|  |  |  |
| --- | --- | --- |
| **FA 3613** | | |
| Adversarial Multi-Agent Transfer Learning | | |
| **Amal Chulliyat Jose** | | |
|  | | |
| **Basis** | | |
|  | Examiner: | Jun.-Prof. Dr.-Ing. Andrey Morozov |
|  | Supervisor: | Joachim Grimstad, M.Sc.Ing |
| Begin: 16.10.2023 | | End: 01.04.2024 |
|  | |  |

**Document Version Management**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Version | Author | QA | Date | Status | Changes |
| 0.1 | ACJ |  | 09.12.23 | in progress | Creation |
|  |  |  |  |  |  |

# Table of Contents

Table of Contents 2

1 Environment and Constraint Conditions 3

2 Reinforcement Learning 4

2.1 Multi-Agent Reinforcement Learning (MARL) 5

3 Transfer Learning 6

3.1 Evaluation Metrics for Transfer Learning Approaches 6

3.2 Different Approaches for Transfer Learning 7

3.2.1 Reward Shaping 7

3.2.2 Learning From Demonstrations 9

3.2.3 Policy Transfer 11

3.2.4 Inter-Task Mapping 13

3.2.5 Representation Transfer 14

4 The Chosen Approach: DQN Transfer 18

4.1 Deep Q-learning 19

4.2 Double Q-learning 19

4.3 Experience Replay Buffer 19

4.4 Dueling Network 19

5 Technological Basics 20

5.1 Game Theory 20

5.2 Model-Based Systems Engineering (MBSE) 21

5.3 Dynamic Fault Trees 21

5.4 Sparse Coding 22

5.5 Generalized Policy Improvement 22

6 References 23

# Environment and Constraint Conditions

This project considers the existence of multiple agents. Hence it is to be carried out using the PettingZoo library, in Python. It is the most popular and standard API in Multi-Agent Reinforcement Learning (MARL) that inherits many features of Gym. Gym API, on the other hand, is used in Reinforcement Learning with a single agent.

PettingZoo’s API is, by default, based upon the AEC (Agent Environment Cycle) game models. In this environment, multiple agents have sequential actions, observations, and rewards, one after the other. This sequential execution prevents the race conditions that may occur in the program code while giving more conceptual clarity to the software implementation. Apart from the usual agents, AEC games also have an additional “environment“ agent, which is updated after the action of each of the other agents, according to a specified probability distribution. This dynamic behavior helps in making the system naturally adapt to the actions of the other interacting agents, without the need for any manual intervention; thus preventing the possibility for user errors.

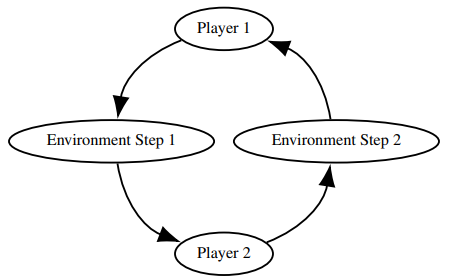


Figure : AEC diagram for a 2-agent system [41]

In addition to this API, PettingZoo also supports a secondary Parallel API, in which all agents have simultaneous actions and observations. Rewards are allocated at the end of each cycle. This is useful for environments with a large number of agents as parallelization reduces the runtime. Conversion between these two APIs can be easily done by using environment transformations called wrappers.

from pettingzoo.classic import rps\_v2

env = rps\_v2.env(render\_mode="human")

env.reset(seed=42)

for agent in env.agent\_iter():

observation, reward, termination, truncation, info = env.last()

if termination or truncation:

action = None

else:

action = env.action\_space(agent).sample() *# this is where you would insert your policy*

env.step(action)

env.close()

Figure : Basic usage of AEC API [38]

The environment should be created such that it could have a variable number of agents and the approach should try to optimize the time and memory consumption by proper utilization of the 2 APIs. Additionally, the Python script must follow object-oriented programming concepts.

Figure : Basic usage of Parallel API [39]

from pettingzoo.butterfly import pistonball\_v6

parallel\_env = pistonball\_v6.parallel\_env(render\_mode="human")

observations, infos = parallel\_env.reset(seed=42)

while parallel\_env.agents:

*# this is where you would insert your policy*

actions = {agent: parallel\_env.action\_space(agent).sample() for agent in parallel\_env.agents}

observations, rewards, terminations, truncations, infos = parallel\_env.step(actions)

parallel\_env.close()

# Reinforcement Learning

In this project, Transfer Learning (TL) is employed in the context of Reinforcement Learning (RL) to overcome the various challenges faced by the latter such as low sample efficiency, sparse or delayed rewards, difficulty in generalizing well to new environments, etc. A typical RL problem involves training an agent to interact with an environment that follows Markov’s Decision Process (MDP). The agent starts with an initial state and moves on to the next one by performing an action, which yields a reward to guide its future actions. The reward represents the quality of the action executed by the agent towards the task solution. An MDP is represented by a tuple, i.e., where:

* is the set of initial states
* represents the state space
* represents the action space
* is the transition probability distribution such that, gives the probability of the state moving on to by taking the action at the state .
* represents a discount factor
* is the reward distribution such that, gives the reward that the agent gets by taking the action to move from state to state .

In RL, the agent follows a policy in this MDP, which specifies the agent’s probability of taking a specific action when it is in a particular state. The following 2 terms are associated with RL to represent the cumulative rewards for the agent being in a particular state and taking certain actions from that state:

* Value function, , estimates the quality of being in a particular state. Here, , is the reward the agent gets for taking the action, , to traverse from the state to .
* Q-function, , estimates the quality of the action taken at a particular state. This is the estimate of the expected future reward of taking the given action in the given state.

The learning objective here is to find the optimal policy to maximize the mean rewards:

Where, is the stationary state-action distribution induced by and is the reward received after k steps [1].

In other words, the optimal policy is learned by iteratively updating the Q-function after each interaction with the environment:

where, is the learning rate, is the discount factor, and is the reward for taking action in the state to move into the new state [2]. Rewards and goals are not the same. Each goal can be differentiated by its reward function. Rewards are the signals that the agent receives from the environment to evaluate its actions, while goals are the desired outcomes that the agent aims to achieve.

## Multi-Agent Reinforcement Learning (MARL)

A multi-agent system consists of an environment which is a physical or a virtual world, and multiple agents with individual goals interacting with it.

