Hamilton Jacobi Value Iteration Network

by

Jiajun Cui

Department of Mechanical Engineering & Materials Science
Duke University

Date: _____

Approved:

_____
Miroslav Pajic, Supervisor

_____
Michael M. Zavlanos

_____
Leila Bridgeman

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Department of Mechanical Engineering & Materials Science
in the Graduate School of
Duke University

2023

ABSTRACT

Hamilton Jacobi Value Iteration Network

by

Jiajun Cui

Department of Mechanical Engineering & Materials Science
Duke University

Date: _____

Approved:

_____
Miroslav Pajic, Supervisor

_____
Michael M. Zavlanos

_____
Leila Bridgeman

An abstract of a thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in the Department of Mechanical Engineering & Materials Science
in the Graduate School of
Duke University

2023

Abstract

In this work, we address the safe navigation problem for the robot equipped with the neural network controller. Our goal is to propose a neural network controller representation that can efficiently and safely learn a safe policy. By following the learned safe policy, the robot can reach the goal state while avoiding hitting obstacles and walls all the time. We use Hamilton Jacobi safety analysis to improve the safety awareness of the policy and integrate it within the value iteration network to generalize to the new, unseen domains outside the training set. Applying the transfer learning techniques, we can learn a reward function that maps each state to a reasonable reward value. We use the learned reward function to construct the unknown part in the discrete-time Hamilton Jacobi value function and integrate this Hamilton Jacobi value function into the value iteration network to construct our Hamilton Jacobi value iteration network model. Finally, we compare the performance of our model with the value iteration network model in the grid world domains to show our model can safely learn a safe policy that generalizes to the new, unseen domains.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In recent years, advanced in the deep learning field and motivated by the success of machine learning in other areas such as computer vision, lots of learning-based controllers have been designed for cyber-physical systems. Robot trajectory planning and control is certainly one of the main applications where learning-based controllers can be successfully employed [KHJ+19], [YG21],[OSL17],[WYK19],[CC21]. Although the property of the learning method improves the robustness of the controller, the issue of safety has arisen.

Researchers in learning and control have developed many approaches to address this issue. One of the categories around this issue is to provide a set constraint certification for the safe controller. By combining the model-free Reinforcement Learning (RL) and control barrier function (CBF) together, [COMB19] proposes a method to provide a safety guarantee during the whole learning process. In the paper, the author integrates trust region policy optimization (TRPO) and deep deterministic policy gradients (DDPG) to learn a controller and combines CBF to provide a safety guarantee during the entire learning. However, this method needs to know the valid safe set, which is non-trivial. [XHLR21] proposes the BarrierNet which is a barrier layer providing a safety guarantee by combining deep learning algorithms. The author relaxes the hard constrained in high-order CBFs (HOCBFs) and forms the HOCBFs

soft constraint as a differentiable layer so that it can be trained end-to-end. The HOCBFs layer then serves as a safety layer by integrating with any neural network controller to provide a safety guarantee. [PWET21] proposes a new end-to-end trainable framework to ensure safety for controller design in a stochastic environment. In the paper, the author integrates stochastic CBFs within Stochastic Optimal Control (SOC) to provide a safety guarantee.

Except for combining control barrier functions, there are other methods that can provide a safety certification such as combining predictive safety filters. [FESF20] proposes the ShieldNN algorithm. The ShieldNN essentially uses a barrier function transforming the safety problem into a constrained policy set problem. The authors designed a novel Barrier Function (BF) for the Kinematic Bicycle Model (KBM) and then used the BF as a safe NN filter to design a safe controller. Hamilton Jacobi reachability also is a powerful tool to provide safety certification for controllers. [FLRR$^+$19] presents a promising tool for offline safety analysis. In the paper, the authors develop a discounted Hamilton- Jacobi Bellman equation formulation that is suitable for using reinforcement learning algorithms to compute the safety policy and the unsafe state set. The [HRRTF21] extended the prior method by adding the target set information. The authors propose algorithms not only can ensure safety but also liveness through the Hamilton Jacobi Reachability reinforcement learning formula. However, although these methods can provide a safety guarantee for learning a safe policy, they can not guarantee the learning process is safe. It is crucial not

only to improve safety for the optimized controller but also during learning to avoid significant hardware damage[BGH+22].

Researchers have developed many safe reinforcement learning approaches to address the issue mentioned above. [CNDGG18] considers the safe RL problem a constrained Markov decision process. The author develops a linear programming method to construct the Lyapunov function. The author also develops safe policy iteration and safe value iteration by integrating the Lyapunov function with a dynamic programming algorithm. To solve the uncertainty in the environment, the author develops a safe DQN and safe DPI. [XZZ22] seeks to find a way to use an extra cost function to provide safety criticism and penalize the Q-value to learning a safe controller. The author proposes the Constraints Penalized Q-learning to solve the conservative problem during this process. [TLM21] mainly focuses on safe exploration in reinforcement learning. The authors develop a model-based RL algorithm Safe Model-Based Policy Optimization (SMBPO) that uses the reward penalty to ensure safety in exploration. By modifying the reward function, the agent can learn a safe policy by penalizing safety violations. [FP18] proposes a novel algorithm, Justified Speculative Control (JSC) by integrating formal verification and the reinforcement learning algorithm to ensure safety for the learned controller. [HFM+21] proposes Verifiably Safe Reinforcement Learning (VSRL) for constrained reinforcement learning to learn safe policy.

Some other learning frames have also been developed to address the safety issue.

[SCKV21] proposed a parameterized model to learn a safe policy called Reachability-based Trajectory design. In the paper, an over-approximating Forward Reachable Set (FRS) is constructed by the bounds of the error between the agent model and the planning model. Then the FRS is used to provide a safe controller. [SCKV21] adopts a parameterized reachable set to construct a safety layer to ensure real-time safe reinforcement learning. This paper proposed a safe layer called a Reachability-based Trajectory Safeguard (RTS) by combining the RTD with RL. A Planning Reachable Set (PRS) is computed offline and to bind the tracking error, an Error Reachable Set (ERS) is also computed. By combining PRS and ERS, a Forward Reachable Set (FRS) is constructed and the safety constraint can be created. Then the agent could safely learn a safe controller using a model-free RL algorithm using RTS. Instead of learning the control input, in this paper, the RL agent learns the plan parameter which greatly reduces the dimension of the agent's action space for continuous control. [IWA+19] presents an approach that can verify safety using a deep neural network (DNN) based controller. The authors use sigmoid to transform the deep neural network verification problem into a hybrid system verification problem which is solvable for reachability algorithms like Flow*. These methods, however, most of them require an accurate robot dynamic.

In this paper, we propose a new learning frame and neural network controller representation by combining the Hamilton Jacobi safety analysis and the value iteration network to learn a safe controller that can generalize better to new, unseen

domains for robot navigation problems. Using imitation learning, a safe policy could be learned offline from an expert behavior, which can also keep the learning process safe. We first train the value iteration to learn a reward function in the training set. Then we use transfer learning, uploading trained parameters to our proposed neural network controller. We train our controller again with the same training set. Our numerical experiment shows that our controller can learn a safe policy that generalizes well to different domain configurations that are unseen in the training set. We organize the thesis as follows: In Chapter 2, we introduce the preliminaries and problem definition. Then we introduce the Hamilton Jacobi safety analysis, and the value iteration network, and present how we design our new learning frame and controller in Chapter 3. Numerical experiment results are shown in Chapter 4. Finally, We conclude in Chapter 5.

