Injecting Logical Constraints into Neural Networks via Straight-Through Estimators

Zhun Yang[1], Joohyung Lee[1 2], Chiyoun Park[2]

[1] Arizona State University
[2] Samsung Research

Arizona State University

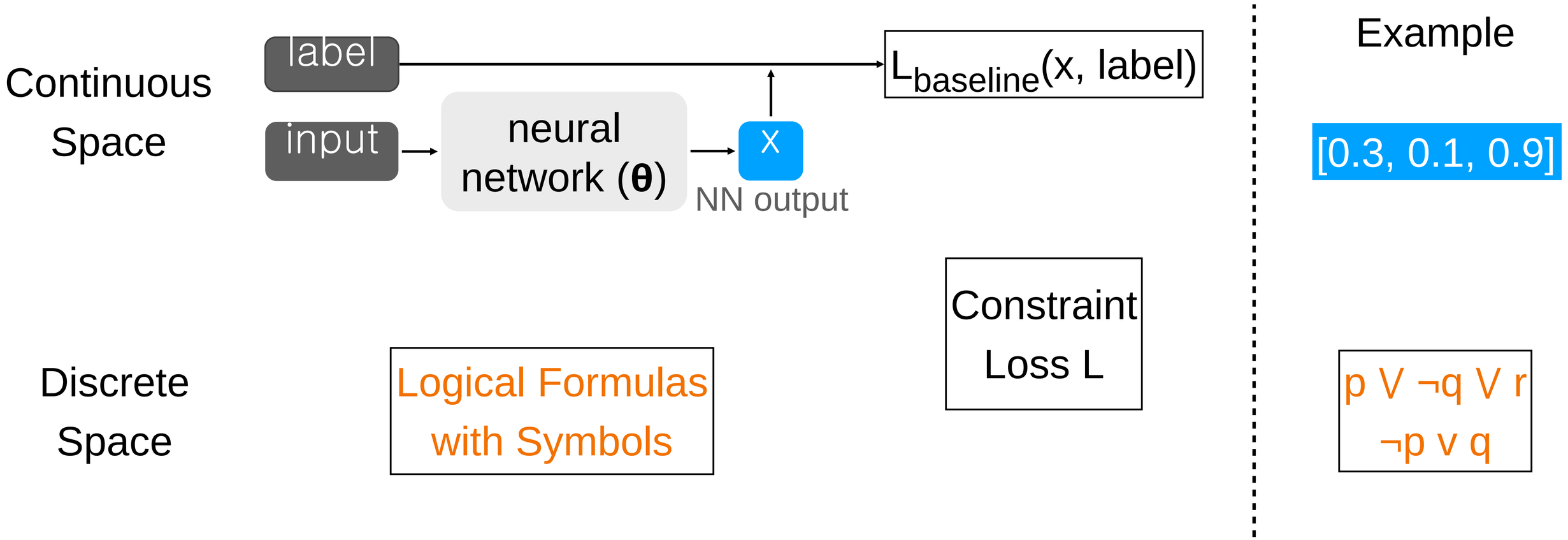
Samsung Research

Introduction: Neuro-Symbolic AI

- Neuro-symbolic AI aims to combine neural network (NN) learning and symbolic AI reasoning.
- Problem of interests: injecting discrete logical constraints into NN learning
  - NN can also learn from known constraints/knowledge
  - Higher accuracy, fewer data, interpretable

