Deep Reward Shaping from Demonstrations

Ahmed Hussein[1] Eyad Elyan[1] Mohamed Medhat Gaber[2] Chrisina Jayne[1]

[1]School of Computing Science and Digital Media
Robert Gordon University
Garthdee Road Aberdeen, UK AB10 7QB
Email: a.s.h.a.hussein@rgu.ac.uk

[2]School of Computing and Digital Technology
Birmingham City University
15 Bartholomew Row, Birmingham B5 5JU

Deep reinforcement learning is rapidly gaining attention due to recent successes in a variety of problems. The combination of deep learning and reinforcement learning allows for a generic learning process that does not consider specific knowledge of the task. However, learning from scratch becomes more difficult when tasks involve long trajectories with delayed rewards. The chances of finding the rewards using trial and error become much smaller compared to tasks where the agent continuously interacts with the environment. This is the case in many real life applications which poses a limitation to current methods. In this paper we propose a novel method for combining learning from demonstrations and experience to expedite and improve deep reinforcement learning. Demonstrations from a teacher are used to shape a potential reward function by training a deep supervised convolutional neural network. The shaped function is added to the reward function used in deep-Q-learning (DQN) to perform off-policy training through trial and error. The proposed method is demonstrated on navigation tasks that are learned from raw pixels without utilizing any knowledge of the problem. Navigation tasks represent a typical AI problem that is relevant to many real applications and where only delayed rewards (usually terminal) are available to the agent. The results show that using the proposed shaped rewards significantly improves the performance of the agent over standard DQN. This improvement is more pronounced the sparser the rewards are.

I. Introduction

Recent years have seen a rise in demand for intelligent agents capable of performing complex actions. Advances in robotics and computational capabilities provide opportunities for many potential applications such as assistive robots [1], autonomous vehicles [2] and human computer interaction [3][4]. However the challenge remains to create intelligent agents capable of robust and effective behavior. Most applications are dynamic (i.e. the agent frequently faces varying scenarios) and involve many variables and are therefore not suitable for manually designed policies. In addition, the dynamic settings mean that tailored learning methods that require expert knowledge in the learning process are not robust or don't generalize well. This motivates the development of general learning processes that require minimal expert knowledge about the task and therefore can be applied to a wider array of problems or generalize to changing scenarios. Deep learning greatly facilitates more generic learning methods by providing the ability to learn from raw sensory data. The ability of deep learning techniques to learn representations directly from raw data eliminates the need for feature extraction methods that are specifically engineered for a particular task or environment. Reinforcement learning (RL) also provides a generic framework for learning tasks. RL utilizes trial and error and learns by receiving feed back from the environment. It, therefore, does not require any description of the task or how to solve it. The only information provided to the agent is a reward based on its actions. While rewards can be designed to describe how well the agent is doing at any given moment, in more realistic settings a reward is only provided when the task is completed. This paradigm is suitable for many real life applications and facilitates using the same process to learn different problems with minimal modifications.

Recently several successful attempts combine the use of deep learning with RL to learn a wide array of tasks such as Atari games [5][6], optimal control [7][8], board and card games [9][10] and navigation [11]. Most efforts use deep learning to directly map raw state representations to action space. RL is a popular choice for learning actions because most tasks can be modeled as a Markov decision process (MDP). Moreover, optimizing a reward function arguably provides a better description of a task than optimizing a policy [12]. Learning from experience can produce robust policies that generalize to dynamic scenarios by balancing exploration and exploitation of rewards. However, finding a solution through trial and error may take too long. Especially in problems that require performing long trajectories of actions with delayed rewards. In such cases it may be extremely difficult to stumble upon rewards by chance and the time to learn a policy to maximize the rewards exponentially increases. Another draw back is that learning through trial and error may result in a policy that solves the problem differently to how a human would. Performing a task in a manner that is intuitive to a human observer may be crucial in applications where humans and intelligent agents interact together in an environment [13]. Nass et al [14] suggest that humans view computers interacting with them as social agents and that humans interact with them in a manner derived from their experiences interacting with other humans. Therefore, even with the conscious knowledge that an agent is not a human, interaction is improved when the agent behaves in a way that is familiar to its human counter part.

An alternative approach to learning actions is learning from demonstrations which aims to learn a policy that mimics a teacher's behavior. Demonstration is an intuitive way of

imparting knowledge to a learner and is easier than describing how to solve a problem [15]. Learning from demonstrations has the advantage of faster learning as it learns from good examples as opposed to random exploration. Moreover, it results in a policy that follows the teacher's way of solving the task. However, learning a direct mapping between observation and action results in a policy that generalizes poorly to unseen scenarios. The supervised policy only learns to deal with situations covered in the demonstrations. Since demonstrations only cover the optimal trajectory, if the agent deviates even slightly from that trajectory (which is expected in any machine learning application), it finds itself in an unseen situation not covered by the training data. So essentially the policy is trained using samples from a distribution that is different to the one it is evaluated on. Moreover, supervised learning needs a sufficient number of demonstrations which for deep network architectures may be large.