If we extend RL to multi-agent systems, then MDP gets replaced by Stochastic Games, which is also represented by a tuple, , where:

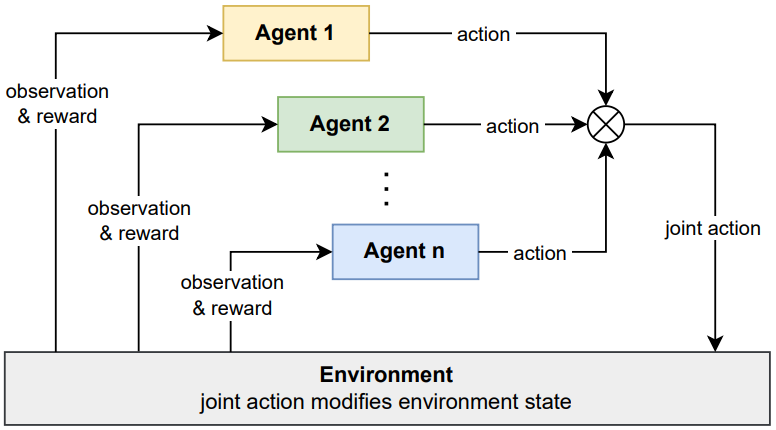
* n is the number of agents

Figure : Schematic of MARL [42]

* represents the state space
* represents the joint action space of all the agents
* is the state transition function (in this case it depends on joint actions instead of individual actions)
* represents a discount factor
* represents the reward function of the agent ‘i’ which is dependent on the state and joint actions.

In Multi-Agent Reinforcement Learning (MARL), we can either consider multiple individual learners each using a single agent RL algorithm, or multiple agents acting jointly over the environment. The generally considered approach is the latter one to avoid the dynamic environment, that happens in the former one. Thus, at a particular state, each agent takes an action and their joint action causes the environment to move to the next state. The agents receive individual rewards as a result of their actions. Sometimes, the agents may not be able to see the full state description of the environment. Instead, they may only receive an observation of the same. Such cases are called the Partially Observable Markov Decision Process (POMDP).

Now, it is not possible to simply apply the action that maximizes the local reward, as we did in the single-agent case, to find the optimal policy. This is because there might be some agents which act adversarial to the agent under consideration and this method becomes ineffective. So, we use adversarial learning algorithms, where the optimal policy is generated by performing the action that maximizes the reward for the agent, supposing that the adversary too performed the best action for itself [3]. The learning objective here is to find the Nash Equilibrium.

Even though these methods produce successful solutions, they are not performance efficient as they require a huge number of agent-environment interactions. Thus, it becomes difficult to apply these algorithms to systems with a large number of agents. This is where Transfer Learning comes into play.

# Transfer Learning

It is well known that learning from scratch is more difficult than learning by reusing the existing knowledge. Transfer Learning (TL) aims at improving the learning performance of the agent in a target domain (t ) by transferring the knowledge it acquired from a different but related source domain (s). The learning objective here is to find the optimal policy for the target domain by making use of exterior information from the source domain and interior information from the target domain.

There is a knowledge space represented by and the learning process maps this knowledge space to a policy . This knowledge space, in general, is composed of samples from the source domain, target domain, and communication with other agents. It is very important to decide on the type, time, and process of taking data from and into this knowledge space. Thus, a challenge in TL is the possibility of reusing unrelated knowledge which negatively affects the learning process. This is known as negative transfer and must be avoided [3].

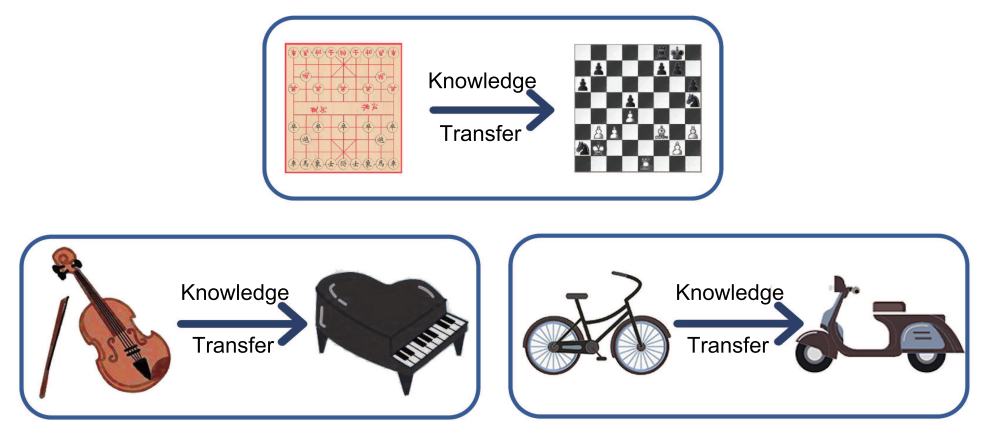


Figure : Some intuitive examples for Transfer Learning [40]

For transfer learning, the traditional MDP is not sufficient as it is not good enough to capture the similarities between the source and target domains [4]. Hence, from this MDP, domain () and task () are defined separately to capture different tasks performed in the same domain, where and .

## Evaluation Metrics for Transfer Learning Approaches

The below-mentioned metrics evaluate how quickly the agent adapts to the target domain and how well it performs there, ultimately.

* Jumpstart performance or the initial performance of the agent.
* Asymptotic performance or the ultimate performance of the agent.
* Total rewards: It is the area under the performance curve.
* Transfer ratio: The ratio of the asymptotic performance of the agent with TL to that without TL.
* Time to threshold: It is the time for the agent to reach a particular performance.
* Performance achieved by the agent after a fixed number of training epochs.

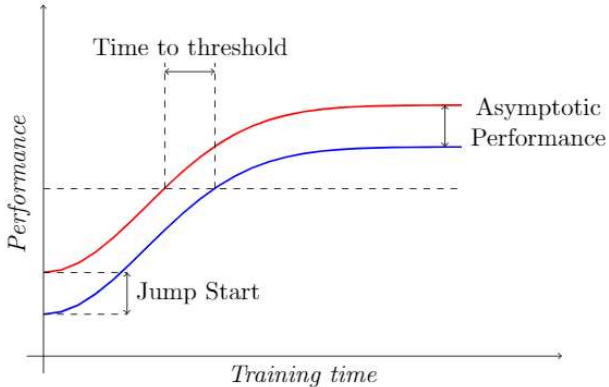


Figure : Performance curve of an agent with and without TL [3]

