Learning Against Non-Stationary Agents with Opponent Modelling and Deep Reinforcement Learning

Richard Everett, Stephen Roberts
Department of Engineering Science
University of Oxford
{richard, sjrob}@robots.ox.ac.uk

Abstract

Humans, like all animals, both cooperate and compete with each other. Through these interactions we learn to observe, act, and manipulate to maximise our utility function, and continue doing so as others learn with us. This is a decentralised non-stationary learning problem, where to survive and flourish an agent must adapt to the gradual changes of other agents as they learn, as well as capitalise on sudden shifts in their behaviour. To learn in the presence of such non-stationarity, we introduce the *Switching Agent Model* (SAM) that combines traditional deep reinforcement learning – which typically performs poorly in such settings – with opponent modelling, using uncertainty estimations to robustly switch between multiple policies. We empirically show the success of our approach in a multi-agent continuous-action environment, demonstrating SAM's ability to identify, track, and adapt to gradual and sudden changes in the behaviour of non-stationary agents.

1 Introduction

Cooperation and competition are the cornerstones of both human and animal societies, appearing deeply embedded in our understanding of social intelligence. For an individual agent to maximise their utility function in these societies, they must learn to interact with and against others, as well as understand the consequences of their actions. Studies on this interaction have a long history across domains including game theory (Rapoport and Chammah 1965), evolutionary biology (Strassmann et al. 2011), and multi-agent systems (Shoham, Powers, and Grenager 2007).

A key component of learning to interact with others is the ability to reason about their behaviour. This is accomplished through constructing and utilising models of their decision-making processes, and is commonly referred to as *opponent modelling*. Due to recent advances in machine learning, autonomous agents are increasingly interacting with others, be it negotiating with humans (Lewis et al. 2017) or communicating with other agents (Foerster et al. 2016). It is therefore important that they are able to take into account more than just their own agency.

If done correctly, these models can be used to derive an optimal policy against an agent, such as by exploiting their suboptimal behaviour to yield a higher reward (Ganzfried and Sandholm 2011). However, the construction of these models is non-trivial as an agent's behaviour is rarely stationary, thus requiring any learned opponent model to be continuously updated (Hernandez-Leal et al. 2017b). For example, such non-stationarity can occur in human-computer interaction, whereby a user's behaviour changes gradually as they become familiar with a system, as well as suddenly when the user switches with another user. Likewise, in a competitive setting an agent may initially learn before switching to an alternative strategy as its beliefs about the world change.

One of the most successful paradigms for learning a policy is reinforcement learning (Sutton and Barto 1998), whereby agents learn to maximise their cumulative long-term reward through trial-and-error interactions with their environment. In recent years, the area has experienced a string of successes in the single-agent domain due to advances in deep learning (Mnih et al. 2015). However, the non-stationarity that arises from interacting with other agents renders many single-agent algorithms unsuitable (Hernandez-Leal et al. 2017a). By not taking into account the agency of others, traditional deep reinforcement learning methods struggle to transfer their successes to the multi-agent setting.

Within deep reinforcement learning, opponent modelling has started to receive increasing attention. It has been successfully applied to modelling the policy of agents which switch between episodes (He et al. 2016), albeit requiring handcrafted behavioural features, and more recently it has been used to learn approximate policies of learning agents (Foerster et al. 2017; Lowe et al. 2017). However, to date there has been little progress made on folding in uncertainty into these deep models – a vital component for a robust opponent model – and both forms of non-stationarity have been rarely considered.

In this work, we consider the setting where multiple independent non-stationary agents interact in the same environment. Specifically, we look at two distinct forms of non-stationary behaviour:

1. *Sudden* changes, whereby an agent switches between a set of behaviours through time.
2. *Gradual* changes, whereby an agent's behaviour slowly adjusts over time as it learns.

