Reward Shaping in Reinforcement Learning (September 2020)

Pleus, Daniel; Schmidt, Patrick; Schwanz, Florian; Vöhringer, Clemens[1]

***Abstract*—Reinforcement Learning teaches an agent to learn in an interactive environment. The convergence speed of learning is heavily dependent on the reward structure. In this paper, we research various approaches to manipulate the reward function. We apply domain knowledge, a potential-based approach and curiosity learning on the Atari games of Pong, Breakout and Freeway. We benchmark how convergence speed was influenced by using those reward shapings. The experiments show that the performance of a shaping approach is highly dependent on the structure of the game and cannot easily be generalized. We also experienced a significant risk from undesired behaviour (reward hacking) when shaped rewards are not correlated with original rewards. Therefore, we suggest "robust reward shaping", which combines manually handcrafting a reward function with automatic mechanisms to align shaped and original rewards.**

***Index Terms*— Reinforcement Learning, Reward Shaping, Q-Learning**

I. Introduction

For a long time, the application of machine learning was focused on supervised or unsupervised problems. It was either used to make predictions on labelled data or to recognize patterns in unlabelled data.

Just recently there has been massive progress in using machine learning in more interactive environments. These approaches are commonly summarized as reinforcement learning. The most popular application of reinforcement learning has been mastering games. During the 1950s Arthur Samuel already worked on a first program to play checkers [1], 1963 Michie used RL to play Tic-Tac-Toe [2] and Tesauro developed a self-teaching backgammon program in 1994 [3]. A widely recognized breakthrough was Deep Blue beating chess world champion Garry Kasparov in 1998. [4]

While those early algorithms were often based on traditional rule-based methods, the recent years have seen huge improvements from reinforcement learning techniques combined with deep learning. These improvements have been enabled by increasing computational power, cheap storage, and the availability of GPUs. In 2015 DeepMind developed an algorithm to reach human-level performance in Atari games without incorporating human knowledge just based on a high-dimensional pixel input. One of the biggest advancements was that the algorithm was even able to generalize on most Atari games. Not only could it learn to play a very specific game, but any of them. [5] Following the hype, the complexity of mastered games increased exponentially. Public attention was captured by DeepMind defeating Lee Sedol, a long-lasting grandmaster in the game of Go. The popular Chinese game is seen as one of the most complex games, averaging 10^{360} potential moves per game and has been considered unsolvable by computers for a long time. [6] Reinforcement learning is now even being researched in more complex games that mimic human and social behaviour. Complex multi-agent games like Dota2 [7] or Google Football [8] have successfully been played.

On the flip side, a lot of the current reinforcement learning progress comes at the cost of model complexity and high computational power. Common reinforcement learning algorithms require learning on millions of samples and high-performance clusters to train on. Compared to humans, that learn from very few examples, this is very inefficient. [9] While computational power is an issue of resources, the required sample size is a big hurdle for many applications, like autonomous driving, where there is not unlimited test data. A lot of recent research, like zero-shot-learning, one-shot-learning or dream-world models has therefore been dedicated to learning with fewer data points.

A phenomenon that is contributing to those challenges is called sparse rewards. Sparse rewards are prevalent in games where the agent very rarely receives feedback for his performance and therefore needs a lot of iterations to uncover the interrelations between his actions and rewards. Reward shaping solves this problem by often using domain knowledge to remodel reward functions to provide more dense rewards. [10] The latest research has been less concerned with shaping rewards and instead has focused on generalizable approaches that will make more progress towards Artificial General Intelligence (AGI).

This paper tries to close this gap by benchmarking various reward shaping approaches. We want to uncover how those approaches are influencing the convergence speed, the number of samples required and maximal rewards. The benchmarks are tested on three Atari games - Pong, Breakout and Freeway and are using a Rainbow DQN to be trained on. The results should ultimately help researchers and practitioners to make more informed resource decisions, when they are confronted with the question whether to focus on increasing model complexity, adding computational power or changing the environment by shaping rewards.

[1] All authors have contributed equally

II. Preliminaries

A. Reinforcement Learning

Reinforcement learning problems are formulated using two elements - an agent and an environment. The agent is acting to achieve rewards in an environment. His main tasks are selecting actions and learning based on the feedback he receives. The environment defines the world the agent is acting in. It provides the agent with the current state and the reward that this action invokes.

Formally speaking, reinforcement learning is a step-based interaction ($t = 0, 1, 2, 3, \dots$) between agent and environment, where the agent receives the state $s_t \in S$, while S is a set of all possible states, and a reward $r_t \in R$. It then executes an action $a_t \in A(s_t)$, where a is the set of actions available in state s. The action is determined by the agent's policy π_t, which is based on the trained representation that the agent created. After executing an action, the environment is progressing into the next state s_{t+1}. [11]

B. Q-Learning

Q-Learning is one of the most common approaches to reinforcement learning. It is model-free, so there is no explicit modelling of the environment.

Q-Learning defines reinforcement learning as a Markov decision process (MDP). An MPD is characterized by having transition probabilities

$$P_{ss'}^{a} = Pr\{s_{t+1} = s' | s_t = s, a_t = a\}$$

This means that the transition probabilities into the next state are only dependent on the last state and action. This leads to the fact that also the expected reward is only dependent on the last state and action. The expected reward can be formulated as

$$R_{ss'}^{a} = E\{r_{t+1} = r | s_t = s, a_t = a, s_{t+1} = s'\}$$ [11]

The expected rewards are estimated by Q-values, that can be interpreted as a score that evaluates an action given the current state. The Q-value is updated iteratively, which is also referenced as temporal-difference-learning. After a random initialization, the Q-values get updated the following way:

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\,[r_{t+1} + \gamma \max_a Q(s_{t+1,a}) - Q(s_t, a_t)$$

Every learning step the old Q-values is updated by the reward r_{t+1} and the best-choice action-value in $s_{t+1,a}$. γ is a factor that is discounting future rewards, while α is the learning rate that ensures a stable learning process.