In this paper we propose a reward shaping method for integrating learning from demonstrations with deep reinforcement learning to alleviate the limitations of each technique. Unlike most reward shaping methods, the reward is shaped directly from demonstrations and thus does not need measures that are tailored specifically for a certain task. Moreover, deep learning is used to learn a mapping between raw observations and rewards from the demonstrations. The proposed method uses a deep convolutional neural network to learn a reward shaping function from demonstrations performed by a teacher. This function provides additional rewards based on the teacher's behavior that are added to the rewards from the environment. The augmented reward function is used to train an agent through Deep-Q-Networks (DQN) [5], a variation of Q-learning that employs deep learning. Both the supervised reward shaping network and the reinforcement learning network utilize stacked convolutional layers to learn reward estimates directly from raw pixels. This approach takes advantage of the extra information provided by demonstrations to expedite and improve reinforcement learning, while being able to generalize by learning through exploration and trial and error.

Moreover we propose an adaptive network updating method based on training loss of the Q-network to speed up and stabilize learning. A contribution of DQN is to use two networks, and freeze the reward network's learning while the policy network learns from it's predictions. The results indicate that the current established method of setting a predefined number of steps before updating the reward network is inefficient and that using an adaptive freezing period significantly improves the convergence of DQN.

The proposed method is evaluated on a 2D navigation task that provides delayed rewards and requires learning from raw visual data. Navigation is an important skill for intelligent agents due to its relevancy to a variety of applications. It is common in realistic applications to only provide a reward when the target is reached. Navigation can be a main task as in autonomous vehicle applications such as aerial vehicles [16][2][17] or land vehicles [18][19][20] or as a base skill for other tasks such as humanoid robots which need to move before performing other tasks [21][22].

In the next section we review related work. Section 3 describes the proposed methods. Section 4 details the experimental setup and the produced results. Finally Section 5 concludes the paper and provides directions for future research.

II. Related Work

In this section we present related work that utilize deep reinforcement learning and describe different methods proposed in the literature to combine learning from demonstrations and experience.

Deep learning methods have shown great success in learning from raw sensory data. This is particularly useful in problems where tailor-made features are difficult to create. In order to combine reinforcement learning with deep learning from high dimensional visual input, RL methods must be scaled to accommodate large non linear function estimators such as convolutional neural networks [8]. DQN [5][23] introduces a version of Q-learning that learns from raw visual data using convolutional neural nets. This is achieved by creating a replay buffer of training samples that are collected off-policy. Random mini-batches from the buffer are selected to train the Q-network. This method is favorable to using incoming observations for online training for two reasons [24]: Firstly, closely correlated samples violate the assumption that the training samples are independent and identically distributed (i.i.d.) and thus create an imbalanced training set. Secondly a model that uses a saved training queue is less prone to forgetting rare training samples that appear early in the learning process. In addition to introducing a training buffer, DQN proposes using a separate network (target network) to generate the estimated rewards which are used to calculate the loss of the Q-network. This approach helps stabilize learning as the target rewards stay constant while the Q-network is learning. This technique showed human level performance on several Atari games and paved the road for deep reinforcement learning methods.

In [8], the replay buffer and sampling methods introduced in DQN are adopted to an actor-critic framework to deal with continuous action spaces (such as analogue control). The action-value function in DQN only allows for a finite and limited number of actions as the neural network predicts an estimated reward for each one. This approach is replaced by an actor-critic method in which the neural network predicts what actions to perform and a critic evaluates this step based on the returned rewards. Therefore a loss function can be calculated for any action space.

In [6], an alternative method is proposed to deal with the correlation of sampled instances and non-stationarity of the updates. Instead of saving samples in a replay buffer, multiple agents are deployed (each in its own copy of the environment) in parallel. The instances sampled from all agents are used to train the single Q-network simultaneously. Since each agent will probably be exploring a different aspect of the task at the same time, the samples they provide will not be highly correlated and provide similar diversity to random mini-batches from the replay buffer. In addition to being efficient with memory and computational resources, this approach enables

gathering samples on-policy or off-policy alike. Asynchronous parallel sampling is applied to a number of deep reinforcement learning algorithms including DQN and an actor-critic method (called A3C) which achieved state of the art results on the Atari benchmark and demonstrated effective performance in a navigation task in a 3D simulated environment. Utilizing additional information about the target is used to speed up and improve deep RL in [11]. Information about the target is provided in the form of images of the target. The network takes pixels representing the target as input as well as pixels representing the current state and learns in a manner similar to A3C. The results show that the addition of target information significantly improves policy learning and allows for transfer learning to similar tasks with different targets.

Although deep reinforcement learning has shown great advancements and demonstrated effective performance on a number of problems, demonstrations can provide extra information that can be useful to learning a policy. A supervised deep learning method is applied to the same Atari benchmark in [25] using demonstrations from an offline Monte Carlo policy. The agent trained using demonstrations is shown to outperform learning from trial and error alone. Another example of demonstrations outperforming trial and error is shown in [26]. DQN is used to play a game of Pacman and the performance of reinforcement learning is compared to supervised learning. The experiments show that learning purely from trial and error failed to produce an effective policy in reasonable time. While supervised learning using training demonstrations played by the authors proved to converge much faster and produce a well performing policy.