## Different Approaches for Transfer Learning

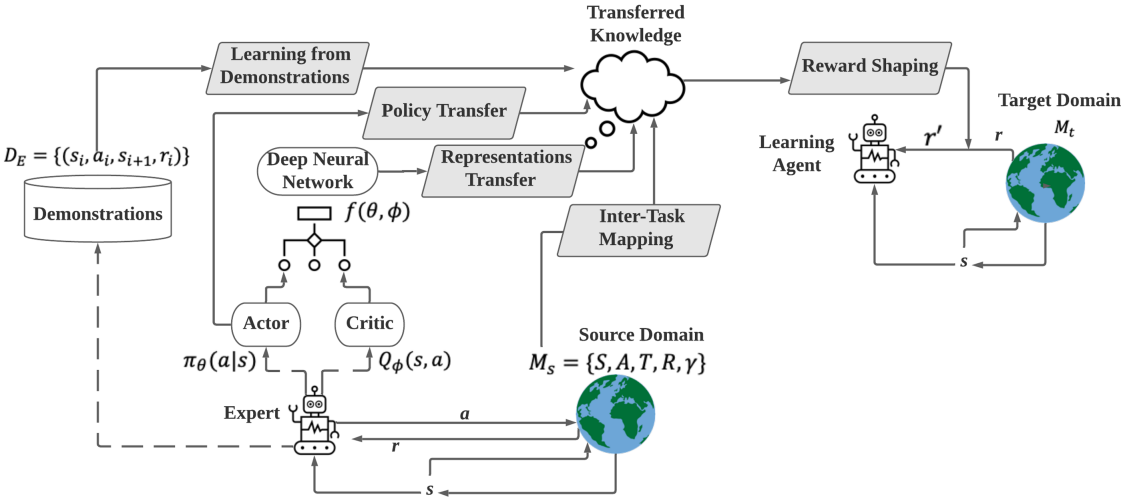


Figure 7: An overview of different TL approaches [1]

### Reward Shaping

This approach utilizes heuristic knowledge to redistribute the reward function of the target domain by using an additional reward-shaping function . The reward-shaping function provides extra rewards apart from the already existing rewards, , in the target domain, to guide the agent to take better actions. Thus the new reward function that the agent uses for policy learning is: .

In other words, we can say that the target domain gets altered from to .

Reward shaping is often used along with other approaches for better convergence of the learning agent, as it reduces the time needed to learn the optimal policy.

There are various definitions for the reward-shaping function:

* Potential Based Reward Shaping (PBRS): Here, is the difference between the potential functions , where, the potential function evaluates the quality of the given state [5].

PBRS, when applied to a MAS, increases the probability of convergence to the Nash equilibrium, without any information about the reward function or the joint action, but can alter the final policy it learns [2].

* Potential-Based state-action Advice (PBA): This is an extension of PBRS to overcome its disadvantage of not taking into consideration the quality of actions. Now, is a function over both state and action space.

Here, is the action taken in the next state () after the agent transitions from the current state (). Thus, is now not only dependent on the current transition but also on the following action , which enables a dependence on the policy the agent is currently learning [6].

* Dynamic Potential Based approach (DPB): The earlier two approaches considered the potential function to be static, and the potential function had to converge first for the agent to converge to the optimal policy. But in DPB, is a function of both state and time, and it was proved that the agent converged to the optimal policy even without the potential function converging. This approach helps agents get the knowledge that has changed while they were learning and can be seen as the dynamic alternative to PBRS [7].

Here, is the time the agent arrived at the state and is the time it reached the next state .

* Dynamic Potential-based value function Advice (DPBA): Similar to DPB, this approach is the dynamic alternative to PBA.

With this framework, we could shape the reward function of the target domain even based on behavioral knowledge, which is not possible with other approaches. Suppose, from the expert knowledge, we have an arbitrary reward , and wish to achieve without losing policy invariance, then, according to PBA, the potential function should satisfy:

The additional state action value function is introduced to accommodate . It is a dynamic function and is learned over time along with the policy. According to Bellman's equation: , which means that learns on the negation of [8].

#### Advantages

* Reward shaping accelerates the learning process by providing incentives to guide the agent toward the desired behavior.
* It allows experts to impart their knowledge into the learning process.
* It helps the agent avoid being stuck in undesirable states by penalizing the agent.

#### Disadvantages

* Reward shaping can cause the agent to prioritize short-term gains at the expense of long-term optimal policy.
* Incorrect design of reward functions can cause unintended behaviors to get rewarded and ultimately lead to undesired outcomes.
* If the expert’s understanding of the problem is different from the actual one, then the heuristics used could be inaccurate and can lead to a bad reward shaping.

### Learning From Demonstrations

Learning from demonstrations (LfD) is an approach where the agent in the target domain is presented with a data set , containing the interactions of an expert with the source domain. The target agent, like in supervised learning, uses this data to learn the policy. In other words, the agent learns to perform a task by imitating the behavior of the expert [9].

There are mainly two ways to acquire demonstrations [10]:

* Direct demonstration, where the learning agent itself performs the demonstration.
* Indirect demonstration, where an external/expert agent performs the demonstration.

The data acquired from demonstrations usually consists of state representation as feature vectors, actions performed by the expert, and rewards for each state-action transition.

LfD approaches are generally classified based on when the demonstrations are used for knowledge transfer:

* Offline approaches: These include offline RL, where the demonstration data set alone is used to train the agent, without any interactions with the environment; and, pretraining of RL components using the demonstration data set before the RL agent starts interacting with the environment.
* Online approaches: In these methods, demonstrations are directly used during the agent’s interaction with the environment to help it explore more efficiently.

Popular online LfD approaches are:

* Direct Policy Iteration with Demonstrations (DPID): This approach combines the samples from the demonstration data set () with the experiences the learning agent gets from the environment (), to estimate the Q-value (), by using Monte Carlo method. The policy is derived from according to:

Further policy improvement is done by comparing with the expert policy , to define a loss function . The policy is adjusted to minimize this loss [11].

* Deep Q-learning from Demonstrations (DQfD): This approach uses the demonstration data to pre-train the learning agent using temporal difference losses and supervised losses. The supervised loss helps the agent to imitate the demonstrator, while the temporal difference loss helps it to develop a reliable value function, that fits the Bellman Equation. These set the foundation for further learning when the agent starts interacting with the environment.

During pre-training, the agent takes small mini-batches from the demonstration and uses them to update its network with four types of losses:

* + - 1. 1-step double Q-learning loss:
      2. n-step double Q-learning loss:

Both of the above Q-learning losses together ensure that the Bellman equation is satisfied.

* + - 1. Supervised large-margin classification loss:

; where is the action taken by the expert in the state , and is the loss function to guide the agent to take action near to .

* + - 1. L2 regularization loss to prevent overfitting on the small mini-batch:

Thus the overall loss is, , where parameters are weight factors.

Once pre-training is done, the agent begins operating in the environment using its learned policy and updates its network by using a combination of both demonstrated behaviors and experiences it gathers by interacting with the environment. For this, the DQfD algorithm maintains two separate replay buffers and samples from each with a certain priority. The same loss function is used for the sampled data too, but supervised loss is not applied for self-generated data [12].

* Generative Adversarial Imitation Learning (GAIL): This method allows the agent to learn the policy directly from the demonstration dataset without interacting with the expert. This is equivalent to the combination of inverse reinforcement learning, to recover the expert’s cost function, and using that to extract the policy by reinforcement learning. For that, it introduces the concept of occupancy measure, , which is the probability distribution of state-action pairs, under the policy .

; where is the probability of landing in state at the time when following the policy .

Based on this formula, the occupancy measure of the current policy (and the expert policy () are found. The divergence between these two is minimized to design a new reward function. However, since is unknown, is determined from the demonstration dataset. The reward function is found using adversarial training, where a discriminator, D learns to distinguish between the state-action pair sampled from the expert policy and from the current policy, using a min-max optimization strategy.

