Learning Graph Neural-Networks for Probabilistic Semantic Maps

M.Tech ICT(ML)
Dhirubhai Ambani Institute of Information and Communication Technology
Jigar Shekhat-202211004@daiict.ac.in
Supervisor: Prof. Tapas Kumar Maiti

***Abstract*—We examine Graph-Structured Sum-Product Networks(GraphSPNs), a probabilistic method for structured prediction in circumstances where latent variable dependencies are represented by arbitrary, dynamic graphs. While many structured prediction methods impose strict limits on the relationships between inferred variables, many real-world Problems can only be described using complicated graph structures of various sizes, which are frequently tainted with noise when derived from actual data.**
Here, we concentrate on a certain robotics-related problem. We show how noisy topological relations found by a robot navigating big office areas can be used to support an inference about semantic, conceptual place descriptions. We implement a graph neural network to address these problems. Graph neural networks (GNNs) are neural models that use message transmission between graph nodes to reflect the dependency of graphs. Variants of GNNs include recurrent graph networks (GRN), graph attention networks (GAT), and graph convolutional networks (GCN) have recently shown ground-breaking performances on a variety of deep learning tasks.

***Index Terms*—SPN, GraphSPN, GNN, GCN**

I. Introduction

A representation of spatial information, a framework that structures the understanding of the environment, is essential for a mobile robot to stay aware of. Robots that move around have access to both local and global information. In order to combine knowledge across spatial scales and levels of abstraction, a representation should use spatial relations that have been identified. An established framework for describing spatial relationships between nearby locations, topological maps provide to facilitate the anchoring of high-level conceptual data and provide simple access for a planning algorithm. Because of this, several approaches to semantic mapping use topological graphs as part of their representation and link topological nodes to semantic location attributes .
Such frameworks frequently use structured prediction algorithms to integrate the gathered spatial knowledge, clear up ambiguities, and generate predictions. Unfortunately, the relations a robot investigating a real-world environment discover are frequently complex and noisy, leading to challenging inference issues. The number of nodes and relations in topological maps varies depending on the environment and expands as the robot explores the environment around it.
However, to achieve tractability, many techniques for structured prediction establish severe limitations on the interactions between inferred variables and ask for a fixed number of latent output variables connected through an underlying global structure. They must compromise on structure complexity, include past structural knowledge, or make firm commitments regarding the values of semantic features,

II. Background

In this section, We provide the background information for this project in detail. First, We describe in detail the properties of GraphSPN and GCN, how inferences are performed, and how the parameters and structure of the network can be learned. Next, the COLD dataset has the Deep Affordance Spatial Hierarchy (DASH), is the spatial knowledge representation that we use to enable environment understanding for mobile robots. Finally, I describe indoor topological maps in general, and how topological maps are generated in DASH.

III. Technologies

A. **SPN**

SPN, proposed by Poon and Domingos, is a new class of probabilistic graphical models with built-in properties that allow tractable inference, a major advantage over traditional graphical models such as Bayesian networks. The idea is built upon Darwiche's work on network polynomial and differentials in arithmetic circuit representation of the polynomial [5]. Here, we provide the definition of SPN and several of its key properties[1].
Definition (SPN) Let $X = \{X_1, \cdots X_n\}$ be a set of variables. A Sum-Product Network(SPN) defined over X is a rooted directed acyclic graph. The leaves are indicators. The internal nodes are sum nodes and product nodes. Each edge (i, j) from sum node i has a nonnegative weight w_ij . The value of a sum node is $\sum_{j \in Ch(i)}, w_{ij} v_j$, where $Ch(i)$ is s the children of i. The value of a product node is the product of the values of its children. The value of an SPN is the value of its root. We use S to denote an SPN as a function of the indicator variables (i.e. the leaves). Let x be an instantiation of the indicator variables, a full state. Let e be an evidence (partial instantiation). For a given node i, we use S_i to denote the sub-SPN rooted at i. Also, we use x_p^a to mean $[X_p = a]$ is true, and use $x_p^- a$ to mean the negation, for simplicity. We define the following properties of SPN.
Figure 1: SPN implementing a naive Bayes mixture model (three components, two variables).

