Logical Neural Networks

Overview

- Central ideas
- Basic Setup
  - Logic
  - Inference
  - Learning
- Inference
- Learning
- Concluding Thoughts

Logical Neural Networks

Central Ideas

Central Idea:
1-1 Correspondence between logic and neural structure

- *Many approaches are based on Markov random fields (MRFs)… where each logical clause has a weight; the clauses are atomic i.e. their internal logical structure is not represented.*

- *The network structure is thus compositional and modular, e.g. able to represent that one clause may be a sub-clause of another. The representation is disentangled, versus approaches such as [19, 17] that use a vector representation, sacrificing interpretability of the network.*

Italicized text is from Riegel et. al, 2020

Central Idea:
Tolerance to incomplete knowledge via truth bounds

- *The line of approach embodied by MRFs make a closed-world assumption, i.e. that if a statement doesn’t appear in the KB, it is false. LNN does not require complete specification of all variables’ exact degree of truth, more generally maintaining upper and lower bounds for each variable*

Italicized text is from Riegel et. al, 2020

Central Idea:
Omnidirectional inference

- *LNN neurons express bidirectional relationships with each neighbor, allowing inference in any direction.*

- *MRF approaches that hide the internal logical structure of clauses cannot draw the same conclusions that a theorem prover can. Many/most neuro-symbolic approaches, e.g. those based on embeddings, are arguably only "logic-like" and typically do not demonstrate reliably precise deduction.*

Italicized text is from Riegel et. al, 2020

Logical Neural Networks

The Basic Setup

The Basic Setup – the Logic

- The underlying logic is fuzzy (assigning values to the logical syntax) and weighted (formulas have logical connectors associated with them)
- Inputs are initial truth value bounds [L,U] for formulas and/or atoms
  - e.g. atom **`inClass(student1)`** has a truth value in [0.9, 1.0]
  - e.g. formula **`inClass(student1) AND inClass(student2)`** has a truth value in [0.8,1.0]
- Outputs are also truth value bounds for formulas and/or atoms
- The formulas are known ahead of time
- The weights and biases associated with formulas are not
- The underlying logical connectors (fuzzy version of AND, OR, quantifiers, etc.) not only depend on the fuzzy values of the formulas being combined, but also the weight and bias of the operators
- Note that the same operator will have a different weight and bias for each formula it appears in

The Basic Setup - Inference

- Input to inference:
  - Set of formulas
  - Initial truth bounds for each atom and formula
  - Bias and weight for each formula (connector)
- Output:
  - Final truth bounds for each formula and atom
- Authors propose a upward-downward pass through the logic *(this is different from forward-backward pass used in gradient descent)*
- The algorithm propagates truth values from atoms to the formula (upward) and from the formulas to the atoms (downward) until convergence

The Basic Setup - Learning

- Neural architecture is derived directly from the a-priori known formulas
- Historical inputs and outputs used as samples during the training process
- Loss function depends not only on standard ML metrics (e.g., MSE) but also the number of inconsistencies (neurons associated with a truth bound where L>U.
- The functions used to combine formulas are differentiable with respect to the weights
- The forward pass of the learning process can be done with the inference algorithm (which is essentially a fixpoint operator)

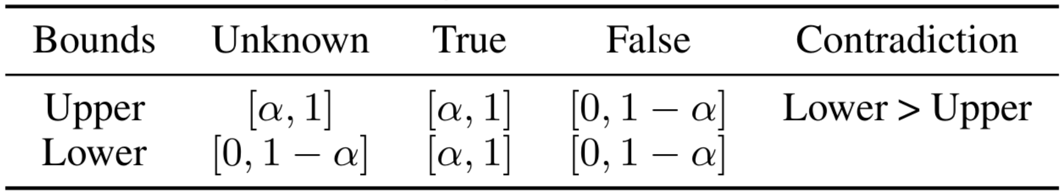
Logical Neural Networks

The Logic

The Logic

- Both atoms and formulas are associated with reals in the interval [0,1]
- In practice, LNN’s only provide information about a bound on the real-value associated with the atoms and formulas
    - This allows for uncertainty
    - The author propose a threshold α for assigning truth/falsehood/uncertainty in the result

