Heuristic Search Planning with Deep Neural Networks using Imitation, Attention and Curriculum Learning

Leah Chrestien, Tomáš Pevný, Antonín Komenda, Stefan Edelkamp

Department of Computer Science, Faculty of Electrical Engineering
Czech Technical University in Prague
chreslea@fel.cvut.cz, pevnytom@fel.cvut.cz, antonin.komenda@fel.cvut.cz, stefan.edelkamp@fel.cvut.cz

Abstract

Learning a well-informed heuristic function for hard planning domains is an elusive problem. Although there are known neural network architectures to represent such heuristic knowledge, it is not obvious what concrete information is learned and whether techniques aimed at understanding the structure help in improving the quality of the heuristics.

This paper presents a network model that learns a heuristic function capable of relating distant parts of the state space via optimal plan imitation using the attention mechanism which drastically improves the learning of a good heuristic function. To counter the limitation of this method in the creation of problems of increasing difficulty, we demonstrate the use of curriculum learning, where newly solved problem instances are added to the training set, which, in turn, helps to solve problems of higher complexities and far exceeds the performances of all existing baselines including classical planning heuristics. We demonstrate its effectiveness on grid-type PDDL domains.

Introduction

Classical Planning has always relied on strong heuristic functions to approximate distances to the nearest goal (Bonet and Geffner 2001). Generally speaking, the quality of heuristics is measured by how well it performs when used inside a planner, i.e., it depends on the quality of the solution and the time taken to generate it. A major drawback of classical planning is the need to formulate problems by extensively capturing information from the environment. Recent years observe a progress in using visual representations to capture the specifics of a problem (Asai and Fukunaga 2017). Yet, there is still a big gap between the length of optimal plans and the plans found by planners using learnt heuristic functions.

A significant amount of importance is given to developing deep networks that are able to learn strong heuristics (Ernandes and Gori 2004) and policies (Torrey et al. 2006). This is particularly common in Reinforcement Learning which uses positive and negative feedback to learn the correct sequence of next actions. In learning for planning, there is no such provision; instead, it relies heavily on either hand-coded logical problem representations (Yoon, Fern, and Givan 2012) or deep convolution neural networks (Groshev et al. 2017) that imitate an expert. While there exists successful approaches in training neural networks to learn heuristic estimates of various problem domains (Arfaee, Zilles, and Holte 2011; Virseda, Borrajo, and Alcázar 2013; Groshev et al. 2017), designing a meaningful neural network architecture to extract the relevant information from the data set is still an open-ended problem.

This work extends the work by Schaal (1999) and Groshev et al. (2017) by addressing limitations of convolutional neural network and by using imitation and curriculum learning to train from plans of more difficult problem instances for hard planning domains. Specifically, we propose to use self-attention and position encoding (Vaswani et al. 2017), as we believe a strong heuristic function needs to relate "distant" parts of the state space.

In our default experimental settings, neural networks (NNs) realizing heuristic functions are trained on plans of small problem instances created by classical planners. While this allows us to generalize across more difficult instances (some of which are still solvable by classical planners) such that we can measure distances to optimal plan lengths, they do not achieve the best results for two reasons. First, even though the generalization of A*-NN is surprisingly good as will be seen below, there is still a scope of large-scale improvement on previously unseen, larger and more complex environments. Second, classical domain independent planners can solve only small problem instances anyway, which means that obtaining plans from large ones is difficult. We demonstrate that this problem can be partially mitigated by curriculum learning, where the NN is retrained / fine-tuned using plans from problems it has previously solved.

The proposed approach is compared to state-of-the-art domain-independent planners, namely SymBA*(Torralba et al. 2014), Lama (Richter and Westphal 2010), and Mercury (Katz and Hoffmann 2014) and to currently best combination of A* and CNNs (Groshev et al. 2017) on three grid domains: (1) **Sokoban** where each maze consists of walls, empty spaces, boxes, targets and an agent; the goal is to push the boxes to target locations; the boxes can only be pushed and not pulled in the game; (2) **Maze-with-Teleports** where the goal for an agent is to reach the goal position via interconnected teleports; (3) **Floor-Tile** where the tiles in a given maze are to be colored alternatively by agents (in our case, two agents) that are assigned a particular color each. At no stage can an agent step on a tile that has been colored.

The approach is domain-independent and only requires the selection of propositions in the PDDL file for spanning the underlying grid and the objects moving on it. It uses either policy or heuristic value heads.

We have chosen these domains because (1) all can be easily translated into an image based representation on which we can apply the convolution operation;[1] (2) Sokoban is PSPACE-complete (Culberson 1999) and known to be a challenging problem in Deep Learning for Planning (Fern, Khardon, and Tadepalli 2011); therefore, an improvement in Sokoban's policy and heuristic learning implies success in other two dimensional planning domains; (3) Maze-with-Teleports have non-local actions as the maze agent is being teleported to different parts of the maze via teleports; (4) Floor-Tile is NP-Complete and allows easy generation of mazes of increasing difficulty, which makes it a great choice of domain for improving the learnt heuristics via curriculum learning (Bengio et al. 2009).

The paper is organised as follows. We first discussed existing relevant research that has been carried out in deep learning for planning, especially in image based games. Next, we explore the formal basics of classical planning. Then, we highlight the shortcomings of a prior state of the art and propose a solution that addresses some of these shortcomings. Here, we introduce the basics of the attention mechanism from NLP and explain the role of positional encoding in learning distances. Then, the proposed networks are compared to other state of the art methods that attempt to solve Sokoban, Floor-Tile and Maze-with-Teleports. Finally, we conclude with an overall synopsis of our work and lay the ground for future work.

