Learning Generalized Heuristics using Deep Neural Networks

Julia Nakhleh, Siddharth Srivastava
School of Computing, Informatics and Decision Systems Engineering
Arizona State University

Abstract

Classical planning is a field of Artificial Intelligence concerned with allowing autonomous agents to make reasonable decisions in complex environments. This work investigates the application of deep learning and planning techniques, with the aim of constructing generalized plans capable of solving multiple problem instances. We construct a Deep Neural Network that, given an abstract problem state, predicts both (i) the best action to be taken from that state and (ii) the generalized "role" of the object being manipulated. The neural network was tested on two classical planning domains: the blocks world domain and the logistic domain. Results indicate that neural networks are capable of making such predictions with high accuracy, indicating a promising new framework for approaching generalized planning problems.

Introduction

Classical planning is a field of Artificial Intelligence concerned with allowing autonomous intelligent agents, such as robots, to make reasonable decisions in complex environments in order to achieve a given objective. In order to perform "simple" tasks such as doing laundry or arranging dishes, a robot must be able to reason about the best course of action that will allow it to achieve a desired outcome (for example, separating clothes such that whites are separated from colors and pants from shirts, or arranging dishes in a cabinet such that bowls are stored on top of plates). Due to the fact that the environments in which robots operate are often large and complex, planning is a computationally challenging problem. Even when the environment in which the agent operates is deterministic and fully-observable, determining the existence of a plan to achieve a given objective in this environment is a PSPACE-complete problem (Bylander 1994).

In order to address this inefficiency, there exist multiple approaches that allow planners to reuse knowledge acquired while planning across multiple problem instances for faster plan computation on a new problem instance (Groshev et al. 2017). One such approach involves the construction of generalized plans that solve multiple problem instances. Classical planners tend to experiment slowdown when the number of elements in the environment increases even slightly; however, such slowdown can be mitigated by identifying common patterns in solutions to various problems, then executing these same solutions with only minor modifications to solve larger problems. By identifying common structures in different problems and their solutions, new problem instances can be solved more efficiently (Srivastava, Immerman, and Zilberstein 2011).

In this project, we combine this approach toward generalized planning with the use of Deep Neural Networks (DNNs) to learn *generalized reactive policies*. Given a planning problem instance and a generalized representation of a state, a generalized reactive policy (GRP) predicts the best action to be taken from that state. We constructed a GRP in the form of a DNN that, given a planning problem instance and a generalized state representation, predicts an action to be taken from that state. In addition, given an generalized state representation, the network also predicts the *role* (defined as the set of unary predicates in the domain that the object satisfies) of the object being manipulated by the action taken from that state.

We tested the network with data from two different classical planning domains: the blocks world domain, which involves stacking and unstacking wooden blocks on a table, and the logistics domain, in which one or more trucks load, unload, and transport crates between a set of distinct locations. Initial results indicate that, in the (simpler) blocks world domain, the network is able to predict both actions and roles with perfect accuracy. In the more-complex logistics domain, the network is able to predict actions with very high accuracy (about 96%) and roles with modestly-high accuracy (about 75%). We hypothesize that the performance of the network when predicting roles can be improved with further tuning of the network structure as well as some minor modifications to the logistics domain itself.

Related Work

There has been extensive research about the application of learning techniques to classical planning interfaces. Martín and Geffner (2004) learn generalized policies in the form of decision lists using *concept languages*, which combine standard logical representations with syntax designed for reasoning about generalized object classes. Yoon, Fern and Givan (2002) extend Martín and Geffner's previous work to learn generalized policies as decision lists for stochastic domains, ensemble learning, and other varieties of problems. Abel et al. (2015) improve planning efficiency by learning action priors that allow robots to ignore irrelevant actions when planning towards a current goal. In a similar vein, Rosman and Ramamoorthy (2012) use reinforcement learning techniques to learn action priors, improving planning efficiency on larger and more complex domains. Reinforcement learning has also been used to allow for "transfers" of knowledge, skills and models such that robots can reuse previously obtained knowledge while planning (Konidaris 2006; Konidaris, Scheidwasser, and Barto 2012).

