A Verified Neurosymbolic Pipeline for Learning Linear Temporal Behaviour

Mark Chevallier, Jacques Fleuriot

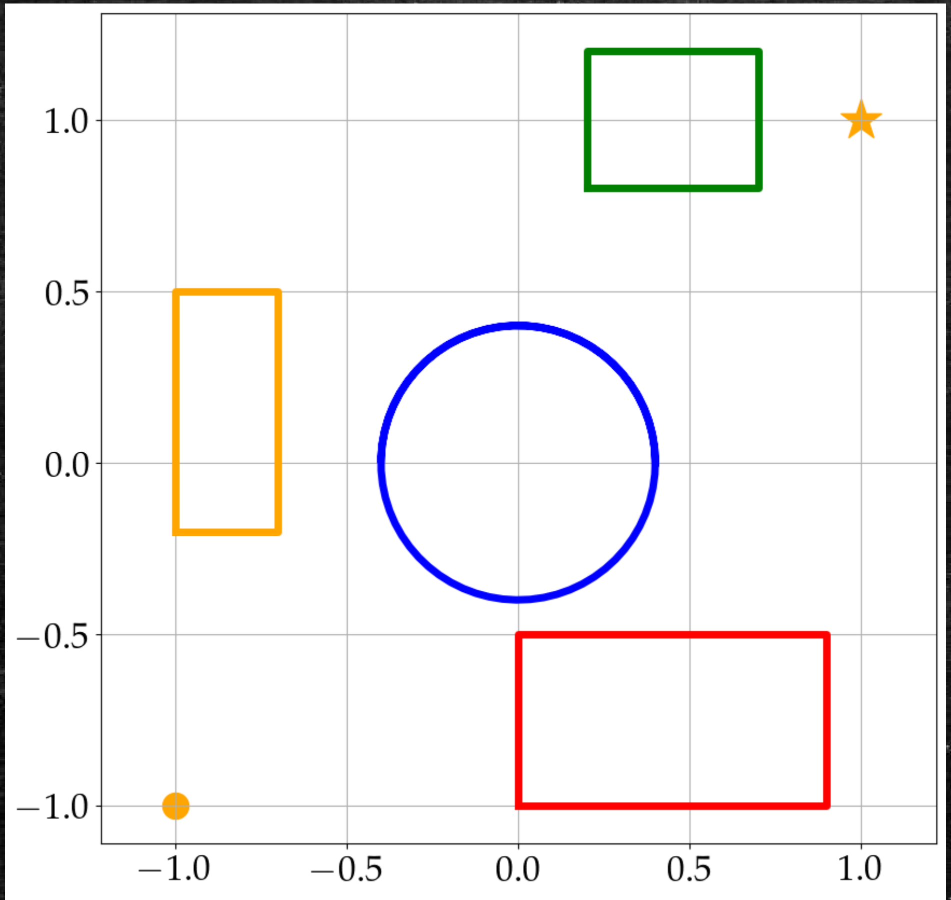
Lecture Structure

Our work on an end-to-end pipeline to inject LTL rules into NN learning

- Train NNs to satisfy temporal constraints
- Improved guarantees of faithfulness to specification
- Experiments show interesting interactions with the domain