Table 1: Primary truth value bound states



Bounds	Unknown	True	False	Contradiction
Upper	$[\alpha, 1]$	$[\alpha, 1]$	$[0, 1-\alpha]$	Lower > Upper
Lower	$[0, 1-\alpha]$	$[\alpha, 1]$	$[0, 1-\alpha]$	

“Table 1” is from Riegel et. al, 2020

Weighted Real-Valued Logic

- The logic is also weighted – but unlike other frameworks, components of logical formulas are weighted, hence they can be decomposed
- This allows real inference, unlike other logics that do not allow such decomposition
- This decomposition allows the logic to be directly translated into a neural structure via the syntax tree of each formula

Example Formula

- Disjunction

$$^{\beta}(\bigoplus_{i \in I} x_i^{\oplus w_i}) = f(1 - \beta + \sum_{i \in I} w_i x_i).$$

- Conjunction

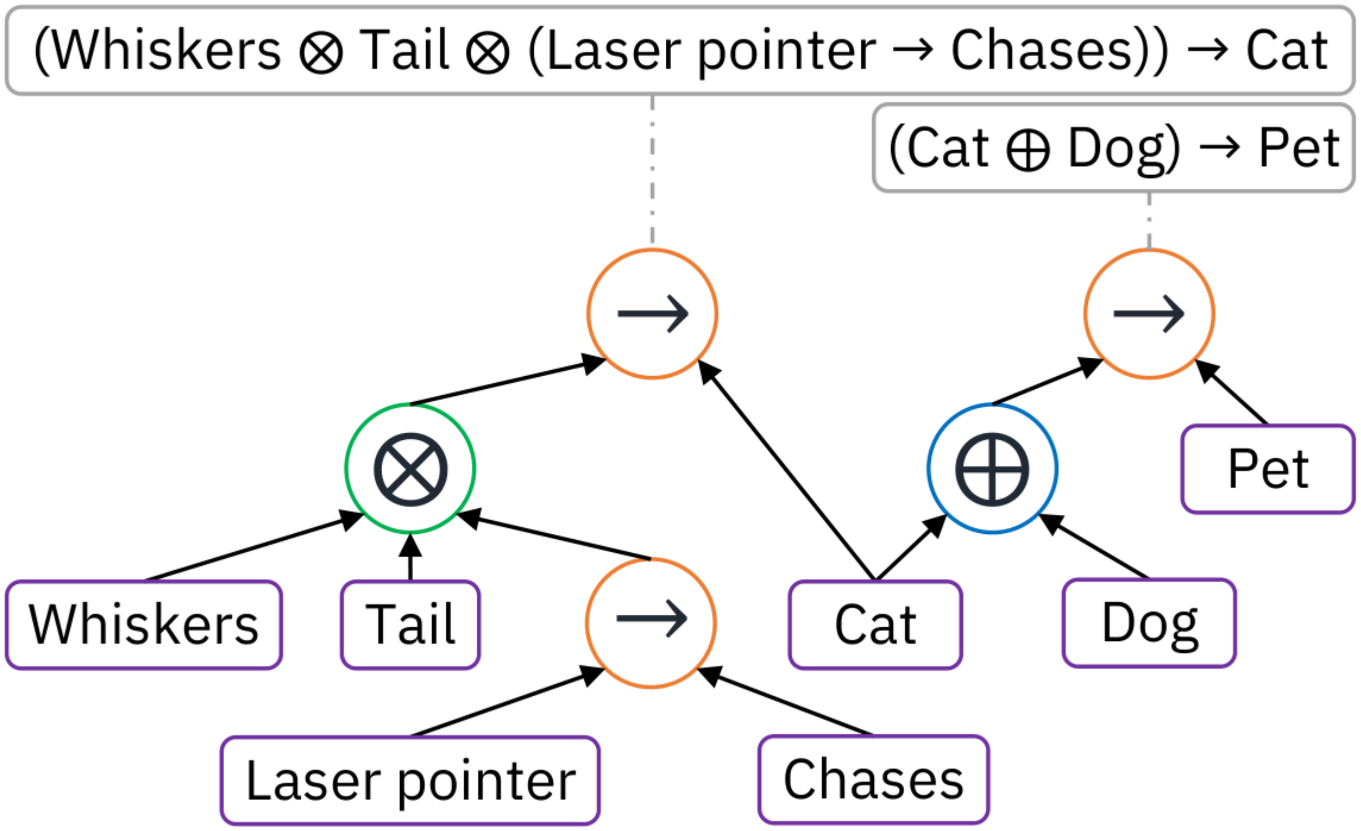
$$^{\beta}(\bigotimes_{i \in I} x_i^{\otimes w_i}) = f(\beta - \sum_{i \in I} w_i (1 - x_i))$$