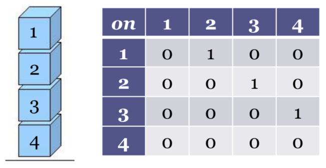
Deep Neural Networks has achieved remarkable success in a wide variety of AI domains. Currently, deep learning represents the state-of-the-art in domains such as image classification (Krizhevsky, Sutskever, and Hinton 2012), natural language processing (Sutskever, Vinyals, and Le 2014), and allowing agents to attain human-level control of an environment (Mnih et al. 2015). Neural networks have also been used to learn heuristic functions (Ernandes and Gori 2004), which are used in a variety of other AI domains such as searching. Previously, deep learning techniques have been combined with planning to aid in tasks such as obstacle avoidance (Ross, Gordon, and Bagnell 2011), path following (Pomerleau 1988), robot motor skills (Mlling et al. 2013), navigation in a 2D gridworld domain (Tamar et al. 2017), and—as in this work—block stacking (Duan et al. 2017).

Deep learning has also been successfully applied to domains that, from a planning perspective, are much more difficult than block stacking or obstacle avoidance. For example, Silver et al. (2017) construct an Alpha-Go playing agent that, through reinforcement learning and self-play, learns a policy in the form of a DNN. This type of self-play strategy has also been used in the construction of a Backgammmon playing agent (Tesauro 1995). In another case, Weber et al. (2017) propose a Deep Neural Network that combines elements of model-based planning and model-free reinforcement learning, the results of which were demonstrated on the Sokoban domain. Groshev, Goldstein, Tamar, Srivastava and Abbeel (2017) construct a deep neural network that learns a GRP based on images of successful traces in the Sokkoban domain; they also show that this learned GRP performs well as a heuristic in directed search algorithms.

In contrast to the above approach, much of the literature on generalized planning has focused on iterative computation of generalized plans with strong guarantees of correctness. Hu and De Giacomo (2011) provide a formal, representation-independent definition of generalized planning, and prove that generalized planning for a finite set of environments is always decidable and EXPSPACE complete. Srivastava et al. (2011) present a framework for generalized planning that relies on state abstraction, allowing for the iterative computation of generalized plans across a variety of domains. Our work aims to combine the latter approach with deep learning techniques to learn generalized reactive policies.

Formal Framework for State Abstraction

Given a domain, a goal formula, and a set of initial states (that may or may not be from the same state space), a *generalized planner* is able to compute a generalized plan that solves all of the initial states. Srivastava, Immerman and Zilberstein (2011) present an approach to the generalized planning problem that relies on a method of state abstraction capable of compactly representing unbounded sets of concrete states. The key insight of this method is that it differentiates between *summary elements*, which may represent multiple concrete physical entities, and *non-summary elements*, which must only represent a single physical entity. To illustrate this distinction, consider the example given in Figure 1 of a state in the blocks world domain:

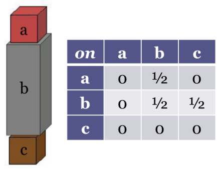


on	1	2	3	4
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	0	0	0	0

Figure 1: Non-abstracted representation of a blocks world state (Srivastava 2011).

Traditionally in classical planning, a state is can be represented as a two-valued logical structure: a 0 indicates falsehood and a 1 indicates truth. In the example above, block 1 is stacked directly on top of block 2: therefore, the predicate (*on* 1 2) evaluates to *true*. In contrast, block 4 is not stacked directly on top of any other blocks, hence the entire row corresponding to block 4 is valued as 0.

Using the method proposed by Srivastava et al., multiple concrete elements may be jointly represented as a single summary element. States are thus represented as a three-valued logical structure: 0 indicates falsehood, 1 indicates truth, and $\frac{1}{2}$ indicates possible truth. Consider the same blocks world as listed above, now represented in abstract:



on	a	b	c
a	0	½	0
b	0	½	½
c	0	0	0

Figure 2: Abstract representation of a blocks world state (Srivastava 2011).

In the above abstract representation, blocks 2 and 3 are combined into a single summary element b, while blocks 1 and 4 correspond to non-summary elements a and c, respectively. Using this encoding, binary relationships become imprecise. While we can definitively say that c is not on a, for example, we can no longer definitively say that a is on b or that b is on c. Block a is directly stacked on top of some of the blocks that b represents, but not on others; the same is true of b and c. Thus (on a b) and (on b c) assume the truth value of $\frac{1}{2}$, indicating an "unknown" or "undefined."