To learn and succeed in this setting, we propose the *Switching Agent Model* (SAM). By combining traditional deep reinforcement learning algorithms with models of the decision-making processes of other agents, our method learns a general policy which is more robust and adaptive to non-stationary agents. We achieve this combination by explicitly learning from an agent's state-action trajectory with an approximate Bayesian neural network, using Monte Carlo dropout (Gal and Ghahramani 2016) to obtain predictive uncertainty of our model's predictions. The model's predictive error and uncertainty are tracked by a switchboard to robustly identify changes in an opposing agent's behaviour, switching between opponent models and their associated policies through time.

We demonstrate the capabilities of SAM through two experiments in a multi-agent continuous-action environment. First, we show that our approach can identify, track, and adapt to the behaviour of an agent which switches between policies over time, outperforming traditional deep reinforcement learning. Next, we show that the same method also helps in the presence of a learning agent, yielding a higher performance as a result. We finish with an analysis into the uncertainty of our learned opponent models throughout training in both experiments.

Algorithm 1: Model Switching Algorithm

Input: opponent models $\hat{\mu} = \{\hat{\mu}^1, ..., \hat{\mu}^K\}$, error threshold $r_{\max}$, error decay d, predict action parameters $Z = (N, p, \mathcal{N})$
Initialise running error $r \leftarrow 0$
Initialise current opponent model index $k \leftarrow 1$
for episode $= 1$ to ... **do**
  Receive initial state s_0
  $\hat{a}_0, \eta_0 \leftarrow$ Predict$(s_0, \hat{\mu}^k, Z)$; `// Algorithm 2`
  for timestep $t = 1$ to ... **do**
    Receive state s_t and action a_{t-1}
    Update running error r using $(a_{t-1}, \hat{a}_{t-1}, \eta_{t-1})$
    if $r >= r_{\max}$ **then**
      Switch to different opponent model, updating k
      Reset rolling error $r \leftarrow 0$
    else
      Decay rolling error $r \leftarrow r - d$
    end
    Train $\hat{\mu}^k$ on (s_{t-1}, a_{t-1})
    $\hat{a}_t, \eta_t \leftarrow$ Predict$(s_t, \hat{\mu}^k, Z)$
  end
end

2 Switching Agent Model (SAM)

The motivation behind our work is that opposing agents can change their behaviour through time, and therefore our ability to derive an optimal policy in their presence depends on how well we can identify and track this change.

To track these changes and learn an appropriate response, we propose the *Switching Agent Model*. SAM is a collection of inferred opposing agent policies $\hat{\mu}$ and associated approximate best-response policies μ which are connected through, and controlled by, a switchboard. In the following sections we describe each of these components in detail. To aid readability, we refer to inferred opposing agent policies as 'opponent models' and learned approximate best-response policies as 'response policies'. Furthermore, to ease notation we omit the parameters θ and ϕ of μ and $\hat{\mu}$ respectively.

2.1 Switchboard

At the heart of SAM is the switchboard. It tracks the performance of opponent models through time, switching between them and their associated response policies as the opposing agent adjusts its behaviour.

As we consider agents which can adapt both gradually and suddenly, it is important that our switching mechanism can operate in the presence of both changes. To achieve this, our switchboard tracks the running error of the opponent models, expecting notable spikes when an agent switches behaviour and a gradual accumulation over time as they learn. In both of these situations, the value of this running error can be used to initiate a switch. We describe this switching process here and also present it in Algorithm 1.

While an opponent model $\hat{\mu}^k$ is active, the switchboard monitors its running error r. At each timestep t, the active model predicts the next action of the opposing agent using Monte Carlo dropout, obtaining an action prediction $\hat{a}_t^j$ along with its associated predictive uncertainty η_t. On the following timestep $t+1$, the true action a_t^j is observed and the running error is updated as follows:

$$r = r + \frac{|a_t^j - \hat{a}_t^j|}{\eta_t}. \tag{1}$$

If the running error r is less than the specified switch threshold, i.e. $r < r_{\max}$, then the running error is decayed by d. Otherwise, a switch occurs whereby a different opponent model is chosen and the running error is reset, $r \leftarrow 0$.

