Proceedings of the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS 2019)

Learning Heuristic Functions for Mobile Robot Path Planning Using Deep Neural Networks

Takeshi Takahashi*
University of Massachusetts Amherst
Amherst, MA 01003, USA
ttakahas@cs.umass.edu

He Sun*
Princeton University
Princeton, NJ 08544, USA
hesun@princeton.edu

Dong Tian*
InterDigital, Inc.
Princeton, NJ 08540, USA
dong.tian@interdigital.com

Yebin Wang
Mitsubishi Electric Research Laboratories
Cambridge, MA 02139, USA
yebinwang@merl.com

Abstract

Resorting to certain heuristic functions to guide the search, the computational efficiency of prevailing path planning algorithms, such as A*, D*, and their variants, is solely determined by how well the heuristic function approximates the true path cost. In this study, we propose a novel approach to learning heuristic functions using a deep neural network (DNN) to improve the computational efficiency. Even though DNNs have been widely used for object segmentation, natural language processing, and perception, their role in helping to solve path planning problems has not been well investigated. This work shows how DNNs can be applied to path planning and what kind of loss functions are suitable for learning such a heuristic. Our preliminary results show that an appropriately designed and trained DNN can learn a heuristic that effectively guides prevailing path planning algorithms.

Introduction

Efficient path planning is required for many applications, for example, self-driving cars and home service robots. A variety of path planning algorithms has been proposed, yet heuristic learning using deep neural networks (DNNs) have not been studied well. Deep learning has been successful in a variety of applications, such as object recognition (Krizhevsky, Sutskever, and Hinton 2012), video games (Mnih et al. 2015), and image generation (Goodfellow et al. 2014). Inspired by the object segmentation and the image generation algorithms, such as U-Net (Ronneberger, Fischer, and Brox 2015) and generative adversarial network (GAN) (Goodfellow et al. 2014), we investigate how to apply these techniques to path planning. We focus on learning a heuristic function instead of taking end-to-end approaches that map sensory inputs to actions directly so that we can still use the current well-developed search-based path planning algorithms. This paper shows a potential direction of heuristic learning for path planning. Our contributions are as follows:

- We applied a neural network architecture commonly used in semantic segmentation in order to learn heuristic functions for path planning.
- We investigated loss functions that are suitable for learning heuristic functions.
- We showed that the learned heuristic functions can reduce the search cost in many scenarios in two domains: 2D grid world and continuous domain.

The overall framework is outlined in Figure 1 and is explained in the "Proposed methods" section.

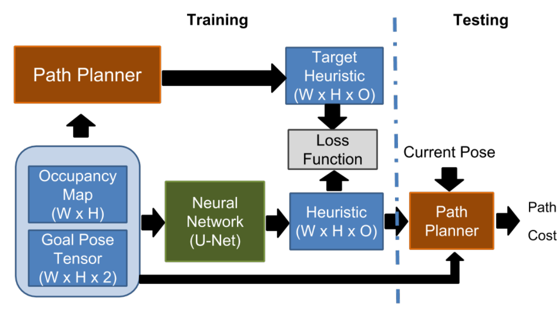


Figure 1: Learning heuristic functions for path planning.

Related Work

Path Planners

A variety of search algorithms have been proposed ,such as A* (Hart, Nilsson, and Raphael 1968), Anytime Repairing A* (ARA*) (Likhachev, Gordon, and Thrun 2004), Rapidly-exploring Random Tree (RRT) (LaValle 1998), Hybrid-A* (Dolgov et al. 2008), harmonic function path planning (Connolly and Grupen 1993), and two-stage RRT (Wang, Jha, and Akemi 2017). Heuristic-based methods often use admissible handcrafted heuristic functions, such as Manhattan distance, Euclidean distance, and length of Reeds-Shepp paths. Cost functions of the states also can be created from the topology of the state (Mahadevan and Maggioni 2007; Connolly and Grupen 1993) and can be learned from demonstration (Ratliff, Bagnell, and Zinkevich 2006). Although optimality can be achieved with admissible heuristics, the complexity can be beyond control with complex environments. ARA* uses a scaling factor to reduce the complexity by adjusting the upper bound of the cost of a path. Hybrid A* uses the maximum of two different admissible heuristics to guide grid cell expansion, although it does not preserve

