Sympathy-based Reinforcement Learning Agents

Manisha Senadeera
A^2I^2, Deakin University
Geelong, Australia
manisha.senadeera@deakin.edu.au

Thommen George Karimpanal
A^2I^2, Deakin University
Geelong, Australia
thommen.karimpanalgeorge@deakin.edu.au

Sunil Gupta
A^2I^2, Deakin University
Geelong, Australia
sunil.gupta@deakin.edu.au

Santu Rana
A^2I^2, Deakin University
Geelong, Australia
santu.rana@deakin.edu.au

ABSTRACT

As artificial agents become increasingly prevalent in our daily lives, it becomes imperative to equip them with an awareness of societal norms; specifically, the ability to account for and be considerate towards others they may cohabit with. In this work, we explore the ability for an agent trained through reinforcement learning to exhibit sympathetic behaviours towards another (independent) agent in the environment. We propose to achieve such behaviours by first inferring the reward function of the independent agent, through inverse reinforcement learning, and subsequently learning a policy based on a sympathetic reward function - a convex combination of the inferred rewards and the agent's own rewards. The corresponding weighting is determined by a sympathy function which is computed based on the estimated return of the agent's current action relative to that of all possible actions it could have taken. We evaluate our approach on adversarial as well as assistive environment settings, and demonstrate the ability of our sympathetic agent to perform well at its own goal, while simultaneously giving due consideration to another agent in its environment. We also empirically examine and report the sensitivity of our agent's performance to the hyperparameters introduced in our proposed framework.

KEYWORDS

Reinforcement Learning; Sympathy; Empathy; Inverse Reinforcement Learning

ACM Reference Format:
Manisha Senadeera, Thommen George Karimpanal, Sunil Gupta, and Santu Rana. 2022. Sympathy-based Reinforcement Learning Agents. In *Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), Online, May 9–13, 2022*, IFAAMAS, 9 pages.

1 INTRODUCTION

Emotions such as empathy and sympathy compel us to feel concern towards others and consider the consequences of our actions towards them. With the prevalence of artificial intelligence [16], particularly in the domain of assistive technologies (robotic or digital assistants [6]), there is a need to incorporate agents with similar considerations towards others they may interact with [2, 10]. In contrast, most interactive agents are primarily trained with a focus on reward maximisation, with little to no awareness of the potential implications of their actions to others in the environment. This continues to be an open problem, and we seek to address it through a sympathy-based framework proposed in this work.

Our framework presents an approach to design agents that behave sympathetically towards other agents (which we refer to as *independent agents*), while simultaneously completing its own tasks. These sympathetic behaviours are achieved by modelling [7] the goals and aversions of another agent, and taking them into account while learning a policy.

We consider environments that consist of a learning agent (to be trained) and an independent agent who behaves according to its own fixed policy (similar to a human who has their own behaviours and values). In learning a sympathetic policy, we assume there exists at least one suboptimal (with respect to reward maximisation) policy that can complete the agent's own task whilst helping or avoiding harm to the other. By observing transitions of the independent agent, we estimate its value function and associated reward function through an inverse reinforcement learning (IRL)[13] process. We subsequently train the learning agent on a shaped sympathetic reward function, constructed as a convex weighted sum of the inferred rewards and the learning agent's own rewards from the environment. Our key contribution is a sympathy function which dynamically determines the convex weighting, which can be interpreted as the degree of selfishness, the learning agent should exhibit based on the situation. The dynamic and state-action-dependent computation of the sympathy function allows the learning agent to behave selfishly in some situations (e.g. protecting itself from harm) and sympathetically in others (e.g. offering assistance). Our proposed framework differs from previous works, which use a constant degree of selfishness throughout training [1, 3, 14]. Rather than simply considering the immediate implications (rewards) of an action, our sympathy function accounts for the long term benefits (return) of an action to both agents, relative to all future states (as computed using a state prediction model) that could have resulted from alternative actions that the learning agent could have taken.

We demonstrate the performance of our proposed framework on two environmental settings. The first is an adversarial setting involving agents that can potentially harm each other. In the next setting (assistive setting), although the agents are incapable of harming each other, the environment is set up such that the independent agent would benefit from assistance from the learning agent. We

Proc. of the 21st International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2022), P. Faliszewski, V. Mascardi, C. Pelachaud, M.E. Taylor (eds.), May 9–13, 2022, Online.

apply our framework in both these settings, and show that it induces harm-avoiding and assistance-providing behaviours in the learning agent without significantly affecting its performance on its own tasks.

The contributions of this work are:

- A novel framework to train an agent to behave sympathetically towards another whilst working towards its own goals.
- A model-based sympathy function that discerns the degree of selfishness to exhibit based on the situation.
- Empirical results demonstrating sympathetic behaviour in both an adversarial and an assistive setting.

