Symbolic Inverse-Planning vs Deep-Learning Plan Recognition*

Mariane Maynard and **Thibault Duhamel** and **Froduald Kabanza**
PLANIART Laboratory
Université de Sherbrooke
Sherbrooke, Québec (Canada) J1K 2R1
{mariane.maynard, thibault.duhamel, froduald.kabanza}@usherbrooke.ca

Abstract

Plan recognition is the problem of observing the effects of actions performed by a person or an artificial agent to infer his intent or most likely goals, and predict his course of action. Deep learning has recently been making significant inroads on various pattern recognition problems, except for plan recognition. Most of the current research efforts to plan recognition in the literature are indeed mainly based on symbolic inference algorithms, which use handcrafted models. One of these approaches, intensely studied, is symbolic inverse planning. This paper compares symbolic inverse planning to deep learning on five synthetic benchmarks often used for comparing symbolic plan recognizers. While preliminary, the results show that the deep learning approach achieves better goal-prediction accuracy and timeliness than the symbolic inverse planner in these domains. This experiment is the first step in a broader research effort to investigate deep-learning approaches to plan recognition.

Introduction

Deep learning has recently made amazing inroads in understanding human behaviors. Using binary sensors, deep learning begins to show promising results in the early detection of dementia in the elderly population (Almeida and Azkune 2018). Computer vision learning algorithms can now distinguish between hundreds of human motor actions based on images (Yan et al. 2016), while video analysis algorithms are starting to understand simple human activities involving time and space constraints, such as talking, drumming, skydiving, or walking (Simonyan and Zisserman 2014; Hou et al. 2018). Such applications are able to recognize individual actions, but they are not geared towards recognizing goal-directed sequences of such actions, that is, plans.

Inferring the goal or intention of other agents, and predicting their course of action is known as the plan recognition problem. This problem still presents a tremendous challenge for artificial intelligence (AI) research. To understand the importance of this problem in AI, it is important to remind that in many contexts, human behaviors are indeed driven by strategic action and activity choices. That is, our behavior is often the result of cognitive planning processes, even though we are not often conscious of it (Schmidt, Sridharan, and Goodson 1978; Baker, Saxe, and Tenenbaum 2009). Autonomous vehicles could benefit from plan recognition to better predict the behaviors of pedestrians. Video game AI, security assessment bots and many more applications could benefit from plan recognition capabilities.

Most to date research in plan recognition uses symbolic inference algorithms, which rely on handcrafted models, unlike the action recognition approaches which have largely shifted to using machine learning approaches, deep learning in particular. Yet, handcrafted models have so far proven difficult to use in real-world practical applications. One key reason is simply the difficulty to model abstractions that represent real-world situations with a fidelity sufficient for the inference method used for plan recognition. Defining efficient inference methods is another related challenge.

The success of deep neural networks in learning complex abstractions from multi-dimensional data prompted us to investigate their use for plan recognition. We discuss preliminary results comparing symbolic inverse planning (Ramírez and Geffner 2010; Masters and Sardiña 2017) against selected architectures of deep neural networks on five different synthetic domains commonly used as benchmarks for symbolic plan recognizers. We consider the case of one plan recognizer inferring the goal of an observed agent. The observed agent has no interest in impeding nor assisting the plan recognition process. This is also known as the keyhole plan recognition problem by analogy to a person observing another in a different room, through a keyhole, and trying to infer his goals.

While preliminary, the results show that the neural networks achieve better goal-prediction accuracy and timeliness than the symbolic inverse planner in all the five domains. It seems that even a simple fully connected network is able to learn abstractions underlying sequential decisions conveyed in the observed patterns of a goal-directed agent enough to outperform an inverted planner. Before the experiment, we expected the inverted planner to perform better since it is inherently tailored to generate sequential decisions. This raises interesting avenues of investigation that we discuss in the paper.

The rest of the paper is organized as follows: first, we briefly review the most related work. Second, we give background on plan recognition and deep learning necessary to

---

[*] A subsequent version of this paper was submitted to ICAPS.

follow the methodology used for the experiments. Then, we present our methodology, including the benchmarks used. We finally discuss on the experiment results followed by a conclusion including avenues of future research.

Related Work

A few approaches combine deep learning and symbolic inference in different ways. For example, (Granada et al. 2017) use a deep learning network to recognize individual actions of an actor cooking recipes in a kitchen, and then use a symbolic algorithm, SBR, to infer the goal underlying an observed sequence of actions. This approach requires as input the sequence of actions recognized by the neural network and a handcrafted model (plan library) representing abstractions of potential plans the observed agent could execute. There is no mechanism allowing the handcrafted plan library to adapt its own abstractions to classification errors of the neural network recognizing individual actions.