Related Work

The implementation of neural networks (NN) in learning policies and heuristics for deducing strategic positions or moves in various game domains has been studied extensively in the past. In Chess (Thrun 1994), NNs have been used to evaluate chess board functions from the outcome of various games. A combination of supervised learning and reinforcement learning has been implemented in games such as Go (Silver et al. 2017), which uses value networks to evaluate board positions and policy networks that choose successive actions. In Backgammon (Tesauro 2002), temporal difference learning was employed to train networks from millions of gameplay. Even single agent games such as Sokoban (Racanière et al. 2017) and Rubik's cube (Agostinelli et al. 2019) use NNs to solve the respective puzzles. The relative success of NNs in reinforcement learning has led researchers to believe that a similar success may be achieved in learning heuristic functions for various classical planning domains.

Fern, Khardon, and Tadepalli (2011) presented the earliest known work that combines planning and learning and concludes that the performance of NNs is promising in learning heuristic functions. The work of (Virseda, Borrajo, and Alcázar 2013) attempts to combine classical planning and deep learning by modifying costs and learning an improved heuristic function that generates good quality plans. Arfaee, Zilles, and Holte (2011) generated strong heuristics by repeatedly training on a set of weak heuristics by the bootstrapping procedure which are then used inside a classical planner. Neural search policies (Gomoluch et al. 2020) rely on parameter learning to generate heuristics during the actual search process. The learning procedure in (Srivastava, Immerman, and Zilberstein 2011) leads to generalizations of classical plans by identifying landmark actions that may be repeatedly applied during learning. Delphi applies deep learning on performance profile graphs to choose the cost-optimal planner in a portfolio (Sievers et al. 2019)

Asai and Fukunaga (2017) used classical planning to generate data and designs in an architecture that returns a visualised plan execution. In (Groshev et al. 2017)'s work, the training samples were generated by classical planning, and imitation learning (Schaal 1999) was performed on this data set for policy and heuristic learning. Unlike Fikes and Nilsson (1971) and Shavlik (1989) that used hand-coded domains to represent problems, Asai and Fukunaga (2017) and Groshev et al. (2017) learned useful information from the input data through image-based state descriptions.

Our approach significantly differs from prior work in planning and learning: (1) we do not require any[2] predesigned model of the problem domain or the state transition system; instead our model learns primarily from optimal plans; (2) our model uses attention mechanism with positional encoding designed to learn distances in the heuristic network; this generates near optimal solutions for complex problem instances where other techniques often fail.

Classical Planning

We construct our problem domains in a classical setting, i.e. fully observable and deterministic.

In classical planning, a STRIPS (Fikes and Nilsson 1971) planning task is defined by a tuple $\Pi = \langle F, A, I, G \rangle$. F denotes a set of facts which can hold in the environment (for instance, in Sokoban, a particular box at a particular position is a fact). A state s of the environment is defined as a set of facts holding in that particular s, i.e. $s \subseteq F$. The set of all states is, therefore, defined as all possible subsets of F as $S = 2^F$. $I \in S$ is the initial state of the problem and $G \subseteq F$ is a goal condition comprising facts which has to hold in a goal state. An action a, if applicable and applied, transforms a state s into a successor state s' denoted as $a(s) = s'$ (if the action is not applicable, we assume it returns a distinct failure value $a(s) = \perp$). All actions of the problem are contained in the action set A, i.e. $a \in A$. The sets S and A define the state-action transition system.

Let $\pi = (a_1, a_2, \ldots, a_l)$, we call π a plan of length l solving a planning task Π iff $a_l(\ldots a_2(a_1(I))\ldots) \supseteq G$. We assume a unit cost for all actions, therefore the plan length and plan cost are equal. Moreover, let π_s denote a plan from a state s, not I. An optimal solution (plan) is defined as a minimal length solution of a problem Π and is denoted as π^* together with its length $l^* = |\pi^*|$.

[1] If the domain can be described by graph, we can replace image convolution by graph-convolution, though we do expect the experimental results on these domains to exhibit very different behavior.

[2] Within the family of the grid domains.

A heuristic function h is defined as $h : S \to \mathbb{R}^{\geq 0}$ and provides an approximation of the optimal plan length from a state s to a goal state $s_g \supseteq G$, formally $h(s) \approx l^*$, where $l^* = |\pi_s^*|$.

In our experiments, we choose domains encoded in PDDL (Fox and Long 2003), where a planning problem is compactly represented in a lifted form based on predicates and operators. This representation is grounded into a STRIPS planning task Π, which is subsequently solved by the planner using a heuristic search navigating in the state-action transition system graph and resulting in a solution plan π.

Planner's Architecture

To learn a heuristic function for a planning domain that estimates the cost-to-go in a current state is one of the holy grails in AI (Edelkamp and Schrödl 2012; Mostow and Prieditis 1989).

In our approach, we combine a domain-independent planner with a trained neural network heuristic for an improved guidance during the overall search. This is how we bridge symbolic and sub-symbolic reasoning, given that neural networks are data-driven, and task planning requires symbolic reasoning in some form of logical calculus.

To bridge this gap, we highlight that any (deterministic) plan is a sequence of actions, but also on states. Given the initial state I, each partial plan $\pi = (a_1, a_2, \ldots, a_k)$, $k < l$ induces a sequence of states $(s_0 = I, s_1, s_2, \ldots, s_k)$ with $s_k = a_k(\ldots a_2(a_1(I))$.

Any encoded state is an input for the network. In our setting we use a *one-hot bit vector encoding* of the propositions. The output of the network is the heuristic value called *value head*, in some cases, together with a distribution of action to take, called *policy head*. For imitation learning, we use optimal plans for selecting training instances for the neural network, which might be generated by any optimal planner. More precisely, given the plans in the training set, we generate pairs $(s_i, \delta(s_i))$, where δ is cost of an optimal plan from s_i to the goal. For the sake of simplicity, $\delta(s_i)$ is the distance $l - i$ of the state s_i to the goal s_l in the optimal plan $(s_0 = I, s_1, s_2, \ldots, s_i, \ldots, s_l)$. Evaluating the network for a given state, directly serves as an estimator in our heuristic search planner. For curriculum learning, we use not only the optimal plans, but we also include newly found close-to-optimal plans to retrain the NN learner.

