Maynooth University
National University of Ireland Maynooth

Property-driven Machine Learning with Differentiable Logics

Thomas Flinkow

Department of Computer Science
Maynooth University

DAIR Course @ Heriot-Watt University
25th September 2024

About

Principles of Programming Research Group @ Maynooth University

https://www.cs.nuim.ie/research/pop/

Principles of Programming Research Group

Principal Researchers: Rosemary Monahan, Hao Wu, Barak A. Pearlmutter, Kevin Casey

2 PostDocs: Medet Inkarbekov, Ali Bukhari

5 PhD Researchers: Oisín Sheridan, Dara MacConville, Thomas Flinkow, Ankit Jha, Huan Zhang

Projects:

- MAIVV: Modular AI Verification and Visualisation (2021 – 2025, SFI)
- VerifAI: Writing Formal Requirements using AI (2024 – 2026, SFI ADAPT)
- VALU3S: Verification and Validation of Automated Systems' Safety and Security (2020 – 2023, EI and H2020 ESCEL)
- A Model Checker for Python (2022 – 2026, SFI CRT Foundations of Data Science)
- A Constructive Framework for Software Specification and Refinement (2012 – 2023, 2 × IRC)

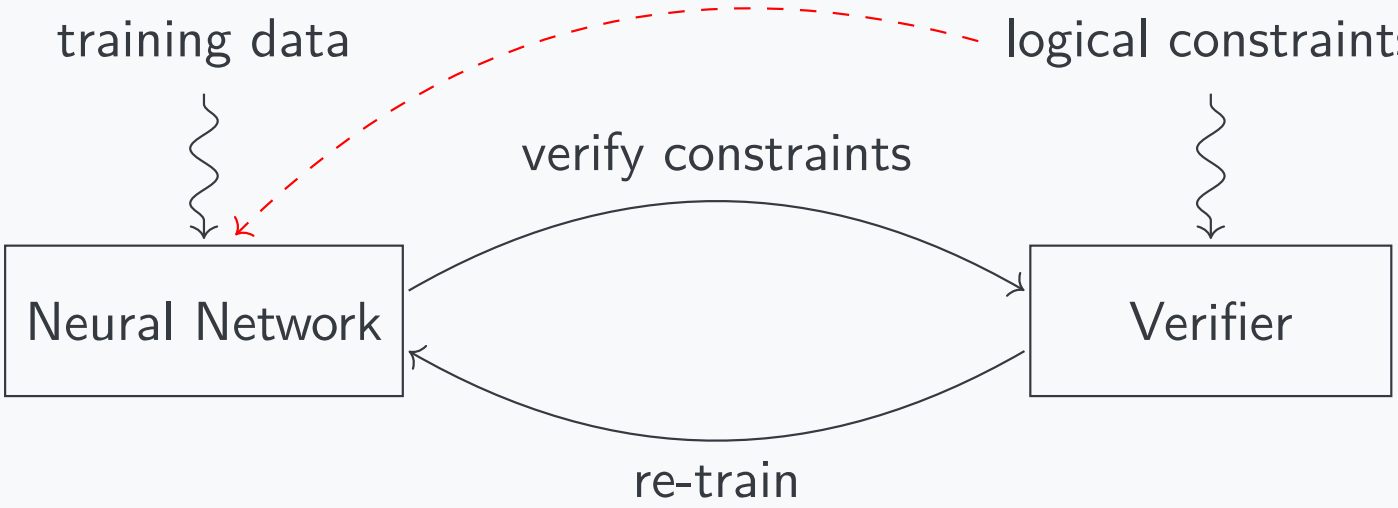
Background

Training with Logical Constraints

Task: train a neural network $\mathcal{N}$ to satisfy constraint ϕ.

Train: given data, labels, and loss function, iteratively update network weights.

Verify: afterwards, α, β-CROWN, Marabou, NNV, ERAN, ...



Note

Training with constraints does not guarantee their satisfaction!

Property-driven Training with Differentiable Logics

Given data $\boldsymbol{x}_0$ and label $\boldsymbol{y}$, and constraint ϕ,
obtain optimal network weights $\boldsymbol{\theta}^+$ by

$$\boldsymbol{\theta}^+ = \arg\min_{\boldsymbol{\theta}} \ \alpha\mathcal{L}_{\mathsf{CE}}(\boldsymbol{x}_0, \boldsymbol{y}) + \beta\mathcal{L}_{\mathsf{C}}(\boldsymbol{x}_0, \boldsymbol{y}, \phi).$$

Optimisation handled by Gradient Descent:

$$\boldsymbol{\theta}_{i+1} := \boldsymbol{\theta}_i - \eta\nabla\mathcal{L}(\boldsymbol{\theta}_i)$$

Implications

Logical constraint ϕ must be differentiable!

