Learning Traveling Salesperson Routes with Graph Neural Networks

Marcelo Prates*
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
morprates@inf.ufrgs.br

Pedro Avelar*
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
pedro.avelar@inf.ufrgs.br

Luis Lamb
Institute of Informatics
Federal University of Rio Grande do Sul
Porto Alegre, Brazil
lamb@inf.ufrgs.br

Abstract

1

1 Introduction

2 Motivation

Graph Neural Networks (GNNs) have been applied to a wide range of relevant problems [citations needed]. Very interesting recent work has shown how GNNs can be employed to bridge the divide between the neural and symbolic schools of artificial intelligence. In particular, Selsam *et al.* [2018] has succeeded in training a GNN as a boolean satisfiability (SAT) predictor by feeding it CNF formulas as features and satisfiability bits (one per formula) as labels, thus learning a rudimentary SAT solver from single-bit supervision. The "neurosat" architecture, as the authors call it, is rather simple: it consists of assigning a multidimensional embedding $\in \mathbb{R}^d$ for each literal (i.e. x_3, $\neg x_1$, x_{10}, etc.) and each clause in a CNF formula and then connecting these embeddings accordingly so that they can send messages to one another – literals are connected with the clauses in which they appear and vice-versa, and each literal is also connected to its negated variant (i.e. x_1 and $\neg x_1$). The model is run for a given number of message passing iterations in which each node in the GNN (i.e. a literal or a clause) adds up element-wise all of the embeddings of its incoming messages, received along its adjacencies, to obtain an accumulated message. The main trainable component of this architecture is a recurrent unit assigned with updating the embedding of a node with this accumulated message. In a nutshell, neurosat iterates a process of "embedding refinement" in which literals and clauses become enriched with information about their symbolic neighborhood. After that, each refined literal embedding is translated (by the means of a multilayer perceptron) into a logit probability, all of which are averaged to produce a final satisfiability prediction.

The success story of neurosat is an invitation to assess whether similar performance can be obtained for other $\mathcal{NP}$-Hard problems. Graph problems are suitable candidates, as they fit nicely into the relational structure of GNNs. The Traveling Salesperson Problem (TSP), assigned with finding the shortest length Hamiltonian path in a graph (that which visits each vertex exactly once), is a natural choice given its last longing relevance to computer science. Studying GNN models for TSP has another advantage: it allows us to assess whether GNNs can tackle problems where the connections

*Equal conttribution

between elements are labeled with numerical information – in our case, edges carrying differing weights.

3 Our Model

Typical GNN architectures link adjacent vertices in the graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ which represents the problem instance by opening a direct communication channel between their corresponding embeddings, in such a way that one can send messages to the other. Concretely, if we picture the collection of all $|\mathcal{V}|$ vertex embeddings at time t as a matrix $\mathbf{V}^t \in \mathbb{R}^{|\mathcal{V}| \times d}$, the process of filtering the incoming messages for each node corresponds to performing a matrix multiplication with the graph's adjacency matrix, $\mathbf{M}$. Then the process of refining all embeddings at once can be expressed by Equation 1, where $\mathbf{V}_u$ is a recurrent unit and $\mathbf{V}_h^t \in \mathbb{R}^{|\mathcal{V}| \times d}$ is a matrix in which each line $\mathbf{V}_h^t[i]$ contains the hidden state, at time t, corresponding to the i-th vertex.

$$\mathbf{V}_h^{t+1}, \mathbf{V}^{t+1} \leftarrow V_u(\mathbf{V}_h^t, \mathbf{M} \times \mathbf{V}^t) \tag{1}$$

However one can see that this approach fails when connections are augmented with some numerical information such as edge weights or capacities. To feed this information into the network, we propose a new architecture which elevates edges to a "node" status in the context of the GNN. Concretely, we will assign embeddings to both vertices and edges, and link the embedding of each edge $e = (v_1, v_2)$ with the embeddings of the two vertices it connects (v_1 and v_2). The importance of having edge embeddings is that now we can feed them their "labels" (i.e. edge weights, edge capacities, etc.) at each message-passing timestep.

In this context, our neural algorithm includes t_{max} message-passing iterations, each of which is composed of two steps: a vertices refinement step and an edges refinement step (Lines 14 and 16 of Algorithm 1 respectively). Each vertex v has its embedding refined upon feeding the recurrent unit V_u with 1) the combined messages received from all the edges $(v, v', w) \in \mathcal{E}$ in which v appears as a source vertex and 2) the combined messages received from all the edges $(v', v, w) \in \mathcal{E}$ in which v appears as a target vertex. Each edge $e = (v_1, v_2, w)$ has its embedding refined upon feeding the recurrent unit E_u with 1) one message received from v_1 2) one message received from v_2 and 3) a scalar message containing the numerical value of w.

