Neural representations of formulae

A brief introduction

Karel Chvalovský

CIIRC CTU

Introduction

- the goal is to represent formulae by vectors (as good as possible)
  - we have seen such a representation using hand-crafted features based on tree walks, ...
  - neural networks have proved to be very good in extracting features in various domains—image classification, NLP, ...
- the selection of presented models is very subjective and it is a rapidly evolving area
- statistical approaches are based on the fact that in many cases we can safely assume that we deal only with the formulae of a certain structure
  - we can assume there is a distribution behind formulae
  - hence it is possible to take advantage of statistical regularities

Classical representations of formulae

- formulae are syntactic objects
- we use different languages based on what kind of problem we want to solve and we usually prefer the weakest system that fits our problem
  - classical / non-classical
  - propositional, FOL, HOL, ...
- there are various representations
  - standard formulae
  - normal forms
  - circuits
- there are even more types of proofs and they use different types of formulae
- it really matters what we want to do with them

Example—SAT

- we have formulae in CNF
  - we have reasonable algorithms for them
  - they can also simplify some things
  - note that they are not unique, e.g.,

    $$(p \to q) \land (q \to r) \land (r \to p)$$

    is equivalent to both

    $$(\neg p \lor q) \land (\neg q \lor r) \land (\neg r \lor p)$$

    and

    $$(\neg p \lor r) \land (\neg q \lor p) \land (\neg r \lor q)$$

- it is trivial to test formulae in DNF, but transforming a formula into DNF can lead to an exponential increase in the size of the formula

Semantic properties

- we want to capture the meaning of terms and formulae that is their semantic properties
- however, a representation should depend on the property we want to test
  - a representation of $(x - y) \cdot (x + y)$ and $x^2 - y^2$ should take into account whether we want to apply it on a binary predicate P which says
    - they are equal polynomials
    - they contain the same number of pluses and minuses
    - they are both in a normal form

Feed-forward neural networks

- in our case we are interested in supervised learning
- it is a function $f\colon \mathbb{R}^n \to \mathbb{R}^m$
- they are good in extracting features from the data

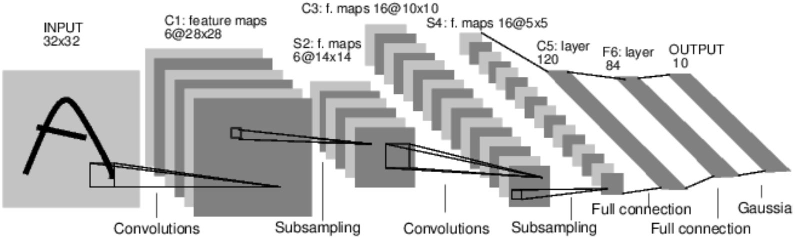


image source: PyTorch

Fully-connected NNs

Neuron

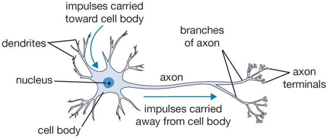


image source: cs231n

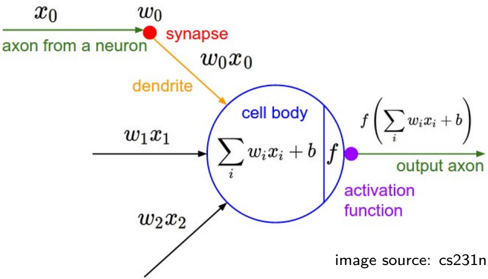


image source: cs231n

NN with two hidden layers

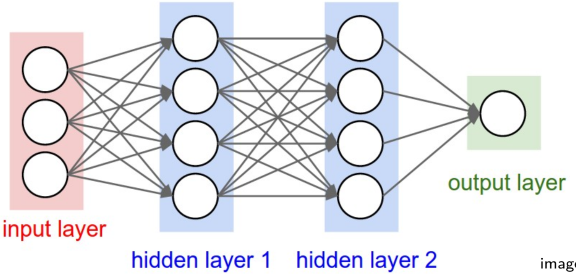


image source: cs231n

Activation functions

- they produce non-linearities, otherwise only linear transformations are possible
- they are applied element-wise

Common activation functions

- ReLU ($\max(0, x)$)
- tanh ($\frac{e^x - e^{-x}}{e^x + e^{-x}}$)
- sigmoid ($\frac{1}{1+e^{-x}}$)

Note that $\tanh(x) = 2\text{sigmoid}(2x) - 1$ and ReLU is non-differentiable at zero.

