Logic as Loss: Overview

THE UNIVERSITY OF UTAH

Logic as loss: The setting

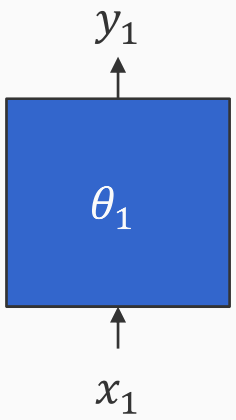
- Suppose we would like to train one or more neural networks on some data
  - Each of them can take different kinds of inputs to produce their own discrete outputs

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(\, y_1 \mid x_1, \theta_1 \,)$$



This network takes inputs X_1 and produces a distribution over labels Y_1

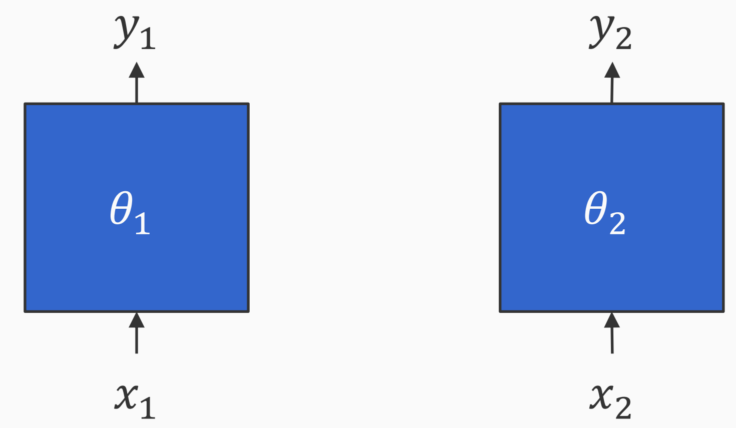
Its architecture and parameters are defined by θ_1

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(\, y_1 \mid x_1, \theta_1 \,), P(\, y_2 \mid x_2, \theta_2 \,)$$



This network takes inputs X_2 and produces a distribution over labels Y_2

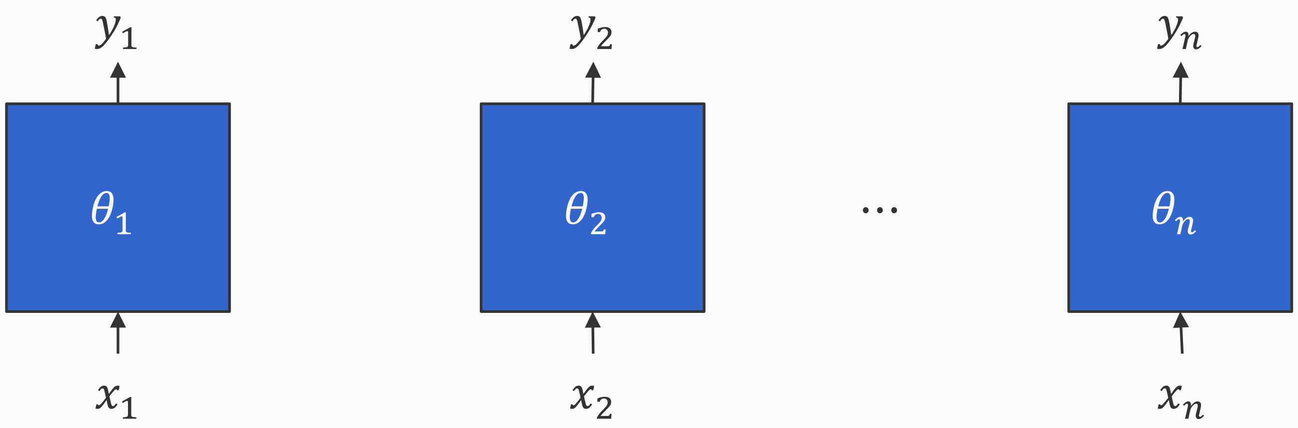
Its architecture and parameters are defined by θ_2

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(y_1 \mid x_1, \theta_1), P(y_2 \mid x_2, \theta_2), \cdots, P(y_n \mid x_n, \theta_n)$$



