Neural representations of formulas

A brief (and light) introduction

Karel Chvalovský

CIIRC CTU

Introduction

- the goal is to represent formulas by vectors (vector embedding) in a vector space (called the latent space); usually in a real vector space of a finite dimension
  - we have seen such a representation using hand-crafted features in ENIGMA
  - 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 formulas of a certain structure
  - we can assume there is a distribution behind formulas
  - hence it is possible to take advantage of statistical regularities

Classical representations of formulas

- formulas 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 formulas
  - normal forms
  - circuits
- there are even more types of proofs and they use different types of formulas
- it really matters what we want to do with them
  - test a property (equivalence, TAUT, SAT, ...)
  - premise selection
  - proof length estimation
  - conjecturing
  - solving equations

Example—SAT

- we have propositional formulas 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 formulas in DNF, but transforming a formula into DNF may lead to an exponential blow-up

Semantic properties

- we want to capture the meaning of terms and formulas that is their semantic properties
- however, a representation should depend on the property we want to test
  - they are equal polynomials
  - they contain the same number of pluses and minuses
  - they are both in a normal form

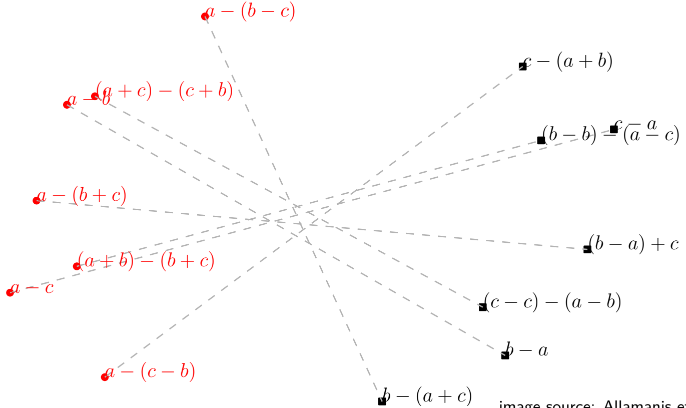


image source: Allamanis et al. 2017

NNs and propositional logic

- already McCulloch and Pitts in their 1943 paper discuss the representation of propositional formulas
- 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 (exclusive or) requires a hidden layer
- John McCarthy: (feed-forward) 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

$$
\begin{aligned}
p \rightarrow (q \rightarrow p) \quad &\Longrightarrow \quad X = \langle p, \rightarrow, (, q, \rightarrow, p,) \rangle \\
P(f(0, \sin(x))) \quad &\Longrightarrow \quad X = \langle P, (, f, (, \sin, (, x,),),) \rangle
\end{aligned}
$$

- 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 formulas
  - $p \rightarrow (q \rightarrow p)$ becomes equivalent to $p \rightarrow (p \rightarrow q)$
- 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 to use BoW 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 $\mathbb{R}^2$ (probability distribution over predicted output classes) 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

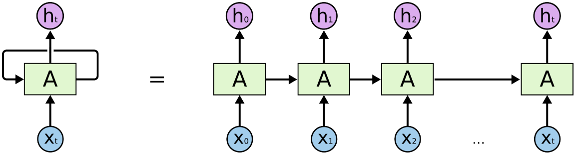


image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

- 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
    - $((p \wedge \neg p) \wedge \dots) \to q$
    - $(p \wedge \dots) \to p$
    - $(\dots \wedge p \wedge \dots) \to (\dots \wedge p \wedge \dots)$

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, ...

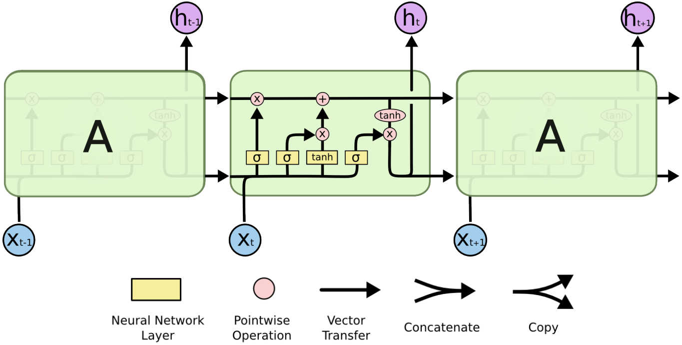


image source: http://colah.github.io/posts/2015-08-Understanding-LSTMs/

Encoder and decoder approach

Advanced models based on a “simple” encoder/decoder idea

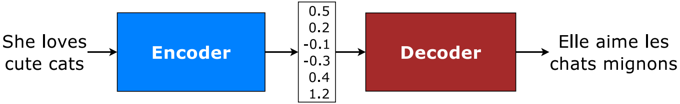


image source: Luong's thesis

have proved to be very successful in NLP (with many improvements like attention, towers of networks,...)