One of the biggest advantages of Q-Learning is its off-policy nature. It enables Q-Learning to buffer samples and separate learning from the exploration process. This leads to samples being used more than once during the training process and samples being less correlated. Overall, these two effects greatly increase sample efficiency and convergence stability. [11]

C. Deep Q-Learning

Deep Neural Networks have evolved as very powerful tools for function approximation. They use multiple layers of neurons that are connected through activation functions. Neurons entail a linear regression, while the input is fed from previous layers. The weights and biases used by this regression are the parameters the neural network must learn. [12]

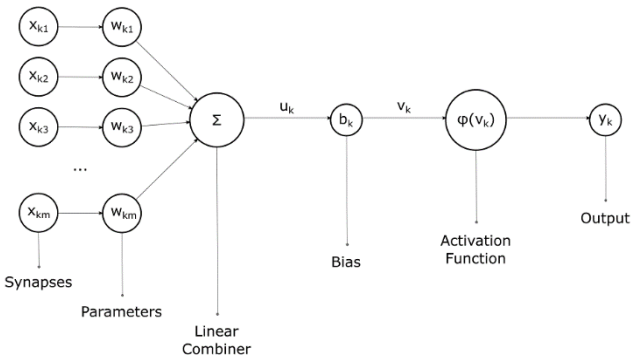

Figure 1: Neurons in Neural Networks [13]

Neural Networks also widely used Q-Learning for mapping the states (input) to the action-value (output). Its advantage compared to tabular methods is the ability to model much more complex and nonlinear functions, which is often required for reinforcement learning problems. [3] Recent research usually uses Convolutional Neural Networks (CNN) as a specific type of Neural Networks. Compared to vanilla Neural Networks they also include several convolutional and pooling layers. These layers are used for compressing the high-dimensional input that images present, by recognizing patterns that emerge from the adjacent pixel. [14]

D. Rainbow DQN

Rainbow DQN is a Deep Q-Learning algorithm developed by Hessel et. al. [15] It combines several improvements that had been made since the invention of the first DQN:

Improvement	Details
Vanilla DQN [16]	Including experience replay and target networks
Double Q-learning [17]	Decoupling action selection and evaluation in learning and therefore reducing overestimation bias
Prioritized Replay [18]	Speed learning by not uniformly selecting samples from replay buffer, but focusing on samples that are more

	insightful based on temporal difference error
Duelling Networks [19]	Improves learning by decoupling values of state and action-advantage
Multi-step learning [20]	In Q-Learning we usually just backpropagate one time-step back to update Q-values, multi-step learning is considering n previous observations
Distributional RL [21]	Not modelling point estimates of Q-values, but a probability distribution
Noisy Nets [22]	Exploration not steered through action selection (epsilon-greedy), but by adding random noise directly to the network

When published, the Rainbow DQN outperformed it predecessors significantly by improving convergence time and highest performance. This research uses the Rainbow DQN as a base model.

E. Reward Shaping

Rewards have always played a central role in enabling learning, especially in social science. The psychological research of Skinner [23] analysed how organisms behave when confronted with rewards. He conducted empirical tests with animals to find out how they can be trained on various task by providing stimuli. These experiments were referred to as reinforcement theory or operant conditioning. Skinner used reinforcers (positive rewards) and punisher (negative rewards) to shape the behaviour of laboratory rats. He scheduled those rewards in different variations:

- *Continuous Reinforcement:* The reward was given after a specific accomplishment, e.g. pressing a button. These rewards led to a slow response rate (efforts invested in achieving reward) and a fast extinction rate (duration of maintaining this behaviour).
- *Fixed Ratio Reinforcement:* Rewards were given after a fixed number of accomplishments, e.g. pressed button for 5th time. In those cases, the response rate was fast and the extinction rate was medium.
- *Fixed Interval Reinforcement:* Rewards were given for achieving success within a certain period. The response and extinction rate were medium.
- *Variable Ratio Reinforcement:* Rewards were given after an unpredictable number of accomplishments. The response rate is fast, and the extinction rate is slow.
- *Variable Interval Reinforcement:* Rewards were given after an unpredictable amount of time. The response rate is fast, and the extinction rate is slow.

The results counterintuitively showed that purely assigning rewards (continuous reinforcement) is not necessarily showing the fastest response and lowest extinction. [24] In our research, we will also modify our reward schedules to research whether these strategies yield more success.

The application of rewards shaping to reinforcement learning was introduced when Ng et al. researched how rewards can be transformed without changing the optimal policy. [25] Their goal was to improve the reward function without adapting undesirable behaviours that are not in line with the optimal policy. They suggest adding reward F to the original rewards R: $R' = R + F$

They define F as depending on the current action a, the state s and the next state s':

$$F(s, a, s') = \gamma\Phi(s') - \Phi(\mathrm{s})$$

Φ is a function that outputs the real potential of a certain state, while $\Phi: \mathrm{S} \rightarrow \mathbb{R}$. If this function aligns with the original/desired reward function the reward shaping should reduce convergence time. As Ng et. al also mentioned, it might not always be a straightforward task to find a suitable Φ, especially in complex environments. A lot of progress has been made since, introducing new approaches that we will partly adopt and benchmark in the experimentation.

One of the biggest challenges in reward shaping is still about reward hacking. It occurs whenever rewards are not potential-based and the agent is misguided to adapt a non-optimal policy. Krakovna et. al identified a list of ~60 examples where researchers were observing this misalignment. Analysing those cases, they identified two kinds of reward hacking - caused by poor reward design and reward tampering. The first kind appears when the reward function is not defined well and leads to undesired outcomes. As an example, they refer to the Coast Runners game, where additional rewards for collecting items makes the agent "forget" to finish the race and to collect items instead. Tampering refers to agents actively re-shaping the reward functions to have higher incentives afterwards. This means that the agent is creating an environment where he can increase his rewards rather than solving the task straightforwardly. [26]

F. OpenAI Gym

In 2016 OpenAI launched a software library that enables researchers to run reinforcement learning experiments in a controlled and accessible environment. It has become a de facto standard for benchmarking RL algorithms then and is used by most of the latest research. OpenAI Gym enables users to test their algorithms on multiple games, from simple text games to more complex Atari games. [27]

In this research, we trained our models on the Atari games Pong, Breakout and Freeway, which are all part of the OpenAI Gym. Pong is a game with dense rewards. The game consists of two bats that try to hit a ball in a way that the other one cannot

bounce back. Every time the other bat fails, the agent receives a reward. Breakout has the goal of hitting a ball that is destroying bricks and then bouncing back. A reward is given when a brick is destroyed. Freeway has very sparse rewards. The agent is controlling a chicken that must cross twelve lanes on a street without being hit by cars. Every time it manages to cross the street it earns a reward.