DL2[1]

- mapping $[\![\cdot]\!]_{\mathsf{DL2}} : \Phi \to [0, \infty)$,
- $[\![\phi]\!]_{\mathsf{DL2}} = 0$ iff ϕ is satisfied,
- $[\![\phi]\!]_{\mathsf{DL2}}$ is differentiable almost everywhere.

Recursive definition of loss translation:

$$
\begin{aligned}
[\![x \leq y]\!]_{\mathsf{DL2}} &:= \max\{x - y, 0\} \\
[\![\phi \wedge \psi]\!]_{\mathsf{DL2}} &:= [\![\phi]\!]_{\mathsf{DL2}} + [\![\psi]\!]_{\mathsf{DL2}} \\
[\![\phi \vee \psi]\!]_{\mathsf{DL2}} &:= [\![\phi]\!]_{\mathsf{DL2}} \cdot [\![\psi]\!]_{\mathsf{DL2}}.
\end{aligned}
$$

[1] Marc Fischer et al.: 'DL2: Training and Querying Neural Networks with Logic' 2019.

Fuzzy Logics[2,3]

- mapping $[\![\cdot]\!]_L : \Phi \to [0,1]$, where $[\![\top]\!]_L = 1$ and $[\![\bot]\!]_L = 0$,
- operators happen to be differentiable almost everywhere

Logic	Conjunction	Disjunction	Implication
Gödel	$\min\{x,y\}$	$\max\{x,y\}$	$\begin{cases} 1, & \text{if } x < y, \\ y, & \text{else.} \end{cases}$
Kleene-Dienes			$S(N(x),y)$
Łukasiewicz	$\max\{0,x+y-1\}$	$\min\{1,x+y\}$	
Reichenbach			
Goguen	xy	$x+y-xy$	$\begin{cases} 1, & \text{if } x < y, \\ y^x, & \text{else.} \end{cases}$

[2]Natalia Ślusarz et al.: 'Logic of Differentiable Logics: Towards a Uniform Semantics of DL' 2023. DOI: 10.29007/c1nt.

[3]Emile van Krieken et al.: 'Analyzing Differentiable Fuzzy Logic Operators' 2022. DOI: 10.1016/j.artint.2021.103602.

Constraint Counterexamples

Given data $\boldsymbol{x}_0$ and label $\boldsymbol{y}$, and constraint ϕ,
obtain optimal network weights $\boldsymbol{\theta}^+$ by

$$\boldsymbol{\theta}^+ = \arg\min_{\boldsymbol{\theta}} \ \alpha\mathcal{L}_{\text{CE}}(\boldsymbol{x}_0, \boldsymbol{y}) + \beta\mathcal{L}_{\text{C}}(\boldsymbol{x}_0, \boldsymbol{y}, \phi).$$

Insight from DL2 (Fischer et al. 2019)

Learning to satisfy $\forall x.\, x \vDash \phi$ by finding x^* such that $x^* \nvDash \phi$.