*Work done at Mitsubishi Electric Research Laboratories

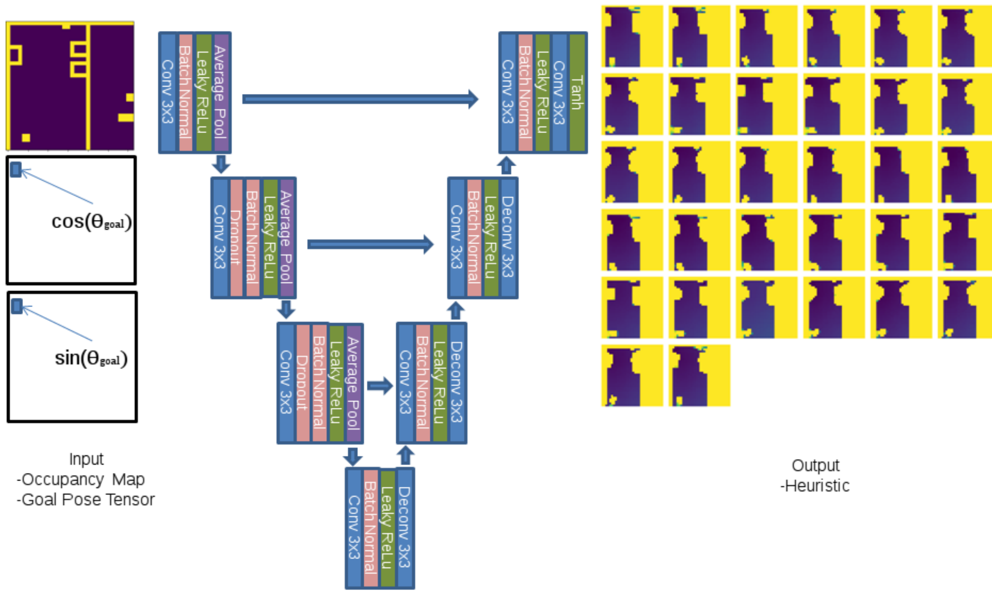


Figure 2: Network architecture used in our experiments (continuous domain).

completeness and optimality. Two-stage RRT (Wang, Jha, and Akemi 2017) uses two RRTs. An upper RRT ignores robot kinematics and produces waypoints, where guide the other lower RRT addressing the robot kinematics.

Navigation with Neural Networks

The recent advancement of DNNs enables learning navigation directly from high-dimensional sensory inputs. Simultaneous Localization and Mapping (SLAM) using neural networks can learn grid-like representations that appear in animal brains (Banino et al. 2018). Imitation learning with DNNs can be used for a real autonomous car to drive off-road at a high speed using raw RGB images with wheel speeds. Model Predictive Control (MPC) were used to provide training data (Pan et al. 2018). Deep reinforcement learning has been used to learn a function that maps raw sensory inputs (images) to actions. However, state-of-the-art algorithms have only been tested in simple domains with little variation of the environment (Tamar et al. 2016; Khan et al. 2017; Panov, Yakovlev, and Suvorov 2018), which is mainly due to the necessity of very long training times. Tamer *et al.* propose Value Iteration Networks (VIN) for value iteration where the learned control dynamics are represented by convolution kernels. VINs produces a better policy than the reactive one learned by standard convolutional neural networks (Tamar et al. 2016). Khan *et al.* combine VINs with differential neural computing to learn navigation (Khan et al. 2017). Panov *et al.* apply Deep Q-Network (DQN) to solve 2D grid navigation using a reward that is a function of optimal paths (Panov, Yakovlev, and Suvorov 2018). Zhang *et al.* use neural networks with external memory for SLAM (Zhang et al. 2017) to help a reinforcement learning algorithm to solve navigation tasks. Imitation learning with neural networks is used to reduce the search cost (Bhardwaj, Choudhury, and Scherer 2017). In their study, a heuristic policy was learned for each domain separately. The algorithm was tested on simple domains but relatively high dimensional states with the fixed start and goal. Once start positions, goal positions, and environments are randomized, local features may not be able to guide the planner to a goal efficiently, as discussed in (Dhiman et al. 2018). On the contrary, we learn a heuristic function defined over the entire environment given the goal configuration and the map to avoid the misguiding problem. In addition, we need only one feed-forward computation of a neural network to obtain the heuristic function. In our toy domain, we learn a heuristic function that can be applied to six different environments rather than a specific environment. We also consider environments where a path does not exist to investigate completeness, whereas the work on navigation with deep learning cited above often considers only environments where a path always exists.

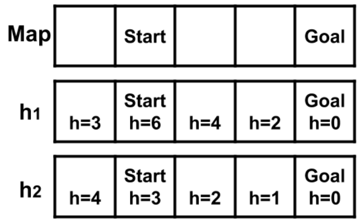
Proposed Methods

Task

In the problem for this study, a robot finds a path $\pi : [0,1] \mapsto C_{free}$ from a start configuration $\pi(0) := s_0 \in C_{free}$ to a goal configuration $\pi(1) := s_{goal} \in C_{free}$ given a map of an environment m and a heuristic function h where C_{free} is free space. Our goal is to learn a heuristic function h using DNNs to reduce the search cost. Our overall approach is shown in Figure 1. We will first introduce the network architecture we adopted and then explain the loss functions and training procedures.

Network Architecture

In this study, we adopted U-Net (Ronneberger, Fischer, and Brox 2015), which was initially proposed for semantic seg-



	MSE	MAE	Search Cost
h_1	5.0	1.0	5
h_2	5.8	1.4	4

Figure 3: Equation 2 does not satisfy in this example. We have a 1D map that contains five nodes. A robot can move either left or right. The cost of an action is 2. h_1 and h_2 represent some learned heuristics. h in each node represents a value of the heuristic function. Mean squared error (Equation 3) and mean absolute error (Equation 4) between the heuristic and the optimal heuristic are shown in the table on the right. The search cost is the number of expanded nodes when we apply the A^* algorithm. h_2 has more errors than h_1, but h_2 has a lower search cost.

mentation, to generate heuristic values. The architecture has recently been used as a generator in GAN architecture to solve image-to-image translation (Isola et al. 2017). U-Net achieves precise image generation by using the information from early convolutional layers directly on the latter layers. However, our approach is not limited to U-Net, and other image generation networks can be used. Figure 2 shows the architecture of the network. The input is an occupancy map and a goal tensor G. A goal tensor consists of two matrices that represent $s_{goal} = (x_{goal}, y_{goal}, \theta_{goal})$ where (x_{goal}, y_{goal}) is a robot's goal position and θ_{goal} is its goal orientation. Let (i_{goal}, j_{goal}) be a position in the occupancy map that contains (x_{goal}, y_{goal}). We define the goal tensor G as follows:

$$G(0,i,j) = \begin{cases} cos(\theta_{goal}), & \text{if } i = i_{goal} \text{ and } j = j_{goal} \\ 0, & \text{otherwise} \end{cases}$$

$$G(1,i,j) = \begin{cases} sin(\theta_{goal}), & \text{if } i = i_{goal} \text{ and } j = j_{goal} \\ 0, & \text{otherwise} \end{cases}$$

We could also discretize θ, but this increases the input dimensions. We used $cosine$ and $sine$ instead of the raw value in radians to compute the proper distance between two orientations (e.g. the distance between 2π and 0 should be 0 instead of 2π). The output is a discretized heuristic value that is a tensor of $W \times H \times O$ where W and H is the number of columns and rows in the occupancy map, respectively, and O is the number of orientations.