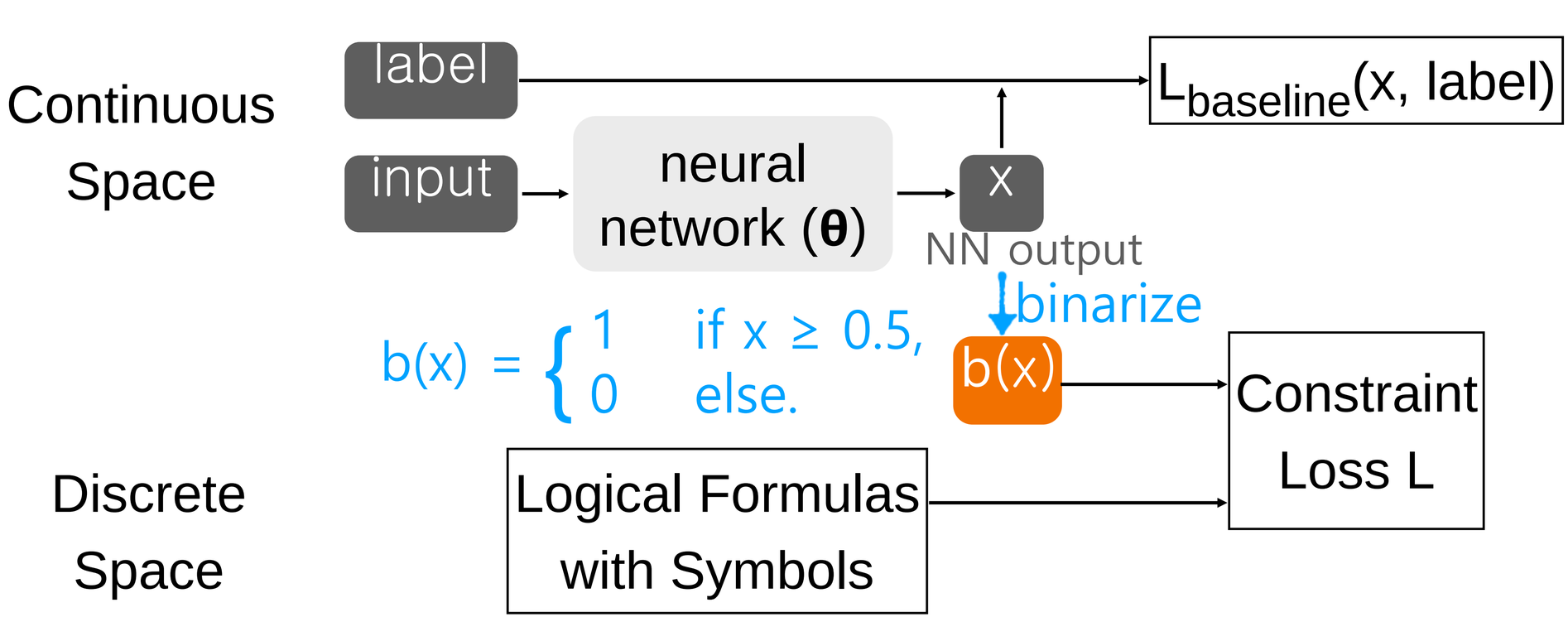
Injecting Constraints into NN

How to define constraint loss using NN output in continuous space and logical formulas in discrete space?