Using this abstraction framework, the *role* of any object is defined as the set of unary predicates it satisfies, which corresponds to the summary element or non-summary element that it belongs to. For example, block 1 in Figure 1 satisfies only the unary predicate $clear$ and block 4 satisfies only the unary predicate $onTable$, while blocks 2 and 3 satisfy neither (i.e. $\neg clear \wedge \neg onTable$). As seen in Figure 2, each summary element corresponds to one of these roles: a encompasses all blocks whose role is $clear$, b encompasses all blocks whose role is $\neg clear \wedge \neg onTable$, and c encompasses all blocks whose role is $onTable$.

This framework for state abstraction was originally proposed with the aim of solving problem instances of unbounded size while maintaining strong guarantees of correctness. However, in this work, we take a new approach by using these abstracted states as input to a Deep Neural Network. Given a matrix representing an abstracted state, our network is able to predict both (a) the correct action to take from that state and (b) the role of the object being acted upon with high accuracy. The structure of the network is illustrated in the "Neural Network Design" section.

Experiments

Testing was done in two different classical planning domains: the blocks world domain and the logistic domain. The blocks world domain consists of a set of wooden blocks sitting atop a table. In our version of blocks world, there are between 2 and 10 blocks which are initially arranged in a random configuration of 1 or more towers. The goal is to unstack all of the blocks, moving only one block at a time. In the logistics domain, one or more trucks is tasked with loading, unloading, and delivering crates to their proper destinations. The initial location of each crate, as well as its goal destination, is specified explicitly. For our tests, this task was performed with 1 truck, 5 locations, and between 15 and 20 crates. All crates start out randomly distributed between each of the 5 locations; the goal state has crates evenly distributed among locations. For example, in a planning instance with 15 states, the goal configuration will always have 3 crates at each of the 5 locations. In our version of the logistics domain, the truck is assumed to have unlimited capacity, and is therefore capable of carrying any number of crates at one time.

Planning in both domains was done using the Fast-Forward (FF) Planner (Hoffmann and Nebel 2001). Given an initial configuration, a goal configuration, and a file specifying the rules of the planning domain, the FF planner uses forward state space search to compute the sequence of actions necessary to reach the goal state from the start state, as well as the state of the environment after each action is performed. This set of states, as well as the corresponding correct action to be taken from each state, form the basis of the training data and labels that are provided to the neural network.

For each problem instance given to the FF planner, all intermediate states and corresponding correct actions were extracted in the form of a graph. Each state was then converted into its abstract representation using the approach described in the previous section, while each action was encoded as a 1-hot vector. In addition, the role of the object being manipulated by the action was encoded as a bit vector. As an example, consider a problem whose initial state is given by the tower in Figure 1. The goal is to unstack the entire tower, leaving blocks 1, 2, 3, and 4 on the table. Given this problem instance, FF will return the following plan:

