A Scalable and Modular Framework for Training Provably Robust Large Transformer Models via Neural Network Duality

Andre (Jianyou) Wang *
CSE Department
University of California San Diego
La Jolla, CA 92093
jiw101@ucsd.edu

Yongce Li †
Halıcıoğlu Data Science Institute
Department of Mathematics
University of California San Diego
La Jolla, CA 92093
yol013@ucsd.edu

Weili Cao ‡
CSE Department
University of California San Diego
La Jolla, CA 92093
w2cao@ucsd.edu

Yang Yue
CSE Department
University of California San Diego
La Jolla, CA 92093
yayue@ucsd.edu

Abstract

We propose a comprehensive modular framework for enhancing and verifying the robustness of deep neural networks against norm-bounded adversarial attacks by leveraging dualized network representations. We provide rigorous mathematical proof for the derivation of dual problem and provide dual layer formulation for common network layers such as linear, residual, ReLU and attention layers. We calculate dual layer upper bounds for various layers in Transformer models such as Fourier Transformer. Our work include robustly training, verifying, and dualizing Transformer-based networks for potential commercialization. Finally, as a proof of concept, we trained a robust classifier on MNIST data set using an automatic dualization algorithm and compare the robust error rate with the standard classifier. We demonstrated that our methods can reliably decrease the robust error rate from 100% to 4.16% with minimal loss in prediction accuracy. Our work could potentially contribute to more secure and robust neural networks, particularly in the domains of natural language processing and computer vision.

*Author of Equal Contribution
†Author of Equal Contribution
‡Author of Equal Contribution

2 Introduction

2.1 Motivation

Neural networks are powerful machine learning models that have been widely used in various applications. However, it is such a black-box dynamic system that is hard to analyze. Sometimes a small perturbation can cause dramatic effect to the model's output. As neural networks become more complex and are applied in more critical domains such as healthcare, finance, and autonomous system like self-driving vehicles, their robustness has become a crucial issue.

Robustness of a neural network refers to its ability to defense different adversarial attacks. Typically, for an input x, we would like to verify whether the network can output the correct label on a norm ball around x (that is, on $\mathcal{B}_\epsilon(x) = \{x + \Delta : \|\Delta\| \leq \epsilon\}$). A common approach to train robust network against adversarial attack is to first find the worst adversarial example within the norm ball, calculate the loss according to this adversarial example, and then updates weights according to such robust loss. In this paper, we formulate the problem of finding the worst adversarial example be as a minimization problem subject to the connection of the neural network. Taking this minimization problem as our primal problem, we can use Lagrangian dual to lower bound it and thus get a lower bound for the network's robustness to the input. Furthermore, the calculation of the robust loss also involves the Lagrangian dual. Surprisingly, the Lagrangian dual can be conveniently calculated with a backward pass through the original network. With this technique, we can train neural networks that are inherently more robust to variations of the input data.

This different approach that we use to improve robustness gain us guarantees to the adversarial attack. Namely, for points that have positive lower bounds, we can guarantee that they are robust to any norm bounded attack within ϵ. So we does not have any false negatives in the result. However, as a drawback, this does slightly increase the test error, meaning that there is greater change that we predict a wrong label.

Contrary to adversarial training in computer vision which is clearly defined as norm-bounded perturbations on the pixel level, adversarial input cannot be so clearly and mathematically defined in natural language processing. However, the need to deal with adversarial perturbation in deployed NLP systems, which has already been used in many sectors such as customer service and robotics assistants, could be even more pressing than that in computer vision systems. With the advent of large language models such as GPT3, GPT4, and chatGPT, there has been an increasing demand for reliable, robust and verifiable natural language understanding and generation systems. [7] [1] [10] [17]

2.2 Previous Works

Since 2013, studies on properties of neural networks found that neural networks are extremely vulnerable towards small perturbations in input data. For computer vision classification tasks, an adversarial input data that is visually indistinguishable from the original input would produce a wrong label, even on accurate models[18]. Since then, research have directed towards defense against these adversarial attacks.

In one direction, researchers have tried to incorporate adversarial examples as part of the training of neural networks. [9] introduces Fast Gradient Sign Method (FGSM) to generate adversarial examples. [15] introduces Projected Gradient Descent (PGD) which performs gradient descent on the input image pixels to produce a small perturbed adversarial noise. PGD is considered one of the most effective benchmark adversarial attacks for training adversarial networks and detecting adversarial input.

In another line of work, there are also a large amount of studies on finding the exact solutions of the optimization problem. One of the solvers is Satisfiability Modulo Theories (SMT), which deals with Satisfiability problem and generalizes the SAT solvers by extending them to handle richer logical structures [11] [12] [8]. Another is Integer Programming (IP), which deals with linear programming problems where some variables are constrained to integer domain [14][20][4]. With the advent of large deep models, exact solvers suffer from their lack of scalability towards hundreds of millions of parameters and have diminished in their importance.

Instead of the exact solvers, studies have also been done on applying relaxations towards the optimization problem. [2] proposed a mixed-integer programming (MIP) approaches for optimization problems containing trained neural networks. One of the main method would be certified defenses that provide provable guarantees on the neural networks [24] [25] [6], where the authors proposed to bound the output of a neural network with a convex polytope and proposed to solve a linear programming problem for bound estimation. Another method is to compute tractable bounds on the perturbation[5]. Specifically, Beta-Crown method uses Bound Propagation which is the state of the art method for verification [23]. There is an important distinction between verification of a existing neural network with pre-trained weights and training a robust optimization objective so that a neural network can be more robust against future adversarial attack. Our paper falls into the latter category and builds on previous aforementioned research by providing explicit proofs for these convex optimization approaches and by extending the capability of analyzing deeper and more complex neural architecture.

2.3 Intended Contributions

Our contributions are:

- We presented a comprehensive framework for calculating the maximum lower bound of the robust training objective via dualizing deep neural network in a modularized fashion, incorporating all previous techniques developed in this field, providing each with a concise mathematical proof using unified notations.
- In our experiments, we implemented efficient algorithms that calculate dual variables within the dualized network. We analyzed the computational runtime. Empirically, our methods are more robust against adversarial attacks with minimal degradation in prediction accuracy and significantly better at detecting adversarial examples in our test benchmark datasets such as MNIST. Specifically, we improved the standard accuracy of MNIST classifier from 98.87% to 99.1% and decreases the robust error from 100% to 4.16% which greatly improved the model robustness against adversarial attacks.
- We formulate the procedure for analytically calculating the dual layer upper bounds for multi-headed attention layers (as well as residual connection layer, ReLU layer and normalization layer) used in Transformer models in natural language processing and computer vision. To the best of our knowledge, our work represents the first attempt in robustly training, verifying and dualizing Transformer styled network, indicating the possibility of deploying our methods in a large neural language model for commercialization.
- We also propose Transformer-styled models that can be more easily dualized and verified by replacing the multi-headed attention layers with structured token-mixing layers across the positional sequence dimension, such as 1D Discrete Fourier Transform, while maintaining high-performance on par with large Transformer models.

3 Statement of the Problem

In deep learning, the classical optimization objective given a data distribution

$$(x, y) \sim (p(X), p(Y)) = \mathcal{D}, \quad x \in \mathbb{R}^{|x|}, y \in \mathbb{R}^{|y|}$$

is typically formulated as:

$$\min_{W} \mathbb{E}_{(x,y)\sim D} \mathcal{L}(f_W(x), y) \tag{1}$$

where $\mathcal{L}(f_W(x), y)$ is a scalar valued loss function. $f_W(x)$ denotes the model parameterized by W.

In the first half of our paper, we consider the multi-class classification problem where y denotes the target class index and $y \in \{1, 2, \ldots, C\}$, where C is the total number of classes. $\mathcal{L}$ is the cross-entropy loss, defined as:

$$\mathcal{L} : \mathbb{R}^C \times \{1, 2, \ldots, C\} \longrightarrow \mathbb{R} := \mathcal{L}(z, y) = -\log \frac{\exp z_y}{\sum_{i=1}^{C} \exp(z_i)} \quad \text{y denotes the target class index} \tag{2}$$

In order to train a robust classifier, we modify the objective in Equation 1 as:

$$\min_{W} \mathbb{E}_{(x,y)\sim D} \max_{||\Delta|| \leq \epsilon} \mathcal{L}(f_W(x+\Delta), y) \tag{3}$$

where Δ is a perturbation from original data points in the training dataset that is bounded by max norm (Though other norm could also apply).

The intuition of this robust training objective is that for each (x, y) in the empirical distribution, we find the most adversarial point $x + \Delta \in B_\epsilon(x)$ within the open ϵ ball of x such that the Loss $\mathcal{L}$ is maximized (the worst). Trained this way, f_W is expected to be empirically robust against norm-bounded adversarial attack without changing its prediction. However, we cannot provably claim that f_W will predict the entire $B_\epsilon(x)$ as the same label. Later, we will develop a technique to show for some input data x, our trained classifier f_W can provably predict the same label over its open neighborhood $B_\epsilon(x)$.

Equation 3 is a standard objective in robust optimization, but the $\max$ makes it impossible to train when f_W is a deep neural network. In this paper, we will develop another differentiable objective $J_\epsilon(x, A)$ that will upper bound Equation 3. Specifically,

Theorem 1 (Duality Upper Bound)**.**

$$\max_{||\Delta|| \leq \epsilon} \mathcal{L}(f_W(x+\Delta), y) \leq \mathcal{L}(-J_\epsilon(x, A), y), \quad A = g_W(1e_y^T - I) \tag{4}$$

where 1 is an all-one column vector, e_y is a standard basis vector with the y^{th} entry =1. I is the identity matrix. $g_W()$ is the dual network w.r.t to $f_W()$, which is the original network. The concept of network duality will be introduced in subsequent sections.

Given Theorem 1, we can relax Equation3 into the following objective:

$$\min_{W} \mathbb{E}_{(x,y)\sim D} \mathcal{L}(-J_\epsilon(x, A), y), \quad A = g_W(1e_y^T - I) \tag{5}$$

This relaxed objective is differentiable and can be solved efficiently via back-propagation. Note, $g_W()$'s parameters are tied with $f_W()$'s parameters.

Without formally defining J, we prove Theorem 1.