Note that because vertex embeddings are updated given two distinct element-wise sums of messages $\in \mathbb{R}^d$, the recurrent unit V_u receives inputs $\in \mathbb{R}^{2d}$. Analogously, the recurrent unit E_u receives inputs $\in \mathbb{R}^{2d+1}$ because of the extra scalar. These recurrent units are main trainable components of our model, learning to do the most important work: refine embeddings by compiling relational data.

Some attention must be given to the components E_{msg} and V_{msg}, which are also trainable. Because the learned projections from vertices and edges to d-dimensional space is expected to be very different, we include two multilayer perceptrons (MLPs) assigned with translating edge embeddings to vertex embeddings (E_{msg}) and vertex embeddings to edge embeddings (V_{msg}).

The final trainable component of our model is E_{vote}, a MLP assigned with translating edge embeddings into logit probabilities, which we interpret as how certain the network is that each edge belongs to the optimal TSP route.

Algorithm 1 Graph Neural Network TSP Solver

procedure GNN-TSP($\mathcal{G} = (\mathcal{V}, \mathcal{E})$)
    Enumerate the edges of the graph, assigning an unique index $\mathcal{E}_{idx}[(v_1, v_2, w)]$ for each edge $(v_1, v_2, w) \in \mathcal{E}$

    // Compute adj. matrix from edges to source vertices
    $\mathbf{M}_{\mathbf{EV_{src}}}[e, v] \leftarrow 1$ if $\exists v'(\mathcal{E}_{idx}[(v, v')] = e) \mid \forall e \in \{1 \dots |\mathcal{E}|\}, \forall v \in \{1 \dots |\mathcal{V}|\}$
    // Compute adj. matrix from edges to target vertices
    $\mathbf{M}_{\mathbf{EV_{tgt}}}[e, v] \leftarrow 1$ if $\exists v'(\mathcal{E}_{idx}[(v', v)] = e) \mid \forall e \in \{1 \dots |\mathcal{E}|\}, \forall v \in \{1 \dots |\mathcal{V}|\}$
    // Compute edge weights vector
    $\mathbf{W}[\mathcal{E}_{idx}[(v_1, v_2, w)]] \leftarrow w \mid \forall (v_1, v_2, w) \in \mathcal{E}$

    // Run t_{max} message-passing iterations
    for $t = 1 \dots t_{max}$ **do**
        // Refine each vertex embedding with messages received from all the edges in which it appears
        // either as a source or a target vertex
        $\mathbf{V}_h^{t+1}, \mathbf{V}^{t+1} \leftarrow V_u(\mathbf{V}_h^t, \mathbf{M}_{\mathbf{EV_{src}}} \times E_{msg}(\mathbf{E}^t), \mathbf{M}_{\mathbf{EV_{tgt}}} \times E_{msg}(\mathbf{E}^t))$
        // Refine each edge embedding with (two) messages received from its source and its target vertex
        $\mathbf{E}_h^{t+1}, \mathbf{E}^{t+1} \leftarrow E_u(\mathbf{E}_h^t, \mathbf{M}_{\mathbf{EV_{src}}}{}^T \times V_{msg}(\mathbf{V}^t), \mathbf{M}_{\mathbf{EV_{tgt}}}{}^T \times V_{msg}(\mathbf{V}^t), \mathbf{W})$

    // Translate edge embeddings into logit probabilities
    $E_{logits} \leftarrow E_{vote}(E^{t_{max}})$
    // Convert logit probabilities into probabilities
    $E_{prob} \leftarrow sigmoid(E_{logits})$
    // Extract route
    $Route \leftarrow \{(v_1, v_2, w) \mid (v_1, v_2, w) \in \mathcal{E}, E_{prob}[\mathcal{E}_{idx}[(v_1, v_2, w)]] > 0.5\}$

3.1 Training

The most natural way to train this model is to perform stochastic gradient descent on the binary cross entropy loss between the computed edge probabilities and an "edges mask" with ones in the indices corresponding to edges in the solution and zeros elsewhere. This training, of course, has the disadvantage of requiring a rather large amount of labels ($|\mathcal{E}|$). We can also anticipate a problem: if the training instances were to allow multiple optimal TSP routes, the network would not be able to choose the same route we compute, as we are required to break ties through some arbitrarily chosen criterion (such as lexicographic ordering of vertices).

4 Experimental Setup

5 Results and Analysis

6 Discussion

References

Daniel Selsam, Matthew Lamm, Benedikt Bunz, Percy Liang, Leonardo de Moura, and David L Dill. Learning a sat solver from single-bit supervision. *arXiv preprint arXiv:1802.03685*, 2018.