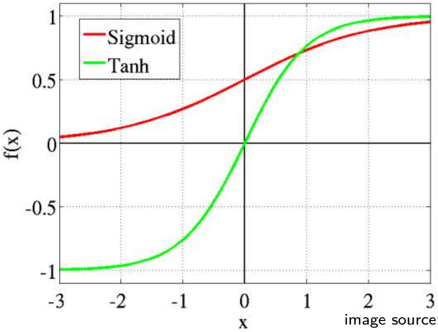


image source: here

Learning of NNs

- initialization is important
- we define a loss function
  - the distance between the computed output and the true output
- we want to minimize it by gradient descent (backpropagation using the chain rule)
  - optimizers—plain SGD, Adam, ...

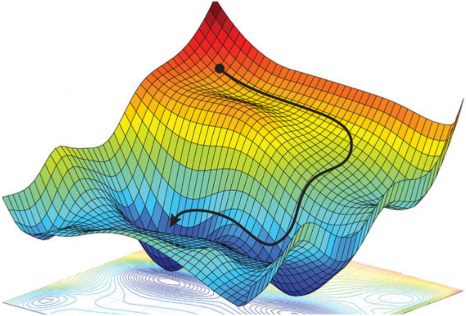


image source: Science

NNs and propositional logic

- already Pitts in his 1943 paper discusses the representation of propositional formulae
- it is well known that connectives like conjunction, disjunction, and negation can be computed by a NN
- every Boolean function can be learned by a NN
  - XOR requires a hidden layer
- John McCarthy: NNs are essentially propositional

Bag of words

- we represent a formula as a sequence of tokens (atomic objects, strings with a meaning) where a symbol is a token

$$p \to (q \to p) \quad \implies \quad X = \langle p, \to, (, q, \to, p,) \rangle$$
$$P(f(0, \sin(x))) \quad \implies \quad X = \langle P, (, f, (, \sin, (, x,),),) \rangle$$

- the simplest approach is to treat it as a bag of words (BoW)
  - tokens are represented by learned vectors
  - linear BoW is $\mathrm{emb}(X) = \frac{1}{|X|} \sum_{x \in X} \mathrm{emb}(x)$
  - we can "improve" it by the variants of term frequency–inverse document frequency (tf-idf)
- it completely ignores the order of tokens in formulae
  - $p \to (q \to p)$ becomes equivalent to $p \to (p \to q)$
- even such a simple representation can be useful, e.g., in Balunovic, Bielik, and Vechev 2018, they use BoW for guiding an SMT solver

Learning embeddings for BoW

- say we want a classifier to test whether a formula X is TAUT
  - a very bad idea for reasonable inputs
  - no more involved computations (no backtracking)
- we have embeddings in $\mathbb{R}^n$
- our classifier is a neural network $\mathrm{MLP}\colon \mathbb{R}^n \to \mathbb{R}^2$
  - if X is TAUT, then we want $\mathrm{MLP}(\mathrm{emb}(X)) = \langle 1, 0\rangle$
  - if X is not TAUT, then we want $\mathrm{MLP}(\mathrm{emb}(X)) = \langle 0, 1\rangle$
- we learn the embeddings of tokens
  - missing and rare symbols
- note that for practical reasons it is better to have the output in $\mathbb{R}^2$ rather than in $\mathbb{R}$

Recurrent NNs (RNNs)

- standard feed-forward NNs assume the fixed-size input
- we have sequences of tokens of various lengths
- we can consume a sequence of vectors by applying the same NN again and again and taking the hidden states of the previous application also into account
- various types
  - hidden state—linear, $\tanh$
  - output—linear over the hidden state

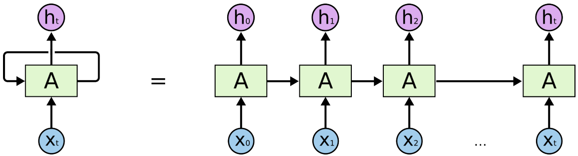


image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Problems with RNNs

- hard to parallelize
- in principle RNNs can learn long dependencies, but in practice it does not work well
  - say we want to test whether a formula is TAUT
    - $\dots \to (p \to p)$
    - $((p \wedge \neg p) \wedge \dots) \to q$
    - $(p \wedge \dots) \to p$

LSTM and GRU

- Long short-term memory (LSTM) was developed to help with vanishing and exploding gradients in vanilla RNNs
  - a cell state
  - a forget gate, an input gate, and an output gate
- Gated recurrent unit (GRU) is a “simplified” LSTM
  - a single update gate (forget+input) and state (cell+hidden)
- many variants — bidirectional, stacked, ...