To achieve the advantages of both approaches, many efforts have been made to combine learning from demonstrations with reinforcement learning. Early research shows that demonstrations can improve and speed up reinforcement learning as well as avoid falling in local minima [27]. A popular method of utilizing demonstrations in reinforcement learning is apprenticeship learning [12]. Apprenticeship learning does not require rewards to be explicitly provided but rather learns a reward function from the demonstrations. It is assumed the the teacher is trying to optimize an unknown reward function and thus the goal is to learn an estimate of this function from demonstrations. A policy to optimize this function is then learned via reinforcement learning. This approach has the advantage of not needing explicit rewards. However, this may affect it's generalization ability specially if the environment is dynamic and the demonstrations doesn't cover all possibilities.

In similar vein, reward shaping [28][29] aims to create a reward function, however, unlike apprenticeship learning the created function is not used on its own to reward the agent. Instead the shaped reward is augmented to the reward from the environment, thus providing extra information to speed up learning while maintaining generalization by exploring environment reward returns. In [30], it is shown that shaping a potential reward function maintains the convergence guarantees of reinforcement learning. A shaped reward function can also be used to assist traditional learning from demonstrations instead of reinforcement learning. In [31], a policy is optimized to mimic given demonstrations (how to solve the problem) while a shaped reward function provides target driven constraints to the optimization. Typically reward shaping approaches do not learn a reward function from demonstrations, but rather use prior knowledge of the task to shape the rewards. For example a function of the Manhattan distance to the target is used in [30]. However, such information about the performance of the agent may not be available in most realistic tasks. Therefore in order to make the learning process more generic and applicable to a wider range of applications, a recent version of reward shaping proposes to use demonstrations to create the reward function [32]. The shaped reward function evaluates the similarity of the action taken by the agent at the current state to the recorded demonstrations. So if the same action was taken by the demonstrator in similar states, the reward would be closer to 1. Manually engineered features for each task are used to provide representations of the states. A non-normalized multivariate Gaussian equation is used to evaluate the similarity of the state-action pairs to the demonstrations.

Another approach to incorporating demonstrations in RL is guided policy search [33] which uses differential dynamic programming (DDP) to generate guiding samples from given demonstrations. These samples are used help a policy search algorithm reach reward dense areas faster. While this method is model-free, it relies on the model based DDP to generate guiding samples which required a working model of the task's dynamics.

With the rising interest in deep reinforcement learning, some efforts explore incorporating learning from demonstrations with RL in a deep learning context. To preserve the generic design of the learning method, learning from demonstrations just as the RL algorithm should not require specific knowledge of the task. In [9], deep reinforcement learning is combined with supervised learning to train an agent to play the board game GO. The acting policy is initialized using weights trained via supervised learning on a dataset of previous games. Moreover, a value function is used to evaluate whether a game will eventually be won or lost given the current state. This function is trained using supervised learning on a specially prepared dataset of recorded games. The reinforcement learning algorithm finds a policy based on terminal rewards and the predictions of the value function. This method has proven to rival human level and significantly outperforms using only supervised learning [34].

In [7], supervised and reinforcement learning are combined to learn object manipulation tasks in a version of guided policy search. RL is initially used under conditions where details of the task are known. The performance of this policy is used as demonstrations to train a supervised convolutional neural network. The supervised policy doesn't require task knowledge such as the position of target objects and learns to perform the task from raw visual input. However, this approach limits the ability to improve the policy through exploration once the policy is learned. Apprenticeship learning is also extended to use a deep neural network to learn from raw visual input [35] which alleviates the need for manually designed reward functions or feature representations. In [36], Monte Carlo Tree Search (MCTS) Methods and deep learning are used for

reward design. This method does not utilize demonstrations in the reward design, but rather uses MCTS to generate trajectories and learns internal rewards based on those trees. The internal rewards are learned using a version of Policy-Gradient for Reward-Design (PGRD) [37] that employs a deep convolutional neural network. Similar to reward shaping, the internal rewards are then augmented to the rewards from the environment.

III. Method

This section presents the proposed method for deep reward shaping from demonstrations. The method uses a deep supervised network to learn a shaping function from demonstration. The shaped reward is added to the environment reward used by DQN [23] to speed up and improve policy learning through reinforcement learning. First we formalize the reinforcement learning and learning from demonstrations approaches.

Reinforcement learning assumes the task takes place in an environment E and is formulated as a Markov Decision Process (MDP). An experience is represented as a tuple (s, a, r, s') where s represents the state as observed by the agent, a is the action taken by the agent at state s, r is the reward received for performing action a and s' is the new state resulting from that action. While demonstrations are presented as pairs of input and output (x, y). Where x is a vector of features describing the state at that instant and y is the action performed by the demonstrator. The pair of observation and action (x, y) in demonstrations corresponds to (s, a) in the Markov Decision Process. So the demonstrator can be considered as an optimal policy π^* which provides the optimal action choice $a^* = \pi^*(s)$

The reinforcement learning algorithm works by training a deep convolutional neural network to predict the discounted reward of performing an action. Figure 1 illustrates the architecture of the network. More formally, the agent learns by optimizing $Q(s, a)$ where Q is an estimation of the return of performing a at state s which uses the recursive Bellman equation.