This network takes inputs X_n and produces a distribution over labels Y_n

Its architecture and parameters are defined by θ_n

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(y_1 \mid x_1, \theta_1), P(y_2 \mid x_2, \theta_2), \cdots, P(y_n \mid x_n, \theta_n)$$

Important:

These n different networks and their inputs/outputs need not be different

- Maybe they are the same model called on different inputs
- Maybe they are different models given to the same network
- Or other combinations

What we have are n different conditional distributions over outputs given inputs

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(\, y_1 \mid x_1, \theta_1 \,), P(\, y_2 \mid x_2, \theta_2 \,), \cdots, P(\, y_n \mid x_n, \theta_n \,)$$

How do we train these networks on their respective datasets?

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(y_1 \mid x_1, \theta_1), P(y_2 \mid x_2, \theta_2), \cdots, P(y_n \mid x_n, \theta_n)$$

How do we train these networks on their respective datasets?

One answer: Minimize cross-entropy loss over the data

The loss penalizes models that assign low probabilities to observed data

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(y_1 \mid x_1, \theta_1), P(y_2 \mid x_2, \theta_2), \cdots, P(y_n \mid x_n, \theta_n)$$

Recall that we can write classification tasks as predicates

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(y_1 \mid x_1, \theta_1), P(y_2 \mid x_2, \theta_2), \cdots, P(y_n \mid x_n, \theta_n)$$

Assign label y_1 to x_1

Recall that we can write classification tasks as predicates

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(\, y_1 \mid x_1, \theta_1 \,), P(\, y_2 \mid x_2, \theta_2 \,), \cdots, P(\, y_n \mid x_n, \theta_n \,)$$

Assign label y_1 to x_1

$\downarrow$

$\text{Label}(X_1, Y_1)$

Recall that we can write classification tasks as predicates

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(\,y_1 \mid x_1, \theta_1\,), P(\,y_2 \mid x_2, \theta_2\,), \cdots, P(\,y_n \mid x_n, \theta_n\,)$$

Assign label y_1 to x_1

Assign label y_2 to x_2

$\downarrow$

$\text{Label}(X_1, Y_1)$

Recall that we can write classification tasks as predicates

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(\,y_1 \mid x_1, \theta_1\,), P(\,y_2 \mid x_2, \theta_2\,), \cdots, P(\,y_n \mid x_n, \theta_n\,)$$

Assign label y_1 to x_1 $\rightarrow$ $\mathrm{Label}(X_1, Y_1)$

Assign label y_2 to x_2 $\rightarrow$ $\mathrm{Label}(X_2, Y_2)$

Recall that we can write classification tasks as predicates

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(y_1 \mid x_1, \theta_1), P(y_2 \mid x_2, \theta_2), \cdots, P(y_n \mid x_n, \theta_n)$$

Assign label y_1 to x_1 → $\mathrm{Label}(X_1, Y_1)$

Assign label y_2 to x_2 → $\mathrm{Label}(X_2, Y_2)$

Assign label y_n to x_n

Recall that we can write classification tasks as predicates

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

Each of them can take different kinds of inputs to produce their own discrete outputs

$$P(\, y_1 \mid x_1, \theta_1 \,), P(\, y_2 \mid x_2, \theta_2 \,), \cdots, P(\, y_n \mid x_n, \theta_n \,)$$

Assign label y_1 to x_1 $\downarrow$ $\mathrm{Label}(X_1, Y_1)$

Assign label y_2 to x_2 $\downarrow$ $\mathrm{Label}(X_2, Y_2)$

Assign label y_n to x_n $\downarrow$ $\mathrm{Label}(X_n, Y_n)$

Recall that we can write classification tasks as predicates

Logic as loss: The setting

Suppose we would like to train one or more neural networks on some data

- Each of them can take different kinds of inputs to produce their own discrete outputs

Corresponding to the networks, we have predicates that serve as a vocabulary for rules

- Suppose we want our models to satisfy some *invariant property* that is written in terms of these predicates *in addition to modeling the data*

How do we proceed?