For some of our domains, we also take pairs of (s_{i-1}, a_i) to train the network for its policy head. Since our aim is finding (close-to-)optimal plans, we employed A* (Hart, Nilsson, and Raphael 1968) as the search algorithm for exploring the planning state space. For training the network, we run the known backpropagation algorithm on input batches together with stochastic gradient decent (Bottou and Bousquet 2007) to minimize the network error, that is computed with a simple loss function applied to the predicted and real value. Once trained, the heuristic estimate can be extracted efficiently for each state from the value head. In some domains, action selection can be based on the policy head.

Learning the Heuristic

This section describes the proposed neural network for planning in maze-like PDDL domains. But before, we introduce the notation and briefly discuss the state-of-the-art along with the proposed modification. In the end, we discuss the concrete architecture we used for all the benchmark domains (Sokoban, Floor-Tile and Maze-with-Teleports). We extract the grid layout automatically, but for the time being, we assume to have the grid dimensions h and w to define the network.

Formal notations for the proposed neural networks

The input to the neural network is denoted as $\vec{x} \in \mathbb{R}^{h,w,d_0}$, where h and w is the height and width of the maze respectively, and d_0 varies with the number of channels as explained above. Intermediate outputs are denoted by $\vec{z}^i = L_i(\vec{z}^{i-1})$, where L is some neural network layer (convolution C, attention A, or position encoding E) and for the sake of convenience, we set $\vec{z}^0 = \vec{x}$. All $\vec{z}$ are three dimensional tensors, i.e. $\vec{z}^i \in \mathbb{R}^{h,w,d_i}$. Notice that all intermediate outputs $\vec{z}^i$ have the same width and height as the maze (ensured by padding), while the third dimension which is the number of output filter(s) differs. Value $\vec{z}^i_{u,v}$ denotes a vector created from $\vec{z}^i$ as $(\vec{z}^i_{u,v,1}, \vec{z}^i_{u,v,2}, \ldots, \vec{z}^i_{u,v,d_i})$. Below, this vector will be called a *hidden vector* at position (u, v) and can be seen as a description of the properties of this position.

The proposed neural network

The best published NN implementing a heuristic function for Sokoban was proposed by Groshev et al. (2017). The network's shape resembled letter Y, as it has two heads, and it contains only convolution layers. The first seven convolution layers were shared (we call them pre-conv layer abbreviating preprocessing-convolution). Then, the network splits to yield two sets of outputs: (i) the estimate of the heuristic function and (ii) the policy. After the split, each path to the output contained seven convolution layers followed by two dense layers. Although the heuristic function should be sufficient for planning purposes, Groshev et al. (2017) state that training the network to estimate the policy helps in computing a better heuristic function.

Our criticism of the architecture is that convolution is strictly a local operator. This means that the hidden vector $z^{i+1}_{u,v,\cdot}$ is calculated from hidden vectors $\{z^i_{u',v',\cdot} | u' \in \{u-1, u, u+1\}, v' \in \{v-1, v, v+1\}\}$, where we assume convolution to have dimension 3×3 as in (Groshev et al. 2017). This limits the neural network in synthesizing information from two distant parts of the maze. Yet, we believe that any good heuristic requires such features, since Sokoban, Floor-Tile and Maze-with-Teleports are non-local problems.

Convolution, Attention, and Position Encoding

Therefore, we turn our attention to self-attention mechanism, (Vaswani et al. 2017) first introduced in NLP, as it allows to relate distant parts of input together. The output

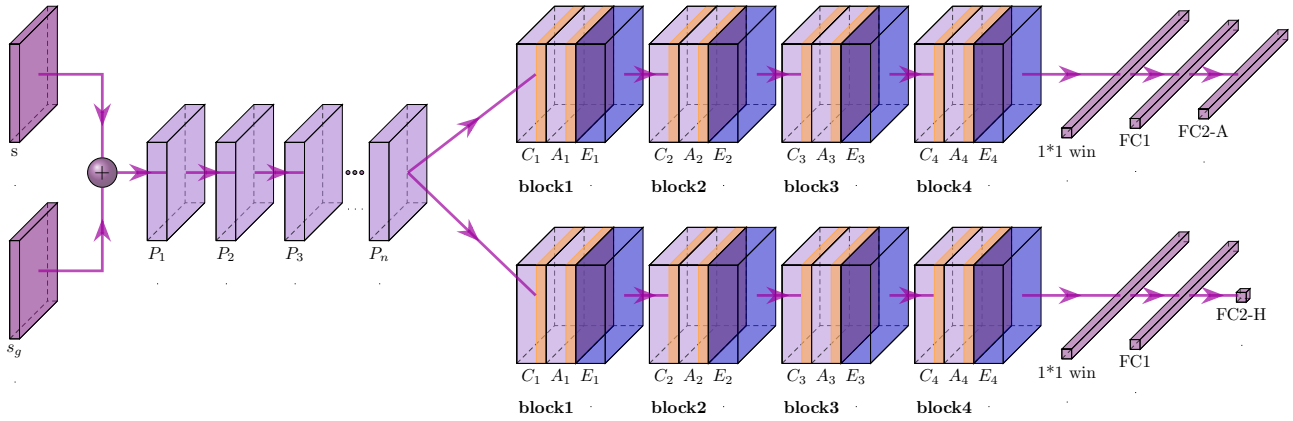

Figure 1: The structure of our neural network. A current state s and a goal state s_g are fed into a variable number of pre-processing convolution (pre-conv) layers, $P_1..P_n$. In our case, we use 7 pre-conv layers. All convolution filters in the pre-conv layers are of the same shape 3×3 with 64 filters. Then the network splits into two branches and each branch has four blocks, each block containing a convolution layer (C) followed by a multi head attention operation with 2 heads (A) and a positional encoding layer (E). There are 180 filters in each of these convolution layers in the blocks. At all stages, the original dimension of the input is preserved. The output from block 4 is flattened by applying a 1×1 window around the agent's location before being passed onto the fully connected layers (FC1) and the action prediction output (FC2-A) and a single output for heuristic prediction (FC2-H). For the sake of picture clarity, skip connections are not shown in the neural network.