III. Methodology

As described in the previous section, sparse rewards make it very difficult for an agent to master a game. Therefore, it requires millions of iterations until significant learning progress can be observed. For some games, like the notorious Montezuma's Revenge, even after more than 50 million iterations the agent completely fails to master it. Though the Rainbow DQN can improve learning over the classical DQN, both algorithms are not even able to finish the first level of this game due to its complexity. In the following, we will describe the general principles and specific methods (domain knowledge, potential-based, curiosity learning) we used to apply reward shaping.

A. General Principles

The general idea of reward shaping is simple. In sparse reward games, the agent is only rewarded when it finishes the end of a level or reaches a final goal. In that case, the reward function would be given as a binary step function, as shown in Figure 2:

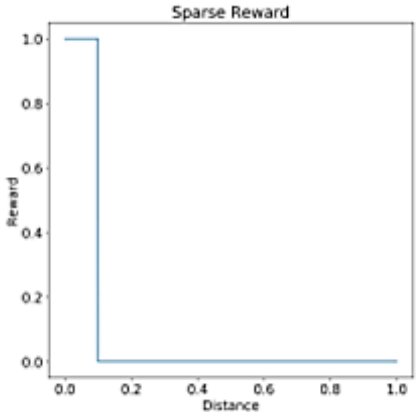

Figure 2: Reward Function - Binary Step [28]

On the one hand, those binary functions will usually guarantee a high correlation between learned and desired behaviour. For example, a reward that is given for completing a level is not prone to reward hacking. On the other hand, finishing a level is very hard, like finding the exit of a huge maze. To achieve any learning progress at all, usually millions of runs are required. Hence, to speed up the training the sparse reward function shall be replaced by a function with a smoothed gradient to allow gradual learning. In this way, the agent receives a lot more feedback, i.e. rewards, already within a current run from the environment so it knows it is getting better and/or getting closer to the exit. Such an ideal shaped reward function is displayed in Figure 3.

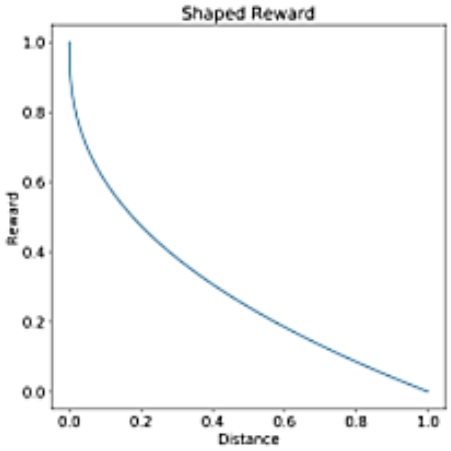

Figure 3: Reward Function - Dense Function [28]

The best way to generate a shaped function is by adding additional reward to the original reward. By reaching extra goals in the game, like reducing the distance to the exit, keeping distance to enemies, crossing obstacles, collecting special items, etc., the agent can be rewarded. By adding this new reward to the game, a transformation from a sparse reward to a dense reward environment takes place. In this way, the agent learns much quicker to interact with the environment and in the end to master the game. [15] [29]

In the presented approaches we will distinguish between shaping methods based on domain knowledge, which can only be applied for a single game, and methods, which in principle work for all available games.

B. Domain knowledge

Domain knowledge provides additional rewards based on the human understanding of a game and manually designed reward functions. Our approaches are based on pixel recognition.[2] The game output is analysed frame by frame for the occurrence of certain events and then additional rewards are applied.

In general, these rewards do not apply to other games. For example, in the game Pong we reward the agent, if it hits the ball. This rule cannot be adapted for the game Freeway, where the agent must avoid getting run over by a car when crossing a freeway. Even for games from the same genre, the graphical structure of the frames will significantly differ, so rules for extra rewards cannot be transferred easily. To derive appropriate additional rewards, it is necessary to have a good understanding of the game and its mechanisms. Their only purpose is to guide the agent through the level to accomplish the original goal. However, the additional reward can influence the agent in a way that it focuses only on reaching it and completely loses sight of

[2] Other ways of rewards shaping are in principle possible, but in our experiments, we focused entirely on pixel recognition.

finishing the game. Of course, this kind of situation shall be strictly avoided so the additional reward must be chosen very carefully. Besides, the agent has to be monitored during the training to detect such behaviour as soon as possible, since it isn't easy to predict how exactly the agent will respond to additional rewards, even if you are very familiar with the game.

C. *Potential-based*

As a potential-based approach we refer to Badnava and Mozayani, [30] who focused on incorporating knowledge from the learning process. They build upon the potential-based reward shaping from Ng, but also encourage or punish the agent based on the historical performance. The reward gets therefore normalized using the minimum and maximum of the past performance. These additional rewards are expressed as:

$$\phi(s,a,t) = \begin{cases} 0, & R(s,a) = 0 \\ 1 + \dfrac{R^{ep} - R_u^{ep}(t)}{R_u^{ep}(t) - R_l^{ep}(t)}, & O.W \end{cases}$$

R^{ep} describes the aggregated rewards in the current episode. $R_u^{ep}(t)$ describes the maximum of all episode rewards and $R_l^{ep}(t)$ is the minimum of all rewards.

D. *Curiosity Learning*

The Curiosity Learning approach by Pathak et al. [31] follows a new idea and rewards an agent for its curiosity in exploring unknown parts of an environment. Curiosity can be interpreted as an intrinsic reward that is derived from the agent's prediction error of the consequences of its own action, i.e. the error between the predicted next state and the observed next state. Since in unknown areas the prediction error is supposed to be high, the agent will be encouraged by the intrinsic reward to spend more time there, to perform actions and in the end to learn the game. So even for environments with sparse rewards the agent will be stimulated by its curiosity to explore the level. The authors show in their paper that in their analysed games, Super Mario Bros and Doom, the agent performs significantly better when their curiosity approach is used. The inner working of Curiosity Learning is displayed in figure 4. In principle, it is based on three different modules:

The **feature extraction module**, denoted by Φ, transforms the rare input states into a lower-dimensional space that only contains relevant information. Without this dimension reduction, all the calculations in this approach would be far too complex. Especially the prediction of the next state to derive the prediction error would be hard, since the whole frame, which comprises thousands of pixels, would have to be predicted. So, focusing on the relevant pixels, that can either be controlled by the agent or can affect the agent, and avoiding distraction by irrelevant information is important. A good example of the latter would be the movement of tree leaves in a breeze. Such movement could not be controlled in a game by the agent and would also have no influence on the state of an environment. Hence, neglecting this kind of information should even help to improve the prediction of new states. In the original paper, the feature extraction module is a neural network consisting of 4 convolutional layers, each with 32 filters, kernel size 3x3, a stride of 2, padding of 1 and with an ELU activation function after each layer. The output dimension of the fourth layer (=$\Phi(s_t)$ is chosen as 288. [31]