1. Introduction to the problem
2. LTL
3. Isabelle formalisation of LTL
4. Implementation and Experiments

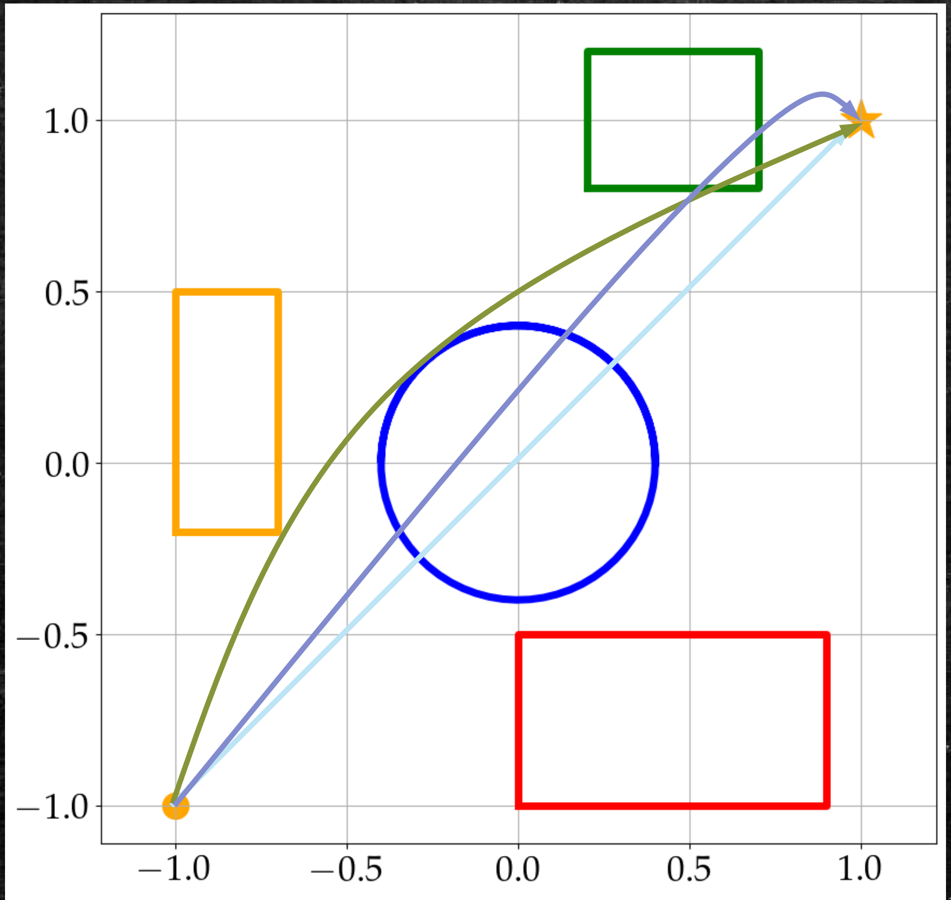
A Problem



Rules based movement in time

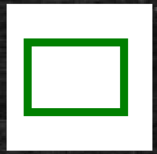
Guaranteed fidelity to rules

Problem: Linear Temporal Logic (LTL)

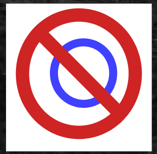


Eventually

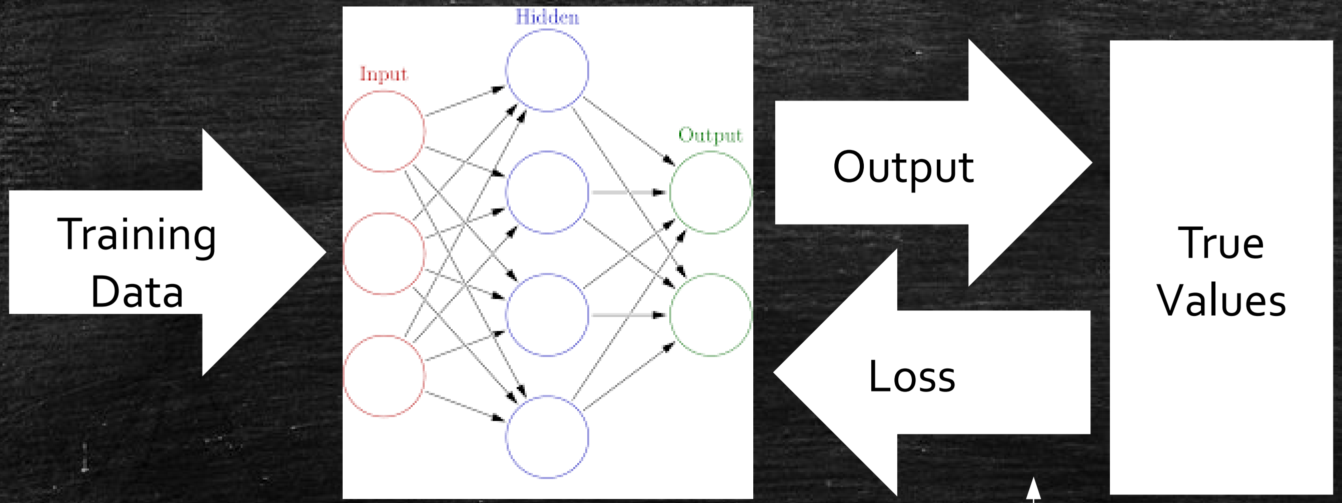
Eventually



Always



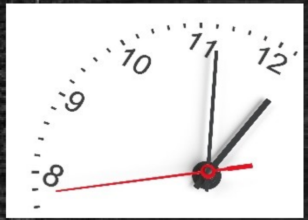
Problem: Integrating LTL into neural networks



Insert extra loss here - violation of LTL

Need *differentiable* loss function
Complicated! Prone to errors

Problem: What does our pipeline offer?