- Implication

$$^{\beta}(x^{\otimes w_x} \rightarrow y^{\oplus w_y}) = f(1 - \beta + w_x(1 - x) + w_y y).$$

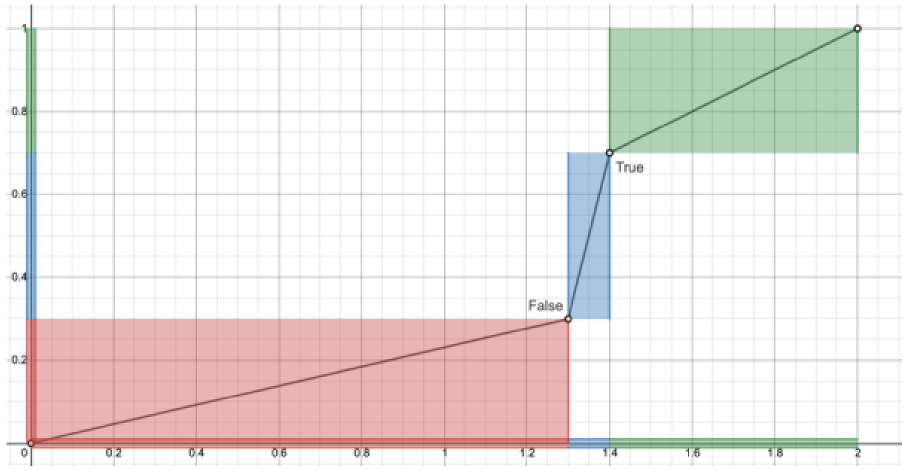
Formulas are from Riegel et. al, 2020 (Figure 1a)

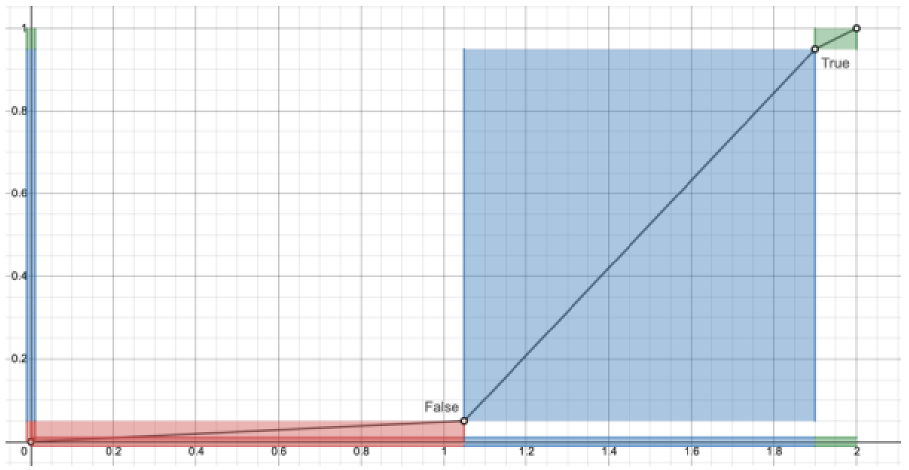
Syntax Tree == Neural Structure



The syntax tree is from Riegel et. al, 2020 (Figure 1a)

(Tailored) Activation Function





The authors are open about a variety of activation functions and settings of α, but do make arguments for a “tailored activation function” in the paper.

