Reliable and Interpretable Artificial Intelligence

Lecture 11: Combining Logic and Deep Learning

Martin Vechev

ETH Zurich

Fall 2020

SRILAB

http://www.sri.inf.ethz.ch

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Now: Logic and Deep Learning

Can we query the network with questions beyond adversarial examples?

Can we enforce properties that the network should satisfy?

What are the guarantees of such methods?

Lecture is based on:

DL2: Training and Querying Neural Networks with Logic, ICML 2019

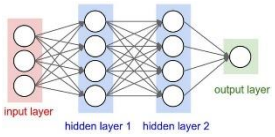
Combining Logic and Deep Learning

As Deep Learning makes more and more decisions (e.g: bank credit, job applications, university admissions, political elections), it becomes critical to be to understand how these decisions can be influenced and understood.

Query

What should the applicant change to receive a bank credit?

Neural Network NN



Rejected for credit by NN

- income
- residence
- conditions
- job
- …

Deep Learning Query Engine

To be in the > 84% probability of receiving credit, increase income by at least 5K and be employed for at least 3 more months…

Combining Logic and Deep Learning

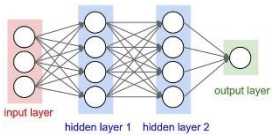
Adversarial examples are in fact just a special case of a query…

Query

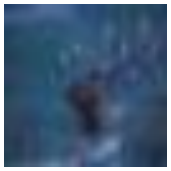
```
find i[32, 32, 3]
where i in [0, 255],
    class(NN(i)) = 9,
    ||i - deer||∞ < 25,
    ||i - deer||∞ > 5
```

Find an image i which gets classified to 9 (truck) where the image i is within some distance of the image deer.

Neural Network NN

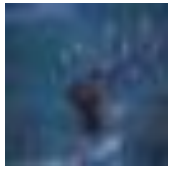


deer



Deep Learning Query Engine

image i



classified as truck by NN!

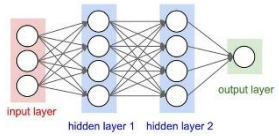
Combining Logic and Deep Learning

Comparing neural networks

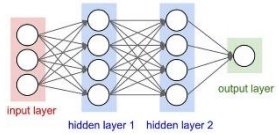
```
find i[28, 28]
where i in [0, 1],
    i[0:9,:] = nine[0:9,:],
    class(NN1(i)) = 8,
    class(NN2(i)) = 9
```

Find an image i which gets classified to 8 with network 1 and to 9 with network 2, such that pixels in row 0:9 of image i are the same as image nine

Network NN1



Network NN2



Deep Learning Query Engine

image i

Image nine

Combining Logic and Deep Learning

Adversarial examples are in fact just a special case of a query…

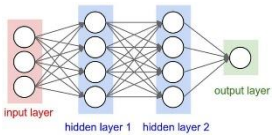
Hmm, but this involves logical constraints and a neural network!

How can we unify these?

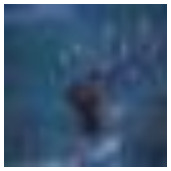
Query

```
find i[32, 32, 3]
where i in [0, 255],
    class(NN(i)) = 9,
    ||i - deer||∞ < 25,
    ||i - deer||∞ > 5
```