Time based property checking

Faithfulness to specification

Easily adopted for different tasks

Specification → Code generation → Calculation → Code used in → Learning

Isabelle | OCaml | PyTorch

Specification

Proof of properties

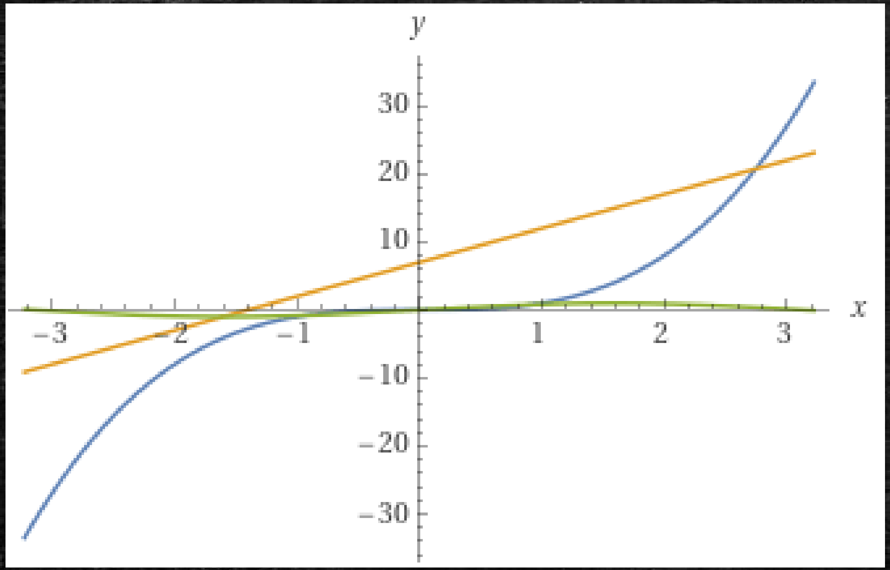
Calculation

Learning

Lecture Structure

1. Introduction to the problem
2. LTL
3. Isabelle formalisation of LTL
4. Implementation and Experiments

LTL: Traces



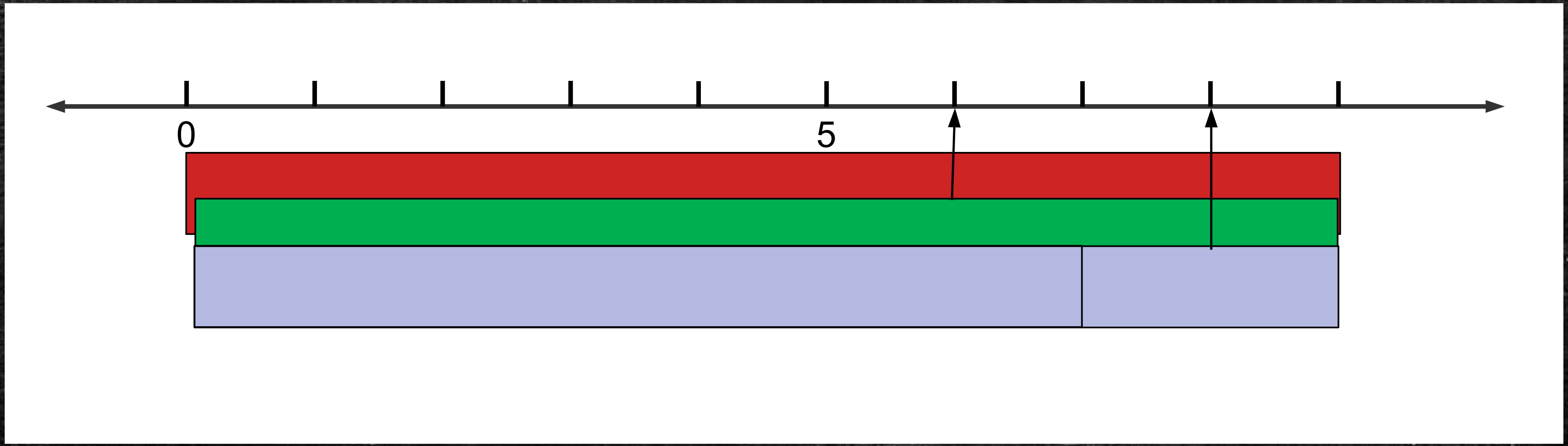
Discrete time!