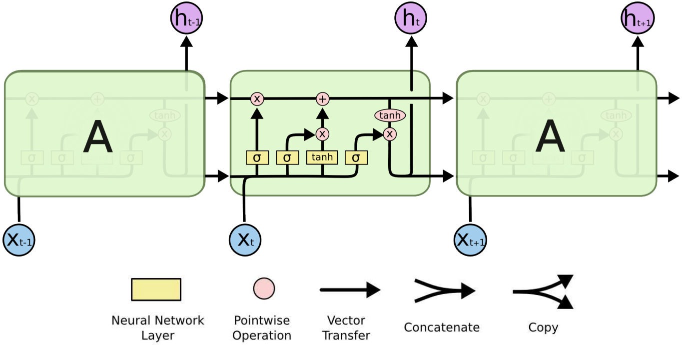


image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Convolutional networks

- very popular in image classification—easy to parallelize
- we compute vectors for every possible subsequence of a certain length
  - zero padding for shorter expressions
- max-pooling over results—we want the most important activation
- character-level convolutions—premise sel. (Irving et al. 2016)
  - improved to the word-level by "definition"-embeddings

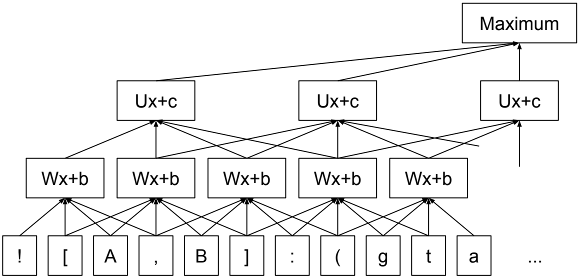


image source: Irving et al. 2016

Convolutional networks II.

- word level convolutions—proof guidance (Loos et al. 2017)
  - WaveNet (Oord et al. 2016) — a hierarchical convolutional network with dilated convolutions and residual connections

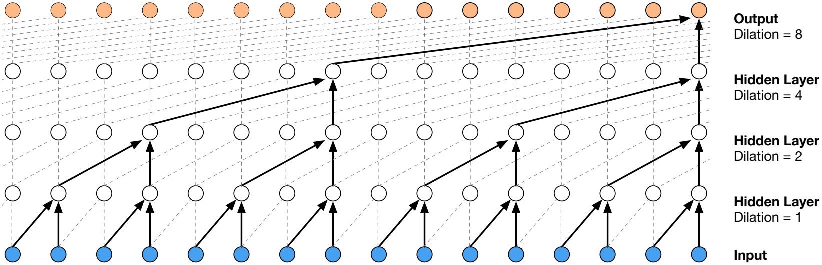
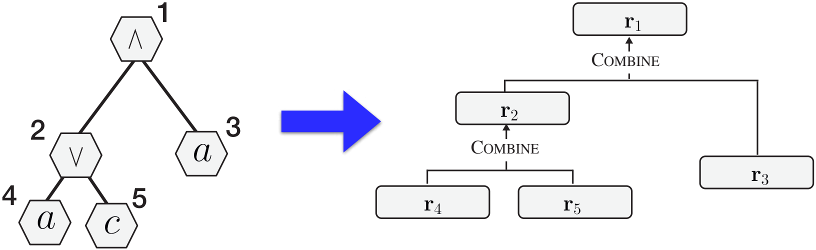


image source: Oord et al. 2016

Recursive NN (TreeNN)

- we have seen them in Enigma
- we can exploit compositionality and the tree structure of our objects and use recursive NNs (Goller and Kuchler 1996)

$(a \vee c) \wedge a$

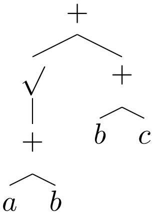


Syntax tree Network architecture

image source: EqNet slides

TreeNN (example)