Proof. Theorem 1
Let $f_W(x + \Delta) = z$ for brevity.

$$\max_{||\Delta|| \leq \epsilon} \mathcal{L}(f_W(x+\Delta), y) = \max_{z} \mathcal{L}(z, y)$$

Since cross entropy is translation invariant, by Equation 2, we know $\mathcal{L}(z, y) = \mathcal{L}(z - z_y\vec{1}, y)$.

$$\max_{||\Delta|| \leq \epsilon} \mathcal{L}(f_W(x+\Delta), y) = \max_{z} \mathcal{L}(z - z_y\vec{1}, y) = \max \mathcal{L}(\begin{bmatrix} z_1 - z_y \\ \vdots \\ 0 \\ z_{y+1} - z_y \\ \vdots \\ z_C - z_y \end{bmatrix}, y)$$

Note, we let $-J_\epsilon(x, A)$ also be a vector of the same shape. In addition, we make $-J_\epsilon(x, A)$ so that each of its component satisfies the following conditions.

$$-J_\epsilon(x, A)_i \geq \max z_i - z_y, \quad i \neq y \tag{6}$$

$$-J_\epsilon(x, A)_y = 0 \tag{7}$$

$$\begin{bmatrix} -J_\epsilon(x, A)_1 \\ \vdots \\ 0 \\ -J_\epsilon(x, A)_{y+1} \\ \vdots \\ -J_\epsilon(x, A)_C \end{bmatrix} \geq \begin{bmatrix} \max_z z_1 - z_y \\ \vdots \\ 0 \\ \max_z z_{y+1} - z_y \\ \vdots \\ \max_z z_C - z_y \end{bmatrix} = \max_z \begin{bmatrix} z_1 - z_y \\ \vdots \\ 0 \\ z_{y+1} - z_y \\ \vdots \\ z_C - z_y \end{bmatrix}$$

We can take the max out because $\max z_i - z_y$ is only maximizing the z_i component of the vector while the z_y component's value do not change the value of the loss $\mathcal{L}$ because of translation invariance of cross entropy.

$$\mathcal{L}(\begin{bmatrix} -J_\epsilon(x,A)_1 \\ \vdots \\ 0 \\ -J_\epsilon(x,A)_{y+1} \\ \vdots \\ -J_\epsilon(x,A)_C \end{bmatrix}, y) = \log(\sum_{i=1}^{C} \exp(-J_\epsilon(x,A)_i)) \geq \log(\sum_{i=1}^{C} \exp(\max_z z_i - z_y) = \mathcal{L}(\begin{bmatrix} \max_z z_1 - z_y \\ \vdots \\ 0 \\ \max_z z_{y+1} - z_y \\ \vdots \\ \max_z z_C - z_y \end{bmatrix}, y)$$

The above equation proves the Duality upper Bound. Assume for now we know Condition 6 and 7 hold. (this will become clear once we introduced the formal definition of J). □

Upon careful inspection of Condition 6

$$-J_\epsilon(x,A)_i \geq \max z_i - z_y, \quad i \neq y$$
$$\implies J_\epsilon(x,A)_i \leq \min_z z_y - z_i, \quad i \neq y$$

where y is the true label and i is any other label. In the terminology of deep learning, we consider z_y and z_i to represent the logit (or probability) of a classifier $f_W()$ to predict the given input x as label i or label y. Since we are minimizing over z when input x is allowed to perturb by a small $||\Delta||_\infty < \epsilon$, we are essentially finding the most adversarial Δ such that $(x + \Delta)$ will be predicted as label i instead of label y.

As it turns out, the definition and formulation of J_ϵ is the dual problem of a primal problem, closely related to $\min_z z_y - z_i$,which we will introduce in the next section.

Note: starting next section, z_i will no longer represent a real value, it will be used as the vector output of the i^{th} layer of a k-layer ReLU-based neural network

3.1 Statement of the Primal and Dual Problem

The primal formulation defines adversarial polytope for neural networks using ReLU as activation function and corresponding convex outer bound. The KKT condition as well as dual formulation is used to optimize the convex outer bound by finding feasible solution of the dual problem using single modified backward pass. The last part is our main algorithm that uses backward pass variable to incrementally compute upper and lower activation bounds for all layers by reformulate our dual problem to a similar neural network.

3.2 Primal Formulation

We define our neural network based on ReLU as follows:

$$\hat{z}_{i+1} = W_i z_i + b_i, \forall i = 1, ..., k-1$$

$$z_i = \max \hat{z}_i, 0, \forall i = 2, ..., k-1$$

We also define the adversarial polytope as follows:

$$\mathcal{Z}_\epsilon(x) = \{f_\theta(x + \Delta) : ||\Delta||_\infty \leq \epsilon\}$$

Notice that ReLU function is not convex since it is taking maximum, but we can apply a convex relaxation which results in:

$$z \geq 0, z \geq \hat{z}, -u\hat{z} + (u - l)z \leq -ul$$

We donate the convex outer bound with this convex relaxation as $\tilde{\mathcal{Z}}_\epsilon(x)$. This outer bound can be used to provide provable guarantees on the adversarial robustness by showing any perturbation on our input would results in a bounded set. Given a sample x with known label y^* and alternative target label y^{targ}, we can write optimization problem as follows:

$$\begin{aligned} \min_{\hat{z}_k} \quad & (\hat{z}_k)_{y^*} - (\hat{z}_k)_{y^{targ}} \equiv c^T \hat{z}_k \\ \text{subject to} \quad & \hat{z}_k \in \tilde{\mathcal{Z}}_\epsilon(x) \end{aligned} \tag{8}$$

where $c \equiv e_{y^*} - e_{y^{targ}}$. With the convex relaxation on the ReLU constraint, the optimization becomes a LP problem. We write it out in formal format with all linear constraints as follows:

$$\begin{aligned} \underset{\hat{z}_k}{\text{minimize}} \quad & c^T \hat{z}_k \\ \text{subject to} \quad & \hat{z}_{i+1} = W_i z_i + b_i, \quad i = 1, \ldots, k-1, \\ & z_1 \leq x + \epsilon, \\ & z_1 \geq x - \epsilon, \\ & z_{i,j} = 0, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i^-, \\ & z_{i,j} = \hat{z}_{i,j}, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i^+, \\ & z_{i,j} \geq 0, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i, \\ & z_{i,j} \geq \hat{z}_{i,j}, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i, \\ & (u_{i,j} - l_{i,j}) z_{i,j} - u_{i,j} \hat{z}_{i,j} \leq -u_{i,j} l_{i,j}, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i \end{aligned} \tag{9}$$

3.3 KKT Conditions

3.3.1 Primal feasibility

$$\begin{aligned} z_1 - x - \epsilon \leq 0 \\ -z_1 + x - \epsilon \leq 0 \\ -z_{i,j} \leq 0, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i \\ \hat{z}_{i,j} - z_{i,j} \leq 0, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i \\ (u_{i,j} - l_{i,j}) z_{i,j} - u_{i,j} \hat{z}_{i,j} + u_{i,j} l_{i,j} \leq 0, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i \\ z_{i,j} = 0, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i^- \\ z_{i,j} = \hat{z}_{i,j}, \quad i = 2, \ldots, k-1, j \in \mathcal{I}_i^+ \end{aligned}$$

3.3.2 Dual feasibility

$$\begin{aligned} \xi^+ \geq 0, \\ \xi^- \geq 0, \\ \forall 2 \leq i \leq k-1, j \in \mathcal{I}, \tau_{i,j} \geq 0, \lambda_{i,j} \geq 0, \mu_{i,j} \geq 0 \end{aligned}$$

3.3.3 Stationary

$$\begin{aligned} \nabla \mathcal{L}(\hat{z}, z, v, p, q, \lambda, \tau, \mu, \xi^+, \xi^-) = 0 \\ \iff \nabla_{\hat{z}} \mathcal{L}(\hat{z}, z, v, p, q, \lambda, \tau, \mu, \xi^+, \xi^-) = 0 \text{ and } \nabla_z \mathcal{L}(\hat{z}, z, v, p, q, \lambda, \tau, \mu, \xi^+, \xi^-) = 0 \end{aligned}$$

See equation 10 for full form of the gradient.

3.3.4 Complementary slackness

$$\begin{aligned} \forall 1 \leq i \leq |x|, \quad \xi_i^+ (z_1 - x - \epsilon) = 0. \\ \forall 1 \leq i \leq |x|, \quad \xi_i^- (-z_1 + x + \epsilon) = 0. \\ \forall 2 \leq i \leq k-1, j \in \mathcal{I}_i, \quad \mu_{i,j}(-z_{i,j}) = 0. \\ \forall 2 \leq i \leq k-1, j \in \mathcal{I}_i, \quad \tau_{i,j}(\hat{z}_{i,j} - z_{i,j}) = 0. \\ \forall 2 \leq i \leq k-1, j \in \mathcal{I}_i, \quad \lambda_{i,j}[(u_{i,j} - l_{i,j}) z_{i,j} - u_{i,j} \hat{z}_{i,j} + u_{i,j} l_{i,j}] = 0. \end{aligned}$$

3.4 Dual Formulation

We define our dual variables corresponding to constraints as follows:

$$
\begin{aligned}
\hat{z}_{i+1} = W_i z_i + b_i \Rightarrow v_{i+1} \in \mathbb{R}^{|\hat{z}_{i+1}|}, &\quad i = 1, ..., k-1 \\
z_1 \leq x + \epsilon \Rightarrow \xi^+ \in \mathbb{R}^{|x|} & \\
-z_1 \leq -x + \epsilon \Rightarrow \xi^- \in \mathbb{R}^{|x|} & \\
z_{i,j} = 0 \Rightarrow p_{i,j} \in \mathbb{R}, &\quad i = 2, ..., k-1, j \in \mathcal{I}_i^- \\
z_{i,j} = \hat{z}_{i,j} \Rightarrow q_{i,j} \in \mathbb{R}, &\quad i = 2, ..., k-1, j \in \mathcal{I}_i^+ \\
-z_{i,j} \leq 0 \Rightarrow \mu_{i,j} \in \mathbb{R}, &\quad i = 2, ..., k-1, j \in \mathcal{I}_i \\
\hat{z}_{i,j} - z_{i,j} \leq 0 \Rightarrow \tau_{i,j} \in \mathbb{R}, &\quad i = 2, ..., k-1, j \in \mathcal{I}_i \\
(u_{i,j} - l_{i,j}) z_{i,j} - u_{i,j} \hat{z}_{i,j} \leq -u_{i,j} l_{i,j} \Rightarrow \lambda_{i,j} \in \mathbb{R}, &\quad i = 2, ..., k-1, j \in \mathcal{I}_i
\end{aligned}
$$