Chapter 2

Preliminaries and Problem Definition

2.1 Preliminaries

2.1.1 Markov Decision Processes

We model the stochastic environment as an MDP M. An MDP M is a tuple (S, A, R, P, γ), where:

- S is a set of states;
- A is a set of actions;
- $P : S \times S \times A \mapsto [0, 1]$ is a state transition function such that $\sum_{s' \in S} P(s', s, a) = 1$, for all $s \in S$, $a \in A$;
- $R : S \mapsto \mathbb{R}$ is a reward function;
- $\gamma \in [0, 1)$ is a discount factor.

In a state $s \in S$, the agent selects an action $a \in A$ and makes a transition to a state $s' \in S$ with the transition probability $P(s', s, a)$ then receives the reward $R(s')$. The actions selected by the agent are determined by a policy π. A deterministic policy $\pi : S \mapsto A$ is a function, for all $s \in S$, that maps a state s to an action a.

The value function $V_\pi : S \mapsto \mathbb{R}$ is a function representing the expected return that the agent can obtain in the state s under the policy π:

$$V_\pi(s) \doteq \mathbb{E}\left[\sum_{\alpha=0}^{k}(\gamma^\alpha R(s_{t+\alpha}) \mid s_t = s\right], \text{ for any } \ s \in S. \tag{2.1}$$

The goal of the agent is to interact with the environment by selecting actions that maximize the expected returns. The optimal value function $V_*(s)$ represents the maximum expected return achievable under any policy:

$$V_*(s) \doteq \max_\pi V(s), \text{ for any } \ s \in S. \tag{2.2}$$

If the optimal value function is available, the optimal policy $\pi_*(s)$ can be retrieved by choosing the action among all actions in the state s that maximizes the expected return: $\pi_*(s) \doteq \text{argmax}_a V_*(s)$.

Figure 2.1 and Figure 2.2 show a simple, straightforward problem that can be modeled as an MDP. Consider a robot moving in the 8×8 grid world. The state space of the robot is 64 cells. The action space of the robot is the eight actions it can move in each of the eight compass directions. Consider two scenarios in the grid world: the state space and action space are defined above; the rewards are +20 for reaching the goal, -10 for hitting an obstacle, and -3 otherwise to encourage finding the shortest path; the discount factor is 0.65. The agent can only move inside a gird-world domain. If the agent hits the border of the gird-world, the agent would

keep in the same position.

The first scenario is that the robot is in a deterministic environment. In this scenario, each action deterministically causes the agent to move one cell in the respective direction. Both an MDP and the optimal policy are demonstrated in Figure 2.1. The circles denote the obstacles. The arrows represent the action north, west, south, east, northwest, northeast, southwest, southeast, and stop. When the optimal values of each cell are large, the colors of the cells are bright. One of the shortest paths to enter the goal position (2,2) from (6,3) is reaching (4,3) via (5,3), yet in this case, the agent hits an obstacle in (4,3) with a penalty of -20. Thus the optimal policy lets the agent visit (4,2) by selecting action northwest in (5,3). Following the optimal policy, the agent ends up entering goal position (2,2) with the optimal value at position (6,3).

The second scenario is the robot is placed in a stochastic environment. The robot may suffer some random disturbance, and its state will transition to a random adjacent state with some probability rather than go to the state in the intended direction. The state transition function encodes this stochastic behavior. (For example, $P((4,5),(4,4),North) = 0.8, P((5,5),(4,4),Northeast) = 0.2.$). Figure 2.2 depicts the optimal policy calculated by value iteration. the path that the agent enters the goal position (5,3) from position (4,1) following the optimal policy. The agent did not go in the intended direction because of the stochastic environment.