Similar to related approaches which consider switching behaviours (Hernandez-Leal et al. 2016), we assume that the modelled agent will not switch while our method is initially learning an opponent model.

2.2 Response Policies

To learn a policy which can act in an environment, as well as take advantage of our inferred opponent models, we use the Deep Deterministic Policy Gradient (DDPG) algorithm (Lillicrap et al. 2015). We refer to these as 'response' policies due to their association with a specific opponent model.

By alternating between models over time according to the opposing agent's behaviour, and therefore alternating between response policies, our general policy is comprised of multiple sub-policies $\mu = \{\mu^1, ..., \mu^K\}$.

Each policy μ^k is trained by sampling from its own replay buffer $\mathcal{D}^k$, learning an approximate best-response to the historical average behaviour of the opponent contained within the buffer. The learned response policies are then used by the agent to select actions given their observed state s_t^i and the predicted next actions from the associated opponent model $\hat{a}_t = \hat{\mu}^k(s_t^i)$, plus some optional noise from an exploration

Algorithm 2: Predict Action Algorithm

Input: state s, opponent policy $\hat{\mu}$, Z=(number of passes N, dropout probability p, inherent noise $\mathcal{N}$)
Output: action prediction $\hat{a}$, predictive uncertainty η
for $n = 1$ to N **do**
  $\hat{a}_n \leftarrow \hat{\mu}(s)$; `// with dropout (p)`
end
$\hat{a} \leftarrow \frac{1}{N}\sum_{n=1}^{N} \hat{a}_n$; `// model prediction`
$\hat{\sigma}^2 \leftarrow \frac{1}{N}\sum_{n=1}^{N}(\hat{a}_n - \hat{a})^2$; `// model uncertainty`
$\eta \leftarrow \sqrt{\hat{\sigma}^2 + \text{Var}(\mathcal{N})}$; `// predictive uncertainty`

process:

$$a_t = \mu^k(s_t^i \| \hat{a}_t) + \mathcal{N}_t. \tag{2}$$

2.3 Opponent Models

The final component of SAM is the set of inferred opposing agent policies, i.e. opponent models $\hat{\mu}$, which are learned from observed state-action trajectories and switched between across time by the switchboard.

Our method takes advantage of these learned opponent models in two distinct ways. First, as shown in Equation 2, we input their predictions of the agent's next actions directly into our response policy. Depending on the quality of our models, this can help reduce the perceived non-stationarity of the environment. Second, we use the model's uncertainty estimations in its predictions to obtain a measure of normal and unexpected behaviour, allowing our switchboard to change models accordingly.

As our agent acts in a domain with continuous actions, predicting an agent's next action is a regression problem. In this context, our model's uncertainty can be thought of as a confidence interval around its predictions.

To learn an approximation of agent j's true policy μ_j^k, we use a neural network $\hat{\mu}_j^k$ parametrised by ϕ_j^k which we optimise by minimising the loss:

$$L(\phi_j^k) = \left(\hat{\mu}^k(s_{t-1}^i) - a_{t-1}^j\right)^2, \tag{3}$$

where s_{t-1}^i is the previously observed state by agent i and a_{t-1}^j is the true action which agent j performed.

To identify changes in the behaviour of an agent, our opponent models need a measure of uncertainty to detect and assign error to unexpected actions. We do this by approximating our model's predictive uncertainty using Monte Carlo dropout (Gal and Ghahramani 2016). The process for this is presented in Algorithm 2 and described below.

First, to obtain our model's prediction of agent j's true next action a_{t+1}^j we pass the observing agent's state s_t^i through the network N times, where on each pass the output is computed by randomly dropping out each hidden unit with probability p:

$$\hat{a}_{t+1}^j = \frac{1}{N}\sum_{n=1}^{N} \hat{\mu}^k(s_t^i). \tag{4}$$

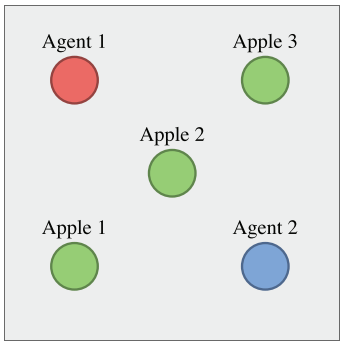

Figure 1: Gathering environment.