Trace

Time	p_1	p_2	p_3
-3	0	-8	-30
-2	-1	-4	-8
-1	-1	4	-1
0	0	8	0
1	1	12	1
2	1	16	8

LTL: Operators

And, Or



Always
Eventually
Until
Next, Release

LTL: Not having "Not"

Not is usually primitive

But causes issues with soundness

We can use Not as a function instead

Constraint A → Not → Constraint B

Constraint B is equivalent to !(Constraint A)

LTL in Finite Time

LTL is over infinite time…

LTL*f* is over *finite* time

So what happens here?

Need to decide if Until/Release/Next are Strong/Weak

Lecture Structure

1. Introduction to the problem
2. LTL
3. Isabelle formalisation of LTL
4. Implementation and Experiments

Isabelle: Formalisation

LTL

L function

dL function

Prove properties

Generate code

Isabelle: Deep embedding of LTL

Operators →

```
datatype comp = Less int int | Lequal int int | Equal int int
  | Nequal int int

datatype constraint = Comp comp | And constraint constraint
  | Or constraint constraint | Next constraint | Always constraint
  | Eventually constraint | Until constraint constraint
  | Release constraint constraint
```

```
type_synonym state = "int ⇒ real"
type_synonym path = "state list"
```

Traces and states

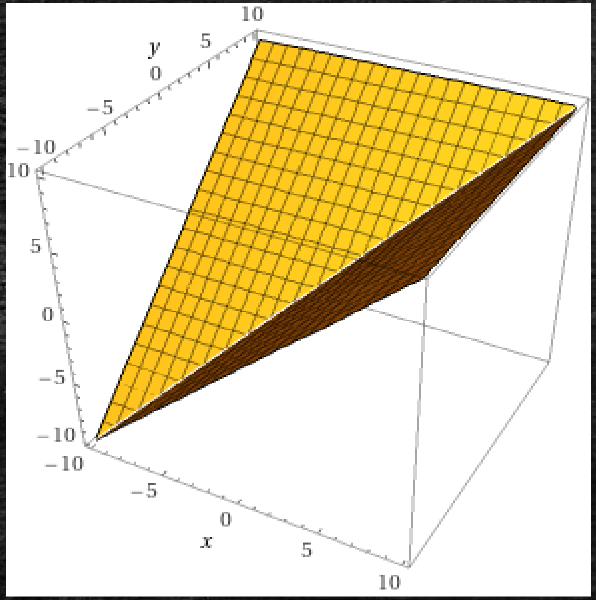
```
function eval :: "constraint ⇒ path ⇒ bool"
```

LTL constraint

trace

evaluation

Isabelle: Smooth functions



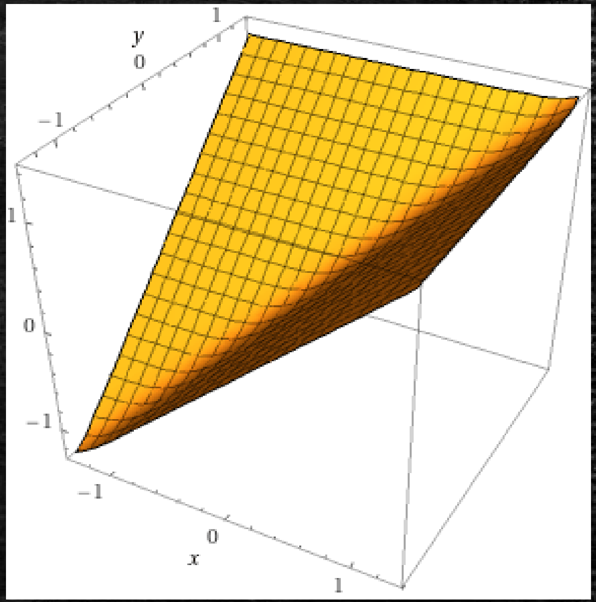
Max functions needed for smooth loss function

But not differentiable everywhere!