i) UNSTACK 1 2
ii) UNSTACK 2 3
iii) UNSTACK 3 4

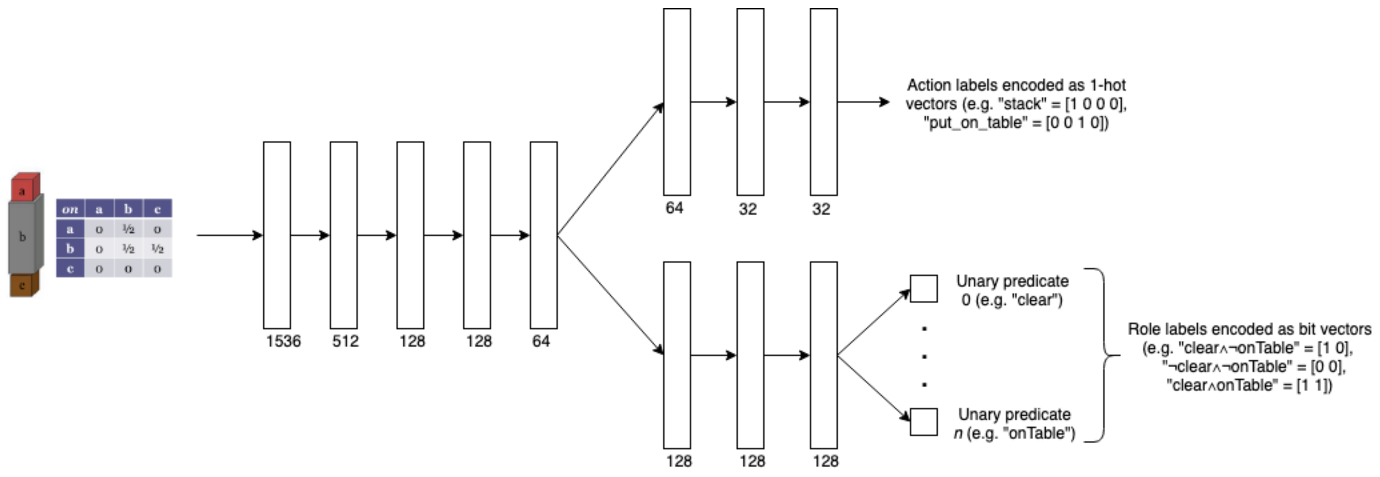
Let the initial state be state A. The correct action to be performed from A is action i, during which block 1 is removed from the tower and placed on the table. In the blocks domain, there are four possible actions: STACK, UNSTACK, PICK-FROM-TABLE, and PUT-ON-TABLE. Action i, UNSTACK, will thus be encoded as a one-hot vector: $[0, 1, 0, 0]$. Before this action is executed, the role of block 1 is just one unary predicate ($clear$). Roles are encoded as an n-bit vector, where n is the number of unary predicates in the domain. Since blocks world has only two unary predicates, the role encoding of block 1 will thus be $[1, 0]$, corresponding to the role $clear \wedge \neg onTable$. This information thus constitutes one input/output/output triplet for the neural network. The abstract representation of state A (Figure 2) is passed to the network as input, along with two labels: i) $[0, 1, 0, 0]$ (the one-hot encoding of the action UNSTACK) and ii) $[1, 0]$ (the bit encoding of the role $clear \wedge \neg onTable$).

To generate training data for the neural network, we ran the FF planner on 6,000 blocks world problem instances and 500 logistics problem instances, which yielded 35,012 and 29,765 training samples, respectively. To generate test data, we ran FF on 1,000 blocks world problems (which yielded 5,762 testing samples) and 100 logistics problems (6,063 testing samples).

Neural Network Design

The network we designed (depicted in Figure 3) is a standard, fully-connected network consisting of: i) five "shared" layers, ii) three layers that learn action labels, and iii) $3 + n$ layers that learn role labels, with n being the number of unary predicates present in the domain. Each of the n layers that correspond to unary predicates have dimensions 1×1, corresponding to one bit of the object role encoding. Each of these "bit" layers takes on either the value 1 (the unary predicate is present in the object role) or 0 (the unary predicate is *not* present in the object role). Encoding roles as bit vectors, rather than one-hot vectors taken across all possible combinations of unary predicates that exist in the domain, greatly improves the network's space efficiency by allowing for more compact role representation; it also has the advantage of allowing us to examine learning on each unary predicate individually, thus identifying which, if any, unary predicates are "easier" or "harder" for the network to identify.

The network trains on a large training data set consisting of three-valued logical structures representing an abstracted blocks world state (as in Figure 2) and two sets of output labels: one consisting of corresponding one-hot encoded vectors representing the correct action to take from that state, and another consisting of bit-encoded vectors representing the role of the concrete object being acted upon. The network is designed to minimize the sum of total loss across



on a b c
a 0 ½ 0
b 0 ½ ½
c 0 0 0
1536
512
128
128
64
64
32
32
128
128
128
Action labels encoded as 1-hot vectors (e.g. "stack" = [1 0 0 0], "put_on_table" = [0 0 1 0])
Unary predicate 0 (e.g. "clear")
Unary predicate n (e.g. "onTable")
Role labels encoded as bit vectors (e.g. "clear∧¬onTable" = [1 0], "¬clear∧¬onTable" = [0 0], "clear∧onTable" = [1 1])

Figure 3: Neural network design. Inputs are three-valued matrices representing abstracted states of the environment. Labels for actions are one-hot encoded vectors; labels for roles are bit-encoded vectors.

both actions and roles (where total loss across roles is computed as a softmax of loss on each 1×1 unary predicate layer) rather than separately minimizing the loss on learned actions and the loss on learned roles. In this way, the network learns holistically rather than effectively operating as two separate networks learning on two different sets of labels and data.