of self-attention from z^i is calculated in the following manner. At first, the output from previous layer z^i is divided into three tensors of the same height, width, and depth, i.e.

$$\vec{k} = z^i_{\cdot,\cdot,j} \quad j \in \left\{1, \ldots, \frac{d_i}{3}\right\}$$
$$\vec{q} = z^i_{\cdot,\cdot,j} \quad j \in \left\{\frac{d_i}{3}+1, \ldots, \frac{2d_i}{3}\right\}$$
$$\vec{v} = z^i_{\cdot,\cdot,j} \quad j \in \left\{\frac{2d_i}{3}+1, \ldots, d_i\right\}$$

then, the output z^{i+1} at position (u, v) is calculated as

$$\vec{z}^{i+1}_{u,v} = \sum_{r=1,s=1}^{h,w} \frac{\exp(\vec{q}_{u,v} \cdot \vec{k}_{r,s})}{\sum_{r'=1,s'=1}^{h,w} \exp(\vec{q}_{u,v} \cdot \vec{k}_{r',s'})} \cdot v_{r,s} \quad (1)$$

Self attention, therefore, makes a hidden vector $z^{j+1}_{u,v}$ dependent on all hidden vectors $\{z^j_{r,s} | r \in \{1, \ldots, h\}, s \in \{1, \ldots, w\}\}$, which is aligned with our intention. The self-attention also preserves the size of the maze. A multi-head variant of self-attention means that z^i is split along the third dimension in multiple $\vec{k}$s, $\vec{q}$s, and $\vec{v}$s. The weighted sum is performed independent of each triple (k, q, z) and the resulting tensors are concatenated along the third dimension. We refer the reader for further details to (Vaswani et al. 2017).

While self-attention captures information from different parts of the maze, it does not have a sense of a distance. This implies that it cannot distinguish close and far neighborhoods. To address this issue, we add positional encoding, which augments the tensor $z^i \in \mathbb{R}^{h,w,d_i}$ with another tensor $\vec{e} \in \mathbb{R}^{h,w,d_e}$ containing outputs of harmonic functions along the third dimension. Harmonic functions were chosen, because of their linear composability properties (Vaswani et al. 2017)[3]. Because our mazes are two dimensional, the distances are split up into row and column distances where $p, q \in [0, d_i/4)$ assigns positions with sine values at even indexes and cosine values at odd indexes. The positional encoding tensor $\vec{e} \in \mathbb{R}^{h,w,d_e}$ has elements equal to

$$\vec{e}_{u,v,2p} = \sin(\theta(p)u) \qquad \vec{e}_{u,v,2p+1} = \cos(\theta(p)u)$$
$$\vec{e}_{u,v,2q+\frac{d_e}{2}} = \sin(\theta(q)v) \qquad \vec{e}_{u,v,2q+1+\frac{d_e}{2}} = \cos(\theta(q)v),$$

where $\theta(p) = \frac{1}{10000^{\frac{4p}{d_e}}}$. On appending this tensor to the input z^i along the third dimension, we get

$$z^{i+1}_{u,v,\cdot} = [z^i_{u,v,\cdot}, e_{u,v,\cdot}].$$

With respect to the above, we propose using blocks combining Convolution, Attention, and Position encoding, in this order (we call them CoAt blocks), as a part of our NN architecture. The CoAt blocks can therefore relate hidden vectors from a local neighborhood through convolution, from a distant part of the maze through attention, and calculate distances between them through position encoding, as has been explained in (Tsai et al. 2019). Since CoAt blocks preserve the size of the maze,[4] they are "scale-free" in the sense that they can be used on a maze of any size. Yet, we need to provide an output of a fixed dimension to estimate the heuristic

[3]The composability of harmonic functions is based on the following property $\cos(\theta_1 + \theta_2) = \cos(\theta_1)\cos(\theta_2) - \sin(\theta_1)\sin(\theta_2) = (\cos(\theta_1), \sin(\theta_1)) \cdot (\sin(\theta_1), \sin(\theta_2))$, where $\cdot$ denotes the inner product of two vectors, which appears in Equation (1) in inner product of $\vec{q}_{u,v}$ and $\vec{k}_{r,s}$.

[4]Convolution layers are appropriately padded to preserve sizes.

function and the policy. The output of the last CoAt block is flattened by a 1×1 window, centered around the agent's location and fed to a fully-connected layer and an output head (see Figure 1). For example, assuming z^L to be the very last layer, and agent is on position u, v, the vector $z_{u,v,\cdot}$ is of constant dimension equal to the number of channels and is used as an input to the fully-connected layers providing the desired outputs (heuristic values, policy).

Next, we describe the implementation of CoAt blocks in the network architectures we used for all the domains. The network is shown in Figure 1 and its structure is similar to that of (Groshev et al. 2017). It uses preprocessing convolution layers $P_1, \ldots, P_n$, $n = 7$, (further called *pre-conv*) containing 64 filters where each convolution filter is of the shape 3×3; after the network splits into two heads, it uses four CoAt blocks in each head instead of seven convolution layers used in (Groshev et al. 2017). The convolution layers in the CoAt blocks contain 180 filters of size 3×3 each. The attention block uses two attention heads. Each head is finished by two fully-connected layers with reduction to a fixed dimension as described above.

The input to the network is the current state of the game and a goal state, s and s_g, respectively. Each state is represented by a tensor of dimensions equal to width and height (fixed to 10×10 for Sokoban, to 15×15 for Maze-with-Teleorts, and to 4×4 for Floor-Tile) of the maze $\times$ objects. The objects stands for one-hot encoding of the object states on a grid position (e.g., for Sokoban, we have wall, empty, box, agent and box target, for Maze-with-Teleport agent, wall, floor, goal and teleports 1-4, and for Floor-Tile agent1, agent2, black and white), which we could derive automatically from the grounded representation. An important design detail is that all convolutions are padded, which means that the output has the same dimension as the input, and they feature skip-connections[5] alleviating a possible vanishing gradient (He et al. 2016).