Chart is from Riegel et. al, 2020 (Sec. F.6)

Logical Neural Networks

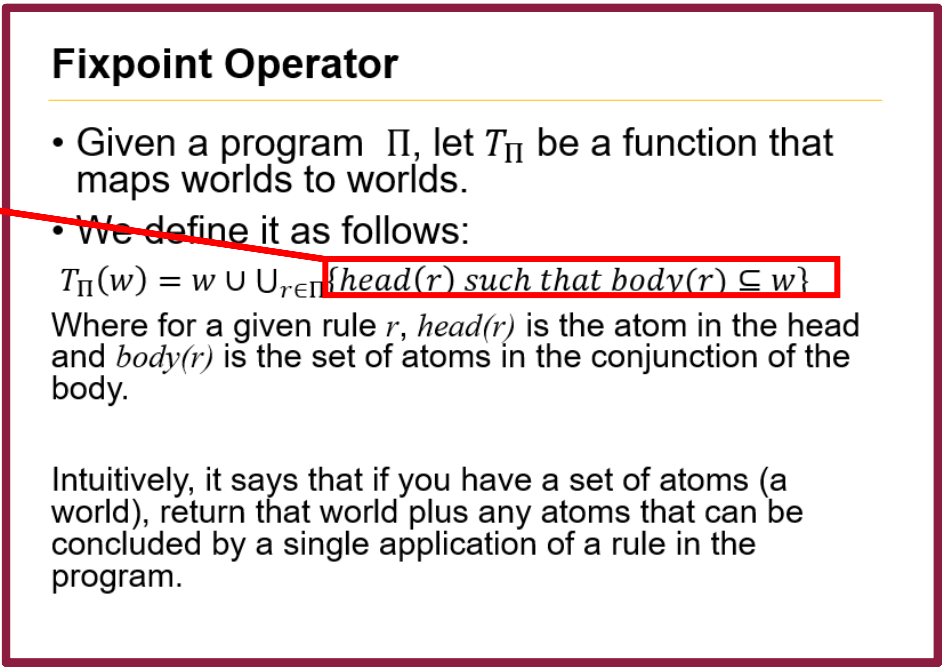
Inference

Recall: Fixpoint operator

We can think of a fixpoint operator, like what we discussed as applying “proof rules”

$$\frac{y}{x, x \rightarrow y}$$

Note: in this example the proof rule is a purely syntactic manipulation, while our fixpoint operator from the last lesson makes reference to the semantic structure



Fixpoint Operator

- Given a program Π, let T_Π be a function that maps worlds to worlds.
- We define it as follows:

$$T_\Pi(w) = w \cup \bigcup_{r\in\Pi} \boxed{\{head(r) \text{ such that } body(r) \subseteq w\}}$$

Where for a given rule r, *head(r)* is the atom in the head and *body(r)* is the set of atoms in the conjunction of the body.

Intuitively, it says that if you have a set of atoms (a world), return that world plus any atoms that can be concluded by a single application of a rule in the program.

Inset slide from Lesson 2: Logic Review

The Strategy Behind LNN Inference

- Define correct proof rules with respect to the logic
- Define how the bounds change based on atoms and negations are concluded from proof rules
- Prove a minimal required set of proof rules based on direction of inference (atoms to formulas vs. formulas to atoms)
- Put it all together in an algorithm that can be proven correct based on the above items

LNN Proof Rules and Tightening of Bounds

- The basic proof rules – these are not the precise definitions used in the algorithm, but rather showing the classical case to which LNN inference corresponds.