The output of the discriminator is optimized as:

where and are the normalized stationary state-action distribution of and respectively; with and

The discriminator’s output is used as a new reward: , which helps to adjust the agent’s policy to match the expert policy distribution. The agent uses this reward to maximize the probability of the discriminator being mistaken about whether the action came from the expert or the agent. In other words, the learning agent tries to maximize its rewards while simultaneously trying to minimize the discriminator's ability to differentiate between its actions and those of the expert, by generating expert-like actions [13], i.e.

* Self-Adaptive Imitation Learning (SAIL): Imperfect demonstrations are a challenge faced by LfD. SAIL is an off-policy approach that can learn sparsely rewarded tasks by using a smaller number of suboptimal demonstrations. Similar to GAIL, this approach also tries for distribution matching between the expert policy and the current policy, but simultaneously, it also asks the agent to deviate from the previously learned lessons to find better behavior compared to the expert demonstrations. Thus, the learning objective is to minimize the divergence between and , while simultaneously maximizing the divergence between and , where , and are the normalized state-action distribution of , and a mixture of previously learned policies, respectively.

Like the DQfD algorithm, this one also maintains two separate replay buffers, namely the expert replay buffer , sampled from an unknown expert policy and the self-replay buffer to store the state-action pair generated by the current policy . It trains a discriminator to distinguish between the expert demonstrations and the self-generated samples. The output of this is used to generate the reward, which is used by the learning agent.

The learning agent consists of an actor and a critic. The actor tries to increase the reward by choosing suitable actions, while the critic tries to maximize the expected Q-value over the distribution . During the training, if any good quality self-generated trajectory is found, then it is added to the expert replay buffer [14].

#### Advantages

* It can be combined with other approaches.
* The agent learns without interacting with the environment. Hence this method is much safer.
* Faster convergence as there is no need for the agent to do trial and error fashioned exploration of the environment, to find the optimal policy.

#### Disadvantages

* Performance depends upon the quality of the demonstration set.
* Difficulty associated with the creation of the demonstration data set.

### Policy Transfer

Policy transfer is the technique of using expert policies from one or more source domains to generate a policy for the target domain.

It is generally classified into two categories:

* Policy distillation: It involves transferring the expert policies by compressing the knowledge from multiple sources into an ensemble for the target domain [15]. This target policy is learned by minimizing the divergence between the expert policy () and the current policy (). There are two types of policy distillation [16]:
  + Teacher distillation: Here, the divergence is minimized by taking the mean over the trajectories taken from , i.e.
  + Student distillation: Here, the divergence is minimized by taking the mean over the trajectories taken from instead of , i.e.

Here, represents the parameters of the target policy .

From another angle, policy distillation can be classified into the following two approaches:

* + Minimize the cross entropy between and , over the actions. The actor-mimic algorithm is a popular example of this approach. It uses deep reinforcement learning and model compression methods to train an agent using guidance from multiple experts [17].

Given a set of source games with corresponding expert networks , the goal is to train a network using the guidance from these expert networks. This guidance is obtained by transforming the Q-values of each expert using a softmax:

Where,

is the action space of the expert and is the temperature parameter.

The knowledge distillation is done by using the following optimization equation:

Where, is the coefficient of weight decay, and Is the cross entropy measure.

* + Maximise the probability of expert policy meeting the trajectories generated by the learning agent. The distral (distill and transfer learning) algorithm is a popular method under this category. Here, the common behaviors from multiple expert policies are distilled to form a shared policy. This distilled policy is then regularised using KL divergence for the target domain. This is done by using the following objective function:

; where

Here, and are scalar factors for KL divergence.

, is the entropy term that encourages exploration.

, is a reward-shaping term, which encourages the agent to take up actions from the distilled policy [18].

* Policy reuse: Policy reuse is a method in which the expert policies are directly used as a weighted sum to form the target policy. Thus, this method uses different policies, that solve different tasks in a particular domain, to bias the policy learning for a similar task in the same domain. The PRQ Learning algorithm is a popular technique under this category. For this, a library of past policies, is maintained, and the reuse gain (W) of each policy is measured. Reuse gain is the total reward obtained by the agent when that particular policy is used in the target domain.

To determine whether a particular policy in this library is to be reused or not we compute the following probability:

Where is the reuse gain of policy from and is a temperature parameter. is the average reward the agent gets by following its policy . The policy with the maximum probability measure is used [19].

#### Advantages

* Policy distillation helps in compressing the knowledge from the complex source domains to a simpler one.
* Policy reuse helps in reducing the need for large labeled data in the target domain.
* The models generated using distillation need less memory.

#### Disadvantages

* Policy reuse cannot be used if the source and target domains are dissimilar.
* There are chances for negative transfer.
* In the case of distillation, fine details from the source domain are lost due to knowledge compression.
* Policy Distillation is sensitive to hyper parameters, which need to be tuned carefully.

### Inter-Task Mapping

The source task and target task can vary in many ways. Inter-task mapping is an approach where mapping functions are used to find the correspondences between the source domain and the target domain. For this, the earliest technique was to find the mapping functions and for the states and actions in the target domain, respectively, to find their corresponding counterpart in the source domain, based on the similarity of transitions. From these functions, another mapping function, , for the Q-values was derived, which transforms the Q-value from the source domain to the target domain [20].

Where,

and are the state and action, respectively, in the target domain, and

and are the corresponding state and action, respectively, in the source domain.

and . When we use the Q-value reuse method,

Another paper [21] proposed to use a common task subspace, , to learn the relationship between the two domains. Here only an interstate mapping is learned. The dimensionality of is lower than that of the state representations in the target or source domain and is described by the control problem’s definition or by the user. To learn the mapping function , this method takes number of state-successor state patterns from the target domain and projects it into the subspace. Similarly, number of state - successor state patterns from the source domain are also taken and projected. The pairs with the minimum distance in this space are determined and their corresponding states in the source domain and target domain are found.

Further, for each state in the target domain the corresponding states in the source domain are found using . It then selects the action in the source domain according to the expert policy to find the next state in the source domain. For each , is used again to find . The , pair with the least distance is found and the action that corresponds to this minimum value is taken as the best action for .

A similar proposal was made in another paper [22] to learn an inter-task mapping, , of the state-action-state triplets from the source domain to the target domain. For this, sparse coding along with the L1 projection scheme was used. The first step is to map the state-action-state triplets from the source domain into a high-feature space, using sparse coding. The task triplets from the target domain are projected onto this new feature space using L1 regularization techniques. Following this, a similarity measure between these representations is done to determine the most similar source and target domain triplets, from which is approximated.

#### Advantages

* Faster training due to the availability of useful features from the source task as a starting point.
* Inter-task learning acts as a form of regularization and prevents overfitting.
* It requires less amount of labeled data from the target domain.