Neural Network NN

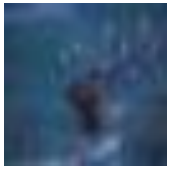


deer



Deep Learning Query Engine

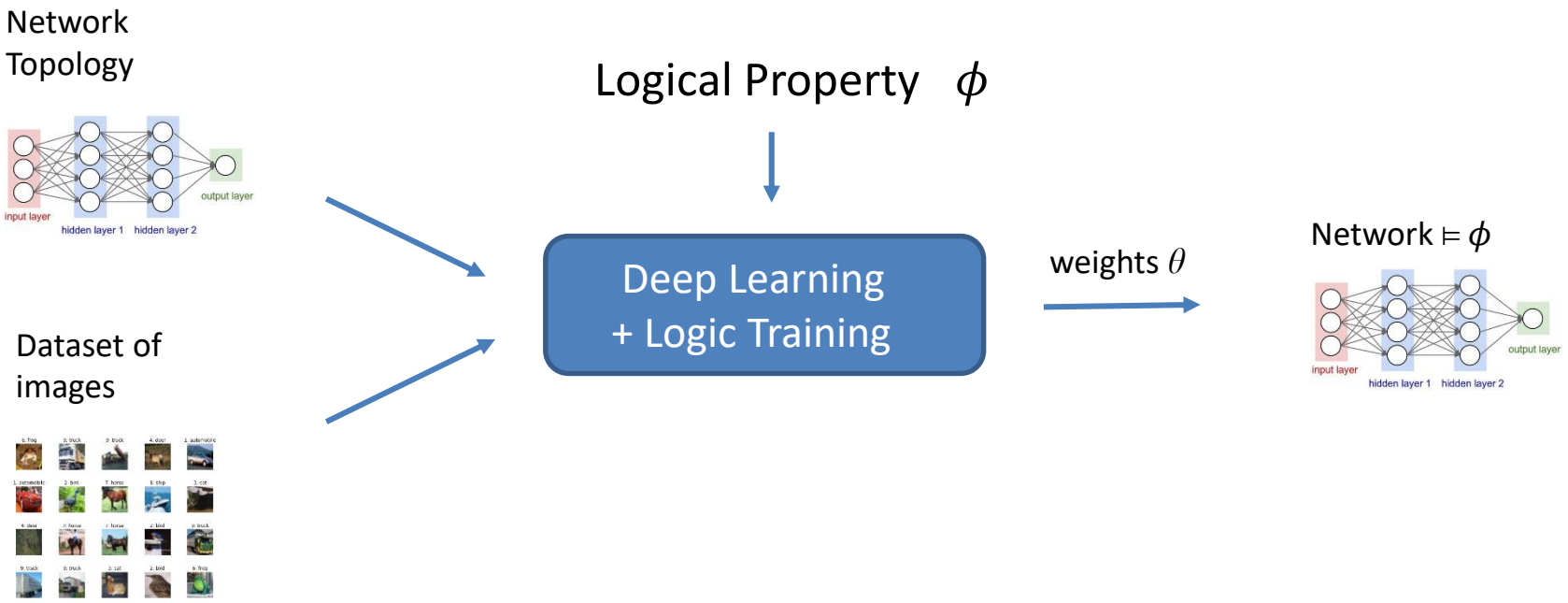
image i



classified as truck by NN!

Combining Logic and Deep Learning

We can also train neural networks to satisfy a logical property



In fact, this can help accuracy as we can label part of the data and specify properties on the remaining, unlabeled data.

Declaratively, to query the network for inputs we need a precise way to impose constraints on these inputs, hence some type of logic.

Operationally, we also need a way to perform queries on the network with these constraints.

Part I:
Querying the Network

Declaratively, to query the network for inputs we need a precise way to impose constraints on these inputs, hence some type of logic.

Lets first define the logic

We introduce a standard logic with:

- no quantifiers: no $\forall, \exists$
- $\neg, \neq, \wedge, \vee, \leq, \geq, <, >, \Rightarrow$
- functions f: $\mathbb{R}^m \rightarrow \mathbb{R}^n$
- terms: variables, constants: represent vectors of reals
- terms: function application
- terms: arithmetic expressions over terms (e.g., +)

Comparison operations on vectors are done point-wise.

If a and b are vectors of dimension 2, then a = b is written as a[0] = b[0] $\wedge$ a[1] = b[1]

The logic used in our example query

$$\mathbf{class}\big(\mathrm{NN(i)}\big) = 9 \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty < 25 \ \wedge \ \|\mathrm{i} - \mathrm{deer}\|_\infty > 5$$

Lets expand this a bit

The logic used in our example query

$$\mathbf{class}\big(\mathrm{NN(i)}\big) = 9 \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty < 25 \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty > 5$$

Syntactic sugar

$$\bigwedge_{\mathrm{j}=1,\, \mathrm{j}\neq 9}^{\mathrm{k}} \mathrm{NN(i)[j]} < \mathrm{NN(i)[9]} \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty < 25 \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty > 5$$

This is the actual formula being used after expanding the syntactic sugar.

Here, k is the number of labels.

The logic used in our example query

$$\phi \doteq \bigwedge_{j=1,\, j\neq 9}^{k} \mathrm{NN}(i)[j] < \mathrm{NN}(i)[9] \wedge \|i - \mathrm{deer}\|_\infty < 25 \wedge \|i - \mathrm{deer}\|_\infty > 5$$

Here we have 2 functions: NN and the norm $\| \ \|_\infty$.

Function NN returns a probability distribution over labels.

We have 4 constants: 9, 5, 25 and deer (real-valued vector)

We have 1 free variable i

Goal: find a value for i that satisfies the constraint above

How do we solve this problem?

$$\phi \doteq \bigwedge_{\mathrm{j}=1,\,\mathrm{j}\neq 9}^{\mathrm{k}} \mathrm{NN(i)[j]} < \mathrm{NN(i)[9]} \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty < 25 \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty > 5$$

One approach to finding the value of i is to invoke standard a constraint solver (e.g., SMT solver which generalize SAT to richer theories). Unfortunately, unless the network NN is really small, these solvers simply time out (one of the problem is the non-linear constraints that the network exhibits). Thus, we need another approach.

Operationally, we also need a way to perform queries on the network with these constraints.

Solve as optimization

$$\phi \doteq \bigwedge_{j=1,\, j\neq 9}^{k} \mathrm{NN(i)[j]} < \mathrm{NN(i)[9]} \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty < 25 \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty > 5$$

Instead, the idea will be to introduce a particular translation T of logical formulas into a differentiable loss function $T(\phi)$ to be solved with (mostly standard) optimization, where the translation T has a certain property.

Wanted Property of Translation

Theorem: $\forall x$, $T(\phi)(x) = 0$ **if and only if** x satisfies ϕ

What this theorem says is that: if we can find a solution x where the loss function of ϕ is 0, then that solution x is a satisfiable assignment to ϕ, that is, it is a solution to our original problem.

Also: If x satisfies ϕ then the loss function at x is 0

Optimize to find a solution

Theorem: $\forall x$, $T(\phi)(x) = 0$ **if and only if** x satisfies ϕ

Given this theorem, our goal is to find an assignment $x = i$ such that $T(\phi)(i)$ is 0.

We can use standard gradient-based optimization to minimize the function $T(\phi)$. There can potentially be many solutions which set the function to 0.