The idea of the “logic as loss” framework

What we want of our models

Fit the available training data

The idea of the “logic as loss” framework

What we want of our models

Fit the available training data

↓

Minimize Data loss (e.g. cross entropy)

The idea of the “logic as loss” framework

What we want of our models

Fit the available training data

Minimize Data loss (e.g. cross entropy)

penalty for disagreeing with data

The idea of the “logic as loss” framework

What we want of our models

Fit the available training data

and

Satisfy invariant properties about predictions

Minimize

Data loss
(e.g. cross entropy)

penalty for
disagreeing with data

The idea of the "logic as loss" framework

What we want of our models

Fit the available training data → Data loss (e.g. cross entropy)

and

Satisfy invariant properties about predictions → Constraint loss

Minimize Data loss (e.g. cross entropy) Constraint loss

penalty for disagreeing with data

The idea of the “logic as loss” framework

What we want of our models

Fit the available training data

and

Satisfy invariant properties about predictions

Minimize

Data loss
(e.g. cross entropy)

Constraint loss

penalty for
disagreeing with data

penalty for
violating constraints

The idea of the "logic as loss" framework

What we want of our models

Fit the available training data

and

Satisfy invariant properties about predictions

Minimize (Data loss (e.g. cross entropy) + Constraint loss)

penalty for disagreeing with data

penalty for violating constraints

The idea of the “logic as loss” framework

What we want of our models

Fit the available training data

and

Satisfy invariant properties about predictions

Minimize Data loss (e.g. cross entropy) + Constraint loss

penalty for disagreeing with data

penalty for violating constraints

The intuition

We want to find models that maximally satisfy the constraints

If the constraint losses are well defined to capture the logic, models that violate constraints will have higher loss

So it is better to minimize the constraint loss

Labeled examples are also predicates about examples

Suppose we have a labeled example in a dataset (x_1, y_1)

We can write this as a predicate: $\text{Label}(x_1, y_1)$

This is the statement "*The label for example* x_1 *is* y_1".

Labeled examples are also predicates about examples

Suppose we have a labeled example in a dataset (x_1, y_1)

We can write this as a predicate: $\text{Label}(x_1, y_1)$

This is the statement "*The label for example x_1 is y_1*".

A dataset is nothing but a collection of examples of the form $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

This is equivalent to the conjunction

$$\bigwedge_{i=1}^{n} \text{Label}(x_i, y_i)$$

Labeled examples are also predicates about examples

Suppose we have a labeled example in a dataset (x_1, y_1)

We can write this as a predicate: $\text{Label}(x_1, y_1)$

This is the statement "*The label for example x_1 is y_1*".

A dataset is nothing but a collection of examples of the form $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$

This is equivalent to the conjunction

$$\bigwedge_{i=1}^{n} \text{Label}(x_i, y_i)$$

Can we fold this into the framework?

The idea of the “logic as loss” framework

What we want of our models

This is a specific set of constraints → Fit the available training data

and

Satisfy invariant properties about predictions

Minimize (Data loss (e.g. cross entropy) + Constraint loss)

penalty for disagreeing with data

penalty for violating constraints

The idea of the “logic as loss” framework

What we want of our models

This is a specific set of constraints → Fit the available training data

and

Satisfy invariant properties about predictions

Let us fold them into one term

Minimize { Data loss (e.g. cross entropy) + Constraint loss }

penalty for disagreeing with data

penalty for violating constraints

The idea of the “logic as loss” framework

What we want of our models

Satisfy a set of invariant properties
(perhaps including properties about specific examples)

Minimize Data loss (e.g. cross entropy) + Constraint loss

penalty for disagreeing with data

penalty for violating constraints

The idea of the "logic as loss" framework

What we want of our models

Satisfy a set of invariant properties
(perhaps including properties about specific examples)

Minimize Constraint loss

penalty for violating constraints

The idea of the "logic as loss" framework

What we want of our models

Satisfy a set of invariant properties
(perhaps including properties about specific examples)

$\downarrow$

Minimize Constraint loss

Let us formally state the setting

Suppose we have a sentence α in predicate logic, defined over some atoms $X = \{X_1, X_2, \cdots, X_n\}$