Loss Functions

The objective of the path planning problem is to find the shortest path with a constraint on search cost. The path cost can be defined as a function of the path length or the number of actions. The search cost can be defined as a function of the number of expanded nodes during the search process. In this paper, the research question regards whether we can learn a heuristic function that is better than handcrafted heuristics regarding the search cost. We will first explain why learning a heuristic function is not a straightforward supervised learning problem and then explain why the choice of the loss function is important.

Let L be the cost of a path generated by a planner and T be the search cost of the planner given m, s_0, s_{goal}, and h. The objective function for learning a heuristic function, $J(h)$, is a reward function of a path cost and/or a search cost. $J(h)$ satisfies the following condition:

$$\text{If } L_1 \geq L_2 \text{ and } T_1 \geq T_2 \text{ , then } J(h_1) \leq J(h_2) \quad (1)$$

Equation 1 means that h_2 has a better reward if both the path cost and the planning cost are lower than those of h_1. This reward can be computed by solving path planning problems with a path planner and h. Thus, J is in general not differentiable. To compute J appropriately, we need to solve a path planning problem using a planner with h. Hence, evaluating J is computationally expensive. Non-differentiability and expensive computation make it challenging to learn heuristics in an end-to-end manner.

Let $d(\cdot)$ be a loss function, h_1 and h_2 be learned heuristics, and h^* be an optimal heuristic. We can reduce the problem above to a supervised learning problem whose objective is to minimize errors between learned heuristics and optimal heuristics if the following condition is satisfied (Barto and Dietterich 2004):

$$\text{If } d(h_1, h^*) \leq d(h_2, h^*) \text{, then } J(h_1) \geq J(h_2) \quad (2)$$

However, standard loss functions, such as mean squared error (MSE) or mean absolute error (MAE), do not satisfy this condition, although lower loss usually indicates higher J. Figure 3 shows one simple example in which Equation 2 does not hold. In this example, both a heuristic function h_1 and a heuristic function h_2 can find an optimal path, but the search cost is different. Although h_1 has better MSE and MAE, it needs to expand more nodes. Therefore, it is essential to understand which loss functions are suitable so that learning a heuristic can be treated by solving the supervised learning problem (minimizing $d(\cdot)$) rather than solving the original computationally expensive optimization problem (maximizing $J(\cdot)$). We will first introduce the standard loss functions and then propose new loss functions for this heuristic learning problem.

MSE between two tensors h and h^* is defined as follows

$$MSE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} \left(h(i) - h^*(i)\right)^2, \quad (3)$$

where h is a learned heuristic, h^* is an optimal heuristic, and N is the number of elements in the tensor.

MAE is computed as follows:

$$MAE(h, h^*) = \frac{1}{N} \sum_{i=0}^{N-1} |h(i) - h^*(i)| \tag{4}$$

With MAE, we equally punished the errors in all directions. However, because we want to generate heuristics that are between known lower bound of the cost and the optimal cost, we can penalize more if the learned heuristic is out of the range between these bounds. If the learned heuristic is larger than the optimal cost, it does not guarantee the optimality anymore. Hence, we propose the piecewise MAE as

$$\begin{aligned} Loss_{piece}(h, h^*) = \frac{1}{N} \Bigg(& \\ & \alpha_1 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h(i) < h_{min}(i)] \\ & + \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h_{min}(i) \leq h(i) \leq h^*] \\ & + \alpha_2 \sum_{i=0}^{N-1} |h(i) - h^*(i)| \, [h^*(i) < h(i)] \Bigg), \end{aligned}$$

where h_{min} is a lower bound of the cost, $[\cdot]$ is an indicator function, and $\alpha_1 \geq 1$ and $\alpha_2 \geq 1$ are positive constants. The lower bound usually can be computed by considering the environment without obstacles. The first term computes the sum of the absolute difference between the learned heuristic and the optimal heuristic if the learned heuristic is lower than the minimum cost at state. Similarly, the second term computes the sum of the absolute difference if the heuristic is between the minimum cost and the optimal cost. The third term is computed if the heuristic is more than the optimal cost. If $\alpha_1 = 1$ and $\alpha_2 = 1$, this loss function is reduced to MAE.

Generally, it is hard to learn the optimal heuristic for every task; thus, the learned heuristic does not necessarily preserve gradients of the heuristic with respect to actions, as shown in Figure 3. We used the following loss function to capture errors on the gradients:

$$Loss_{grad}(h, h^*) = \sum_{a \in A} MAE(K_a * h, K_a * h^*) \tag{5}$$

In this function, the $*$ operator denotes a discrete convolution operation. $K_a * h$ approximates the gradient of h with respect to $a \in A$ where A is an action set. This operation is similar to the Sobel-Feldman operator (Sobel 1968) that captures gradients of an image. For example , K_a for "*north*" action in a 2D grid world is as follows:

$$K_{north} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

We used the weighted sum of these loss functions for our neural network as

$$Loss = Loss_1 + \alpha Loss_{grad}, \tag{6}$$

where α is a constant that controls the importance of $Loss_{grad}$. We used MSE, MAE, or $Loss_{piece}$ as $Loss_1$. Both $Loss_1$ and $Loss_{grad}$ are differentiable, and we can apply backpropagation to update the weights of the network.