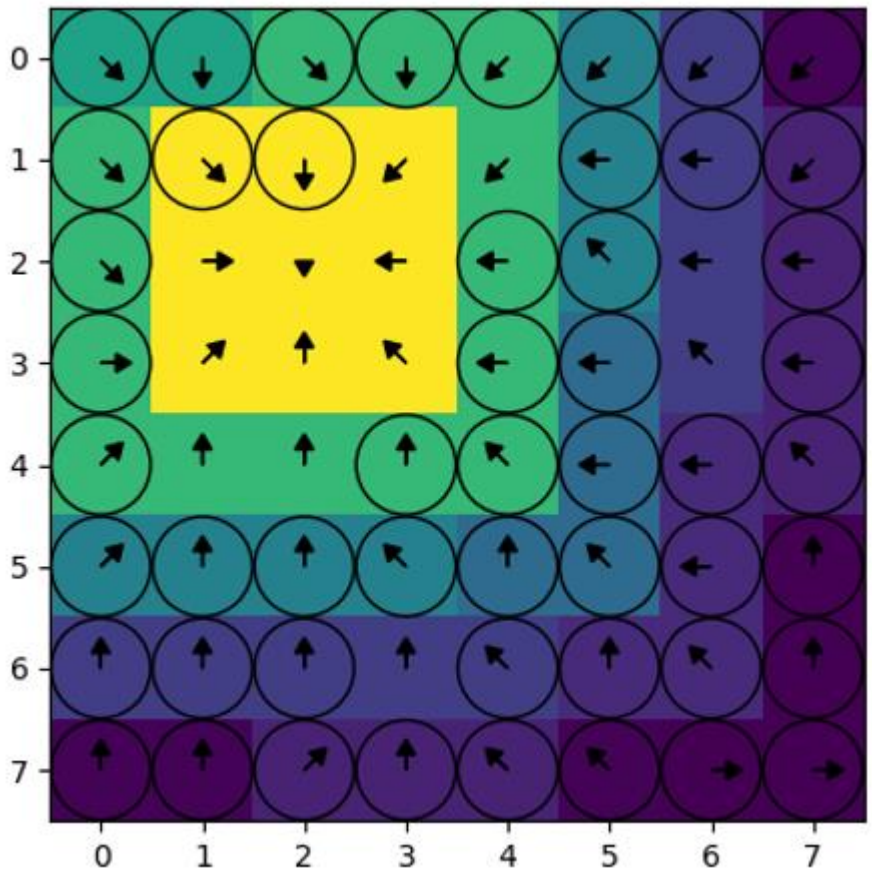


Figure 2.1: The optimal policy for the deterministic environment

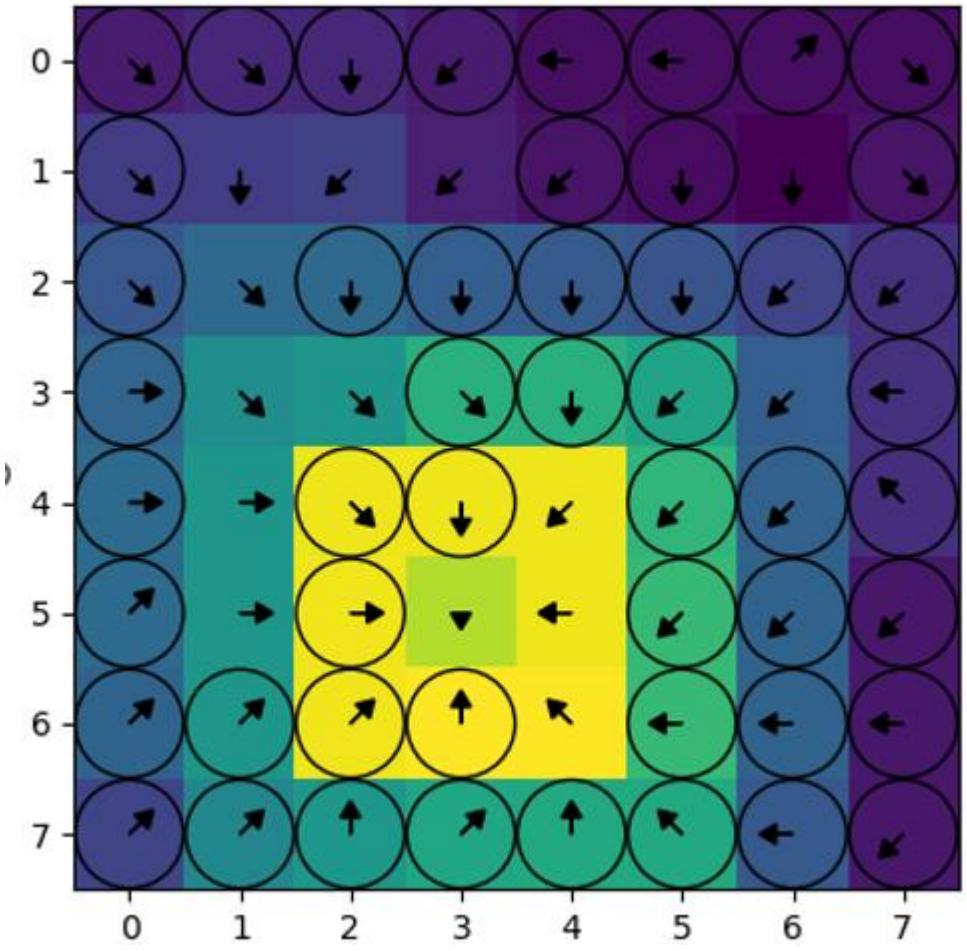


Figure 2.2: The optimal policy for the stochastic environment

2.1.2 Value Iteration

When the reward function R and the state transition function P both are known, VI is a common algorithm to find an optimal value function and optimal policy V_* and π_* in an MDP:

$$V_{k+1}(s) = \max_{a \in A} \left[R(s) + \gamma \sum_{s'} P(s', s, a) V_k(s') \right], \text{ for any } s \in S. \tag{2.3}$$

The use of the discount factor γ allows the agent to value the immediately received reward more than the future reward. The value function V_k will converge to V_* as iteratively applying this updating rule and when $k \to \infty$. However, in practice considering the computational efficiency, value iteration terminates when the value function barely changes in an iteration. Algorithm 1 shows the complete value iteration algorithm.

Algorithm 1 Value iteration

Initialize V(s) for all $s \in S$, a small threshold $\Theta > 0$.
while flag **do** :
  $\Delta \leftarrow 0$
  for each $s \in S$ **do**
    $v \leftarrow V(s)$
    $V(s) \leftarrow \max_{a \in A} [R(s) + \gamma \sum_{s'} P(s', s, a) V(s')]$
    $\Delta \leftarrow \max(\Delta,\ |V(s) - v|)$
  if $\Delta < \Theta$ **then**
    flag $\leftarrow$ False
Output: $\pi(s) = \text{argmax}_a [R(s) + \gamma \sum_{s'} P(s', s, a) V(s')]$

2.1.3 Reinforcement Learning

Reinforcement learning is a class of learning algorithms for solving incompletely-known MDPs. The task of reinforcement learning is to learn a policy that can maximize a value function. The agent is not told directly which actions to choose but discovers which policy yields the most reward by following it. The actions may not only influence the immediate reward but also all future rewards. These are two distinguishing features of reinforcement learning. These two features also bring a challenge to reinforcement learning, which is the trade-off between exploration and exploitation. The agent needs to exploit experiences to calculate reward but also needs to explore to seek a better policy. The dilemma is that in order to pursue either exploration or exploitation exclusively, the agent must exhaust all the possible actions, which will inevitably encounter failing at the task. On a stochastic task, each action even must be tried many times to obtain a reliable reward expectation, which will also arise a series of safe problems.

Reinforcement learning is different from another class of learning frame which is supervised learning. Supervised learning is a learning frame the agent learns from a set of labeled training data generated by the external expert. Each training data is a description of a scene combined with a label which is the correct action the agent should take under this scene. This could also be considered to classify a category to which the scene belongs. The main goal of supervised learning is to generalize its responses so that the agent could take the right action in the scene that is not pre-

sented in the training set. Although this is also a very important learning algorithm, it is not able to learn from the interaction. In some situations such as uncharted territory, it is very important for the agent to learn from its own experience.

There are generally two types of reinforcement learning algorithms: model-based learning algorithms and model-free learning algorithms. The model-based learning tries to understand the environment and create a model for it. The model can mimic the behavior of the environment. That is to say, the model can make inferences about how the environment behaves. For example, given the current position and action, the model is able to make a prediction about the next state and the reward. The model is used for planning, which means the agent can select actions by predicting future situations before the actions are actually performed. On the other hand, in model-free learning, the agent has to actually choose an action to see what happens and learn from it. A common model-free learning framework is that we directly estimate the state-action value function $Q : S \times A \mapsto \mathbb{R}$, without the knowledge of the dynamics of the system, and the agent will follow a policy that selects the action with the highest value in each state. Model-free learning is more computationally efficient. Model-based learning is more data-efficient and safer to explore but the policy learned can only be as good as the model learned.

Tabular Q-learning is a type of model-free learning. However, it still suffers from the curse of dimensionality, which means that the growth of the state space and the action space will lead to an exponential growth in storage requirement [HRRTF21].

Although deep reinforcement learning has the potential to alleviate the curse of dimensionality by using CNN-based deep neural networks to find approximate optimal policies, it does not generalize well because those learned policies are inherently reactive and lack planning nature [TWT+16].

2.1.4 Imitation Learning

In some cases, especially when the rewards are sparse or there is no direct reward function, using RL algorithms to approximate the best policy is especially challenging. To solve this issue, the researchers can manually design a reward function to provide agents with more frequent rewards. However, it is difficult to design a reward function manually to satisfy generalized situations. One way to approach this issue is imitation learning(IL). Instead need to know the reward function, the agent tries to learn the optimal policy by imitating an expert behavior.

For imitation learning, the environment is also an important component. The environment is also modeled as an MDP, which means it also has a state space S, an action space A, a transition probability $P(s', s, a)$, and an unknown reward function $R(s)$. There is also a set of expert behavior set $\tau = (s_0, a_0, s_1, a_1, ...)$. The other two important components of IL are the loss function and the learning algorithm.