Smooth version of max

0.1*log(exp(x/0.1)+exp(y/0.1))

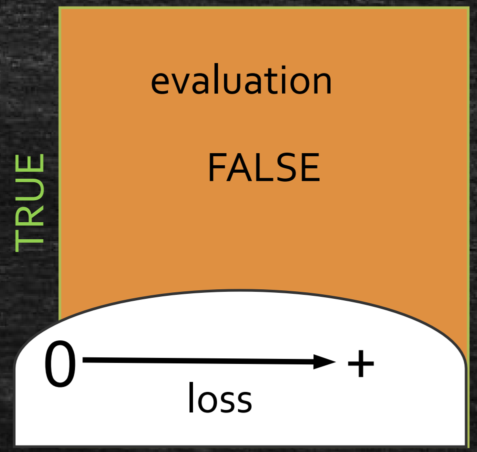
0.1 = smoothing factor gamma



Isabelle: L function

L c p g = 0 iff eval c p g (as g tends to 0)

Derivative of L (dL) also



loss is sound wrt LTL semantics

softening factor
trace
loss
LTL constraint

```
function L :: "constraint ⇒ path ⇒ real ⇒ real"
```

Isabelle: Structure of L

Comparisons come first!

```
"Lequal_gamma γ a b = Max_gamma γ (a-b) 0"
```

```
"L (Comp (Equal v1 v2)) (s # ss) γ =
  Max_gamma_comp γ (L (Comp (Lequal v1 v2)) (s # ss) γ)
    (L (Comp (Lequal v2 v1)) (s # ss) γ)"
```

Rest of function translated simply

```
"eval (Until c1 c2) (s # ss) = ((((eval c1 (s # ss))
   ∧ (if ss = [] then True else (eval (Until c1 c2) ss))))
      ∨ eval c2 (s # ss))"
```

⇓

```
"L (Until c1 c2) (s # ss) γ = Min_gamma γ (L c2 (s # ss) γ)
  (Max_gamma γ (L c1 (s # ss) γ) (if ss = [] then 0
    else (L (Until c1 c2) ss γ)))"
```

Or becomes Min

And becomes Max

Extra parameter gamma for smoothing

Isabelle: Proofs

```
lemma L_eval_sound:
  fixes c :: constraint and ss :: path
  shows "((λγ. L c ss γ) —0→ 0) = (eval c ss)"
```

Soundness of loss function

```
lemma Eventually_works:
  fixes ss :: path and c :: constraint
  shows "(∃n < length ss. eval c (drop n ss)) = eval (Eventually c) ss"
```

LTL semantics match expectations

What can we prove?

```
theorem L_has_derivative:
  fixes x γ :: real and pth :: path
  assumes gamma_gt_zero: "γ > 0"
  shows "⋀c i j. ((λy. L c (update_path pth i j y) γ)
  has_field_derivative (dL c (update_path pth i j x) γ i j)) (at x)"
```

Derivative function is correct

Isabelle: How do we prove against L?

INDUCTION!

…along two objects

Size of the path (base case empty path)

Along constraint (each operator)

Isabelle: Code Generation

```
fun Bell_gamma :: "real ⇒ real ⇒ real" where
  "Bell_gamma γ x =
  (if γ≤(0::real) then (Nzero x) else (1::real)/exp((x*x)/(γ+γ)))"
```

Isabelle specification

thin translation layer, no manual intervention

OCaml code

```
let rec bell_gamma
  gamma x =
    (if Pervasives.(<=) gamma 0.0 then nzero x
      else Pervasives.( /. ) 1.0
             (Pervasives.exp
               (Pervasives.( /. ) (Pervasives.( *. ) x x)
                 (Pervasives.( +. ) gamma gamma))));;
```

Lecture Structure

1. Introduction to the problem
2. LTL
3. Isabelle formalisation of LTL
4. Implementation and Experiments

Experiments: Using OCaml with Python

Python library by Laurent Mazare

Restricted subset of OCaml

Using Python and OCaml in the same Jupyter notebook

DEC 16, 2019 | 11 MIN READ

By: Laurent Mazare

Experiments: DMP neural network

DMPs use differential equations to describe a path

NNs learn to imitate paths of motion using DMPs

Can we use LTL to learn?

