           IEEE, 2017.

[XKS06]   Xiaopeng Xi, Eamonn Keogh, Christian Shelton, Li Wei, and Chotirat Ann Ratanamahatana. "Fast time series classification using numerosity reduction." In *International Conference on Machine Learning*, pp. 1033–1040, 2006.

[XRV17]   Han Xiao, Kashif Rasul, and Roland Vollgraf. "Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms." *CoRR*, **abs/1708.07747**, 2017.

[XWZ18]   Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. "Mitigating Adversarial Effects Through Randomization." In *International Conference on Learning Representations*, 2018.

[YDY19]   Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. "XLNet: Generalized Autoregressive Pretraining for Language Understanding." *arXiv preprint arXiv:1906.08237*, 2019.

[YRW20]   Yao-Yuan Yang, Cyrus Rashtchian, Yizhen Wang, and Kamalika Chaudhuri. "Robustness for Non-Parametric Classification: A Generic Attack and Defense." In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[YZS16]   Han-Jia Ye, De-Chuan Zhan, Xue-Min Si, Yuan Jiang, and Zhi-Hua Zhou. "What Makes Objects Similar: A Unified Multi-Metric Learning Approach." In *Advances in Neural Information Processing Systems (NeurIPS)*, pp. 1235–1243, 2016.

[ZCX20]   Huan Zhang, Hongge Chen, Chaowei Xiao, Sven Gowal, Robert Stanforth, Bo Li, Duane Boning, and Cho-Jui Hsieh. "Towards Stable and Efficient Training of Verifiably Robust Neural Networks." In *International Conference on Learning Representations (ICLR)*, 2020.

[ZF14]   Matthew D Zeiler and Rob Fergus. "Visualizing and understanding convolutional networks." In *European conference on computer vision*, pp. 818–833. Springer, 2014.

[ZGL03]   Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. "Semi-supervised learning using gaussian fields and harmonic functions." In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pp. 912–919, 2003.

[ZNR17]   Valentina Zantedeschi, Maria-Irina Nicolae, and Ambrish Rawat. "Efficient defenses against adversarial attacks." In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pp. 39–49. ACM, 2017.

[ZWC18]   Huan Zhang, Tsui-Wei Weng, Pin-Yu Chen, Cho-Jui Hsieh, and Luca Daniel. "Efficient neural network robustness certification with general activation functions." In *Advances in Neural Information Processing Systems*, pp. 4944–4953, 2018.

[ZZH19]   Huan Zhang, Pengchuan Zhang, and Cho-Jui Hsieh. "RecurJac: An Efficient Recursive Algorithm for Bounding Jacobian Matrix of Neural Networks and Its Applications." In *AAAI Conference on Artificial Intelligence*, pp. 5757–5764, 2019.

[ZZL15]   Xiang Zhang, Junbo Zhao, and Yann LeCun. "Character-level convolutional networks for text classification." In *Advances in neural information processing systems*, pp. 649–657, 2015.