The **inverse module** tries to predict the action taken based on the features of two consecutive states $\Phi(s_t)$ and $\Phi(s_{t+1})$. This module on its own is not sufficient, since the prediction of the action a_t is not needed for the determination of curiosity, i.e. the prediction error of the new state. However, the feature extraction module is trained via the inverse module by backpropagating the prediction error between a_t and $\hat{a}_t$ to the feature extraction module. By this means the module will learn to encode states in a way that is useful for the task of the inverse model. So ideally, as discussed earlier, only features that directly or indirectly interact with the agents will be considered. This module can be designed as a simple two-layer neural network with fully connected linear layers. The inputs, Φ(st) and Φ(st+1), are concatenated into a single vector and passed into the first layer of 256 tensors. The size of the output layer depends on the number of possible actions of the game. In Pathak et al. (2017) the action number is 4 for Doom and 14 for Super Mario. A softmax function is applied in the end.

The **forward module** predicts the feature representation of the next state, denoted by $\Phi(s_{t+1})$, with the current action a_t and features of the current state $\Phi(s_t)$ as input variables. This module is also a simple two-layer neural network with linear layers. Its input is constructed by concatenating $\Phi(s_t)$ and at and a ReLu-activation function after the first layer.

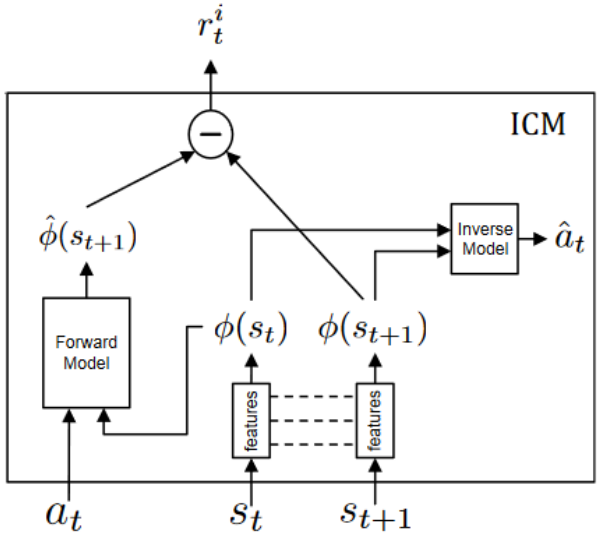

Figure 4: Curiosity Learning [31]

As shown in figure 4, in the last step the intrinsic reward r_t^i can be easily derived as the mean squared error between the prediction of the feature representation of the next state $\Phi(s_{t+1})$ and the actual observed representation $\Phi(s_{t+1})$. This reward is added to the extrinsic reward from the environment, which is

mostly zero if focusing on sparse reward environments.[3] Finally, this new derived reward will be used within the Q-learning to maximize the expected value of total rewards. Within the training of the whole model, the losses of the forward and the inverse module are considered to optimize the neural networks.

Although the curiosity approach can significantly improve the learning process, in some situations serious issues can arise. As shown in Burda et al. [32] when a source of stochastic elements exists within a level the next state can be unpredictable. This is called the noisy TV problem. A TV showing random pictures in a maze could affect an agent to stay in front of the TV to maximize its reward. In each timestamp, the agent would gain a high intrinsic reward since the next state is random and so a high prediction error from the curiosity model would be generated. Instead of searching for the exit, the agent would remain in front of the TV and just watch it. The authors show that a new curiosity reward that is based on predicting the output of a fixed and randomly initialized neural network on the next state, given the next state itself can avoid this problem. Their so-called Random Network distillation approach even works for Montezuma's Revenge and occasionally completes the first level. However, a huge drawback of this model is the high number of required iterations until the agent makes a significant learning process. So, we decided to use the classical curiosity model instead.

E. Other approaches

Other approaches, which can also be applied to all types of games, can be found in literature. Examples are hindsight experience replay, which allows an agent to learn from failures, or learning from human feedback. See for example Andrychowicz et al [33], Fang et al. [34], Mnih et al [16], Christiano et al. [35] and Griffith et al. [36]. However, these approaches are not in the scope of our experiments and will not be further elaborated here.

IV. Experiment

A. Technical Setup

Due to the high requirements that reinforcement learning puts on computational power training infrastructure had been one of the main challenges of this research.

Our first approach using Jupyter Notebooks on Colab (free version) displayed some critical performance issues not having guaranteed a full GPU for training purposes. Therefore, we moved over to evaluate enterprise cloud providers (Amazon SageMaker, Kaggle and Google Cloud) using our base model to compare the speed, the flexibility of deployments (Jupyter Notebook and pure virtual machine) and the underlying pricing. In our case, Google Cloud offered the best solution based on the stated points above and presented an extra $300 in free credits. Using Google Cloud, we developed a simple bash-based pipeline to automatically start training by using a single command which can be found in our repository. We additionally changed our development environment from Jupyter Notebooks to pure python scripts to automate the training even more and to dismiss another layer. After confronting an infinite memory consumption problem and consuming all our free credits we had the opportunity to use an internal GPU cluster. Switching over to the GPU cluster we were forced to change the automatic deployment because the cluster was only accessible through Kubernetes API. Building a Docker Container with the whole software inside

based on the official PyTorch Container image enabled us to keep on training the models.

After spending a lot of time using Jupyter Notebooks on Colab changing over to an enterprise Cloud Provider has been one essential learning for future projects. Using a reliable provider with guaranteed resources accelerate training.

B. Games

Using OpenAI Gym we benchmarked our reward shaping on the Atari games Freeway, Pong and Breakout.

Pong

The game Pong, which is based on a table-tennis theme, lets the player move an in-game paddle up and down with the objective to hit a ball which is moving horizontally between left and right. The player will earn a point when their opponent fails to return the ball to the other side. The player who earns 11 points first wins the game.

