Recurrent unsupervised neural network for quantified Boolean formulae – project notes

Miika Hannula

April 17, 2021

1 Introduction

Recently there has been a flurry of interest in neural networks addressing NP-complete problems [1, 4, 5, 6, 7]. This project, which is still in progress, aims at harder problems that are typically represented as quantified Boolean formulae.

The architecture and source code are modified and expanded from those behind [7], which introduces an unsupervised recurrent neural network for the maximum constraint satisfaction problem (RUN-CSP). The idea is clever yet simple. In short, the network outputs a soft variable assignment which yields a soft truth value for each constraint. While training, the network seeks to minimize the combined negative log-likelihood over the soft truth values of the constraints. For evaluation, a hard assignment is obtained by taking the argmax of the soft assignment. The authors show, for instance, that a trained network is able to compete with state-of-the-art solvers over the maximum 2-satisfiability problem.

This project seeks to expand ideas from [7] to quantified Boolean formulae. We introduce a RUN-QBF network for predicting the truth value of a quantified Boolean formula. While the network is in principle designed to accommodate instances with unrestricted quantifier alternation, training such models is not expected to be feasible. Our focus is thus restricted on 2QBF instances for the second level Π_2^{P} of the polynomial hierarchy. That is, we consider quantified Boolean formulae in which a single universal quantifier block is followed by a single existential quantifier block.

Foremost, this is a personal project for practicing RNN networks with the Tensorflow framework. Not coincidentally, the subject matter relates to the authors' recent research interests [2]. At the moment of writing these notes, the network is up and running but would need additional tuning and re-training.

Related work. There are different ways to utilize neural networks in QBF solving. One is to learn heuristics for existing algorithms [3], to compete with hand-written ones. Another is to create architectures that directly address the problem. How feasible, then, is this approach? [9] suspects that the same graph neural networks which have succesfully tackled SAT (cf. [6]) do not generalize to QBF because of their inability to reason about unsatisfiability. Very recently, [8] used neural Monte Carlo Tree Search for solving QBF instances. The test cases in that paper, however, are very small, consisting of only 21 quantified variables and 9 clauses. It seems not much is currently known about the prospects of deep learning for decision problems beyond NP.

2 Preliminaries

The canonical complete problem for non-deterministic polynomial time (NP) is the *Boolean satisfiability problem* SAT. This problem is to determine whether there exists a variable assignment that satisfies a given a Boolean formula. Beside NP, other levels of the polynomial hierarchy, as well as polynomial space (PSPACE), are likewise captured logically. The standard problem here is the quantified Boolean formula problem (QBF). The input is a *quantified Boolean formula* of the form

$$Q_1\vec{p}_1 \dots Q_n\vec{p}_n\theta,$$

where $\vec{p}_1, \dots, \vec{p}_n$ are sequences of Boolean variables, $Q_1, \dots, Q_n \in \{\exists, \forall\}$ is an alternating sequence of quantifiers, and θ is a quantifier-free Boolean formula. The output is "yes" if the formula is true, and "no" otherwise. This problem is complete for PSPACE, and for fixed n, it is complete for Σ_n^{P} if $Q_1 = \exists$ (respectively, Π_n^{P} if $Q_1 = \forall$).

3 RUN-QBF

3.1 Idea

For a high-level illustration of RUN-QBF, consider an example input formula

$$\phi_{\Pi_2} = \forall x \exists y (\overline{y} \wedge (x \vee y)).$$

For solving formulae of this form, we first train a RUN-QBF model $\mathcal{N}_{\Sigma_1^{\mathsf{P}}}$ to output variable assignments that maximize the number of satisfied clauses in Boolean formulae. Then, we train another RUN-QBF model $\mathcal{N}_{\Pi_2^{\mathsf{P}}}$ that outputs variable assignments for universally quantified variables and then calls $\mathcal{N}_{\Sigma_1^{\mathsf{P}}}$ to run in inference mode. Thus, these two models essentially play the role of *falsifier* and *verifier* in the game semantics of classical logic. The challenge is to train them as well as possible.

The training flow for $\mathcal{N}_{\Pi_2^{\mathsf{P}}}$ w.r.t. a batch $\{\phi_{\Pi_2}\}$ of size one would be as follows:

1. $\mathcal{N}_{\Pi_2^{\mathsf{P}}}$ outputs soft assignments for x w.r.t. ϕ_{Π_2}, iterating its LSTM cell.
2. For each assignment of x: $\mathcal{N}_{\Sigma_1^{\mathsf{P}}}$
   (a) computes weights[1] for clauses in
   $$\phi_{\Sigma_1} = \exists y (\overline{y} \wedge y),$$
   obtained from ϕ_{Π_2} by removing all universally quantified literals, and then
   (b) outputs soft assignments for y w.r.t. the weighted ϕ_{Σ_1}, iterating its LSTM cell.
3. Each soft assignment yields the combined negative log-likelihood over the soft truth values of the clauses.
4. $\mathcal{N}_{\Pi_2^{\mathsf{P}}}$ is updated to maximize a weighted average over these values.