The approach in (Bisson, Larochelle, and Kabanza 2015) also uses a symbolic inference algorithm, which requires as input a sequence of observations of actions performed by an agent and a plan library. Like the SBR algorithm in the approach above, the plan library represents abstractions of potential plans the observed agent could execute. One component of the plan library representation is a probabilistic model of the choices the observed agent could make when selecting and executing plans from the plan library. A neural network learns this probabilistic model whereas the rest of the plan library is handcrafted.

In both approaches, goal inferences or plan predictions are done by a symbolic inference engine, not a deep neural network. Deep learning is involved only as an auxiliary procedure either to scan individual actions (Granada et al. 2017), or to learn a probabilistic model (Bisson, Larochelle, and Kabanza 2015). In contrast, in the experiments we discuss herein, a neural network does all the inference.

To the best of our knowledge, the approach in (Min et al. 2014) is among the earliest to use a plan recognition pipeline only made of a neural network. They use feed-forward n-gram to learn the player's objective from a sequence of his actions in the CRYSTAL ISLAND game. The follow-up approach in (Min et al. 2016) uses Long Short-Term Memory (LSTM) networks, better suited to learn long short-term patterns in sequences. In both approaches, the features fed to the neural network were handcrafted instead of merely being raw player's events such as mouse clicks and key presses. While these approaches demonstrate interesting results in a specific domain, they do not include a systematic comparison to symbolic approaches.

Thus, although deep learning has started to be investigated for plan recognition in different approaches, we are not aware of any systematic comparison using a complete deep-learning pipeline for plan recognition versus using a symbolic approach or a hybrid approach like those discussed above. In particular, we are not aware of any comparison between neural networks and symbolic inverse planning, which is the experiment specifically discussed herein.

Background

Before discussing the experiment, let us first go through the background necessary to understand the two approaches we compared: the plan recognition problem, deep neural networks, and inverse planning.

The Problem

The plan recognition problem is to infer the goal pursued by an actor from an observed sequence of effects of his actions, and also to extract the plan pursued by the actor from these observations (Schmidt, Sridharan, and Goodson 1978). There is a link between goals, plans and intention. A plan is a sequence of actions achieving a goal, whereas an intention is a commitment to execute a plan. It could be argued that from a plan one can infer the likelihood of goals and vice-versa. Thus, in the AI literature, plan recognition has come to encompass all problems related to understanding goal-oriented behaviors, whether the focus is on goal inference, intention inference, plan prediction, or a combination of those three. The preliminary experiments discussed in this paper concern only inferring the goal.

Definition 1. A plan recognition problem is a tuple $\langle G, o_\pi \rangle$ where G is the set of possible goals of an actor and o_π is a sequence of observations of the effects of the agent's actions.

A solution to a plan recognition problem is a posterior probability distribution across G, $P(G|o_\pi)$. The optimal solution is the one where $P(g|o_\pi)$ is maximal for the true goal $g \in G$ of the observed agent.

A plan recognition problem is also a pattern recognition problem, but not vice-versa. The aim is to predict the true goal of an actor only from a pattern of observations where each observation is the effect of some action executed by the actor. That is, the sequence of observations represents the actor's behaviors as perceived by the plan recognizer.

Deep Learning

In order to follow the methodology used for the experiments, it is useful to have some background on deep neural networks.

Given a set of sequences of observations $\mathcal{O}$ and a set of potential goals G, let us assume that there exists a true recognition function f that maps perfectly each $o_\pi \in \mathcal{O}$ to its true goal $g_{o_\pi} \in G$, that is, $f(o_\pi) = g_{o_\pi}$.

While f is unknown (this is what we want to infer), we assume we have access to a training dataset of paired examples (o_π, g_{o_π}), i.e. we know the true goal g_{o_π} for some $o_\pi \in \mathcal{O}$. A supervised learning algorithm will seek to approximate f with a function f' parametrized by some set of parameters θ that minimizes the number of erred predictions in our dataset of examples. In other words, f' minimizes

$$L = \sum_{n=0}^{N} l(f'(o_\pi^n; \theta), g_{o_\pi^n})$$

where l is a loss function that is 0 when f' predicts accurately, and > 0 otherwise.