Then we can write out the Lagrangian:

$$
\begin{aligned}
&\mathcal{L}(\hat{z}, z, v, p, q, \lambda, \tau, \mu, \xi^+, \xi^-) = \\
&\qquad c^T \hat{z}_k + \sum_{i=1}^{k-1} v_{i+1}^T (\hat{z}_{i+1} - W_i z_i - b_i) + \xi^+ (z_i - x - \epsilon) + \xi^- (-z_i + x - \epsilon) \\
&+ \sum_{i=2, j \in \mathcal{I}_i^-}^{k-1} p_{i,j} z_{i,j} + \sum_{i=2, j \in \mathcal{I}_i^+}^{k-1} q_{i,j} (z_{i,j} - \hat{z}_{i,j}) + \sum_{i=2, j \in \mathcal{I}_i}^{k-1} -\mu_{i,j} z_{i,j} + \sum_{i=2, j \in \mathcal{I}_i^+}^{k-1} \tau_{i,j} (-z_{i,j} + \hat{z}_{i,j}) \\
&\qquad\qquad + \sum_{i=2, j \in \mathcal{I}_i^+}^{k-1} \lambda_{i,j} [(u_{i,j} - l_{i,j}) z_{i,j} - u_{i,j} \hat{z}_{i,j} + u_{i,j} l_{i,j}]
\end{aligned}
$$

Taking gradient of each variable, we get the following equations:

$$
\nabla \mathcal{L}(\hat{z}, z, v, p, q, \lambda, \tau, \mu, \xi^+, \xi^-) = \left[\begin{array}{c}
\frac{\partial f}{\partial \hat{z}_k} = c^T + v_k^T \\
\frac{\partial f}{\partial z_{i,j}} = p_{i,j} - (W_i^T v_{i+1})_j, \quad i = 2, ..., k-1, j \in \mathcal{I}_i^- \\
\frac{\partial f}{\partial \hat{z}_{i,j}} = v_{i,j}, \quad i = 2, ..., k-1, j \in \mathcal{I}_i^- \\
\frac{\partial f}{\partial z_{i,j}} = q_{i,j} - (W_i^T v_{i+1})_j, \quad i = 2, ..., k-1, j \in \mathcal{I}_i^+ \\
\frac{\partial f}{\partial \hat{z}_{i,j}} = v_{i,j} - q_{i,j}, \quad i = 2, ..., k-1, j \in \mathcal{I}_i^+ \\
\frac{\partial f}{\partial z_1} = -(W_1^T v_2) + \xi^+ - \xi^- \\
\frac{\partial f}{\partial z_{i,j}} = -(W_i^T v_{i+1})_j - \mu_{i,j} - \tau_{i,j} + \lambda_{i,j} (u_{i,j} - l_{i,j}), \quad i = 2, ..., k-1, j \in \mathcal{I}_i \\
\frac{\partial f}{\partial \hat{z}_{i,j}} = v_{i,j} + \tau_{i,j} - \lambda_{i,j} u_{i,j}, \quad i = 2, ..., k-1, j \in \mathcal{I}_i
\end{array} \right] \tag{10}
$$

By solving$\nabla\mathcal{L}(\hat{z}, z, v, p, q, \lambda, \tau, \mu, \xi^+, \xi^-) = 0$, we get our constraints for the dual problem (all i, j here corresponds to i, j above):

$$\begin{aligned}
v_k &= -c \\
p_{i,j} &= (W_i^T v_{i+1})_j \\
v_{i,j} &= 0 \\
q_{i,j} &= (W_i^T v_{i+1})_j \\
v_{i,j} &= q_{i,j} = (W_i^T v_{i+1})_j \\
W_1^T v_2 &= \xi^+ - \xi^- \\
(W_i^T v_{i+1})_j &= -\mu_{i,j} - \tau_{i,j} + \lambda_{i,j}(u_{i,j} - l_{i,j}) \\
v_{i,j} &= -\tau_{i,j} + \lambda_{i,j} u_{i,j}
\end{aligned}$$

Finally we get the dual form:

$$\begin{aligned}
\text{maximize} \quad & -\sum_{i=1}^{k-1} v_{i+1}^T b_i - (x+\epsilon)^T \xi^+ + (x-\epsilon)^T \xi^- + \sum_{i=2}^{k-1} \lambda_i^T (u_i l_i) \\
\text{subject to} \quad & v_k = -c, \\
& (W_1^T v_2) = \xi^+ - \xi^-, \\
v_{i,j} = 0, \quad & i = 2, \ldots, k-1, j \in \mathcal{I}_i^-, \\
v_{i,j} = (W_i^T v_{i+1})_j, \quad & i = 2, \ldots, k-1, j \in \mathcal{I}_i^+, \\
z_{i,j} \geq 0, \quad & i = 2, \ldots, k-1, j \in \mathcal{I}_i, \\
(W_i^T v_{i+1})_j = -\mu_{i,j} - \tau_{i,j} + \lambda_{i,j}(u_{i,j} - l_{i,j}), \quad & i = 2, \ldots, k-1, j \in \mathcal{I}_i, \\
v_{i,j} = -\tau_{i,j} + \lambda_{i,j} u_{i,j}, \quad & i = 2, \ldots, k-1, j \in \mathcal{I}_i, \\
& \lambda, \tau, \mu, \xi^+, \xi^- \geq 0
\end{aligned} \tag{11}$$

3.5 Feasible Solution of the Dual Problem Through Dual Network

For our primal problem $c^T \hat{z}_k$, instead of finding the optimal solution, we find lower bound which can be used for verification. Specifically, if the lower bound is positive, then we know that no norm-bounded adversarial perturbation of the input could change our output. The weak duality theorem states that any feasible solution from the dual problem gives us a lower bound to the primal. Most importantly, we can reformulate our dual problem to be a neural network that is similar to the standard back-propagation neural network.

Consider constraints:

$$(W_i^T v_{i+1})_j = -\mu_{i,j} - \tau_{i,j} + \lambda_{i,j}(u_{i,j} - l_{i,j}), \quad v_{i,j} = -\tau_{i,j} + \lambda_{i,j} u_{i,j}, \quad \text{where} \quad j \in \mathcal{I}_i$$

We know that λ corresponds to the upper bound of the ReLU relaxation, and μ, τ corresponds to the two lower bounds. Either one bound is tight, and the bound point is achieved at optimum. So we know that either $\lambda = 0$, or $\mu + \tau = 0$.

We define $[x]_+$ to be $\max(x, 0)$ and $[x]_-$ to be $-\min(x, 0)$ (Note that $[x]_+ + [x]_- = x$). The first constraint can therefore be written as

$$\lambda_{i,j}(u_{i,j} - l_{i,j}) = [(W_i^T v_{i+1})_j]_+, \quad \mu_{i,j} + \tau_{i,j} = [(W_i^T v_{i+1})_j]_-$$

Combining this with the second constraint we get

$$v_{i,j} = \frac{u_{i,j}}{u_{i,j} - l_{i,j}} [(W_i^T v_{i+1})_j]_+ - \alpha [(W_i^T v_{i+1})_j]_-$$

where $j \in \mathcal{I}_i$ and $\alpha = \frac{\tau}{\mu + \tau}$ is a variable between 0 and 1 representing the weight of τ over $\tau + \mu$. Similarly, we can replace ξ^+ and ξ^- with $[W_1^T v_2]_+$ and $[W_1^T v_2]_-$ since each represents the positive and negative portion of $W_1^T v_2$.

Another approach we can use to show this result is complementary slackness. Notice that according to complementary slackness, we have $\xi^+(z_1 - x - \epsilon) = 0, \quad \xi^-(-z_1 + x + \epsilon) = 0$. If $\xi^+ \neq 0$, then $z_1 - x - \epsilon = 0$, then $-z_1 + x + \epsilon \neq 0$, so $\xi^- = 0$. Similarly, $\xi^- \neq 0$ gives $\xi^+ = 0$, so either one of them must be zero. Combining with $W_1^T v_2 = \xi^+ - \xi^-$, we get the same result.

Next, we can rewrite our dual problem in terms of neural network so that we can easily find its feasible solution.

Define $v = g_\theta(c, \alpha)$ to be a k layer neural network given by the following equations. Note, we introduce substitute variable $\hat{v}_i$ to replace $W_i^T v_{i+1}$, for ease of notation.

$$v_k = -c \tag{12}$$

$$\hat{v}_i = W_i^T v_{i+1}, \quad \forall i = k-1, ..., 1 \tag{13}$$

$$v_{i,j} = \begin{cases} 0 & j \in \mathcal{I}_i^- \\ \hat{v}_{i,j} & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - l_{i,j}} [\hat{v}_{i,j}]_+ - \alpha_{i,j} [\hat{v}_{i,j}]_- & j \in \mathcal{I}_i \end{cases} \tag{14}$$

Equation 12 can be thought of as the forward propagation of a neural network since the input is $v_k = -c$, from v_k, we get $\hat{v_{k-1}} = W_{k-1}^T v_k$, which can be thought of as going through a linear layer parametrized by W_{k-1}. Equation 14 can be though of as a leaky-ReLU activation function. Thus, the above structure can be represented as a neural network piggybacking the original network from top to bottom with propagation.

Our dual problem becomes

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & J_\epsilon(x, g_\theta(c, \alpha)) \\ \text{subject to} \quad & \alpha_{i,j} \in [0,1], \forall i, j \end{aligned} \tag{15}$$

where

$$J_\epsilon(x, v) = -\sum_{i=1}^{k-1} v_{i+1}^T b_i - x^T \hat{v}_1 - \epsilon ||\hat{v}_1||_1 + \sum_{i=2}^{k-1} \sum_{j \in \mathcal{I}_i} l_{i.j} [v_{i,j}]_+$$

Notice that our dual network is almost the same of the original network, except that we have an additional α representing the weight of τ that we can optimize over. See section 4.5 for a possible choice of α.

3.6 Computing Activation Bound

We are now left to compute the lower and upper bounds for the pre-ReLU activations, l and u. Since $J_\epsilon(x, g_\theta(c))$ is the dual for our primal problem $c^T \hat{z}_k$, $J_\epsilon(x, g_\theta(c))$ provides a lower bound for $c^T \hat{z}_k$. If we plug in $c = I$ and $c = -I$ (here $\bar{J}$ becomes a vector and it provides bounds for $\hat{z}_k$ in an element-wise manner), we can get:

$$\bar{J}_\epsilon(x, g_\theta(I)) \leq \hat{z_k} \leq -\bar{J}_\epsilon(x, g_\theta(-I))$$

As mentioned in section 2.5 that the structure of $g_\theta(c, \alpha)$ is very similar to the original feed-forward neural network, we can therefore calculate J using a backward pass.

