Introduction to the Logic Tensor Networks (LTN) Framework

Assembled for PhD Students of Ernesto & Tillman Interested in Neural-Symbolic Integration (NSI)

Dave Herron

City, University of London

December 2021

CITY
UNIVERSITY OF LONDON
EST 1894

Introduction

What we're going to talk about

- see Navigation bar

Introduction

What we're going to talk about

- see Navigation bar

Key messages:

Introduction

What we're going to talk about

- see Navigation bar

Key messages:

- LTN is a **framework** for combining **prior knowledge** with **data** in the training of **conventional** neural networks

Introduction

What we're going to talk about

- see Navigation bar

Key messages:

- LTN is a **framework** for combining **prior knowledge** with **data** in the training of **conventional** neural networks
- LTN is **not** a special **type** of neural network (NN)

Introduction

What we're going to talk about

- see Navigation bar

Key messages:

- LTN is a **framework** for combining **prior knowledge** with **data** in the training of **conventional** neural networks
- LTN is **not** a special **type** of neural network (NN)
- **knowledge** is expressed as **logical axioms** over **data** using a **fuzzy, first-order logic** language

Introduction

What we're going to talk about

- see Navigation bar

Key messages:

- LTN is a **framework** for combining **prior knowledge** with **data** in the training of **conventional** neural networks
- LTN is **not** a special **type** of neural network (NN)
- **knowledge** is expressed as **logical axioms** over **data** using a **fuzzy, first-order logic** language
- neural networks are **predicates** in the knowledge axioms

Introduction

What we're going to talk about

- see Navigation bar

Key messages:

- LTN is a **framework** for combining **prior knowledge** with **data** in the training of **conventional** neural networks
- LTN is **not** a special **type** of neural network (NN)
- **knowledge** is expressed as **logical axioms** over **data** using a **fuzzy, first-order logic** language
- neural networks are **predicates** in the knowledge axioms
- conventional **loss minimisation** drives the **satisfiability** of the **knowledge axioms** which, in turn, drives the **training** of the **neural network**

Introduction

What we're going to talk about

- see Navigation bar

Key messages:

- LTN is a **framework** for combining **prior knowledge** with **data** in the training of **conventional** neural networks
- LTN is **not** a special **type** of neural network (NN)
- **knowledge** is expressed as **logical axioms** over **data** using a **fuzzy, first-order logic** language
- neural networks are **predicates** in the knowledge axioms
- conventional **loss minimisation** drives the **satisfiability** of the **knowledge axioms** which, in turn, drives the **training** of the **neural network**
- the **optimal function** the NN learns is the one that **best satisfies** the **knowledge axioms** (given the **data**)

Context of LTN re Neural-Symbolic Integration

Neural-symbolic integration (NSI)

- combining connectionist AI with symbolist AI
- several classes of approach have emerged

Context of LTN re Neural-Symbolic Integration

Neural-symbolic integration (NSI)

- combining connectionist AI with symbolist AI
- several classes of approach have emerged

One class **integrates learning and reasoning**

- blending neural network **deep learning** with logical **reasoning** in a fully-differentiable, end-to-end architecture

Context of LTN re Neural-Symbolic Integration

Neural-symbolic integration (NSI)

- combining connectionist AI with symbolist AI
- several classes of approach have emerged

One class **integrates learning and reasoning**

- blending neural network **deep learning** with logical **reasoning** in a fully-differentiable, end-to-end architecture

One **subclass** of this class

- measures the **satisfiability** of **logical knowledge** and weaves this into the **loss function**
- **LTN** is **one member** of this subclass

Versions of LTN

Inspiration

- "Reasoning with Neural Tensor Networks for Knowledge Base Completion"; Socher, Chen, Manning & Ng, 2013.

Versions of LTN

Inspiration

- "Reasoning with Neural Tensor Networks for Knowledge Base Completion"; Socher, Chen, Manning & Ng, 2013.

Original version (2016)

- "Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge"; Serafini & Garcez, 2016
- TensorFlow 1 (explicit, static computational graphs)

Versions of LTN

Inspiration

- "Reasoning with Neural Tensor Networks for Knowledge Base Completion"; Socher, Chen, Manning & Ng, 2013.