Next, to obtain predictive uncertainty we first determine our model's uncertainty which can be approximated using the sample variance:

$$\hat{\sigma^2}(\hat{\mu}^k(s_t^i)) = \frac{1}{N}\sum_{n=1}^{N}\left(\hat{\mu}^k(s_t^i) - \hat{a}_{t+1}^j\right)^2. \tag{5}$$

Our complete predictive uncertainty is thus given by:

$$\eta = \sqrt{\hat{\sigma^2}(\hat{\mu}^k(s_t^i)) + \text{Var}(\mathcal{N})}, \tag{6}$$

where $\mathcal{N}$ is the inherent noise in the agent's actions.

Both of the outputs of this process, namely the prediction $\hat{a}$ and its predictive uncertainty η, are used by the switchboard to determine the running error r of the current opponent model (see Equation 1).

3 Experiments

In our experiments, we train an agent against a non-stationary adversary with the aim of showing that traditional reinforcement learning performs sub-optimally in such a setting, and that our proposed methods – i.e. deep opponent modelling with uncertainty estimates and multiple policies – help improve performance.

We compare the performance of SAM and DDPG. This comparison is done indirectly in our first experiment, placing both agents against a separate switching adversary, while in second experiment we compare them directly.

3.1 Experimental Details

Environment: Both of our experiments take place in a gathering scenario between two independent agents in a continuous world populated by apples as shown in Figure 1.

In this scenario, agents can receive a reward in two ways:

1. Collecting apples by moving onto them, yielding a small reward.
2. Stealing from another agent by colliding into them, yielding a larger reward but penalising the other agent.

Under this reward structure, an agent's behaviour can be characterised along a scale ranging from purely cooperative (i.e. never stealing) to purely defective (i.e. only stealing). An agent's optimal behaviour along this scale depends on

the behaviour of their opposing agent. For example, against a purely cooperative agent it is desirable to steal more, while against a purely defective agent it is advantageous to focus on avoiding them while collecting apples as fast as possible.

At each timestep, agents observe their own position and velocity, their relative position and velocity to the opposing agent, their relative position to each apple, and whether they can steal. In addition, they can move around the environment by exerting a continuous force in two directions.

To help speed up training, each agent is given a small negative reward proportional to their distance to the nearest apple, as well as a small negative reward proportional to their distance to the opposing agent when it is possible to steal.

Model Architecture: The actor and critic networks in our response policies have 2 hidden layers with 64 hidden units in each layer, ReLU activation functions, and tanh on the actor's output layer. Opponent model networks consist of 3 hidden layers with 64 hidden units in each layer, ReLU activation functions, and tanh on the output layer.

Algorithm Parameters: We train using Adam (Kingma and Ba 2014) with a learning rate of 0.0001 and 0.001 for the actors and critics respectively. Soft target updates are done with $\tau = 0.01$. We set the discount factor to $\gamma = 0.95$, store 10^6 transitions in our replay buffer (which are equally distributed between the K buffers in SAM), and train on mini-batches of 64 transitions. During training, exploration is achieved by adding noise using an Ornstein-Uhlenbeck process with $\sigma = 0.1$.

Our opponent models are trained with a learning rate of 0.001. Predictive uncertainty is obtained using $N = 30$ forward passes with dropout probability $p = 0.10$, and errors are decayed by $d = 5$ with the switch threshold set to $r_{\max} = 50$. Additionally, we set $K = 2$ (i.e. two opponent models and two response policies), and therefore switch calls alternate k between 1 and 2.

Train & Test Procedure: For training, we segment timesteps into episodes of 1,000 timesteps, resetting the environment at the start of each episode. Agents sample a mini-batch of transitions to learn from at each timestep, and actions are selected with an added exploration noise. Between training episodes we test agents for a further 1,000 steps, selecting actions with no exploration noise.