Naïve Idea

- **binarize NN output x into Boolean values b(x).**



Example

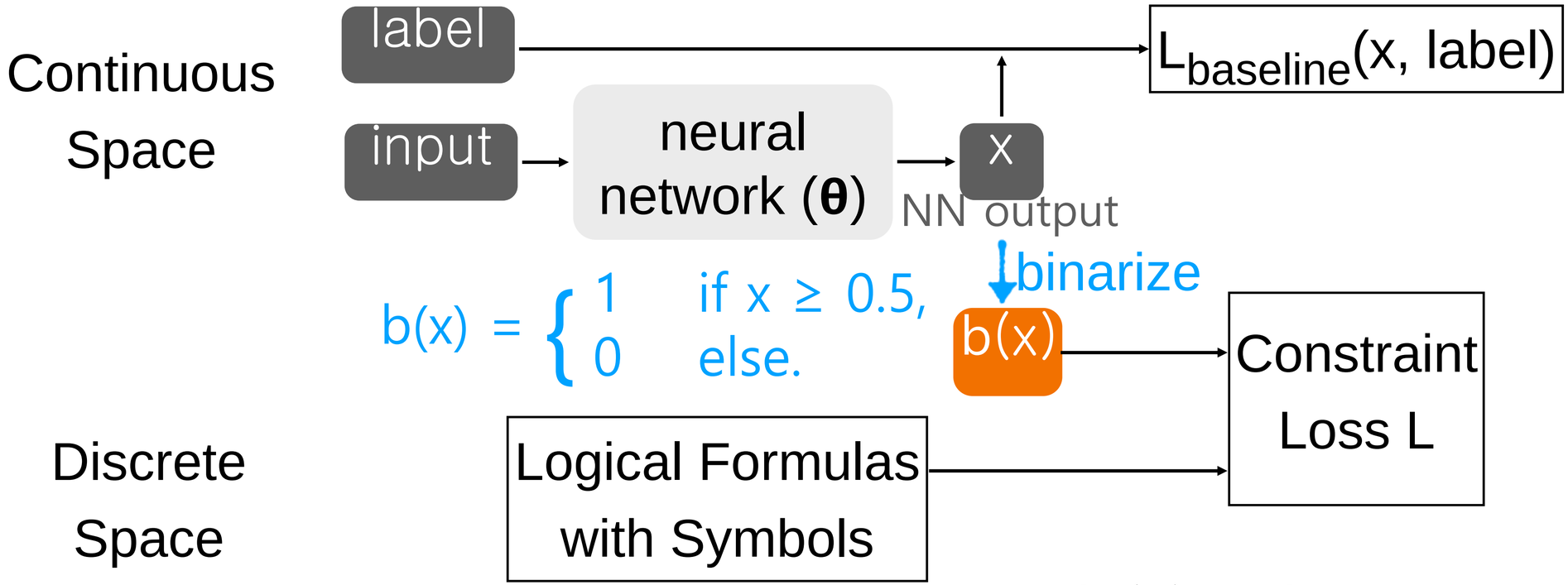
[0.3, 0.1, 0.9]

[0, 0, 1]

p ∨ ¬q ∨ r
¬p ∨ q

Naïve Idea

- **binarize NN output x into Boolean values b(x).**



$$\frac{\partial L}{\partial \theta} = \frac{\partial L}{\partial b(x)} \times \frac{\partial b(x)}{\partial x} \times \frac{\partial x}{\partial \theta}$$

Example

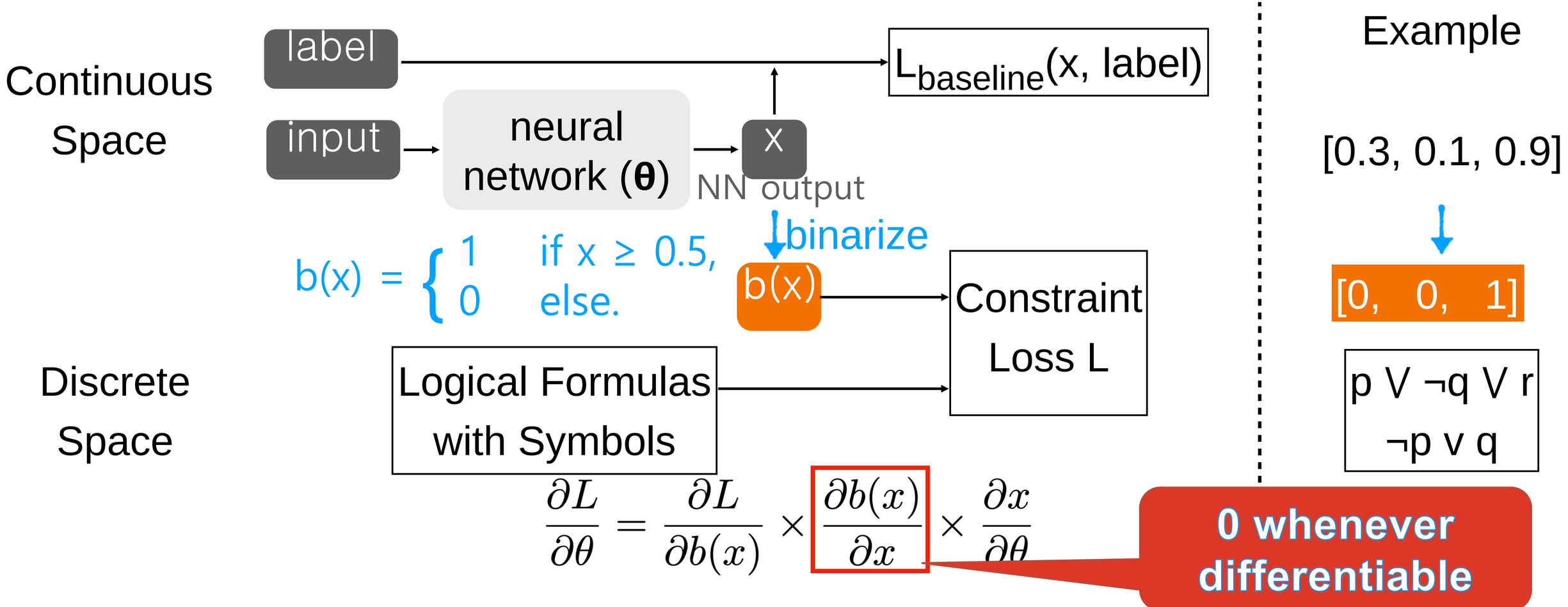
[0.3, 0.1, 0.9]