A single-layer neural network uses a simple linear transformation of the input using weight and bias parameters followed by a non-linear function in place of f':

$$f'(o_\pi) = \gamma(W o_\pi + b)$$

where W and b are the weight and bias parameters, respectively, and γ is a non-linear function such as sigmoid, hyperbolic tangent (tanh), rectifier linear units (ReLU), softmax, etc. A (deep) neural network is a composition of several of these transformations, usually with a different set of parameters at each layer (Goodfellow, Bengio, and Courville 2016). These parameters are trained in order to converge to the minimum of the objective, usually by gradient descent of the loss function, and updated at each training step by the rule:

$$W \leftarrow W - \alpha \nabla W$$

$$b \leftarrow b - \alpha \nabla b$$

where α is the learning rate, a scalar tuned by hand by the developer. Specialized types of networks, who include variation in the way data is transformed, can prove more useful for plan recognition problems in general, particularly navigation problems. For instance, convolutional neural networks (CNNs) use filters of parameters and the convolution operation to process the input:

$$h_{i',j'} = \gamma(W * o) = \gamma \sum_{i,j=0}^{M,N} W_{i,j} o_{i'-i,j'-j}$$

where * is the convolution operation, M and N is the height and width of the matrix W, also called kernel, o is an observation where spatial structure is preserved (in case of grid-type input for example), γ is the activation function and $h_{i',j'}$ is one of the unit of the generated matrix h, also called feature map. CNNs thus limit greatly the number of trained parameters through parameter sharing or local connectivity and learn efficiently on spatial information such as images or grid-worlds type of data.

Recurrent neural networks (RNNs), as opposed to feed-forward neural networks, process each element of the sequence individually. Furthermore, the previous state of the network is fed back to compute the next state:

$$h_t = \gamma(W o_t + U h_{t-1} + b)$$

where o_t is the t^{th} observation of the sequence o_π, h_{t-1} is the previous state, and W, U and b are matrices and vector of trainable parameters. Parameter sharing for input processing and network state processing favors learning over the content of each element and the order of the elements of the sequence, rather than the position of the element, which would be the case in a typical feed-forward fully connected network. This is particularly useful when treating sequences of observations.

Long Short-Term Memory networks (LSTM) used by (Min et al. 2016) is an improvement of RNNs that allows for better gradient propagation and thus shows better learning results than RNNs on longer sequences.

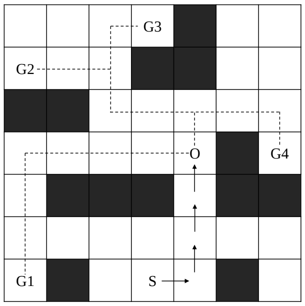


Figure 1: A navigation grid example, where the agent is constrained with obstacles.

Symbolic Inverse Planning

The intuition behind inverse planning is the *principle of rationality*: people tend to act optimally to the best of their knowledge (Baker, Saxe, and Tenenbaum 2009). Under this assumption, the most logical approach to solve a plan recognition problem is to reason from the observed agent's point of view, that is to compute plans for his possible goals. In this line of thinking, inverse planning algorithms compare the costs of the found plans π to infer the posterior probability distribution. Let us look for example at the grid-world depicted in figure 1. From the observation sequence, the agent's logical goal is unlikely G1, since we can find a shorter path from its start state to G1 other than the one it is currently taking. Thus, let us define the likelihood of an observation sequence o_π to reach a goal g to be:

$$P(o_\pi | g) = \frac{e^{-\beta \, costdif(s,g,o_\pi)}}{1 + e^{-\beta \, costdif(s,g,o_\pi)}}$$

where β is a positive constant determining how optimal we assess the observed agent's behaviour to be.

From this, we can derive the posterior probability of the goal using the Bayes rule: $P(g|o_\pi) = \alpha P(o_\pi|g)P(g) \forall g \in G$, where $P(g)$ is the prior probability (often assumed to be uniform) and α is a normalization factor. *costdif* is a function that compares the costs of plans that solves the planning problem for start state s and goal state g. (Ramírez and Geffner 2010) defines it to be:

$$costdif(s, g, o_\pi) = c(s, g, o_\pi) - c(s, g, \neg o_\pi)$$

where $c(s, g, o_\pi)$ is the cost of the optimal plan π_o between s and g that complies with the observations (i.e. all observed actions of o_π are embedded monotonically in the plan) and $c(s, g, \neg o_\pi)$ is the cost of the optimal plan $\pi_{\neg o}$ that does not comply with the observations (o_π is not embedded in π). (Masters and Sardiña 2017) simplifies the formula for navigation domains by noticing first that in most cases, the cost of the plan ignoring the observations naturally does not comply with the observations, and second, that the cost of π_o from the start s up to the corresponding state n of the last observation is constant among all goal plans, and is not a useful information to discriminate the goals. Here is the simplified formula:

$$costdif(s, g, n) = c(n, g) - c(s, g)$$

In this approach, the key is being able to compute plan costs. Symbolic inverse plan recognizers use a symbolic planner to compute plan costs (Ramírez and Geffner 2010). The symbolic planner requires as input models handcrafted by human experts. Such models convey knowledge abstractions about the environment, the possible primitive actions that the observed agent is capable of doing and heuristics. Depending on the symbolic planning algorithm used, heuristics can be handcrafted or extracted automatically by the planner from the models of primitive actions.

