Learning Search Heuristics by Graph Convolutional Networks

Pedro Ballester* and **Rodrigo Barros** and **Felipe Meneguzzi**
Pontifcia Universidade Catlica do Rio Grande do Sul
pedro.ballester@acad.pucrs.br

Abstract

Automated Planning is highly dependent on the trade-off between high quality heuristics and computational time for computing them. Out-of-the-shelf and computationally inexpensive heuristics tend to work well on several domains but can be uninformative or expensive to compute on more peculiar scenarios. By exploiting new advances in Convolutional Neural Networks, such as Graph Convolutional Networks (GCN), we aim at learning heuristics focused on the domain by training on simple instances of the problem and thus generalizing for more demanding ones.

Introduction

Automated Planning is a widespread field and its applications range from general game playing (Genesereth, Love, and Pell 2005) to space exploration (Norris et al. 2005). A good planner can eventually provide plans, if one exists, in non-critical time. Among the most important aspects of a good planner is the correct choice of a heuristic.

There are several heuristics that work well in multiple application domains and depending on their characteristics they will always lead to a correct, commonly suboptimal, plan. However, in some specific scenarios, some off-the-shelf heuristics work poorly both in performance and plan quality, and urge the need to design domain specific ones.

Handcrafting a heuristic relies on both domain knowledge and planning knowledge, which leads to very limited possibility of design for complex scenarios. One alternative for such issue is, instead of depending on human knowledge on a task for designing precise heuristics, one can use a data-driven approach and develop them using Machine Learning algorithms.

Considering the complex data structures available for planning tasks, a possible path for learning in this situation is using Deep Learning. Deep Learning has been popularized by its applications on unstructured data such as image and text. More recently has also been applied to complex structures such as graphs, with promising results (Kipf and Welling 2016).

Released in 2012 with AlexNet (Krizhevsky, Sutskever, and Hinton 2012), Convolutional Neural Networks (Deep Learning most famous neural network design and known as CNNs) set record results in ImageNet classification task (Deng et al. 2009; Russakovsky et al. 2015). Several other architectures have been proposed after, such as GoogleNet (Szegedy et al. 2015) with inception layers and ResNet (He et al. 2016) with residual layers. Each one of them add design choices that further enhance learning capabilities of Convolutional Neural Networks.

To explore other types of data, architectures that aimed at graphs, instead of images were also proposed (Kipf and Welling 2016; Defferrard, Bresson, and Vandergheynst 2016), called Graph Convolutional Networks (GCNs). This would allow Convolutional Neural Networks to capture neighbourhood information and properly handle this kind of data.

To surpass the problem of domain specific heuristic design for complex scenarios, this work proposes the use of GCNs for learning heuristics. We also want to address transferability and how to classify such heuristics according to planning literature.

Related Work

Long before Deep Learning, traditional Machine Learning was already being used for Planning tasks. There are studies with a thorough review of such methods and addressing them is out of the scope of this work (Jiménez et al. 2012).

Some works deploy Deep Learning in an offline matter to choose a search heuristic that best suites the task (Sigurdson and Bulitko 2017; Loreggia et al. 2016).

To the best of our knowledge this work is the first to deploy Deep Learning to find new search heuristics tailored for the problem. We will improve the related work section for the next iteration of this assignment.

Graph Convolutional Networks

Graph Convolutional Networks (GCN) (Kipf and Welling 2016) are Convolutional Neural Networks that work directly on graphs and are able to take advantage on specificities of how graphs are encoded. Graphs can be dense or sparse and encode relationship between different nodes in a different perspective as traditional Convolutional Neural Networks expect (e.g. for images and text). Whereas traditional CNNs would look for local relationship by observing proximity in the input matrix, GCNs aim at understanding the