Behavior cloning is one of the common IL forms, which uses supervised learning to learn the expert's policy. The expert behavior set is divided into state and action pairs. These state and action pairs are used as training data and test data, then

supervised learning is applied. The loss function depends on the task. For some situations, behavior cloning works well. However, in most cases, behavior cloning is quite problematic. These state and action pairs need to follow the independent and identically distributed assumption, but in MDP an action taken in a state would induce the next state, which breaks the independent and identically distributed assumption. The error would be added up along the trajectory. The agent would usually mistake putting it into some state the expert has never come in before. In those states, the behavior usually will lead to failure.

2.1.5 Transfer Learning

Transfer learning(TL) is a machine learning technique where a neural network pretrained on one task is reused as the starting point for training on another task. Transfer learning aims to improve learning performance on a new task (target task) by using the knowledge learned from a related but distinct task (source task). The followings are some benefits of TL: solving challenges by constructing a new network to train and test; Approaching the relation between the data set from a different perspective. TL is also a tool that can help to learn and generate more accurate to the target task.

2.2 Problem Definition

The objective of this thesis is to learn a safe policy that generalizes better to new domains with different obstacle configurations. Specifically, consider giving a bunch of grid world instances, which contain randomly placed obstacles, start position, and goal position. The goal of this paper is to develop a new neural network policy representation that can use these given grid world instances and the expert behavior in these instances to learn a safe policy π. By following the learned safe policy π, the agent can reach the goal state in new grid world instances while avoiding hitting obstacles and walls all the time.

Chapter 3

Our Approach

In this Chapter, we develop the proposed Hamilton Jacobi value iteration network model which integrated both the value iteration network (VIN) and the Hamilton Jacobi safety analysis (HJ-SA). In Section 3.1, we introduce the VIN, and in Section 3.2, we introduce the Hamilton-Jacobi safety analysis. In Section 3.3, we will introduce how we combine these two methods to construct the Hamilton Jacobi value iteration network (HJ-VIN) and the learning frame to learn a safe policy.

3.1 Value Iteration Network

The VIN is a convolutional neural network with a value iteration algorithm embedded within it. The task of the VIN is to learn a policy from a bunch of domain instances and the learned policy generalizes to domain instances that have new obstacle configurations [TWT+16].

A $(2 \times W \times H)$-sized observation image is fed into the VIN as the input image, in which the first channel is an image of the position of the agent, and the second channel is an image of the map of obstacles and goal position. This input image is first fed into some convolutional layers in order to map the state image to a reward

image $\bar{R}$ of dimensions $1, W, H$. This reward image $\bar{R}$ is then fed into the value iteration (VI) module, which is the most distinguishing part of the VIN. The VI module is a neural network that approximates the value iteration computation. The reward image $\bar{R}$ is convolved with the transition function $\bar{P}$ which is defined as 3×3 kernels. Each channel of this layer can be seen as an approximation of the Q-function for a specific action, $\bar{Q}_{a,i,j} = \sum_{l,i',j'} \bar{P}^a_{l,i',j'} \bar{R}_{l,i-i',j-j'}$. This layer is then max-pooled along the channel to approximate the computation $\bar{V}(s) = \max_a \bar{Q}(s,a)$ to obtain the value image $\bar{V}$. The value image $\bar{V}$ is then stacked with the reward image $\bar{R}$ and fed back into the VI module to iterate $K-1$ times in order to approximate value iteration. After $K-1$ times iteration, the value image $\bar{V}$ is stacked with the reward image $\bar{R}$ and fed back only into the convolutional layer of the VI module, and the output of this convolutional layer is seen as the final output of the VI module. This output of the VI module is a Q-value image with dimension l_a, W, H which contains the information of the optimal policy π. The Q-value image is fed into a fully connected layer and outputs the distribution over actions. To train the VIN, use the across-entropy loss function to compute the loss between the predicted action of a state and the expert action executed in the same state.

Compared with some deep reinforcement learning methods in which the policies are represented as standard CNN-based networks, such as DQN or DDQN, the VIN generalizes better to new tasks outside the training set[TWT+16]. However, the VIN in practice can only apply to small state space because the value iteration algorithm

embedded needs to visit every state space[OSL17]. It also failed to provide a safety guarantee not only for the learned policies.

Here are the implementation details of the VIN. The first 2 layers of the VIN are convolutional layers mapping the grid world image to a state reward image. The first convolutional layer is a hidden layer. The number of the input channel of this layer is 2. The first channel of the input image is the map of obstacle positions and the second channel of the input image is the map of the goal position. Then the input image is convolved with a 3×3 kernel with striding 1 and padding 1 so that the output image has the same size as the input image. The number of the output channel is 150 to increase the robustness of the VIN. The second convolutional layer has 150 input channels and 1 output channel. The output image of the last convolutional layer is convolved with a 1×1 kernel with striding 1 and 0 paddings. The output image is the state reward image of the grid world image. The third layer is a convolutional layer mapping a state reward image to a state-action pair reward image. The number of input channels is 1 and the number of output channels is 10. The state reward image is convolved with a 3×3 kernel with 1 stride and 1 padding. The kernel size is 3×3 because the agent can only visit its nearby state taking one action. The output channel is 10 to further increase the robustness of the VIN. Then the maximum state-action pair reward image is then convolved with a 3×3 kernel which denotes the transition probability with 1 stride and 1 padding and adds on the state-action pair reward image to approximate one times value iteration. After K times iteration, The

output is a state-action pair reward image. Feed the value in this image where the agent is currently positioned into the fully connected layer. The number of inputs is 10 and the number of outputs is 8 for 8 actions.

3.2 Hamilton-Jacobi Safety Analysis

Hamilton-Jacobi safety analysis (HJ-SA) is a formal verification approach for safety that can provide state constraint set certification for a learning-based controller. The most important concepts of HJ-SA are the backward reach-avoid set and reachable set.

Let us first define the safety problem for HJ-SA and then the backward reach-avoid set and reachable set under the safety problem. Consider the safety problem as the agent wants to avoid entering some undesirable states during the operation. We define the set of these undesirable states as a failure set $\mathcal{F}$, the complement of the failure set as a constraint set $\mathcal{K}$, and the set of states we want to finally reach as the target set $\mathcal{T}$. If the agent dynamics are assumed Lipschitz continuous, then the system trajectories $\xi : \mathbb{R}_+ \mapsto S$ are unique[HRRTF21]. We denote $\xi_s^u(t)$ as a trajectory that starts from a state s following control $u \in U$, where U is a compact set of control input. A trajectory can be seen as a safe trajectory if $\xi_s^u(t) \in \mathcal{K}$ for all $t \geq 0$. Define a function $l : S \mapsto \mathbb{R}$ such that $\mathcal{K} \doteq \{s \in S \mid l(s) \geq 0\}$, a function $g : s \mapsto \mathbb{R}$ such that $\mathcal{T} \doteq \{s \in S \mid g(s) \geq 0\}$. Then the backward-reachable set of $\mathcal{T}$ is formally

defined as:

$$R(\mathcal{T}) \doteq \{s \in S \mid \exists u \in U,\ \forall t \geq 0, \xi_s^u(t) \in \mathcal{T}\}. \tag{3.1}$$