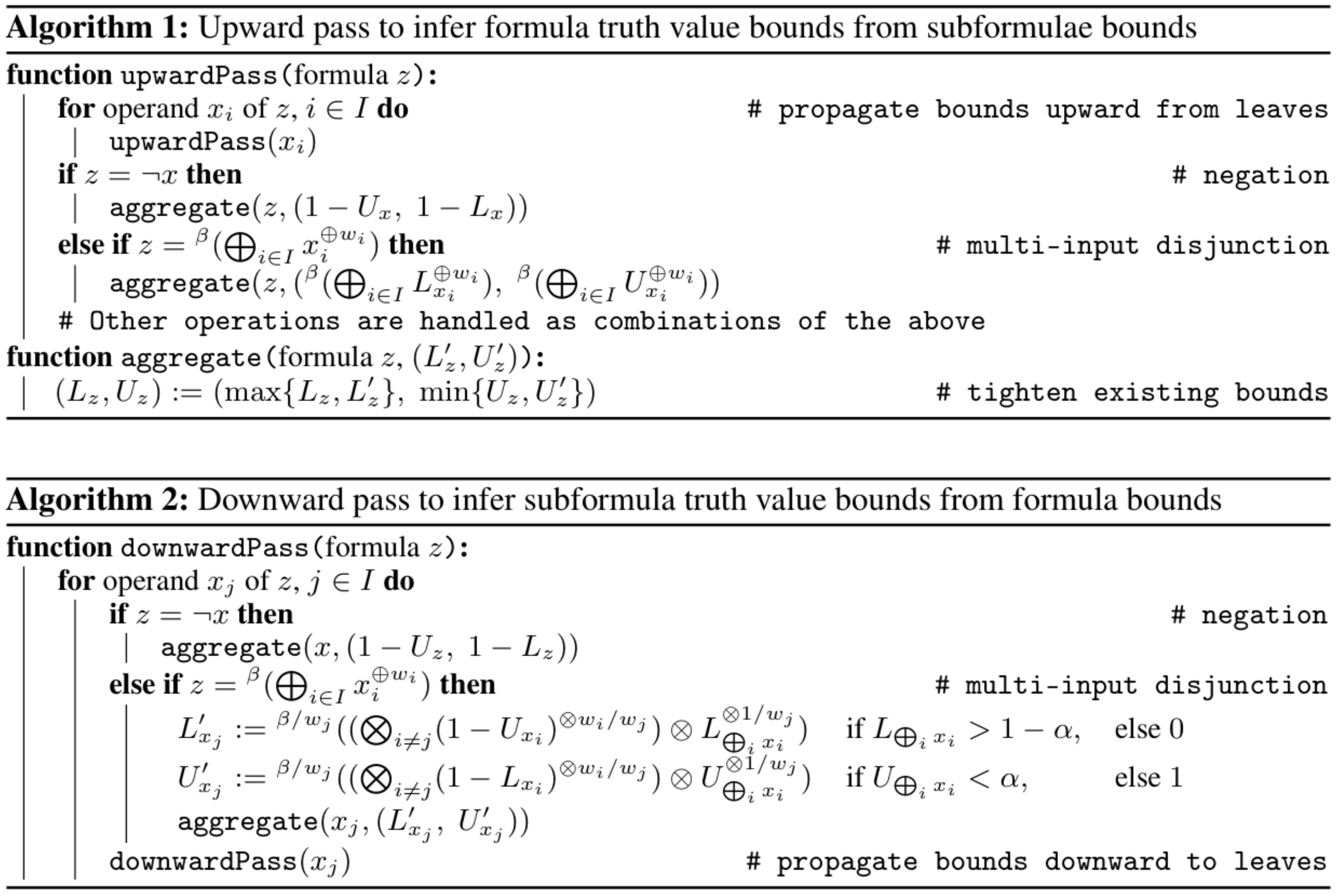
$$\begin{array}{rlrl} x, x \to y \vdash & y \quad (\textit{modus ponens}) & x, \neg(x \wedge y) \vdash \neg y & \text{(conjunctive syllogism)} \\ \neg y, x \to y \vdash & \neg x \quad (\textit{modus tollens}) & \neg x, \quad x \vee y \vdash \quad y & \text{(disjunctive syllogism)} \end{array}$$

- As LNN is not classical, it is tightening bounds based on the inference rules. This occurs as it concludes a literal (either an atom or its negation).

$$\begin{array}{ll} L_{\neg x} \geq \neg U_x = 1 - U_x, & L_x \geq \neg U_{\neg x} = 1 - U_{\neg x}, \\ U_{\neg x} \leq \neg L_x = 1 - L_x, & U_x \leq \neg L_{\neg x} = 1 - L_{\neg x}. \end{array}$$

Formalism is from Riegel et. al, 2020 (Sec. 4)

LNN Inference



Algorithm 1: Upward pass to infer formula truth value bounds from subformulae bounds

```
function upwardPass(formula $z$):
    for operand $x_i$ of $z$, $i \in I$ do                          # propagate bounds upward from leaves
        upwardPass($x_i$)
    if $z = \neg x$ then                                                  # negation
        aggregate($z, (1 - U_x,\ 1 - L_x)$)
    else if $z = {}^{\beta}(\bigoplus_{i \in I} x_i^{\oplus w_i})$ then                # multi-input disjunction
        aggregate($z, ({}^{\beta}(\bigoplus_{i \in I} L_{x_i}^{\oplus w_i}),\ {}^{\beta}(\bigoplus_{i \in I} U_{x_i}^{\oplus w_i}))$)
    # Other operations are handled as combinations of the above
function aggregate(formula $z$, $(L'_z, U'_z)$):
    $(L_z, U_z) := (\max\{L_z, L'_z\},\ \min\{U_z, U'_z\})$                    # tighten existing bounds
```