2 RELATED LITERATURE

There has been a growing interest in the design of agents that exhibit considerate behaviour. Work by Franzmeyer et al. [5] designed agents who are motivated to behave altruistically. The authors considered scenarios involving a lead agent and an altruistic agent, which was trained to behave in a manner that maximised the lead agent's future choices (e.g., opening a door to increase the lead agent's accessible states). While our work also deals with learning behaviours to account for the presence of another agent, it differs considerably, as our learning agent is not altruistic - it has its own goals, which it pursues while learning to behave sympathetically towards the other agent. Inspired by the golden rule "Do unto others as you would have them do unto you", Bussmann et al. [3] design an Empathetic DQN that avoids negative side effects. In order to empathise with the independent agent, the authors impose the value function of the learning agent on the independent agent, examining how the learning agent would 'feel' if it was in the position of the independent agent. A convex weighted sum of the value functions of both itself and the other agent (through the imposed value function) is used to construct an empathetic Q-value function. A key limitation is the assumption that the other agent has the same value function as the learning agent (implying the independent agent has the same goals as the learning agent). In contrast, the framework we propose in the present work is more general, as it is free of such assumptions, and we employ IRL methods to infer the value function of the independent agent. Due to this fact, our framework is designed to handle a broader range of environments where the goals of the agents differ (and at times conflict with each other). Other papers have taken the approach of involving humans in the process of training considerate agents. Noothigattu et al. [14] trains an agent to act ethically via a contextual multi-arm bandit approach. In the work the agent must select between two pre-trained policies - a reward maximisation policy where the agent behaves completely selfishly in the game, and a constraint policy. The constraint policy is constructed through human demonstrations of playing the game in a "socially acceptable behaviour", the rewards of which are extracted through IRL. A limitation of this work is the need to involve a human to create the constraint policy, and new human demonstrations being required for each new game. Additionally the work requires both policies to be trained beforehand, with the algorithm focused only on learning which of the policies to employ based on the situation. Alamdari et al. [1] propose methods for training an agent to behave considerately of another in both a sequential setting and a simultaneous setting. In the sequential setting the learning agent completes a task alone and learns to leave the environment in a layout that is beneficial to another (independent) agent who enters after it. In the simultaneous setting, a learning agent coexists within the environment with multiple independent agents, with all agents having their own goals. The work investigates the ability of learning agents to leave behind tools (after using them) in locations that are beneficial to the independent agent's tasks. To do this the authors construct a reward function that is a weighted composite of the learning agents rewards and the rewards of the independent agents. Knowledge of a set of possible reward functions and their associated probability distributions to describe the rewards of each of the independent agents is assumed. The weighting must be specified by the user beforehand. Our work is most similar to the simultaneous agent environment algorithm of Alamdari et al. [1] but differs in the fact that instead of assuming foreknowledge of possible reward distributions, we employ IRL to infer the reward function based on the independent agent's behaviour. Additionally the authors assume a constant and pre-specified weighting for each agent's reward function. Our work obviates the need for such user specifications, and instead, learns the situation-dependent weighting online via the sympathy function.

2.1 Problem Formulation

We formulate our problem as a Markov Decision Process (MDP) [15] $< \mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma >$ where $\mathcal{S}$ is the state space, $\mathcal{A}$ is a finite set of actions, $\mathcal{T} : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{S}$ is the transition function which governs the probability of reaching the next state having taken an action in the present state, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function which defines the reward an agent receives for taking an action in the present state, and $\gamma \in (0, 1]$ is the discount factor. We assume the reward function for transitioning from state s to s' can be expressed as a linear combination of binary features $f_{ss}(s, s')$ with pre-specified weights for each feature [12].

In reinforcement learning (RL) [17], the goal of the agent is to learn the optimal policy $\pi^*(s)$ by learning a Q-function $Q^*_{selfish}$ that maps states to actions in order to maximise the expected long term rewards. However, in this work, we aim to learn a sympathetic policy π^*_{symp} by learning a corresponding Q-function Q^*_{symp} that considers not only the agent's own reward function (the weights of which are known), but also the inferred rewards of an independent agent. The features of the independent agent's reward function are assumed to be the same as that of the learning agent. However, the feature weights are inferred through IRL, with the assumption that the independent agent acts per an arbitrary fixed policy. Specifically, we use the Cascaded Supervised Learning Approach [8], which is well suited to an online setting whilst also producing a value function corresponding to the agent's behaviour in the process.

In our work the environment is comprised of two agents, a learning agent who works towards its own goal represented by a reward function, and an independent agent who acts as per a pre-specified fixed policy. The objective is to train the learning agent to complete its task whilst acting sympathetically towards the other (independent) agent. Note that our approach is not a method for multiagent RL as only the learning agent is trained. The state and actions of both agents are observable to the learning agent. However, the

learning agent can only observe its own rewards (and not the independent agent's rewards) while interacting with the environment. We assume the rewards of the learning agent are specified in a comparable range to the inferred rewards (through IRL) of the independent agent to ensure equitable comparison between the agents for the same event. The agents take actions sequentially, meaning when one is taking an action, the other is stationary.

Our proposed framework is illustrated in Figure 1.

2.2 Framework

Our proposed framework consists of four main functions being learned concurrently. The first of these is the reward function of the independent agent inferred via IRL, as described below.

2.2.1 Inferring the reward function of the independent agent: To infer the independent agent's reward function, we use the approach of Cascaded Supervised Learning [8] which first learns a proxy Q-function $\hat{Q}_{indep}(s,a)$ corresponding to the independent agent's policy, using observations of the independent agent's state, action, next-state transitions via supervised learning (SL). The second stage applies a rearrangement of the Bellman equation to infer the reward feature weights, expressed by the vector $\hat{\mathbf{r}}_{indep} \in \mathbb{R}^D$. Algorithm 2 and 5 detail this process. As per Noothigattu et al. [14] the inferred reward is scaled to have an $\ell 1$ norm equal to that of the learning agent's reward weights $\mathbf{r} \in \mathbb{R}^D$ to allow them to exist on a comparable scale as shown in Equation 1. Scaling by the $\ell 1$ norm was chosen for simplicity, however more sophisticated methods of scaling the reward [11] can be utilised.

2.2.2 Value function learning: For the learning agent, two value functions are trained. The first of these represents the value function from behaving selfishly (focusing on only maximising its own rewards as dictated by $\mathbf{r}$, without consideration to the independent agent). This value function $Q_{selfish}$ is trained using the rewards R returned to the agent through its interaction with the environment. Due to the sequential nature of the environment, these rewards are derived from features activated between s_t and s_{t+2}, with the transition from s_t to s_{t+1} resulting from the learning agent's action, and the transition from s_{t+1} to s_{t+2} from the independent agent's action. The second value function Q_{symp} is a sympathetic value function trained on a sympathetic reward R_{symp}, constructed through a convex combination of the inferred independent agent reward $\hat{R}_{indep_t} = \hat{\mathbf{r}}_{indep} \cdot f_{ss}(s_t, s_{t+2}) \in \mathbb{R}$ and the agent's own reward $R_t = \mathbf{r} \cdot f_{ss}(s_t, s_{t+2}) \in \mathbb{R}$ as shown in Equation (2). The convex weighting is determined by a sympathy function $\beta(s_t, a_t)$ which uses the transition from state s_t to the state immediately after the learning agent has acted a_t. Details of the sympathy function will be described in Section 2.2.3.

$$L = \frac{\|\mathbf{r}\|_1}{\left\|\hat{\mathbf{r}}_{indep}\right\|_1} \tag{1}$$

$$\beta_t = \beta(s_t, a_t)$$

$$R_{symp_t} = \beta_t R_t + (1-\beta_t) L \hat{R}_{indep_t} \tag{2}$$

For clarity, $Q_{selfish}$ and Q_{symp} are trained simultaneously using the same state and action input but differing rewards. The learning agent selects actions based on its sympathetic value function Q_{symp}. Algorithm 3 details the training of $Q_{selfish}$ and Q_{symp}.