Original version (2016)

- "Logic Tensor Networks: Deep Learning and Logical Reasoning from Data and Knowledge"; Serafini & Garcez, 2016
- TensorFlow 1 (explicit, static computational graphs)

New version (early 2021)

- "Logic Tensor Networks"; Badreddine et al., 2021
- TensorFlow 2 (implicit, dynamic computational graphs)
- extended features; gradient stability enhancements
- https://github.com/logictensornetworks/logictensornetworks
  - LTN API code, tutorials, examples, link to 2021 paper
  - **this presentation borrows from these assets!**

Classical vs Fuzzy Logics

Classical logics

- assume **precision** (e.g. John is a person)
- **binary** truth values: $\{0, 1\}$ (False or True)
- main types: **propositional** logic; **first-order** logic

Classical vs Fuzzy Logics

Classical logics

- assume **precision** (e.g. John is a person)
- **binary** truth values: $\{0, 1\}$ (False or True)
- main types: **propositional** logic; **first-order** logic

Fuzzy logics

- arose to handle **vagueness** (e.g. John is tall)
- infinite **degrees of truth**: $\in [0, 1]$
- main types: **fuzzy propositional** logic; **fuzzy first-order** logic
- major fuzzy logic systems: Łukasiewicz, Gödel, product (Goguen)

First-Order Logic Example

A typical first-order logic axiom (formula)

$$\forall x \left(\exists y \left(R(x,y) \wedge A(y) \right) \right)$$

Components

- variables: x, y
- predicates (n-ary relations): R, A
- logical operators/connectives: $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$
- quantifiers: $\forall$, $\exists$

First-Order Logic Example

A typical first-order logic axiom (formula)

$$\forall x \left(\exists y \left(R(x, y) \land A(y) \right) \right)$$

Components

- variables: x, y
- predicates (n-ary relations): R, A
- logical operators/connectives: $\land$, $\lor$, $\neg$, $\rightarrow$, $\leftrightarrow$
- quantifiers: $\forall$, $\exists$

One potential interpretation

Let x and y be **people**, $R \equiv$ **friend**, $A \equiv$ **Italian**:

First-Order Logic Example

A typical first-order logic axiom (formula)

$$\forall x \left(\exists y \left(R(x,y) \wedge A(y) \right) \right)$$

Components

- variables: x, y
- predicates (n-ary relations): R, A
- logical operators/connectives: $\wedge$, $\vee$, $\neg$, $\rightarrow$, $\leftrightarrow$
- quantifiers: $\forall$, $\exists$

One potential interpretation

Let x and y be **people**, $R \equiv$ **friend**, $A \equiv$ **Italian**:

axiom translation: "everyone has a friend that's Italian"

LTN Real Logic - The Basis of LTN

What is LTN Real Logic?

- a **fuzzy, first-order logic language** for NSI

LTN Real Logic - The Basis of LTN

What is LTN Real Logic?

- a **fuzzy, first-order logic language** for NSI

Components of the language

- **individuals** (constants) are real-valued tensors

LTN Real Logic - The Basis of LTN

What is LTN Real Logic?

- a **fuzzy, first-order logic language** for NSI

Components of the language

- **individuals** (constants) are real-valued tensors
- **variables** are **collections** of individuals

LTN Real Logic - The Basis of LTN

What is LTN Real Logic?

- a **fuzzy, first-order logic language** for NSI

Components of the language

- **individuals** (constants) are real-valued tensors
- **variables** are **collections** of individuals
- **predicates** are:
  - **neural networks** that output **truth degrees** $\in [0, 1]$
  - **functions** that output **truth degrees** $\in [0, 1]$

LTN Real Logic - The Basis of LTN

What is LTN Real Logic?

- a **fuzzy, first-order logic language** for NSI

Components of the language

- **individuals** (constants) are real-valued tensors
- **variables** are **collections** of individuals
- **predicates** are:
  - **neural networks** that output **truth degrees** $\in [0, 1]$
  - **functions** that output **truth degrees** $\in [0, 1]$
- **operators** have (differentiable) **fuzzy logic** semantics

LTN Real Logic - The Basis of LTN

What is LTN Real Logic?

- a **fuzzy, first-order logic language** for NSI

Components of the language

- **individuals** (constants) are real-valued tensors
- **variables** are **collections** of individuals
- **predicates** are:
  - **neural networks** that output **truth degrees** $\in [0, 1]$
  - **functions** that output **truth degrees** $\in [0, 1]$
- **operators** have (differentiable) **fuzzy logic** semantics
- **quantifiers** have (differentiable) **fuzzy logic** semantics

Real Logic “stable product configuration”

Various fuzzy **semantics** exist for operators/quantifiers

- Łukasiewicz, Gödel, product (Goguen), others ...
- LTN API offers a **wide selection** of these semantics

Real Logic “stable product configuration”

Various fuzzy **semantics** exist for operators/quantifiers

- Łukasiewicz, Gödel, product (Goguen), others ...
- LTN API offers a **wide selection** of these semantics

But: LTN recommends a specific **subset** of semantics

- called the “stable product configuration”
- it’s the **safe choice semantics** for each operator/quantifier

Real Logic “stable product configuration”

Various fuzzy **semantics** exist for operators/quantifiers

- Łukasiewicz, Gödel, product (Goguen), others ...
- LTN API offers a **wide selection** of these semantics

But: LTN recommends a specific **subset** of semantics

- called the “stable product configuration”
- it’s the **safe choice semantics** for each operator/quantifier

Why?