It is possible to abuse such advanced models directly (or with small modifications) for various tasks like

- autoformalization,
- conjecturing,
- solving equations, and
- symbolic integration.

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

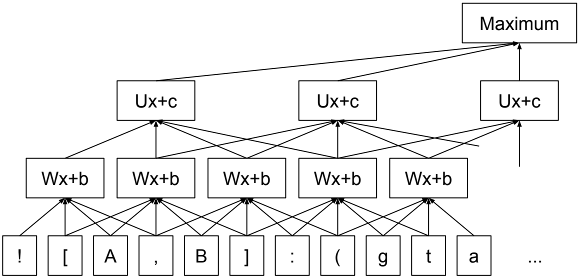


image source: Irving et al. 2016

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

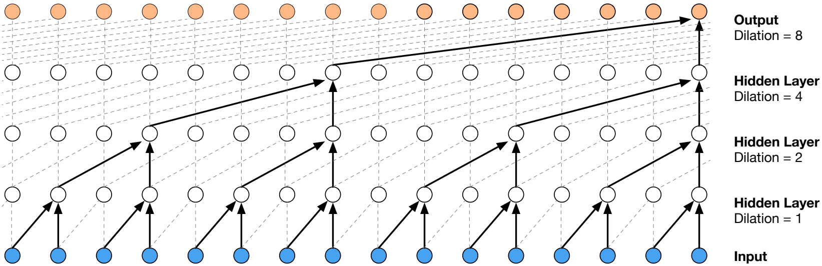


image source: Oord et al. 2016

Recursive NN (TreeNN)

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

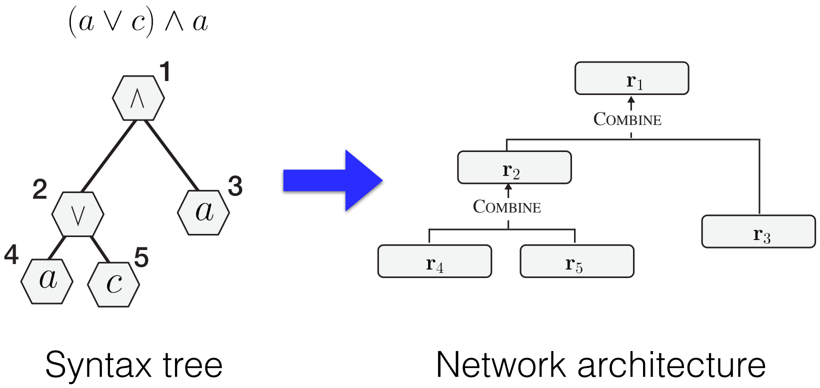
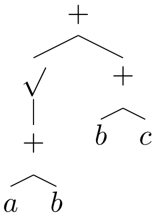


image source: EqNet slides

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 an unknown number of 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
  - natural and straightforward—in ENIGMA for FOL clauses
  - all occurrences of subexpressions have a fixed meaning (caching)
- disadvantages
  - all occurrences of subexpressions have a fixed meaning (no communication in the opposite direction)
  - quite expensive to train
  - usually take syntax too much into account
  - hard to express that, e.g., variables are invariant under renaming in many contexts
    - in ENIGMA we abstract away all first-order variables by a single embedding and similarly for Skolem symbols (arity matters)
  - hard to deal with symbols unseen during training
- many variants, e.g., PossibleWorldNet (Evans et al. 2018)
  - randomly generated "worlds" that are combined with the embeddings of atoms in propositional logic
  - 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)