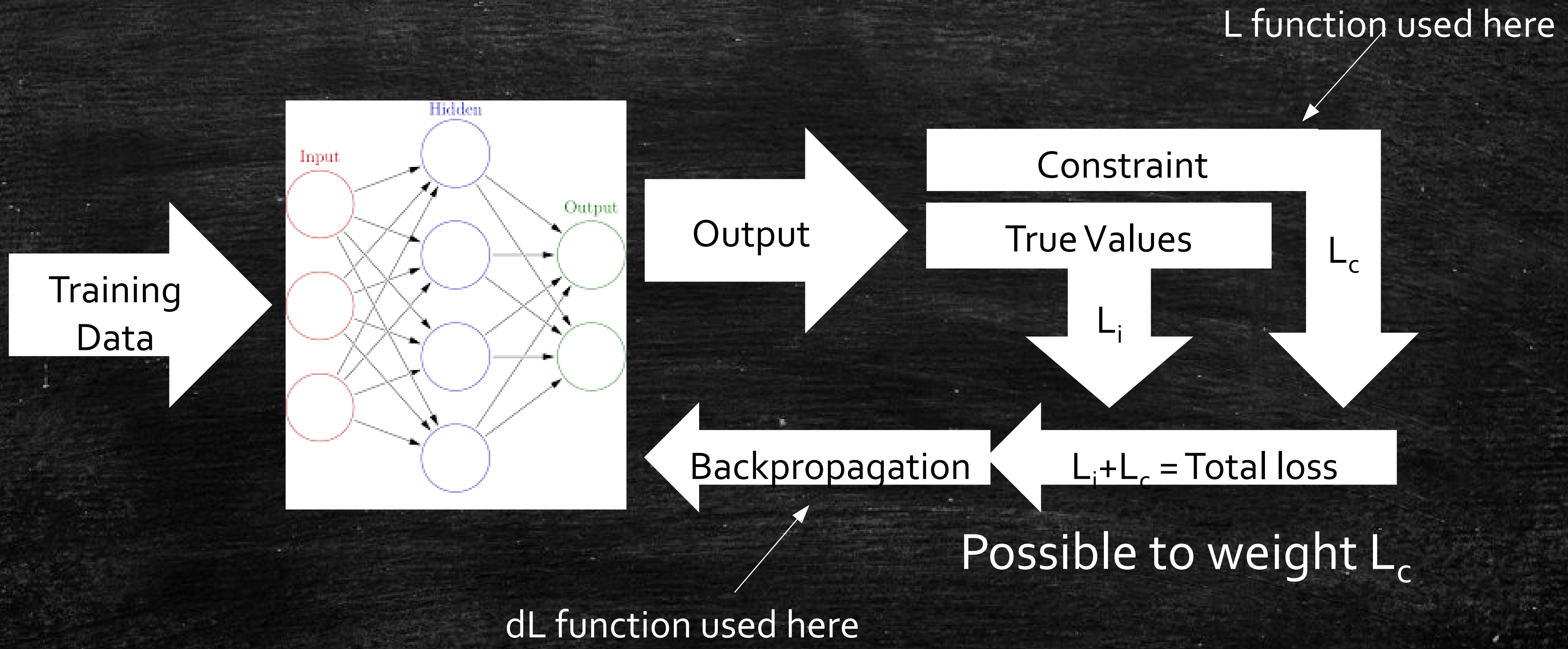
ADVANTAGES

Smooth paths
Trajectory structure "built in"