[0, 0, 1]

p ∨ ¬q ∨ r
¬p ∨ q

Naïve Idea

- **binarize NN output x into Boolean values b(x).**



Existing Methods to Link NN and Constraints

- Probability
  - Loss: -log(Pr(S)) where S denotes all satisfying states
  - Problem: NP-hard (large circuit, enumerate all proofs or stable models)
  - Example: Semantic Loss, DeepProbLog, NeurASP, NeuroLog, etc
- Fuzzy value
  - Loss: -Fuzzy(F) where F is a set of logic formulas
  - Problem: Alters the logical properties of the original theory
  - Example: Logic Tensor Network, Semantic Based Regularization, etc

Straight-Through Estimators (STE)

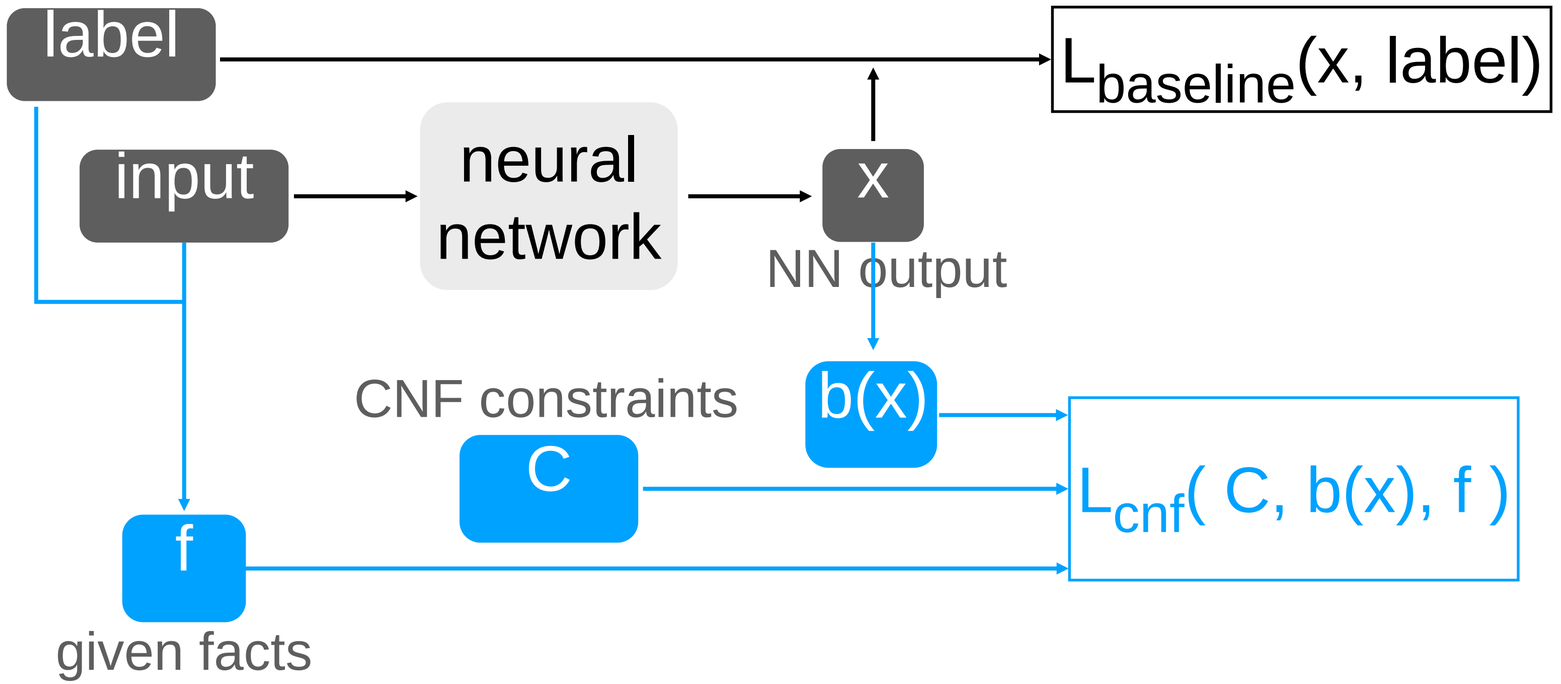
- Originally introduced to train binary neural network [Courbariaux et al., 2015]