Copyright © 2018, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

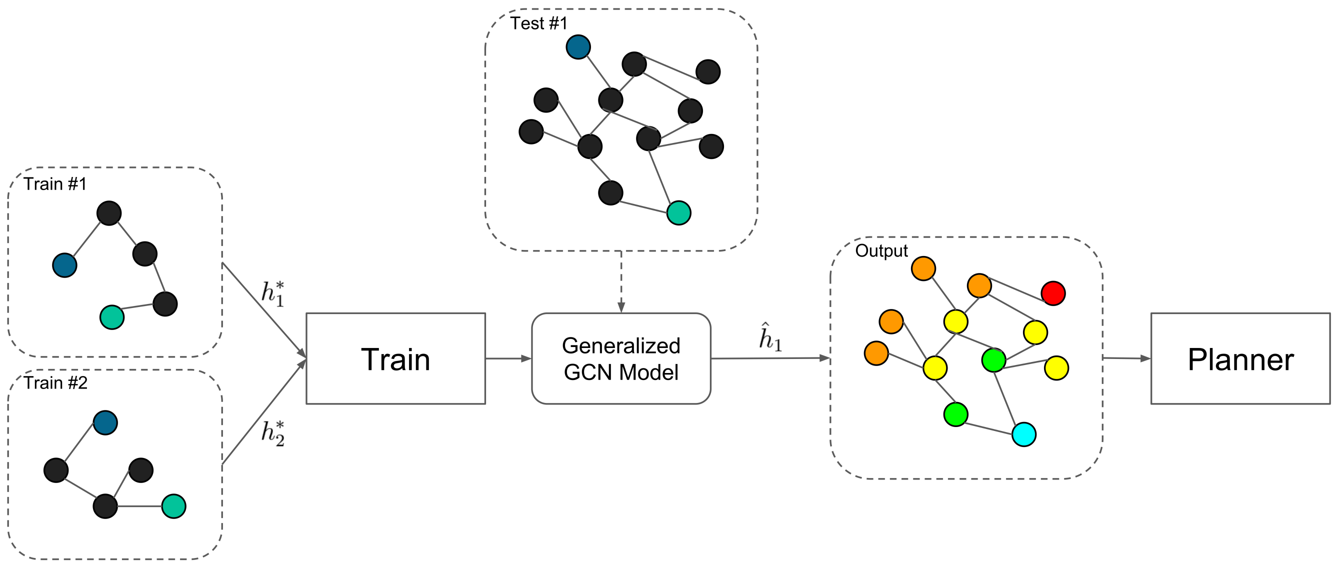

Figure 1: Training and testing protocol for learning heuristics with examples from a task. Both Train #1 and #2 are non-consuming examples from the task and Test #1 is a hard to compute example that is forwarded to the network to extract heuristic values for each state expanded. The train input h_1^* and h_2^* represent the perfect heuristic for train instance 1 and 2. The output $\hat{h}_1$ is the output heuristic value for each node of test instance 1. Best viewed in color.

node neighbourhood instead. This is a key difference considering that most common graph encoding do not group neighbourhood by the node i, j position distance in the matrix.

Formally defining, a GCN is a Convolutional Network that takes as input a graph, in this case represented as a NxN adjancency matrix being N the number of nodes and a NxF matrix, being F the size of the feature encoding for each node. The output from the network is a node-wise value NxO being O the size of output for each node.

Graph Convolutional Networks rely on spectral graph theory, thus designing spectral convolutions. Simplifications of such operations speed up training and allow for non-prohibitive computation time for large graphs.

Method

To proper use Graph Convolutional Networks for learning heuristics, we must carefully evaluate several design options. There are also important differences on evaluating during training and inference time.

The method is composed of a GCN with each node representing a state from the search tree. Each input graph is a subset of the whole search tree, and each node is simultaneously evaluated. For each node, the network will predict a heuristic value, thus being able to be incorporated with traditional search methods, such as A-star –CITE–.

The proposed method can be seen in Figure 1.

Design and encoding

The network is composed of two Graph Convolutional layers with ReLU and Dropout (Srivastava et al. 2014). The network architecture is more thoroughly described in Table 1.

Table 1: Proposed GCN Architecture

Layer	Input	Output
Graph Convolution	$2\|F\|$	60
ReLU	-	-
Dropout	-	-
Graph Convolution	60	1
ReLU	-	-

Each node from the input graph is composed of F features that represent the state. As a proof of concept, our features do not contain any domain expert knowledge, being just the combination of every possible predicate. A binary value codes a single combination of predicate operators and group of objects, thus the final vector F is a set of binary values that fully represent the state. For every predicate operator with arity k, there are o^k generated binary values concatenated to compound the final feature vector F, being o the number of objects. Elucidating, for an instance in the Blocks World Domain with 2 objects A and B, the final feature vector consists of a vector F of size 9. In this case 2^1 for $CLEAR$, 2^1 for $ONTABLE$, 2^2 for ON and, finally 2^0 for $HANDEMPTY$.