- leaves are learned embeddings
  - both occurrences of b share the same embedding
- other nodes are NNs that combine the embeddings of their children
  - both occurrences of $+$ share the same NN
  - we can also learn one apply function instead
  - functions with many arguments can be treated using pooling, RNNs, convolutions etc.



term	representation
a	$\mathbb{R}^n$
b	$\mathbb{R}^n$
c	$\mathbb{R}^n$
$\sqrt{}$	$\mathbb{R}^n \to \mathbb{R}^n$
$+$	$\mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$

Notes on compositionality

- we assume that it is possible to "easily" obtain the embedding of a more complex object from the embeddings of simpler objects
- it is usually true, but

$$f(x, y) = \begin{cases} 1 & \text{if } x \text{ halts on } y, \\ 0 & \text{otherwise.} \end{cases}$$

- even constants can be complex, e.g., $\{\, x \colon \forall y (f(x, y) = 1) \,\}$
- very special objects are variables and Skolem functions (constants)
- note that different types of objects can live in different spaces as long as we can connect things together

TreeNNs

- advantages
  - powerful and straightforward—in Enigma we model clauses in FOL
  - caching
- disadvantages
  - quite expensive to train
  - usually take syntax too much into account
  - hard to express that, e.g., variables are invariant under renaming
- PossibleWorldNet (Evans et al. 2018) for propositional logic
  - randomly generated "worlds" that are combined with the embeddings of atoms
  - we evaluate the formula against many such worlds

EqNet (Allamanis et al. 2017)

- the goal is to learn semantically equivalent representations (equal terms should be as close as possible, i.e., the k-nearest neighbors algorithm)

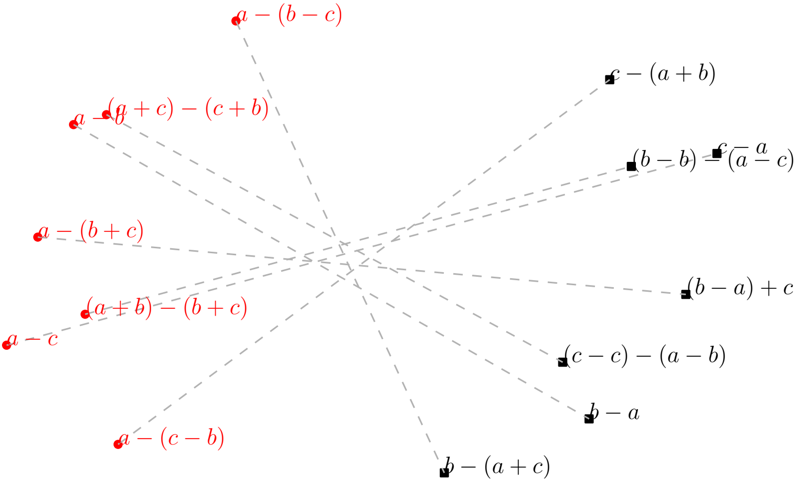


image source: Allamanis et al. 2017

EqNet

- a standard TreeNN improved by
  - normalization (embeddings have unit norm)
  - regularization (subexpression autoencoder)
    - aiming for abstraction and reversibility
    - denoising AE — randomly turn some weights to zero

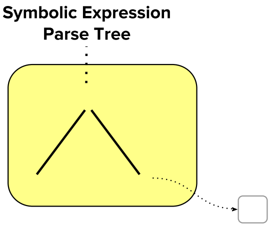


image source: Allamanis et al. 2017

Tree-LSTM (Tai, Socher, and Manning 2015)

- gating vectors and memory cell updates are dependent on the states of possibly many child units
- it contains a forget gate for each child
  - child-sum or at most N ordered children