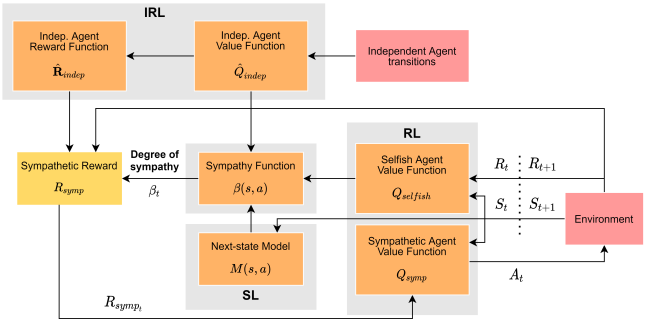


Figure 1: Proposed Sympathy Framework. IRL is used to infer the value function ($\hat{Q}_{indep}$) and associated rewards ($\hat{R}_{indep}$) of the independent agent. For the learning agent, a selfish value function ($Q_{selfish}$) is trained on rewards observed from the environment (R) and a sympathetic value function (Q_{symp}) is trained on sympathetic rewards (R_{symp}). R_{symp} is a convex weighted sum of R and $\hat{R}_{indep}$. The weighting is determined by the sympathy function $\beta(s,a)$ which outputs the degree of selfishness β. $\beta(s,a)$ takes as inputs the value functions of the independent agent and selfish learning agent, and utilised a state prediction model $M(s,a)$.

2.2.3 Computing the sympathy function. The convex weighting in Equation (2) is determined by the sympathy function which outputs β_t which defines the degree of selfishness the agent should exhibit for the current state and action. When the learning agent is completely selfish, $\beta_t = 1$, it takes only its own rewards into consideration, while completely sympathetic behaviours, $\beta_t = 0$, take only the independent agent's rewards into consideration (as per Equation (2)).

Although β_t can be varied to achieve a spectrum of sympathy levels, in order to set its value dynamically, we consider the relative difference in the value (expected returns) of a state-action pair for the agent and the independent agent. The intuition for setting β_t based on the relative difference in values is to ensure that the agent avoids being overly sympathetic in scenarios where a given state-action pair is simultaneously highly beneficial for the independent agent and very harmful for the agent (such as being eaten by a predator). In this work, we model this relative difference in values using Equation (3) and (4).

$$\beta(s_t, a_t) \quad = \quad sigmoid(c \cdot \rho_t) \tag{3}$$

$$\rho_t = \frac{\max_{a' \in \mathcal{A}} \hat{Q}_{indep}(s_{t+1}, a') - \max_{a' \in \mathcal{A}} Q_{selfish}(s_{t+1}, a')}{\max_{\hat{s}_{t+1} \in \hat{S}_{t+1}} \left| \max_{a' \in \mathcal{A}} \hat{Q}_{indep}(\hat{s}_{t+1}, a') - \max_{a' \in \mathcal{A}} Q_{selfish}(\hat{s}_{t+1}, a') \right|} \tag{4}$$

where $c \geq 0$ is a user specified hyperparameter. The numerator in Equation (4) computes the difference in the optimistic estimates of the returns from the next state s_{t+1} (resulting from the action taken by the learning agent) for both agents.

Algorithm 1 Sympathy Framework

Initialise:
Sympathetic value function Q_{symp} with weights θ_{symp}
Sympathetic target value function $\hat{Q}_{symp}$ with weights $\hat{\theta}_{symp} = \theta_{symp}$
Selfish value function $Q_{selfish}$ with weights $\theta_{selfish}$
Selfish target value function $\hat{Q}_{selfish}$ with weights $\hat{\theta}_{selfish} = \theta_{selfish}$
Independent agent value function $\hat{Q}_{indep}$ with weights θ_{indep}
Replay memory D_{symp}, D_{indep} and D^{U}_{indep} to capacity N
Next state model M with weights θ_M
Indep Agent rewards weights $\hat{\mathbf{r}}_{indep}$
f_{ss} - function to return reward features activated between states
C number of episodes between updates of $\hat{\mathbf{r}}_{indep}$, $\hat{Q}_{selfish}$ and $\hat{Q}_{symp}$
T_s = samples before training begins
N = number of training episodes
T = termination
Parameter: $c \geq 0$, $\epsilon \in (0, 1)$

for $episodes = 1, 2, ..., N$ **do**
  obtain initial full game state s_1
  for $t = 1, 3, ..., T$ **do**
    if random probability $< \epsilon$ **then**
      select random action a^{symp}_t
    else
      select $a^{symp}_t = \underset{a}{\text{argmax}} Q_{symp}(s_t, a; \theta_{symp})$
    end if
    Execute action a^{symp}_t
    Observe reward R_t
    Observe state s_{t+1}
    if s_{t-1} **exists then**
      Store transitions $(s_{t-1}, a^{indep}_{t-1}, s_{t+1})$ in D_{indep}
      Store transitions $(s_{t-1}, a^{indep}_{t-1}, s_{t+1})$ in D^{U}_{indep} if unseen
    end if
    Observe a^{indep}_{t+1} given s_{t+1}
    Observe reward R_{t+1}
    Observe state s_{t+2}
    $R_{feat} = f_{ss}(s_t, s_{t+2})$, $R = R_t + R_{t+1}$
    Store $(s_t, a^{symp}_t, R, R_{feat}, s_{t+1}, s_{t+2})$ in D_{symp}
    Update $\hat{Q}_{indep}$ as per Algorithm 2
    Update Q_{symp} and $Q_{selfish}$ as per Algorithm 3
    Update M as per Algorithm 4