Instead of using some handcrafted properties like the gradient described above, it may be possible to use another DNN to learn properties associated with optimal heuristics automatically. Thus, we investigated the use of GAN for this purpose. We introduced another DNN to discriminate the learned heuristic with the optimal heuristic. We trained a generator network (U-Net) that generates heuristic values which minimize $Loss$ and fools a discriminator network f_D. We used a loss from a discriminator of a Wasserstein GAN (WGAN) (Arjovsky, Chintala, and Bottou 2017). In our case, we use the goal pose tensor and a heuristic function as an input to the discriminator. Then, a loss function for the discriminator tensor $Loss_{des}$ is computed as follows:

$$Loss_{des} = f_D(h, G) - f_D(h^*, G), \tag{7}$$

where h is the learned heuristic (output of the generator network), and G is a goal tensor. The learned heuristics are the fake examples, and the optimal heuristics are the real examples. We trained the discriminator network using $loss_{des}$, and we trained the generator network (U-Net) using the loss function of WGAN $Loss_{WGAN}$, $Loss_1$, and $Loss_{grad}$:

$$Loss_{WGAN} = -f_D(h, G) \tag{8}$$

The loss function from WGAN computes the loss regarding the properties of the optimal heuristic. We used a weighted sum of loss functions to train the generator network as

$$Loss = Loss_1 + \alpha Loss_{grad} + \beta Loss_{WGAN}, \tag{9}$$

where β is a constant. The discriminator network architecture we used in our experiments is as follows: Input - Convolution - Convolution - Average Pooling - Dropout - Dense - ReLU - Dense - ReLU - Dense - ReLU - Dense - Output. One of the major drawbacks of WGAN is that it needs twice as much time and memory as other methods in the training stage.

Training

For training purposes, we make environments with randomized start and goal poses. We then created the optimal heuristic h^* using the Dijkstra algorithm starting from s_{goal} and assuming that each action has an inverse action. We trained the network with h^* using the Adam optimization (Kingma and Ba 2014). To evaluate the network, we generated h using the network, and passed it to a path planner (either A^* or Hybrid A^*), and measured the search cost. As we explained earlier, the lower loss does not necessarily imply better performance, and we used path planners with the learned heuristic to evaluate the network.

Experiments

We considered two simulated domains: the toy 2D domain and the continuous 2D domain.

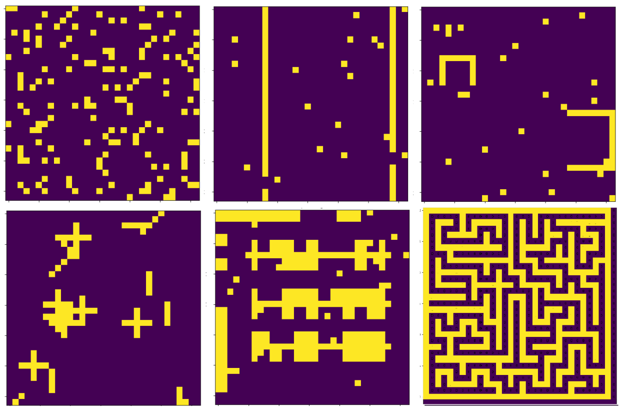


Figure 4: Examples of environments used in the toy domain.

Toy domain

The toy domain is a simple 2D grid world in which a robot can move either north, west, east, or south. The robot volume is equivalent to one cell. There is no orientation. Thus, we used a one-hot matrix to represent the goal position instead of using the goal tensor explained earlier for the input. We generated 180,000 training examples from six different environments (Figure 4). We used different rules to create such environments with random parameters. We also generated 10,000 test examples. We used Manhattan distance and scaled Manhattan distance (inflation factor is 1.5) as baselines. We introduced *difficulty* of a task to evaluate performance in the toy domains as

$$\textit{difficulty} = \frac{h_{Opt}(s_0)}{h_{Base}(s_0)}, \tag{10}$$

where h_{Base} is a commonly used simple heuristic function that is admissible and consistent, such as Manhattan distance or Euclidean distance in our task. We use this criterion for a fair comparison against baselines because baselines work well in simple environments. In this equation, *difficulty* measures how the optimal cost at s_0 deviates from the estimated cost (heuristic). If this value is large, the estimated cost is much smaller than the optimal cost. Hence, a simple heuristic may not be able to guide the planner appropriately. We used MAE, MSE, $Loss_{grad}$, $Loss_{piece}$, and $Loss_{WGAN}$ for loss functions. There will be a number of combinations of these, and we investigated some of the combinations. We use $\alpha_1 = 1.0$, $\alpha_2 = 2.0$, $\alpha = 1.0$ (when we use $Loss_{grad}$), and $\beta = 1.0$ (when we use $Loss_{WGAN}$). We selected these hyper parameters after trying several values.