We have implemented two different versions of the network according to their heads: dual-head estimating policy and heuristic value and a single-head estimating the heuristic value. In Maze-with-Teleports and Sokoban, the dual-head representation performed best, while the best network for Floor-tile uses just a single head estimating heuristic value (which means there is no separate head estimating the policy). The presence of two agents would make it difficult to design a policy network in a domain-independent setting and would result in a much larger network, which is inconvenient, time-consuming and computationally expensive. The other implementation details of the Floor-Tile network are the same as the heuristic networks of Sokoban and Maze-with-Teleports (see Figure 1).

Imitation and Curriculum Learning

Imitation learning (Pomerleau 1989) is a framework for learning a behavior policy from demonstrations. We present demonstrations in the form of optimal state-action plans, with each pair indicating the action to take at the state being visited. Generally, imitation learning is useful when it is easier for an expert to demonstrate the desired behaviour rather than to specify a reward function which would generate the same behaviour or to directly learn the policy.

A curriculum refers to an interactive system of instruction and learning with specific goals, contents, strategies, measurement, and resources. The desired outcome of curriculum is a successful transfer and/or development of knowledge, skills, and attitudes. In the context of AI, curriculum learning is a way of training a machine learning model where more difficult aspects of a problem are gradually introduced in such a way that the model is always optimally challenged.

Curriculum learning (Elman 1993) describes a type of learning in which we first start out with only easy examples of a task and then gradually increase the task difficulty. Humans have been learning according to this principle ever since, but in the common learning setting, we train the neural network on the whole data set.

Curriculum learning strategies have been successfully employed in different areas of machine learning, for a wider range of tasks (Bengio et al. 2009). However, the necessity of finding a way to rank the samples from easy to hard, as well as the right pacing function for introducing more difficult data can limit the usage of the curriculum approaches.

In order to extend the training set without providing any additional plans that the neural network would not be able to solve, we turn our attention to a form of curriculum learning for neural networks. This approach partially circumvents this problem by re-training from unseen test samples of increasing complexity.

In our case, curriculum learning is used to develop scale-free heuristic values for a wider selection of AI planning problems. Specifically, in our experiments, we have quickly reached the capability of planners at larger sizes. To further improve our heuristic function to scale to bigger problems, we re-train our network by extending the training set to include harder problem instances.

We first train the heuristic network on a training set containing easy problem instances quickly solvable by an optimal planner, then use this NN as a heuristic function inside A*, and then extend the training set by more difficult problem instance the NN has solved and finally, re-train the NN. Thus, we perform a bootstrap, where the NN is gradually trained on more difficult problem instances.

This way, curriculum learning plays an important role in improving the performance of the heuristic network on not just the trained dimensions but also on higher dimensions by extrapolation. For curriculum learning, the learning rate is reduced in successive training iterations.

Experimental Results

This section briefly describes the details of training and presents the experimental results on the compared PDDL benchmark domains: Sokoban, Maze-with-Teleports, and Floor-Tile. A* algorithms with learnt heuristic functions realized by the proposed convolution-attention-position networks (further denoted as A*-CoAt) are compared to A*

[5] A layer c augmented by a skip connection calculates the output as $x + c(x)$ instead of the usual x.

with learned heuristic function realized by convolutional networks as proposed in (Groshev et al. 2017) (denoted as A*-CNN), and to the state of the art planners LAMA (Richter and Westphal 2010), SymBA* (Torralba et al. 2014), and Mercury (Katz and Hoffmann 2014). We emphasize that A*-CNN and A*-CoAt uses vanilla A* search algorithm (Hart, Nilsson, and Raphael 1968) without any additional tweaks. In case of Sokoban, we also compare our planner to a solution based on Reinforcement Learning (Racanière et al. 2017).

On all the compared domains, we analyse the strength of our learnt heuristic and generalization property by solving grid mazes of increasing complexities, approximated by the number of boxes in Sokoban, grids of higher dimensions in Floor-Tile and Maze-with-Teleports, and rotated mazes in Maze-with-Teleports.

Training

Sokoban: The policy-heuristic network we wish to learn accepts a state of a game, s as an input and returns the next action, a, and a heuristic value, $h(s)$, as an output. The training set $\mathcal{X}_{\text{trn}} = \{(s_i, a_i, |\pi^*(s_i)|)\}_{i=1}^{n}$. therefore consists of $n \approx 10^6$ of these triples, i.e. $\mathcal{X}_{\text{trn}} = \{(s_i, a_i, |\pi^*(s_i)|)\}_{i=1}^{n}$. The triples in the training set were created by randomly generating 40000 Sokoban instances using gym-sokoban (Schrader 2018). Each instance has dimension 10×10 and it always contains only 3 boxes (and an agent). SymBA* (Torralba et al. 2014), a planner that generates optimal solutions, was used to generate optimal plans π^* for each of these *n* Sokoban instances. In each plan trajectory, the distance from a current state to the goal state is learned as the heuristic value, $h(s_i)$. Thus, the collection of all state-action-heuristic triples form the training set $\mathcal{X}_{\text{trn}}$.

The Sokoban mazes in the *training set were created with only three boxes*. This means that in the testing set, when we are solving instances with more boxes, we are evaluating its *extrapolation* to more complex unseen environments, which cannot be solved by naive memorisation. However, our limited training set (containing 3 boxes) hinders the full potential of the neural network. With curriculum learning, we fine-tune the neural network using a training set containing Sokoban mazes of dimensions 10×10 with 3,4 5 6 and 7 boxes that have been already solved by the A*-NN with the corresponding architecture. This, therefore improves the heuristic function without the need to train the network from scratch and, more importantly, without the need to use other planners to create new plans.