Specifically, for $c = I$, the backward pass variables are given by

$$\hat{\nu}_i = -W_i^T D_{i+1} W_{i+1}^T \dots D_n W_n^T, \; \nu_i = D_i \hat{\nu}_i \tag{16}$$

where

$$(D_i)_{jj} = \begin{cases} 0, & j \in \mathcal{I}_i^- \\ 1, & j \in \mathcal{I}_i^+ \\ \frac{u_{i,j}}{u_{i,j} - l_{i,j}}, & j \in \mathcal{I}_i \end{cases} \tag{17}$$

Proof. Since we fixed $\alpha_{i,j} = \frac{u_{i,j}}{u_{i,j}-l_{i,j}}$, when $j \in \mathcal{I}_i$, we have $\nu_{i,j} = \frac{u_{i,j}}{u_{i,j}-l_{i,j}}[\hat{\nu}_{i,j}]_+ - \frac{u_{i,j}}{u_{i,j}-l_{i,j}}[\hat{\nu}_{i,j}]_- = \frac{u_{i,j}}{u_{i,j}-l_{i,j}}(max(0,\hat{\nu}_{i,j}) - |min(0,\hat{\nu}_{i,j})|) = \frac{u_{i,j}}{u_{i,j}-l_{i,j}}\hat{\nu}_{i,j}$. By setting D_i as (17), we can represent the transition constraint from $\hat{\nu}_{i,j}$ to $\nu_{i,j}$ easily through a single matrix multiplication using D_i.

Then we can use induction to prove (16). Assume $g_\theta(c,\alpha)$ is a k layer feed-forward neural network, and setting $\alpha = \frac{u_{i,j}}{u_{i,j}-l_{i,j}}$, $c = I$.

Base case: $\nu_k = -c = -I$, $\hat{\nu}_{k-1} = W_{k-1}^T \nu_k = -W_{k-1}^T$, $\nu_{k-1} = D_{k-1}\hat{\nu}_{k-1}$ satisfies (16).

Assume (16) is satisfied for $i = k', ..., k-1$, then $\hat{\nu}_{k'} = -W_{k'}^T D_{k'+1} W_{k'+1}^T \dots D_{k-1} W_{k-1}^T$, $\nu_{k'} = D_{k'}\hat{\nu}_{k'}$. For $k'-1$, we have

$$\hat{\nu}_{k'-1} = W_{k'-1}\nu_{k'} = W_{k'-1} D_{k'} \hat{\nu}_{k'} = -W_{k'-1} D_{k'} W_{k'}^T D_{k'+1} W_{k'+1}^T \dots D_{k-1} W_{k-1}^T$$

and

$$\nu_{k'-1} = D_{k'-1}\hat{\nu}_{k'-1}$$

both satisfies (16) for $2 \le k' \le k-1$.

□

Algorithm to compute l and u is described in (0). The algorithm is doing a forward pass through the network. The takeaway is that, it computes l_i and u_i for the current layer's output $\hat{z}_i$ with previously computed l_j and $u_j (j < i)$. At iteration i, it treats the 1^{st} to i^{th} layers as an isolated sub-network, setting $v_i = \pm I$, and perform a backward pass as $g_W(\pm I)$ to compute the current lower and upper bounds. Repeating such iteration all the way to the final layer, we can obtain all the l_i and u_i and use them to compute the dual J_ϵ. Figure 1 provides a visualization of Algorithm 1 at iteration i.

Algorithm 1 Computing Activation Bounds via Backward Pass

procedure (Computing Activation Bounds)
  // initialization
  $\hat{\nu}_1 \leftarrow W_1^T$
  $\gamma_1 \leftarrow b_1^T$
  $l_2 \leftarrow x^T \hat{\nu}_1 + \gamma_1 - \epsilon \|\hat{\nu}_1\|_{1,:}$
  $u_2 \leftarrow x^T \hat{\nu}_1 + \gamma_1 + \epsilon \|\hat{\nu}_1\|_{1,:}$
  // $\|\cdot\|_{1,:}$ for a matrix denotes l_1 norm of all columns
  for i = 2 to k-1 **do** ▷ Here, i is the index of the layers.
    form $\mathcal{I}_i^-, \mathcal{I}_i^+, \mathcal{I}_i$
    form D_i
    // initialize new terms
    $\nu_{i,\mathcal{I}_i} \leftarrow (D_i)_{\mathcal{I}_i} W_i^T$
    $\gamma_i \leftarrow b_i^T$
    // propagate existing terms backward
    for j = 2 to i-1 **do**
      $\nu_{j,\mathcal{I}_j} \leftarrow \nu_{j,\mathcal{I}_j} D_i W_i^T$
      $\gamma_j \leftarrow \gamma_j D_i W_i^T$
    end for
    $\gamma_1 \leftarrow \gamma_1 D_i W_i^T$
    $\hat{\nu}_1 \leftarrow \hat{\nu}_1 D_i W_i^T$
    // compute bounds
    $\psi_i \leftarrow x^T \hat{\nu}_1 + \sum_{j=1}^{i} \gamma_j$
    $l_{i+1} \leftarrow \psi_i - \epsilon \|\hat{\nu}_1\|_{1,:} + \sum_{j=2}^{i} \sum_{i' \in \mathcal{I}_i} ([-v_{j,i'}]_+) l_{j,i'}$
    $u_{i+1} \leftarrow \psi_i + \epsilon \|\hat{\nu}_1\|_{1,:} - \sum_{j=2}^{i} \sum_{i' \in \mathcal{I}_i} ([v_{j,i'}]_+) l_{j,i'}$
  end for
  return $\{l_i, u_i\}_{i=2}^k$
end procedure

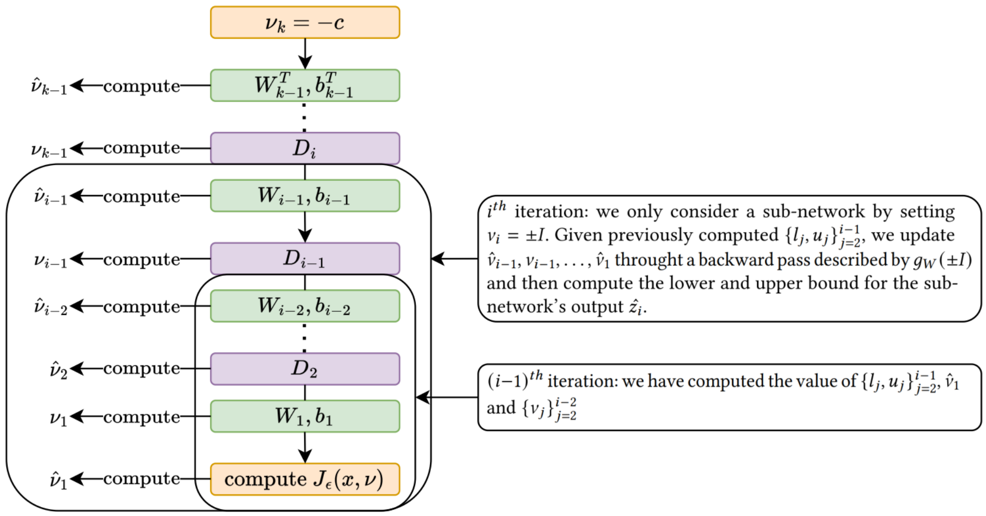

Figure 1: Visualization of Algorithm 1

4 Generalized Dual Network

In this section, we present a modular formulation of constructing the aforementioned dual network for dual variables v. For notation simplicity, we use

$$v_{i:j} = \{v_i, v_{i+1}, \ldots, v_j\}$$

to denote a collection of vector (tensor) variables (primal or dual) across different layers. For each layer, we calculate its dual layer (h, g) where h is a functional upper bound and g is a functional condition for h. Together, a summation of different hs from different layers form our lower bound objective J. The collection of different gs form the structure of our dual network.

In this section, we generalize the notion of network layer by incorporating residual connections (which are the characteristics of successful deep networks since they effectively avoid the gradient vanishing/exploding problem). To be precise,

$$f_{ij}(\cdot) := \text{dom(Layer } j) \subseteq \mathbb{R}^{|z_j|} \longrightarrow \text{dom(Layer } i) \subseteq \mathbb{R}^{|z_i|}, \quad j < i$$

f_{ij} takes the value at layer j and maps it into layer i.

In Section 2 and 3, we only use $f_{(i+1,i)}$ for layer-by-layer forward propagation, but in Section 4 we can expand to cross layer propagation (e.g. residual connections). Our primal formulation in Section 2 is thus generalized.

4.1 Generalized Primal Problem Formulation

For a k-layer network, where the i^{th} layer value is given by:

$$z_i = \sum_{j=1}^{i-1} f_{ij}(z_j), \quad i = 2, \ldots, k \tag{18}$$

Note, we dispense with the notion of activation function, pre-activation and post-activation. We view activation as $f_{(i+1,i)}$ that takes value from layer i^{th} (pre-activation) to layer $(i+1)^{\text{th}}$ (post-activation).