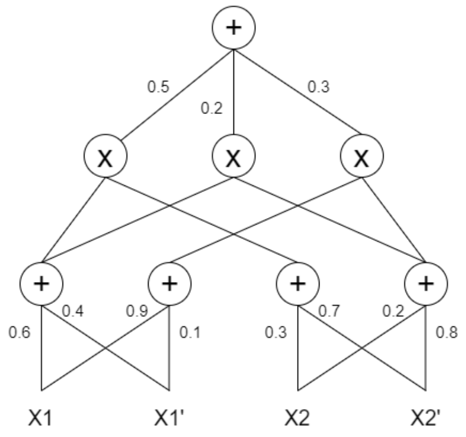


Fig. 1: Example of SPNs

Fig. 1: An simple SPN for a naive Bayes mixture model P(X1, X2), with three components over two binary variables. The bottom layer consists of indicators for X1 and X2. Weighted sum nodes, with weights attached to inputs, are marked with +, while product nodes are marked with ×. we will assume (without loss of generality) that sums and products are arranged in alternating layers, i.e., all children of a sum are products or leaves, and vice-versa.

For example, for the SPN in Figure 1, $S(x_1, x_2, \overline{x}_1, \overline{x}_2) = 0.5(0.6x_1 + 0.4\overline{x}_1)(0.3x_2 + 0.7\overline{x}_2) + 0.2(0.6x_1 + 0.4\overline{x}_1)(0.2x_2 + 0.8\overline{x}_2) + 0.3(0.9x_1 + 0.1\overline{x}_1)(0.2x_2 + 0.8\overline{x}_2)$ The network polynomial $S(x) = (0.5 \times 0.6 \times 0.3 + 0.2 \times 0.6 \times 0.2 + 0.3 \times 0.9 \times 0.2)x_1x_2 + \cdots$ If a complete state x is $X_1 = 1$, $X_2 = 0$, then $S(x) = S(1, 0, 0, 1)$ If The evidence e is $X_1 = 1$, then $S(e) = S(1, 1, 0, 1)$. Finally, $S(*) = S(1, 1, 1, 1)$.

B. Topological Graphs

GraphSPNs are applicable to arbitrary graphs. However, here our dataset is specifically a topological graph built by a mobile robot exploring a large-scale environment. The primary purpose of our topological graph is to support the behavior of the robot. As a result, nodes in the graph represent places the robot can visit and the edges represent navigability. The graph nodes are associated with latent variables representing semantics and the edges can be seen as spatial relations forming a global semantic map. Local evidence about the semantics of a place might be available and we assume that such evidence is inherently uncertain and noisy. Additional nodes in the graph are created to represent exploration frontiers, possible places the robot has not yet visited, but can navigate to. We call such nodes placeholders and assume that the robot has not yet obtained any evidence about their semantics. The topological graph is assembled incrementally based on a dynamically expanding 2D occupancy map. The 2D map is built from laser range data captured by the robot using a grid mapping approach based on Rao-Blackwellized particle filters. Placeholders are added at neighboring, reachable, but unexplored locations and connected to existing places. Then, once the robot performs an exploration action, a placeholder is converted into a place and local evidence captured by the robot about the semantic place category is anchored to the graph node [4]. An example of such semantic-topological the map is shown in Fig. 2

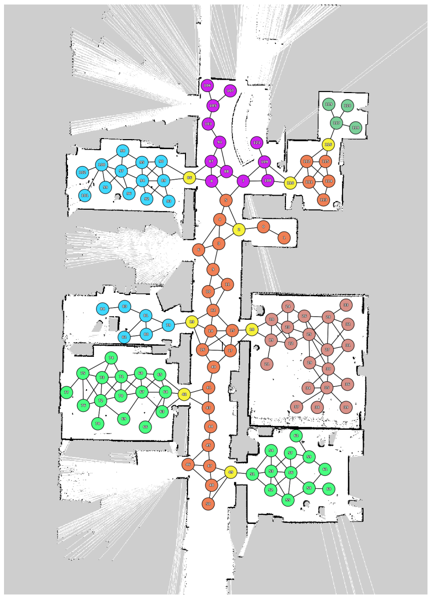


Fig. 2: Topological graph (Stockholm-floor4)