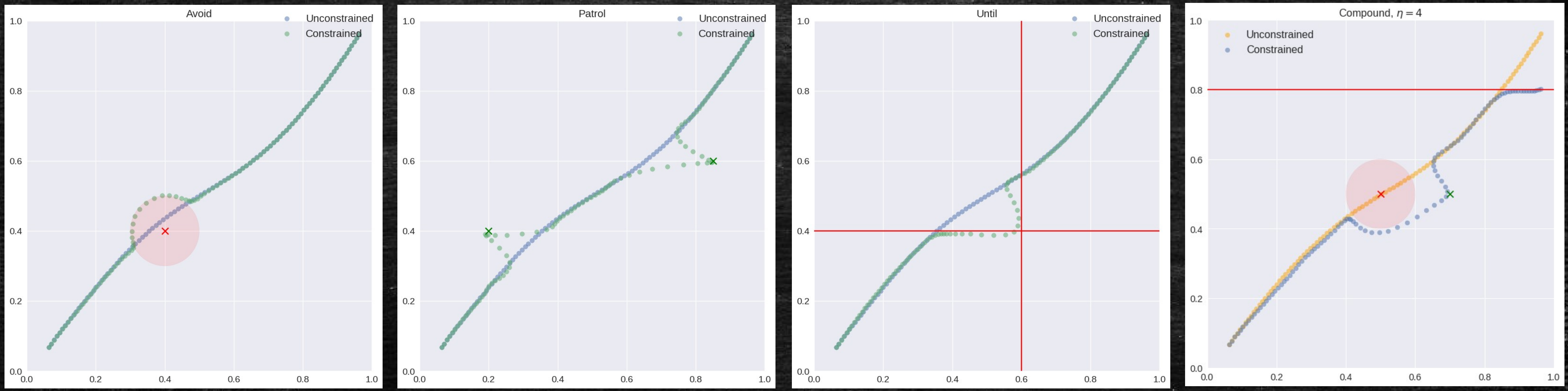
DISADVANTAGES

Indirect

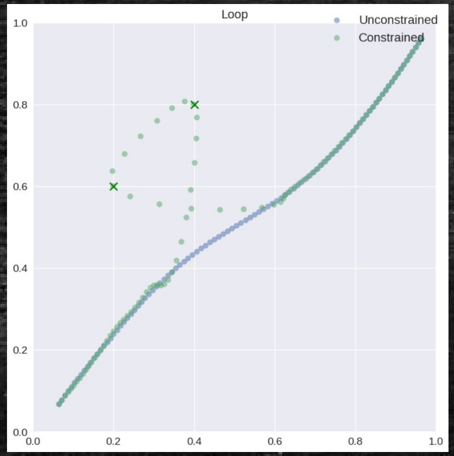
Experiments: How NN Learns from Constraints



Experiments: Results

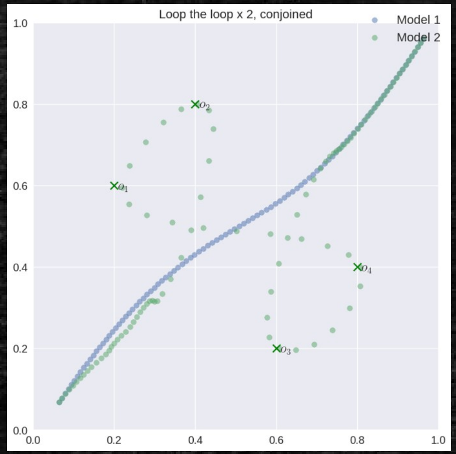


Experiments: More complicated paths



Eventually (reach pt 1 AND Eventually (reach pt 2))

Nested temporal constraints costly – $O(t^n)$



Fully nested constraint very very costly and slow

DMP allows CONJOINED constraint instead

MUCH faster! $O(t^2 + t^2)$

Does not work on more direct path prediction methods

Further developments

More experiments

- Planning schedules
- Robotic movement

Tensor implementation

- Much faster execution
- More practical implementation

More complicated logics…

Conclusion

End-to-end pipeline to inject LTL rules into NN learning

- Train NNs to satisfy temporal constraints
- Guarantee faithfulness to specification
- Domain can impact results

Chevallier, M., Whyte, M., & Fleuriot, J. D. (2022). *Constrained training of neural networks via theorem proving*. In Short Paper Proceedings of the 4th Workshop on Artificial Intelligence and Formal Verification, Logic, Automata, and Synthesis, Vol. 3311. CEUR-WS.org.