- The idea is to replace $\frac{\partial b(x)}{\partial x}$ with $\frac{\partial s(x)}{\partial x}$ where s(x) is a meaningfully differentiable function, e.g., s(x) = x.

$$\frac{\partial L}{\partial \theta} \approx \frac{\partial L}{\partial b(x)} \times \frac{\partial s(x)}{\partial x} \times \frac{\partial x}{\partial \theta}$$

1 when s(x)=x

Our Approach: Constraint Loss via STE (CL-STE)



Constraint Loss L_{cnf}

$$\mathbf{L}_f = \mathbf{C} \odot \mathbf{f} \tag{1}$$

$$\mathbf{L}_v = \mathbb{1}_{\{1\}}(\mathbf{C}) \odot \mathbf{v} + \mathbb{1}_{\{-1\}}(\mathbf{C}) \odot (1 - \mathbf{v}) \tag{2}$$

$$\mathbf{deduce} = \mathbb{1}_{\{1\}}\Big(sum(\mathbf{C} \odot \mathbf{C}) - sum(\mathbb{1}_{\{-1\}}(\mathbf{L}_f))\Big) \tag{3}$$

$$\mathbf{unsat} = prod(1 - \mathbf{L}_v) \tag{4}$$

$$\mathbf{keep} = sum(\mathbb{1}_{\{1\}}(\mathbf{L}_v) \odot (1 - \mathbf{L}_v) + \mathbb{1}_{\{0\}}(\mathbf{L}_v) \odot \mathbf{L}_v) \tag{5}$$

$$L_{deduce} = sum(\mathbf{deduce} \odot \mathbf{unsat}) \tag{6}$$

$$L_{unsat} = avg(\mathbb{1}_{\{1\}}(\mathbf{unsat}) \odot \mathbf{unsat}) \tag{7}$$

$$L_{sat} = avg(\mathbb{1}_{\{0\}}(\mathbf{unsat}) \odot \mathbf{keep}) \tag{8}$$

$$L_{cnf}(\mathbf{C}, \mathbf{v}, \mathbf{f}) = L_{deduce} + L_{unsat} + L_{sat} \tag{9}$$