Algorithm 2: Downward pass to infer subformula truth value bounds from formula bounds

```
function downwardPass(formula $z$):
    for operand $x_j$ of $z$, $j \in I$ do
        if $z = \neg x$ then                                              # negation
            aggregate($x, (1 - U_z,\ 1 - L_z)$)
        else if $z = {}^{\beta}(\bigoplus_{i \in I} x_i^{\oplus w_i})$ then            # multi-input disjunction
            $L'_{x_j} := {}^{\beta / w_j}((\bigotimes_{i \neq j}(1 - U_{x_i})^{\otimes w_i / w_j}) \otimes L_{\bigoplus_i x_i}^{\otimes 1 / w_j})$   if $L_{\bigoplus_i x_i} > 1 - \alpha$,   else 0
            $U'_{x_j} := {}^{\beta / w_j}((\bigotimes_{i \neq j}(1 - L_{x_i})^{\otimes w_i / w_j}) \otimes U_{\bigoplus_i x_i}^{\otimes 1 / w_j})$   if $U_{\bigoplus_i x_i} < \alpha$,   else 1
            aggregate($x_j, (L'_{x_j},\ U'_{x_j})$)
        downwardPass($x_j$)                                         # propagate bounds downward to leaves
```

Formalism is from Riegel et. al, 2020 (Sec. 4)

Logical Neural Networks

Learning

Overall Strategy

- Use gradient descent to find weights, using normal back-propagation and the aforementioned inference process for the forward pass
- Loss function combines normal metrics (e.g., MSE) and a count of inconsistent neurons

Key Issues to Consider

- Parameters w and β must be set in a way such that classical logic outcomes for the operators behave as expected (i.e., if either proposition a or b has a value of 1, the disjunction should always result in 1)
- Learning parameters that fit vs. interpretability

Objective Function with Constraints

- To ensure reasonable settings of parameters, the authors show how the parameter-learning problem can be framed as an optimization problem with constraints

$$\begin{array}{llll} \min_{B,W} & E(B,W) + \sum_{k \in N} \max\{0, L_{B,W,k} - U_{B,W,k}\} & & \\ \text{s.t.} & \forall k \in N,\ i \in I_k, \quad \alpha \cdot w_{ik} - \beta_k + 1 \geq \alpha, & w_{ik} \geq 0 & (6) \\ & \forall k \in N, \quad \sum_{i \in I_k} (1-\alpha) \cdot w_{ik} - \beta_k + 1 \leq 1 - \alpha, & \beta_k \geq 0 & (7) \end{array}$$

- Here, $E(B,W)$ is the traditional loss function and the summation $\sum_{k \in N} \max\{0, L_{B,W,k} - U_{B,W,k}\}$ is designed to reduce the number of inconsistencies
- Some Issues: (fully listed in section F.1)
  - Requirement of additional slack parameters
  - Parameter updates require constraint satisfiaction
  - β must be learnt for each neuron, hinders interpretability and leads to overfitting

Formalism is from Riegel et. al, 2020 (Sec. 6)

Tailored Activation Function

Shown here is a tailored activation function for disjunction with $\beta=1$.

$$f_{\mathbf{w}}(x) = \begin{cases} x \cdot (1-\alpha)/x_{\mathrm{F}} & \text{if } 0 \leq x \leq x_{\mathrm{F}}, \\ (x - x_{\mathrm{F}}) \cdot (2\alpha - 1)/(x_{\mathrm{T}} - x_{\mathrm{F}}) + 1 - \alpha & \text{if } x_{\mathrm{F}} < x < x_{\mathrm{T}}, \\ (x - x_{\mathrm{T}}) \cdot (1-\alpha)/(x_{\max} - x_{\mathrm{T}}) + \alpha & \text{if } x_{\mathrm{T}} \leq x \leq x_{\max}, \end{cases}$$