In the inverse planning approach as described above, the planner needs to be run twice for each goal considered. Computing a plan, even in the simple case of a deterministic environment under full observability, is NP-Complete (Cooper 1990). This makes it hard to apply inverse planning in situations where an agent needs to infer the intention of others quickly, almost instantaneously, as is the case when people interact with each other.

The approach was described above assuming a computation of optimal costs. As noted by (Ramírez and Geffner 2009), approximate, suboptimal costs can be used instead. The benefit is that, in many cases, planning algorithms that compute suboptimal plans run faster than those computing optimal plans. When a suboptimal planner is used, the computed goal probability distribution is an approximation of the correct distribution. This can be helpful in situations where what is most important is to identify the most likely goals, not necessarily their accurate probabilities.

While an improvement in runtime complexity, even heuristic planners which compute suboptimal plans still take too much time. They cannot support real-time plan recognition in high-tempo situations. A more efficient approach is to pre-compute the plan costs offline. This way, instead of invoking a planner, we have a lookup in a table or a map of plan costs. The problem is that, for a general planning domain, there is no well-known method of accurately pre-computing and storing plan costs for all possible combinations of initial and goal states. This remains an interesting research challenge. Various recent studies present different ideas that can lead to pre-computing approximate costs useable in inverse planning (Freedman et al. 2018; E.-Martín, R.-Moreno, and Smith 2015; Pereira, Oren, and Meneguzzi 2017; Vered and Kaminka 2017). With approximate costs, the plan recognizer may not produce accurate goal probability distributions but, like in the case of using a heuristic planner (Ramírez and Geffner 2009), we can hope for an approximate enough distribution to rank goals accurately. For navigation problems, where the issue is to predict the destination of an agent moving around, (Masters and Sardiña 2017) describes an approach for accurately precomputing plan costs.

Comparison Methodology

We can now present the benchmarks used for comparing deep neural networks and symbolic inverse planning, followed by the design of the neural network and the implementation of the inverse planner.

Benchmarks

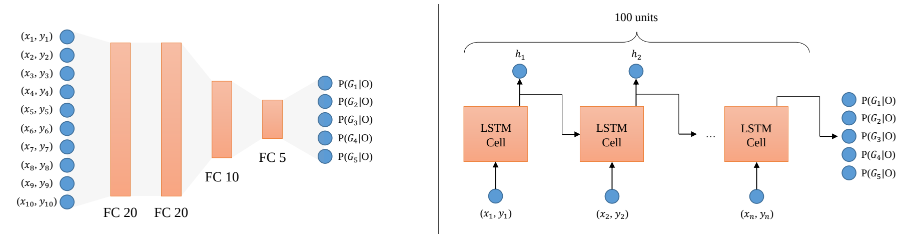
The comparison was made on five domains, using synthetic data:

1. NAVIGATION: Predicting the goal destination of an agent navigating a map (Masters and Sardiña 2017). The domain consists of 20 maps from StarCraft, provided by the MovingAI website[1], downscaled to 64x64 pixels, where the agent can perform actions limited to the discrete set of cardinal directions (North, South, East and West). A plan recognition problem consists in predicting the goal destination of an agent given a sequence of observed moves. Plan recognition problems were generated by placing one initial position and 5 goals on the maps.
2. INTRUSION DETECTION: Predicting the goals of network hackers with their activities (Geib and Goldman 2002). The observed agent is a user who may perform attacks on 10 hosts. There are 6 possible goals that the hacker might reach by performing 9 actions (recon, info-gathering, break-into, modify-files, clean, vandalize, gain-root, download-files and steal-data) on those servers. Observation sequences are approximately between 8 and 14 observations long.
3. KITCHEN: Inferring the activity of a cook in a smart home kitchen (Wu et al. 2007). The cook can either prepare breakfast, lunch or dinner (possible goals) (Wu et al. 2007). He can take objects, use them, and perform numerous high-level activities. Observation sequences are approximately between 3 and 8 actions long.
4. BLOCKSWORLD: Predicting the goal of an agent assembling 8 blocks labeled with letters, arranged randomly at the beginning (Ramírez and Geffner 2009). Achieving a goal consists in ordering blocks into a single tower to spell one of the 21 possible words by the use of 4 actions (pick-up, put-down, stack and unstack). Observation sequences are approximately between 6 and 10 actions long.
5. LOGISTICS: Predicting package delivery in a transport domain. Six packages must be conveyed between 6 locations in 2 different cities, using 1 airplane, 2 airports and 2 trucks (Ramírez and Geffner 2009). There are 6 possible actions available (load-truck, load-airplane, unload-truck, unload-airplane, drive-truck and fly-airplane) to achieve around 10 possible goals. Observation sequences are approximately between 16 and 22 actions long.

