Training Robust Models

EECE 7398

Lecture 5

Recap from Last Lecture

- Convex Barrier proposed theoretical framework & suggested reasons why many existing NN verification methods run into same limits
- More recently, many proposed algorithms aim to reduce gap
  - Branch and Bound (BaB)
  - Sampling-Based Methods
  - K-Neuron Constraints
  - SDP
- Discussed 2 excellent libraries that implement various algorithms
  - Can use these in your own research and stay up-to-date as field advances

Today's Plan

- Robust Training as Minimax Optimization
- Using verification methods during training
- Training models for fast verification

Robust Training as Minimax Optimization

Robust Training: Minimax Optimization

- Example optimization problem for training in presence of adversary:

$$\min_{\theta} \underset{(x,y)\in\mathcal{X}}{E} \left[\max_{\delta\in S} L(x+\delta; y; \theta) \right]$$

- **Inner maximization:** adversary chooses input perturbation to increase loss
- **Outer minimization:** training algorithm chooses parameters that produce lowest expected loss under perturbation
- Ultimately obtain **model parameters** (not just verification result)

[Madry19]

Inner Maximization: Adversary

- Inner + outer optimization problems are each difficult individually
- Typically, assume adversary acts first
- Can the inner maximization problem be solved?
  - How could we under-approximate it?
  - How could we over-approximate it?
  - What are the implications of either of those choices?

$$\min_{\theta} \mathop{E}_{(x,y)\in\mathcal{X}} \left[\max_{\delta\in S} L(x+\delta; y; \theta)\right]$$

Outer Minimization: Robust Trainer

- After adversary acts, have an estimate of ρ(θ)
- Can the outer minimization problem be solved?
  - Could we just use SGD? Objective needs to be differentiable w.r.t. parameters
  - What is the impact of the choice of inner maximization solver?
- Doesn’t provide a formal robustness guarantee, just encouragement
  - Could provide guarantee on training points, but unclear how useful that is

$$\min_{\theta} \underset{(x,y)\in\mathcal{X}}{E} \left[\max_{\delta\in S} L(x+\delta; y; \theta)\right]$$

Verification During Training

Verification during training

- Linear Relaxations during training
  - [Wong18], [Mirman18], [Wang18], [Dvijotham18]
- **Benefits:**
  - Provides reasonably tight bounds
- **Downsides:**
  - Slow computation time
  - May reduce model's "standard accuracy"

- IBP during training
  - [Gowal18], [Mirman18]

$$\min_{\theta} \underset{(\boldsymbol{x},y)\in\mathcal{X}}{E}\left[\kappa L(\boldsymbol{x};y;\theta) + (1-\kappa)L(-\underline{\boldsymbol{m}}_{\text{IBP}}(\boldsymbol{x},\epsilon);y;\theta)\right]$$

[Gowal18]: ϵ-schedule, combined C.E. + IBP loss

- **Benefits:**
  - Much faster runtime than LiRPA
- **Downsides:**
  - Bounds can be very loose, especially early in training → unstable training

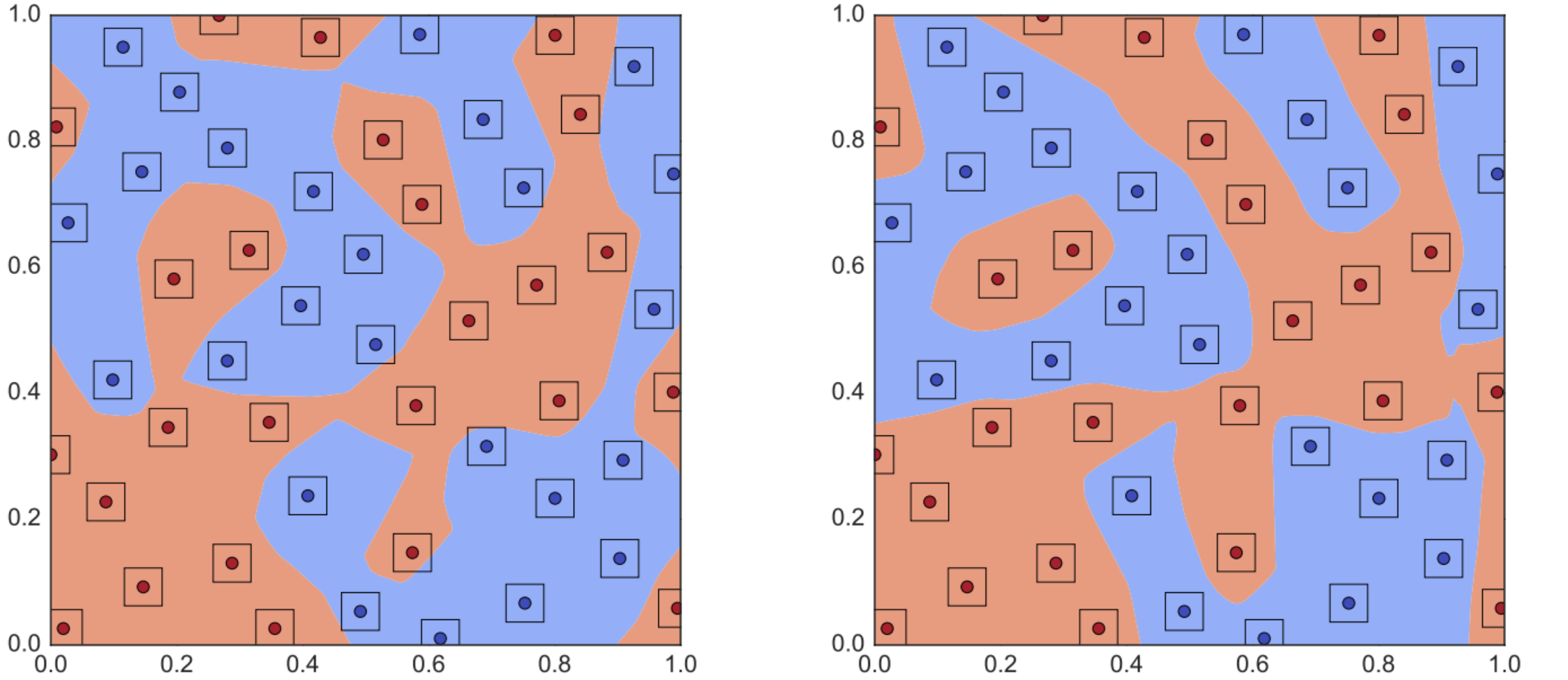
Combinations of CROWN + IBP discussed in:

Towards Stable and Efficient Training of Verifiably Robust Neural Networks

Huan Zhang[1]* Hongge Chen[2] Chaowei Xiao[3] Sven Gowal[4] Robert Stanforth[4] Bo Li[5] Duane Boning[2] Cho-Jui Hsieh[1]

Decision Boundaries After Robust Training

- Is this good?



[Wong & Kolter 18]

How strong of an adversary is necessary?

- Adversarial training may have side-effect of harming natural accuracy
- [Zhang20] suggests a "friendly" adversary enables adversarial robustness & maintains natural accuracy

$$\tilde{x}_i = \underset{\tilde{x} \in \mathcal{B}_\epsilon[x_i]}{\arg\min}\ \ell(f(\tilde{x}), y_i) \quad \text{s.t. } \ell(f(\tilde{x}), y_i) - \min\nolimits_{y \in \mathcal{Y}} \ell(f(\tilde{x}), y) \geq \rho.$$

  - Adversary chooses "least bad" input that classifier mis-labels
  - Accounts for whether classifier mis-labels input or not
- Friendly adversary serves as auto-adjusting curriculum

Attacks Which Do Not Kill Training Make Adversarial Learning Stronger

Jingfeng Zhang [*†1] Xilie Xu [*2] Bo Han [3 4] Gang Niu [4] Lizhen Cui [5] Masashi Sugiyama [4 6] Mohan Kankanhalli [1]

Aside: Constrained Optimization of NNs

- In a perfect world, we could specify robustness as a constraint
  - Or, could select a model class that inherently includes this constraint (e.g., L1-robust neuron paper presented last week)
- That way, any trained model would be robust
- However, constrained optimization with NNs is pretty difficult

Two-Player Games for Efficient Non-Convex Constrained Optimization

Andrew Cotter ACOTTER@GOOGLE.COM
Heinrich Jiang HEINRICHJ@GOOGLE.COM
Google AI

Karthik Sridharan SRIDHARAN@CS.CORNELL.EDU
Cornell University

Lagrangian Duality for Constrained Deep Learning

Ferdinando Fioretto
Syracuse University
ffiorett@syr.edu

Pascal Van Hentenryck
Georgia Institute of Technology
pvh@isye.gatech.edu

Terrence W.K. Mak
Georgia Institute of Technology
wmak@gatech.edu

Cuong Tran
Syracuse University
cutran@syr.edu

Federico Baldo
University of Bologna
federico.baldo2@unibo.it

Michele Lombardi
University of Bologna
michele.lombardi2@unibo.it

Training Models for Fast Verification

Motivation

- Typically, adversarial training aims to increase model's **robust accuracy**
- But there could be many models with similar robustness
- Can we focus training to produce models that can be quickly verified?

TRAINING FOR FASTER ADVERSARIAL ROBUSTNESS VERIFICATION VIA INDUCING RELU STABILITY

Kai Y. Xiao **Vincent Tjeng** **Nur Muhammad (Mahi) Shafiullah** **Aleksander Mądry**
Massachusetts Institute of Technology
Cambridge, MA 02139
{kaix, vtjeng, nshafiul, madry}@mit.edu

Speedup Idea 1: Weight Sparsity

- **Issue:** Runtime of MILP and LP solvers increases with number of constraints/variables
  - For NN verification, each term in each weight matrix imposes a constraint
- **Idea:** Can reduce number of constraints/variables by learning models with sparse weight matrices
- How to do this?