Continuous domain

We considered a four-wheeled robot that measured $0.25m \times 0.20m$. The environment is a $2.318m \times 2.318m$ parking lot. We randomized occupied parking spaces and added small obstacles randomly. We used a bicycle model (Rajamani 2011) to describe the robot's kinematics. The robot uses five steering angles for both forward and backward direction with the fixed speed. There are ten actions in total. Collision checks are performed in continuous space. To plan a path, we used a Hybrid A^*. This planner does not hold optimality anymore, but it can still produce suboptimal drivable paths. We discretized states (x, y, θ) into $32 \times 32 \times 32$ states for the Hybrid A^*. We generated $3,040$ training samples and 130 test samples. We also randomized s_0 and s_{goal} in continuous space. The robot reaches s_{goal} when it reaches a voxel containing s_{goal}. Hence, the robot's orientation must be close to the goal orientation. We use $\alpha_1 = 1.0$ and $\alpha_2 = 2.0$ for $Loss_{piece}$, and we use $\alpha = 0.01$ for $Loss_{grad}$. The *difficulty* measure was not used because Hybrid A^* is not guaranteed to produce an optimal solution or to find a path even if a path exists. We did not investigate WGAN loss in this domain because it does not bring many benefits in the toy domain (we describe it in the next section) despite the fact that it takes a very long time to train.

Results and Discussion

Toy domain

Figure 5 shows successful cases, and Table 1 summarizes the results. As shown in Figure 5, a planner with the learned heuristic function does not need to expand many nodes (green stars) if the solution exists. If there is no path, the learned heuristic produces high values for unreachable states from the goal. Note that we used a single learned network to generate heuristic values in all testing environments. Hence, the results also show the generalizability of our proposed algorithm. In Table 1, the bold numbers show that the learned heuristic function outperforms baselines regarding the search cost. The gradient loss helps MSE and MAE reduce the search cost. Use of GAN also contributes to reducing the search cost of MSE and MAE. Piecewise loss works better, especially when tasks are difficult. Although WGAN improves the performance a little, it needs twice as much training time and memory as other methods. We do not provide the statistical testing because the number of testing examples is too large for the meaningful statistical testing for our experiments, and a small difference can easily lead to the statistical significance (Lin, Lucas Jr, and Shmueli 2013). Such statistical significance does not indicate meaningful results. Thus, we show the mean of the ratio of the number of expanded nodes only. When the tasks are easy (*difficulty* $= [1.0, 1.2]$), the scaled Manhattan distance is better than the learned heuristics.

Contrary to computer vision tasks, such as object segmentation and object recognition, a change of one cell can change the optimal heuristic drastically. Hence, learning heuristic functions using DNN can have a difficulty that does not appear in other DNN applications. As the results show, even when we randomized goal positions, the neural network could still learn a heuristic that is better than the baselines. Thus, our results indicate that DNN is capable of dealing with such difficulty. There are some cases in which the learned heuristic does not produce good results, as shown in Figure 6. The number of unsuccessful examples can be reduced by increasing training samples.

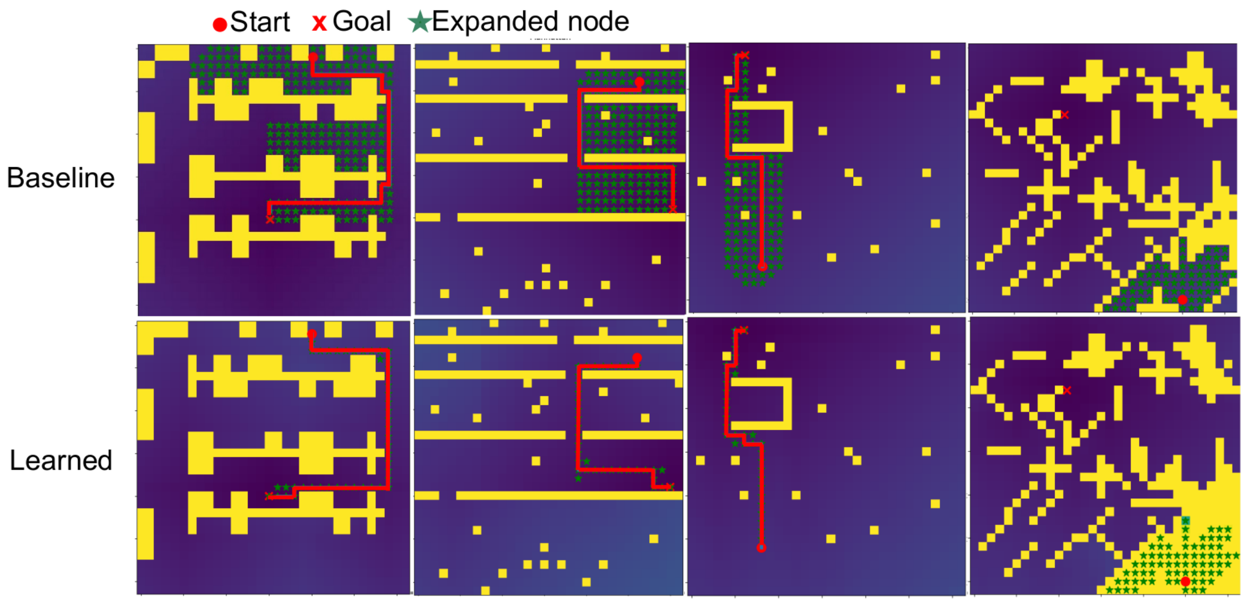


Figure 5: Successful results in toy domains. The top row shows the results of baselines that use the Manhattan distance, and the bottom row shows the results of the learned heuristic that used our proposed piecewise loss function. The color of each cell represents the value of the heuristic. Dark blue represents small heuristic. Yellow indicates the highest heuristic, and it usually represents the obstacles or unreachable region from a goal. Red circles are s_0, red crosses are s_{goal}, and green stars are expanded nodes in A^*. The rightmost column shows an environment where a path does not exist.