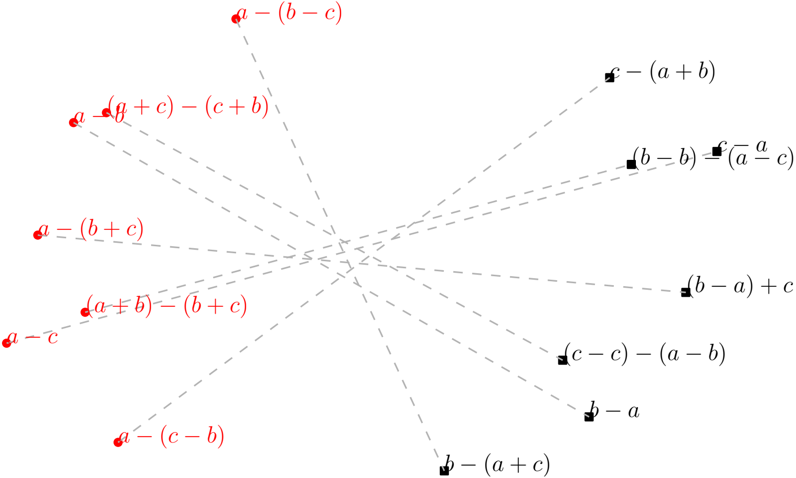


image source: Allamanis et al. 2017

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

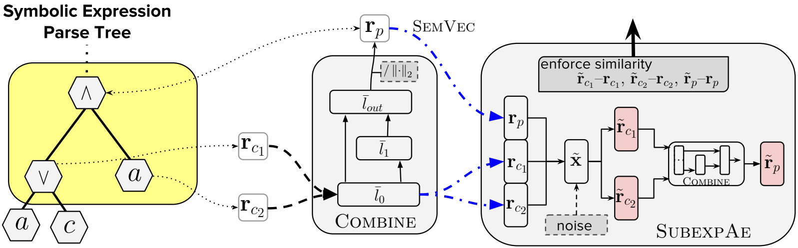


image source: Allamanis et al. 2017

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

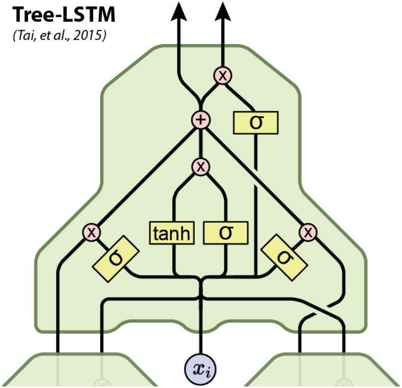
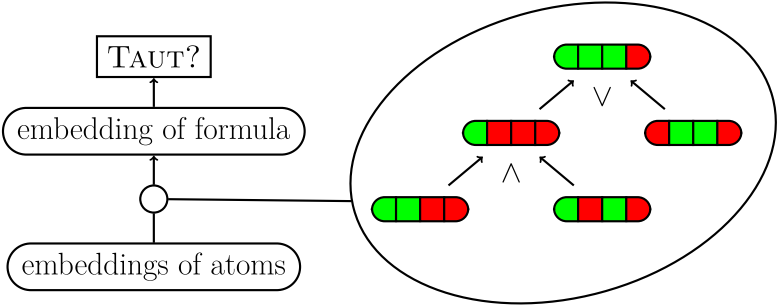


image source: Chris Olah

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

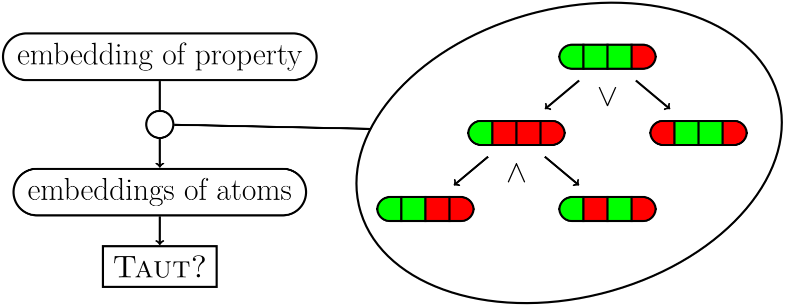
- the embeddings of atoms and
- NNs for logical connectives (combine).