We want the robot to reach the target set without entering failure set $\mathcal{F}$. This gives us the backward reach-avoid set formal definition:

$$RA(\mathcal{T};\mathcal{F}) \doteq \{s \in S \mid \exists u \in U,\ \forall t \geq 0,\ \xi_s^u(t) \in \mathcal{T} \wedge \forall \tau \in [0,t],\ \xi_s^u(\tau) \notin \mathcal{F}\}. \tag{3.2}$$

The Hamilton Jacobi safety value function $V : S \mapsto \mathbb{R}$ then can be defined as:

$$V(s) \doteq \sup_u \inf_{t \geq 0} l(\xi_s^u). \tag{3.3}$$

This value function keeps track of the minimum l obtained over time by a trajectory that starts from a state $s \in S$ following the best control u. During a finite time horizon $t \in [0,T]$, we can compute the optimal value function by computing the solution of a Hamilton-Jacobi-Bellman variational inequality:

$$0 = \min\{l(s) - V(s,t), \frac{\vartheta V}{\vartheta t} + \max_{u \in U} \nabla_x V^\intercal f(s,u)\}, \tag{3.4}$$

$$V(s,T) = l(s), \text{for all } s \in S, \tag{3.5}$$

where $\dot{s} = f(s,u)$ is defined as a continuous-time dynamic, and f is assumed bounded and Lipschitz continuous.

The discrete-time infinite-horizon counterpart can be written as:

$$V(s) = \min\{l(s), \max_{u \in U} V(s')\}, \tag{3.6}$$

which satisfies the fixed-point Bellman equation. To be used in reinforcement learning, [FLRR$^+$19] introduces a discrete-time discounted dynamic programming equation which also induces a contraction mapping on V:

$$V(s) = (1-\lambda)l(s) + \lambda \min\{l(s), \max_{u \in U} V(s')\}. \tag{3.7}$$

In a similar way, [HRRTF21] generalizes a value function formula that can consider safety and reaching $\mathcal{T}$ at the same time:

$$V(s) = (1-\lambda)\min\{l(s), g(s)\} + \lambda \min\{\max\{g(s), \max_{u \in U} V(s')\}, l(s)\}. \tag{3.8}$$

The HJ-SA gives a nice value function format suited for any reinforcement learning algorithms. Compared with other methods that provide formal safety guarantees by providing set constraints, such as control barrier functions (CBFs), HJ-SA is easier to design and adapt for the general system. However, this value function format can only compute a policy that cares most about safety rather than finding a short and safe way to reach the target. It also needs to manually distinguish the obstacles in the image and compute the function l according to the position of the obstacles.

3.3 The Hamilton Jacobi Value Iteration Network

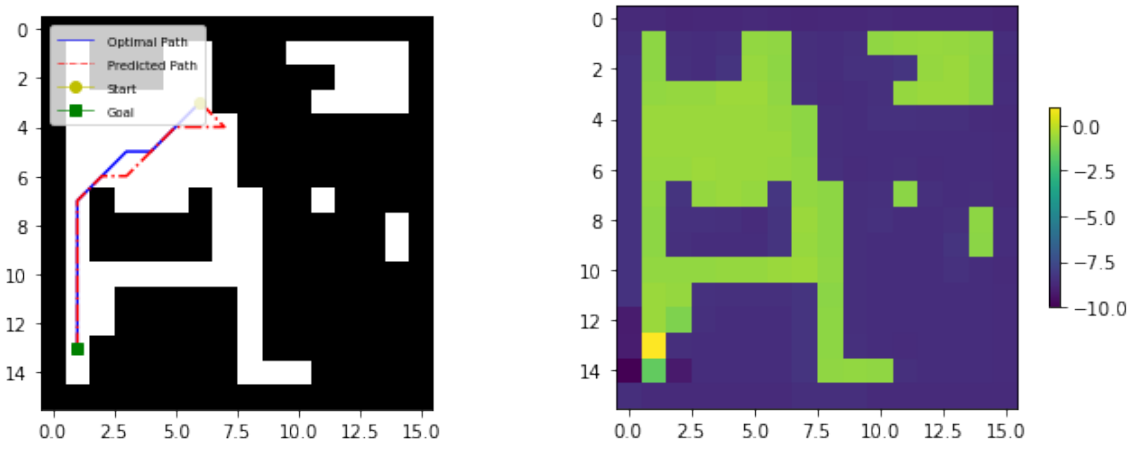
In this section, we will introduce a learning framework that integrates the VIN and the HJ-SA, called the HJ-VIN. This framework not only keeps the advantage of the VIN, which generalizes better to new, unseen domains but also integrates Hamilton Jacobi safety analysis to improve safety.

The most different part of VIN is its VI module - which embeds a value iteration algorithm in it. The value iteration algorithm can also be seen as dynamic programming. It will converge to the optimal value function Bellman equation after K iterations. Notice that in [FLRR$^+$19], the author presents a discrete-time discounted dynamic programming equation (3.7). After K iterations, this equation will also converge to the fixed point Hamilton Jacobi safety value function (3.6). These two features from the VI module and the Hamilton Jacobi value function are starting points for combining these approaches. Figure 3.2 shows the architecture of the Hamilton Jacobi value iteration (HJ-VI) module.

There is still a problem that remains. In Figure 3.2, L denotes the l value image. Notice that we do not specifically know what the function l is. What we only know about the function l, as mentioned in Section 3.3, is that the value of l can distinguish between the obstacles and the free spaces. It would be better if we could learn l. We also notice that the VIN is not only able to learn a policy but also a reward function. In figure 3.1, we plot the learned reward for the grid world task. As we can see, the rewards at the obstacles are very negative, the rewards at free spaces are slightly

negative, and the reward is positive at the goal. This learned reward from the VIN embedded all useful information we want for the function l. Thus we construct the function l by multiplying the learned reward function R with a trainable parameter θ.

Following, we will introduce the learning frame of the HJ-VIN and its complete structure with the embedded HJ-VI model. Figure 3.3 shows the HJ-VIN structure. We notice that the HJ-VIN has part of the same structure as the VIN, which makes us easier to implement transfer learning. To use the learned reward function R to construct the function l, we apply a two-stage learning framework. We first train the VIN on the training set to learn a proper reward function R. Then we load the trained parameters to our HJ-VIN model and freeze the first two convolutional layers of the HJ-VIN model. These first two layers are the same as the VIN, mapping the states to rewards. Then we train the HJ-VIN in the same training set again to learn the function l and a safe policy π.