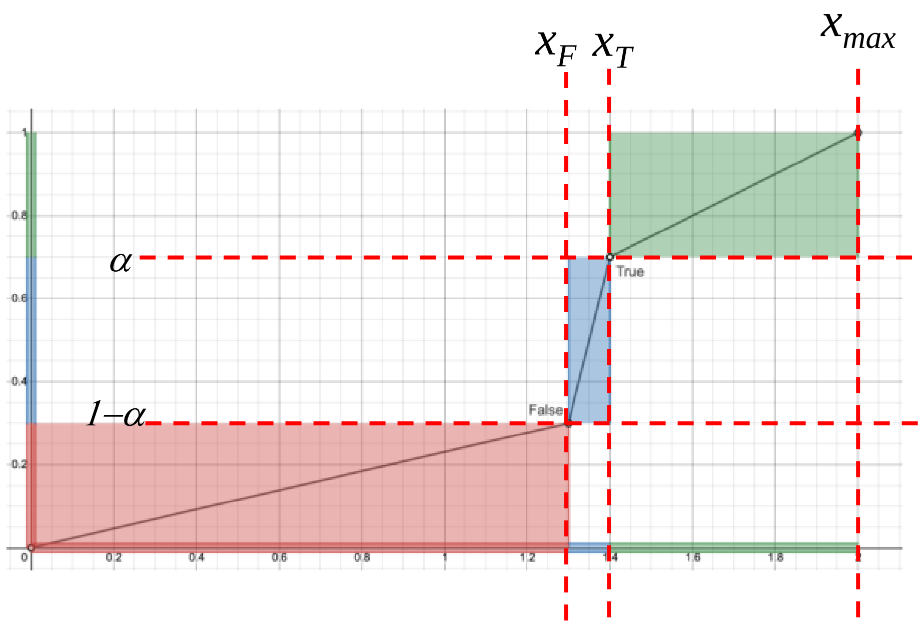
$$x_{\mathrm{F}} = \textstyle\sum_{i \in I} w_i \cdot (1-\alpha), \qquad x_{\mathrm{T}} = w_{\max} \cdot \alpha, \qquad x_{\max} = \textstyle\sum_{i \in I} w_i .$$

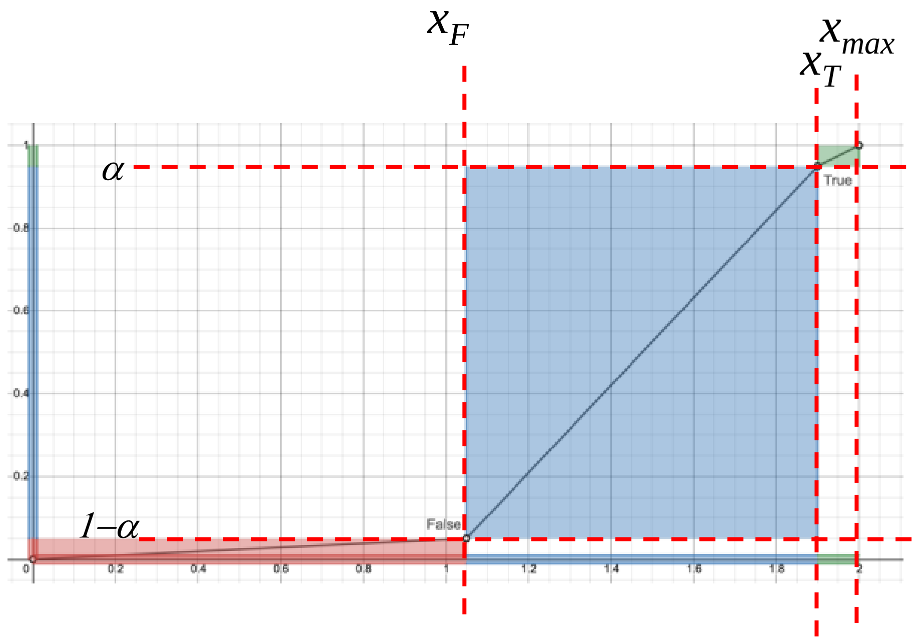
Key advantages:

- The logic is maintained regardless of the weights
- The authors claim that the output is independent of β, so they define it as $\beta_C = \sum_{i \in I} w_i$

Formalism is from Riegel et. al, 2020 (Sec. 6)

Tailored Activation Function with Four Points of Interest





$$(x_{min}, y_{min}) = (\beta - \sum_{i \in I} w_i, 0)$$

$$(x_F, y_F) = (\beta - \alpha w_{max}, 1 - \alpha)$$

$$(x_T, y_T) = (\beta - \sum_{i \in I} w_i \cdot (1 - \alpha), \alpha)$$

$$(x_{max}, y_{max}) = (\beta, 1)$$

Chart and formalism is from Riegel et. al, 2020 (Sup. F)

So what do we end up learning?