    Every C steps:
    Update $\hat{Q}_{symp} = Q_{symp}$ and $\hat{Q}_{selfish} = Q_{selfish}$
    Update $\hat{\mathbf{r}}_{indep}$ as per Algorithm 5
  end for
end for

Algorithm 2 Independent agent Value Function Update

Inputs: D_{indep}, $\hat{Q}_{indep}$

Sample random batch of transitions from D_{indep}
for $j = 1, 2, ..., batch$ **do**
  Set y_j = one hot(a^{indep}_j)
  Perform gradient descent with respect to θ_{indep} on
  $\pi(a^{indep}_j | s_j) = \frac{e^{Q_{indep}(s_j, a^{indep}_j; \theta_{indep})}}{\sum_a e^{Q_{indep}(s_j, a; \theta_{indep})}}$
  $\frac{1}{n} \sum \left(y_j - \pi(a^{indep}_j | s_j) \right)$
end for

Algorithm 3 Learning Agent Value Function Update

Inputs: D_{symp}, Q_{symp}, Q_{symp}, $\hat{Q}_{indep}$, $\hat{\mathbf{r}}_{indep}$

Sample random batch of transitions from D_{symp}
for $j = 1, 2, ..., batch$ **do**
  Determine selfishness β_j from Equation (3)
  Determine R_{symp_j} from Equation (2), where $\hat{R}_{indep} = \hat{\mathbf{r}}_{indep} \cdot R_{feat_j}$
  Set $y_{symp_j} = R_{symp_j} + \gamma \max_{a'} \hat{Q}_{symp}(s_{j+1}, a'; \hat{\theta}_{symp}))$
  Set $y_j = R_j + \gamma \max_{a'} \hat{Q}_{selfish}(s_{j+1}, a'; \hat{\theta}_{selfish}))$
  Perform gradient descent with respect to θ_{symp} on
  $(y_{symp_j} - Q_{symp}(s_j, a^{symp}_j; \theta_{symp}))^2$
  Perform gradient descent with respect to $\theta_{selfish}$ on
  $(y_j - Q_{selfish}(s_j, a^{symp}_j; \theta_{selfish}))^2$
end for

Algorithm 4 Updating state prediction model

Inputs: D_{symp}, M

Sample random batch of transitions from D_{symp}
for $j = 1, 2, ..., batch$ **do**
  $y_j = s_{j+1}$
  Perform gradient descent with respect to θ_M on
  $\frac{1}{n} \sum \left(y_j - M(s_j, a^{symp}_j; \theta_M) \right)^2$
end for

Algorithm 5 Updating Independent Agent Rewards

Inputs: D^{U}_{indep}, $\hat{Q}_{indep}$, $\hat{\mathbf{r}}_{indep}$

Sample random batch of transitions from D^{U}_{indep}
for $j = 1, 2, ..., batch$ **do**
  $R_{feat_j} = f_{ss}(s_j, s_{j+2})$
  $y_j = \hat{Q}_{indep}(s_j, a^{indep}_j) - \gamma \max_{a'} \hat{Q}_{indep}(s_{j+2}, a')$
  Perform gradient descent with respect to $\hat{\mathbf{r}}_{indep}$ on
  $\left(y_j - \hat{\mathbf{r}}_{indep} \cdot R_{feat_j} \right)^2$
end for

The denominator of Equation (4) considers the possible next states $\hat{s}_{t+1}$ that could have resulted as a consequence of all possible actions the learning agent could have taken from state s_t. The next possible states $\hat{s}_{t+1}$ are predicted using a state prediction model $M(s_t, a)$, which is trained in parallel, more details discussed in 2.2.4. In addition, the denominator plays a normalising role in scaling

ρ to the range $[-1, 1]$ (assuming a well trained $M(s_t, a)$). $\beta(s_t, a_t)$ conveys a measure of the degree to which the action that was actually taken by the learning agent compares to all other actions it could have taken from the state s_t. Intuitively, Equation (4) captures the "goodness" of an action for the independent agent compared to the learning agent. If ρ is a large positive number (indicating that the next state for the independent agent is much better than for the learning agent), then β_t tends towards 1 (encouraging the learning agent to be more selfish). Conversely, if ρ is a large negative (indicating a situation that is significantly worse for the independent agent than for the learning agent) β_t is closer to 0 (encouraging the learning agent to behave sympathetically). If the outcomes are is equivalent for both agents, $\beta_t = 0.5$ and the rewards of both agents are equally weighted. This normalised "goodness" ρ is scaled by c and fed into a sigmoid function to ensure the value of β_t ranges between 0 and 1. Additionally the function is centred, such that when the input is 0 (neutral sympathy), their rewards are weighted equally with $\beta_t = 0.5$. c is set such that volatile switching in β_t values for small changes in ρ are avoided. More details in Section 3.4.

2.2.4 State prediction model learning. As described in Section 2.2.3, computing the sympathy function requires a state prediction model $M(s, a)$. This model takes in as input, the current state s_t, and for each action possible from s_t, predicts the next state $\hat{s}_{t+1} \in \hat{S}_{t+1}$ where $\hat{S}_{t+1}$ is the set of all possible future states from s_t and taking available actions. This model is trained in a supervised manner using the state-action-next state (s, a, s') transitions of the learning agent, behaving as per Q_{symp}. Details of the model training is described in Algorithm 4. Although it may not always be possible to build an accurate model of the environments, we show in Appendix A that our approach is robust to certain levels of model errors.

Details of the full framework is described in Algorithm 1.

3 EXPERIMENTS

In this work, we consider environments where a learning agent cohabits with the independent agent, which is assumed to be following a fixed policy (e.g. a robot learning to share the environment with a human). We consider two environment settings: an adversarial setting, where both agents can harm each other (and the learning agent is rewarded for doing so), and an assistive setting, where the independent agent benefits from being helped by the learning agent [1]. Both environments are set up such that the agent can only observe its immediate surroundings (controlled through a field of vision around the agent), and auxiliary information such as the position of the agent is shared with both agents. In both settings, we evaluate our approach against (a) fully selfish agents, (b) fully altruistic agents, and (c) the case where β_t is set to constant intermediate weightings, as per Alamdari et al. [1]. We now describe each environment setting in greater detail.