C. GraphSPNs

GraphSPNs learn a template model over arbitrary graph-structured data, with local evidence Xi and latent variables $Y_i = \{Y_{i1}, \cdots, Y_{iM}\}$ for each graph node or edge i, with dependencies between the latent variables expressed in terms of the graph structure. Then, an instance of GraphSPN distribution $P(X_1, Y_1, \cdots, X_N, Y_N)$ is assembled for a specific graph to perform inference. To this end, we define a set S of template sub-graphs, and associate each template sub-graph S $\in \mathcal{S}$ with a separate template SPN modeling the distribution over variables X_i and Y_i corresponding to the nodes and edges of the template sub-graph. The structure and parameters of each template SPN can be learned directly from data obtained by decomposing training graphs into sub-graphs corresponding to $\mathcal{S}$.

Given a set of trained template SPNs, and a specific graph to be modeled, an instance of GraphSPN is assembled as illustrated in Fig. 2. First, the graph is decomposed multiple times, each time differently, into sub-graphs using sub-graph templates S in descending order of the template size (i.e. more complex templates have priority). The subgraphs should not overlap in each decomposition and the corresponding template SPNs should cover all variables X_i, Y_i in the model. This condition guarantees the completeness and decomposability resulting in a valid instance GraphSPN. For each decomposition and each sub-graph, we instantiate the corresponding template SPN resulting in multiple SPNs sharing weights and structure. The instantiations for a single graph decomposition is combined with a product node and the product nodes for all decompositions become children of a root sum node realizing

the complete mixture model.
In order to incorporate the latent variables Y_{ij}, we include an intermediate layer of product nodes into the template SPNs. Each such product node combines arbitrary distribution of ${D_i}_j^k(X_i)$ with an indicator $\lambda_{Y_{ij}} = c_j^k$ for a specific value c_j^k of Y_{ij}. The template SPN is built on top of the product nodes and can be learned from data and the distributions ${D_i}_j^k(X_i)$ can be arbitrary, potentially also realized with an SPN with a data-driven structure[5].
In our experiments, we assumed only one latent variable (semantic place category) Y_i per graph node i, with $Val(Y_i) = \{c^1, \cdots, c^L\}$, and we defined $D_i^k(X_i)$ for a single hypothetical binary observation x_i, which we assumed to be observed:

$$D_i^k(X_i) = \begin{cases} \alpha_i^k & \text{if } X_i = x_i \\ 1 - \alpha_i^k & \text{if } X_i = \overline{x}_i \end{cases}$$

Such simplification allows us to thoroughly evaluate GraphSPNs for the problem of learning topological semantic maps by directly simulating hypothetical evidence about the semantic category of varying uncertainty and under various noise conditions. Furthermore, it allows us to compare GraphSPNs with Markov Random Fields using the same α_i^k as the value of local potentials, i.e. $\phi_i(Y_i = c^k) = \alpha_i^k$. The proposed approach naturally extends to the case where a more complex distribution is used to model semantic place categories based on robot observations.

D. GraphNNs

Graph Neural Networks (GNNs) are a type of neural network that are designed to process graph-structured data. They are used to perform various tasks such as node classification, graph classification, and link prediction. GNNs have become popular due to their ability to handle complex relationships and dependencies in graph data.
GNNs work by using a message-passing algorithm to update the hidden states of nodes in a graph based on the hidden states of their neighbors. The algorithm involves passing messages between nodes and aggregating the messages to update the hidden states. The process is repeated for multiple iterations until the hidden states converge to a stable state. GNNs can be designed with different types of message-passing functions such as convolutional or attention-based functions.
There are different architectures of GNNs such as Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), GraphSAGE, and Graph Isomorphism Networks (GINs). Each of these architectures has its own strengths and weaknesses, and the choice of architecture depends on the specific task and the characteristics of the graph data.
GCNs are one of the most popular GNN architectures and are designed to perform convolution operations on graphs[2].