Given datapoint (x, y), our generalized primal formulation is:

$$\min_{z_k} c^T z_k; \quad c = e_{y^\star} - e_{y^{\text{targ}}} \tag{19}$$

$$\text{s.t.} \quad z_i = \sum_{j=1}^{i-1} f_{ij}(z_j), \quad i = 2, \ldots, k. \tag{20}$$

$$\text{s.t.} \quad ||z_1 - x|| \leq \epsilon \Longleftrightarrow z_1 \in B_\epsilon(x) \tag{21}$$

For mathematical convenience, we put the last inequality (though convex because norm is convex) directly into the objective and do not count it as an explicit constraint:

$$\min_{z_1, z_k} c^T z_k + I_{B_\epsilon(x)}(z_1); \quad c = e_{y^\star} - e_{y^{\text{targ}}} \tag{22}$$

$$\text{s.t.} \quad z_i = \sum_{j=1}^{i-1} f_{ij}(z_j), \quad i = 2, \ldots, k. \tag{23}$$

$$\tag{24}$$

where $I_{B_\epsilon(x)}$ is an indicator function, defined as :

$$I_{B_\epsilon(x)}(z_1) = \begin{cases} 0 & \text{if } z_1 \in B_\epsilon(x) \Longleftrightarrow ||z_1 - x|| \leq \epsilon \\ \infty & \text{otherwise} \end{cases}$$

4.2 Generalized Dual Formulation

We first convert the primal problem into a Lagrangian by introducing dual variables $v_{2:k}, v_1$.

$$\begin{aligned}
L(z_{1:k}, v_{1:k}) &= c^T z_k + I_{B_\epsilon(x)}(z_1) + \sum_{i=2}^{k} v_i^T (z_i - \sum_{j=1}^{i-1} f_{ij}(z_j)) = c^T z_k + I_{B_\epsilon(x)}(z_1) + \sum_{i=2}^{k} (v_i^T z_i - \sum_{j=1}^{i-1} v_i^T f_{ij}(z_j)) \\
&= c^T z_k + I_{B_\epsilon(x)}(z_1) + \sum_{i=2}^{k} v_i^T z_i - \sum_{i=2}^{k} \sum_{j=1}^{i-1} v_i^T f_{ij}(z_j) = c^T z_k + I_{B_\epsilon(x)}(z_1) + \sum_{i=2}^{k} v_i^T z_i - \sum_{j=1}^{k-1} \sum_{i=j+1}^{k} v_i^T f_{ij}(z_j) \\
&= c^T z_k + + I_{B_\epsilon(x)}(z_1) \sum_{j=2}^{k} v_j^T z_j - \sum_{j=1}^{k-1} \sum_{i=j+1}^{k} v_i^T f_{ij}(z_j) \\
&= c^T z_k + I_{B_\epsilon(x)}(z_1) + v_k^T z_k - \sum_{i=2}^{k} v_i^T f_{i1}(z_1) + \sum_{j=2}^{k-1} \left(v_j^T z_j - \sum_{i=j+1}^{k} v_i^T f_{ij}(z_j) \right) \\
&= c^T z_k + I_{B_\epsilon(x)}(z_1) + v_k^T z_k - v_1^T z_1 + (v_1^T z_1 - \sum_{i=2}^{k} v_i^T f_{i1}(z_1)) + \sum_{j=2}^{k-1} \left(v_j^T z_j - \sum_{i=j+1}^{k} v_i^T f_{ij}(z_j) \right) \\
&= c^T z_k + I_{B_\epsilon(x)}(z_1) + v_k^T z_k - v_1^T z_1 + \sum_{j=1}^{k-1} \left(v_j^T z_j - \sum_{i=j+1}^{k} v_i^T f_{ij}(z_j) \right)
\end{aligned}$$

We minimize $L(z, v)$ over z. We first make the observation from $(c^T + v_k^T)z_k$ that $c_k = -c$ by stationarity of the gradient w.r.t. z_k.

$$\min_{z_{1:k}} L(z_{1:k}, v_{1:k}) \geq \min_{z_1} \left(I_{B_\epsilon(x)}(z_1) - v_1^T z_1 \right) + \sum_{j=1}^{k-1} \min_{z_j} \left(v_j^T z_j - \sum_{i=j+1}^{k} v_i^T f_{ij}(z_j) \right)$$

We further note that

$$\min_{z_1}\Big(I_{B_\epsilon(x)}(z_1) - v_1^T z_1\Big) = \min_{z_1} I_{B_\epsilon(x)}(z_1) - v_1^T(z_1 + x - x) = \min_{z_1} I_{B_\epsilon(x)}(z_1) - v_1^T x - v_1^T(z_1 - x)$$
$$= -v_1^T x - \max_{||z_1 - x|| \le \epsilon} v_1^T(z_1 - x) = -v_1^T x - \epsilon||v_1||_* \quad ;||\cdot||_* \text{ is the dual norm by definition}$$

As mentioned in the beginning of this chapter, suppose we know:

$$\min_{z_j}\Big(v_j^T z_j - \sum_{i=j+1}^{k} v_i^T f_{ij}(z_j)\Big) \ge -h_j(v_{j:k}) \iff \max_{z_j}\Big(-v_j^T z_j + \sum_{i=j+1}^{k} v_i^T f_{ij}(z_j)\Big) \le h_j(v_{j:k})$$
$$\text{given } v_j = \sum_{i=j+1}^{k} g_{ji}(v_i) \quad \text{where } \{g_{ji}\}_{ji} \text{ is the dual network}$$

Then we have:

$$\min_{z_{1:k}} L(z_{1:k}, v_{1:k}) \ge -v_1^T x - \epsilon||v_1||_* - \sum_{j=1}^{k-1} h_j(v_{j:k})$$

The dual formulation then becomes

$$J_\epsilon(x, v_{1:k}) = \max_{v_{1:k}} -v_1^T x - \epsilon||v_1||_* - \sum_{j=1}^{k-1} h_j(v_{j:k})$$
$$\text{subject to: } v_k = -c, v_j = \sum_{i=j+1}^{k} g_{ji}(v_i)$$

The parallel structures of the primal and dual network can be visualized in Figure 2, where we showcases cross-layer connections such as residual connections can be easily represented.

In the next few subsections, we will derive the dual layer for several common neural network layers.

4.3 The Dual Layer for Linear Layer

A linear layer is characterized by

$$f_{(j+1,j)}(z_j) = W_j z_j + b_j$$

$$\max_{z_j} -v_j^T z_j + v_{j+1}^T f_{(j+1,j)}(z_j) = -v_j^T z_j + v_{j+1}^T(W_j z_j + b_j) = (-v_j^T + v_{j+1}^T W_j)z_j + v_{j+1}^T b_j$$
$$= 0 + v_{j+1}^T b_j \quad \text{subject to } v_j = v_{j+1}^T W_j$$

4.4 The Dual Layer for Residual Linear Layer

Suppose from layer j there is a residual connection to layer i (i.e. z_j is copied to z_i), and layer j also has a normal linear connection to layer $(j+1)$

$$\max_{z_j} -v_j^T z_j + v_{j+1}^T f_{(j+1,j)}(z_j) + v_i^T z_j$$
$$= (v_i^T - v_j^T + v_{j+1}^T W_j)z_j + v_{j+1}^T b_j$$
$$= 0 + v_{j+1}^T b_j \quad \text{subject to } v_j^T = v_i^T + v_{j+1}^T W_j$$

Note that v_i (which is at an upper layer) is directly copied down to layer j, which is a lower layer through the dual of the residual connection (which is also the identity operator).

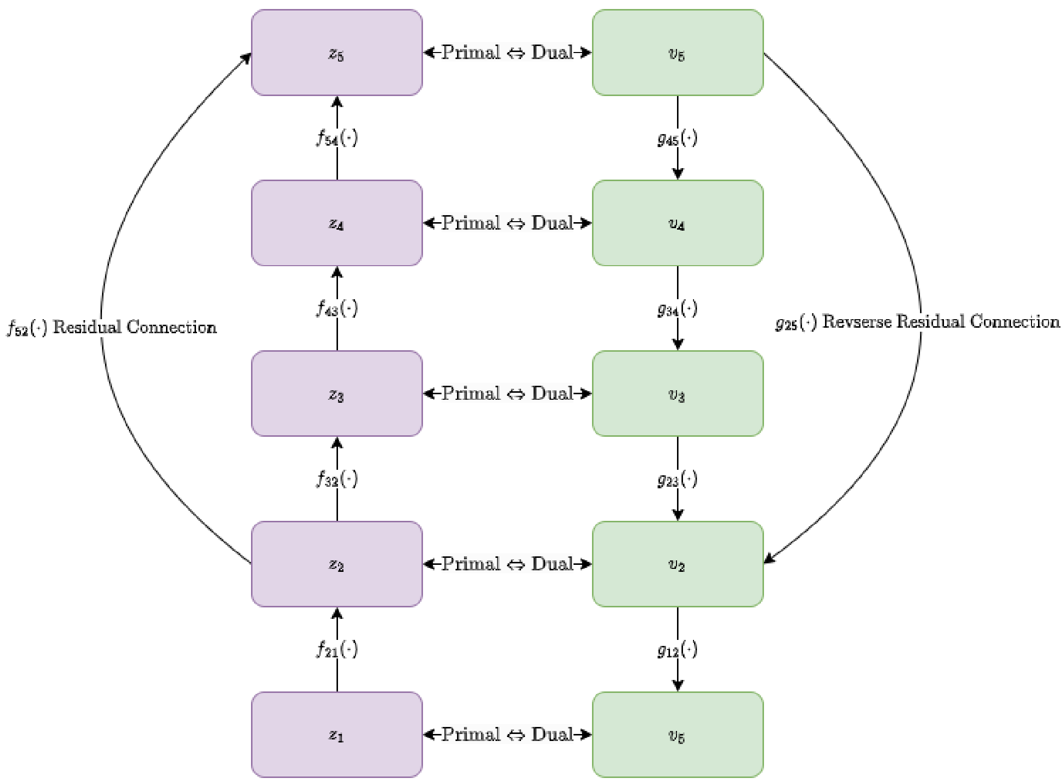

Figure 2:

4.5 The Dual Layer of ReLU Layer

The ReLU layer can be represented as a connection from layer j to layer $(j+1)$.

$$\max_{z_j} -v_j^T z_j + v_{j+1}^T \mathrm{ReLU}(z_j) = \max_{z_j} -v_j^T z_j + v_{j+1}^T \max(z_j, 0)$$

Suppose we know $l_j \leq z_j \leq u_j$. If $u_i \leq 0$, then $\max(z_i, 0) = 0$, and so

$$\max_{z_j} -v_j^T z_j + v_{j+1}^T \max(z_j, 0) = \max_{z_j} -v_j^T z_j = 0 \text{ subject to } v_j = 0$$

Otherwise, if $l_i \geq 0$, then $\max(z_j, 0) = z_j$ and we have

$$\max_{z_j} -v_j^T z_j + v_{j+1}^T \max(z_j, 0) = (v_{j+1} - v_j)^T z_j = 0 \quad \text{subject to } v_{j+1} = v_j$$

Lastly, suppose $l_{ji} < 0 < u_{ji}$.

$$\max_{z_j} \Big(-v_j^T z_j + v_{j+1}^T \text{ReLU}(z_j) \Big) = \sum_i \max_{z_{ji}} -v_{ji}^T z_{ji} + v_{j+1,i} \text{ReLU}(z_{ji})$$

$$\le \max_{z_{ji}} -v_{ji}^T z_{ji} + v_{j+1,i} \frac{u_{ji}}{u_{ji} - l_{ji}} (z_{ji} - l) = \max_{z_{ji}} z_{ji} (-v_{ji} + \frac{u_{ji} v_{j+1,i}}{u_{ji} - lji}) - \frac{u_{ji} l_{ji}}{u_{ji} - l_{ji}} v_{j+1,i}$$

$$= -\frac{u_{ji} l_{ji}}{u_{ji} - l_{ji}} v_{j+1,i} \text{ subject to } v_{ji} = \frac{u_{ji}}{u_{ji} - l_{ji}} v_{j+1,i}$$

4.6 The AutoDual Algorithm

Bound computation defined in Algorithm 1 has two main weakness. First, it assumes the network is fully connected with ReLU followed by each affine transformation, and it's not able to handle other non-linearity like max-pooling, normalization, etc. Second, it assumes the network is connected sequentially and only depends on the previous layer, and it cannot handle residual/skip connections. To generalize the bound computation and robust training in [24], instead of explicitly specifying the linear and ReLU layers, [25] proposed a new representation of general network architecture by treating it as an arbitrary sequence of k functions defined as below.