Real Logic "stable product configuration"

Various fuzzy **semantics** exist for operators/quantifiers

- Łukasiewicz, Gödel, product (Goguen), others ...
- LTN API offers a **wide selection** of these semantics

But: LTN recommends a specific **subset** of semantics

- called the "stable product configuration"
- it's the **safe choice semantics** for each operator/quantifier

Why?

- alternative semantics induce **vanishing** or **exploding** or **single-passing gradients**

Real Logic "stable product configuration"

Various fuzzy **semantics** exist for operators/quantifiers

- Łukasiewicz, Gödel, product (Goguen), others ...
- LTN API offers a **wide selection** of these semantics

But: LTN recommends a specific **subset** of semantics

- called the "stable product configuration"
- it's the **safe choice semantics** for each operator/quantifier

Why?

- alternative semantics induce **vanishing** or **exploding** or **single-passing gradients**
- source: van Krieken et al. (2020). "Analyzing Differentiable Fuzzy Logic Operators". *arXiv:2002.06100*

Real Logic “stable product configuration”

Stable Logical Operator Semantics

Operator	Semantics *	Fuzzy Logic Source System
And ($\wedge$)	$u \wedge v = uv$	product
Or ($\vee$)	$u \vee v = u + v - uv$	product
Not ($\neg$)	$\neg u = 1 - u$	Łukasiewicz
Implies ($\rightarrow$)	$u \rightarrow v = 1 - u + uv$	Reichenbach **
Equiv ($\leftrightarrow$)	$u \leftrightarrow v = (u \rightarrow v) \wedge (v \rightarrow u)$	(mixed)

* u and v represent degrees of truth $\in$ $[0,1]$

** $u \rightarrow v \equiv (\neg u) \vee v = (1-u) \vee v = (1-u) + v - (1-u)v = 1-u+uv$

Real Logic “stable product configuration”

Existential quantifier $\exists$ - ‘there exists’ (at least one)

Stable Semantics

The “generalised mean” (p-mean), here denoted pM:

$$pM(u_1, \ldots, u_n) = \left(\frac{1}{n} \sum_{i=1}^{n} u_i^p \right)^{\frac{1}{p}} \quad p \geq 1$$

Real Logic “stable product configuration”

Existential quantifier $\exists$ - ‘there exists’ (at least one)

Stable Semantics

The “generalised mean” (p-mean), here denoted pM:

$$pM(u_1, \ldots, u_n) = \left(\frac{1}{n} \sum_{i=1}^{n} u_i^p \right)^{\frac{1}{p}} \quad p \geq 1$$

where:

- $u_i \in [0, 1] \quad i = 1 \ldots n$ (degrees of truth)
- $0 \leq pM \leq 1$
- if $p = 1$, $pM =$ arithmetic mean
- as $p \to \infty$, $pM \to max(u_1, \ldots, u_n)$

Real Logic "stable product configuration"

Existential quantifier $\exists$ - 'there exists' (at least one)

Stable Semantics

The "generalised mean" (p-mean), here denoted pM:

$$pM(u_1, \ldots, u_n) = \left(\frac{1}{n} \sum_{i=1}^{n} u_i^p \right)^{\frac{1}{p}} \quad p \geq 1$$

where:

- $u_i \in [0,1] \quad i = 1 \ldots n$ (degrees of truth)
- $0 \leq pM \leq 1$
- if $p = 1$, $pM =$ arithmetic mean
- as $p \to \infty$, $pM \to max(u_1, \ldots, u_n)$

So, as $p \to \infty$, pM **approximates** a **maximum** function

Real Logic “stable product configuration”

Universal quantifier $\forall$ - ‘for all’

Stable Semantics

The “generalised mean of deviations from the truth” (p-mean error), here denoted pME:

$$pME(u_1, \ldots, u_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - u_i)^p \right)^{\frac{1}{p}} \qquad p \geq 1$$

Real Logic "stable product configuration"

Universal quantifier $\forall$ - 'for all'

Stable Semantics

The "generalised mean of deviations from the truth" (p-mean error), here denoted pME:

$$pME(u_1, \ldots, u_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - u_i)^p \right)^{\frac{1}{p}} \qquad p \geq 1$$

where:

- $u_i \in [0, 1] \quad i = 1 \ldots n$ (degrees of truth)
- $0 \leq pME \leq 1$
- if $p = 1$, $pME =$ arithmetic mean
- as $p \to \infty$, $pME \to min(u_1, \ldots, u_n)$

Real Logic "stable product configuration"

Universal quantifier $\forall$ - 'for all'

Stable Semantics

The "generalised mean of deviations from the truth" (p-mean error), here denoted pME:

$$pME(u_1, \ldots, u_n) = 1 - \left(\frac{1}{n} \sum_{i=1}^{n} (1 - u_i)^p \right)^{\frac{1}{p}} \qquad p \geq 1$$

where:

- $u_i \in [0, 1] \quad i = 1 \ldots n$ (degrees of truth)
- $0 \leq pME \leq 1$
- if $p = 1$, $pME =$ arithmetic mean
- as $p \to \infty$, $pME \to min(u_1, \ldots, u_n)$