Those domains are among the benchmarks traditionally used for planning algorithms and are also often selected for evaluating the performance of a symbolic plan recognizer as referenced above. Ultimately, we want to evaluate plan recognizers using real-world benchmarks. Meanwhile, the synthetic domains can provide some useful insight.

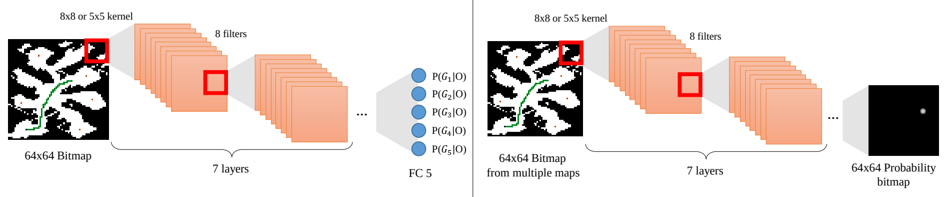
---

[1] MovingAI Lab: https://movingai.com/



(a) Fully Connected Network

(b) LSTM Network



(c) CNN Bitmap

(d) CNN Multimaps

Figure 2: Representation of our architectures for the navigation domain. (x_i, y_i) stands for the coordinates of the agent's location in the grid. (a), (b), and (c) were trained on a single map, while (d) was trained on multiple maps.

Deep Neural Networks

The deep neural networks were built using specific tools and hyper-parameters depending on the benchmark. For the navigation benchmark, we experimented with three different neural networks (see figure 2): a fully connected network, an LSTM network and a convolutional neural network (CNN). We felt the LSTM network and the CNN were appropriate for this domain, given that LSTMs usually perform well learning from sequences, whereas the CNN is usually appropriate for learning from 2D data (the graphic map in our case). The fully connected network has 4 dense layers of 20, 20, 10 and 5 units respectively, the LSTM has 100 units, and convolutional networks have 7 layers of 8 filters of size 8x8 or 5x5. For the four other domains, we used a fully connected network with three dense layers of 256, 32 and 5 units respectively. We do not need a CNN here since the input is not a 2D map as in the navigation domain.

Besides the architecture, an implementation of a neural network involves the choice of specific parameters, activation functions, and optimization algorithm. Given that we want to find a correct goal amongst a set of possible ones and work with probabilistic scores, we quantify the loss with the categorical cross-entropy function and work with accuracy metrics, as it is common in such a context of application. Hidden layers are activated with the ReLU function, while the output layer was activated with the softmax function. To train the networks, the Adam optimizer (Kingma and Ba 2014) was used, with a learning rate of 0.001, beta1 of 0.9, beta2 of 0.999 and no decay. To prevent overfitting, we also used dropout (Srivastava et al. 2014) for all layers with a drop chance set to 0.1 or 0.2. Finally, inputs were shuffled uniformly prior to training.

Experiments and Results

We used the different learning approaches described earlier for the navigation domain and for the remaining domains. As for the inverse planning algorithm, in the navigation domain, we used the one from (Masters and Sardiña 2017) (M-S) which pre-computes map costs offline. For the other benchmarks, we used the original probabilistic inverse planning implementation (Ramírez and Geffner 2010) (R-G). There is yet no proven method for pre-computing plan costs for these domains.

Navigation Domain

As mentioned above, three types of network architectures were trained for the navigation benchmark: one simple feed-forward fully connected network, an LSTM network and a fully convolutional network. Each of them was trained for 15 epochs on observations from a single map, with 100 observed paths. We additionally trained the convolutional network on multiple maps, regardless of their goals, start and obstacle positions, to see if and how it could generalize across multiple plan recognition navigation domains. To mimic suboptimal behavior, we started by generating noisy optimal paths to these goals with a modified A* algorithm. As paths were generated, we split our data into 2 parts: the training set was filled with entire paths (from start to goal), while the test set was made of sub-paths sampled from 0% to $X\%$ of the total path, where $X \in]0, 100[$.

We used (x, y) coordinates as input for the fully connected network (FC) and LSTM methods. As paths lengths may differ, we eventually retained a fixed number of positions among the ones available to form inputs of fixed size, padding with zeros shorter sequences. The output is a probability distribution over the possible goals.