$$Q(s, a) = E_{s' \ E}[r + \gamma max_{a'} Q(s', a')|s, a] \quad (1)$$

Where r is the actual reward for performing a at state s, γ is a discount factor for potential future rewards and $max_{a'} Q(s', a')$ is the maximum estimated reward possible at the next state s'. If s is a terminal state (one which ends the task, regardless of result), then $Q(s, a) = r$. The function $Q(s, a)$ is learned via a deep convolutional neural network and is used to provide the agent with actions when presented with a new state. In practice a second network is used to predict the target rewards $Q'(s', a')$ used in training $Q(s, a)$. The reason for that is to provide a constant target for training while updating $Q(s, a)$ to stabilize learning. The target network is updated periodically to be equivalent to $Q(s, a)$. This raises the issue of how long to freeze the target network for before updating it. A freezing period that is too short will not allow $Q(s, a)$ to converge to the target rewards and results in unstable learning. A freezing period that is too long is inefficient since $Q(s, a)$ continues to learn outdated targets. To improve the learning efficiency we propose an adaptive method to update the target network. Convergence will occur at different rates for different tasks or even for different batches within the same task. Therefore rather than a constant freezing period, we set a condition -based on training loss- for updating the target network. Equation 2 shows the updating condition.

$$Loss = \frac{(Q(s, a) - [r + \gamma max_{a'} Q'(s', a')|s, a])^2}{2} \leq \varepsilon \quad (2)$$

Where ε is a constant indicating how small the loss needs to be before updating the targets.

The main contribution of this paper is to incorporate reward shaping from demonstrations with reinforcement learning in a deep learning context. A shaped reward is an extra reward that is derived from extra information and is added to the reward from the environment. A shaping function $F(s, a, s')$ is used to generate the shaped reward. The shaping function is added to the target reward in equation 1 yielding :

$$Q(s, a) = E_{s' \ E}[r + \gamma max_{a'} Q'(s', a')|s, a + F(s, a, s')] \quad (3)$$

Ng et al [30] proved that forming F as function of the transition between states (i.e. the difference in potential between the states) rather than a function of the current state-action pair (s, a) maintains the convergence guarantees of reinforcement learning and preserves the optimal policy. Therefore we express F as the difference between potential functions for states s and s'.

$$F(s, a, s') = \gamma max_{a'} P(s', a') - max_a P(s, a) \quad (4)$$

Where $P(s, a)$ is a function estimating the potential of the pair (s, a). We use as the potential function a convolutional neural network trained in a supervised manner on a set of collected demonstrations $D = (x, y)$. The network has the same architecture as the network used to learn $Q(s, a)$ and therefore produces an estimated potential for each action given s. The target outputs y are encoded as one-hot labels, i.e the output is a vector of possible actions with value one for the performed action and zero otherwise. The output layer of the network uses a linear activation function instead of the softmax activation function commonly used in supervised classification problems to avoid sharp potential estimates for unseen states. So P is used as a multivariate regression network rather than a classification network and the predicted potential for each action is a real number. Using a deep network for the potential function has the advantage of being able to learn from raw data and doesn't require designing representations of the demonstrations. Moreover, unlike [32] the demonstrations don't need to be stored or traversed to calculate the potential for a new state-action pair.

Utilizing this potential based function in equation 3 provides extra information from demonstrations to the reinforcement learning algorithm. This alleviates the challenges of sparse environment rewards and allows the agent to get more frequent feedback. Without this extra knowledge, the only policy available for the agent is to explore randomly until it has sampled enough experiences, which is not efficient when the rewards

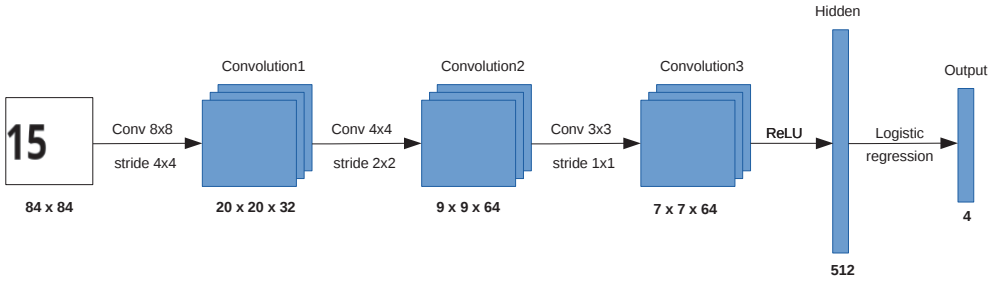

Fig. 1. Architecture of the neural network used for training

are sparse. Reward shaping speeds up reinforcement learning by limiting the need for extensive random exploration.

Algorithm 1 presents the pseudo code for reinforcement learning with deep reward shaping from demonstrations.

Algorithm 1 DQN with Deep Reward Shaping from Demonstrations

given: Teacher demonstrations $D = (x, y)$
Network $Q(s, a)$ with random weights Network $Q'(s', a')$ with random weights Network $P(s, a)$ with random weights
Empty replay buffer B
Loss threshold ε for adaptive updates
Train $P(s, a)$ on D
for episodes **do**
  for timestep $t = 1 : T$ **do**
    With probability ϵ, a_t = random action
    Otherwise $a_t = max_a Q(s_t, a; \theta)$
    Perform a_t and get r_t,s_{t+1}
    Store the tuple (s_t, a_t, r_t, s_{t+1}) in B
    Randomly select minibatch of experiences (s_i, a_i, r_i, s_{i+1}) from B:
    $F(s_i, a_i, s_{i+1}) = \gamma max_{a'} P(s_{i+1}, a') - max_{a_i} P(s_i, a_i)$
    if s_{i+1} is terminal:
      $y_i = r_i$
    else
      $y_i = r_i + \gamma max_{a'} Q'(s_{t+1}, a'; \theta) + F(s_i, a_i, s_{t+1})$
    Optimize θ using gradient descent for: $loss = \frac{(y_t - Q(s_i, a_i; \pi))^2}{2}$
    if $loss \leq \varepsilon$:
      $Q'(s', a') \leftarrow Q(s, a)$