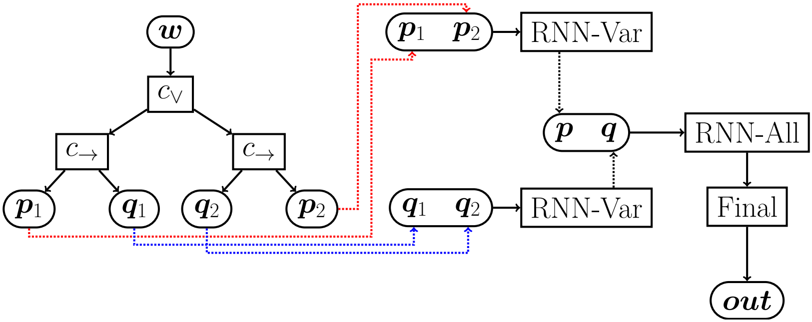
Top-down recursive model

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

- the embedding of the property (tautology) and
- 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 formulas. For a single formula we produce a model (neural network) recursively from them.

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 formulas,
- make it harder to interpret the produced representations, and
- can be reasonably extended to FOL, but it more or less leads to more complicated structures and hence graph NNs (GNNs).

Graph Neural Networks

GNNs generalize recursive NNs (TreeNNs) to arbitrarily structured graphs (data) and were introduced in Gori, Monfardini, and Scarselli 2005 and Scarselli et al. 2009.

A possible model is

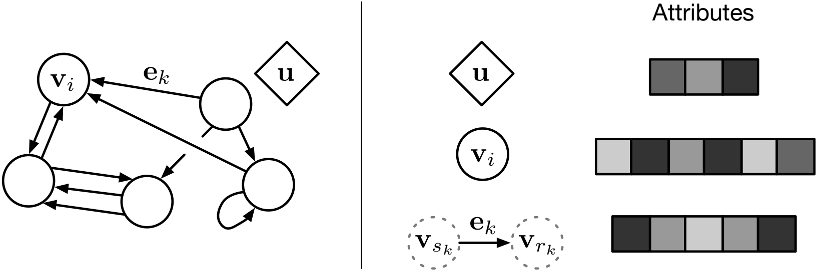


image source: Battaglia et al. 2018

which contains vertices (nodes), edges, and $\mathbf{u}$ is a global attribute.

Updates

GNNs use an iterative process to update embeddings:

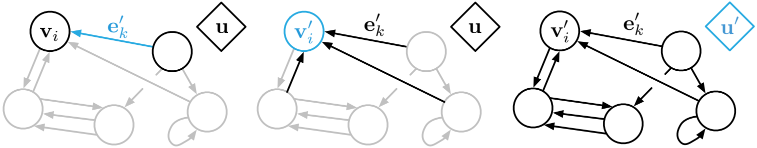


image source: Battaglia et al. 2018

- three types of updates — edges, nodes, and global attributes
- blue indicates what is being updated
- black indicates other elements that are involved in the update
  - various subsets and orders of these three updates are possible
  - before the update the values of same type are aggregated
    - the number of aggregated values is not bounded
- we update based on values in the previous step or use even the updated values (then the order of updates matters)
  - initial values — learned, random, . . .

Example: message passing

The spread of the information through the graph from a node:

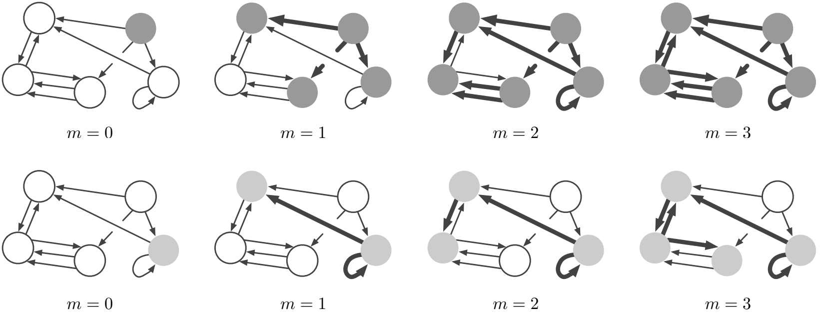


image source: Battaglia et al. 2018

This happens simultaneously for all nodes (and edges) not only for the emphasized ones.