A hard assignment is obtained by taking the argmax of the soft assignment associated with the maximum number of iterations.

[1]Here, the weight of the singleton clause $C_1 = \overline{y}$ is 1, and the weight of the other singleton clause $C_2 = y$ is $(1-p)$, where $p \in [0,1]$ is the soft value of x.

3.2 Architecture

For now, see [7] for the architecture of RUN-CSP. We have expanded it as follows:

- RUN-CSP handles only binary constraints. In order to deal with arbitrary QBF instances, RUN-QBF has been extended to deal with ternary clauses. Thus, while RUN-CSP contains two different message passing protocols (one for symmetric and another for non-symmetric binary constraints), RUN-QBF contains more protocols to take into account all possible symmetries in a ternary constraint. Another factor increasing the number of message networks is that different variables in the same constraint may be quantified at different levels.
- Each message that is sent between variable states in the same constraint is now weighted by the corresponding weight of that clause.

The implementation is extended to accommodate quantified formulae and the layered training style described above. A QBF formula in QDIMACS format is transformed to a Python object representing an equivalent formula in 3CNF form. This object is then transformed to an input dictionary.

4 Challenges

1. An architecture designed to minimize or maximize satisfied constraints may not be that informative in terms of SAT or QBF, unless the network obtains a very good performance. It may be easy to satisfy 99.9% clauses for one false formula, and hard to do the same for another true formula. Our tests with benchmark 2QBF instances achieved $> 99\%$ satisfaction rate for most instances, but 100% satisfaction rate for only 1/100 tested formulae.[2] Possible remedy: train $\mathcal{N}_{\Sigma_1^P}$ with larger and more complex datasets and/or with greater state size.
2. The loss function needs recalibration for $\mathcal{N}_{\Pi_2^P}$. Currently, the values of universally quantified variables converge to 1 after few iterations. Possible remedy: less weight on early iterations.

[2]Additionally, the universally quantified variables were not optimal, as explained in the next item.

3. The loss function is too heavy for the second level. Consequently, training $\mathcal{N}_{\Pi_2^{\mathrm{P}}}$ requires too much resources.[3] Possible remedy: get rid of the Cartesian logic behind the loss function.

One thing to consider is to start with much simpler instances, as in [8], instead of the benchmark instances used in solver competitions.

5 Conclusion

Creating neural networks for solving QBF instances is an interesting open problem. These notes describe the author's side project which for now has to postponed due to lack of time.

References

[1] S. Amizadeh, S. Matusevych, and M. Weimer. Learning to solve circuit-sat: An unsupervised differentiable approach. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

[2] M. Hannula, J. Kontinen, M. Lück, and J. Virtema. On the complexity of horn and krom fragments of second-order boolean logic. In C. Baier and J. Goubault-Larrecq, editors, *29th EACSL Annual Conference on Computer Science Logic, CSL 2021, January 25-28, 2021, Ljubljana, Slovenia (Virtual Conference)*, volume 183 of *LIPIcs*, pages 27:1–27:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

[3] G. Lederman, M. N. Rabe, S. Seshia, and E. A. Lee. Learning heuristics for quantified boolean formulas through reinforcement learning. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

[4] H. Lemos, M. O. R. Prates, P. H. C. Avelar, and L. C. Lamb. Graph colouring meets deep learning: Effective graph neural network models for combinatorial problems. In *31st IEEE International Conference on Tools with Artificial Intelligence, ICTAI 2019, Portland, OR, USA, November 4-6, 2019*, pages 879–885. IEEE, 2019.

[3] Tested with Google Colab and the AWS EC2 instance g4dn.xlarge.

[5] M. O. R. Prates, P. H. C. Avelar, H. Lemos, L. C. Lamb, and M. Y. Vardi. Learning to solve np-complete problems: A graph neural network for decision TSP. In *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*, pages 4731–4738. AAAI Press, 2019.

[6] D. Selsam, M. Lamm, B. Bünz, P. Liang, L. de Moura, and D. L. Dill. Learning a SAT solver from single-bit supervision. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019.* OpenReview.net, 2019.

[7] J. Tönshoff, M. Ritzert, H. Wolf, and M. Grohe. Graph neural networks for maximum constraint satisfaction. *Frontiers Artif. Intell.*, 3:580607, 2020.

[8] R. Xu and K. J. Lieberherr. Solving QSAT problems with neural MCTS. *CoRR*, abs/2101.06619, 2021.

[9] Z. Yang, F. Wang, Z. Chen, G. Wei, and T. Rompf. Graph neural reasoning for 2-quantified boolean formula solvers. *CoRR*, abs/1904.12084, 2019.