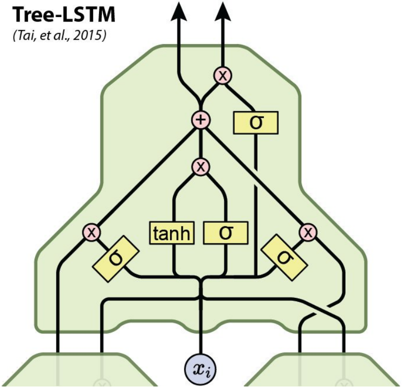
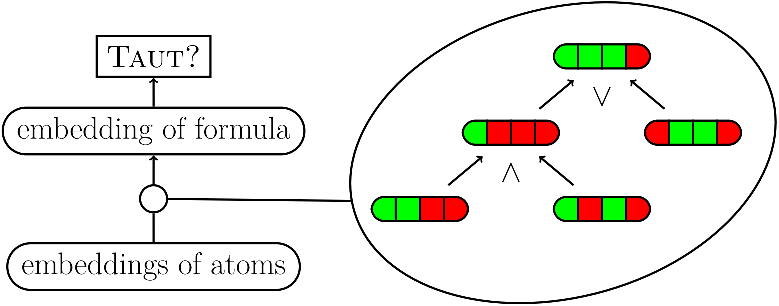


image source: Chris Olah

Bottom-up recursive model

Say we want to test whether a propositional formula is TAUT. We compute the embeddings of more complex objects from the embeddings of simple objects. We learn

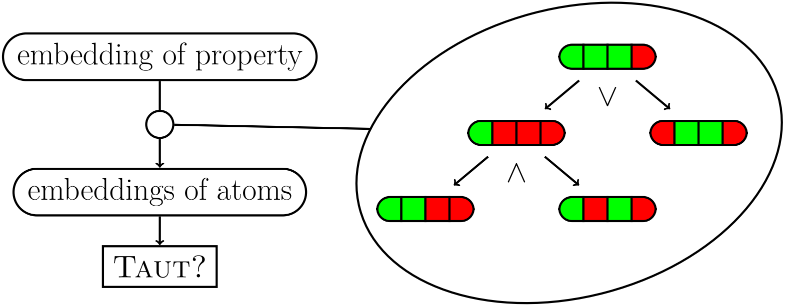
- the embeddings of atoms
- NNs for logical connectives (combine)



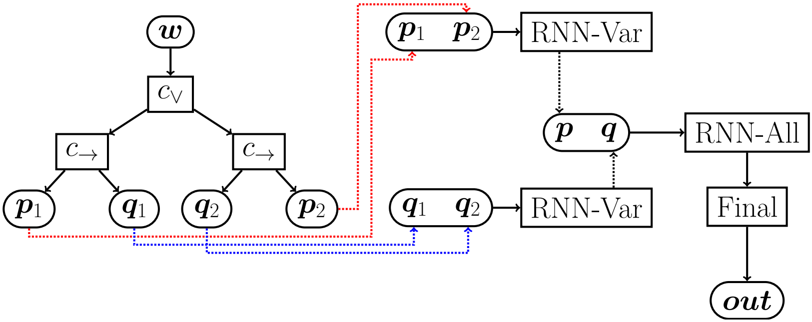
Top-down recursive model

We change the order of propagation; the embedding of the property is propagated to subformulae. We learn

- the embedding of the property (tautology)
- NNs for logical connectives (split)



Top-down model for $F = (p \to q) \vee (q \to p)$



We train the representations of $\boldsymbol{w}$, c_i, RNN-Var, RNN-All, and Final. These components are shared among all the formulae. For a single formula we produce a model (neural network) recursively from them.

Top-down model

Vectors (in $\mathbb{R}^d$):

- $\boldsymbol{w}$ is the input embedding of the property (tautology)
- $\boldsymbol{p}_1, \boldsymbol{p}_2, \boldsymbol{q}_1$, and $\boldsymbol{q}_2$ represent the individual occurrences of atoms in F, where $\boldsymbol{p}_1$ corresponds to the first occurrence of the atom p in F
- $\boldsymbol{p}$ and $\boldsymbol{q}$ represent all the occurrences of p and q in F, respectively
- $\boldsymbol{out} \in \mathbb{R}^2$ gives true/false

Neural networks:

- $\mathbf{c}_\vee$ and $\mathbf{c}_\rightarrow$ represent binary connectives $\vee$ and $\rightarrow$, respectively
  - they are functions $\mathbb{R}^d \rightarrow \mathbb{R}^d \times \mathbb{R}^d$, because $\vee$ and $\rightarrow$ are binary connectives
- **RNN-Var** aggregates vectors corresponding to the same atom
- **RNN-All** aggregates the outputs of RNN-Var components
- **Final** is a final decision layer