So, as $p \to \infty$, pME **approximates** a **minimum** function

Real Logic "stable product configuration"

The LTN "stable product configuration" in code

```python
And = ltn.Wrapper_Connective(ltn.fuzzy_ops.And_Prod())

Or = ltn.Wrapper_Connective(ltn.fuzzy_ops.Or_ProbSum())

Not = ltn.Wrapper_Connective(ltn.fuzzy_ops.Not_Std())

Implies = ltn.Wrapper_Connective(ltn.fuzzy_ops.Implies_Reichenbach())

Equiv = ltn.Wrapper_Connective(ltn.fuzzy_ops.Equiv( \
                                ltn.fuzzy_ops.And_Prod(), \
                                ltn.fuzzy_ops.Implies_Reichenbach()))

Exists = ltn.Wrapper_Quantifier(ltn.fuzzy_ops.Aggreg_pMean(p=6), \
                                           semantics="exists")

Forall = ltn.Wrapper_Quantifier(ltn.fuzzy_ops.Aggreg_pMeanError(p=4), \
                                           semantics="forall")
```

quantifier p-values are **hyper-parameters** to be tuned

LTN Real Logic Axioms

What is an LTN Real Logic axiom?

LTN Real Logic Axioms

What is an LTN Real Logic axiom?

- an **assertion** of **knowledge** over **data**

LTN Real Logic Axioms

What is an LTN Real Logic axiom?

- an **assertion** of **knowledge** over **data**
- a **logical constraint** to be **satisfied**

LTN Real Logic Axioms

What is an LTN Real Logic axiom?

- an **assertion** of **knowledge** over **data**
- a **logical constraint** to be **satisfied**
- **executable** Python **code**
- a nested function call that **returns** a **satisfaction level** (truth degree) $\in [0, 1]$

LTN Real Logic Axioms

What is an LTN Real Logic axiom?

- an **assertion** of **knowledge** over **data**
- a **logical constraint** to be **satisfied**
- **executable** Python **code**
- a nested function call that **returns** a **satisfaction level** (truth degree) $\in [0, 1]$

Recall our example first-order logic axiom

$$\forall x \left(\exists y \left(R(x, y) \wedge A(y) \right) \right)$$

LTN Real Logic Axioms

What is an LTN Real Logic axiom?

- an **assertion** of **knowledge** over **data**
- a **logical constraint** to be **satisfied**
- **executable** Python **code**
- a nested function call that **returns** a **satisfaction level** (truth degree) $\in [0, 1]$

Recall our example first-order logic axiom

$$\forall x \left(\exists y \left(R(x, y) \wedge A(y) \right) \right)$$

The equivalent LTN Real Logic axiom might look like

sat_level = Forall(x, Exists(y, And(Friend(x,y), Italian(y))))

LTN Real Logic Axioms

Our example axiom again

$$\forall x \left(\exists y \left(R(x,y) \wedge A(y) \right) \right)$$

$$\equiv$$

Forall(x, Exists(y, And(Friend(x,y), Italian(y))))

LTN Real Logic Axioms

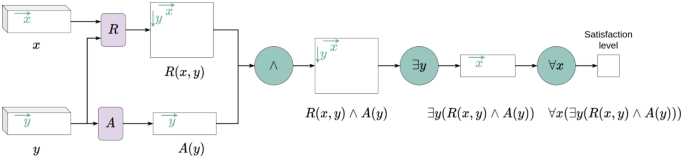
Our example axiom again

$$\forall x \left(\exists y \left(R(x,y) \wedge A(y) \right) \right)$$

$$\equiv$$

Forall(x, Exists(y, And(Friend(x,y), Italian(y))))

Computational graph for our example axiom



LTN Knowledge Base

What is an LTN Knowledge Base (KB)?

LTN Knowledge Base

What is an LTN Knowledge Base (KB)?

A **collection** of LTN Real Logic **axioms** over **data**

- implemented within a **Python function**

LTN Knowledge Base

What is an LTN Knowledge Base (KB)?

A **collection** of LTN Real Logic **axioms** over **data**

- implemented within a **Python function**
- that **returns** a single, overall KB **satisfaction level** (truth degree) $\in [0, 1]$

LTN Knowledge Base

What is an LTN Knowledge Base (KB)?

A **collection** of LTN Real Logic **axioms** over **data**

- implemented within a **Python function**
- that **returns** a single, overall KB **satisfaction level** (truth degree) $\in [0, 1]$

KB 'formula aggregator'

- **aggregates** the **satisfaction levels** of individual axioms
- computes the **overall** KB **satisfaction level**

LTN Knowledge Base

What is an LTN Knowledge Base (KB)?