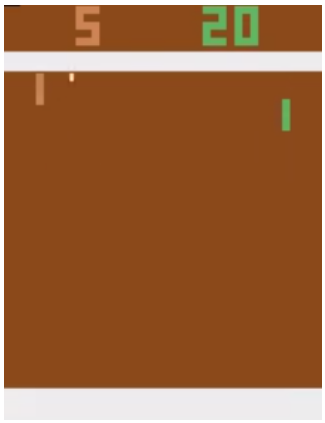

Figure 5: OpenAI Gym - Pong

Breakout

In Breakout the top of the screen is filled with lines of brick which the player who controls a horizontally moving paddle on the bottom of the screen needs to destroy. They do so by hitting the bricks with a ball which returns after hitting a brick or the wall. The player has 5 turns to destroy as many bricks as he can. After failing to touch the ball when returning to the player, the player loses one turn.

[3] The authors even showed in their paper that the extrinsic reward can be completely neglected, and learning solely based on intrinsic reward is possible.

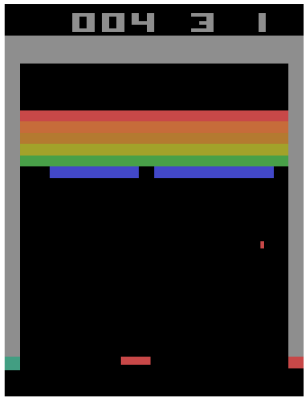

Figure 6: OpenAI Gym - Breakout

Freeway

The main objective of freeway is to move a chicken safely over a ten-lane highway with passing cars from the bottom of the display to the very top to earn a point. In 120 seconds time, the player must gain as many points as he can. Getting hit by a car will cause the chicken to be pushed down.

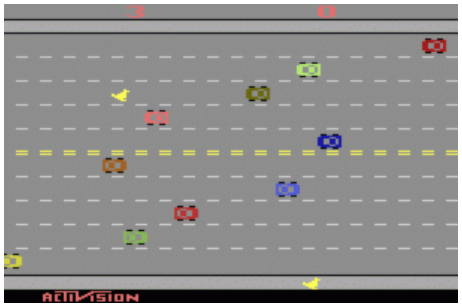

Figure 7: OpenAI Gym - Freeway

C. Implementation

For the implementation we developed a modular application that can combine a base model, various shaping approaches, and multiple auxiliary modules. It was developed in Python 3.8. The GitHub repository can be found under the following link: https://github.com/fom-big-data/fom-openai-gym-rl

Base model

We developed a base model that serves as a benchmark for the applied reward shaping. This research used a custom rainbow DQN implementation developed in PyTorch. In addition to a vanilla DQN it is using double Q-learning, prioritized replay, duelling networks, multi-step learning, distributional RL and noisy nets. The neural network uses three convolutional layers, followed by two fully connected layers. These layers (see noisy layers) also add noise to weights and biases to regularize the training process. The last layer outputs the distributional Q-value for each action (see distributional RL). The individual layers were connected by ReLU activation functions.

We used the following hyperparameter to initialize the training:

- Batch Size = 32
- Learning Rate = 0.0001
- Gamma = 0.99
- Num_Atoms = 51
- Vmin = -10
- Vmax= 10
- Eta = 0.0
- Beta = 0.0
- Lambda =0.0
- Optimizer: ADAM
- Loss Function: custom log function, see multi-step learning and distributional RL

Reward Shaping

As described earlier, four different approaches of Reward Shaping were used - Domain Knowledge, Curiosity Learning, a potential-based method and regularization methods.

Domain knowledge approaches are based on using human knowledge to provide additional rewards. We implemented a pixel-based object recognition that was able to detect and locate elements within the games. A challenge was to keep computational cost low, so we decided not to analyse every pixel individually, but to focus on every fourth. Based on the recognized element positions we provided additional rewards. These are listed below:

Game	**Shaping**	**Explanation**
Pong	Racket hits ball	
	Racket covers ball	Racket is horizontally on same height as ball
	Racket close to ball - linear	Distance based on Manhattan distance
	Racket close to ball - quadratic	Distance based on Euclidean distance
	Opponents hits ball	Negative reward
	Opponents racket covers ball	Negative reward: Racket is horizontally on same height as ball
	Opponents racket - linear	Negative reward: Distance based on Manhattan distance
	Opponents racket close to ball - quadratic	Negative reward: Distance based on Euclidean distance
Breakout	Racket hits ball	
	Racket covers ball	Racket is vertically on same height as ball
	Racket close to ball - linear	Distance based on Manhattan distance
	Racket close to ball - quadratic	Distance based on Euclidean distance

Freeway	Distance walked	Height in current step
	Distance to car	Sum of distance to all cars

Curiosity Learning and the potential-based approach were implemented on top of the base model and could be activated on demand. They were implemented using the same parameters that were described within the methodology section.

Regularization methods are commonly used to prevent overfitting in regressions or neural networks. In the context of our research, their goal was to prevent overfitting. We applied a reward dropout, which provides rewards with a predefined probability. For example, a dropout rate of 0.6 will only provide an additional reward in 40% of the cases where it was applicable.

Auxiliary modules

As auxiliary supplements, we developed an event logger, training scripts, a gif creator and telegram notifications. The event logger tracks all relevant metrics like loss, original rewards, additional rewards, iterations elapsed and time elapsed. Furthermore, it is already plotting those metrics using matplotlib. The training scripts automate the training on cloud infrastructure by containing a pre-defined set of instructions and variables. The gif creator uses the output frames to create gifs to visually inspect the training progress. Telegram notifications were sent to make the training process more transparent. After a fixed number of iterations, a message including the key metrics and a gif were sent.

D. Results

Metrics and Repeatability

The metrics we used to measure the quality of our reward shaping approaches is the number of frames needed to achieve a certain number of points for the first time. To make the latter more robust towards volatility we took the average score over the last 50 episodes. Since environments differ a lot in terms of the maximum score that can be gained the metrics need to be defined individually. As described in IV we take a training session without any reward shaping as a baseline to compare the results of different reward shaping approaches against.

To ensure stable and reproducible results across different reward shaping approaches we apply different measures. First, we set a seed for random number generation. Second, we use the same hyper-parameters such as batch size, learning rate, gamma, epsilon, number of atoms, Vmin, Vmax, target update rate, size of the replay memory and the total number of frames for each training session for a given environment.

Pong

The score range in Pong reaches from -21 (when the opponent achieves a perfect score) to +21 points (the agent achieves a perfect score) per episode. The number of frames per episode can vary significantly depending on the number of rallies between the agent and the opponent. As thresholds for this environment we picked the number of frames necessary to achieve an average result of -15, -10, -5, 0, 5, and 10 points respectively over a 50-frame floating window.