With dual layers, we can generalize the bound computation algorithm in 0 to general networks with residual/skip connections. Specifically, if the operators g_{ij} of the dual layers are all affine operators $g_{ij}(\nu_j) = A_{ij}^T v_j$ for some affine operator A_{ij}, we can compute pre-function bounds $\{l_i, u_i\}_{i=2}^k$ and dual layer $\{h_i\}_{i=1}^{k-1}$ through a single pass of the network layer by layer.

Algorithm 2 AutoDual algorithm

```
procedure (AutoDual)
    // initialization
    ν_1 ← I
    l_2 ← x − ε
    u_2 ← x + ε
    for i = 2, 3, ..., k do                          ▷ Here, i is the index of the layers.
        // initialize new dual layer
        form A_ij and h_i from f_ij, l_j and u_j for all j ≥ i
        ν_i ← I                      ▷ At the current loop, we only consider 1 to i th layers
        // updates dual variables
        for j = i-1, i-2, ... 1 do
            ν_j ← Σ_{p=j+1}^{i} A_jp ν_p
        end for
        // compute bounds
        l_{i+1} ← x^T ν_1 − ε‖ν_1‖_: + Σ_{j=1}^{i} h_j(ν_{j:i})
        u_{i+1} ← x^T ν_1 + ε‖ν_1‖_: − Σ_{j=1}^{i} h_j(−ν_{j:i})
    end for
    return {l_i, u_i}_{i=2}^k, {A_ij}, {h_i}
end procedure
```

We notice that both Algorithm 1 and AutoDual are not computationally efficient. They first compute a forward pass through the network on I, resulting in k outer iterations. Second, whenever it encounters an activation layer, it needs to compute the bounds for that layer. In this case, assume the activation layer is at index i, it then treats the 1^{st} to i^{th} layers as an isolated sub-network, updating the νs and computing the lower and upper bounds for the pre-activation values through another pass from the 1^{st} layer to $(i-1)^{th}$ layer. The two algorithms both compute νs explicitly using the procedure described as above, while computing νs requires matrix multiplication which results in $\mathcal{O}(|z_i|^3)$ ($|z_i|$ is the number of hidden units in layer i). Thus, the total time complexity of algorithm 1 is $\mathcal{O}(k \sum_{i=2}^{k-1} |z_{i-1}||z_i|^2) = \mathcal{O}(k^2 |\bar{z}|^3)$. For AutoDual, since each layer not only depends on the previous layer but also the layers before, it takes $\mathcal{O}(k|\bar{z}|^3)$ when we update each ν, so the total time complexity of AutoDual would be $\mathcal{O}(k^3|\bar{z}|^3)$. However, the residual/skip connections are relatively rare in the network, so many of the $A_{ij}s$ would be 0. In such case, the time complexity of AutoDual would not be too much worse than Algorithm 1.

5 The Dual Network for Transformer Models

Last but not least, we derive the dual layer for the self-attention layers in the Transformer architecture, which has been the predominant approach in modern natural language processing and computer vision tasks [21]. We first give a mathematical formulation for the attention layer [21], which is arguably one of the most important building blocks in the Transformer model.

5.1 Self-Attention in Encoder

For the encoder in Transformer model, the methodology for self attention is:
Assume now we have batch size = 1, the input is $[x_1 \quad x_2 \quad \cdots \quad x_n]$, where x_i is a sub-word (token) id at ith position in sentence and n is the number of sub-words in sentence. After we input through embedding layer, each x_i is embedded by size E such that $\vec{x_i} \in \mathbb{R}^E$. We pack together all the embedding of input into a matrix Q, K, and V, where $Q = \begin{bmatrix} \vec{x_1} \\ \vec{x_2} \\ \vdots \\ \vec{x_n} \end{bmatrix} \in \mathbb{R}^{n \times E}$, and $Q = K = V$. Then

$$QK^T = \begin{bmatrix} \vec{x_1} \\ \vec{x_2} \\ \vdots \\ \vec{x_n} \end{bmatrix} [\vec{x_1} \quad \vec{x_2} \quad \cdots \quad \vec{x_n}] = \begin{bmatrix} <\vec{x_1}, \vec{x_1}> & <\vec{x_1}, \vec{x_2}> & \cdots & <\vec{x_1}, \vec{x_n}> \\ <\vec{x_2}, \vec{x_1}> & <\vec{x_2}, \vec{x_2}> & \cdots & <\vec{x_2}, \vec{x_n}> \\ \vdots & \ddots & & \\ <\vec{x_n}, \vec{x_1}> & <\vec{x_n}, \vec{x_2}> & \cdots & <\vec{x_n}, \vec{x_n}> \end{bmatrix} \in \mathbb{R}^{n \times n}$$

We perform softmax on each row of QK^T, where

$$\text{Softmax } (QK^T) = \begin{bmatrix} P(<\vec{x_1}, \vec{x_1}>) & P(<\vec{x_1}, \vec{x_2}>) & \cdots & P(<\vec{x_1}, \vec{x_n}>) \\ P(<\vec{x_2}), \vec{x_1}>) & P(<\vec{x_2}, \vec{x_2}>) & \cdots & P(<\vec{x_2}, \vec{x_n}>) \\ \vdots & \ddots & & \\ P(<\vec{x_n}, \vec{x_1}>) & P(<\vec{x_n}, \vec{x_2}>) & \cdots & P(<\vec{x_n}, \vec{x_n}>) \end{bmatrix} \in \mathbb{R}^{n \times n}$$

where $\sum_{j=1}^n P(<\vec{x_i}, \vec{x_j}>) = 1$ for ith row, $1 \leq i \leq n$.

$$\text{Softmax } (QK^T)V = \begin{bmatrix} \sum_{j=1}^n P(<\vec{x_1}, \vec{x_j}>)\vec{x_j} \\ \vdots \\ \sum_{j=1}^n P(<\vec{x_n}, \vec{x_j}>)\vec{x_j} \end{bmatrix} \in \mathbb{R}^{n \times E}$$

The softmax function is defined for the i[th] row as:

$$P(<x_i, x_j>) = \frac{\exp(<x_i, x_j>)}{\sum_{p=1}^n \exp(<x_i, x_p>)} \tag{25}$$

A simplified version (ignoring multiple weight projection matrices W_q, W_k, W_v, W_o) of the aforementioned self-attention layer can be represented as a value matrix X from the i^{th} layer to the $i+1^{\text{th}}$ layer.

$$Z = f_{i+1,i}(X) = \text{Softmax}(XX^T)X \tag{26}$$

5.2 The Dual Layer for Attention Layer

Note, since the input is no longer a vector but a matrix, our dual variables V_1, V_2 will correspondingly be changed to matrices of the appropriate shape. The dot product will be changed to the trace operator for matrices dot product. We use subscript 1 and 2 to avoid introducing too many lettered subscripts.

$$\max_X f(X) = \max_X - \operatorname{Tr}(V_1^T X) + \operatorname{Tr}(V_2^T \text{Softmax}(XX^T)X)$$

In order to maximize over X, we resort to taking the matrix gradient.

$$\begin{aligned}
\frac{\partial f}{\partial X} &= -V_1 + \frac{\partial \operatorname{Tr}(V_2^T \text{Softmax}(XX^T)X)}{\partial X} && (27)\\
&= -V_1 + \{\frac{\partial \operatorname{Tr}(V_2^T \text{Softmax}(XX^T)X)}{\partial X_{ij}}\}_{ij} && (28)\\
&= -V_1 + \{\operatorname{Tr}(\frac{\partial \operatorname{Tr}(V_2^T U)}{\partial U}\frac{\partial U}{\partial X_{ij}})\}_{ij} \quad ; U = \text{Softmax}(XX^T)X && (29)\\
&= -V_1 + \{\operatorname{Tr}\left(V_2^T \frac{\partial \text{Softmax}(XX^T)}{\partial X_{ij}} X + V_2^T \text{Softmax}(XX^T)\frac{\partial X}{\partial X_{ij}}\right)\}_{ij} && (30)\\
&= -V_1 + \{\operatorname{Tr}\left(V_2^T \frac{\partial \text{Softmax}(XX^T)}{\partial X_{ij}} X\right) + \operatorname{Tr}\left(V_2^T \text{Softmax}(XX^T)\frac{\partial X}{\partial X_{ij}}\right)\}_{ij} && (31)
\end{aligned}$$

$$\operatorname{Tr}(V_2^T \text{Softmax}(XX^T)\frac{\partial X}{\partial X_{ij}}) \tag{32}$$

$$= \operatorname{Tr}\left(\begin{bmatrix}\vec{V_{2_1}}\\ \vec{V_{2_2}}\\ \vdots \\ \vec{V_{2_j}}\\ \vdots\end{bmatrix}\begin{bmatrix} P(<\vec{x_1},\vec{x_1}>) & P(<\vec{x_1},\vec{x_2}>) & \cdots & P(<\vec{x_1},\vec{x_n}>)\\ P(<\vec{x_2}),\vec{x_1}>) & P(<\vec{x_2},\vec{x_2}>) & \cdots & P(<\vec{x_2},\vec{x_n}>)\\ \vdots & \ddots & & \\ P(<\vec{x_n},\vec{x_1}>) & P(<\vec{x_n},\vec{x_2}>) & \cdots & P(<\vec{x_n},\vec{x_n}>)\end{bmatrix}\begin{bmatrix}0 & 0 & \dots & 0\\ 0 & 0 & \dots & 0\\ \vdots & \vdots & 1_{ij} & 0\\ 0 & 0 & \dots & 0\end{bmatrix}\right) \tag{33}$$