A **collection** of LTN Real Logic **axioms** over **data**

- implemented within a **Python function**
- that **returns** a single, overall KB **satisfaction level** (truth degree) $\in [0, 1]$

KB 'formula aggregator'

- **aggregates** the **satisfaction levels** of individual axioms
- computes the **overall** KB **satisfaction level**
- has same **semantics** as the universal quantifier ($\forall$)
  - the p-**mean error** (pME), typically with $p = 2$
  - a (lenient) **approximator** of the **minimum** satisfaction level

LTN Knowledge Base – toy examples

Toy example KB with 1 axiom

```
def axioms(x,y):
    sat_level = Forall(x, Exists(y, And(Friend(x,y), Italian(y))))
    return sat_level
```

LTN Knowledge Base – toy examples

Toy example KB with 1 axiom

```
def axioms(x,y):
    sat_level = Forall(x, Exists(y, And(Friend(x,y), Italian(y))))
    return sat_level
```

Toy example KB with 2 axioms

```
formula_aggregator = ltn.Wrapper_Formula_Aggregator( \
                     ltn.fuzzy_ops.Aggreg_pMeanError(p=2))

def axioms(x,y):
    axioms = [
        Forall(x, Exists(y, And(Friend(x,y), Italian(y)))),
        Exists(x, Exists(y, And(Friend(x,y), Not(Tall(y)))))
    ]
    sat_level = formula_aggregator(axioms)
    return sat_level
```

The (standard) LTN Loss Function

We **call** the KB and it **returns**

$$sat\_level \in [0, 1]$$

The (standard) LTN Loss Function

We **call** the KB and it **returns**

$$sat\_level \in [0, 1]$$

The (standard) LTN **loss function** is simply

$$loss = 1 - sat\_level$$

As $sat\_level \to 1$, $loss \to 0$.

A representative LTN training loop

```
...
ds_train = # conventional training set
...
model = MyNeuralNetworkModelClass()
Pred = ltn.Predicate(model) # NN (predicate) used in KB axioms
...
trainable_vars = Pred.trainable_variables
optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)

for epoch in range(n_epochs):
    for features, labels in ds_train:
        with tf.GradientTape() as tape:
            sat_level = axioms(features, labels)
            loss = 1 - sat_level
        grads = tape.gradient(loss, trainable_vars)
        optimizer.apply_gradients(zip(grads, trainable_vars))
        ...
...
```

How the LTN framework trains neural networks:

- gradient descent drives **loss minimisation**
- loss minimised by **maximising KB satisfiability**
- KB satisfiability maximised by the **NN learning the function** that **best satisfies** all the **axioms**

Github Examples Illustrating Range of LTN

Note: The Github examples illustrate the mechanisms and flexibility of LTN, not how it's intended to be used in practice.

Github Examples Illustrating Range of LTN

Note: The Github examples illustrate the mechanisms and flexibility of LTN, not how it's intended to be used in practice.

Classification & Regression (supervised)

Github Examples Illustrating Range of LTN

Note: The Github examples illustrate the mechanisms and flexibility of LTN, not how it's intended to be used in practice.

Classification & Regression (supervised)

Clustering (unsupervised)

Github Examples Illustrating Range of LTN

Note: The Github examples illustrate the mechanisms and flexibility of LTN, not how it's intended to be used in practice.

Classification & Regression (supervised)

Clustering (unsupervised)

Addition with MNIST digits (semi-supervised)

Github Examples Illustrating Range of LTN

Note: The Github examples illustrate the mechanisms and flexibility of LTN, not how it's intended to be used in practice.

Classification & Regression (supervised)

Clustering (unsupervised)

Addition with MNIST digits (semi-supervised)

Learning embeddings for individuals

Github Examples Illustrating Range of LTN

Note: The Github examples illustrate the mechanisms and flexibility of LTN, not how it's intended to be used in practice.

Classification & Regression (supervised)

Clustering (unsupervised)

Addition with MNIST digits (semi-supervised)

Learning embeddings for individuals

Deductive reasoning

- determining whether a proposition is entailed by a KB?
- e.g. $(A \vee B) \models A$?
- reasoning by **refutation**: learning a **counter-example** where $(A \vee B)$ **is** satisfied but A is **not** satisfied

An LTN KB for Binary Classification

An LTN KB for Binary Classification

Predicate (NN): **Cat** – cat classifier

An LTN KB for Binary Classification

Predicate (NN): **Cat** – cat classifier

```
def axioms(features, labels):
    pos = ltn.Variable("pos",features[labels])
    neg = ltn.Variable("neg",features[tf.logical_not(labels)])
    axioms = [
        Forall(pos, Cat(pos)),
        Forall(neg, Not(Cat(neg)))
    ]
    sat_level = formula_aggregator(axioms)
    return sat_level
```

An LTN KB for Binary Classification

Predicate (NN): **Cat** – cat classifier

```python
def axioms(features, labels):
    pos = ltn.Variable("pos",features[labels])
    neg = ltn.Variable("neg",features[tf.logical_not(labels)])
    axioms = [
        Forall(pos, Cat(pos)),
        Forall(neg, Not(Cat(neg)))
    ]
    sat_level = formula_aggregator(axioms)
    return sat_level
```