E. GCNs

Graph convolutional networks (GCNs) (Kipf & Welling, 2017) generalize convolutional neural networks (CNNs) (LeCun et al., 1995) to graph-structured data. To learn the graph representations, the "graph convolution" operation applies the same linear transformation to all the neighbors of a node followed by a nonlinear activation function. GCNs have become one of the most popular GNN architectures due to their simplicity, effectiveness, and scalability.
GCNs work by propagating information from a node's neighbors to update its own representation. This is achieved through a convolution operation in the graph domain, where the filter is applied to the node and its neighbors. The output of the convolution operation is a new feature representation of the node, which incorporates information from its local neighborhood.
The mathematical formula for a single layer of GCN can be written as follows:

$$H^{(l+1)} = \sigma(\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}H^{(l)}W^{(l)})$$

where $H^{(l)}$ is the hidden state matrix of the nodes in layer l, $\hat{A}$ is the adjacency matrix of the graph with added self-loops, $\hat{D}$ is the diagonal node degree matrix of $\hat{A}$, $W^{(l)}$ is the weight matrix of layer l, and σ is the activation function.
The $\hat{D}^{-\frac{1}{2}}\hat{A}\hat{D}^{-\frac{1}{2}}$ term is known as the normalized adjacency matrix, which ensures that the weight of the connections between nodes is scaled based on their degree. The output of the convolution operation is then passed through the activation function to produce the updated hidden state matrix $H^{(l+1)}$. Multiple layers of GCN can be stacked to learn increasingly complex representations of the nodes in the graph. The output of the final layer can be used for various downstream tasks such as node classification and graph classification[3].

IV. Dataset

Cognitive rObot Localization Database (COLD), a large database of localization, sensory information and local environment representations (place scans), as well as topological maps, completely annotated with semantic place categories. There are 100 sequences in total, collected by a mobile robot in three buildings at three different locations: Stockholm, SE (40 seqs), Freiburg, DE (26 seqs), and Saarbrucken, DE (34 seqs).
The **COLD-TopoMaps Dataset** consists of topological maps collected as the robot explores the environment. Each topological map is an undirected graph (also called "topological graph"), where vertices are places that the robot could access, and edges indicate navigability. Each place is annotated by a semantic place category, such as corridor, kitchen, or doorway.
nodes.dat A CSV file where each row is the data for a certain

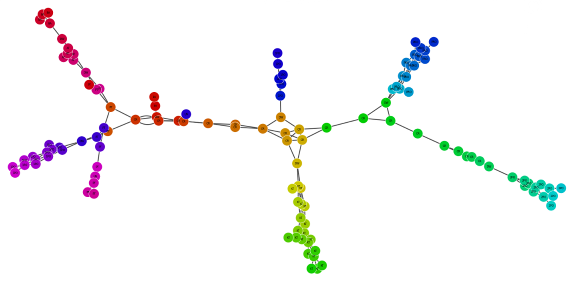
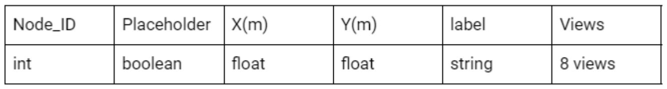


Fig. 3: **Single Floor Graph**

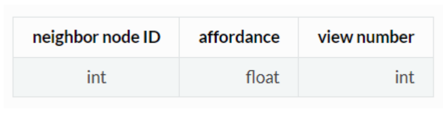
node on the topological map. The column values (and types) are below:



Node_ID	Placeholder	X(m)	Y(m)	label	Views
int	boolean	float	float	string	8 views

Fig. 4: **Data Attributes**

Each place has 8 views (as shown in the figure below). For every edge connected to the node through a view, there are three entries of data in the row for this node:



neighbor node ID	affordance	view number
int	float	int

Fig. 5: **3 Entries for Each view**

If the neighbor node ID is -1 for a view number, then this view has no connected edge. If we think of a node as a disk, then the view numbers correspond to regions as annotated in the following illustration.