3.1 Adversarial Setting

The adversarial setting was realised through the Pacman environment (adapted from The Pacman project[2]). As in Figure 2(a)-(c) (showing three game sizes), the game consists of two agents, Pacman (learning agent) and a single ghost (independent agent). The ghost behaves according to a fixed policy that actively aims to kill Pacman with a probability of 0.8, and with a probability of 0.2, it takes random actions. The goal of the learning agent (Pacman) is to consume all of the pellets (+10 reward each) in order to win (+100), while avoiding being killed (when their positions coincide) by the independent agent, for which it receives a negative reward of −500.

Power pellets, when consumed by the learning agent, temporarily allow it to harm the independent agent, awarding the learning agent positive rewards (+50), and resetting the independent agent to its starting point.A time penalty (−1) applies for each step taken in the absence of other rewards.

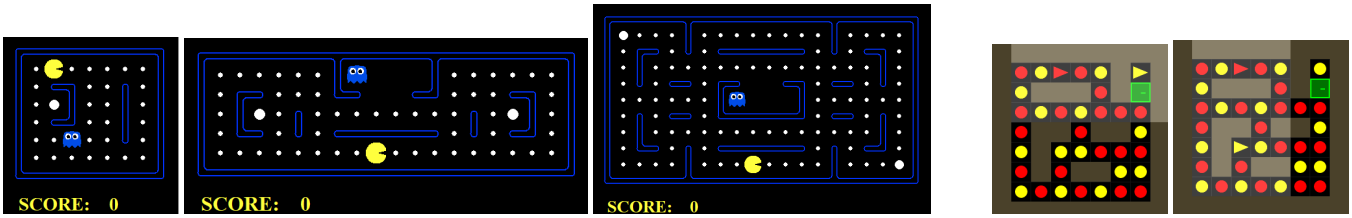
The reward features for this game are: consuming a pellet (*Pellet*), harming the ghost (*Harm*), being killed by the ghost (*Killed*), winning the game (*Win*), and taking a step (*Step*). When trained using a standard reward maximising algorithm such as DQN [9], the learning agent has an incentive to consume the power pellets and harm the independent agent while attempting to consume all the pellets. For the learning agent, an episode terminates when all pellets are consumed, or if it is killed by the independent agent. For each game size in Figure 2(a)-(c), the field of vision (representing part of the agent's state) is of size 7×7, centred around the agent. Additional state information includes the position of both agents as well as whether the power pellet is active.

3.1.1 Results. Trained with $c = 1$, Figure 3(a) - (c) show the sympathetic Pacman achieving high win rates, while reducing the number of times the Ghost/independent agent is harmed in the process, shown in Figure 3(d)-(f). Interestingly the sympathetic framework has resulted in win rates and rewards that are higher than that obtained by 'selfish' Pacman. We speculate this is likely due to the simpler policy required when behaving sympathetically (i.e. learning to avoid crossing paths with the ghost regardless of whether the power pellet is active). When comparing against the various constant β_t values, the results show that not all values of β_t produced both high win rates and low harm to the ghost. Though there are certain values of β_t that produce the desired results, setting the correct value before beginning experimentation can be difficult. In our work, we alleviate the need to pre-specify a constant value of β_t, and instead, allow the β_t value to adapt dynamically depending on the situation (state-dependent sympathy). Although the behaviour of our agent depends on the hyperparameter c, it is relatively easier to set a reasonable value for c, as discussed later in Section 3.4.

3.2 Assistive Setting

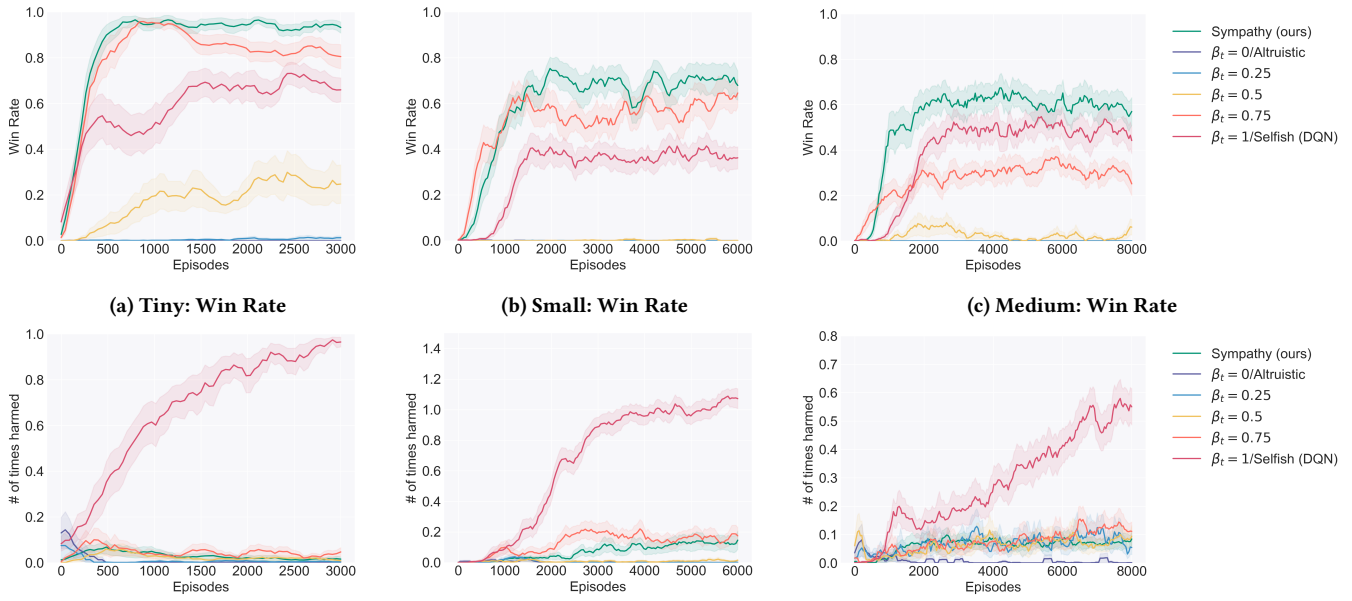
The second environment setting, we adapt from the MARL Minigrid environment [4] and shown in Figures 2(d)-(e), is one in which the the independent agent (yellow arrow) requires assistance from the learning agent (red arrow) to achieve its goal (but not vice versa). In this game, each agent is tasked with consuming their corresponding coloured pellets. Similar to the adversarial setting, the independent agent acts according to an ϵ-greedy policy (where $\epsilon = 0.2$). The learning agent receives a reward for each red pellet consumed (+10), a penalty for each move (−1) and reward for collecting all the pellets (ending the game) (+5). A closed door is located in the space which only the learning agent can open. However, it incurs a