Loss	Complexity Evaluation Ratio of the number of expanded nodes on different task difficulties									
	1.0-1.2	1.2-1.4	1.4-1.6	1.6-1.8	1.8-2.0	2.0-2.2	2.2-2.4	2.4-2.6	2.6-2.8	$\geq$ 2.8
MSE	0.64	0.57	**0.56**	**0.57**	**0.56**	**0.56**	**0.60**	**0.64**	**0.65**	**0.71**
MSE, $loss_{grad}$	0.50	**0.42**	**0.41**	**0.40**	**0.46**	**0.46**	**0.48**	**0.53**	**0.53**	**0.62**
MAE	0.57	**0.46**	**0.45**	**0.42**	**0.48**	**0.51**	**0.58**	**0.60**	**0.59**	**0.62**
MAE, $loss_{grad}$	0.60	**0.45**	**0.45**	**0.46**	**0.47**	**0.51**	**0.51**	**0.60**	**0.55**	**0.65**
MSE, $loss_{WGAN}$	0.58	0.53	**0.52**	**0.49**	**0.47**	**0.50**	**0.51**	**0.53**	**0.63**	**0.71**
MAE, $loss_{grad}$, $loss_{WGAN}$	0.54	**0.45**	**0.42**	**0.41**	**0.43**	**0.45**	**0.47**	**0.53**	**0.56**	**0.61**
Piecewise	0.57	**0.46**	**0.44**	**0.42**	**0.44**	**0.47**	**0.46**	**0.54**	**0.52**	**0.58**
Piecewise, $loss_{grad}$	0.54	**0.45**	**0.45**	**0.43**	**0.46**	**0.51**	**0.48**	**0.58**	**0.51**	**0.58**
Baseline (Manhattan)	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
Baseline (Scaled Manhattan)	0.41	0.49	0.63	0.69	0.72	0.75	0.78	0.81	0.82	0.84

Table 1: Results of experiments in the toy domain. The number shows the mean of the ratio of the number of expanded nodes where the ratio $= \frac{N}{N_b}$, N is the number of expanded nodes using the learned heuristic, and N_b is the number of expanded nodes using the Manhattan distance. Task difficulty is computed by Equation 10. If the task difficulty is $[1.0, 1.2]$, this indicates the task is "easy". The bold numbers indicate that the leaned heuristic is better than the baselines.

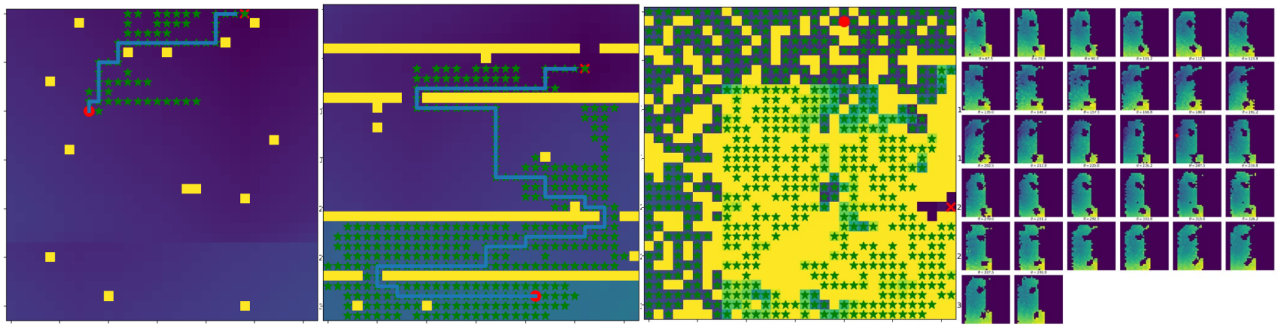


Figure 6: The first three figures show unsuccessful results in the toy domain, and the rightmost figure shows an unsuccessful result in the continuous domain. The learned heuristic expanded many unnecessary nodes in the first, second, and fourth figures. The third figure from the left shows the environment where the path does not exist. In the third figure, the learned heuristic function generated small heuristic values in many unreachable states from the goal. However, all cells, except several cells around the goal, should be yellow (high heuristic values) because there are no paths from these cells. Note that the learned heuristic function generates these images (except for the green star, path, red circle, and red cross). Thus, the free space can be yellow if the space is unreachable from the goal. You see some expanded nodes in yellow cells in the third figure because of this.