Observe:

- here the **knowledge** is limited to **what's in the data**

An LTN KB for Binary Classification

Predicate (NN): **Cat** – cat classifier

```
def axioms(features, labels):
    pos = ltn.Variable("pos",features[labels])
    neg = ltn.Variable("neg",features[tf.logical_not(labels)])
    axioms = [
        Forall(pos, Cat(pos)),
        Forall(neg, Not(Cat(neg)))
    ]
    sat_level = formula_aggregator(axioms)
    return sat_level
```

Observe:

- here the **knowledge** is limited to **what's in the data**
- label supervision **replaced** by **constraints** to be **satisfied**

An LTN KB for Clustering

An LTN KB for Clustering

Predicate (NN): **C** – cluster membership classifier

An LTN KB for Clustering

Predicate (NN): **C** – cluster membership classifier

```
...
cluster = ltn.Variable("cluster",clst_ids) # 0...K-1
...

# Axiom interpretations:
# 1: every example should be assigned to a cluster
# 2: every cluster should be non-empty
# 3: if the points are near, they should belong to the same cluster
# 4: if the points are far, they should belong to different clusters
# 5: clusters should be disjoint; handled IMPLICITLY by the softmax
#    activation function in the output layer of predicate C, which
#    returns probabilities that are mutually exclusive

def axioms(x,y):
    axioms = [
        Forall(x, Exists(cluster, C([x,cluster]))),
        Forall(cluster, Exists(x, C([x,cluster]))),
        Forall([cluster,x,y], Equiv(C([x,cluster]),C([y,cluster])),
            mask = is_less_than([eucl_dist([x,y]),close_thr])),
        Forall([cluster,x,y], Not(And(C([x,cluster]),C([y,cluster]))),
            mask = is_greater_than([eucl_dist([x,y]),distant_thr]))
    ]
    sat_level = formula_aggregator(axioms)
    return sat_level
```

An LTN KB for Clustering

Predicate (NN): **C** – cluster membership classifier

```
...
cluster = ltn.Variable("cluster",clst_ids) # 0...K-1
...

# Axiom interpretations:
# 1: every example should be assigned to a cluster
# 2: every cluster should be non-empty
# 3: if the points are near, they should belong to the same cluster
# 4: if the points are far, they should belong to different clusters
# 5: clusters should be disjoint; handled IMPLICITLY by the softmax
#    activation function in the output layer of predicate C, which
#    returns probabilities that are mutually exclusive

def axioms(x,y):
    axioms = [
        Forall(x, Exists(cluster, C([x,cluster]))),
        Forall(cluster, Exists(x, C([x,cluster]))),
        Forall([cluster,x,y], Equiv(C([x,cluster]),C([y,cluster])),
            mask = is_less_than([eucl_dist([x,y]),close_thr])),
        Forall([cluster,x,y], Not(And(C([x,cluster]),C([y,cluster]))),
            mask = is_greater_than([eucl_dist([x,y]),distant_thr]))
    ]
    sat_level = formula_aggregator(axioms)
    return sat_level
```

Observe:

- here the **knowledge** is all **external** (supplementary) to the **data**

Research Work using LTN

We briefly review 4 research papers using LTN

LTN for Zero-Shot Learning with Images

Paper

- Donadello & Serafini (2019). **Compensating Supervision Incompleteness with Prior Knowledge in Semantic Image Interpretation.**

LTN for Zero-Shot Learning with Images

Paper

- Donadello & Serafini (2019). **Compensating Supervision Incompleteness with Prior Knowledge in Semantic Image Interpretation.**

Description

- task: predict `(subject,relation,object)` triples **(visual relationships)** in images; e.g. `(person,ride,horse)`
- approach:
  - object detector detects and classifies subject/object bboxes
  - LTN-trained NNs predict the relations
- strategy: inject **relational knowledge** in the form of **negative domain and range** constraints
  - e.g. $\forall xy \; Ride(x,y) \rightarrow Not(Table(x))$
  - e.g. $\forall xy \; Ride(x,y) \rightarrow Not(Chair(y))$

  nb: the unary predicates (e.g. Table(x)) for the object classes are rule-based functions returning 1 or 0;

  the binary predicates (e.g. Ride(x,y)) for the relations are trainable NNs

LTN for Zero-Shot Learning with Images

Results: NNs trained over LTN prior knowledge ...

- deliver **better** relation **predictive performance**
- including better **zero-shot** relation predictions

LTN for Zero-Shot Learning with Images

Results: NNs trained over LTN prior knowledge ...