Maze-with-Teleports: The policy and heuristic network returns an action a, and a heuristic value, $h(s)$ as outputs. The set $\mathcal{X}_{\text{trn}}$consists of training triplets from about 10000 maze problems of dimension 15×15. The mazes in $\mathcal{X}_{\text{trn}}$were generated using a maze creator[6] by breaking random walls. We added a total of 4 pairs of teleports that connect different parts of the maze inside each training sample. As in the case of Sokoban, SymBA* (Torralba et al. 2014) was used to generate optimal solutions for the problems in the training set. The mazes for training were generated such that the initial position of the agent was in the upper-left corner and the goal was in the lower-right corner. Later, in our evaluations, we rotate each maze to investigate whether the heuristic function is rotation independent.

Floor-Tile: The Floor-Tile heuristic network accepts a state of a game, s as an input and returns a heuristic value, $h(s)$ as an output. The training set initially consists of triplets from about 10000 Floor-Tile instances of dimension 4×4 with 2 agents. In our version of Floor-Tile, we assign *white* to the first agent and *black* to the second agent. The colors assigned to the agents are fixed and cannot be flipped at any stage of the game. SymBA* (Torralba et al. 2014) was used to generate optimal solutions for the Floor-Tile domain.

Since it is easy to generate Floor-Tile instances of increasing complexity (by increasing the size and varying the initial positions of the two agents), we experiment again with curriculum learning (Bengio et al. 2009) to develop scale-free heuristic values. Specifically, in our experiments we have quickly reached the capability of planners at size 6×5 (more on this below). To further improve our heuristic function to scale to bigger problems, we re-train our network by extending the training set to include harder problem instances (of size 4×4, 5×5, 6×5 and 6×6) the heuristic network has already solved.

All neural networks were trained by the Adam optimiser (Kingma and Ba 2014) with a default learning rate of 0.001 for optimisation. In Sokoban and Maze-with-Teleports, the categorical cross entropy loss function was used to minimise the loss in the action prediction network and the mean absolute error loss was the loss function in the heuristic network. In Floor-Tile, the mean absolute loss was used in the heuristic network. For curriculum learning, the learning rate was reduced to about 1×10^{-4} in successive training iterations. Our experiments were conducted in Keras-2.4.3 framework with Tensorflow-2.3.1 as the backend. We used a NVIDIA Tesla GPU model V100-SXM2-32GB for training the neural networks.

Comparison to prior State-of-the-Art

Sokoban: The evaluation set consists of 2000 mazes of dimensions 10×10 with 3, 4, 5, 6 or 7 boxes (recall that the training set contain mazes with only 3 boxes). Unless said otherwise, the quality of heuristics is measured by the relative number of solved mazes, which is also known as *coverage*. Table 1 shows the coverage of compared planners, where *all* planners were given 10 minutes to solve each Sokoban instance. We see that the classical planners solved all test mazes with three and four boxes but as the number of boxes increase, the A*-NN starts to have an edge. On problem instances with six and seven boxes, A*-CoAt achieved the best performance, even though it was trained only on mazes with three boxes. The same table shows, that A*-CoAt offers better coverage than A*-CNN, and we can also observe that curriculum learning (see column captioned curr.) significantly improves the coverage.

We attribute SymBA*'s poor performance to its feature of always returning optimal plans while we are content with

[6]https://github.com/ravenkls/Maze-Generator-and-Solver

#b	SBA*	Mrcy	LAMA	normal CNN	normal CoAt	curr. CoAt
3	**1**	**1**	**1**	0.92	0.94	0.95
4	**1**	**1**	**1**	0.87	0.91	0.93
5	0.95	0.75	0.89	0.83	0.89	0.91
6	0.69	0.60	0.65	0.69	**0.76**	**0.85**
7	0.45	0.24	0.32	0.58	**0.63**	**0.80**

Table 1: Fraction of solved Sokoban mazes (coverage, higher is better) of SymBA* (SBA*), Mercury (Mrcy), LAMA, A*-CNN (caption CNN) and the proposed A*-CoAt (caption CoAt). A*-CNN and A*-CoAt (with caption normal) use networks trained on mazes with three bozes; A*-CoAt (with caption curr.) used curriculum learning.

#b	SBA*	Mrcy	CNN	CoAt
3	21.40	21.70	24.20	22.20
4	34.00	34.33	40.53	36.00
5	38.82	42.83	45.52	39.11
6	41.11	-	51.00	42.11
7	-	-	54.33	44.17

Table 2: Average plan length of SymBA*, Mercury, A* - CNN (Groshev et al. 2017) (denoted as CNN) and that with the A*-CoAt. For clarity, we do not show results of LAMA, as it is performs exactly like SymBA* for 3 and 4 boxes. Column captioned #b indicates the number of boxes in different categories.

sub-optimal plans. LAMA had even lower success in solving more complicated mazes than SymBA*, despite having the option to output sub-optimal plans. To conclude, with an increase in the complexity of the mazes, the neural networks outshine the classical planners which makes them a useful alternative in the Sokoban domain.

The average plan length, shown in Table 2, reveals that the heuristic learnt by the CoAt network is strong, as the average length of the plans is close to that of SymBA* which always returns optimal solutions. We conclude that the proposed CoAt network delivers a strong heuristic outside its training, much better than that of the CNN (Groshev et al. 2017) network and the planners (for mazes with more than 6 boxes).

CoAt network is also on par with Deep Mind's implementation of Reinforcement Learning (DM-RL) in solving Sokoban (Racanière et al. 2017). Instead of re-implementing DM-RL by ourselves, we report the results on their test set[7] containing 10×10 Sokoban mazes with 4 boxes. While DM-RL had a coverage of 90%, our A*-CoAt (trained on mazes with three boxes) has a coverage 87%, and our A*-CoAt with curriculum learning has a coverage of 98.29% [8]. Taking into account that DM-RL's training set contained 10^{10} state-action pairs from mazes **with 4 boxes**, A*-CoAt achieves higher coverage using several orders of magnitude smaller training set.

Maze-with-Teleports: The evaluation set contains a total of 2100 training samples of dimensions 15×15, 20×20, 30×30, 40×40, 50×50, 55×55 and 60×60. Each maze in the evaluation set contains 4 pairs of teleports that connects different parts of the maze. From Table 3, we see that the performance of A*-CNN and A*-CoAt (initially trained on 15×15 mazes) is the same as SymBA*[9] for dimensions up to 40×40 and is consistently better for problem instances of size 50×50, 55×55 and 60×60.