The teacher provides demonstration as in traditional learning by demonstration problems. Unlike [9], no specially designed labeled dataset (that includes extra information other than state and action, such as evaluation measures of the performance) is needed to pre-train $Q(s', a')$ or $F(s, a, s')$, which makes the training process more generic and streamlined. The task is assumed to be an MDP where the current state represents all past information (no extra context is needed to make a decision). Therefore a single image frame is used as the agent's observation and the resulting policy is stationary (i.e doesn't require information about the current position in the trajectory).

The neural network architecture used to optimize Q and P is a deep architecture with three convolutional layers that follows the network architecture in [5] with the exception of using a single frame as input. The convolutional layers are followed by a fully connected (FC) hidden layer and finally an output layer. A rectified linear activation function (ReLU) is used for all layers apart from the output layer in which a linear activation function is used. Table I summarizes the network architecture.

TABLE I
NEURAL NETWORK ARCHITECTURE

Layer	Size of activation volume
Input	84×84
Conv1	$8 \times 8 \times 32$
Conv2	$4 \times 4 \times 64$
Conv3	$3 \times 3 \times 64$
FC	512
Output(FC)	4

IV. EXPERIMENTS

In this section we describe the experiments conducted to evaluate the proposed approach and present the results. Implementation of the proposed method including the task used for evaluation is available at https://github.com/ahmedsalaheldin/MashRL.git

A. Grid Navigation Task

The proposed method is demonstrated on a navigation task in a 2D simulated environment. The environment consists of a grid with dimensions set before starting the experiment. Each cell in the grid corresponds to a state where state s_t is a raw image of the agent's observation at frame t. The states are represented as images of the number of their respective cells. So for example cell 25 is represented by an image of the number 25. All images are 84×84 pixels and are greyscale. The grid dimensions can be set to any size and the state representations are generated automatically. We conduct experiments on grids of sizes 5×5, 15×15 and 30×30. The grid is also initialized with the agent's starting position and the position of the target. The agent is able to move in the 4 directions of the compass $a \in \{$GO_LEFT, GO_RIGHT, GO_FORWARD, GO_BACK$\}$. If the agent attempts to move off the grid, it will remain in the same state. The agent is rewarded with a positive reward (+1) only when it reaches

the target and the episode is terminated. The agent receives a negative reward (-1) for attempting to step off the grid. Figure 2 illustrates the navigation task on a 5×5 grid. Each cell consists of an image representing that state. The blue marker signifies the agent's starting position while the green marker indicates the target state. The agent and target's positions are fixed to facilitate evaluation.

Although the task is simple it provides a number of challenges typical with real applications. Firstly, no designed features are available for reinforcement learning or learning from demonstrations and the agent is required to learn solely from raw pixels. Moreover, Only terminal delayed positive rewards are provided which require the agent to perform long trajectories. For comparison, in our experiments on a 30x30 Grid, the shortest route to the target consists of 57 actions. While a recent study proposed a new complex simulator with realistic graphics [11] where the shortest trajectory to the target is typically less than 20 actions.

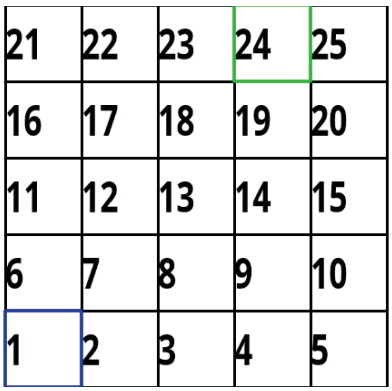

Fig. 2. Illustration of the Grid Navigation Taks

B. Experimental Setup

To evaluate the proposed method we conduct several experiments for each grid size. Firstly, the adaptive method for updating the target network is evaluated against static freeze parameters. Typically a large freeze parameter (10000) is used to ensure convergence but smaller values may result in faster training. Adaptive updating is compared against freezing the target network for 10000, 2500, 500 and 100 steps. The loss threshold ε is set to 0.02. This comparison is done using the DQN algorithm without reward shaping. The second experiment compares the proposed reward shaping approach with DQN using adaptive updating for both approaches. The second experiment is repeated using the best performing static freeze parameter (500) to show that RL can benefit from demonstrations regardless of the updating method. Finally we compare the proposed approach with DQN while using limited exploration. As mentioned in Section 3, the prior knowledge incorporated through reward shaping from demonstrations provide a base for the policy to start learning. As such, the need for extensive random exploration is limited. Exploration is controlled by the parameter ϵ which decides if the agent gathers samples randomly or according to the current learned policy. Following [5], the learning rate used is 0.00025 and ϵ decays to 0.1 over training time. The supervised training of the potential function $P(s, a)$ is executed on demonstrations performed by a deterministic optimal policy. The policy performs 5 complete trajectories of the task to gather enough samples for supervised training. For all experiments, the agent is allowed to train for 1000 epochs. After each epoch, the agent performs the current policy in a test session and the score is reported. Since success in this task is binary, the score is defined as the number of successful test sessions up to the current epoch. Using such an expanding window produces a monotonically increasing graph that reflects the rate of learning and the stability of the learned policy.