- deliver **better** relation **predictive performance**
- including better **zero-shot** relation predictions

Issue: LTN scalability limitation

- dataset has 70 relations and 100 object classes
- $\approx$ 25 relations permit domain/range restrictions
  $\Rightarrow$ $\approx$ **25 NNs** to be trained
- 100 domain + 100 range constraints $\approx$ 200 axioms/relation
  $\Rightarrow$ 25 relations $\times$ 200 axioms $\approx$ **5000 axioms** in KB
- **repeated calls** to the same NNs per mini-batch
  $\Rightarrow$ $\approx$ **200 calls** per binary predicate NN
- (nb: Donadello uses only a **"tractable sample"**)

LTN with Deep Reinforcement Learning

Paper

- Badreddine & Spranger (2019). **Injecting Prior Knowledge for Transfer Learning into Reinforcement Learning Algorithms using Logic Tensor Networks.**

Discussion

- task: **grid game**; **agent** needs to learn to **collect** certain objects and **avoid** others to **maximise** cumulative **reward**
- LTN used to **preprocess** input data to **derive explicit knowledge** of which grid cells to target/avoid
- they show **agent** learns to **exploit** the **prior knowledge**
- role of LTN here is minor (incidental)

LTN with Deep Reinforcement Learning

Knowledge derivation; CNN architecture for RL system

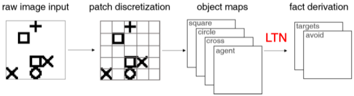


Figure 5: Priors on the grid game

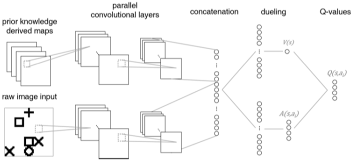


Figure 6: A Double Dueling architecture architecture with conjoint image and prior knowledge inputs.

LTN axioms for one preprocessing scenario

$$\forall x \in \mathcal{O} : \mathrm{circle}(x) \leftrightarrow \mathrm{goto}(x)$$
$$\forall x \in \mathcal{O} : \mathrm{cross}(x) \leftrightarrow \mathrm{avoid}(x)$$

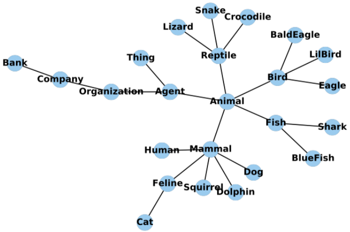
LTN Reasoning Capabilities

Paper

- Bianchi & Hitzler (2019). **On the Capabilities of Logic Tensor Networks for Deductive Reasoning.**

One experiment: taxonomy reasoning

- train set: 24-class taxonomy with 23 subClassOf relations



- test set: the full transitive closure + all false relations
- objective: train NN to predict $sub(x, y)$ given arbitrary x, y

LTN Reasoning Capabilities

LTN KB axioms

- $\forall a, b, c \quad sub(a, b) \wedge sub(b, c) \rightarrow sub(a, c)$ (transitive)
- $\forall a \quad \neg sub(a, a)$ (not reflexive)
- $\forall a, b \quad sub(a, b) \rightarrow \neg sub(b, a)$ (not symmetric)

LTN Reasoning Capabilities

LTN KB axioms

- $\forall a, b, c \quad sub(a,b) \wedge sub(b,c) \rightarrow sub(a,c)$ (transitive)
- $\forall a \quad \neg sub(a,a)$ (not reflexive)
- $\forall a, b \quad sub(a,b) \rightarrow \neg sub(b,a)$ (not symmetric)

Results: best performing model (highest satisfiability)

Pred \ Actual	**T**	**F**	Precision
T	55 (tp)	36 (fp)	0.60
F	26 (fn)	459 (tn)	
Recall	0.68		$576 = 24 \times 24$

- accuracy of 0.89
  - but a naive classifier predicting only 0s delivers 0.84

LTN Reasoning Capabilities

LTN general observations:

- higher KB **satisfiability** $\Rightarrow$ better predictive **performance**
- more **axioms** in KB $\Rightarrow$ better predictive **performance**

LTN Reasoning Capabilities

LTN general observations:

- higher KB **satisfiability** $\Rightarrow$ better predictive **performance**
- more **axioms** in KB $\Rightarrow$ better predictive **performance**

LTN weaknesses:

- **prone** to predicting **false positives**
- **struggle** with **multi-hop** inferences
  - e.g. $sub(cat, animal)$ is a bridge too far
  - (i.e. training NNs to generalise remains a challenge)

LTN Reasoning Capabilities

LTN general observations:

- higher KB **satisfiability** $\Rightarrow$ better predictive **performance**
- more **axioms** in KB $\Rightarrow$ better predictive **performance**

LTN weaknesses:

- **prone** to predicting **false positives**
- **struggle** with **multi-hop** inferences
  - e.g. $sub(cat, animal)$ is a bridge too far
  - (i.e. training NNs to generalise remains a challenge)