Translation: Formula to Loss

Logical Term	Translation
$t_1 \leq t_2$	$\max(0, t_1 - t_2)$
$t_1 \neq t_2$	$[t_1 = t_2]$
$t_1 = t_2$	$T(t_1 \leq t_2 \wedge t_2 \leq t_1)$
$t_1 < t_2$	$T(t_1 \leq t_2 \wedge t_1 \neq t_2)$
$\varphi \vee \psi$	$T(\varphi) \cdot T(\psi)$
$\varphi \wedge \psi$	$T(\varphi) + T(\psi)$

Translation: Formula to Loss

Logical Term	Translation
$t_1 \leq t_2$	$\max(0, t_1 - t_2)$
$t_1 \neq t_2$	$[t_1 = t_2]$
$t_1 = t_2$	$T(t_1 \leq t_2 \wedge t_2 \leq t_1)$
$t_1 < t_2$	$T(t_1 \leq t_2 \wedge t_1 \neq t_2)$
$\varphi \vee \psi$	$T(\varphi) \cdot T(\psi)$
$\varphi \wedge \psi$	$T(\varphi) + T(\psi)$

Two comments on the translation:

- Translation is recursive: translating a term defined in a way which refers to how the constituents of the term are translated.
- The resulting loss function is non-negative

Intuition behind the translation

Logical Term	Translation
$t_1 \leq t_2$	$\max(0, t_1 - t_2)$

what this says is:

as long as $t_1 \leq t_2$, the translated value is 0.

However, if $t_1 > t_2$ then the magnitude of the violation is measured by $t_1 - t_2$ and the closer the two values get, the smaller the violation is.

Intuition behind the translation

Logical Term	Translation
$t_1 \neq t_2$	$[t_1 = t_2]$

Translated value is equal to 0 except when $t_1 = t_2$. In the case of $t_1 = t_2$, translated values is equal to 1.

Intuition behind the translation

Logical Term	Translation
$t_1 = t_2$	$T(t_1 \leq t_2 \wedge t_2 \leq t_1)$

Translated value is computed recursively - it is equal to:

$$T(t_1 \leq t_2) + T(t_2 \leq t_1) =$$
$$\max(0, t_1 - t_2) + \max(0, t_2 - t_1) = |t_1 - t_2|.$$

As t_1 and t_2 get closer, the translated value gets closer to 0.

Intuition behind the translation

Logical Term	Translation
$t_1 < t_2$	$T(t_1 \leq t_2 \wedge t_1 \neq t_2)$

what this says is:

To encode $t_1 < t_2$, we will encode conjunction of $t_1 \leq t_2$ and $t_1 \neq t_2$.

The translated value is then equal to $\max(0, t_1 - t_2) + [t_1 = t_2]$.

The Translation: Formula to Loss

Logical Term	Translation
$\varphi \vee \psi$	$\mathrm{T}(\varphi) \cdot \mathrm{T}(\psi)$

what this says is:

if one of the terms is 0, then the entire translated expression will be 0 (that is, the formula is satisfied).

The Translation: Formula to Loss

Logical Term	Translation
$\varphi \wedge \psi$	$\mathrm{T}(\varphi) + \mathrm{T}(\psi)$

what this says is:

for the result to be 0, both of terms should be 0.

Example: a satisfying formula

Logical formula:

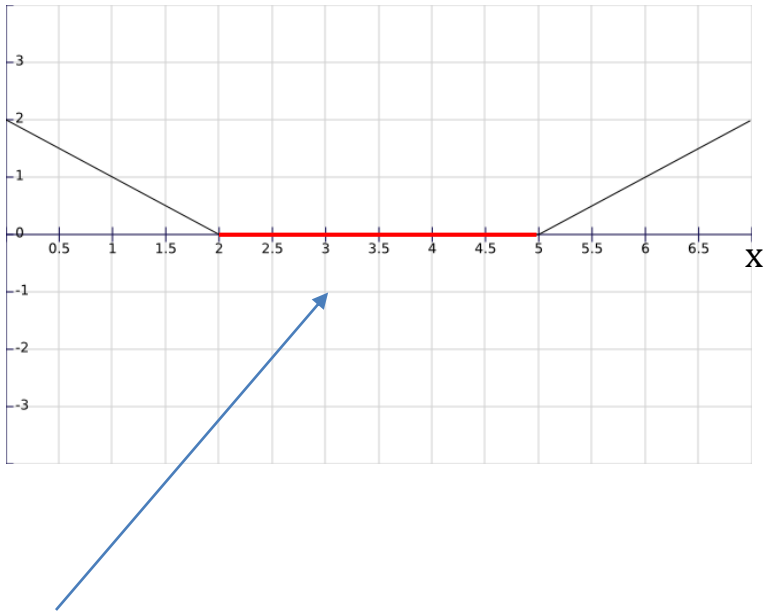
$$x \geq 2 \wedge x \leq 5$$

Translated Loss:

$$\max(0, 2 - x) + \max(0, x - 5)$$

Satisfying assignments:

Any value between 2 and 5, inclusive



The function is 0 when x is between 2 and 5.
We need to find one such assignment.