#### Disadvantages

* It is computationally complex.
* If there is no similarity between the source task and the target task, then a negative transfer can occur.

### Representation Transfer

Representation transfer is a kind of transfer learning in which knowledge is transferred from the source to the target domain in the form of representations learned by deep neural networks. The idea is to separate the input space (which can be the state space, action space, or reward space) into different subspaces that are orthogonal to each other and can be used across different tasks. Thus, it is easier to transfer knowledge between tasks as the knowledge that is relevant to a particular task can be isolated and reused for another task.

The approaches under this topic are classified into two:

* Reuse the representations from the source domain:
  + Progressive Net:

Progressive networks retain a pool of pre-trained models during the training and extract useful characteristics from these for a new task, by learning lateral connections. The knowledge transfer in this manner helps to prevent catastrophic forgetting and improves convergence speed. The network is made up of several columns, each of which is a policy network related to a particular task. Initially, it contains only a single column for that first task. Later when more tasks are added, the network expands to include more columns. The weights of the neurons in the previous columns remain frozen while the representations from them are transferred to the new column to aid knowledge transfer. Here, each column defines a policy taking as input a state from the environment and outputting the probability of action, following that policy [23].

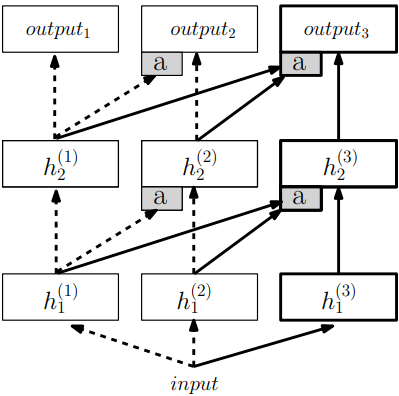


Figure : A progressive network [23]

* + PathNet:

A disadvantage of the Progressive Net is the growth of the network with the number of tasks. This is because it uses a hard-wired design for knowledge transfer. PathNet overcomes this by allowing the relationship between the columns to evolve over time. The knowledge transfer in this network takes place by reusing the optimal pathway in the source task, by fixing its parameters, while allowing the rest of the parameters to change and evolve for the target task. A pathway is a particular route through the neural network that chooses which subset of parameters to use and update in the forward and backward propagation. Each pathway represents a different way of analyzing the input to produce an output prediction. The optimal pathway is selected through an evolutionary process that maximizes the network's performance on a given task. [24]

* + Modular Networks:

The policy network is segmented into task-specific and agent-specific modules in this method. This implies that several agents can do the same task by using a task-specific module. Comparably, many tasks on the same agent may be performed with an agent-specific module [25].

Similarly, another paper [26] proposed to decouple the learning process into separate modules for the state dynamics model and reward function, where, each module is learned separately.

The state dynamics module learns a latent representation of the dynamics of the environment.

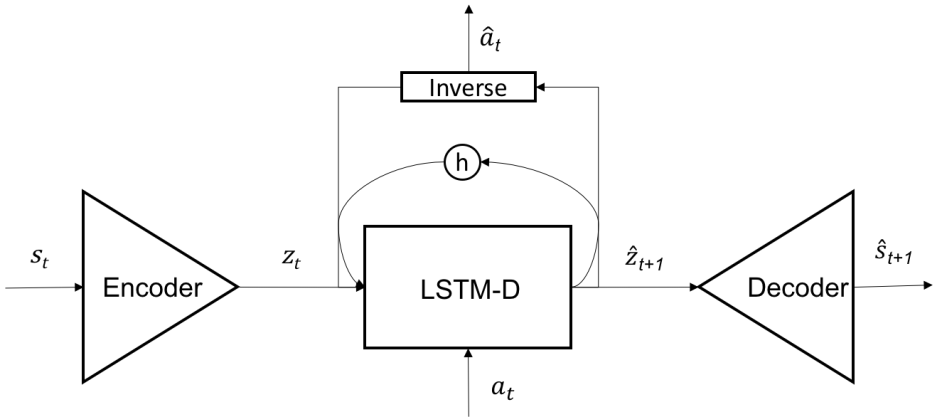


Figure : Dynamics Module [26]

The encoder and decoder learn a mapping between the state space and a representation space . The forward model predicts the transition probability in and the inverse model takes in the current state and the next state in to predict the action to do so.

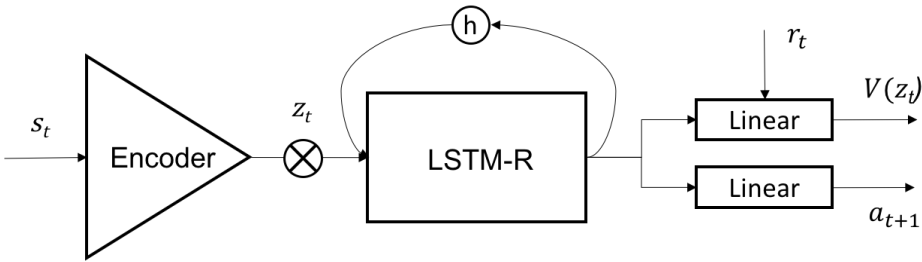


Figure : Reward Module [26]

The reward module learns the value function and the policy in, by using the actor-critic method.

During training, the proposed learning framework is first pre-trained on a set of related tasks to learn a forward dynamics model and an inverse dynamics model. Once the pre-training is complete, the pre-trained model is fine-tuned on a target task with a new reward function. Thus we can reuse the pre-trained models for task representation and dynamics, which are likely to be relevant to the new task, while only learning the reward function from scratch.

* Learn to split the source domain representations into independent sub-feature representations:
  + Successor Representation:

It is an approach where the state features are decoupled from the reward distribution. Successor Representation is a state representation that captures the underlying structure of the environment without being biased toward any specific task or goal. This type of state representation is useful for transfer learning because it can be used across multiple tasks that share similar underlying structures, even if their reward functions are different.

For that, the value function is decomposed into two independent components:

Where, is called the reward mapping function, which maps the states to rewards, and is the successor representation. It represents a state as the occupancy measure of the future states.

is the indicator function.

This method allows knowledge transfer across domains that have only their reward distribution different [27]. Successor representation was originally proposed for discrete state spaces. We get successor features when we extend it to continuous spaces. In successor features, each state is represented by a vector that describes the expected future occurrence of all states under a fixed policy.

Now, the reward associated with the transition may be written as:

Where, is a one-hot coded vector, which is the latent feature of transition and is the task-specific reward mapper containing the weights. From this, we get the Q-function:

is the successor features of and the elements of contain the values of .

If ,represents the Q-value for the expert task in the source domain, then for the agent in the target domain, the corresponding Q-value can be derived as:

Here, we assumed that the reward function is a linear combination of successor features [28]. However, the reward functions do not always have this linear behavior.