C. Results

This section presents the results of the proposed method. Figures 3 to 6 show the results of the 4 experiments conducted on grids of sizes 5×5, 15×15 and 30×30. The X axis represents the number of epochs used for training. The Y axis represents the score achieved by the agent at that test session. The score shows how many training epochs resulted in a policy that successfully solves the problem up to the current epoch. Figure 3 shows a comparison of different static freezing parameters against the proposed adaptive measure for updating the target network. The graphs show that adaptive updating achieves better results than all static parameters. For grid sizes 5×5 and 15×15, setting the freezing parameter to 100 produced the second best results with a very slightly lower curve than adaptive updating. Freezing parameter 500 closely follows, and increasing the freezing parameter results in slower learning. Similar results are shown on grid size 30×30, however, the best performing static parameter is 500 as using 100 fails to learn completely. This is due to requiring more time to converge on more complex tasks and highlights the difficulty of choosing static freezing parameters for different tasks and the advantage of using adaptive updating.

Figure 4 evaluates the proposed reward shaping method against DQN. Both approaches use adaptive updates for this comparison. The graphs show that using reward shaping results in a more stable policy faster than traditional DQN. The same observations are made in figure 5 which uses a static freezing parameter to show that the reward shaping approach does not depend adaptive updates. These results demonstrate the benefits of this paper's main contribution over standard deep reinforcement learning.

Figure 6 compares the reward shaping approach to DQN using an initial ϵ of 0.4 instead of 1. This decreases the initial exploration performed by the agent to collect samples and their corresponding rewards. The graphs show that reward shaping continues to outperform DQN. On the 30×30 grid, the performance of DQN drops significantly due to the increased search space, while reward shaping results in faster and more stable learning. This indicates that the prior knowledge extracted from demonstrations provides guidance to the sampling policy and results in a learning process that is more robust to changes in exploration parameters. A noteworthy observation is that all experiments show that the benefits of the proposed method are more pronounced the larger the grid size. This is