[1] Experimental code and supplementary materials found at https://tinyurl.com/4cj76pbs
[2] http://ai.berkeley.edu/project_overview.html



(a) Tiny (b) Small Classic (c) Medium Classic (d) Gridworld 1 (e) Gridworld 2

Figure 2: Adversarial Environments (a) - (c): Goal in each game is for Pacman (learning agent) to consume all the small white pellets. A Ghost (independent agent) exists within the environment and behaves according to a fixed ϵ-greedy policy where by it chases after Pacman to kill him. Within the environment are power pellets (large dots) which, when consumed by Pacman, endow him with the ability to harm the Ghost. In small classic and medium classic environments, two power pellets exist, allowing Pacman two opportunities to harm the Ghost. Assistive environment (d) - (e): Goal of learning agent (red triangle) and independent agent (yellow triangle) is to collect their corresponding coloured pellets. Closed door can only be opened by learning agent (and inflicts a negative reward). Gridworld 1: Independent agent requires learning agent to open door to access room and pellets. Gridworld 2: Independent agent requires learning agent to open door to access one of its pellets.



(a) Tiny: Win Rate (b) Small: Win Rate (c) Medium: Win Rate

(d) Tiny: Times Ghost is harmed (e) Small: Times Ghost is harmed (f) Medium: Times Ghost is harmed

Figure 3: Results of Pacman environments. Comparison between Sympathetic Agent, Selfish Agent and various constant β_t values.

negative reward (−1) for doing so. Once opened, the door remains open, but no further negative penalty applies to the learning agent. Episodes terminate when the learning agent has consumed all of its pellets, or when the time limit of 1000 moves (between both agents) is reached. The reward features for the games are: the independent agent consuming a pellet (*Indep Pellet*), the learning agent consuming a pellet (*Learn Pellet*), the door being opened (or open status in Gridworld 2) (*Door*), the independent agent winning the game (*Win*), and taking a step (*Step*). The field of vision around each agent is of size 5×5, with additional state information being the position of both agents and whether the door is open/closed.

We test our framework on two layouts, shown in Figure 2(d) and (e). In the first (Gridworld 1), the independent agent is locked behind the door and cannot access pellets unless the learning agent opens the door. When opened, the reward function feature for the door opening is only activated for the initial opening of the door. In the second environment (Gridworld 2), one of the independent agent's pellets is behind the door and requires assistance from the learning agent to reach it. In this environment the reward feature for the open door remains active as long as the door is open. Gridworld 1 is designed to demonstrate a situation where the benefit from the learning agent opening the door is realised immediately with

Table 1: Scaled reward weights inferred via IRL

Gridworld			Pacman		
	1	2		Tiny	Small
Indep Pellet	1.2±0.4	3.2±0.7	*Pellet*	-29±14	-17±20
Learn Pellet	-1.7±0.5	-3.3±0.8	*Win*	-14±18	-17±21
Door	10.8±1.3	-2.3±0.9	*Harm*	-290±29	-150±38
Win	-0.02±0.6	0.2±1.0	*Kill*	28±10	75±14
Step	-1.9±0.6	-4.2±0.8	*Step*	-257±26	-224±28

a high positive reward associated with opening the door. This is observed in the IRL results in Table 1, where the *Door* feature is associated with high reward values. Gridworld 2 highlights a situation where the benefit of opening the door may not be realised immediately (as the independent agent may be located anywhere in the environment when the door is opened). This is reflected in Table 1, showing a negative reward for actually opening the door, but a positive reward for the independent agent consuming a pellet.

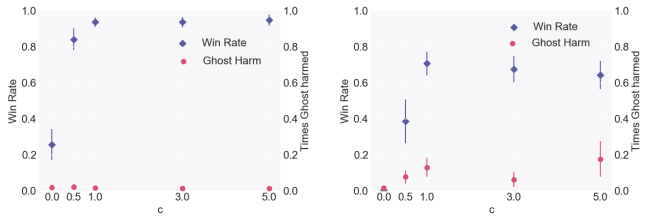
3.2.1 Results. Figure 4 illustrates the results of experimentation on the Assistive environments. In both environments, if trained to merely maximise rewards ($\beta = 1$), the learning agent has no incentive to assist the independent agent, as indicated by the selfish baseline performance curves. In Gridworld 1, the sympathetic framework results in an agent who not only completes its task (and consumes all its own pellets), but also opens the door for the independent agent (Figure 4(b)-(c)). In Gridworld 2, compared to Gridworld 1, a similar win rate is seen, but the door opens a little less frequently (Figure 4(e)-(f)). We attribute this difference to the delay in observing a positive benefit gained by the independent agent from the door being opened in Gridworld 2 compared to Gridworld 1. When comparing performance of constant β values, although some resulted in the door being opened more frequently, this also lead to behaviours associated with lower win rates.