All "No Rotation" mazes were created such that the agents start in the top left corner and the goal is in the bottom right corner. This allows us to study to which extent the learnt heuristic is rotation-independent (domain independent planners are rotation invariant by default). The same Table therefore reports fraction of solved mazes that have been rotated by 90°, 180° and 270°. The results clearly show that the proposed heuristic function featuring CoAt blocks generalizes better than the one utilizing only convolutions, as the solved rotated instances of A*-CoAt network are comparable to the non-rotated case. Rotating mazes have no effect on SymBA* (the complexity is solely dependent on the grid size) and the coverage rate stays unaffected.

From the results in Table 3, it can be concluded that the CoAt blocks (1) improve detection of non-local actions (teleports) compared to state-of-the-art planners such as SymBA*; (2) learn 'useful' information from the mazes which makes the network robust to rotations; (3) learn to approximate distances inside the mazes which results in a scale-free heuristic function.

Floor-Tile: The evaluation set consists of 400 problem instances of sizes 5×5, 6×5, 6×6 and 7×7. Similarly to Sokoban, we first trained the CNN and CoAt NNs on small problem instances of size 4×4 (see columns denoted as "normal") and extrapolated it to instances of higher dimensions outside of the training set. Table 4 shows the coverage of SymBA* and the heuristics learnt by the NNs inside A* for different problem instances. The results are similar to those we have observed in Sokoban. On smaller problem instances, the classical planner SymBA* is better; on larger problem instances, the NNs are better. The test set deliberately includes problem instances of size 6×5 to demonstrate the exact break point up to which SymBA* is able to generate solutions. Beyond grid size 6×5, *all* state-of-the-art planners fail and the solutions are generated only by the NNs. The heuristics generated by the A*-CoAt network can be extrapolated to solve 71% of the tiling problems of size 6×6. On further increasing the grid dimension, the coverage of the A*-CoAt decreases to 29% and even lower while A*-CNN is unable to generate solutions. The rightmost column (see column captioned "curr") in Table 4 shows an improvement in coverage for dimensions 6×6 and 7×7 on implementing curriculum learning.

[7] Available at https://github.com/deepmind/boxoban-levels.

[8] https://github.com/deepmind/boxoban-levels/blob/master/unfiltered/test/000.txt

[9] The planners and NNs were given 10 minutes to solve each maze instance.

size	No Rotation			90° rotation		180° rotation		270° rotation	
	SBA*	CNN	CoAt	CNN	CoAt	CNN	CoAt	CNN	CoAt
15×15	1	1	1	1	1	1	1	1	1
20×20	1	1	1	1	1	1	1	1	1
30×30	1	1	1	1	1	1	1	1	1
40×40	1	1	1	1	1	1	1	1	1
50×50	0.92	0.94	**1**	0.91	1	0.92	1	0.91	1
55×55	0.55	0.78	**0.89**	0.71	0.85	0.70	0.87	0.69	0.87
60×60	-	0.73	**0.76**	0.68	0.75	0.66	0.74	0.68	0.75

Table 3: Fraction of solved mazes with teleports (coverage) of SymBA*, A* algorithm with convolution network (Groshev et al. 2017) (denoted as CNN) and that with the proposed Convolution-Position-Attention (CoAt) network. Only non-rotated mazes (No Rotation) of size 15×15 were used to train the heuristic function. On mazes rotated by $90°$, $180°$, $270°$, the heuristic function has to extrapolate outside its training set.

size	SBA*	normal		curr.
		CNN	CoAt	CoAt
4×4	**1**	0.92	0.96	0.96
5×5	**1**	0.89	0.93	0.93
6×5	**1**	0.78	0.88	0.91
6×6	-	0.54	**0.71**	**0.89**
7×7	-	-	**0.29**	**0.76**

Table 4: Fraction of solved Floor-Tile problem instances (coverage) of SymBA*, A*-CNN, and the proposed A*-CoAt. Fractions in columns captioned "normal" / "curr." are of A* with heuristic functions trained on problem instances of size 4×4 / by curriculum training respectively (see details in the text).

Conclusion and Future Work

We showed that learning a strong heuristic function for PDDL domains with an underlying grid structure is possible without the need for any specific domain knowledge. In fact, the architecture of the learning approach is general to any PDDL-type planning problem, and the learning can be executed on any vector of propositions, or finite-domain variables as states forming input to the neural network. Identifying some structure(s) in advance is advantageous.

While the heuristic function can generate sub-optimal plans, our experiments suggest that the plan quality is not far from the optimum. Moreover, while we have generated training data from a classical planner on small problem sizes, the proposed architecture is able to generalize and successfully solve more difficult problem instances, where it surpasses classical domain-independent planners, while improving on previously known state-of-the-art.

Our experiments further suggest that the learnt heuristic can further improve, if it is retrained / fine-tuned on problem instances it has previously solved. This form of curriculum learning aids the heuristic function in solving mainly large and more complex problem instances that are otherwise not solvable by domain independent planners within 10 minutes.

As future work, our next goal would be to better understand if the learnt heuristic function is similar to something that is already known, or something so novel that it can further enrich the field; i.e., what kind of underlying problem structure we can learn by which network type, possibly in form of studying generic types (Long and Fox 2000).

Our experiments with curriculum learning suggest that the neural network might boot-strap itself by first learning on simple trivial examples and gradually solving more difficult ones by adding them to the training set. This raises two questions, mainly regarding the limit of this process and if the gains obtained by using larger networks will vanish.