1. Approximate counterexample *outside* of training set using PGD:

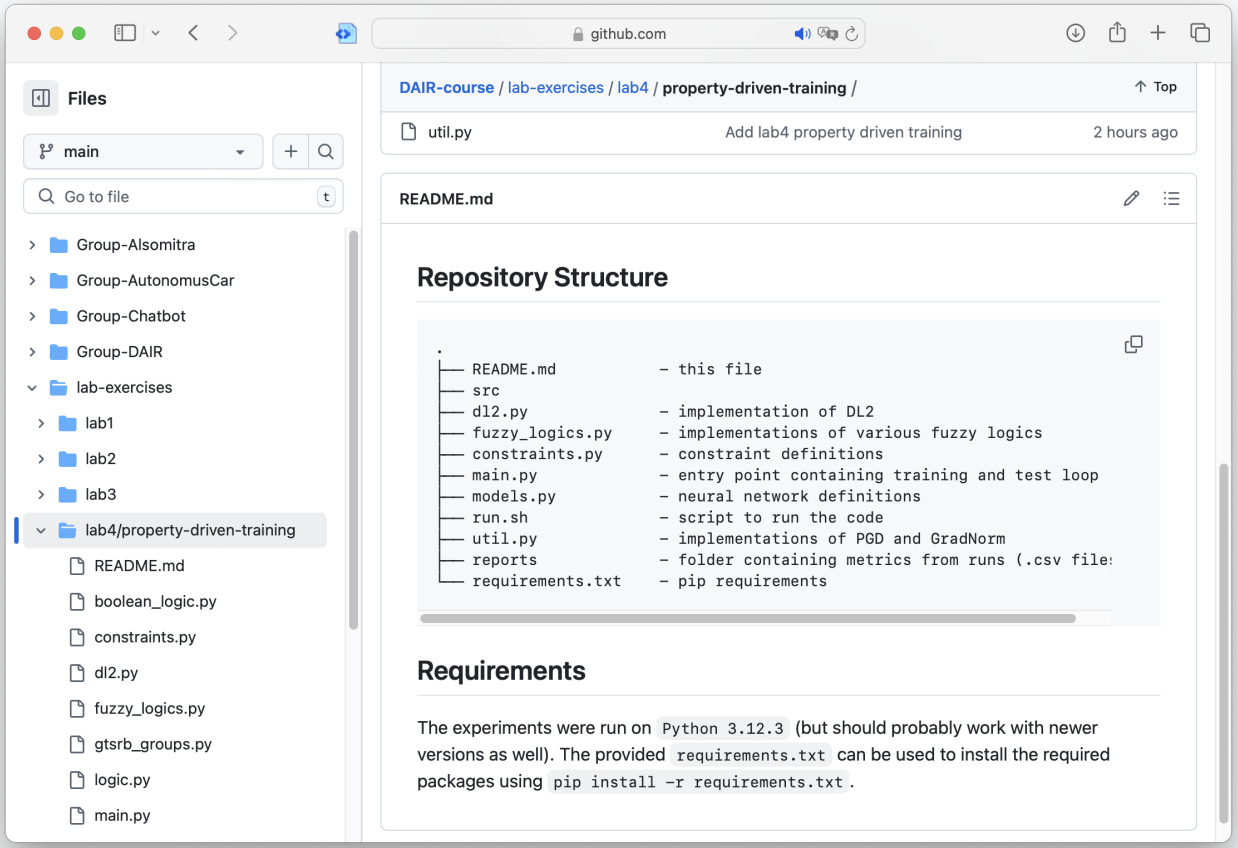
$$\boldsymbol{x}^* = \arg\max_{\boldsymbol{x} \in \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \epsilon} \mathcal{L}_{\text{C}}(\boldsymbol{x}_0, \boldsymbol{x}, \boldsymbol{y}, \phi)$$

2. Use this counterexample in training:

$$\boldsymbol{\theta}^+ = \arg\min_{\boldsymbol{\theta}} \ \alpha\mathcal{L}_{\text{CE}}(\boldsymbol{x}_0, \boldsymbol{y}) + \beta\mathcal{L}_{\text{C}}(\boldsymbol{x}_0, \boldsymbol{x}^*, \boldsymbol{y}, \phi).$$

Code

Full Code On DAIR Course GitHub[4]



[4] https://github.com/KatyaKom/DAIR-course/tree/main/lab-exercises/lab4/property-driven-training/

Integration into PyTorch

Standard PyTorch training + logical property

```
def train(..):
    for _, (inputs, labels) in enumerate(train_loader):
        outputs = NN(inputs)
        ce_loss = F.cross_entropy(outputs, labels)

        adv = pgd.attack(NN, inputs, labels, constraint)
        dl_loss = constraint.eval(NN, inputs, adv, labels)

        loss = alpha * ce_loss + beta * dl_loss

        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Balancing Loss

Problem

It is *crucial* to find close to optimal values for α and β!

$$\boldsymbol{\theta}^{+} = \arg\min_{\boldsymbol{\theta}} \; \alpha \mathcal{L}_{\mathrm{CE}}(\boldsymbol{x}_0, \boldsymbol{y}) + \beta \mathcal{L}_{\mathrm{C}}(\boldsymbol{x}_0, \boldsymbol{x}^*, \boldsymbol{y}, \phi).$$

Adaptive Loss Balancing with GradNorm[5]

- Key point: $\alpha(t)$ and $\beta(t)$
- Better results than expensive grid search!

Loss balancing in code[6]