3.3 Sympathy Function Values

It is of interest to examine the final learned values of selfishness β_t for various events within the games. Table 2 shows the converged β_t values (calculated as an average of the last 100 episodes of training) for various events of significance. In the Pacman environment, both sympathetic and selfish behaviours may be desirable, depending on the situation. Examining the values for $c = 1$ in the 'Tiny' Pacman environment, for actions that lead to Pacman harming the ghost (*harm* column), a relatively low value of $\beta_t = 0.27$ is learned, indicating more importance placed on the Ghost's inferred rewards (as opposed to Pacman's own rewards) for that action. In contrast, a relatively high value of $\beta_t = 0.86$ is learned for an action that leads to Pacman being killed (*Killed* column) (more weight placed on Pacman's own reward).

The Assistive Gridworld environments require less drastic differences in the β_t values based on the situation (as there are no scenarios in which the learning agent is harmed), and this is reflected in the differences in β_t values for opening the door, the independent agent consuming a pellet, and the independent agent completing the game shown in Table 2. In the Gridworlds, the β_t for opening the door and consuming pellets is relatively low (less than the 0.5 neutral value) and decreases further as c increases, thereby weighting the independent agent's rewards more heavily than the learning agent's rewards. For the event of winning, the β_t value remains roughly consistent (Gridworld 1) or increases (Gridworld 2), placing higher weighting on the reward of the learning agent.

To understand the resulting behaviour due to the changing β_t, it is important to also take into consideration the reward values themselves (for both the learning agent and the inferred rewards of the independent agent). In Gridworld 1, the inferred rewards of the independent agent ($\hat{\mathbf{r}}_{indep}$) associated with opening the door is a strong positive (shown in Table 1), which together with the low β_t value (Table 2), leads to door opening behaviours observed in Figure 4(b). In Gridworld 2, the door is also opened, but this is driven by the positive reward associated with the consumption of pellets by the independent agent. In Gridworld 2 however, the IRL associates a negative reward for the learning agent consuming pellets, which, together with the smaller β_t observed with increasing c (Table 2), lead to lower win rates. As such it is clear that the sympathetic behaviour induced is dependent not just on the reward values, but the intensity of the c hyperparameter (and subsequent β_t values).



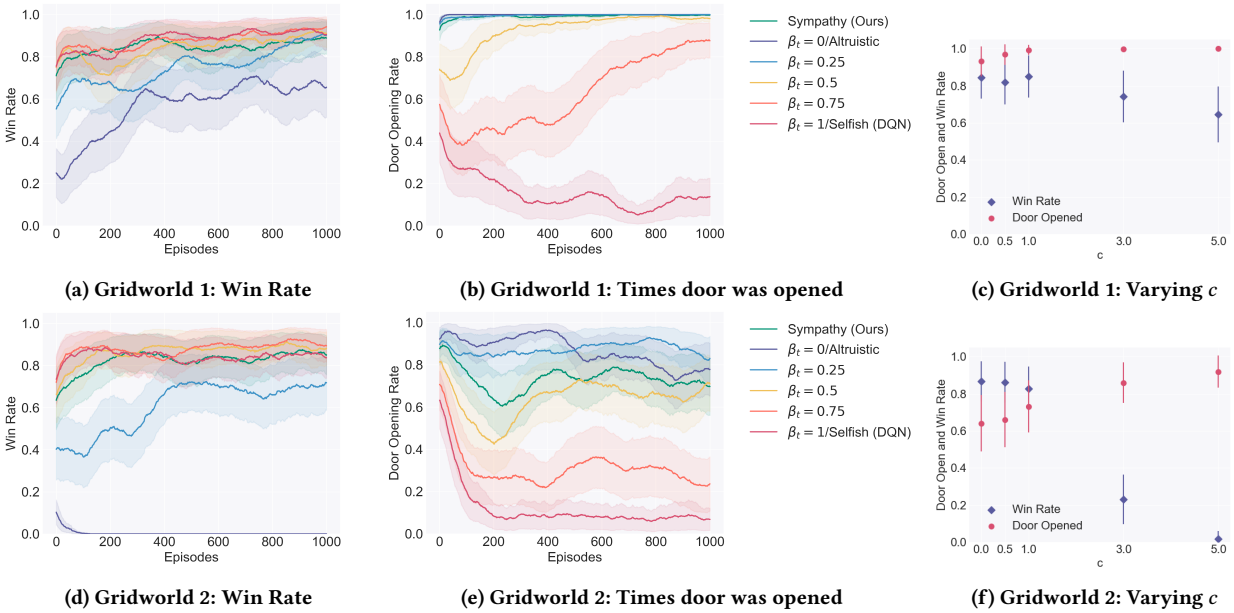
(a) Pacman (Tiny) c values **(b) Pacman (Small) c values**

Figure 5: Impact of varying the c hyperparameter on adversarial Tiny and Small classic games. Average (last 100 episodes) win rate and times ghost is harmed

3.4 c hyperparameter

We report the effect of the c hyperparameter on the performance of our sympathetic agent. This behaviour was discussed in the previous section, and is illustrated in Figure 5 for the Pacman environments, and Figure 4(c) and (f) for the Gridworld environments. In Pacman, altering c had little impact on the sympathetic behaviour (except at the minimum value of $c = 0$). For the Gridworld environments however, higher values of c lead to increased opening of the door, but reduced the win rate.

The c hyperparameter governs the reactiveness and sensitivity of the agent to inequalities in the long term returns between the two agents. For low c values, the agent is less sensitive to differences in long term returns between the agents. With c set to 0, the agent's behaviour can be described as emotionally neutral, applying a constant and equal weighting to the reward functions of both agents regardless of the situation. As c increases (forcing $\beta(s, a)$ to tend towards a step function), the agent becomes more volatile, reacting to minute differences in ρ in Equation 4 with extreme values (0 or 1) of β_t. As a rule-of-thumb, we found setting $c = 1$ provides an effective balance for the environments considered in this work.



(a) Gridworld 1: Win Rate **(b) Gridworld 1: Times door was opened** **(c) Gridworld 1: Varying c**

(d) Gridworld 2: Win Rate **(e) Gridworld 2: Times door was opened** **(f) Gridworld 2: Varying c**

Figure 4: Results of Gridworlds: Comparison between Sympathetic Agent, Selfish agent and various constant β_t values. (c) and (f) win rate and door open rate as c is varied, averaged over the last 100 episodes.