$$=< V_{2_j}, \begin{bmatrix} P(<x_1,x_i>)\\ \vdots \\ P(<x_n,x_i>)\end{bmatrix} > \quad \text{the dot product between the } j \text{ column of } V_2 \text{ and the } i \text{ column of the Softmax matrix} \tag{34}$$

$$\frac{\partial \text{Softmax}(XX^T)_{kl}}{\partial X_{ij}} = \frac{\partial(\frac{\exp(x_k^T x_l)}{\sum_{p=1}^n \exp(x_k^T x_p)})}{\partial x_{ij}} \begin{cases} -\exp(x_k^T x_l) x_{kj} \frac{\exp(x_k^T x_i)}{\left(\sum_{p=1}^n \exp(x_k^T x_p)\right)^2} & i \neq k, i \neq l\\ \exp(x_k^T x_l) x_{kj} \frac{(\sum_{p\neq i}^n \exp(x_k^T x_p))}{\left(\sum_{p=1}^n \exp(x_k^T x_p)\right)^2} & i \neq k, i = l\\ \exp(x_k^T x_l) \frac{\sum_{p=1}^n \exp(x_k^T x_p)(x_{lj} - x_{pj})}{\left(\sum_{p=1}^n \exp(x_k^T x_p)\right)^2} & i = k, i \neq l\\ \exp(x_i^T x_i) \frac{\sum_{p=1}^n 2x_{ij}\exp(x_i^T x_p) - \sum_{p=1}^n x_{pj}\exp(x_i^T x_p)}{\left(\sum_{p=1}^n \exp(x_k^T x_p)\right)^2} & i = k = l \end{cases} \tag{35}$$

Plugging Equations 32, 35 into Equation 27, we can calculate $\frac{\partial f}{\partial X}$ for any data matrix. Note, in the middle of the network, X will naturally become $Z_i \in \mathbb{R}^{n \times E}$.

5.3 The Dual Layer for Structured Layer

Since traditional attention layers prevent an efficient computation of their dual layers, we propose to use other structured or sparsely parameterized transformation across the positional sequence dimension, which are empirically good substitute [13][26][16] for large transformer models in several NLP tasks including the GLUE[22], LRA[19] and machine translation WMT[3] benchmarks.

We introduce one of the several promising unparametrized token-mixing techniques: the Discrete Fourier Transform based encoder for Transformer model in sentiment analysis, other natural language understanding tasks, and machine translation [21].

5.3.1 Fourier Transform

The Fourier Transform converts a function into a form that describes the frequencies present in the original function. The discrete Fourier transform (DFT) as a transformation matrix at N-point is experessed as $X = Wx$ where x is the original input and W is the DFT matrix. The transformation matrix W_N for sentence DFT in encoder, W_M for DFT in decoder, and W_E for embedding DFT are

$$W_N = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix}$$

$$W_M = \frac{1}{\sqrt{M}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{M-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{M-1} & \omega^{2(M-1)} & \cdots & \omega^{(M-1)(M-1)} \end{bmatrix}$$

$$W_E = \frac{1}{\sqrt{E}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{E-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{E-1} & \omega^{2(E-1)} & \cdots & \omega^{(E-1)(E-1)} \end{bmatrix}$$

where $\omega = e^{-2\pi i/N}$ is a primitive Nth root of unity satisfied that $z^n = 1$ for number z. Thus, the ω is independent of the actual value of x, it only depend on the length of x and the position in sequence.

5.3.2 Fourier Transformer

Assume now we have batch size = 1, we have input sequence as $[x_1 \quad x_2 \quad \cdots \quad x_N]$ where x_i is a sub-word (token) id at ith position in sentence and n is the number of sub-words in sentence. We perform the Fast Fourier Transform (FFT) and matrix multiplication on Encoder part. After we input through embedding layer, we have embedded input as $X = \begin{bmatrix} \vec{x_0} \\ \vec{x_1} \\ \vdots \\ \vec{x_{n-1}} \end{bmatrix} \in \mathbb{R}^{N \times E}$. We perform DFT on both dimension of embedded input by multiplying W_N and W_E on left and right sides of X.

$$\frac{1}{\sqrt{NE}} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{N-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{N-1} & \omega^{2(N-1)} & \cdots & \omega^{(N-1)(N-1)} \end{bmatrix} \begin{bmatrix} \vec{x_0} \\ \vec{x_1} \\ \vdots \\ \vec{x_{n-1}} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ 1 & \omega & \omega^2 & \cdots & \omega^{E-1} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & \omega^{E-1} & \omega^{2(E-1)} & \cdots & \omega^{(E-1)(E-1)} \end{bmatrix}$$

After the DFT, the embedded input is transformed to

$$\left\{ (\sum_{k=0}^{N-1} W_{N_{ik}} \vec{x_k}) W{E_j}^T \right\}_{ij}$$

at ith row and jth column in the embedded input.

5.3.3 The Dual Layer for Fourier Transformer

The Fourier Transformation of the data matrix X can be represented as $f_{21}(X) = W_N X W_E$ assuming the transform layer happens at the first and the second layers.

$$
\begin{aligned}
&\max_X(-\operatorname{Tr}(V_1^T X) + \operatorname{Tr}({}_2^T W_N X W_E)) \\
&\nabla_X = -V_1 + \frac{\partial \operatorname{Tr}(W_E V_2^T W_N X)}{\partial X} = -V_1 + W_N^* V_2 W_E^* \\
&\nabla_X = 0 \Longleftrightarrow V_1 = W_N^* V_2 W_E^*; \quad \text{* means transpose of complex conjugate}
\end{aligned}
$$

6 Experiment Results

6.1 2D example

We repeated the 2D experiment in [24]. Consider a robust binary classifier on a 2D input space with randomly generated spread out data points. We used a 2-100-100-100-100-2 fully connected network to train the classifier, used standard training strategy and robust training strategy and compared their performance.

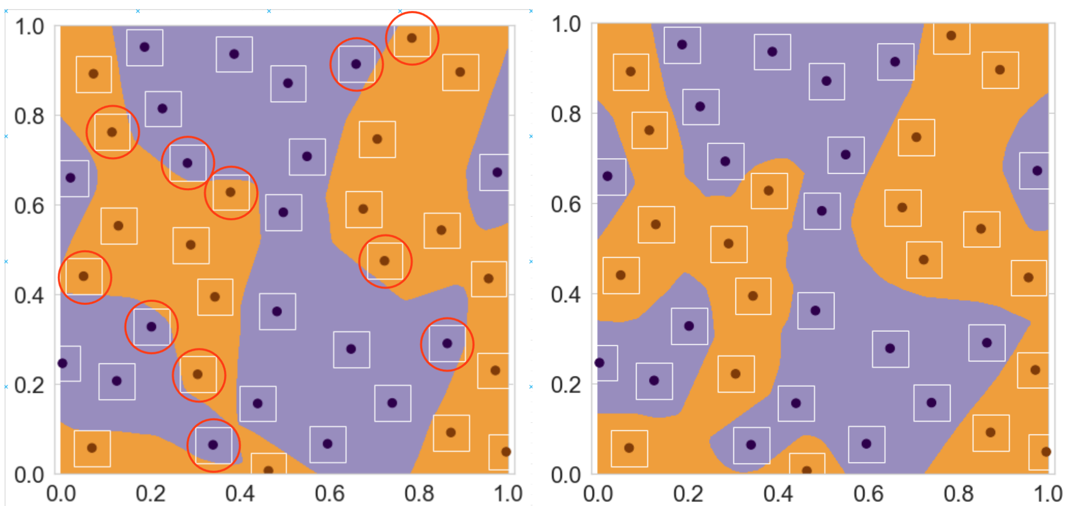

Figure 3: Visualization of 2D classifier with standard training and robust training, (Left: Standard training, Right: Robust training)

Figure 3 showcases the classification outcome of 40 two-dimensional scattered points. While both standard and robust training accurately classify all 40 points in the plane, the standard training classifier (left) exhibits lower robustness than the robust training classifier (right). Specifically, many of the points in the left figure (marked in red circles) exhibit incorrect classification within their l_∞ ball with $\epsilon = 0.04$. Such points are vulnerable to attacks via adversarial examples.

In contrast, the right figure demonstrates that the robust training classifier accurately classifies all the points within their l_∞ ball, providing guaranteed robustness against l_∞ adversarial attacks with $\epsilon = 0.04$. While robust training may slightly decrease accuracy in more complex examples, it significantly enhances each input's robustness against adversarial attacks.

6.2 MNIST

We trained a robust classifier on MNIST data set and compared the error and the loss with the standard classifier. Specifically, the classifiers have the same architecture. We first pass the image to two Convolutional layers with 16 and 32 channels respectively. The two convolutional layers are followed by ReLU activation but without max-pooling layer. Then we passed the output into two fully connected layers with 100 and 10 hidden units each followed by a ReLU and Softmax activation respectively.

We set 1-norm-bound ϵ to 0.08, batch size 50, learning rate 0.001, and run 50 epochs to train both the robust classifier and standard classifier and compared their performance. For the robust classifier, it reached a test robust error rate 4.16% and standard error rate 0.9%.

Figure 4 (left) shows the Error curve for standard classifier. As we can see, although the standard error rate is decreasing, the robust error rate remains to be 1, indicating that all the input can be attacked by adversarial examples. Figure 4 (right) shows the loss curve for standard training, the robust loss increases as epochs grows, meaning that the standard training will not improve the robustness of the classifier.

Conversely, in Figure 5 (left), we find that in the first ~ 5 epochs, although the standard test error is low, the robust error rate is close to 1, indicating that we've made correct classification to inputs but almost all of them can be attacked using adversarial examples. As we trained more epochs using robust training, the robust error decreases and finally reached 4.16%, meaning that after the robust training, 95.84% of the testing input is provably defensive to adversarial attack with norm-bound $\epsilon = 0.08$.