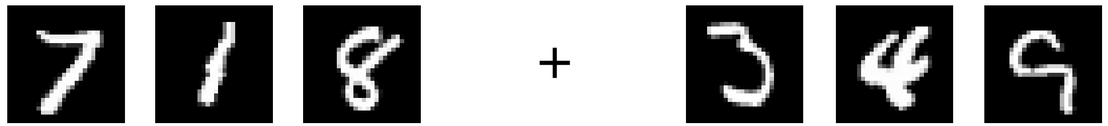
L_{cnf} has 3 components

- Minimizing L_{deduce} makes the NN take a deduction step.
- Minimizing L_{unsat} makes the NN change its predictions to decrease the number of unsatisfied clauses.
- Minimizing L_{sat} makes the NN more confident in its predictions in the satisfied clauses.

CL-STE Advantages

- Much more scalable by leveraging GPU and batch training
  - Ex. mnistAdd-3 problem: given two numbers (each formed by 3 digit images) and their sum as the label, the goal is to train an MNIST classifier.

input **label**

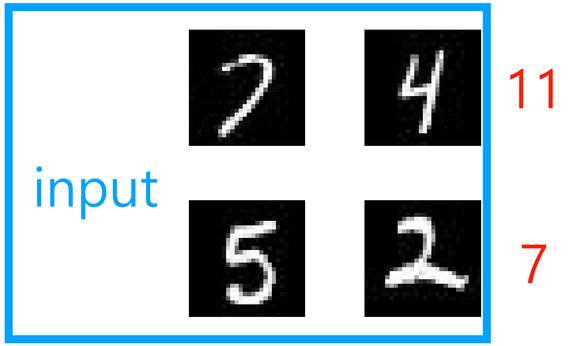
 = 1067

	mnistAdd-1	mnistAdd-2	mnistAdd-3
DeepProbLog	98.36% 2565s	97.57% 22699s	timeout (>24h)
NeurASP	97.87% 292s	97.85% 1682s	timeout (>24h)
CL-STE	97.48% 22s	98.12% 92s	97.78% 402s

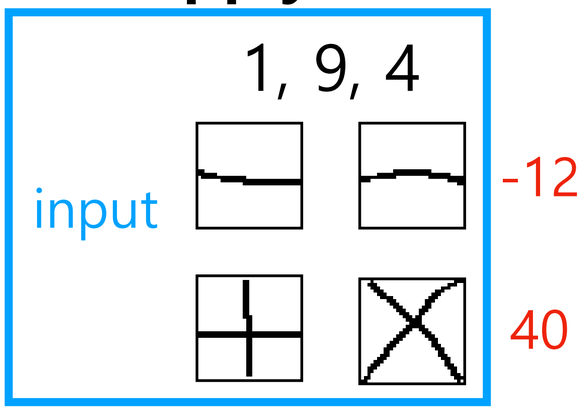
CL-STE: Higher Scalability Across Benchmarks

	add2x2	apply2x2	member(3)	member(5)
accuracy(%)				
DeepProbLog	88.4±0.7	100±0	96.3±0.3	timeout
NeuroLog	97.5±0.4	100±0	94.5±1.5	93.9±1.5
NeurASP	97.6±0.2	100±0	93.5±0.9	timeout
CL-STE	98.0±0.2	100±0	95.5±0.7	95.0±0.5
time(s)				
DeepProbLog	1035±71	586±9	2218±211	timeout
NeuroLog	2400±46	2428±29	427±12	682±40
NeurASP	142±2	47±1	253±1	timeout
CL-STE	54±4	112±2	43±3	49±4