FormulaNet (Wang et al. 2017)

We represent higher-order formulas by graphs (GNNs):

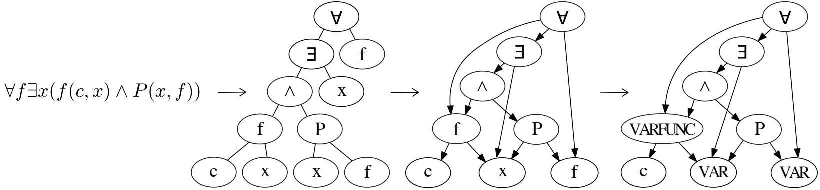
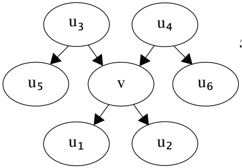


image source: Wang et al. 2017

Note that such a representation does not take the order of arguments into account—$f(c, x)$ is indistinguishable from $f(x, c)$. There are various way how to deal with this issue, which occurs also in other GNN models.

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
    - 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



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

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 under the renaming of variables, negating all literals, the permutations of literals and clauses

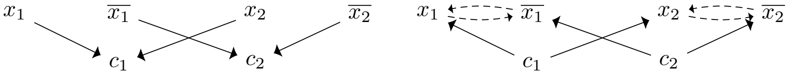


image source: Selsam, Lamm, et al. 2018

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
- in NeuroCore (Selsam and Bjørner 2019) a variant of NeuroSAT is used to help a CDCL solver by periodically adjusting variable activity scores

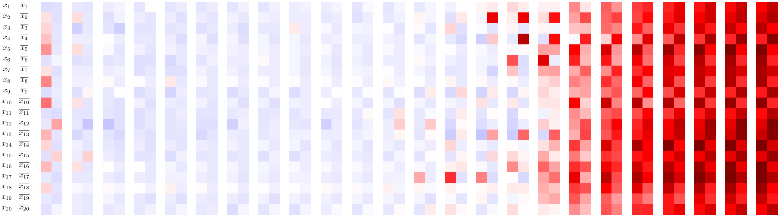


image source: Selsam, Lamm, et al. 2018

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

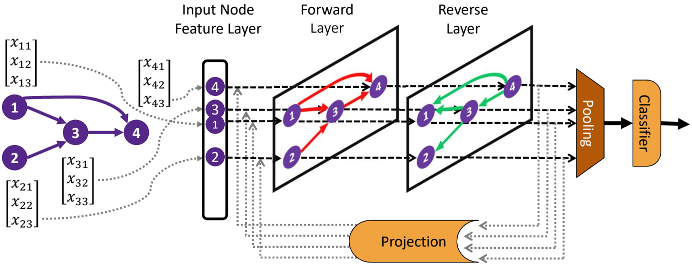


image source: Amizadeh, Matusevych, and Weimer 2019

Properties of representations using GNNs

- they are invariant under various permutations and the renamings of symbols; usually only relations between symbols (and their types) matter not their actual names
- it is possible to naturally produce a graph containing more formulas that share various components and hence encode the whole problem, e.g.,

$$\Gamma \vdash \varphi$$

- standard GNNs are unable to distinguish various non-isomorphic structures (graphs); note that only the neighbors are taken into account
  - there is a connection with the Weisfeiler–Leman algorithm
- there have been proposed many GNN models recently

Conclusion

- we have seen various approaches how to represent formulas (and you will see one more instance of GNNs)
- 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 formulas. . .
  - 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.

Battaglia, Peter W. et al. (2018). “Relational inductive biases, deep learning, and graph networks”. In: *CoRR* abs/1806.01261. arXiv: 1806.01261. URL: http://arxiv.org/abs/1806.01261.

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.

Gori, M., G. Monfardini, and F. Scarselli (2005). “A new model for learning in graph domains”. In: *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.* Vol. 2, 729–734 vol. 2.

Grohe, Martin (2020). *word2vec, node2vec, graph2vec, X2vec: Towards a Theory of Vector Embeddings of Structured Data*. arXiv: 2003.12590 [cs.LG].

Bibliography II

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.

Scarselli, F. et al. (2009). "The Graph Neural Network Model". In: *IEEE Transactions on Neural Networks* 20.1, pp. 61–80.

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 (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.