For the fully convolutional network (CNNBitmap), the map was converted to a 4-channel image, where each channel displays in a black or white manner information on different aspects of the observations and the problem, such as the initial position, the positions of the potential goals, the visited positions, and walkable positions that are neither of the above. This transformed bitmap was afterwards given as an input to the network, to retrieve as an output a probability distribution over the goals.

Since methods FC, LSTM and CNNBitmap were trained and tested on the same map, where goals were known in advance, it was possible to deduce a probability distribution array of fixed size (equal to the number of goals). However, we could not make this assumption for the general fully convolutional method (CNNMultimaps) trained on multiple, different maps. The latter outputs a probability distribution over the entire map, representing the belief that the agent's goal is at one position or another, allowing any number of goals and positions in general.

For the (Masters and Sardiña 2017)'s method (labeled M-S), only the last position of sub-paths was used. The output is, again, a probability distribution over the possible goals.

We compared the accuracy of those 4 different networks on test sets with M-S. Cost maps were generated using optimal paths returned by the A* algorithm and stored prior to the call at the plan recognition algorithm. To compute the accuracy for both networks and inverse planning algorithms, we simply checked if the goal with maximal posterior probability matched the true goal of the agent. To compute the posterior probabilities in the case of M-S, we assumed prior probabilities to be uniform and used the default value of the beta parameter (1).

Results are shown in figure 3. The Y-axis represents the average accuracy on different maps, which is the percentage of correct predictions (a prediction is said to be correct if the true goal has the highest probability). The X-axis refers to the percentage sampled from total paths in the test set (for instance, if a path is a sequence of 100 positions, a sub-path sampled from 0% to 25% will only be comprised of some of the first 25 positions).

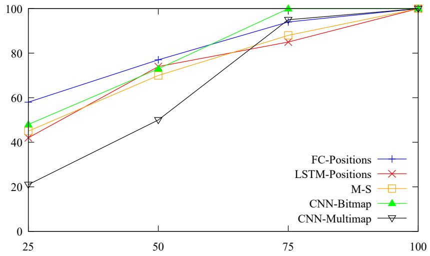


Figure 3: Results of accuracy depending on the percentage retained from the complete observed path, in the navigation domains.

As it can be seen, method FC (with an input of size 10) ranks first, followed by the fully convolutional network (CNNBitmap). The reason could be that the local connectivity of the convolution filters of the network impairs it for early predictions since it cannot reason about the observed path and the goals at the same time, as their positions often do not both enter the parameters window (i.e. the stacked convolution filters) for the first observations.

We have the intuition that better predictions could be achieved by using modified convolutional network architectures specialized for planning and predicting in 2D inputs like grid-worlds, such as value iteration network (Tamar et al. 2017) and visual relational reasoning (Watters et al. 2017). We will explore this avenue in future experiments.

The convolutional network trained on all maps shows relatively bad early predictions, proving there is still room for improvement in order for neural networks to generalize to multiple maps. Nonetheless, the method shows already great prediction potential and may be significantly improved by the use of the specialized architectures mentioned above.

Computing plan costs takes time, even offline. The results suggest that training neural networks, even if computationally complex, may be advantageous in this regard thanks to the trivially parallelizable nature of its operations and the computation power of modern hardware. However, a computation time comparison does not enlighten new advantages for this kind of context. Table 1 gives a summary of the offline and online computation times. The LSTM networks have longer training times but may generalize better to longer sequences of observations with bigger sliding windows (since we fixed the maximum number of observations input to 10 and thus do not benefit fully from LSTM's training power over sequences). The CNN trained on multiple maps takes a long time to train but could have the potential to generalize to every navigation problem, so no additional training would be required for an unseen map. Symbolic approaches have no need of training nor dataset, but knowledge about the domain is required to handcraft the model and costs must be generated for every new map, whether it is offline or online (during prediction).

	T	P
FC	10 s	10μs
LSTM	30 s	4 ms
CNNBitmap	10 s	4 ms
CNNMultimaps	20 min	4 ms
R-G	0	1 s
M-S	7 s	10μs

Table 1: Comparison of rough average computation times of the evaluated approaches on the navigation domain. T is the offline computation time, prior to the prediction. It corresponds to the training time for neural networks and the generation of costs maps for M-S. R-G do not precompute costs and reruns the planner for every $2|G|$ planning problems of a plan recognition problem. P is the online prediction time. It is the time required to output a probability distribution over the goals of a plan recognition problem.

Other Domains

The navigation benchmark deals with path-planning problems requiring much less domain abstractions than the other domains, that are Blocks World, Intrusion Detection, Kitchen and Logistics. The last four benchmarks correspond to task-planning problems, involving constraints that differ from those in the navigation benchmarks, thus requiring different kinds of domain representations. In fact, we represented the last four benchmarks using the Planning Domain Definition Language (PDDL) (Ramírez and Geffner 2010).