We believe that an improvement in the heuristic function is tied to the generation of problem instances that inherently possess the right level of difficulty, by which we mean that they have to be just on the edge of solvability, such that the plan can be created and added to the training set. We are fully aware that the problem instance generation itself is a hard problem, but we cannot imagine the above solution to be better than specialized domain-dependent Sokoban solvers without such a generator (unless the collection of all Sokoban mazes posses this property). We also question the average estimation errors minimized during learning of the heuristic function. It might put too much emphasis on simple problem instances that are already abundant in the training set while neglecting the difficult ones. We wish to answer some of the above question in the future in an endeavour to generate strong, scale-free heuristics.

Acknowledgement

The research leading to this paper has received funding from OP VVV project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics" and from Czech Ministry of Education 19-29680L

References

Agostinelli, F.; McAleer, S.; Shmakov, A.; and Baldi, P. 2019. Solving the Rubik's cube with deep reinforcement learning and search. *Nature Machine Intelligence*, 1(8): 356–363.

Arfaee, S. J.; Zilles, S.; and Holte, R. C. 2011. Learning heuristic functions for large state spaces. *Artificial Intelligence*, 175(16-17): 2075–2098.

Asai, M.; and Fukunaga, A. 2017. Classical planning in deep latent space: Bridging the subsymbolic-symbolic boundary. *arXiv preprint arXiv:1705.00154*.

Bengio, Y.; Louradour, J.; Collobert, R.; and Weston, J. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, 41–48.

Bonet, B.; and Geffner, H. 2001. Planning as heuristic search. *Artificial Intelligence. 2001 Jun; 129 (1-2): 5-33.*

Bottou, L.; and Bousquet, O. 2007. The Tradeoffs of Large Scale Learning. In Platt, J. C.; Koller, D.; Singer, Y.; and Roweis, S. T., eds., *NIPS*, 161–168.

Culberson, J. 1999. Sokoban is PSPACE-complete. *Proceedings of the International Conference on Fun with Algorithms*, 65–76.

Edelkamp, S.; and Schrödl, S. 2012. *Heuristic Search - Theory and Applications*. Academic Press.

Elman, J. L. 1993. Learning and development in neural networks: the importance of starting small. *Cognition*, 48(1): 71–99.

Ernandes, M.; and Gori, M. 2004. Likely-admissible and sub-symbolic heuristics. In *Proceedings of the 16th European Conference on Artificial Intelligence*, 613–617. Citeseer.

Fern, A.; Khardon, R.; and Tadepalli, P. 2011. The first learning track of the international planning competition. *Machine Learning*, 84: 81–107.

Fikes, R. E.; and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial intelligence*, 2(3-4): 189–208.

Fox, M.; and Long, D. 2003. PDDL2. 1: An extension to PDDL for expressing temporal planning domains. *Journal of artificial intelligence research*, 20: 61–124.

Gomoluch, P.; Alrajeh, D.; Russo, A.; and Bucchiarone, A. 2020. Learning Neural Search Policies for Classical Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, 522–530.

Groshev, E.; Goldstein, M.; Tamar, A.; Srivastava, S.; and Abbeel, P. 2017. Learning generalized reactive policies using deep neural networks. *arXiv preprint arXiv:1708.07280*.

Hart, P. E.; Nilsson, N. J.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2): 100–107.

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.

Katz, M.; and Hoffmann, J. 2014. Mercury planner: Pushing the limits of partial delete relaxation. *IPC 2014 planner abstracts*, 43–47.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Long, D.; and Fox, M. 2000. Automatic Synthesis and Use of Generic Types in Planning. In *AAAI*, 196–205. AAAI Press.

Mostow, J.; and Prieditis, A. 1989. Discovering Admissible Heuristics by Abstracting and Optimizing: A Transformational Approach. In *IJCAI*.

Pomerleau, D. A. 1989. Alvinn: An autonomous land vehicle in a neural network. In *Advances in neural information processing systems*, 305–313.

Racanière, S.; Weber, T.; Reichert, D.; Buesing, L.; Guez, A.; Jimenez Rezende, D.; Puigdomènech Badia, A.; Vinyals, O.; Heess, N.; Li, Y.; et al. 2017. Imagination-augmented agents for deep reinforcement learning. *Advances in neural information processing systems*, 30: 5690–5701.

Richter, S.; and Westphal, M. 2010. The LAMA planner: Guiding cost-based anytime planning with landmarks. *Journal of Artificial Intelligence Research*, 39: 127–177.

Schaal, S. 1999. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6): 233–242.

Schrader, M.-P. B. 2018. gym-sokoban. https://github.com/mpSchrader/gym-sokoban.

Shavlik, J. W. 1989. Acquiring Recursive Concepts with Explanation-Based Learning. In *IJCAI*, 688–693. Citeseer.

Sievers, S.; Katz, M.; Sohrabi, S.; Samulowitz, H.; and Ferber, P. 2019. Deep Learning for Cost-Optimal Planning: Task-Dependent Planner Selection. In *AAAI*, 7715–7723. AAAI Press.

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 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.

Tesauro, G. 2002. Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1-2): 181–199.

Thrun, S. 1994. Learning to play the game of chess. *Advances in neural information processing systems*, 7: 1069–1076.

Torralba, A.; Alcázar, V.; Borrajo, D.; Kissmann, P.; and Edelkamp, S. 2014. SymBA*: A symbolic bidirectional A* planner. In *International Planning Competition*, 105–108.

Torrey, L.; Shavlik, J.; Walker, T.; and Maclin, R. 2006. Skill acquisition via transfer learning and advice taking. In *European Conference on Machine Learning*, 425–436. Springer.

Tsai, Y.-H. H.; Bai, S.; Yamada, M.; Morency, L.-P.; and Salakhutdinov, R. 2019. Transformer Dissection: An Unified Understanding for Transformer's Attention via the Lens of Kernel. *arXiv preprint arXiv:1908.11775*.

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. *Advances in neural information processing systems*, 30: 5998–6008.

Virseda, J.; Borrajo, D.; and Alcázar, V. 2013. Learning heuristic functions for cost-based planning. *Planning and Learning*, 6.

Yoon, S. W.; Fern, A.; and Givan, R. 2012. Inductive policy selection for first-order MDPs. *arXiv preprint arXiv:1301.0614*.