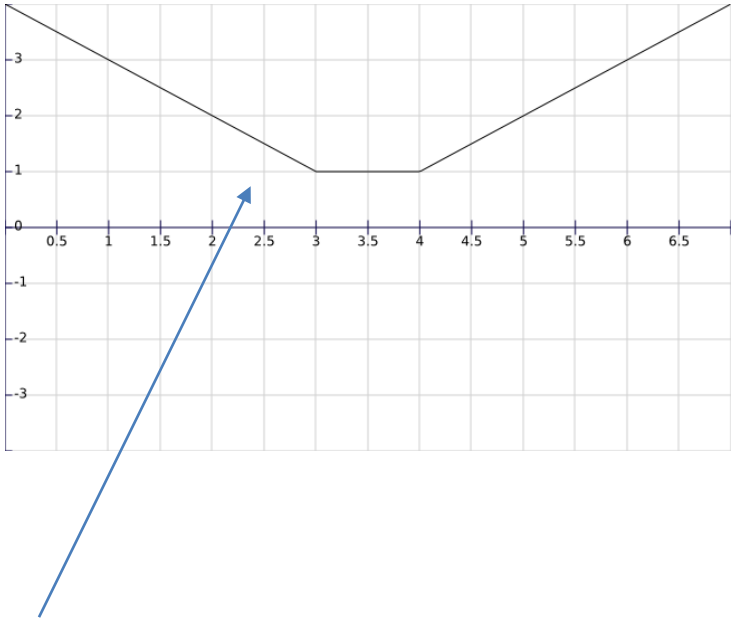
Example: an unsatisfiable formula

Logical formula:

$x \geq 4 \wedge x \leq 3$

Translated Loss:

$\max(0, 4 - x) + \max(0, x - 3)$



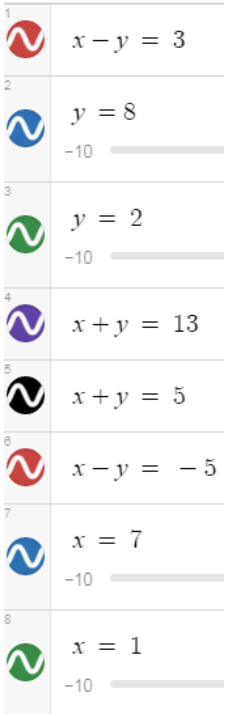
Satisfying assignments:

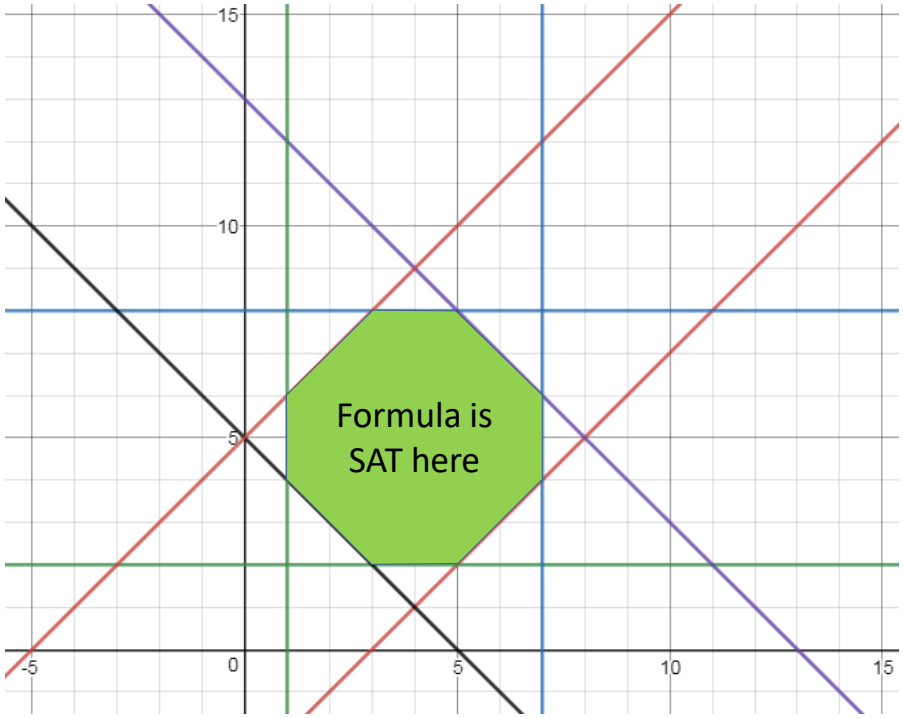
There are no satisfying assignments

The function is never 0

Something more fun: Octagon

See plot here:
https://www.desmos.com/calculator/kw38cpoirk





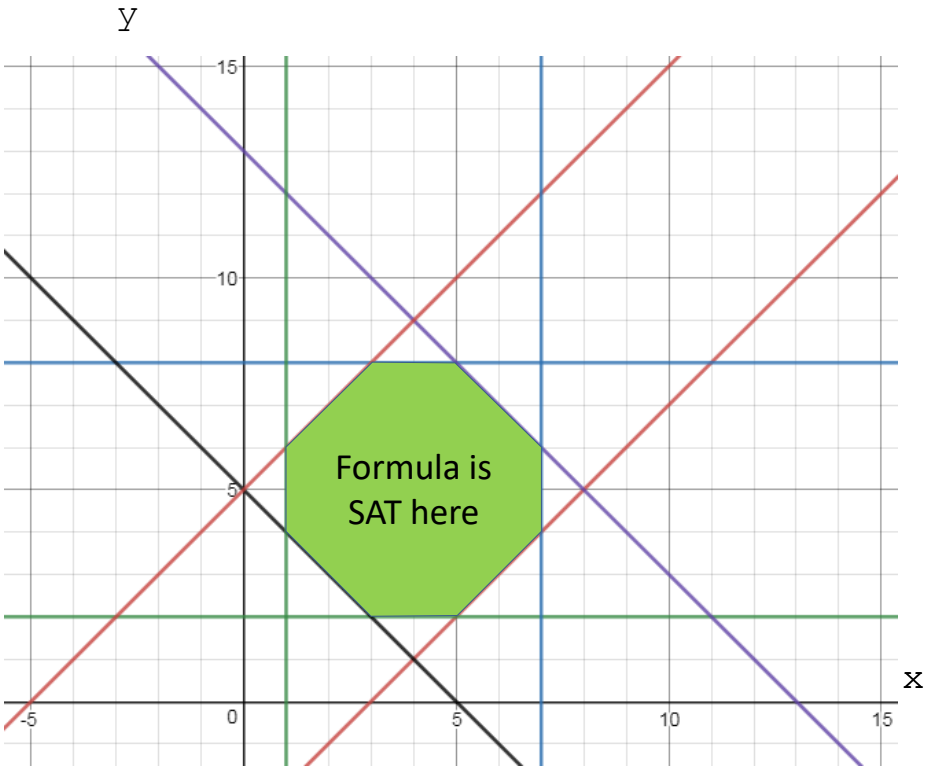
$$
\begin{aligned}
& x - y \le 3 && \wedge \\
& y \le 8 && \wedge \\
& y \ge 2 && \wedge \\
& x + y \le 13 && \wedge \\
& x + y \ge 5 && \wedge \\
& x \ge 1 && \wedge \\
& x - y \ge -5 && \wedge \\
& x \le 7
\end{aligned}
$$

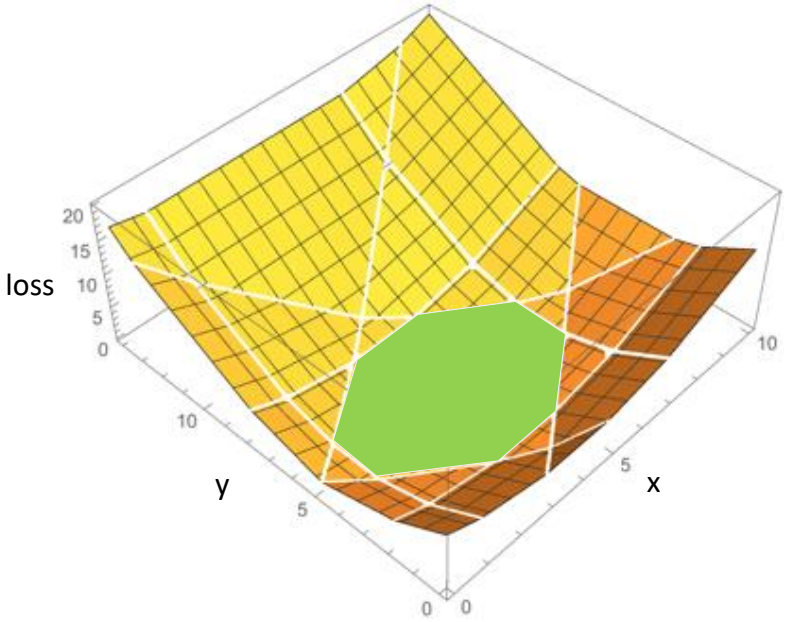
Translate

$$
\begin{aligned}
& \max(0,\ x-y-3) && + \\
& \max(0,\ y-8) && + \\
& \max(0,\ 2-y) && + \\
& \max(0,\ x+y-13) && + \\
& \max(0,\ 5-x-y) && + \\
& \max(0,\ 1-x) && + \\
& \max(0,\ -5-x+y) && + \\
& \max(0,\ x-7)
\end{aligned}
$$

Plot with Mathematica

Plot3D
[Max[0, 2 - y] + Max[0, -3 + x - y] + Max[0, -8 + y] + Max[0, -13 + x + y] + Max[0, 5 - x - y] + Max[0, 1 - x] + Max[0, -5 - x + y] + Max[0, x - 7], {x, 0, 10}, {y, 0, 14}]





We can visually see that for values of x between 1 and 7 and of y between 2 and 8, the loss is 0

Back to Neural Nets

$$\bigwedge_{j=1,\, j\neq 9}^{k} \mathrm{NN(i)[j]} < \mathrm{NN(i)[9]} \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty < 25 \wedge \|\mathrm{i} - \mathrm{deer}\|_\infty > 5$$

Original Formula ϕ

$$\mathrm{NN(i)[1]} < \mathrm{NN(i)[9]} \wedge$$
$$\mathrm{NN(i)[2]} < \mathrm{NN(i)[9]} \wedge$$
$$\mathrm{NN(i)[3]} < \mathrm{NN(i)[9]} \wedge$$
$$\mathrm{NN(i)[4]} < \mathrm{NN(i)[9]} \wedge$$
$$\mathrm{NN(i)[5]} < \mathrm{NN(i)[9]} \wedge$$
$$\mathrm{NN(i)[6]} < \mathrm{NN(i)[9]} \wedge$$
$$\mathrm{NN(i)[7]} < \mathrm{NN(i)[9]} \wedge$$
$$\mathrm{NN(i)[8]} < \mathrm{NN(i)[9]} \wedge$$
$$\|\mathrm{i} - \mathrm{deer}\|_\infty < 25 \wedge$$
$$\|\mathrm{i} - \mathrm{deer}\|_\infty > 5$$