(a) (b)

Figure 3.1: Left: A sample grid world. Right: The learned reward image

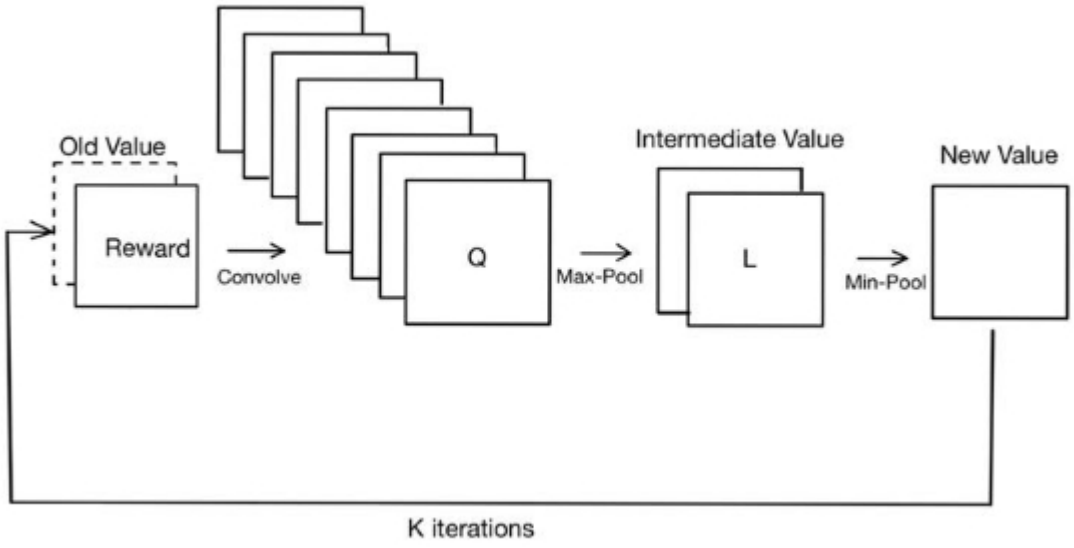


Figure 3.2: The architecture of the HJ-VI module

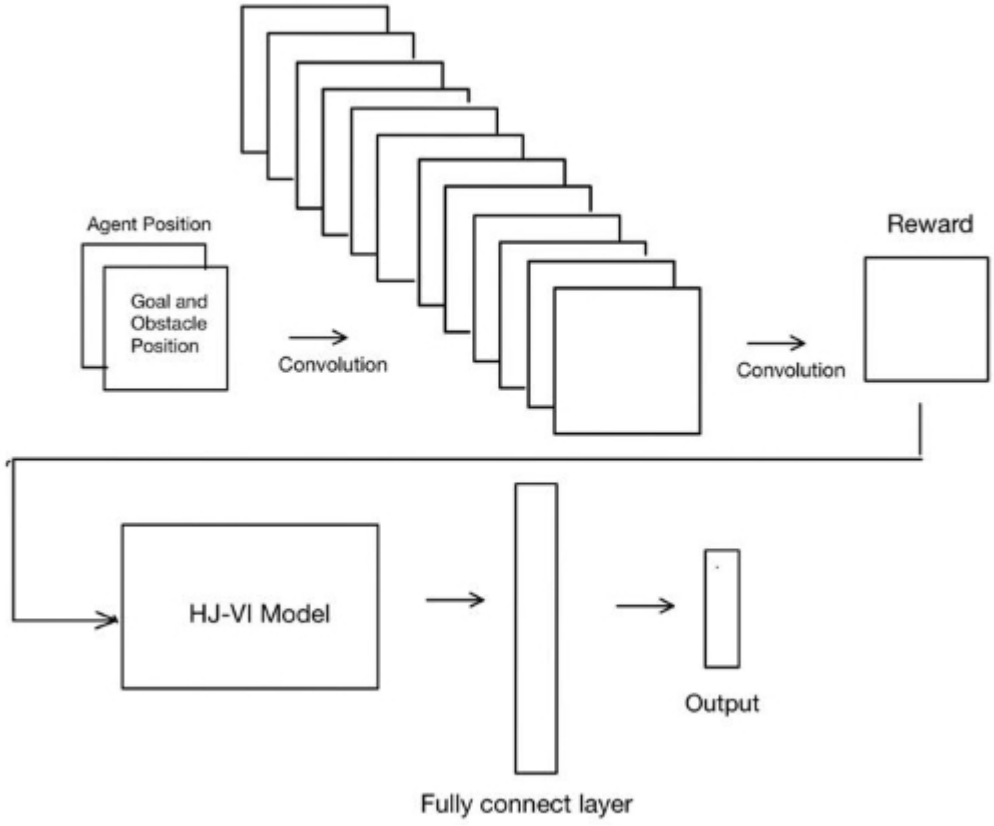


Figure 3.3: The architecture of HJ-VIN

Chapter 4

Numerical Experiments

In this Chapter, we evaluate the effectiveness of the proposed HJ-VIN on safe planning examples in the grid world. All the experiments in this Chapter were done on a Python implementation on a laptop computer with a Core i7 9750H processor and an NVIDIA GeForce RTX 2070 using PyTorch.

Consider a synthetic 16×16 - sized grid world domain with randomly placed obstacles and goal positions. We aim to safely drive the agent from the start to the goal position. We assume the proposed algorithm would learn a safe policy from these grid world instances. The learned policy can safely drive the agent to the goal position in the grid world instances with different obstacle configurations not seen in the training set. The action space A is defined as $A \doteq \{\text{N, S, E, W, NE, NW, SE, SW}\}$, which denotes to move north, south, east, west, northeast, northwest, southeast, southwest.

Our training set contains 5000 grid world domains, and at each domain, we use the Dijkstra algorithm to generate seven paths randomly. The expert behaviors are extracted from these trajectories. These expert behaviors are divided into state and action pairs. To train our model, the input of our model is a $2 \times 16 \times 16$ image and the agent's current position. The first channel of the image encodes the presence of

obstacles (0 for free space and 1 for obstacles), and the second channel denotes the goal position information. Iteration K in the HJ-VI module is chosen based on the size of the grid world, which ensures cover from the goal state to any other state. We use the cross-entropy loss function to compute the loss between the predicted action of the state and the expert behavior executed in the state. We first train the VIN in the training set to learn a reward function. Then we loaded the trained parameter into our model. The first two convolutional layers of our model map the image input to a reward image. It should be able to distinguish between obstacles, non-obstacles, and goal positions and assign proper reward value for each of them. Since these two layers have already been learned from the VIN, we freeze these two layers in the HJ-VIN model and then train the HJ-VIN again using the same training set. We evaluate our model on domain instances with different obstacle configurations than the training set to see if the learned policy can safely drive the agent to the goal position. The validation set is outside the training set.

We compare our HJ-VIN model with the VIN model. We consider two scenarios. The first scenario is mainly to evaluate the performance of the VIN and our model in the general grid world domains to show our model can also generalize well. In the second scenario, to emphasize the safety improvement of the HJ-VIN model we evaluate our model's and the VIN's performance on the domain instances where the trajectories generated by expert behaviors are detours.

Table 4.1: Performance of the VIN and the HJ-VIN for general cases

Model	Testing Accuracy	Success rate	Safety Violation rate
VIN	97.65%	96.99%	3%
HJ-VIN	95.42%	91.00%	0%

4.1 Case Study I

Table 4.1 shows the performance of the HJ-VIN model and the VIN model. Here, a trajectory succeeds if it reaches the target. A trajectory is unsafe if it hits a wall or hits an obstacle. Our model test accuracy and success rate are very close to the VIN, which shows that our model also generalizes well to new domains outside the training set. The first two figures in Figure 4.1 show two random examples of the size 16×16 grid world, with the prediction of the HJ-VIN and the expert trajectory between the goal and start position. The two figures at the bottom of Figure 4.1 show the prediction trajectories generated by the VIN and the expert trajectories.