Dataset	Epsilon		Training Method	Test Set Accuracy	Provable Adversarial Accuracy	Average Solve Time (s)
MNIST	$\epsilon = 0.1$	1	Adversarial Training	99.17%	19.00%	2970.43
		2	+ℓ_1-Regularization	99.00%	82.17%	21.99
		3	+Small Weight Pruning	98.99%	89.13%	11.71
		4	+ReLU Pruning (control)	98.94%	91.58%	6.43

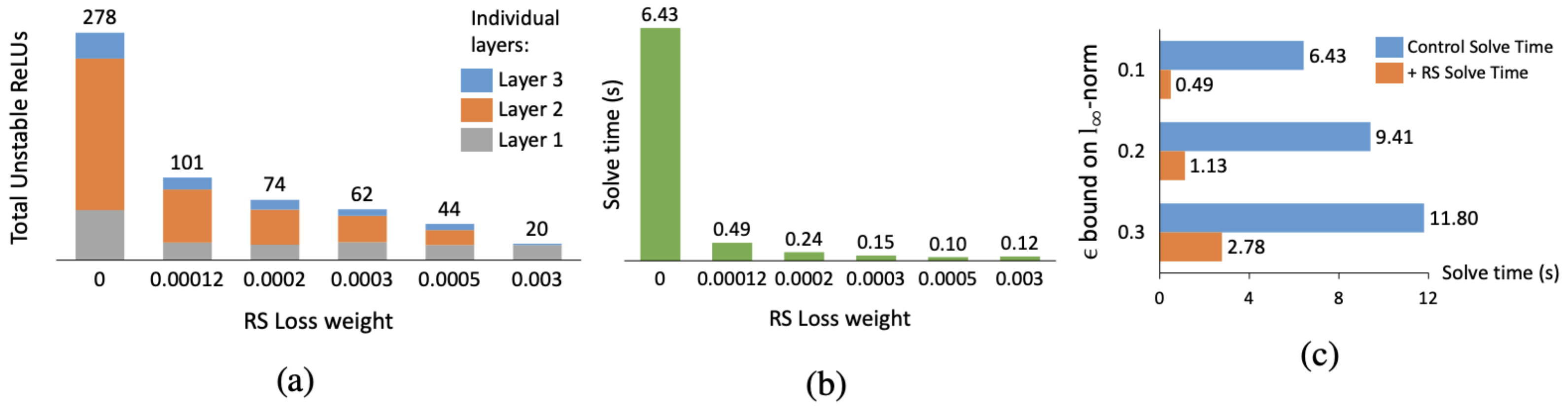
Speedup Idea 2: ReLU Stability

- **Issue:** Complete verifiers need to branch on each unstable ReLU
- **Idea:** Add loss term to discourage unstable ReLUs
  - Function that indicates when ReLU is stable is non-differentiable

$$F^*(\hat{u}_{ij}, \hat{l}_{ij}) = \text{sign}(\hat{u}_{ij}) \cdot \text{sign}(\hat{l}_{ij}).$$

  - Instead, use smooth / differentiable approximation of F*

$$F(\hat{u}_{ij}, \hat{l}_{ij}) = -\tanh(1 + \hat{u}_{ij} \cdot \hat{l}_{ij})$$



Recap

- Robust Training as Minimax Optimization
- Using verification methods during training
  - Tradeoff between speed & strength of adversary
  - Adversary strength may need to be monitored/adapted throughout training
- Training models for fast verification
  - Modifying training loss to encourage ReLU stability, weight sparsity

Plan for Wednesday: Paper Discussion

- **Individual presentations:** Pick your paper and add it to the spreadsheet
- **Group discussion:** Recent Advances in Adversarial Training for Adversarial Robustness
  - https://arxiv.org/pdf/2102.01356.pdf
- **Next Lecture (10/4):** Verifying Neural Feedback Loops

Final Project

- Form a team with one other person in the class
- **Goal:** Apply some idea(s) from this class to a problem in your research
- Some Possible Approaches:
  - Re-implement a decently complicated paper (e.g., beta-CROWN) & apply to some new problems
  - Apply an existing algorithm on a real system (e.g., hardware)
  - Extend an existing algorithm in a new way (e.g., improve its scalability)
  - Develop some theory in this domain
- Deliverables:
  - Presentations: Last week of class (12/4 and 12/6)
    - 15min (12min + 3min Q&A) per team
  - Report: Due 12/9
    - Baseline Target: Something that's ready to submit to a good workshop
    - Written in IEEE or NeurIPS format