	≥ -20	≥ -15	≥ -10	≥ -5	≥ 0	≥5	≥ 10	≥ 15
no reward shaping	2k	625k	1001k	1123k	1264k	1339k	1423k	-
potential-based	**1k**	-	-	-	-	-	-	-
curiosity-based	1k	707k	896k	**1035k**	**1093k**	**1164k**	**1273k**	**1589k**
positive domain-based	243k	**368k**	**879k**	-	-	-	-	-
positive domain-based + dropout	1k	681k	-	-	-	-	-	-
positive + negative domain-based	1k	234k	-	-	-	-	-	-
positive + negative domain-based + dropout	1k	588k	-	-	-	-	-	-

Figure 8: Pong - Threshold Rewards

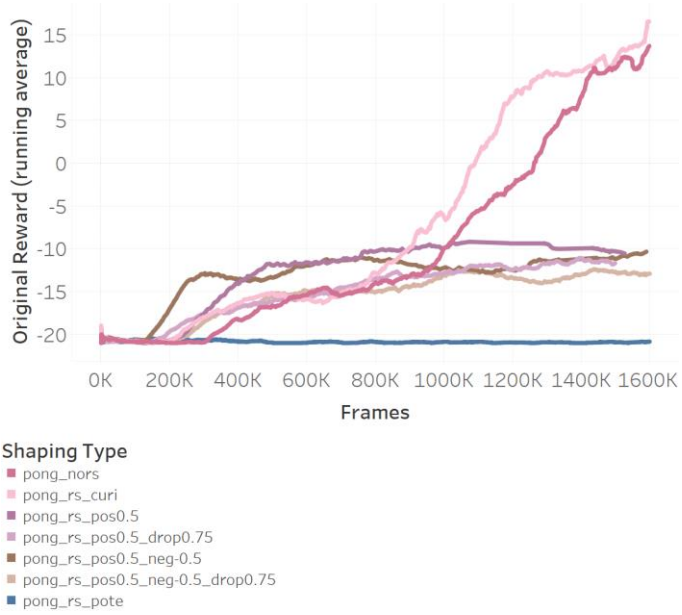

Figure 9: Pong - Rewards per Episode

In our setup, the base model is one of two approaches that surpass a floating average reward better than –5 points per game. The other one being a curiosity-based approach which performs slightly better. The potential-based approach does not indicate learning progress at all and fails to reach any of the predefined thresholds. Using a curiosity-based approach we reached a level that is similar to base model but converges slightly faster.

All approaches based on domain knowledge manage to reach the first threshold of -15 points in fewer frames than the baseline. All but one of them also reach the second threshold for -10 points faster than the baseline. While all domain-knowledge-based reward shaping approaches show good progress during the first period of a training session none of them manages to reach the maximum score of 21 points. After around 1 million frames all of them are outperformed by the

baseline.

Breakout

The maximum score that can be gained in an episode of Breakout is 108 points which correspond to the total number of bricks on the screen. The number of frames per episode varies depending on the number of times the agent manages to hit the ball. As thresholds for this environment, we picked the number of frames necessary to achieve an average result of 10, 20, 30, 40, 50, 60, 70 and 80 points respectively over a 50-frame floating window.

	≥ 10	≥ 20	≥ 30	≥ 40	≥50	≥60	≥70	≥80
no reward shaping	109k	196k	616k	1281k	-	-	-	-
potential-based	118k	221k	565k	-	-	-	-	-
curiosity-based	101k	198k	354k	1149k	-	-	-	-
positive domain-based (racket)	**43k**	**87k**	**276k**	**509k**	**634k**	**767k**	**927k**	**1141k**
positive domain-based (racket) + dropout	90k	902k	-	-	-	-	-	-
positive domain-based (block) + dropout	111k	217k	938k	-	-	-	-	-
positive domain-based (block)	155k	274k	-	-	-	-	-	-
positive domain-based (block) + potential-based + dropout	**114k**	**979k**	-	-	-	-	-	-

Figure 10: Breakout - Threshold Rewards

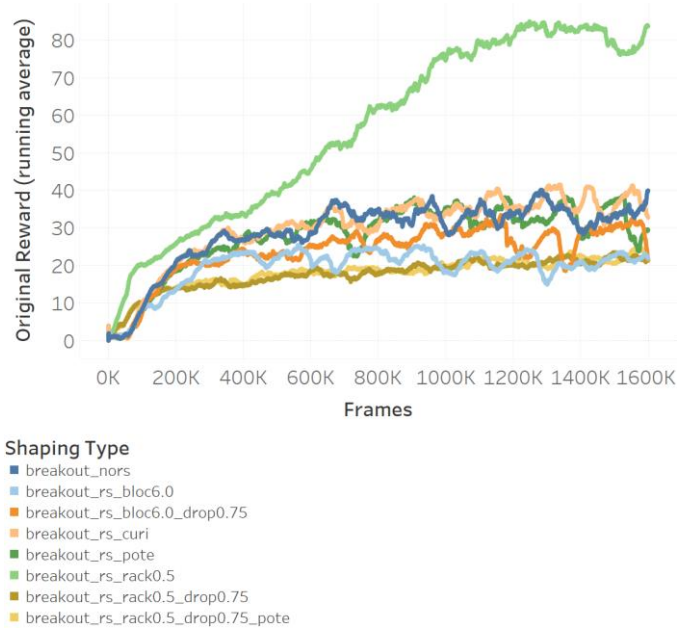

Figure 11: Breakout - Rewards per Episode

Potential-based reward shaping performs on a comparable level to the base model but fails to reach the threshold of 40 points. The curiosity-based approach also performs similarly to the baseline in terms of rewards per episode. Adding positive rewards based on domain knowledge that incentivizes the agent to position close to the ball manages to reach each threshold first and achieves the highest maximum score. Contrary to our expectation, adding a dropout to the domain knowledge approach makes the performance fall behind the baseline. Incentivizing the agent to hit blocks of a higher level in order to "dig a tunnel" through all layers (which opens the possibility of the ball being trapped above the blocks which would results in consecutive points independently of the agent's actions) performs worse than the baseline for the first three thresholds. Also adding a dropout of 75% to this approach only makes it reach the third threshold. A training session in which we combined domain knowledge with a potential-based approach falls behind most of the other approaches.

Freeway

In Freeway the maximum score is based on the number of times the agent crosses the street and is therefore theoretically unlimited. Though, there is a practical limit of around 26 points due to the time limit of 2 minutes per episode. Due to this fact, the number of frames per episode stays almost constant. As thresholds for this environment, we picked the number of frames necessary to achieve an average result of 5, 10, 15 and 20 points respectively over a 50-frame floating window.