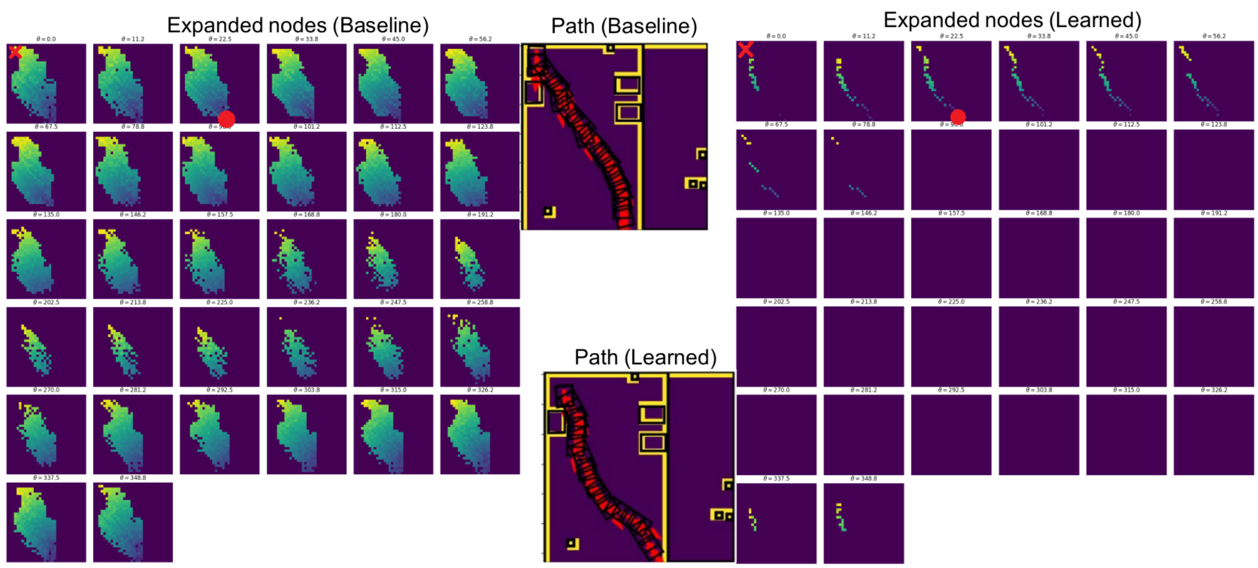


Figure 7: A successful example in the continuous domain. The left figures show the results of Euclidean distance, and the right figures show results of piecewise loss. We use 32 orientations in the Hybrid A^*, and each figure corresponds to each orientation. The start orientation is about $\theta = 22.5°$ and the goal orientation is about $\theta = 0°$. The red circle is the start position (first row, third column) and the red cross is the goal position (first row, first column). Each figure represents the expanded nodes. Cells with dark blue are nonexpanded nodes. Cells with other colors represent the expanded nodes. The learned heuristic does not need to expand many nodes. The figures labeled "Path" represent a path generated with the heuristics. Red lines indicate, paths and black lines indicate either contours of a robot or obstacles. Yellow cells indicate obstacles in an occupancy map.

Loss functions	Complexity Evaluation								
	Comparison against baselines 130 test tasks N: The number of expanded Nodes N_b : Baseline, N: Learned				Ratio of the number of expanded nodes q th percentile Ratio $= \frac{N}{N_{Euclidean}}$				
	Euclidean		Scaled Euclidean		25th	Median	75th	90th	95th
	$N_b < N$	$N_b > N$	$N_b < N$	$N_b > N$					
MSE	76	44	102	15	0.650	1.939	4.348	8.256	12.380
MSE, $loss_{grad}$	60	62	90	32	0.423	1.014	2.397	6.314	9.527
MAE	34	**87**	69	51	0.145	0.383	1.095	2.351	5.272
MAE, $loss_{grad}$	21	**99**	53	67	0.084	0.179	0.859	1.456	4.292
Piecewise	13	**107**	36	**83**	0.035	0.081	0.248	0.770	1.082
Piecewise, $loss_{grad}$	16	**105**	37	**83**	0.046	0.109	0.307	0.908	1.656
Baseline (Euclidean)	NA	NA	87	42	1.00	1.00	1.00	1.00	1.00
Baseline (Scaled Euclidean)	42	87	NA	NA	0.122	0.265	0.641	0.888	1.500

Table 2: Results of experiments in the continuous domain. The numbers in the comparison against baselines show the number of tasks that are better than the other method. For example, $N_{base} < N$ shows the number of tasks where the baseline has fewer expanded nodes than that of the learned heuristic. The bold numbers indicate that the leaned heuristic is statistically better than the baseline (χ^2 test, $p < 0.05$). Outliers dominate the mean and the variance. Instead, we show the q th percentile.

Continuous domain

Figure 7 shows a successful example in the continuous domain where the Hybrid A^* with the learned heuristic does not need to expand many nodes. With only 130 testing examples, such as, χ^2 test can be applied to produce statistically meaningful results. Table 2 summarizes the results of statistical testing. The results are similar to those of the toy 2D domain. MAE is better than MSE. The use of gradient loss improves the results for MSE and MAE; however, there is no statistical significance when we compare them against the scaled Euclidean heuristic. Piecewise loss is statistically better than the scaled Euclidean. The loss of gradient degrades the results of the piecewise loss. We also show the ratio of the number of expanded nodes in Table 2. If we use the piecewise loss, the ratio of expanded nodes is less than 0.081 in 50% of tasks, 0.248 in 75% of tasks, and 0.77 in 90% of tasks. These results indicate that it is possible for DNNs to learn a heuristic function that is better than simple baselines. It also shows that the proposed piecewise loss function captures errors that simple loss functions, such as MSE and MAE cannot. There are several cases where the learned heuristic function does not work well. One of the examples is shown in Figure 6. In this case, Hybrid A^* expands almost all of the reachable nodes from s_0. Coarse discretization of orientations is one of the issues that a Hybrid A^* needs to expand many nodes to find a path. Increasing the number of orientations can avoid such cases. When we create h^* during training, we need to apply many collision checks which are computationally expensive. However, once the DNN learns h, we do not need to apply collision checks to generate h during the testing time.