Despite a drop in win rate for values of c that significantly deviate from our recommendation ($c = 1$), we contend that our approach achieves state and action dependent sympathetic behaviours, unlike other approaches [1] that rely on a user-specified, constant sympathy parameter. In addition, different values of c only serve to either amplify or reduce the agent's sensitivity to the inferred IRL rewards. In this sense, c allows the user to tune the volatility of the agent's behaviour for the same inferred rewards, which is not possible in other approaches that directly tune the sympathy value.

4 CONCLUSION

In this work, we presented a novel framework for training a reinforcement learning agent to behave sympathetically towards another agent in its environment. This sympathetic behaviour was achieved via a sympathetic reward function expressed as a convex combination of the learning agent's reward and the inferred reward of the other agent, which we learned through inverse reinforcement learning. We showed that the corresponding convex weight (the sympathy function) can be computed online on the basis of the difference in the estimated long term rewards of an action for both agents, relative to all other actions that could have been taken. We validated our proposed framework on the adversarial Pacman environment, as well as on assistive Gridworld environments which we introduced. In both environment settings, we empirically demonstrated the ability of our learning agent to behave considerately, while simultaneously pursuing its own goals. We discussed the properties of our framework and described the influence of the introduced hyperparameter through appropriate studies.

We believe our proposed framework could serve as a platform for conducting research into other related directions for incorporating sympathetic behaviours. Our presented framework assumes a discrete action space and shared reward feature space between agents. Hence, extending our approach to continuous action spaces, and heterogeneous feature spaces could enhance its practical feasibility. In addition, our framework currently considers only one independent agent. In the future, we seek to extend the framework to accommodate multiple independent agents, each with their own unique (and possibly conflicting) goals. Investigating model-free alternatives to the sympathy function used in our framework could also be a future line of research worth exploring.

Table 2: Learned β_t values for key events in Tiny Pacman and Gridworld environments.

	Gridworld 1			Gridworld 2			Pacman (Tiny)	
c	*Door*	*Pellet*	*Win*	*Door*	*Pellet*	*Win*	*Harm*	*Killed*
0.5	0.39±0.001	0.4±0.001	0.55±0.01	0.56±0.01	0.41±0.001	0.58±0.01	0.37±0.05	0.72±0.04
1	0.29±0.001	0.32±0.001	0.54±0.01	0.57±0.02	0.32±0.001	0.66±0.02	0.27±0.07	0.86±0.02
3	0.09±0.001	0.1±0.001	0.57±0.01	0.11±0.01	0.08±0.01	0.63±0.02	0.06±0.05	0.98±0.05
5	0.02±0.001	0.04±0.001	0.56±0.01	0.01±0.001	0.01±0.001	0.71±0.03	0.05±0.05	0.99±0.05

REFERENCES

[1] Parand Alizadeh Alamdari, Toryn Q Klassen, Rodrigo Toro Icarte, and Sheila A McIlraith. 2021. Be Considerate: Objectives, Side Effects, and Deciding How to Act. *arXiv preprint arXiv:2106.02617* (2021).

[2] Nick Bostrom and Eliezer Yudkowsky. 2014. The ethics of artificial intelligence. *The Cambridge handbook of artificial intelligence* 1 (2014), 316–334.

[3] Bart Bussmann, Jacqueline Heinerman, and Joel Lehman. 2019. Towards empathic deep q-learning. In *Artificial Intelligence Safety 2019 (CEUR Workshop Proceedings)*, Huáscar Espinoza, Han Yu, Xiaowei Huang, Freddy Lecue, Cynthia Chen, José Hernández-Orallo, Seán Ó hÉigeartaigh, and Richard Mallah (Eds.). CEUR-WS.org, 1–7.

[4] Maxime Chevalier-Boisvert, Lucas Willems, and Suman Pal. 2018. Minimalistic Gridworld Environment for OpenAI Gym. https://github.com/maximecb/gym-minigrid.

[5] Tim Franzmeyer, Mateusz Malinowski, and João F Henriques. 2021. Learning Altruistic Behaviours in Reinforcement Learning without External Rewards. *arXiv preprint arXiv:2107.09598* (2021).

[6] Jacaueline Hemminahaus and Stefan Kopp. 2017. Towards adaptive social behavior generation for assistive robots using reinforcement learning. In *2017 12th ACM/IEEE International Conference on Human-Robot Interaction (HRI*. IEEE, 332–340.

[7] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. 2019. Agent modeling as auxiliary task for deep reinforcement learning. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 15. 31–37.

[8] Edouard Klein, Bilal Piot, Matthieu Geist, and Olivier Pietquin. 2013. A Cascaded Supervised Learning Approach to Inverse Reinforcement Learning. In *Machine Learning and Knowledge Discovery in Databases*, Hendrik Blockeel, Kristian Kersting, Siegfried Nijssen, and Filip Železný (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–16.

[9] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *nature* 518, 7540 (2015), 529–533.

[10] Kamal K Ndousse, Douglas Eck, Sergey Levine, and Natasha Jaques. 2021. Emergent social learning via multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 7991–8004.

[11] Andrew Y. Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *In Proceedings of the Sixteenth International Conference on Machine Learning*. Morgan Kaufmann, 278–287.

[12] Andrew Y. Ng and Stuart J. Russell. 2000. Algorithms for Inverse Reinforcement Learning. In *Proceedings of the Seventeenth International Conference on Machine Learning (ICML '00)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 663–670.

[13] Andrew Y Ng, Stuart J Russell, et al. 2000. Algorithms for inverse reinforcement learning.. In *Icml*, Vol. 1. 2.

[14] Ritesh Noothigattu, Djallel Bouneffouf, Nicholas Mattei, Rachita Chandra, Piyush Madan, Kush R Varshney, Murray Campbell, Moninder Singh, and Francesca Rossi. 2019. Teaching AI agents ethical values using reinforcement learning and policy orchestration. *IBM Journal of Research and Development* 63, 4/5 (2019), 2–1.

[15] Martin L Puterman. 2014. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

[16] Stuart Russell and Peter Norvig. 2002. Artificial intelligence: a modern approach. (2002).

[17] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.