Results - Blocks World Domain

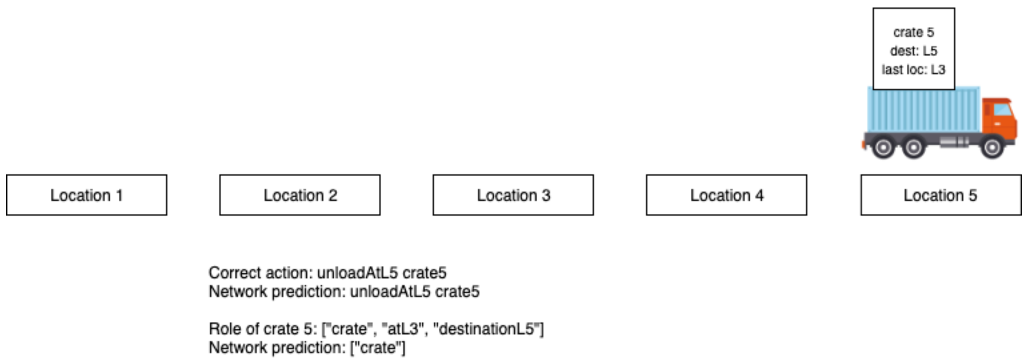
In the blocks world domain, the neural network almost immediately achieved 100% accuracy on both action and role predictions with $\approx$ 0.00 total loss. This extremely high accuracy is almost certainly due to the simple nature of the blocks world problems that the network was given to learn on. Because these tests were initially carried out as a proof of concept, we restricted problems to 2-10 blocks in which the goal was always "all blocks on table." For this reason, in every blocks world state, the correct action was to unstack the top block, meaning that the network only had to learn two actions ("unstack" and "putOnTable") and one role ($clear$). Despite the limited nature of this success, the network's high accuracy demonstrates the feasability of our approach, indicating that networks are in fact capable of learning meaningful information given only an abstract representation of a state (rather than its complete representation).

Results - Logistic Domain

In the logistic domain, the network was able to predict correct actions with very high accuracy (ranging from 90-100% throughout testing, generally landing at 96%). Role prediction accuracy hovered around 75% throughout testing. To illustrate why this may be the case, consider Figures 4-7, which show four sample states in the logistics domain as well as our network's action and role predictions for each state. In all four cases, the network predicts the correct action. In States 1-3 (Figures 1-6), the network only correctly predicts one predicate within the manipulated object's role ($crate$), but fails to predict predicates that indicate the object's current position and destination. In State 4, however, the network correctly predicts the entire role of the moved object ($truck \wedge atL4$).

We speculate that this may be a result of way the logistics domain is specified: specifically, the fact that crates are listed as having their previous destination, even *after* being loaded into a truck and driven away, until they are delivered to their next destination. In other words, if a crate is initially located at location x, then loaded into a truck which drives to destination y, the crate's location is still listed as x for as long as the crate remains within the truck. Only once the crate is unloaded at location y is its current location updated to y. This encoding of locations makes it difficult for the network to correctly predict the role of unloaded crates, because the previous location of an object has nothing to do with the location where it is being currently unloaded. In other words, if a crate is about to be unloaded at location z, the network has no way of knowing whether its "current" location (i.e. the location where the truck most recently picked it up) is location y, location x, location w, or any other possible location.