```
loss = grad_norm.weighted_loss(batch_index, ce_loss, dl_loss, alpha)
# loss = alpha * ce_loss + beta * dl_loss
```

[5] Zhao Chen et al.: 'GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks' 2018.

[6] https://github.com/KatyaKom/DAIR-course/blob/main/lab-exercises/lab4/property-driven-training/main.py#L56

Basics – Logic[7]

```
class Logic(ABC):
    @abstractmethod
    def LEQ(self, x, y):
        pass

    @abstractmethod
    def NOT(self, x, y):
        pass

    @abstractmethod
    def AND(self, x, y):
        pass

    @abstractmethod
    def OR(self, x, y):
        pass

    def IMPL(self, x, y):
        return self.OR(self.NOT(x), y)

    def EQUIV(self, x, y):
        return self.AND(self.IMPL(x, y), self.IMPL(y, x))
```

[7]

Basics – Constraints[8]

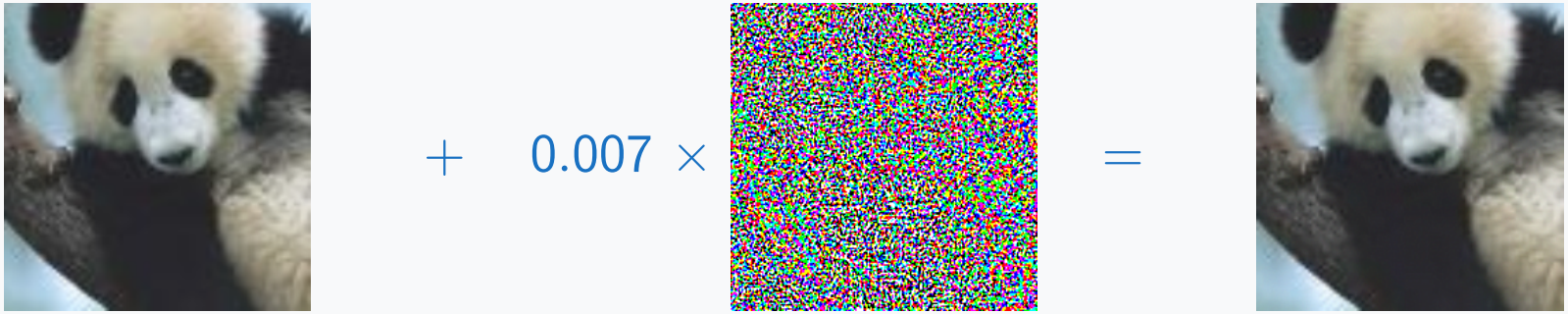
```
class Constraint(ABC):
    def __init__(self, eps):
        self.eps = eps

    @abstractmethod
    def get_constraint(self, NN, inputs, adv, labels):
        pass

    # usage:    loss, sat = eval()
    def eval(self, NN, inputs, adv, labels, logic, train):
        ...
```

Remark

Why ϵ in the constructor? All constraints use PGD!

[8] https://github.com/KatyaKom/DAIR-course/tree/main/lab-exercises/lab4/property-driven-training/constraints.py#L14

Local Robustness Constraint



‘panda’
57.7 % confidence

adversarial
noise

‘gibbon’
99.3 % confidence

Figure 1: Adversarial attack (Goodfellow et al. 2015).

Definition

A neural network is locally robust in input $\boldsymbol{x}_0$, if

$$\underbrace{\forall \boldsymbol{x}.\, \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \varepsilon}_{\text{all elements in the input space close to } \boldsymbol{x}_0} \quad \text{implies} \quad \underbrace{\|\mathcal{N}(\boldsymbol{x}) - \mathcal{N}(\boldsymbol{x}_0)\|_\infty \leq \delta}_{\text{the classification is roughly the same}}$$

Local Robustness Constraint – Code[9]

Definition

A neural network is locally robust in input $\boldsymbol{x}_0$, if

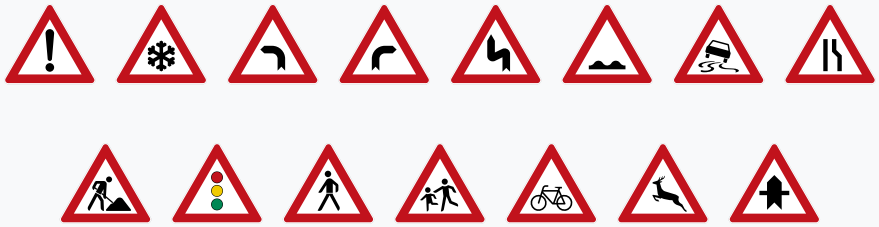
$$\underbrace{\forall \boldsymbol{x}.\, \|\boldsymbol{x} - \boldsymbol{x}_0\|_\infty \leq \varepsilon}_{\text{handled by PGD}} \quad \text{implies} \quad \underbrace{\|\mathcal{N}(\boldsymbol{x}) - \mathcal{N}(\boldsymbol{x}_0)\|_\infty \leq \delta}_{\text{the classification is roughly the same}}$$