The solution is to learn a matrix instead of a vector. This matrix consists of the basis vectors of the latent space. If there are number of individual expert tasks in the source domain out of which are linearly independent tasks, then those tasks form the basis vectors. The latent features of the target task can be developed as a linear combination of these basis vectors along with its reward. Thus the latent features can be represented as:

where the ith element of is which is the reward for the ith task in the source domain.

Similarly, we get the successor features:

where, is the Q=value for the ith task in the source domain.

The learned and are then used to compute the policy for the target task using the Generalised Policy Improvement algorithm [29].

* + Universal Function Approximation (UVFA):

This approach also, permits transfer learning only for tasks that differ solely in their reward functions, much like the Successor Representation (SR). However, unlike SR which focuses on learning a state representation that is independent of the reward function, this method aims to find a function approximator that is generalized for both states and goals. The method involves learning a matrix of latent features, through a matrix factorization process that decomposes the UVFA, into a set of basis functions, and. These functions can be used to represent both states and goals.

For an optimal policy , the corresponding optimal value function for a task is . In this method, is used to approximate the optimal value functions over the state and goal spaces.

,

Here, , where represents the state embedding function and represents the goal embedding function.

The state embedding function is a neural network, that maps a state to a fixed-length feature vector. It captures the relevant information about the state for the given task. By using the same state embedding function across tasks that differ only in goals, the UVFA can generalize to new tasks with different goals. When a new task with an unseen goal is encountered, the UVFA can be initialized with the values learned from a previous task. This is possible because the state embedding function is shared across tasks, and the same state features can be used for both tasks. By initializing the UVFA with the values learned from a previous task, the agent can start with a good estimate of the optimal value function for the new task, which can speed up learning considerably [30].

#### Advantages

* These methods reduce the amount of data required to learn a new task, as it leverages knowledge from previously learned tasks.
* It can improve the generalization performance of the agent as it encourages the model to learn more abstract and task-invariant representations

#### Disadvantages

* It can be challenging when the source and target tasks are significantly different, as the transferred knowledge may not be relevant or may even be harmful to the target task.
* It can introduce bias into the learned model if the transferred knowledge is not properly adapted to the target task.
* It can be computationally expensive, as it requires training multiple models on multiple tasks and selecting the best model for the target task.

# The Chosen Approach: DQN Transfer

Q-learning is a well-known reinforcement learning algorithm in which the Q-value function is estimated using the Bellman equation iteratively till it converges to the optimal value.

Here, the optimal Q-value, is the highest Q value that can be obtained for a given state-action pair, by following a strategy. It helps the agent to choose the best action in a given state, which in turn leads to the highest expected future reward. During each iteration, the agent interacts with the environment by taking action and receiving rewards. The agent then updates its value estimate and uses it to guide the agent's future actions, as the agent seeks to maximize its expected future reward. Q-learning is a model-free, off-policy algorithm as the observations are taken directly from the environment without creating a model for it, and the agent learns about the optimal policy while following a behavior policy that is different from the optimal policy.

## Deep Q-learning

In deep Q-learning, convolutional neural networks are used to estimate for by learning [31]. This Q network is trained by minimizing the loss function , which changes with each iteration. This loss function is optimized using the stochastic gradient descent method, where we minimize the loss function.

Where is the optimal Q-value or target Q-value for iteration :

## Double Q-learning

Conventional Q-learning is affected by maximization bias as the same value is used to select and evaluate it. In other words, the expectation of the maximum is greater than or equal to the maximum of the expectation [32]. To avoid that and also to have a fixed target, double Q-learning is employed, where two different networks are used to find the action in the next state and target Q-value:

Here, are the main network parameters, and are the target network parameters. The target parameters are periodically updated by copying from the main parameters.

## Experience Replay Buffer

Consecutive samples in reinforcement learning often have strong correlations, which can lead to inefficient learning and unwanted feedback loops [33]. To avoid that, an experience replay buffer is used. It is a dataset, that the agent maintains while learning, where is the experience. While training the network, instead of using only the current experience, a random minibatch is sampled from the buffer to train on.

## Dueling Network

In many reinforcement learning problems, the value of each action is not equally important [33]. In some cases, it may be more important to know the value of a state than the value of each action in that state. To take it into account, a dueling network architecture is considered, in which the last layers of the normal convolutional structure are split into separate streams to predict the value function and an advantage function , from which can be computed as:

is the dimension of the vector

represents the parameters of the CNN while and represent the parameters of the two streams of fully connected layers.

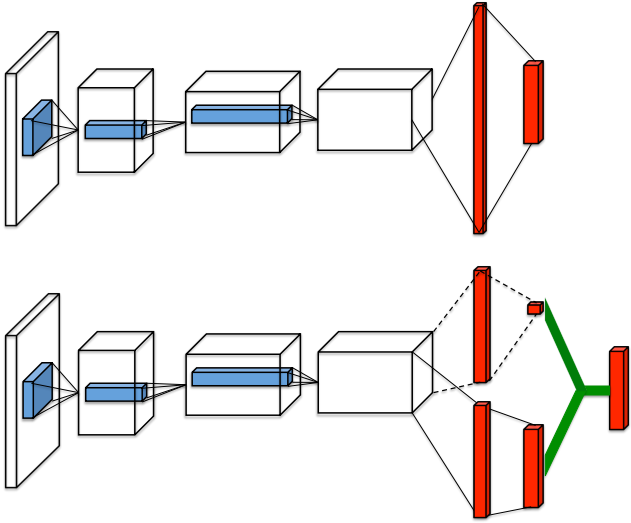


Figure 11: Dueling Network Architecture [33]

estimates how good the state is.

estimates how good it is to perform the action in the state .

estimates the total value of being in the state and performing the action .

The initial layers of the CNN learn the low-level data from the task which can be generalized for the succeding learning task. So, once the learning is completed for an environment, the parameters of the initial layers are copied and used for the initial layers of the DQN for the next environment. These parameters are then fine-tuned to fit the new task. Thus transfer learning happens.

# Technological Basics

## Game Theory

Game theory is the study of interaction and conflict between several players or agents in an environment. It provides a mathematical basis for studying possible approaches that players could employ when engaging in competitive or cooperative play. In a game, multiple players interact with one another, with each player's outcome influenced by the actions of all the other players. The choices made by the players lead to different rewards, which represent the value of the outcome. Each player aims to maximize their rewards by anticipating the actions of the other players. The game models are mainly classified into the following types:

* Non-cooperative Games: Each player acts independently and chooses their strategy to maximize their rewards while taking into account the strategies chosen by the other players. This model is ideal for adversarial situations where agents are competing against each other and aiming for the best outcome for themselves.
* Cooperative Games: Players collaborate and establish coalitions to achieve a common goal.