add2x2

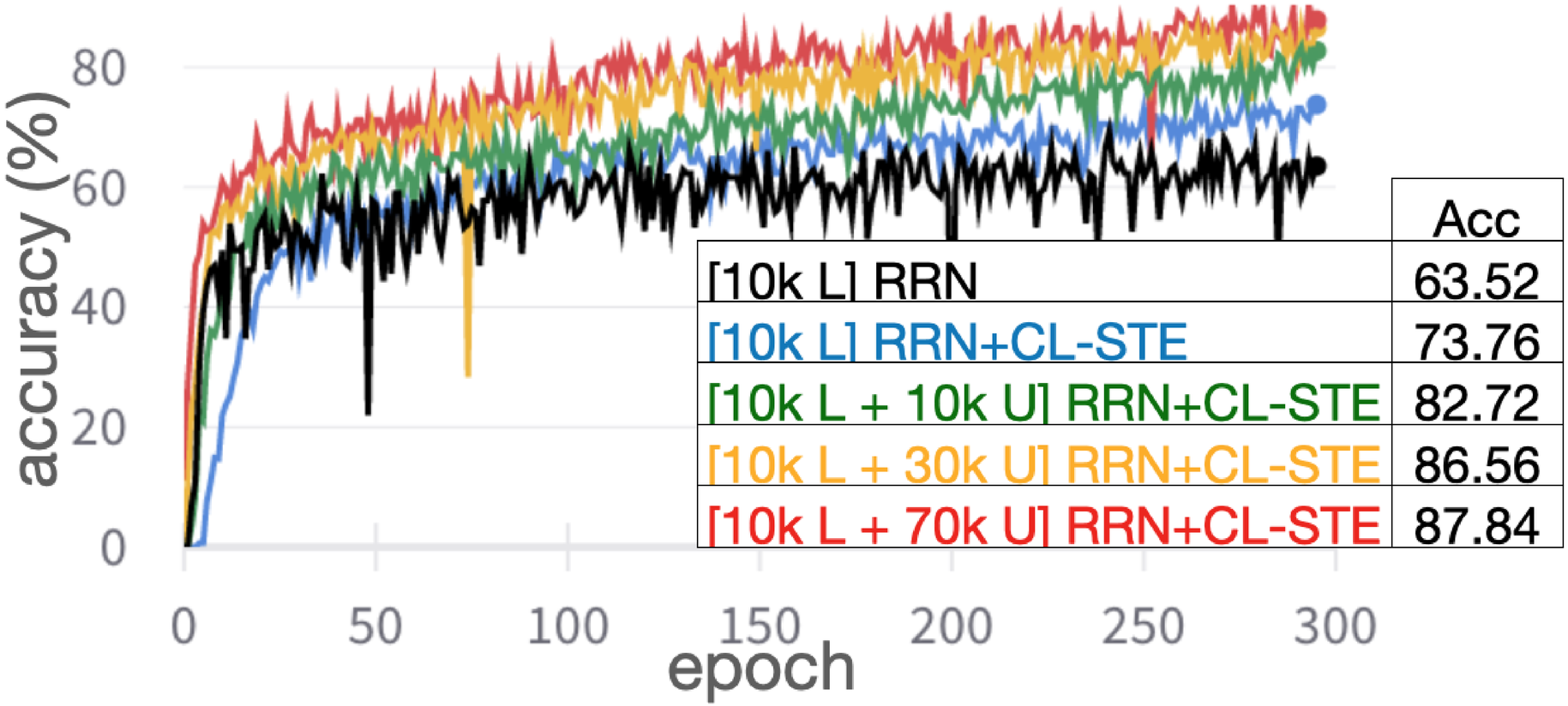


apply2x2



CL-STE on Semi-Supervised Learning

- Under semi-supervised setting, more unlabeled data, bigger improvement.
  - CL-STE helps to train Recurrent Relational Network (a GNN) better on textual Sudoku problem



	Acc
[10k L] RRN	63.52
[10k L] RRN+CL-STE	73.76
[10k L + 10k U] RRN+CL-STE	82.72
[10k L + 30k U] RRN+CL-STE	86.56
[10k L + 70k U] RRN+CL-STE	87.84

Summary

- CL-STE encodes CNF formulas into a loss function on discretized NN outputs where
  - the STE method makes a binarization function meaningfully differentiable,
  - by leveraging GPUs and batch training, CL-STE scales significantly better compared to state-of-the-art neuro-symbolic methods,
  - CL-STE works well with both labeled and unlabeled data, and with different types of NNs: MLP, CNN, and GNN.

Thank you!