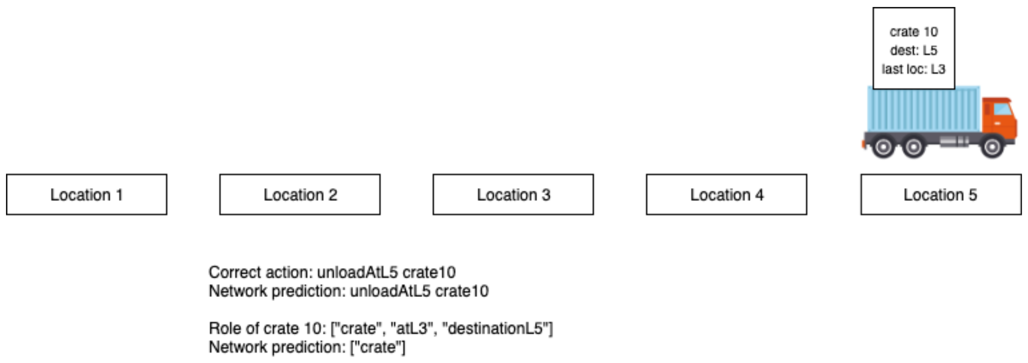
Thus, to predict unary predicates that indicate the current location of crates ($atL5$, $atL4$, etc.), the network is essentially reduced to random guessing. One possible way to eliminate this source of uncertainty would be to modify the rules of the logistics domain such that, while a crate is inside a truck, its current location is undefined (rather than being defined as the most recent location where it was loaded onto the truck). Using this framework, as long as a crate is still inside a truck, all location predicates ($atL5$, $atL4$, etc.) will evaluate to false for that crate. This would eliminate the need for the network to "guess" which of these predicates evaluates to true for crates still located inside trucks, which would likely improving the network's accuracy on role predictions as a whole.



Correct action: unloadAtL5 crate5
Network prediction: unloadAtL5 crate5

Role of crate 5: ["crate", "atL3", "destinationL5"]
Network prediction: ["crate"]

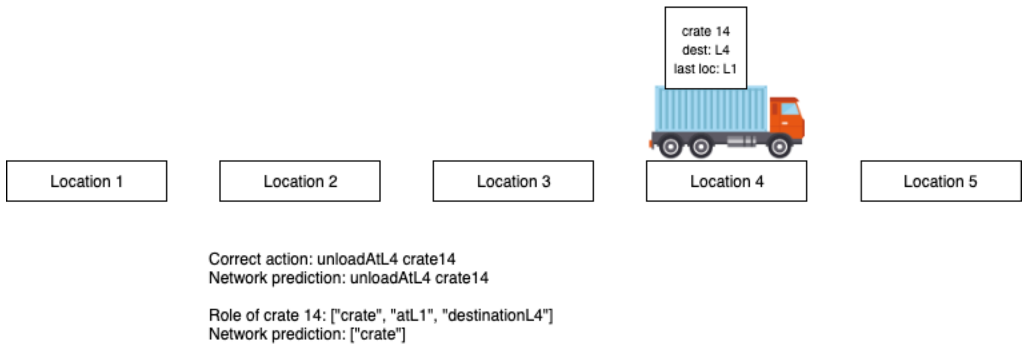
Figure 4: State 1. Here, the network predicts the correct action (“unloadAtL5 crate5”) but only one correct unary predicate within the object’s role ($crate$).



Correct action: unloadAtL5 crate10
Network prediction: unloadAtL5 crate10

Role of crate 10: ["crate", "atL3", "destinationL5"]
Network prediction: ["crate"]

Figure 5: State 2. Here, the network predicts the correct action (“unloadAtL5 crate10”) but only one correct unary predicate within the object’s role ($crate$).



Correct action: unloadAtL4 crate14
Network prediction: unloadAtL4 crate14

Role of crate 14: ["crate", "atL1", "destinationL4"]
Network prediction: ["crate"]

Figure 6: State 3. Here, the network predicts the correct action (“unloadAtL4 crate14”) but only one correct unary predicate within the object’s role ($crate$).