The non-cooperative game model used for an adversarial multi-agent system can be shown as a strategic form game, which is usually represented by a matrix, where each row represents a strategy for one player, and each column represents a strategy for another player. The reward for each agent depends on the combination of strategies chosen by all the agents. In such a game, the optimal solution is a Nash equilibrium, which is a stable state where no agent can improve their reward by unilaterally changing their strategy, assuming that all other agents keep their strategies fixed. However, in adversarial multi-agent systems, there may be multiple Nash equilibria, and some of them may be undesirable or unstable. In such cases, we may use other solution concepts, such as the minimax solution, to identify the optimal solution. The minimax solution is the strategy that minimizes the maximum possible loss for an agent, assuming that the other agents are playing optimally.

The minimax solution is also used in Zero-Sum games. A zero-sum game is a type of game in game theory where the total rewards of all players is zero. The concept of zero-sum games is used in adversarial multi-agent systems when the interests of the agents are opposed, and one agent's gain is another agent's loss. Here, the agents can use the minimax solution to minimize the maximum loss or maximize the minimum gain [34].

## Model-Based Systems Engineering (MBSE)

MBSE is a structured approach to systems engineering that utilizes modeling to assist with system requirements, design, analysis, verification, and validation activities. The approach starts with the conceptual design phase and continues through the development and subsequent life cycle stages. It is a shift away from traditional document-centric approaches to systems engineering, which rely on textual descriptions of system requirements, designs, and other aspects of the system. Instead, MBSE emphasizes the use of graphical models to represent system elements and their relationships. These models can be used to simulate system behavior, analyze their performance, and evaluate the impact of changes to the system [35].

## Dynamic Fault Trees

Fault Tree Analysis is an important technique to evaluate and mitigate the risks related to complex systems. A fault tree illustrates a graphical representation of the logical relationships between the components of a system and the events that can lead to failure. It is a tree with component failures as leaves and gates to represent the propagation of these failures. Fault tree analysis is done by converting a fault tree into a binary decision diagram (BDD).

Static fault trees contain basic gates AND and OR apart from VOT(k) gates. VOT(k) will fail if k out of the n signals fails. Static fault trees (SFT) are used to model the static behavior of a system, where the components of the system are assumed to be independent and the system is assumed to be in a steady state. This means that they cannot capture the dynamic behavior of a system, such as the effects of time-dependent failures.

Dynamic fault trees are an extension of SFTs that allow for the modeling of sequence dependencies in the system. Unlike static fault trees, DFTs can model dynamic systems where conditions change over time, such as repairs, maintenance, or system reconfiguration. Since DFTs consider failure sequences instead of boolean combinations, we cannot use BDD to represent them. Instead, they are represented by Markov models or Petri nets. Apart from the gates used in SFTs, the following additional gates are used in DFTs:

* Priority-AND gate (PAND): It fails only if its children fail from left to right.
* Priority-OR gate (POR): It fails if the first child fails before any other child does.
* Spare gate (SPARE): This gate manages the usage of shared spare modules, which are groups of redundant parts. The first child of the SPARE gate is the primary and the other inputs are spares. Initially, the spare uses the primary. Whenever the currently used child fails, the SPARE gate attempts to switch to a child that has not yet failed and is not used by another SPARE. If no such child exists, the SPARE fails.
* Sequence enforcer (SEQ): It ensures that its children fail only from left to right, and unlike other common gates it does not propagate failures.
* Functional dependency (FDEP): It forces all the other children, called the “dependent events“, to fail when its first child, called the “trigger“, fails. Like SEQ, this gate also does not have a parent to propagate faults.
* Probabilistic dependency (PDEP): it is an extension of FDEP with a probability. When the trigger of the PDEP fails, the dependent events fail with probability p. Thus, a PDEP with p=1 is equivalent to an FDEP.

Failure propagation in a DFT is similar to that in a SFT. But, for DFT, apart from the usual upward propagation, it also has a propagation through FDEPs [36].

## Sparse Coding

Sparse coding is a method to find basis functions that extract high-level features from an unlabeled data input. Thus, the goal is to represent the input vectors as a linear combination of the basis vectors.

For an input vector , we need to have the basis vectors ,…, and a vector of weights , so that

The number of basis vectors can be more than the dimension of the input vector; i.e. can be over-complete. For obtaining good basis vectors, we assume the reconstruction error, to be normally distributed, and to have the weight vector to be sparse, we define the distribution of each element as . The L1 regularization produces sparse coefficients or weights and hence avoids irrelevant features [37].

## Generalized Policy Improvement

Bellman’s policy improvement theorem states that “acting greedily with respect to the value function of a policy, gives rise to another policy whose performance is no worse than the former’s”. Generalized policy improvement extends this theorem such that now the new policy is derived from the value functions of a set of policies. This is done by acting greedily with respect to the maximum of the value functions available [28].

If are the policies available with the Q-value functions respectively, then the new policy can be found as:

With the Q-value function .