Translated Loss $T(\phi)$

$$\max(0, \mathrm{NN(i)[1]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[1]} = \mathrm{NN(i)[9]}]$$
$$+ \max(0, \mathrm{NN(i)[2]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[2]} = \mathrm{NN(i)[9]}]$$
$$+ \max(0, \mathrm{NN(i)[3]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[3]} = \mathrm{NN(i)[9]}]$$
$$+ \max(0, \mathrm{NN(i)[4]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[4]} = \mathrm{NN(i)[9]}]$$
$$+ \max(0, \mathrm{NN(i)[5]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[5]} = \mathrm{NN(i)[9]}]$$
$$+ \max(0, \mathrm{NN(i)[6]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[6]} = \mathrm{NN(i)[9]}]$$
$$+ \max(0, \mathrm{NN(i)[7]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[7]} = \mathrm{NN(i)[9]}]$$
$$+ \max(0, \mathrm{NN(i)[8]} - \mathrm{NN(i)[9]}) + [\mathrm{NN(i)[8]} = \mathrm{NN(i)[9]}]$$
$$+ \max(0, \|\mathrm{i} - \mathrm{deer}\|_\infty - 25) + [\|\mathrm{i} - \mathrm{deer}\|_\infty = 25]$$
$$+ \max(0, \quad 5 - \|\mathrm{i} - \mathrm{deer}\|_\infty) + [\|\mathrm{i} - \mathrm{deer}\|_\infty = 5]$$

Tricky detail for optimization: box constraints

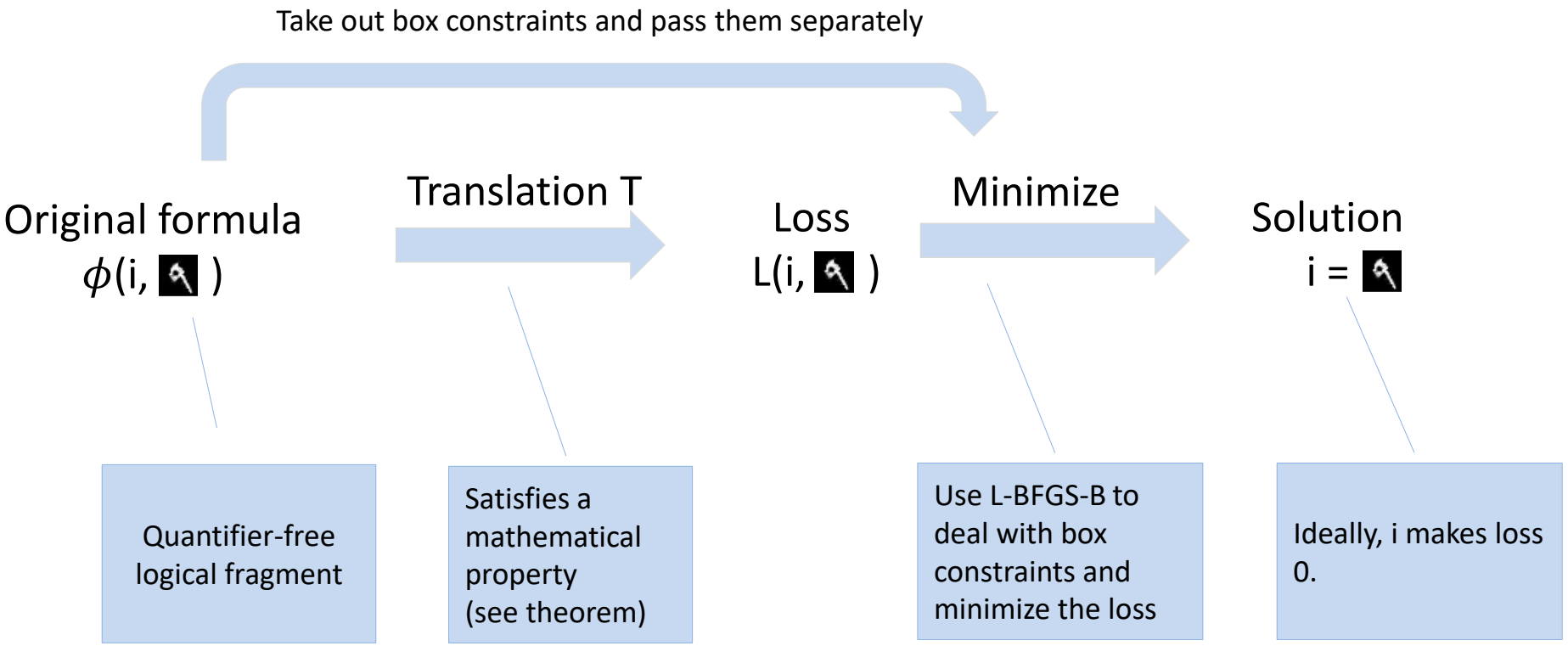
```
find i[28, 28]
where i in [0, 1],
    i[0:9,:] = nine[0:9,:],
    class(NN1(i)) = 8,
    class(NN2(i)) = 9
```

As we discussed before, optimizing for 'box constraints' with gradient descent is ineffective. Here, the [0,1] restricts each pixel in image i to be of a particular concrete value.

Thus, rather than translating these constraints like all other constraints, we can ``take out'' the box constraints out and give them to the optimizer directly.