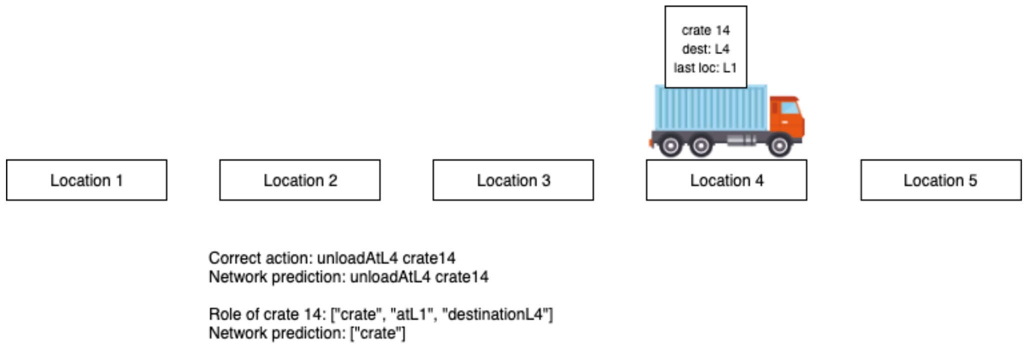

crate 14
dest: L4
last loc: L1
Location 1
Location 2
Location 3
Location 4
Location 5
Correct action: unloadAtL4 crate14
Network prediction: unloadAtL4 crate14
Role of crate 14: ["crate", "atL1", "destinationL4"]
Network prediction: ["crate"]

Figure 7: State 4. Here, the network predicts the correct action (“moveToL5 truck1”) and correctly predicts both unary predicates present in truck 1’s role ($truck \wedge atL4$).

Future Work

In the logistics domain, further work remains to be done to examine whether the network’s prediction accuracy on roles can be improved from its current 75% average. Implementing the previously-discussed change to the logistics domain would very likely improve role-prediction accuracy by eliminating the network’s need to guess about the current location of crates that are inside trucks. We also plan to re-run the network using different methods to calculate total loss (sum-of-squares, for example) and compare performance against our current softmax loss calculation.

Further experimentation with different network structures may present new insight into the network’s mechanics and improve its capacity to identify patterns. In particular, using a Convolutional Neural Network (CNN) in place of a standard neural network, as was used in this project, may provide greater accuracy and/or additional insights. Because CNNs are principally designed to perform pattern extraction in tasks such as image recognition, their usefulness on the three-valued matrix input may be somewhat limited. On the other hand, a CNN be able to identify meaningful clusters or correlations of values in the state abstractions that a standard DNN would not identify. There also remains the possibility of constructing a CNN that operates on input in the form of images, rather than 3-valued matrices, that represent abstract states. This approach is more complex, as images are more difficult for neural networks to process and identify than three-valued matrices: however, a successful image-based approach would provide a useful point of comparison to our current matrix-based approach.

In the future, we also plan to investigate the usefulness of our network’s learned GRPs as planning heuristics by comparing them against greedy (i.e. best-first) search results. The research done by Groshev et al. (2017) indicates that Deep Neural Networks are capable of learning policies that perform very well in comparison to greedy search approaches. We plan to perform similar tests in our domain by running a search algorithm on various logistics problems that uses our network’s learned GRP as an action-selection heuristic, and compare its performance against a similar search that relies purely on a best-first action-selection approach.

Finally, we also hope to test our neural network on a variety of planning problems from other classical planning domains. Different domains present different challenges, and the complexity of planning problems varies greatly among domains. Comparing neural network performance on action and role prediction in several different domains may provide useful insight into which types of domains are best, or worst, suited to generalized planning and learning of generalized heuristics.

Conclusion

In this paper, we present a novel approach to generalized planning that combines state abstraction methods with deep learning. Preliminary results are promising, indicating that, given an abstract representation of a state in which concrete objects are “generalized” into roles based on which unary predicates they fulfill, our network is able to predict both the correct action to take from the given state and the role of the object being acted upon with high accuracy. In the blocks world domain, these predictions achieve 100% accuracy for both actions and roles; in the logistics domain, predictions for actions average 96% accuracy and predictions for roles average 75% accuracy. Although we hope to make adjustments such that the neural network achieves greater accuracy when predicting roles as well as actions, its success in predicting actions is notable because it suggests a viable deep learning-based method for constructing GRPs and for learning generalized heuristics in classical planning domains.

References

Abel, D.; Hershkowitz, D. E.; Barth-Maron, G.; Brawner, S.; O’Farrell, K.; MacGlashan, J.; and Tellex, S. 2015. Goal-based Action Priors. In *Proceedings of the Twenty-Fifth International Conference on International Conference on Automated Planning and Scheduling*, ICAPS’15, 306–314. AAAI Press. event-place: Jerusalem, Israel.