- With the tailored activation function, we are essentially learning parameters of the operators that allow the logic to return values appropriately in the various regions

Loss Function

- As the loss function includes a measurement of inconsistency, in experiments oscillation was observed
- In particular, parameter settings were causing differing groups of neuron to be inconsistent, and this changed with each epoch
- It is noteworthy that this illustrates that there could be multiple solutions to the same optimization problems with different parameters

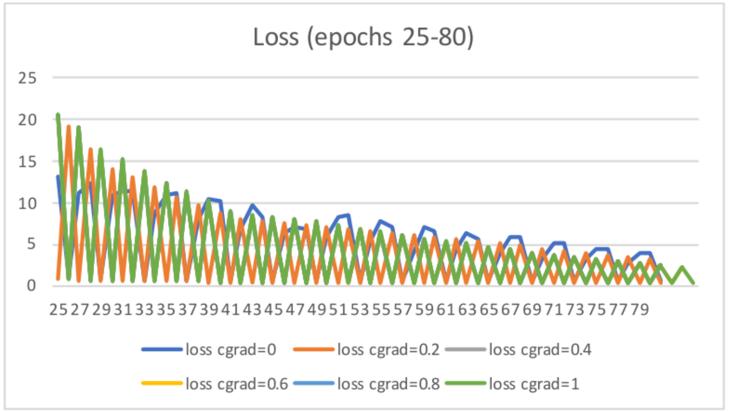


Chart and formalism is from Riegel et. al, 2020 (Sup. H)

Logical Neural Networks

Concluding Thoughts

LTN and LNN

- LTN’s (Badreddine et al., 2022) and LNN’s (Riegel et al., 2020)
- Key idea: use gradient descent to assign weight to fuzzy logic facts and formulas
- Minimized inconsistency via loss function
- Does not support learning of new logical structures (i.e. rules) – only assigns weights

Criteria	LNN	LTN
1. Arbitrary Queries	NO	YES
2. Symbolic explanation	YES	NO
3. Integrate prior knowledge and constraints	YES	YES
4. Assures consistency	NO	NO
5. Learn rule structure	NO	NO
6. Learn classical rules	NO	NO
7. Scalability	YES	YES

Kautz's Taxonomy for LNN: Category 4. Neuro:Symbolic → Neuro

- Uses architecture of #1 but with symbolic rules to guide the neural training process
- Note: this does not provide a symbolic derivation of the result
- Kautz's taxonomy does not specify if such frameworks should have guarantees, so LNN's fall into this category as well (at least according to Garcez and Lamb)
- Examples vary widely:
  - Symbolic information is normally compiled into the training data (not covered in this course)
  - LNN's
  - Differentiable ILP

Note that Garcez is a co-author of the LTN papers. LTN is considered category 5 – a higher tier.

Inconsistency

- Not fundamentally guaranteed
- Authors do mention that consistency check can be done during the training process
- It is noteworthy that consistency is based on neurons, not atoms – so for all formulas known a-prior, you know which ones will be inconsistent
- However, if you are checking an entailment query against the logic, there are no guarantees if it is correct

Modularity

- Does the learning of parameters around the operators lead to a modular logic program?
- If you want to check an entailment query of a formula, what weights do you use for the operator?
- If you must re-run training to check for entailment, then is it modular?
    - Would that a scalable practice?

Prior Knowledge

- Are LNN’s really incorporating prior knowledge or are they just using it in a way to guide model fit?
- Perhaps we can force prior knowledge neurons to always be consistent
- Does the assignment of weights to the operators change the meaning of the prior knowledge?

Ira A. Fulton Schools of Engineering

Arizona State University