```
class RobustnessConstraint(Constraint):
    def __init__(self, eps, delta):
        super().__init__(eps)
        self.delta = delta

    def get_constraint(self, NN, inputs, adv, _labels):
        diff = F.softmax(NN(adv)) - F.softmax(NN(inputs))
        norm = torch.linalg.vector_norm(diff, ord=float('inf'))

        return lambda l: l.LEQ(norm, self.delta)
```

[9] https://github.com/KatyaKom/DAIR-course/tree/main/lab-exercises/lab4/property-driven-training/constraints.py#L45

Group Constraint

(a) unique signs



(b) danger signs

(c) derestriction signs

(d) speed limit signs

(e) other prohibitory signs

(f) mandatory signs

Definition

$$\underbrace{\forall \boldsymbol{x} \in ||\boldsymbol{x} - \boldsymbol{x}_0|| \leq \epsilon}_{\text{handled by PGD}} \quad \rightarrow \quad \bigwedge_{G \in \mathcal{G}} p_G \leq \delta \ \vee \ p_G \geq 1 - \delta.$$

Group Constraint – Code[10]

Definition

$$\underbrace{\forall \boldsymbol{x} \in ||\boldsymbol{x} - \boldsymbol{x}_0|| \leq \epsilon}_{\text{handled by PGD}} \quad \rightarrow \quad \bigwedge_{G \in \mathcal{G}} p_G \leq \delta \ \vee \ p_G \geq 1 - \delta.$$

```
class GroupConstraint(Constraint):
   def __init__(self, eps, delta, groups):
      super().__init__(eps)
      self.delta = delta

   def get_constraint(self, NN, _inputs, adv, _labels):
      probs = F.softmax(NN(adv))
      sums = [torch.sum(probs[:, indices]) for indices in group_indices]

      return lambda l: reduce(l.AND, \
         [l.OR(l.LEQ(s, delta), l.LEQ(1. - delta, s)) for s in sums])
```

[10] https://github.com/KatyaKom/DAIR-course/tree/main/lab-exercises/lab4/property-driven-training/constraints.py#L55

Specifying Hypershapes[11]

Defining ℓ_∞ hyperrectangles

```
hyper_rectangles = []

for data, _ in train_loader:
    data = data.to(device).view(data.size(0), -1)
    hyper_rectangles.append(torch.stack((data - eps, data + eps)))
```

[11] https://github.com/KatyaKom/DAIR-course/tree/main/lab-exercises/lab4/property-driven-training/main.py#L208

Thank you!
Any questions?

References

Chen, Zhao et al. (July 2018). 'GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks'. In: *Proceedings of the 35th International Conference on Machine Learning*. PMLR, pp. 794–803.

Fischer, Marc et al. (24th May 2019). 'DL2: Training and Querying Neural Networks with Logic'. In: *Proceedings of the 36th International Conference on Machine Learning*. International Conference on Machine Learning. PMLR, pp. 1931–1941.

Goodfellow, Ian J. et al. (Mar. 2015). *Explaining and Harnessing Adversarial Examples*. DOI: 10.48550/arXiv.1412.6572. arXiv: 1412.6572 [cs, stat].

References (cont.)

Ślusarz, Natalia et al. (3rd June 2023). 'Logic of Differentiable Logics: Towards a Uniform Semantics of DL'. In: *EPiC Series in Computing*. Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning. Vol. 94. EasyChair, pp. 473–493. DOI: 10.29007/c1nt.

Van Krieken, Emile et al. (Jan. 2022). 'Analyzing Differentiable Fuzzy Logic Operators'. In: *Artificial Intelligence* 302, p. 103602. ISSN: 00043702. DOI: 10.1016/j.artint.2021.103602. arXiv: 2002.06100 [cs].