4.2 Case Study II

Let us consider a scenario where the VIN may easily fail. The VIN essentially learns a general reward function that appropriately assigns rewards for each state so that learned policy generalizes well. However, if a safe trajectory is a long detour, the goal position is just next to the start position with several obstacles in between, just as shown in Figure 4.2. This is the case where the rewards need to be carefully assigned. The agent may take the risk to go through the obstacles but still can get the maximum expected return. For instance, if the reward is -50 for the robot hitting the walls; the reward is -5 for the robot taking any actions; the reward is 100 for

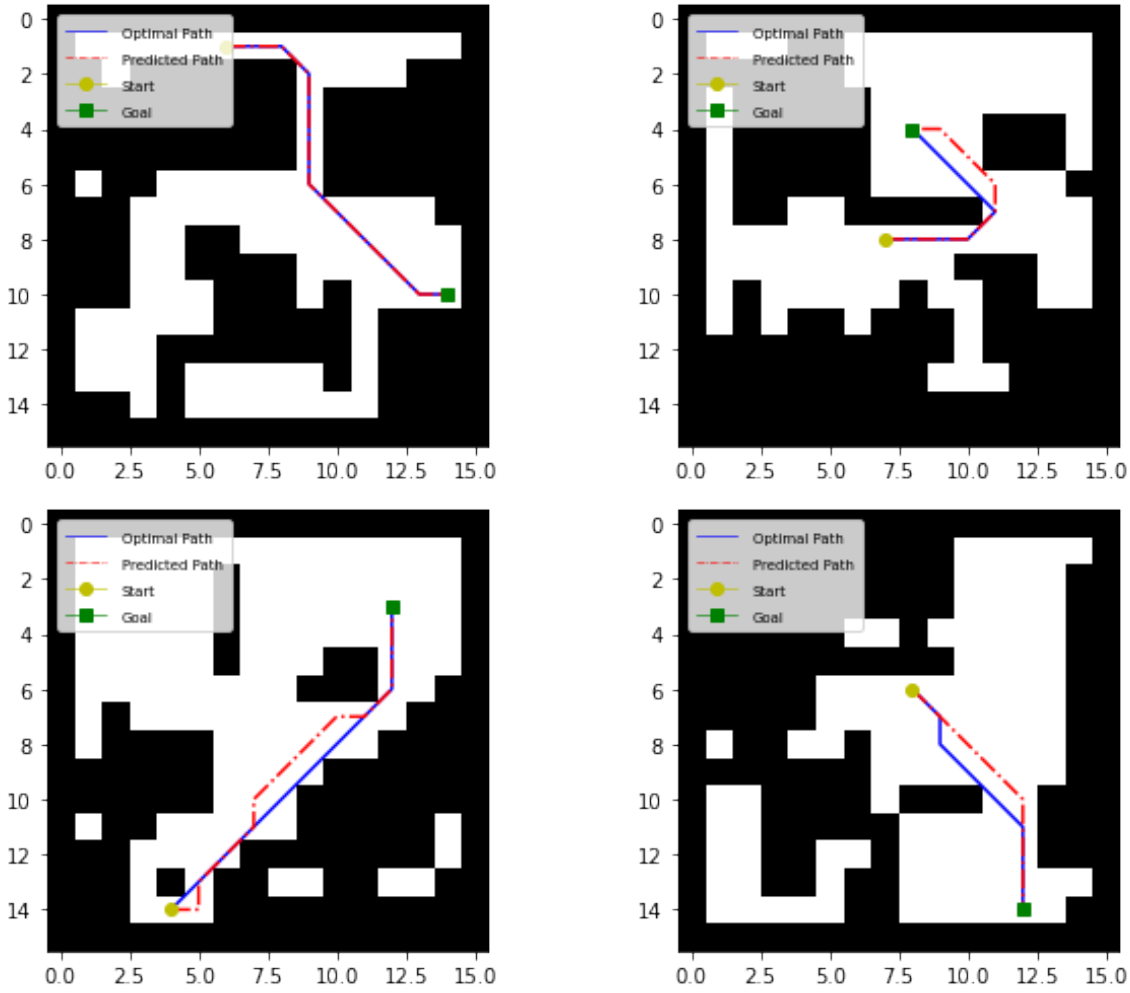


Figure 4.1: Top: Two instances that trajectories generated by the HJ-VIN. Bottom: Two instances that trajectories generated by the VIN

Table 4.2: Performance of the VIN and the HJ-VIN when expert trajectories are detours

Model	Testing Accuracy	Success rate	Safety Violation rate
VIN	97.65%	69.31%	23.54%
HJ-VIN	94.42%	64.23%	0.23%

reaching the goal. In this case, if the agent passes through the obstacles to reach the goal position, it will receive a return of 40. However, if the agent follows the expert trajectory, it can only receive a return of 25. We can see the expected return of an unsafe trajectory would be larger than an expert trajectory. For value iteration, it will be natural to choose the trajectory that maximizes the expected return.

Table 4.2 presents the performance of the VIN and the HJ-VI in this scenario. We can see that the VIN does feel a little challenging to learn a safe policy. However, the HJ-VIN performs better in this scenario than the VIN. The benefits of using the HJ-VIN in this scenario are more apparent. The success rate of the HJ-VIN is close to the VIN, but the HJ-VIN only has a 0.23% safety violation compared to a 23.54% safety violation for the VIN. This shows a safety improvement in our model. In Figure 4.2, we present some failure examples of the predicted trajectories generated by the VIN and successful examples of the predicted trajectories generated by our model.