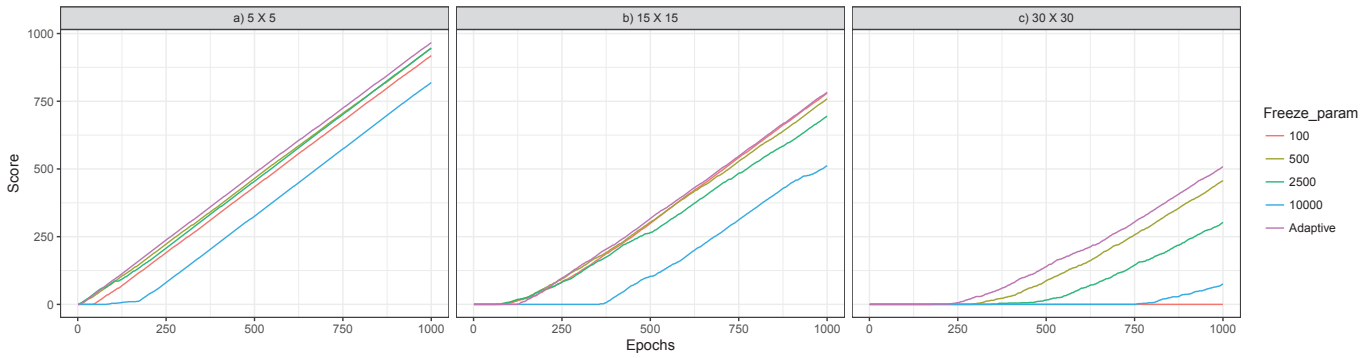

Fig. 3. Adaptive vs static network updating

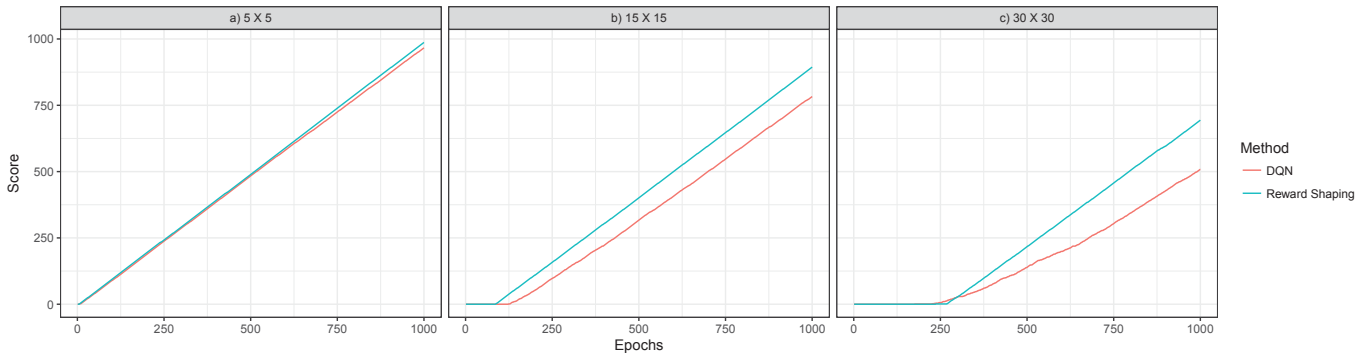

Fig. 4. Reward shaping vs DQN, adaptive updating

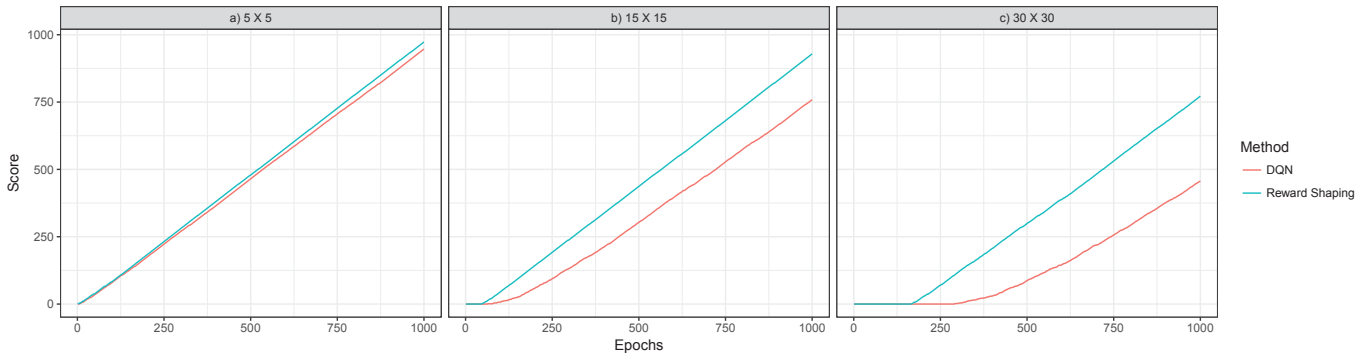

Fig. 5. Reward shaping vs DQN, freeze parameter = 500

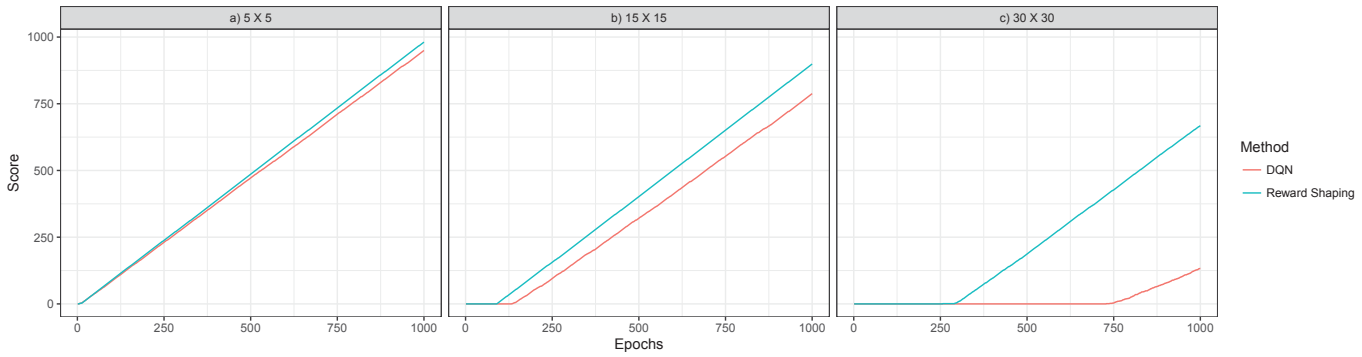

Fig. 6. Reward shaping vs DQN, $\epsilon = 0.4$

because the challenges addressed in this method increase as the grid becomes bigger. The number of states increases and the number of steps needed to complete the task also increase which makes the rewards available to the agent even sparser.

V. Conclusion

This paper proposes a novel method for deep reward shaping from demonstrations to improve deep reinforcement learning. Learning from demonstrations allows a generic approach to reward shaping and learns from raw visual data without requiring specific information about the task. Moreover, an adaptive approach to updating the target network is proposed that is shown to benefit deep reinforcement learning whether with or without the use of reward shaping and alleviates the need to manually select parameters suitable for the task. The results are conducted on a 2D navigation task and show that the proposed reward shaping approach speeds up and improves deep reinforcement learning and provides increased stability against exploration policies. Our next step is to test the proposed approach on learning various navigation tasks in a more realistic simulator [38]. We also aim to incorporate reward shaping from demonstration with A3C [6] which is considered the current state of the art in deep reinforcement learning.

References

[1] R. Bemelmans, G. J. Gelderblom, P. Jonker, and L. De Witte, "Socially assistive robots in elderly care: A systematic review into effects and effectiveness," *Journal of the American Medical Directors Association*, vol. 13, no. 2, pp. 114–120, 2012.

[2] P. Abbeel, A. Coates, M. Quigley, and A. Y. Ng, "An application of reinforcement learning to aerobatic helicopter flight," *Advances in neural information processing systems*, vol. 19, p. 1, 2007.

[3] S. Calinon and A. Billard, "Incremental learning of gestures by imitation in a humanoid robot," in *Proceedings of the ACM/IEEE international conference on Human-robot interaction*. ACM, 2007, pp. 255–262.