Properties of top-down models

Top-down models

- are insensitive to the renaming of atoms
- can evaluate unseen atoms and the number of distinct atoms that can occur in a formula is only bounded by the ability of RNN-All to correctly process the outputs of RNN-Var
- work quite well for some sets of formulae
- make it harder to interpret the produced representations
- can be probably reasonably extended to FOL, but it more or less leads to more complicated structures and hence graph NNs (GNNs)

FormulaNet (Wang et al. 2017)

- we represent higher-order formulae by graphs (GNNs)

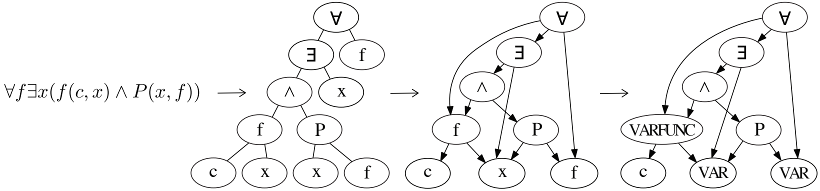
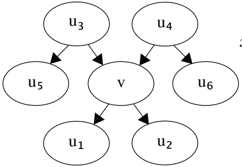


image source: Wang et al. 2017

FormulaNet — embeddings

- init is a one-hot repr. for every symbol (f, $\forall$, $\wedge$, VAR, ...)
- F_I and F_O are update functions for incoming and outgoing edges, respectively
- F_P combines F_I and F_O
- F_R, F_L, F_H are introduced to preserve the order of arguments (otherwise $f(x, y)$ is the same thing as $f(y, x)$)
  - F_R (F_L) is a treelet (triples) where v is the right (left) child
  - F_H is a treelet where v is the head
- updates are done in parallel
- the final representation of the formula is obtained by max-pooling over the embeddings of nodes

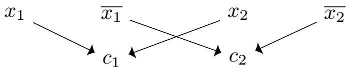


$$x_v^{t+1} = F_P^t\Big(x_v^t + \frac{1}{4}\Big[F_I^t(x_v^t, x_{u_1}^t) + F_I^t(x_v^t, x_{u_2}^t) + F_O^t(x_{u_3}^t, x_v^t) + F_O^t(x_{u_4}^t, x_v^t)\Big] + \frac{1}{3}\Big[F_R^t(x_{u_5}^t, x_{u_3}^t, x_v^t) + F_L^t(x_v^t, x_{u_4}^t, x_{u_6}^t) + F_H^t(x_{u_1}^t, x_v^t, x_{u_2}^t)\Big]\Big)$$

image source: Wang et al. 2017

NeuroSAT (Selsam, Lamm, et al. 2018)

- the goal is to decide whether a prop. formula in CNF is SAT
- two types of nodes with embeddings
  - literals
  - clauses
- two types of edges
  - between complementary literals
  - between literals and clauses
- we iterate message passing in two stages (back and forth)
  - we use two LSTMs for that
- invariant to the renaming of variables, negating all literals, the permutations of literals and clauses



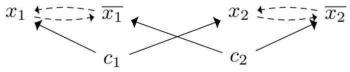


image source: Selsam, Lamm, et al. 2018

NeuroSAT voting

- we have a function vote that computes for every literal whether it votes SAT (red) or UNSAT (blue)
- all votings are averaged and the final result is produced
- it is sometimes possible to read an assignment—darker points
- it is sometimes possible to read an UNSAT core, but see NeuroCore (Selsam and Bjørner 2019)

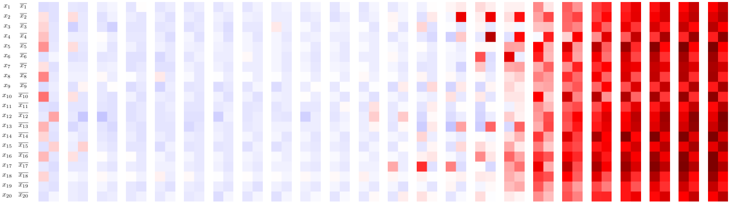


image source: Selsam, Lamm, et al. 2018

Circuit-SAT (Amizadeh, Matusevych, and Weimer 2019)

- we have a circuit (DAG) instead of a CNF
- they use smooth $\min$, $\max$ (fully differentiable w.r.t to all inputs), and $1 - x$ functions for logical operators
- GRUs are used for updates