	≥ 5	≥ 10	≥ 15	≥ 20
no reward shaping	63k	90k	121k	147k
potential-based	31k	45k	74k	119k
curiosity-based	12k	18k	**39k**	**98k**
positive domain-based (horizontal)	**8k**	**14k**	-	-
positive domain-based (horizontal) + dropout	16k	27k	51k	-
positive domain-based (horizontal + vertical) + dropout	49k	-	-	-

Figure 12: Freeway - Threshold Rewards

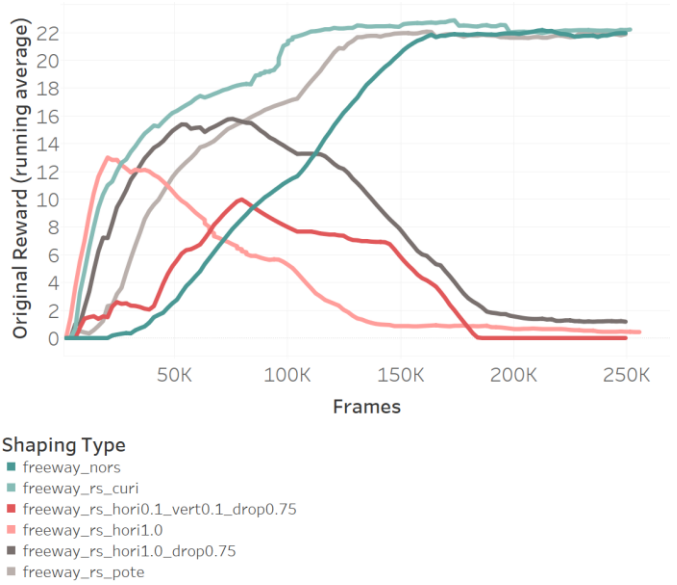

Figure 13: Freeway - Rewards per Episode

Potential-based reward shaping reaches every threshold

before the base model. So does the curiosity-based approach which reaches each of the thresholds even faster. Using domain knowledge to add an additional reward based on the horizontal distance between the agent and an approaching car leads to the quickest accomplishment of the first two thresholds. Though, after reaching a maximum of 13 points the score starts to decrease. A countermeasure to prevent this early decline is to use a dropout of 75% which only adds the domain-based reward in randomly selected 25% of all frames. This approach leads to a more platykurtic curve with a higher maximum value. Eventually, the performance of this setup also declines after reaching a maximum value. Combining different domain knowledge approaches and a dropout of 75% performs worse than any other of the described setups.

V. Conclusion

Comparison of reward shaping approaches and challenges

Across the three environments under test, the effect of different reward shaping mechanisms varies immensely. In Pong – an environment with a relatively dense reward – none of our approaches could outperform the baseline in the long run. This is even though most approaches achieve better results in the first stages of a training session. In Breakout – which also has a dense reward – we managed to find a setup using domain knowledge that performs better than the baseline in all metrics. In Freeway - a game with a sparse reward – generic reward shaping methods worked better than the baseline while domain-knowledge based approaches the performance breaks down after reaching a maximum.

In future experiments, we may use a two-phase approach in which we first figure out which kind of reward shaping approach works best for a given environment followed by a detailed optimization of hyperparameters.

Major challenges in reward shaping can be seen in a) to pick a suitable reward shaping method for a given game b) to ensure constant improvement of the approach, c) to prevent a decline of performance after reaching a maximum, d) to fine-tune hyper-parameters for approaches based on domain knowledge.

Methodology and approach

During the experimental phase of this project we learned that working incrementally turned out to be more productive than starting with a big-bang approach. The foundation of all experiments must be a solid and well-understood base model that can be easily configured, started, and extended by new approaches.

Environment and automation

To run training sessions efficiently and cost-effective it is recommended to automate steps in terms of the preparation and shutdown of a virtual environment, the execution of a training session and the transmission of (intermediate) results. Another key learning of our experiments is to start profiling of GPU, CPU, RAM and storage early to prevent failing training sessions due to memory leaks. Making use of a professional cloud provider saves efforts for maintenance in the long run.

VI. Future Work

One of the most prevalent challenges for reward shaping is managing the trade-off between convergence time and robustness. While a very specific shaping is often able to accelerate learning significantly, it is also prone to reward hacking. A possible solution, in this case, would be to select a less prescriptive shaping which will decrease the possibility of reward hacking, but ultimately lead to slower learning.

We are therefore proposing robust reward shaping as a new field of research. Robust reward shaping can be described as algorithms that are actively preventing reward hacking by evaluating shapings. There are multiple approaches that also have been partly tested within our research:

- Weaning: The environment is slowly decaying the provided rewards. In this way, there is a strong reward shaping effect upfront, while the agent is moving into its original optimum later. This will not lead to better maximum performance but increase the convergence speed.
- Dropout: Additional rewards were given randomly instead of deterministic. The original reward is still the strongest source of reinforcement.
- Historical Context: The strength of additional rewards is based on historical performance.
- Normalization: The original reward for a specified period (e.g. epoch) is used to normalize the additional reward to a pre-set level (e.g. 0.5 of the original reward)
- Correlation: The last occurrences of original and additional rewards are checked for correlation. If they are uncorrelated the additional rewards will automatically adjust.

More thorough research in robust reward shaping can ultimately combine the best of both worlds - domain knowledge of humans and the unbiased learning that machines do.

References

[1] G. Wiederhold and J. McCarthy, “Arthur Samuel: Pioneer in Machine Learning,” *IBM J. Res. & Dev.*, vol. 36, no. 3, pp. 329–331, 1992, doi: 10.1147/rd.363.0329.

[2] O. Child, “Menace: the Machine Educable Noughts And Crosses Engine,” *chalkdust*, 2016, 2016.

[3] G. Tesauro, “TD-Gammon, a Self-Teaching Backgammon Program, Achieves Master-Level Play,” *Neural Computation*, vol. 6, no. 2, pp. 215–219, 1994, doi: 10.1162/neco.1994.6.2.215.

[4] M. Campbell, A.J. Hoane, and F.-h. Hsu, “Deep Blue,” *Artificial Intelligence*, vol. 134, 1-2, pp. 57–83, 2002, doi: 10.1016/S0004-3702(01)00129-1.