The GCN framework, however, demands a fixed-length feature vector. Thus, as a current limitation, we must define the maximum number of objects for the domain to create a fixed-length F vector. The total number of nodes in a graph, however, does not need to be fixed, and thus can vary depending on the branching factor of each node and its neighbors.

In order to encode goal information for the network, the goal state is encoded in the same way as other nodes and its feature vector is concatenated with every input node for the network. Thus, the input shape for the the first layer is $2|F|$. We are also investigating whether instead of concatenating the goal features, a simple average between the goal and the node vectors is not enough to encode the goal information.

Training

The first step for training the network is to identify simple non-consuming domain instances of a given task. We define s starting points for the domain instances to generate several training instances for every domain instance d in order to improve generalization. The number of starting points s is defined as a hyperparameter for the framework. The input for the network, however, is a subgraph of the whole search graph. We thus must expand the knowledge frontier for p levels while annotating the adjacency for every expanded node to generate the final input graph. We set p as an hyperparameter and it must remain the same during training and inference time. Every subgraph generated is an individual and independent instance for the network. The final dataset G is composed of $d * s$ graphs.

In order to generate the ground-truth we depend on finding the perfect heuristic value for every node in the dataset. Thus, considering an optimal plan for a node n, $plan(n)$:

$$h^*_{n,g} = |plan(n)|, \forall n \in g, \forall g \in G \tag{1}$$

We then can define a loss function. We define the loss as a regularized Mean Squared Error over the node predictions for a graph:

$$L_h(g) = \sum_{i=1}^{N} ||\hat{h}_{i,g} - h^*_{i,g}||_2 + \lambda||\Theta||_2 \tag{2}$$

where $\hat{h}_i$ is the computed heuristic value by the neural network for a node in the graph, h^*_i is the perfect heuristic outcome for a node, and N is the total number of nodes. λ and Θ represent the $l2$ regularization hyperparameter and model weights respectively.

A possible limitation of this approach is finding non-consuming instances of a given task, as not all problems can be trivially simplified as to find easy examples for training the network. One could potentially train at large examples, but this would drastically impact the performance of the training as generating full graphs for hard examples alongside computing the perfect heuristic can be very demanding.

Inference

The method behaves differently than most heuristics during inference time. As described, the input for the neural network is a graph, not a node. For that reason, during inference time, neighbour nodes from the one being currently evaluated must be expanded. Evaluating node n, thus consists of expanding its frontier for p levels. The adjacency matrix is computed the same way as during training. Both n and its neighbors expanded nodes must be encoded following the previously described method. The graph becomes a NxF feature matrix alongisde a NxN adjacency matrix representing the N expanded nodes.

The resulting graph is then fed to the network obtaining $\hat{h}$. Although this is seemly less straight-forward than purely evaluating $\hat{h}_{n,.}$, the GCN framework can compute $\hat{h}$ for every node on the input graph simultaneously, thus decreasing the total amount of computation necessary.

Adversarial Training

A common problem of training models with mean squared error is that the model frequently learns a somewhat mean version of the distribution. This means that the values are usually within a small range from the mean and sometimes ignore the task completely.

For that purpose, we inserted an optional adversarial-based regularization. The problem thus becomes the optimization of three different objectives:

$$L_h(g) = \sum_{i=1}^{N} \Big[||\hat{h}_{i,g} - h^*_{i,g}||_2 + \lambda_1||\Theta||_2\Big] - \lambda_2 L_D(g) \tag{3}$$

where all variables belonging to 2 remain the same, except for λ that becomes λ_1. The equation adds λ_2 and L_D that represents a hyperparameter to module the discriminator influence and the discriminator loss.

The discriminator loss can be represented as:

$$L_D(g) = -log(D(h_{.,g}) + log(1 - D(\hat{h}_{.,g})) \tag{4}$$

where D is a discriminator model. Both discriminator and GCN are optimized simultaneously once every step. The training follows a traditional zero-sum game and mimics Generative Adversarial Network (GANs) training (Goodfellow et al. 2014).

Results

We will evaluate some domains at multiple aspects. First, how well the heuristic can mimic the perfect heuristic for problems with the same size as the ones presented during training. Second, how well does the model generalizes for larger, more complex scenarios. Other experiments are being designed.