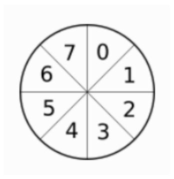


Fig. 6: **8 views**

V. Working Steps

The project described above involves node classification in the first step of semantic mapping. Here are the general steps that we followed to create this project:

1) First, the COLD (Cognitive rObot Localization Database) dataset was used to launch the project. Working with this dataset proved challenging. Topological maps are acquired and saved in the COLD-TopoMaps Dataset while the robot explores its surroundings. Each map is an undirected graph, with vertices serving as potential access places for the robot and edges denoting navigability. Each place is annotated with a semantic place category.
2) We attempted to create GraphSPN while working with the COLD dataset, but as a result, we encountered several library dependency issues, which we later identified as deprecated versions of the same.
3) In the following phase, we choose to implement GNN. Graph Neural Networks (GNNs) are built to work with graph data, representing a network of connected nodes and edges.
4) An adjacency matrix and a feature matrix are used to represent a graph as the input data for the GCN node classification algorithm. While the feature matrix comprises feature vectors for each node in the graph, the adjacency matrix encodes the connections between the nodes in the network. subsequently turned the dataset into a feature matrix and an adjacency matrix for our model's input.
5) The first step in GCN is to perform a graph convolution operation. This operation involves multiplying the feature matrix with the adjacency matrix to capture the relationship between nodes in the graph.
6) After performing the graph convolution operation, an activation function is applied to introduce non-linearity to the model. The most commonly used activation function is the Rectified Linear Unit (ReLU) function.
7) The output of the final layer of GCN is then fed into a softmax function to obtain a probability distribution over the different classes of nodes.
8) The GCN model is trained using labeled data to minimize the cross-entropy loss between the predicted probability distribution and the true class labels.
9) The trained GCN model can then be used to predict the class labels of unseen nodes in the graph.

VI. Results

In this experiment, we compared the performance of two different models for node classification: a graph convolutional network (GCN) and an artificial neural network (ANN). We used the COLD-topo map dataset with 22 classes and trained both models to classify nodes into their respective classes.
After training and testing the models, we found that the GCN achieved an accuracy of **39.5%**, while the ANN achieved an accuracy of **30%**. This suggests that the GCN model was better suited to this particular task of node classification, and was able to capture the relationships between nodes in the graph more effectively than the ANN.
One possible reason for the superior performance of the GCN is that it was able to leverage the structure of the graph to improve its predictions. The GCN uses the adjacency matrix of the graph to propagate information between neighboring nodes, allowing it to capture higher-order interactions between nodes. In contrast, the ANN treats each node as an independent data point and does not explicitly model the relationships between nodes.

VII. Conclusion

On the COLD dataset, which has 22 distinct labels, we examined the effectiveness of graph convolutional networks (GCNs) in this study to classify nodes. According to our findings, the GCN model classified the nodes into their respective labels with an accuracy of 39.5%.
These findings indicate that GCNs are a potential method for node classification tasks in challenging datasets like COLD, where typical machine learning models have difficulty capturing connections between nodes.
Future research will need to solve some of the GCN model's remaining limitations. For instance, the selection of hyperparameters, such as the number of layers and the size of the filters, might have an impact on the GCN's performance. The

GCN model is also computationally demanding and can be challenging to train on huge datasets.
In conclusion, our work underlines the need for more research to improve the performance of GCNs and solve their weaknesses while also demonstrating the promise of GCNs for node classification tasks in challenging datasets like COLD.

VIII. Limitations and future scope

The project has a lot of potential for future development. Here are some possible areas for improvement:

1) Memory requirement: Memory use in the current configuration of full-batch gradient descent increases linearly with dataset size. We have demonstrated that training on the CPU may still be an effective choice for huge graphs that do not fit in GPU memory.
2) Real-time data: The COLD dataset we are using right now has only udirected graphs. As a result, we will begin working on a real-time dataset that will provide us with a wide range.

References

[1] Raquel Sanchez-Cauce, Iago París, and Francisco Javier Díez. Sum-product networks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3821–3839, 2021.

[2] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 32(1):4–24, 2020.

[3] Si Zhang, Hanghang Tong, Jiejun Xu, and Ross Maciejewski. Graph convolutional networks: a comprehensive review. *Computational Social Networks*, 6(1):1–23, 2019.

[4] Kaiyu Zheng and Andrzej Pronobis. From pixels to buildings: End-to-end probabilistic deep networks for large-scale semantic mapping. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3511–3518. IEEE, 2019.

[5] Kaiyu Zheng, Andrzej Pronobis, and Rajesh Rao. Learning graph-structured sum-product networks for probabilistic semantic maps. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.