# References

|  |  |
| --- | --- |
| [1] | Z. Zhu, K. Lin, A. K. Jain und J. Zhou, „Transfer Learning in Deep Reinforcement Learning: A Survey,“ *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE, VOL. 45,* p. 13345, 2023. |
| [2] | S. Devlin und D. Kudenko, „Theoretical Considerations of Potential-Based Reward Shaping for Multi-Agent Systems,“ in *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems*, 2011. |
| [3] | F. Leno Da Silva and A. Helena Reali Costa, “A Survey on Transfer Learning for Multiagent Reinforcement Learning Systems,” *Journal of Artificial Intelligence Research 64,* pp. 645-703, 2019. |
| [4] | A. A. Sherstov und P. Stone, „Improving Action Selection in MDP’s via Knowledge Transfer,“ *Proceedings of the Twentieth National Conference on Artificial Intelligence,* 2005. |
| [5] | A. Y. Ng, D. Harada and S. J. Russell, “Policy invariance under reward transformations: Theory and application to reward shaping,” in *International Conference on Machine Learning*, 1999. |
| [6] | E. Wiewiora, G. Cottrell und C. Elkan, „Principled Methods for Advising Reinforcement Learning Agents,“ in *International Conference of Machine Learning*, 2003. |
| [7] | S. Devlin und D. Kudenko, „Dynamic Potential-Based Reward Shaping,“ in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, 2012. |
| [8] | A. Harutyunyan, S. Devlin, P. Vrancx and A. Nowe, “Expressing Arbitrary Reward Functions as Potential-Based Advice,” in *Association for the Advancement of Artificial Intelligence*, 2015. |
| [9] | A. Correia and L. A. Alexandre, “A Survey of Demonstration Learning,” 2023. [Online]. Available: https://arxiv.org/abs/2303.11191. |
| [10] | B. Fang, S. Jia, D. Guo, M. Xu, S. Wen und F. Sun, „Survey of imitation learning for robotic manipulation,“ *International Journal of Intelligent Robotics and Applications,* pp. 362-369, 2019. |
| [11] | J. Chemali und A. Lazaric, „Direct Policy Iteration with Demonstrations,“ *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence,* p. 3380–3386, 2015. |
| [12] | T. Hester, M. Vecerik, O. Pietquin, M. Lanctot, T. Schaul, B. Pilot, D. Horgan, J. Quan, A. Sendonaris, I. Osband, G. Dulac-Arnold, J. Agapiou, J. Z. Leibo and A. Gruslys, “Deep Q-learning from demonstrations,” *Proc. Conf. Association for the Advancement of Artificial,* p. 3223–3230, 2018. |
| [13] | J. Ho and S. Ermon, "Generative Adversarial Imitation Learning," *Proc. Int. Conf. Neural Inf. Process. Syst.,* p. 4572–4580, 2016. |
| [14] | Z. Zhu, K. Lin, B. Dai and J. Zhou, “Learning sparse rewarded tasks from sub-optimal demonstrations,” arXiv: 2004.00530, 2020. |
| [15] | A. A. Rusu, . S. Gomez Colmenarejo, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, R. Hadsell and C. Gulcehre, “POLICY DISTILLATION,” arXiv:1511.06295, 2015. |
| [16] | W. M. Czarnecki, R. Pascanu, S. Osindero, S. Jayakumar, G. Swirszcz und M. Jaderberg , „Distilling Policy Distillation,“ in *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics*, 2019. |
| [17] | E. Parisotto, J. Lei Ba und R. Salakhutdinov, „Actor-Mimic: Deep Multitask and Transfer Reinforcement Learning,“ 2015. |
| [18] | Y. Whye Teh, V. Bapst, W. Marian Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess und R. Pascanu, „Distral: Robust multitask reinforcement learning,“ *Proc. Int. Conf. Neural Inf. Process. Syst.,* p. 4499–4509, 2017. |
| [19] | F. Fernandez und M. Veloso, „Probabilistic Policy Reuse in a Reinforcement Learning Agent,“ *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems,* p. 720–727, 2006. |
| [20] | M. E. Taylor , P. Stone und Y. Liu, „Transfer Learning via Inter-Task Mappings for Temporal Difference Learning,“ *Journal of Machine Learning Research 8,* pp. 2125-2167, 2007. |
| [21] | H. B. Ammar and M. E. Taylor, “Common Subspace Transfer for Reinforcement Learning Tasks,” *Proc. 11th Int. Conf. Adaptive Learn. Agents,* p. 21–36, 2012. |
| [22] | H. B. Ammar, K. Tuyls, M. E. Taylor, K. Driessens and G. Weiss, “Reinforcement Learning Transfer via Sparse Coding,” in *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, Valencia, Spain, 2012. |
| [23] | A. A. Rusu, N. C. Rabinowitz, G. Desjardins , H. Soyer, J. Kirkpatrick, K. Kavukcuoglu, R. Pascanu and R. Hadsell, “Progressive Neural Networks,” arXiv:1606.04671, 2016. |
| [24] | C. Fernando, D. Banarse, C. Blundell, Y. Zwols, D. Ha, A. A. Rusu, A. Pritzel and D. Wierstra, “PathNet: Evolution Channels Gradient Descent in Super Neural Networks,” arXiv:1701.08734, 2017. |
| [25] | C. Devin, A. Gupta, T. Darrell, P. Abbeel and S. Levine, “Learning Modular Neural Network Policies for Multi-Task and Multi-Robot Transfer,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2017. |
| [26] | A. Zhang, H. Satija and J. Pineau, “Decoupling Dynamics and Reward for Transfer Learning,” arXiv:1804.10689, 2018. |
| [27] | P. Dayan, "Improving generalization for temporal difference learning: The successor representation," *Neural Computation, vol. 5, no. 4,* pp. 613-624, 1993. |
| [28] | A. Barreto, W. Dabney, R. Munos, J. J. Hunt , T. Schaul, H. van Hasselt und D. Silver, „Successor Features for Transfer in Reinforcement Learning,“ *31st Conference on Neural Information Processing Systems,* 2017. |
| [29] | A. Barreto, D. Borsa, J. Quan, T. Schau, D. Silver, M. Hessel, D. Mankowitz, A. Zˇ´ıdek und R. Munos, „Transfer in deep reinforcement learning using successor features and generalised policy improvement,“ *Proceedings of the 35 th International Conference on Machine,* p. 501–510, 2018. |
| [30] | T. Schaul, D. Horgan, K. Gregor and D. Silver, “Universal Value Function Approximators,” *Proceedings of the 32 nd International Conference on Machine Learning,* p. 1312–1320, 2015. |
| [31] | V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," CoRR, abs/1312.5602, 2013. [Online]. Available: https://arxiv.org/abs/1312.5602. |
| [32] | G. He, D. Yang and K. Shen, "Improving DQN Training Routines with Transfer Learning," 2019. [Online]. Available: https://georgehe.me/dqn\_transfer.pdf. |
| [33] | Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot und N. de Freitas, „Dueling Network Architectures for Deep Reinforcement Learning,“ 2015. [Online]. Available: https://arxiv.org/abs/1511.06581. |
| [34] | J. Burguillo, Game Theory, 10.1007/978-3-319-69898-4\_7., 2018. |
| [35] | N. Shevchenko, “An Introduction to Model-Based Systems Engineering (MBSE),” Carnegie Mellon University, Software Engineering Institute's Insights (blog), 21 Dec 2020. [Online]. Available: https://insights.sei.cmu.edu/blog/introduction-model-based-systems-engineering-mbse/. [Accessed 30 Nov 2023]. |
| [36] | S. Junges, D. Guck, . J.-P. Katoen and M. Stoelinga, “Uncovering Dynamic Fault Trees,” in *46th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Toulouse, France*, 2016. |
| [37] | B. Schölkopf, J. Platt und T. Hofmann, „Efficient sparse coding algorithms,“ in *Advances in Neural Information Processing Systems 19: Proceedings of the 2006 Conference*, 2007. |
| [38] | "AEC API," [Online]. Available: https://pettingzoo.farama.org/api/aec/. |
| [39] | "Parallel API," [Online]. Available: https://pettingzoo.farama.org/api/parallel/. |
| [40] | F. Z. Z. Qi, K. Duan, D. Xi, Y. Zhu, . H. Zhu, H. Xiong und Q. He, „A Comprehensive Survey on Transfer Learning,“ Cornell University, 2020. |
| [41] | J. K. Terry, B. Black, N. Grammel, M. Jayakumar, A. Hari, R. Sullivan, L. Santos, R. Perez, C. Horsch, C. Dieffendahl, N. L. Williams, Y. Lokesh and P. Ravi, "PettingZoo: Gym for Multi-Agent Reinforcement Learning," Cornell University, 2020. |
| [42] | S. V. Albrecht, F. Christianos and L. Schäfer, Multi-Agent Reinforcement Learning: Foundations and Modern Approaches, London, England: The MIT Press, Cambridge, Massachusetts. |