The idea of the "logic as loss" framework

What we want of our models

Satisfy a set of invariant properties (perhaps including properties about specific examples)

Minimize Constraint loss

Let us formally state the setting

Suppose we have a sentence α in predicate logic, defined over some atoms $X = \{X_1, X_2, \cdots, X_n\}$

Suppose each atom X_i is associated with a probability p_i, possibly from a neural model

Let the vector $\mathbf{p}$ denote the collection of probabilities $[p_1, p_2, \cdots, p_n]$ over the atoms

The idea of the "logic as loss" framework

What we want of our models

Satisfy a set of invariant properties
(perhaps including properties about specific examples)

↓

Minimize Constraint loss

Let us formally state the setting

Suppose we have a sentence α in predicate logic, defined over some atoms $X = \{X_1, X_2, \cdots, X_n\}$

Suppose each atom X_i is associated with a probability p_i, possibly from a neural model

Let the vector $\mathbf{p}$ denote the collection of probabilities $[p_1, p_2, \cdots, p_n]$ over the atoms

Our goal:

To define a loss function $L(\alpha, p)$ such that minimizing it produces a model (and associated probabilities) that assigns labels satisfying the sentence α

Operationalizing “logic-as-loss”

How do we convert constraints into logic?

Operationalizing "logic-as-loss"

How do we convert constraints into logic?

- Different approaches exist. We will look at two: semantic loss and t-norm losses

Operationalizing “logic-as-loss”

How do we convert constraints into logic?

- Different approaches exist. We will look at two: semantic loss and t-norm losses

Can we recover standard data losses (e.g. cross entropy) from them?

Operationalizing “logic-as-loss”

How do we convert constraints into logic?

- Different approaches exist. We will look at two: semantic loss and t-norm losses

Can we recover standard data losses (e.g. cross entropy) from them?

- For both the above approaches, yes.

Operationalizing “logic-as-loss”

How do we convert constraints into logic?

- Different approaches exist. We will look at two: semantic loss and t-norm losses

Can we recover standard data losses (e.g. cross entropy) from them?

- For both the above approaches, yes.

Do we need to change anything in the network architectures? Can we use any architectures?

Operationalizing “logic-as-loss”

How do we convert constraints into logic?

- Different approaches exist. We will look at two: semantic loss and t-norm losses

Can we recover standard data losses (e.g. cross entropy) from them?

- For both the above approaches, yes.

Do we need to change anything in the network architectures? Can we use any architectures?

- No. The only thing that changes is the loss function we are minimizing. We can use any network architecture and any optimizer

Operationalizing “logic-as-loss”

How do we convert constraints into logic?

- Different approaches exist. We will look at two: semantic loss and t-norm losses

Can we recover standard data losses (e.g. cross entropy) from them?

- For both the above approaches, yes.

Do we need to change anything in the network architectures? Can we use any architectures?

- No. The only thing that changes is the loss function we are minimizing. We can use any network architecture and any optimizer

A technical point: If our constraints are universally quantified over all possible inputs, do we compute a loss over infinite terms?

Operationalizing "logic-as-loss"

How do we convert constraints into logic?

- Different approaches exist. We will look at two: semantic loss and t-norm losses

Can we recover standard data losses (e.g. cross entropy) from them?

- For both the above approaches, yes.

Do we need to change anything in the network architectures? Can we use any architectures?

- No. The only thing that changes is the loss function we are minimizing. We can use any network architecture and any optimizer

A technical point: If our constraints are universally quantified over all possible inputs, do we compute a loss over infinite terms?

- Often in practice, we will instantiate the constraint to a finite set of examples (possibly and often unlabeled)

Summary

The main idea:

- Convert constraints in symbolic logic to loss functions
- Minimize the logic-based loss
- Use any network architecture and any optimizer
- At inference time, there is no change in how we use our networks

  The hope: By minimizing constraint violations, our resulting models will also satisfy constraints in future examples

Coming up: Two instantiations of constraint loss

1. Semantic loss
2. T-norm losses