Conclusion and Future work

In this paper, we applied the neural network architecture to learn heuristic functions for path planning. Our preliminary results showed that the learned heuristic function can reduce the search cost in both the toy domain and the continuous domain. We also investigated loss functions and showed that the proposed piecewise loss helps DNNs to learn better heuristics. The learned DNN does not produce high-quality heuristics in some cases, but the path planner can still find a path even with those low-quality heuristics. This is one of the benefits of combining existing path planning algorithms with DNNs. Loss of gradient with respect to actions also helps other loss functions, such as MSE and MAE, to learn heuristic functions. Use of GAN automatically captures the properties of optimal heuristics and produces good heuristics, but it requires longer training time compared to other approaches.

As future work, we want to investigate network architectures, including the representation of the goal tensor, to produce better heuristic values. We also would like to examine the scalability of the proposed approach using large maps. We consider our proposed method as suitable for parking scenarios because occupancy maps can easily contain both start and goal poses. Although we worked on the fully observable environments in this study, the proposed method can easily create new heuristic values given the new information because feed-forward computation of the DNN is fast thanks to high-performance GPUs. Thus, the DNNs should be able to deal with partially observable environments by replanning. We can also easily change the inputs to multiple frames of occupancy maps to deal with dynamic objects. Therefore, we would like to work on modifying the proposed approach for partially observable environments and dynamic environments as future work.

Acknowledgments

The authors would like to thank Scott M. Jordan for having valuable discussions on DNNs.

References

Arjovsky, M.; Chintala, S.; and Bottou, L. 2017. Wasserstein gan. *arXiv preprint arXiv:1701.07875*.

Banino, A.; Barry, C.; Uria, B.; Blundell, C.; Lillicrap, T.; Mirowski, P.; Pritzel, A.; Chadwick, M. J.; Degris, T.; Modayil, J.; et al. 2018. Vector-based navigation using grid-like representations in artificial agents. *Nature* 557(7705):429.

Barto, A. G., and Dietterich, T. G. 2004. Reinforcement learning and its relationship to supervised learning. *Handbook of learning and approximate dynamic programming. John Wiley and Sons, Inc*.

Bhardwaj, M.; Choudhury, S.; and Scherer, S. 2017. Learning heuristic search via imitation. *arXiv preprint arXiv:1707.03034*.

Connolly, C. I., and Grupen, R. A. 1993. The applications of harmonic functions to robotics. *Journal of Robotic Systems* 10(7):931–946.

Dhiman, V.; Banerjee, S.; Griffin, B.; Siskind, J. M.; and Corso, J. J. 2018. A critical investigation of deep reinforcement learning for navigation. *arXiv preprint arXiv:1802.02274*.

Dolgov, D.; Thrun, S.; Montemerlo, M.; and Diebel, J. 2008. Practical search techniques in path planning for autonomous driving. *Ann Arbor* 1001(48105):18–80.

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

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.

Isola, P.; Zhu, J.-Y.; Zhou, T.; and Efros, A. A. 2017. Image-to-image translation with conditional adversarial networks. *arXiv preprint*.

Khan, A.; Zhang, C.; Atanasov, N.; Karydis, K.; Kumar, V.; and Lee, D. D. 2017. Memory augmented control networks. *arXiv preprint arXiv:1709.05706*.

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

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

LaValle, S. M. 1998. Rapidly-exploring random trees: A new tool for path planning.

Likhachev, M.; Gordon, G. J.; and Thrun, S. 2004. ARA*: Anytime a* with provable bounds on sub-optimality. In *Advances in neural information processing systems*, 767–774.

Lin, M.; Lucas Jr, H. C.; and Shmueli, G. 2013. Research commentary-too big to fail: large samples and the p-value problem. *Information Systems Research* 24(4):906–917.

Mahadevan, S., and Maggioni, M. 2007. Proto-value functions: A laplacian framework for learning representation and control in markov decision processes. *Journal of Machine Learning Research* 8(Oct):2169–2231.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529.

Pan, Y.; Cheng, C.-A.; Saigol, K.; Lee, K.; Yan, X.; Theodorou, E.; and Boots, B. 2018. Agile autonomous driving using end-to-end deep imitation learning. *Proceedings of Robotics: Science and Systems. Pittsburgh, Pennsylvania*.

Panov, A. I.; Yakovlev, K. S.; and Suvorov, R. 2018. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia Computer Science* 123:347–353.

Rajamani, R. 2011. *Vehicle dynamics and control*. Springer Science & Business Media.

Ratliff, N. D.; Bagnell, J. A.; and Zinkevich, M. A. 2006. Maximum margin planning. In *Proceedings of the 23rd international conference on Machine learning*, 729–736. ACM.

Ronneberger, O.; Fischer, P.; and Brox, T. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, 234–241. Springer.

Sobel, I. 1968. History and definition of the so-called" sobel operator", more appropriately named the sobel-feldman operator. *Sobel, I., Feldman, G.," A 3x3 Isotropic Gradient Operator for Image Processing", presented at the Stanford Artificial Intelligence Project (SAIL) in* 2015.

Tamar, A.; Wu, Y.; Thomas, G.; Levine, S.; and Abbeel, P. 2016. Value iteration networks. In *Advances in Neural Information Processing Systems*, 2154–2162.

Wang, Y.; Jha, D. K.; and Akemi, Y. 2017. A two-stage rrt path planner for automated parking. In *Automation Science and Engineering (CASE), 2017 13th IEEE Conference on*, 496–502. IEEE.

Zhang, J.; Tai, L.; Boedecker, J.; Burgard, W.; and Liu, M. 2017. Neural SLAM. *CoRR* abs/1706.09520.