[4] S. Ikemoto, H. B. Amor, T. Minato, B. Jung, and H. Ishiguro, "Physical human-robot interaction: Mutual learning and adaptation," *Robotics & Automation Magazine, IEEE*, vol. 19, no. 4, pp. 24–35, 2012.

[5] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[6] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *arXiv preprint arXiv:1602.01783*, 2016.

[7] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *arXiv preprint arXiv:1504.00702*, 2015.

[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[9] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[10] J. Heinrich and D. Silver, "Deep reinforcement learning from self-play in imperfect-information games," *arXiv preprint arXiv:1603.01121*, 2016.

[11] Y. Zhu, R. Mottaghi, E. Kolve, J. J. Lim, A. Gupta, L. Fei-Fei, and A. Farhadi, "Target-driven visual navigation in indoor scenes using deep reinforcement learning," *arXiv preprint arXiv:1609.05143*, 2016.

[12] P. Abbeel and A. Y. Ng, "Apprenticeship learning via inverse reinforcement learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 1.

[13] D. Coyle, J. Moore, P. O. Kristensson, P. Fletcher, and A. Blackwell, "I did that! measuring users' experience of agency in their own actions," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2012, pp. 2025–2034.

[14] C. Nass, J. Steuer, and E. R. Tauber, "Computers are social actors," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 1994, pp. 72–78.

[15] S. Raza, S. Haider, and M.-A. Williams, "Teaching coordinated strategies to soccer robots via imitation," in *Robotics and Biomimetics (ROBIO), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1434–1439.

[16] C. Sammut, S. Hurst, D. Kedzier, D. Michie *et al.*, "Learning to fly," in *Proceedings of the ninth international workshop on Machine learning*, 1992, pp. 385–393.

[17] A. Y. Ng, A. Coates, M. Diel, V. Ganapathi, J. Schulte, B. Tse, E. Berger, and E. Liang, "Autonomous inverted helicopter flight via reinforcement learning," in *Experimental Robotics IX*. Springer, 2006, pp. 363–372.

[18] D. Silver, J. Bagnell, and A. Stentz, "High performance outdoor navigation from overhead data using imitation learning," *Robotics: Science and Systems IV, Zurich, Switzerland*, 2008.

[19] S. Chernova and M. Veloso, "Confidence-based policy learning from demonstration using gaussian mixture models," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 2007, p. 233.

[20] M. Ollis, W. H. Huang, and M. Happold, "A bayesian approach to imitation learning for robot navigation," in *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*. IEEE, 2007, pp. 709–714.

[21] S. Chernova and M. Veloso, "Confidence-based policy learning from demonstration using gaussian mixture models," in *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*. ACM, 2007, p. 233.

[22] J. Saunders, C. L. Nehaniv, and K. Dautenhahn, "Teaching robots by moulding behavior and scaffolding the environment," in *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*. ACM, 2006, pp. 118–125.

[23] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[24] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," *arXiv preprint arXiv:1511.05952*, 2015.

[25] X. Guo, S. Singh, H. Lee, R. L. Lewis, and X. Wang, "Deep learning for real-time atari game play using offline monte-carlo tree search planning," in *Advances in Neural Information Processing Systems*, 2014, pp. 3338–3346.

[26] K. Ranjan, A. Christensen, and B. Ramos, "Recurrent deep q-learning for pac-man."

[27] L.-J. Lin, "Self-improving reactive agents based on reinforcement learning, planning and teaching," *Machine learning*, vol. 8, no. 3-4, pp. 293–321, 1992.

[28] M. Dorigo and M. Colombetti, "Robot shaping: Developing autonomous agents through learning," *Artificial intelligence*, vol. 71, no. 2, pp. 321–370, 1994.

[29] M. J. Mataric, "Reward functions for accelerated learning," in *Machine Learning: Proceedings of the Eleventh international conference*, 1994, pp. 181–189.

[30] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *ICML*, vol. 99, 1999, pp. 278–287.

[31] K. Judah, A. P. Fern, P. Tadepalli, and R. Goetschalckx, "Imitation learning with demonstrations and shaping rewards." in *AAAI*, 2014, pp. 1890–1896.

[32] T. Brys, A. Harutyunyan, H. B. Suay, S. Chernova, M. E. Taylor, and A. Nowé, "Reinforcement learning from demonstration through shaping," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2015, p. 26.

[33] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of The 30th International Conference on Machine Learning*, 2013, pp. 1–9.

[34] C. Clark and A. Storkey, "Training deep convolutional neural networks to play go," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1766–1774.

[35] M. Wulfmeier, P. Ondruska, and I. Posner, "Maximum entropy deep inverse reinforcement learning," *arXiv preprint arXiv:1507.04888*, 2015.

[36] X. Guo, S. Singh, R. Lewis, and H. Lee, "Deep learning for reward design to improve monte carlo tree search in atari games," *arXiv preprint arXiv:1604.07095*, 2016.

[37] J. Sorg, S. P. Singh, and R. L. Lewis, "Internal rewards mitigate agent boundedness," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, 2010, pp. 1007–1014.

[38] A. Hussein, M. M. Gaber, and E. Elyan, "Deep active learning for autonomous navigation," in *International Conference on Engineering Applications of Neural Networks*. Springer, 2016, pp. 3–17.