[5] V. Mnih *et al.*, “Playing Atari with Deep Reinforcement Learning,” 2013. [Online]. Available: http://arxiv.org/pdf/1312.5602v1

[6] T. Cazenave, “Residual Networks for Computer Go,” *IEEE Trans. Games*, vol. 10, no. 1, pp. 107–110, 2018, doi: 10.1109/TCIAIG.2017.2681042.

[7] OpenAI *et al.*, "Dota 2 with Large Scale Deep Reinforcement Learning," Dec. 2019. [Online]. Available: http://arxiv.org/pdf/1912.06680v1

[8] K. Kurach *et al.*, "Google Research Football: A Novel Reinforcement Learning Environment," Jul. 2019. [Online]. Available: http://arxiv.org/pdf/1907.11180v2

[9] A. Irpan, *Deep Reinforcement Learning Doesn't Work Yet.* [Online]. Available: https://www.alexirpan.com/2018/02/14/rl-hard.html

[10] M. D. Buhmann *et al.*, "Reward Shaping," in *Encyclopedia of Machine Learning*, C. Sammut and G. I. Webb, Eds., Boston, MA: Springer US, 2010, pp. 863–865.

[11] R. S. Sutton and A. G. Barto, "Reinforcement Learning: An Introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, 1998, doi: 10.1109/TNN.1998.712192.

[12] Y. Bengio, I. Goodfellow, and A. Courville, *Deep learning*. Massachusetts: MIT Press, 2017.

[13] S. Rowe, *Introduction to Neurons in Neural Networks.* [Online]. Available: https://medium.com/artificial-neural-networks/introduction-to-neurons-in-neural-networks-71828d040a65

[14] K. O'Shea and R. Nash, "An Introduction to Convolutional Neural Networks," Nov. 2015. [Online]. Available: http://arxiv.org/pdf/1511.08458v2

[15] M. Hessel *et al.*, "Rainbow: Combining Improvements in Deep Reinforcement Learning," Oct. 2017. [Online]. Available: http://arxiv.org/pdf/1710.02298v1

[16] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, 2015, doi: 10.1038/nature14236.

[17] H. van Hasselt, A. Guez, and D. Silver, "Deep Reinforcement Learning with Double Q-learning," Sep. 2015. [Online]. Available: http://arxiv.org/pdf/1509.06461v3

[18] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized Experience Replay," Nov. 2015. [Online]. Available: http://arxiv.org/pdf/1511.05952v4

[19] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. d. Freitas, "Dueling Network Architectures for Deep Reinforcement Learning," Nov. 2015. [Online]. Available: http://arxiv.org/pdf/1511.06581v3

[20] K. D. Asis, J. F. Hernandez-Garcia, G. Z. Holland, and R. S. Sutton, "Multi-step Reinforcement Learning: A Unifying Algorithm," 2018. [Online]. Available: http://arxiv.org/pdf/1703.01327v2

[21] M. G. Bellemare, W. Dabney, and R. Munos, "A Distributional Perspective on Reinforcement Learning," Jul. 2017. [Online]. Available: http://arxiv.org/pdf/1707.06887v1

[22] M. Fortunato *et al.*, "Noisy Networks for Exploration," Jun. 2017. [Online]. Available: http://arxiv.org/pdf/1706.10295v3

[23] B. F. Skinner, *The behavior of organisms: An experimental analysis*. New York: Appleton, 1938.

[24] S. McLeod, *Skinner - Operant Conditioning.* [Online]. Available: https://www.simplypsychology.org/operant-conditioning.html

[25] A. Y. Ng, D. Harada, and S. Russell, "Policy invariance under reward transformations: Theory and application to reward shaping," in *In Proceedings of the Sixteenth International Conference on Machine Learning*, 1999, pp. 278–287.

[26] V. Krakovna *et al., Specification gaming: the flip side of AI ingenuity.* [Online]. Available: https://deepmind.com/blog/article/Specification-gaming-the-flip-side-of-AI-ingenuity

[27] G. Brockman *et al.*, "OpenAI Gym," Jun. 2016. [Online]. Available: http://arxiv.org/pdf/1606.01540v1

[28] Bonsai, *Deep Reinforcement Learning Models: Tips & Tricks for Writing Reward Functions.* [Online]. Available: https://medium.com/@BonsaiAI/deep-reinforcement-learning-models-tips-tricks-for-writing-reward-functions-a84fe525e8e0

[29] Y. Burda, H. Edwards, D. Pathak, A. Storkey, T. Darrell, and A. A. Efros, "Large-Scale Study of Curiosity-Driven Learning," Aug. 2018. [Online]. Available: http://arxiv.org/pdf/1808.04355v1

[30] B. Badnava and N. Mozayani, "A new Potential-Based Reward Shaping for Reinforcement Learning Agent," Feb. 2019. [Online]. Available: http://arxiv.org/pdf/1902.06239v2

[31] D. Pathak, P. Agrawal, A. A. Efros, and T. Darrell, "Curiosity-driven Exploration by Self-supervised Prediction," May. 2017. [Online]. Available: http://arxiv.org/pdf/1705.05363v1

[32] Y. Burda, H. Edwards, A. Storkey, and O. Klimov, "Exploration by Random Network Distillation," Oct. 2018. [Online]. Available: http://arxiv.org/pdf/1810.12894v1

[33] M. Andrychowicz *et al.*, "Hindsight Experience Replay," Jul. 2017. [Online]. Available: http://arxiv.org/pdf/1707.01495v3

[34] M. Fang, T. Zhou, Y. Du, L. Han, and Z. Zhang, "Curriculum-guided Hindsight Experience Replay," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d.\t. Alché-Buc, E. Fox, and R. Garnett, Eds.: Curran Associates, Inc, 2019, pp. 12623–12634. [Online]. Available: http://papers.nips.cc/paper/9425-curriculum-guided-hindsight-experience-replay.pdf

[35] P. Christiano, J. Leike, T. B. Brown, M. Martic, S. Legg, and D. Amodei, "Deep reinforcement learning from human preferences," Jun. 2017. [Online]. Available: http://arxiv.org/pdf/1706.03741v3

[36] S. Griffith, K. Subramanian, J. Scholz, C. L. Isbell, and A. L. Thomaz, "Policy Shaping: Integrating Human Feedback with Reinforcement Learning," in *Advances in Neural Information Processing Systems 26*, C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, Eds.: Curran Associates, Inc, 2013, pp. 2625–2633. [Online]. Available: http://papers.nips.cc/paper/5187-policy-shaping-integrating-human-feedback-with-reinforcement-learning.pdf