A fully connected network was trained on each of these task-planning domains during 15 epochs, with a number of training examples ranging from 1000 to 3000 depending on the domain. We also trained an LSTM on these examples, but it ended up taking more time without providing significant result improvements. A training example in the datasets is a sequence of observations from PDDL files, encoded with zeros or ones. The datasets are split 80%-20% for training and test.

A training example corresponds to a sequence of observations. Each observation in the sequence is one action type plus its arguments which are transformed to a one-hot vector (for instance, if there are 3 possible action types, there will be 3 corresponding vectors: 100, 010, and 001). The neural network receives the complete sequence of transformed observations. To match a fixed input size, sequences shorter than the maximum size are padded with zeros and shifted $maxSize - size + 1$ times (for instance, if one observation is AB and the maximum size is 4, 3 new observations will be created: $AB00$, $0AB0$, $00AB$), hence generating new training data. The network returns a probability distribution over the potential goals pursued, given the observations.

We compared the accuracy of the network with the one from (Ramírez and Geffner 2010), labeled R-G. The costs were generated online, as implemented by (Ramírez and Geffner 2010), from optimal plans found by the HSP planner. The beta parameter was left at its default value (1) and the prior probabilities of the goals were presumed to be uniform.

Results in figure 4 show that the fully connected network trained with observations outperforms the R-G approach almost every time. In addition to providing higher prediction rates, networks are also quicker: on such plan recognition problems, the training part takes approximately 1 minute to infer reusable weights, while one prediction is made under 1ms. The R-G approach does not require training nor offline computation, but provides a prediction in minutes, sometimes hours, which is really long and cannot be applied to real-time decision making. A suboptimal planner might reduce computation times, but we can reasonably assume that it would still remain above several minutes or so for each goal prediction.

Conclusion

The above comparison between deep learning and symbolic inverse planning, although still preliminary, suggests that deep learning outperforms symbolic inverse planning, at least in the five domains considered.

A next step of the pursued experimentation is validating the experiment on real word data. Further experimentations can try different deep neural networks (Goodfellow, Bengio, and Courville 2016), different symbolic plan recognition methods, different multi-agent configurations for plan recognition, sensor limitations (partial observability vs full observability), attitudes between the observed agent and the observer (cooperative, adversarial, neutral), and different domains of application.

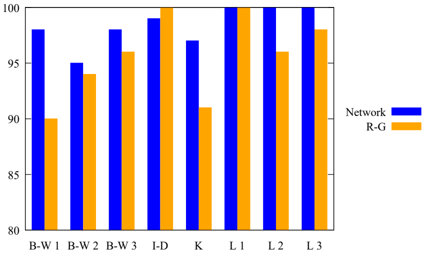


Figure 4: Results of accuracy for the task-planning domains (B-W, I-D, K, and L stand for Blocks World, Intrusion Detection, Kitchen and Logistics respectively). The neural network outperforms R-G in almost every plan recognition domain.

It should be noted that inverse planning is a principle, albeit implemented so far by symbolic inference techniques (Ramírez and Geffner 2010; Masters and Sardiña 2017; Baker, Saxe, and Tenenbaum 2009). Various deep neural network architectures attempt to implement planning techniques previously implemented using symbolic approaches. Value iteration networks implementing the value iteration algorithm for Markov decision planning problems (Tamar et al. 2016) or deep reinforcement learning implementing traditional reinforcement learning algorithms (Mnih et al. 2015; 2016) are some examples. In a similar vein, it would be interesting to implement inverse planning using a neural network. Since inverse planning is a concept, here we used the terminology symbolic inverse planning to refer to the symbolic implementation of the concept.

In some applications, it is important that the plan recognizer explains the rationale of its inferences. However, extracting a meaningful explanation from a neural network still remains a challenge. In contrast, the symbolic representation of symbolic plan recognizers makes the explanations easier, except that, as we have argued, those approaches are difficult to ground in real-world environments. This suggests that the exploration of hybrid approaches, such as those discussed in the related work section, remains worth pursuing.

Acknowledgements

This work was supported by grants from the Natural Sciences and Engineering Research Council (NSERC) of Canada, and the *Fonds de recherche du Québec – Nature et technlogies* (FRQNT). We are also thankful to Julien Filion, Simon Chamberland and anonymous reviewers for their insightful feedback that helped improve the paper.

References

Almeida, A., and Azkune, G. 2018. Predicting human behaviour with recurrent neural networks. *Applied Sciences* 8(2).

Baker, C. L.; Saxe, R.; and Tenenbaum, J. B. 2009. Action understanding as inverse planning. *Cognition 2009* 113(3):329–349.