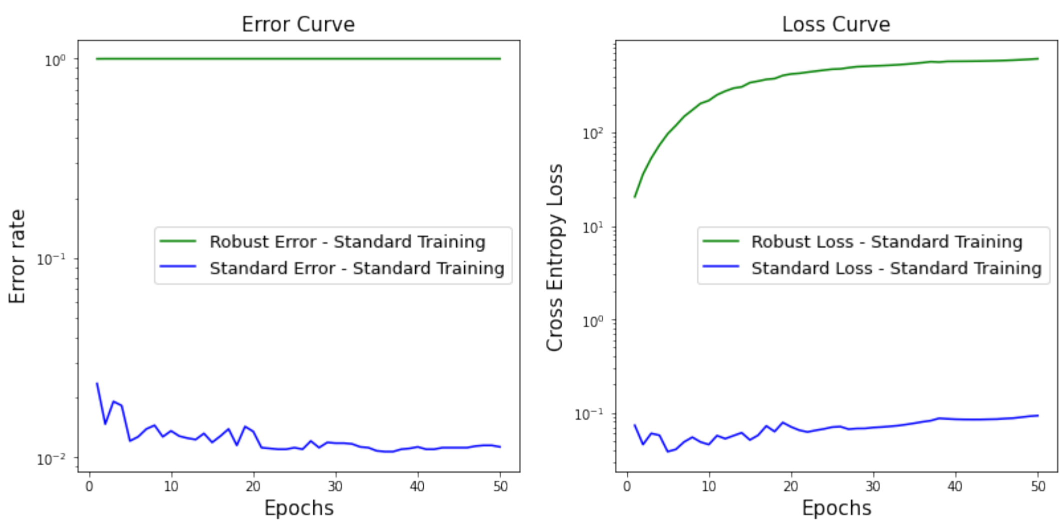

Figure 4: Test Error Rate and Test Loss Curve for Standard Classifier

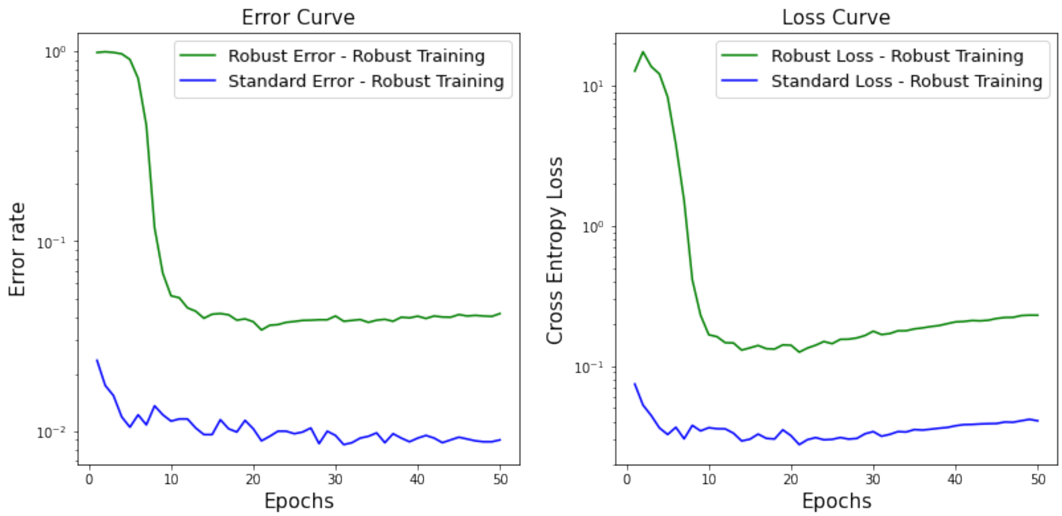

Figure 5: Test Error Rate and Test Loss Curve for Robust Classifier

7 Conclusion and Future Work

The importance of having provable guarantees for deployed large scale neural networks goes without reiterating. Therefore, in our work, we consider the pursuit of a scalable and provable robust training and verification procedure fundamental for a seamless integration of AI technologies into society. Though the performance of our proposed provable model does not represent the state of the art in specialized downstream tasks, we provide a provable method to detect any norm-bounded adversarial attack and we provide provable guarantees to the result that we generated (i.e. our prediction will not change in a small open neighborhood of the input data) with the trade-off of getting slightly diminished accuracy on the prediction.

The importance of this work lies in its modular approach to calculate a robust training objective (or detect the worst case error) via dualizing the original network layer by layer. This general framework allows future work to focus on optimizing the per layer upper bound.

As an example, we consider the task of provable robust natural language models where we first showed the formulation of a dualized attention and then demonstrated the benefits of structured transformation in lieu of attention for a Fourier Transformer. A possible line of future work is to build upon our formulations for dualized Transformer and Fourier Transformer for real-world natural language processing tasks such as classification and generation. In addition to creating efficient and scalable algorithms to calculate bounds or calculate differentiable robust objective, we need also a clear and actionable formulation of the perturbation method for NLP problems.

8 Team Contribution

- Jianyou (Andre) Wang worked on the primal and dual formulation of the original problem with Weili Cao and Yongce Li, as well as the primal and dual formulation for the generalized dual network. He proposed to incorporate new methods for reliably verifying the transformer models. He also worked with Weili Cao and Yongce Li on deriving primal and dual problems, deriving bound propagation algorithms, and running experiments. He discussed literature review for perturbation problems in NLP with Yue Yang.
- Weili Cao worked on formulation and mathematical proof of the primal and dual problem, proving correctness of the algorithms and computing complexity together with Andre Wang and Yongce Li. He also did literature review and wrote previous work accordingly. He also analyzed the results from the experiment with Yongce Li.

- Yongce Li worked on the primal, dual, and KKT condition formulation of the original problem together with Jianyou (Andre) Wang and Weili Cao. He also analyzed the correctness and the time complexity of the original algorithm and AutoDual algorithm, and created 2D examples and ran experiments on MNIST data set.
- Yang Yue worked on literature review, making charts, and latex typing. He also proposed questions to the theoretical work to to address potential concern.

References

[1] Moustafa Alzantot et al. *Generating Natural Language Adversarial Examples*. 2018. arXiv: `1804.07998 [cs.CL]`.

[2] Ross Anderson et al. *Strong mixed-integer programming formulations for trained neural networks*. 2020. arXiv: `1811.01988 [math.OC]`.

[3] Ondřej Bojar et al. "Proceedings of the Ninth Workshop on Statistical Machine Translation". In: *Proceedings of the Ninth Workshop on Statistical Machine Translation*. 2014.

[4] Chih-Hong Cheng, Georg Nührenberg, and Harald Ruess. "Maximum resilience of artificial neural networks". In: *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*. Springer. 2017, pp. 251–268.

[5] Moustapha Cisse et al. "Parseval networks: Improving robustness to adversarial examples". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 854–863.

[6] Jeremy Cohen, Elan Rosenfeld, and Zico Kolter. "Certified adversarial robustness via randomized smoothing". In: *international conference on machine learning*. PMLR. 2019, pp. 1310–1320.

[7] Xinshuai Dong et al. *Towards Robustness Against Natural Language Word Substitutions*. 2021. arXiv: `2107.13541 [cs.CL]`.

[8] Ruediger Ehlers. "Formal verification of piece-wise linear feed-forward neural networks". In: *Automated Technology for Verification and Analysis: 15th International Symposium, ATVA 2017, Pune, India, October 3–6, 2017, Proceedings 15*. Springer. 2017, pp. 269–286.

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. "Explaining and harnessing adversarial examples". In: *arXiv preprint arXiv:1412.6572* (2014).

[10] Po-Sen Huang et al. *Achieving Verified Robustness to Symbol Substitutions via Interval Bound Propagation*. 2019. arXiv: `1909.01492 [cs.CL]`.

[11] Xiaowei Huang et al. "Safety verification of deep neural networks". In: *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer. 2017, pp. 3–29.

[12] Guy Katz et al. "Reluplex: An efficient SMT solver for verifying deep neural networks". In: *Computer Aided Verification: 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I 30*. Springer. 2017, pp. 97–117.

[13] James Lee-Thorp et al. "FNet: Mixing Tokens with Fourier Transforms". In: *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2022, pp. 4296–4313.

[14] Alessio Lomuscio and Lalit Maganti. "An approach to reachability analysis for feed-forward relu neural networks". In: *arXiv preprint arXiv:1706.07351* (2017).

[15] Aleksander Madry et al. "Towards deep learning models resistant to adversarial attacks". In: *arXiv preprint arXiv:1706.06083* (2017).

[16] Alessandro Raganato, Yves Scherrer, and Jörg Tiedemann. "Fixed Encoder Self-Attention Patterns in Transformer-Based Machine Translation". In: *Findings of the Association for Computational Linguistics: EMNLP 2020*. 2020, pp. 556–568.

[17] Motoki Sato et al. *Interpretable Adversarial Perturbation in Input Embedding Space for Text*. 2018. arXiv: `1805.02917 [cs.LG]`.

[18] Christian Szegedy et al. "Intriguing properties of neural networks". In: *arXiv preprint arXiv:1312.6199* (2013).

[19] Yi Tay et al. "Long Range Arena: A Benchmark for Efficient Transformers". In: *ICLR 2021* abs/2011.04006 (2020).

[20] Vincent Tjeng, Kai Xiao, and Russ Tedrake. “Evaluating robustness of neural networks with mixed integer programming”. In: *arXiv preprint arXiv:1711.07356* (2017).

[21] Ashish Vaswani et al. “Attention is all you need”. In: *Advances in neural information processing systems* 30 (2017).

[22] Alex Wang et al. “GLUE: A multi-task benchmark and analysis platform for natural language understanding”. In: *arXiv preprint arXiv:1804.07461* (2018).

[23] Shiqi Wang et al. *Beta-CROWN: Efficient Bound Propagation with Per-neuron Split Constraints for Complete and Incomplete Neural Network Robustness Verification*. 2021. arXiv: `2103.06624 [cs.LG]`.

[24] Eric Wong and Zico Kolter. “Provable defenses against adversarial examples via the convex outer adversarial polytope”. In: *International conference on machine learning*. PMLR. 2018, pp. 5286–5295.

[25] Eric Wong et al. “Scaling provable adversarial defenses”. In: *Advances in Neural Information Processing Systems* 31 (2018).

[26] Weiqiu You, Simeng Sun, and Mohit Iyyer. “Hard-Coded Gaussian Attention for Neural Machine Translation”. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 2020, pp. 7689–7700.