LTN strength: explainability / interpretability

- LTN can provide partial explanations
- existing (or new) axiom **satisfaction levels** can be **queried** after (or during) training to provide **insight** into 1) what a model is **learning**, 2) a model's predictive **performance**
  - **query** axiom $\equiv$ **execute** axiom

LTN for Fairness

Paper

- Wagner & Garcez (2021). **Neural-Symbolic Integration for Fairness in AI.**

One experiment: fairness in credit risk assessment

- objective: train binary classifier to avoid gender bias
  - predicate D denotes **Default** (on granted credit)
- strategy: add gender **fairness axioms** to LTN KB

$$
\begin{array}{rcr}
\forall x \in \mathcal{T}_D & : & D(x) \\
\forall x \in \mathcal{T}_N & : & \neg D(x) \\
\hline
\forall x \in \mathcal{R}_{F1}, y \in \mathcal{R}_{M1} & : & D(x) \leftrightarrow D(y) \\
\forall x \in \mathcal{R}_{F2}, y \in \mathcal{R}_{M2} & : & D(x) \leftrightarrow D(y) \\
\forall x \in \mathcal{R}_{F3}, y \in \mathcal{R}_{M3} & : & D(x) \leftrightarrow D(y) \\
\forall x \in \mathcal{R}_{F4}, y \in \mathcal{R}_{M4} & : & D(x) \leftrightarrow D(y) \\
\forall x \in \mathcal{R}_{F5}, y \in \mathcal{R}_{M5} & : & D(x) \leftrightarrow D(y)
\end{array}
$$

people split into 5 credit quality groups

fairness constraints guide the NN to not learn gender bias

female male

- results: the **fairness** axioms **suppress** the gender bias

LTN Academic Use - Reflections

Uptake of LTN has been limited, so far

- 2 of the 4 papers are by students of the originators of LTN
- TensorFlow's transition from v1 to v2 likely a factor
  - TF 2 released Sept 2019
  - LTN for TF 2 not available until Jan 2021

Summary

Summary

The LTN framework approach to NSI allows one to ...

- blend **knowledge** with **data** using **logical constraints** expressed in a **fuzzy, first-order logic** language
- weave a measure of overall **constraint satisfaction** into the **loss function**

Summary

The LTN framework approach to NSI allows one to ...

- blend **knowledge** with **data** using **logical constraints** expressed in a **fuzzy, first-order logic** language
- weave a measure of overall **constraint satisfaction** into the **loss function**

Types of knowledge expressible

- **data** knowledge: knowledge **internal** to the **data**
- **real-world** knowledge: knowledge **external** to the **data**

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs
- the NN still learns an **optimal function**

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs
- the NN still learns an **optimal function**
- **optimality** governed by **knowledge axioms** over **data**

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs
- the NN still learns an **optimal function**
- **optimality** governed by **knowledge axioms** over **data**
- the **optimal function** learned by the NN is the one that **best satisfies** the **knowledge axioms** (given the **data**)

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs
- the NN still learns an **optimal function**
- **optimality** governed by **knowledge axioms** over **data**
- the **optimal function** learned by the NN is the one that **best satisfies** the **knowledge axioms** (given the **data**)

Miscellaneous observations

- LTN is **flexible** enough to handle **many** ML tasks

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs
- the NN still learns an **optimal function**
- **optimality** governed by **knowledge axioms** over **data**
- the **optimal function** learned by the NN is the one that **best satisfies** the **knowledge axioms** (given the **data**)

Miscellaneous observations

- LTN is **flexible** enough to handle **many** ML tasks
- LTN academic **up-take** has been **limited**, so far

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs
- the NN still learns an **optimal function**
- **optimality** governed by **knowledge axioms** over **data**
- the **optimal function** learned by the NN is the one that **best satisfies** the **knowledge axioms** (given the **data**)

Miscellaneous observations

- LTN is **flexible** enough to handle **many** ML tasks
- LTN academic **up-take** has been **limited**, so far
- LTN **reasoning capabilities** are **modest**

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs
- the NN still learns an **optimal function**
- **optimality** governed by **knowledge axioms** over **data**
- the **optimal function** learned by the NN is the one that **best satisfies** the **knowledge axioms** (given the **data**)

Miscellaneous observations

- LTN is **flexible** enough to handle **many** ML tasks
- LTN academic **up-take** has been **limited**, so far
- LTN **reasoning capabilities** are **modest**
- LTN has **scalability** limitations

Summary

LTN is **not** a special **type** of neural network

- it's a special **way** of **training** conventional NNs
- the NN still learns an **optimal function**
- **optimality** governed by **knowledge axioms** over **data**
- the **optimal function** learned by the NN is the one that **best satisfies** the **knowledge axioms** (given the **data**)

Miscellaneous observations

- LTN is **flexible** enough to handle **many** ML tasks
- LTN academic **up-take** has been **limited**, so far
- LTN **reasoning capabilities** are **modest**
- LTN has **scalability** limitations
- LTN offers **explainability** via **querying** axiom satisfiability

All Done

Thank You!