Bisson, F.; Larochelle, H.; and Kabanza, F. 2015. Using a recursive neural network to learn an agent's decision model for plan recognition. In *IJCAI 2015*, 918–924.

Cooper, G. F. 1990. The computational complexity of probabilistic inference using bayesian belief networks (research note). *Artificial Intelligence* 42(2-3):393–405.

E.-Martín, Y.; R.-Moreno, M. D.; and Smith, D. E. 2015. A fast goal recognition technique based on interaction estimates. In *IJCAI 2015*, 761–768.

Freedman, R. G.; Fung, Y. R.; Ganchin, R.; and Zilberstein, S. 2018. Towards quicker probabilistic recognition with multiple goal heuristic search. In *AAAI 2018*, 601–606.

Geib, C. W., and Goldman, R. P. 2002. Requirements for plan recognition in network security systems. In *International Symposium on Recent Advances in Intrusion Detection 2002*.

Goodfellow, I.; Bengio, Y.; and Courville, A. 2016. *Deep Learning*. MIT Press.

Granada, R.; Pereira, R.; Monteiro, J.; Barros, R.; Ruiz, D.; and Meneguzzi, F. 2017. Hybrid activity and plan recognition for video streams. In *AAAI 2017*.

Hou, J.; Wu, X.; Chen, J.; Luo, J.; and Jia, Y. 2018. Unsupervised deep learning of mid-level video representation for action recognition. In *AAAI 2018*, 6910–6917.

Kingma, D. P., and Ba, J. 2014. Adam: A method for stochastic optimization. *CoRR* abs/1412.6980.

Masters, P., and Sardiña, S. 2017. Cost-based goal recognition for path-planning. In *AAMAS 2017*, 750–758.

Min, W.; Ha, E.; Rowe, J. P.; Mott, B. W.; and Lester, J. C. 2014. Deep learning-based goal recognition in open-ended digital games. In *AIIDE 2014*.

Min, W.; Mott, B. W.; Rowe, J. P.; Liu, B.; and Lester, J. C. 2016. Player goal recognition in open-world digital games with long short-term memory networks. In *IJCAI 2016*, 2590–2596.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M. A.; Fidjeland, A.; Ostrovski, G.; Petersen, S.; Beattie, C.; Sadik, A.; Antonoglou, I.; King, H.; Kumaran, D.; Wierstra, D.; Legg, S.; and Hassabis, D. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Mnih, V.; Badia, A. P.; Mirza, M.; Graves, A.; Lillicrap, T. P.; Harley, T.; Silver, D.; and Kavukcuoglu, K. 2016. Asynchronous methods for deep reinforcement learning. In *ICML 2016*, volume 48, 1928–1937.

Pereira, R. F.; Oren, N.; and Meneguzzi, F. 2017. Landmark-based heuristics for goal recognition. In *AAAI 2017*, 3622–3628.

Ramírez, M., and Geffner, H. 2009. Plan recognition as planning. In *IJCAI 2009*, 1778–1783.

Ramírez, M., and Geffner, H. 2010. Probabilistic plan recognition using off-the-shelf classical planners. In *AAAI 2010*.

Schmidt, C. F.; Sridharan, N. S.; and Goodson, J. L. 1978. The plan recognition problem: An intersection of psychology and artificial intelligence. *Artificial Intelligence* 11(1-2):45–83.

Simonyan, K., and Zisserman, A. 2014. Two-stream convolutional networks for action recognition in videos. In Ghahramani, Z.; Welling, M.; Cortes, C.; Lawrence, N. D.; and Weinberger, K. Q., eds., *NIPS 2014*, 568–576.

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

Tamar, A.; Levine, S.; Abbeel, P.; Wu, Y.; and Thomas, G. 2016. Value iteration networks. In *NIPS 2016*, 2146–2154.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2017. Value iteration networks. In *IJCAI 2017*, 4949–4953.

Vered, M., and Kaminka, G. A. 2017. Heuristic online goal recognition in continuous domains. In *IJCAI 2017*, 4447–4454.

Watters, N.; Tacchetti, A.; Weber, T.; Pascanu, R.; Battaglia, P.; and Zoran, D. 2017. Visual interaction networks. *CoRR* abs/1706.01433.

Wu, J.; Osuntogun, A.; Choudhury, T.; Philipose, M.; and Rehg, J. M. 2007. A scalable approach to activity recognition based on object use. In *ICCV 2007*.

Yan, S.; Teng, Y.; Smith, J. S.; and Zhang, B. 2016. Driver behavior recognition based on deep convolutional neural networks. In *ICNC-FSKD 2016*, 636–641.