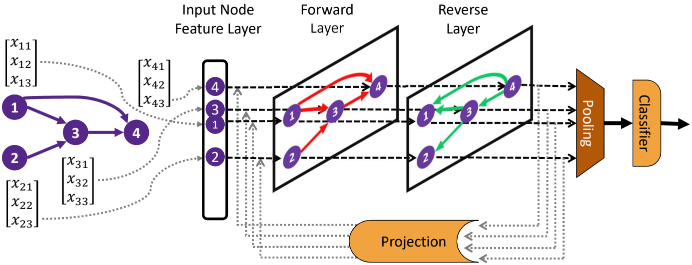


image source: Amizadeh, Matusevych, and Weimer 2019

Conclusion

- we have seen various approaches how to represent formulae
- it really matters what we want to do with our representations (property)
- there are many other relevant topics
  - attention mechanisms
    - popular for aggregating sequences
    - sensitive to hyperparameters
  - approaches based on ILP
    - usually we ground the problem to make it propositional
- maybe it is even better to formulate our problem directly in a language friendly to NNs and not to use classical formulae...
  - non-classical logics

Bibliography I

Allamanis, Miltiadis et al. (2017). "Learning Continuous Semantic Representations of Symbolic Expressions". In: *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 80–88. URL: http://proceedings.mlr.press/v70/allamanis17a.html.

Amizadeh, Saeed, Sergiy Matusevych, and Markus Weimer (2019). "Learning To Solve Circuit-SAT: An Unsupervised Differentiable Approach". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=BJxgz2R9t7.

Balunovic, Mislav, Pavol Bielik, and Martin Vechev (2018). "Learning to Solve SMT Formulas". In: *Advances in Neural Information Processing Systems 31*. Ed. by S. Bengio et al. Curran Associates, Inc., pp. 10337–10348. URL: http://papers.nips.cc/paper/8233-learning-to-solve-smt-formulas.pdf.

Chvalovský, Karel (2019). "Top-Down Neural Model For Formulae". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Byg5QhR5FQ.

Evans, Richard et al. (2018). "Can Neural Networks Understand Logical Entailment?" In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=SkZxCk-0Z.

Goller, C. and A. Kuchler (1996). "Learning task-dependent distributed representations by backpropagation through structure". In: *ICNN*, pp. 347–352.

Irving, Geoffrey et al. (2016). "DeepMath - Deep Sequence Models for Premise Selection". In: *Advances in Neural Information Processing Systems 29*. Ed. by D. D. Lee et al. Curran Associates, Inc., pp. 2235–2243. URL: http://papers.nips.cc/paper/6280-deepmath-deep-sequence-models-for-premise-selection.pdf.

Loos, Sarah M. et al. (2017). "Deep Network Guided Proof Search". In: *CoRR* abs/1701.06972. arXiv: 1701.06972. URL: http://arxiv.org/abs/1701.06972.

Oord, Aäron van den et al. (2016). "WaveNet: A Generative Model for Raw Audio". In: *CoRR* abs/1609.03499. arXiv: 1609.03499. URL: http://arxiv.org/abs/1609.03499.

Bibliography II

Selsam, Daniel and Nikolaj Bjørner (2019). "NeuroCore: Guiding High-Performance SAT Solvers with Unsat-Core Predictions". In: *CoRR* abs/1903.04671. arXiv: 1903.04671. URL: http://arxiv.org/abs/1903.04671.

Selsam, Daniel, Matthew Lamm, et al. (2018). "Learning a SAT Solver from Single-Bit Supervision". In: *CoRR* abs/1802.03685. arXiv: 1802.03685. URL: http://arxiv.org/abs/1802.03685.

Tai, Kai Sheng, Richard Socher, and Christopher D. Manning (July 2015). "Improved Semantic Representations From Tree-Structured Long Short-Term Memory Networks". In: *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, pp. 1556–1566. DOI: 10.3115/v1/P15-1150.

Wang, Mingzhe et al. (2017). "Premise Selection for Theorem Proving by Deep Graph Embedding". In: *Advances in Neural Information Processing Systems 30*. Ed. by I. Guyon et al. Curran Associates, Inc., pp. 2786–2796. URL: http://papers.nips.cc/paper/6871-premise-selection-for-theorem-proving-by-deep-graph-embedding.pdf.