For example, we can use the **L-BFGS-B** optimizer where the **B** stands for Box constraints.

Finding SAT with optimization: solution flow



We can think of this pipeline as using optimization to find counter-examples to a given property that the network should satisfy

Beyond counter examples however, a fundamental question is:

can we somehow force the network to **satisfy the property** we want? Essentially, **defend the network** w.r.t to a particular property, beyond only robustness.

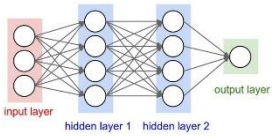
Part II:

Training the Network with Background Knowledge

(Generalized Adversarial Training beyond Robustness)

Training the Network with Logic

Neural Network Topology



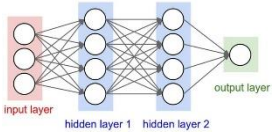
Dataset of images

Logical Property ϕ

Deep Learning
+ Logic Training

weights θ

Network $\models \phi$



Principles:

- We still use **the same logic as before**
- We still ``compile'' **property** ϕ **into a loss** as before
- We need a way to define the **optimization problem** now

To motivate:

Lets look at some general class of use cases before discussing how it actually works

Use in supervised learning (entire dataset labeled): Example training constraints on CIFAR-10

All images in dataset that are classified as a car have a higher probability for the label $truck$ than the probability for dog:

$$y = car \Rightarrow NN(x)[truck] > NN(x)[dog] + \delta$$

Same as above, but now constraints should also hold for all images close to x:

$$\forall z \in L_\infty(x, \epsilon).\ y = car \Rightarrow NN(z)[truck] > NN(z)[dog] + \delta$$

Experimentally, once we finish training with either of the two constraints , the accuracy tends to dip slightly (1-2%), however, the constraint accuracy (how many images in the dataset satisfy the constraint) tends to increase by about 10% which is significant.

However, not all images in the dataset satisfy the constraint. There are several reasons for that: (i) we typically do not stop at a solution with loss 0, (ii) the constraint is not supposed to hold for all images anyway, (iii) the loss is not yet ideal and can be improved.

Use in semi-supervised (part of dataset labeled): Example training constraints on CIFAR-100

Labeled Dataset (e.g. 20%)

Unlabeled Dataset (e.g. 80%)
☹ but we provide a constraint ☺

Here we provide the labels as usual

Here **we do not provide labels** as its often difficult to give labels
However, we can provide a constraint (belief):

$$p(people) < \epsilon \vee p(people) > 1 - \epsilon \wedge$$
$$p(insects) < \epsilon \vee p(insects) > 1 - \epsilon \wedge \ldots$$

Either the probability **of a group** is very high or very low:
$p(people) = NN(x)[baby] + NN(x)[boy] + NN(x)[girl] + \ldots$

Approach I: train with just the labeled (green) dataset, **ignoring** the red one.

or

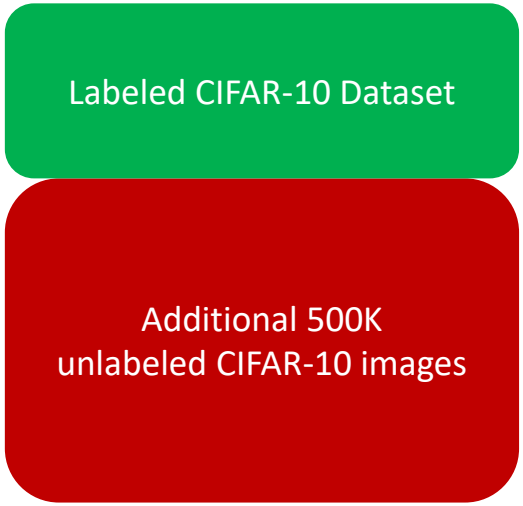
Approach II: train with both data sets, and leverage the constraint also on the red one.

With Approach II, can raise prediction accuracy on test data set by 2% and constraint accuracy by 26% ☺

Connection to L_{inf} robustness:

Semi-supervised learning increases robustness on CIFAR-10

[Carmon et al. 2019, Uesato et al. 2019]



Here we provide the labels as usual

Simplest approach:

1) Train a base classifier $\hat{\Theta}$ on labeled dataset
2) Use $\hat{\Theta}$ to infer the labels for unlabeled dataset.
3) Use standard adversarial training to learn robust classifier Θ on all images (both originally labeled and pseudo-labeled by $\hat{\Theta}$.

Result: State-of-the-art standard accuracy and empirical robustness against strong adversaries.

Benefits over TRADES, previously known best defense, are 5% higher standard accuracy and 7% higher empirical robustness (against PGD).

Problem Statement

find	θ
maximize	$\rho(\theta)$
where	$\rho(\theta) = \mathbf{E}_{s \sim D}\left[\forall \boldsymbol{z} \ . \ \phi(\boldsymbol{z},\ s,\ \theta)\right]$

What this says is: we want to find such parameters/weights θ for the network, so the expected value of the property increases.

Note that we even allow restricted quantified formulas here.

Rephrasing : Step I

became minimization

find such $\boldsymbol{z}$ where violation is maximized

find θ

minimize $\rho(\theta)$

where $\rho(\theta) = \mathbf{E}_{s\sim D}[\max_{\boldsymbol{z}} \neg\phi(\boldsymbol{z},\ s,\ \theta)]$

What this says is: we want to find such parameters/weights θ for the neural network, so that the maximum violation of the property ϕ is minimized.