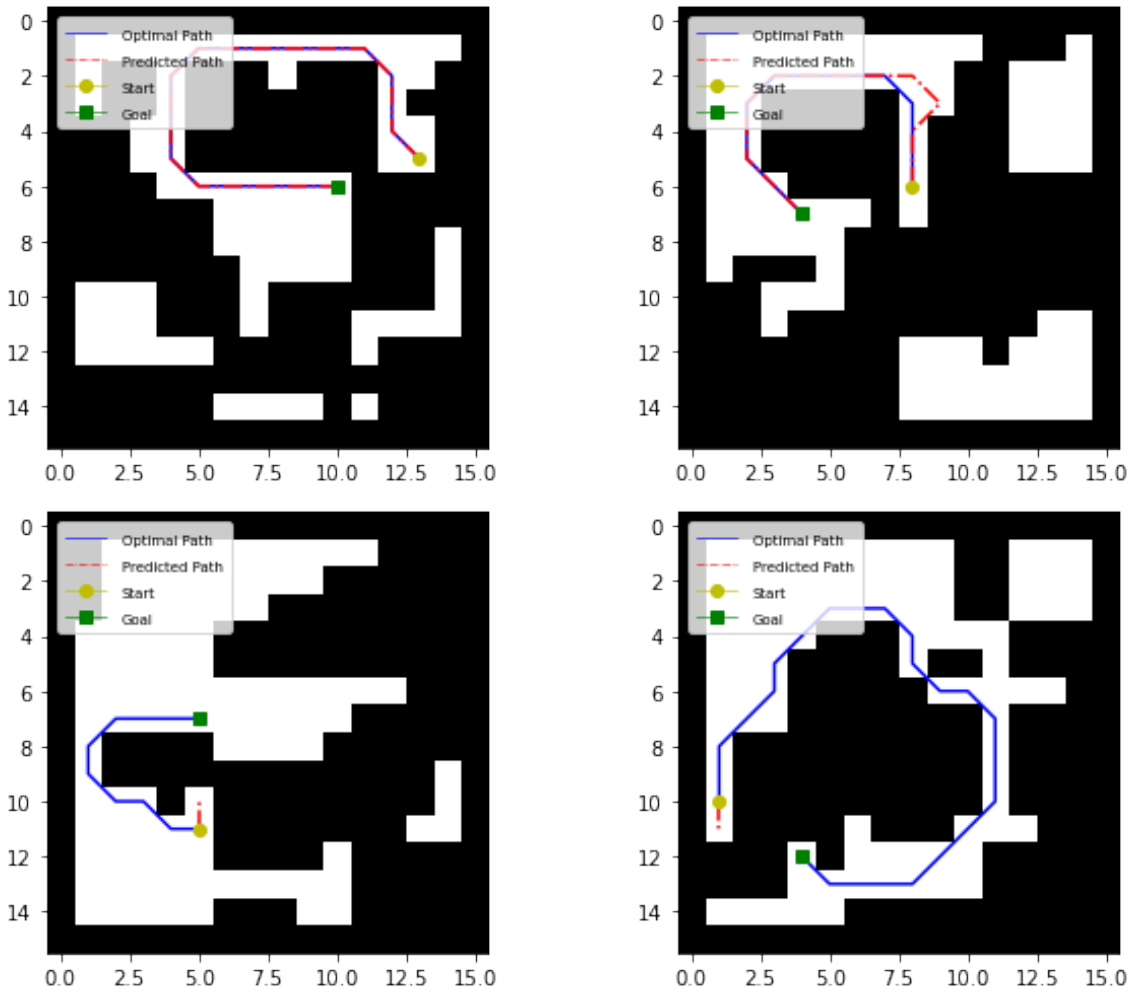


Figure 4.2: Top: Two instances with the prediction of the HJ-VIN. Bottom: Two instances with the prediction of the VIN.

Chapter 5

Conclusion

In this paper, we propose the HJ-VIN architecture for imitation learning to learn a safe policy efficiently. By following the learned policy, the agent can reach the goal without hitting obstacles or hitting the wall. We combine the VIN and the Hamilton Jacobi value function and use the learned reward R to construct function l. We show that the HJ-VIN can learn a safer policy that still generalizes to the domain instances that are outside the training set compared to the VIN. However, this work has only been evaluated on a simple grid world example. One of the future works could be to evaluate the performance of the HJ-VIN model on a more sophisticated task for example continuous control.

Bibliography

[BGH+22] Lukas Brunke, Melissa Greeff, Adam W Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P Schoellig. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems*, 5:411–444, 2022.

[CC21] I-Ming Chen and Ching-Yao Chan. Deep reinforcement learning based path tracking controller for autonomous vehicle. *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, 235(2-3):541–551, 2021.

[CNDGG18] Yinlam Chow, Ofir Nachum, Edgar Duenez-Guzman, and Mohammad Ghavamzadeh. A lyapunov-based approach to safe reinforcement learning. *Advances in neural information processing systems*, 31, 2018.

[COMB19] Richard Cheng, Gábor Orosz, Richard M Murray, and Joel W Burdick. End-to-end safe reinforcement learning through barrier functions for safety-critical continuous control tasks. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 3387–3395, 2019.

[FESF20] James Ferlez, Mahmoud Elnaggar, Yasser Shoukry, and Cody Fleming. Shieldnn: A provably safe nn filter for unsafe nn controllers. *arXiv preprint arXiv:2006.09564*, 2020.

[FLRR+19] Jaime F Fisac, Neil F Lugovoy, Vicenç Rubies-Royo, Shromona Ghosh, and Claire J Tomlin. Bridging hamilton-jacobi safety analysis and reinforcement learning. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8550–8556. IEEE, 2019.

[FP18] Nathan Fulton and André Platzer. Safe reinforcement learning via formal methods: Toward safe control through proof and learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[HFM+21] Nathan Hunt, Nathan Fulton, Sara Magliacane, Trong Nghia Hoang, Subhro Das, and Armando Solar-Lezama. Verifiably safe exploration for end-to-end reinforcement learning. In *Proceedings of the 24th International Conference on Hybrid Systems: Computation and Control*, pages 1–11, 2021.

[HRRTF21] Kai-Chieh Hsu, Vicenç Rubies-Royo, Claire J Tomlin, and Jaime F Fisac. Safety and liveness guarantees through reach-avoid reinforcement learning. *arXiv preprint arXiv:2112.12288*, 2021.

[IWA+19] Radoslav Ivanov, James Weimer, Rajeev Alur, George J Pappas, and Insup Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 169–178, 2019.

[KHJ+19] Alex Kendall, Jeffrey Hawke, David Janz, Przemyslaw Mazur, Daniele Reda, John-Mark Allen, Vinh-Dieu Lam, Alex Bewley, and Amar Shah. Learning to drive in a day. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 8248–8254. IEEE, 2019.

[OSL17] Junhyuk Oh, Satinder Singh, and Honglak Lee. Value prediction network. *Advances in neural information processing systems*, 30, 2017.

[PWET21] Marcus Pereira, Ziyi Wang, Ioannis Exarchos, and Evangelos Theodorou. Safe optimal control using stochastic barrier functions and deep forward-backward sdes. In *Conference on Robot Learning*, pages 1783–1801. PMLR, 2021.

[SCKV21] Yifei Simon Shao, Chao Chen, Shreyas Kousik, and Ram Vasudevan. Reachability-based trajectory safeguard (rts): A safe and fast reinforcement learning safety layer for continuous control. *IEEE Robotics and Automation Letters*, 6(2):3663–3670, 2021.

[TLM21] Garrett Thomas, Yuping Luo, and Tengyu Ma. Safe reinforcement learning by imagining the near future. *Advances in Neural Information Processing Systems*, 34:13859–13869, 2021.

[TWT+16] Aviv Tamar, Yi Wu, Garrett Thomas, Sergey Levine, and Pieter Abbeel. Value iteration networks. *Advances in neural information processing systems*, 29, 2016.

[WYK19] Joohyun Woo, Chanwoo Yu, and Nakwan Kim. Deep reinforcement learning-based controller for path following of an unmanned surface vehicle. *Ocean Engineering*, 183:155–166, 2019.

[XHLR21] Wei Xiao, Ramin Hasani, Xiao Li, and Daniela Rus. Barriernet: A safety-guaranteed layer for neural networks. *arXiv preprint arXiv:2111.11277*, 2021.

[XZZ22] Haoran Xu, Xianyuan Zhan, and Xiangyu Zhu. Constraints penalized q-learning for safe offline reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 8753–8760, 2022.

[YG21] Chenning Yu and Sicun Gao. Reducing collision checking for sampling-based motion planning using graph neural networks. *Advances in Neural Information Processing Systems*, 34:4274–4289, 2021.