Blocks World

We are currently performing experiments on Blocks World.

Sokoban

We are currently performing experiments on Sokoban.

Discussion and Future Work

To this date we still did not manage to generate a good model for predicting the heuristic. We believe that the lack of hyperparameter tuning and few experiments with different architectures is underestimating the potential of such networks. We should also further evaluate our loss function. We believe that by incorporating the standard deviation of the heuristic during training could perhaps influence positively the training. Also, better understand the impact of adding the adversarial loss objective could lead us to more promising models.

Some other hyperparameters still were not carefully evaluated. The expansion levels p should be taken into consideration as an important hyperparameter to investigate as it changes drastically the number of nodes at each graph. The learning rate could be improved for the task by following some kind of policy such as rampup and rampdown in respect to the number of epochs.

Penalizing dead-ends and increasing the heuristic value for landmarks could potentially improve $\hat{h}$. As a secondary objective, we will try to understand whether this kind of additional information during training can lead to better heuristics or decrease the amount of training data necessary for satisfactory heuristics.

Some relevant information for the generated heuristic is how it is defined in the space of possible heuristics. It is important for engineering and mathematical proofs purpose that one know, for example, whether the generated heuristic is Safe, Goal-Aware and other definitions. We will explore how one can try to infer those characteristics from data and by measuring boundaries for a specific domain. We believe that guaranteeing those definitions for any domain for a given heuristic will be harder due to hard to interpret internal operations of neural networks. However, changing the mean squared error for a more suitable loss function designed specifically for planning could lead to more realistic heuristics.

Memory constraints are a big issue in planning. This approach requires a fair amount of memory by requiring the expansion of the graph for N adjacent states. We will investigate on improvements for domain encoding. Mainly, how we can best represent the domain keeping the necessary information, diminishing memory constraints and processing time while not sacrificing predictive power.

Finally, we will see how the generated heuristic generalizes for new domains. There are several ways we can observe this effect. First, we will run the generated heuristic for new domains and check how it behaves. As a next step, we can compare training the network from scratch and tuning in both unsupervised and supervised manners for a new domain.

Material and Technical Details

Our work will be implemented using the available library PyperPlan[1]. We also found an implementation of GCN in PyTorch [2]. The available implementation has some differences in respect to the original code in TensorFlow, but PyTorch allows for easier prototyping and experimentation, and thus is the code of choice for the assignment due to time constraints.

References

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, 3844–3852.

Deng, J.; Dong, W.; Socher, R.; Li, L.-J.; Li, K.; and Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, 248–255. Ieee.

Genesereth, M.; Love, N.; and Pell, B. 2005. General game playing: Overview of the aaai competition. *AI magazine* 26(2):62.

Goodfellow, I.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; and Bengio, Y. 2014. Generative adversarial nets. In *Advances in neural information processing systems*, 2672–2680.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Jiménez, S.; De la Rosa, T.; Fernández, S.; Fernández, F.; and Borrajo, D. 2012. A review of machine learning for automated planning. *The Knowledge Engineering Review* 27(4):433–467.

Kipf, T. N., and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, 1097–1105.

Loreggia, A.; Malitsky, Y.; Samulowitz, H.; and Saraswat, V. A. 2016. Deep learning for algorithm portfolios. In *AAAI*, 1280–1286.

Norris, J. S.; Powell, M. W.; Vona, M. A.; Backes, P. G.; and Wick, J. V. 2005. Mars exploration rover operations with the science activity planner. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, 4618–4623. IEEE.

Russakovsky, O.; Deng, J.; Su, H.; Krause, J.; Satheesh, S.; Ma, S.; Huang, Z.; Karpathy, A.; Khosla, A.; Bernstein, M.; et al. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision* 115(3):211–252.

Sigurdson, D., and Bulitko, V. 2017. Deep learning for real-time heuristic search algorithm selection.

Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; and Salakhutdinov, R. 2014. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research* 15(1):1929–1958.

Szegedy, C.; Liu, W.; Jia, Y.; Sermanet, P.; Reed, S.; Anguelov, D.; Erhan, D.; Vanhoucke, V.; and Rabinovich, A. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 1–9.

[1]https://bitbucket.org/malte/pyperplan

[2]https://github.com/tkipf/pygcn