This is essentially: **generalized adversarial training beyond robustness**

Rephrasing: Step II

find θ

minimize $\rho(\theta)$

where $\rho(\theta) = \mathbf{E}_{s\sim D}[\mathsf{T}(\phi)(z\_worst,\ s, \theta)]$

and $z\_worst = \underset{z}{\textbf{argmin}}(\mathsf{T}(\neg\phi)(z,\ s, \theta))$

minimize the expected value of the worst counter-example

find the worst-case counter example *z_worst*

The translation leads to a differentiable function which we can optimize. Intuitively, we are trying to get the worst possible violation of the formula and then to find a network that minimizes its effect.

Solving the inner minimization problem

find	θ
minimize	$\rho(\theta)$
where	$\rho(\theta) = \mathbf{E}_{s\sim D}[\mathsf{T}(\phi)(z\_worst,\ s, \theta)]$
and	z_worst = **argmin**$_{z}$($\mathsf{T}(\neg\phi)(z,\ s, \theta)$)

In principle, we can use a standard optimizer to solve for the inner minimization problem. However, the variable z can participate in all kinds of constraints in ϕ. And as we saw earlier, even if its just norm constraints that z participates in, a pure SGD-style optimizer can have a hard time.

Solving the inner minimization problem

find	θ
minimize	$\rho(\theta)$
where	$\rho(\theta) = \mathbf{E}_{s\sim D}[\mathsf{T}(\phi)(z\_worst,\ s, \theta)]$
and	z_worst = **argmin**$_{z}$($\mathsf{T}(\neg\phi)(z,\ s, \theta)$)

Thus, we will focus on a restricted fragment where z participates in constraints that restrict z to be a convex set where we have an efficient algorithm for projection (a closed form solution). Note that in general, projection onto arbitrary convex sets is hard.

Example: Generating the Loss

$$\phi(z, x, \theta) = \|x - z\|_\infty \leq \epsilon \Rightarrow NN_\theta(z)[3] > \delta$$

we want to enforce the constraint $\forall z. \phi(z, x, \theta)$

expansion of $\Rightarrow$

$$\phi(z, x, \theta) = \neg\|x - z\|_\infty \leq \epsilon \vee NN_\theta(z)[3] > \delta$$

negation $\neg$

$$\neg\phi(z, x, \theta) = \|x - z\|_\infty \leq \epsilon \wedge NN_\theta(z)[3] \leq \delta$$

translation to loss

$$loss(z, x, \theta) = \max(0, \|x - z\|_\infty - \epsilon) + \max(0, NN_\theta(z)[3] - \delta)$$

difficult to solve: minimization over arbitrary constraints

A possible solution to minimizing the loss

this part aims to restrict z to be in the L_∞ ball around x of size ϵ

$$\underset{z}{\mathbf{argmin}}\ loss(z,x,\theta) = \max(0, \|x - z\|_\infty - \epsilon) + \max(0, NN_\theta(z)[3] - \delta)$$

split the problem

L_∞ ball around x of size ϵ

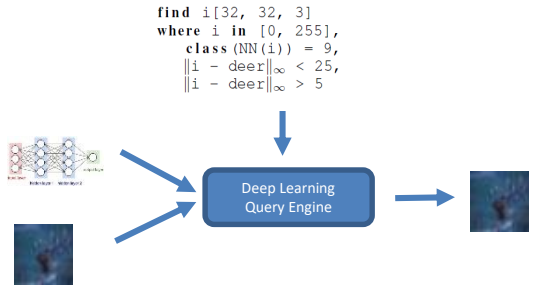
$$\underset{z}{\mathbf{argmin}}\ loss(z,x,\theta) = \max(0, NN_\theta(z)[3] - \delta) \qquad L_\infty(x,\epsilon)$$

Solve with Projected Gradient Descent (PGD) while projecting z onto the L_∞ ball

Note: in general, efficient projections (closed form solutions) on convex sets is a hard problem. Such algorithms exist for L_1, L_2, L_∞ and some others. Because of this, in practice, the logic is restricted to having z participate only in constraints where **efficient projections** are possible.

Lecture Summary

Combine Deep Learning with Logic



Logic to loss

Logical Term	Translation
$t_1 \leq t_2$	$\max(0, t_1 - t_2)$
$t_1 \neq t_2$	$[t_1 = t_2]$
$t_1 = t_2$	$T(t_1 \leq t_2 \wedge t_2 \leq t_1)$
$t_1 < t_2$	$T(t_1 \leq t_2 \wedge t_1 \neq t_2)$
$\varphi \vee \psi$	$T(\varphi) \cdot T(\psi)$
$\varphi \wedge \psi$	$T(\varphi) + T(\psi)$

Training with logic as maximization

find θ
maximize $\rho(\theta)$

where $\rho(\theta) = \underset{s \sim D}{\mathbf{E}} [\forall \boldsymbol{z} \ . \ \phi(\boldsymbol{z}, \ s, \theta)]$

Restrictions so we can use PGD

$$\min_z \quad loss(z, x, \theta) = \max(0, \|x - z\|_\infty - \epsilon) + \max(0, NN(z) - \delta)$$

$$\min_z \quad loss(z, x, \theta) = \max(0, NN(z) - \delta) \qquad L_\infty(x, \epsilon)$$

L_∞ ball around x of size ϵ