Bylander, T. 1994. The computational complexity of propositional STRIPS planning. *Artificial Intelligence* 69(1):165–204.

Duan, Y.; Andrychowicz, M.; Stadie, B. C.; Ho, J.; Schneider, J.; Sutskever, I.; Abbeel, P.; and Zaremba, W. 2017. One-Shot Imitation Learning. *arXiv:1703.07326 [cs].* arXiv: 1703.07326.

Ernandes, M., and Gori, M. 2004. Likely-admissible and Sub-symbolic Heuristics. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, 613–617. Amsterdam, The Netherlands, The Netherlands: IOS Press. event-place: Valencia, Spain.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2017. Learning Generalized Reactive Policies using Deep Neural Networks. 12.

Hoffmann, J., and Nebel, B. 2001. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research* 14:253–302.

Hu, Y., and De Giacomo, G. 2011. Generalized Planning: Synthesizing Plans that Work for Multiple Environments. *IJCAI* 6.

Konidaris, G.; Scheidwasser, I.; and Barto, A. G. 2012. Transfer in Reinforcement Learning via Shared Features. *J. Mach. Learn. Res.* 13:1333–1371.

Konidaris, G. 2006. A Framework for Transfer in Reinforcement Learning.

Krizhevsky, A.; Sutskever, I.; and Hinton, G. E. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, 1097–1105. USA: Curran Associates Inc. event-place: Lake Tahoe, Nevada.

Martn, M., and Geffner, H. 2004. Learning Generalized Policies from Planning Examples Using Concept Languages. *Applied Intelligence* 20(1):9–19.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; 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.

Mlling, K.; Kober, J.; Kroemer, O.; and Peters, J. 2013. Learning to select and generalize striking movements in robot table tennis. *The International Journal of Robotics Research* 32(3):263–279.

Pomerleau, D. A. 1988. ALVINN: An Autonomous Land Vehicle in a Neural Network. In *Proceedings of the 1st International Conference on Neural Information Processing Systems*, NIPS'88, 305–313. Cambridge, MA, USA: MIT Press.

Rosman, B., and Ramamoorthy, S. 2012. What good are actions? Accelerating learning using learned action priors. *2012 IEEE International Conference on Development and Learning and Epigenetic Robotics (ICDL)* 1–6.

Ross, S.; Gordon, G. J.; and Bagnell, J. A. 2011. A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning. *AISTATS*. arXiv: 1011.0686.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; Chen, Y.; Lillicrap, T.; Hui, F.; Sifre, L.; van den Driessche, G.; Graepel, T.; and Hassabis, D. 2017. Mastering the game of Go without human knowledge. *Nature* 550(7676):354–359.

Srivastava, S.; Immerman, N.; and Zilberstein, S. 2011. A new representation and associated algorithms for generalized planning. *Artificial Intelligence* 175(2):615–647.

Srivastava, S. 2011. Hybrid Search for Generalized Plans Using Classical Planners.

Sutskever, I.; Vinyals, O.; and Le, Q. V. 2014. Sequence to Sequence Learning with Neural Networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, 3104–3112. Cambridge, MA, USA: MIT Press. event-place: Montreal, Canada.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2017. Value Iteration Networks. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, 4949–4953. Melbourne, Australia: International Joint Conferences on Artificial Intelligence Organization.

Tesauro, G. 1995. Temporal Difference Learning and TD-Gammon. *Commun. ACM* 38(3):58–68.

Weber, T.; Racanire, S.; Reichert, D. P.; Buesing, L.; Guez, A.; Rezende, D. J.; Badia, A. P.; Vinyals, O.; Heess, N.; Li, Y.; Pascanu, R.; Battaglia, P.; Hassabis, D.; Silver, D.; and Wierstra, D. 2017. Imagination-Augmented Agents for Deep Reinforcement Learning. *arXiv:1707.06203 [cs, stat].* arXiv: 1707.06203.

Yoon, S.; Fern, A.; and Givan, R. 2002. Inductive Policy Selection for First-order MDPs. In *Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence*, UAI'02, 568–576. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc. event-place: Alberta, Canada.