3.2 Experiment 1: Sudden Changes

In this experiment, we consider the setting of a learning agent competing against a non-stationary adversary that switches between policies throughout time. To perform optimally in this setting, an agent needs to be able to quickly identify when the adversary has switched behaviours and respond accordingly.

Switching Adversary To construct the policies used by the switching adversary, we train two agents with different reward schedules:

1. Passive: rewarded for collecting apples and penalised for collisions.

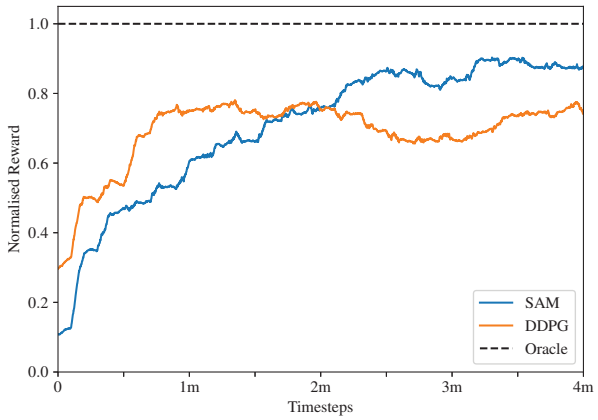

Figure 2: Average reward of SAM (blue) and DDPG (orange) against the switching adversary. Normalised by the oracle's best performance (black dashed line) and smoothed across switches to improve readability. Average of 5 trials.

2. Offensive: only rewarded for stealing apples from the opposing agent.

The switching adversary is made non-stationary by switching between these two learned policies, whereby the agent follows one policy for a number of timesteps before switching to the other, repeating this process through time. To meet our assumption mentioned in Section 2.1, we enforce a minimum number of timesteps between switches.

As the switching adversary's behaviour changes through time, so too does the optimal response. Specifically, the optimal strategy against a passive agent is to steal apples, while the optimal strategy against an offensive agent is to avoid them while collecting apples.

When comparing agents against the switching adversary, we normalise their performance metrics by the highest metrics achieved by an agent with access to true state of the switching adversary (known as the oracle).

Results In Figure 2, we present the reward of each agent throughout training, normalising by the performance of the oracle. As can be seen, there are clear differences between the performances of the two agents.

Due to the DDPG agent learning one response policy rather than two, it can reuse what is has learned in new situations, even if that behaviour is not necessarily optimal. This leads to DDPG quickly performing better than SAM. However, again due to its single policy, it learns a best response to the switching adversary's average behaviour, leading to a suboptimal performance as training continues. In contrast, SAM is able to learn a specific best response to each of the agent's behaviours, yielding an eventual higher reward.

Switch Analysis As our model relies on detecting changes in the opposing agent's behaviour, it is important to know how long it takes to detect a change once is has occurred. To do this, we run our trained opponent model over known change points, which the model does not know, measuring the error rate and logging when a change occurs.

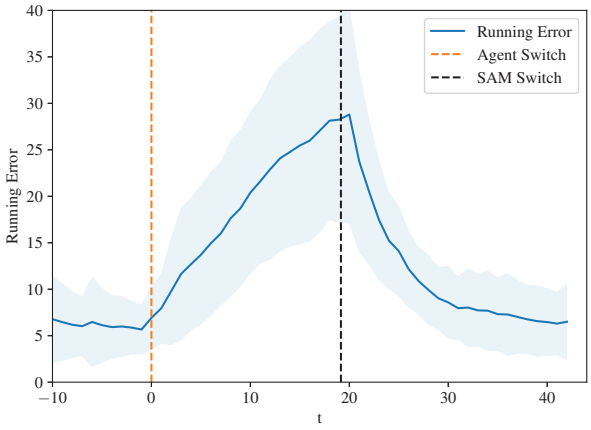

Figure 3: Average running error of our opponent model (blue). Adversary agent switches behaviour at $t = 0$ (dotted orange) and SAM switches around $t = 19$ (dotted black). Results are averaged across 100 examples.

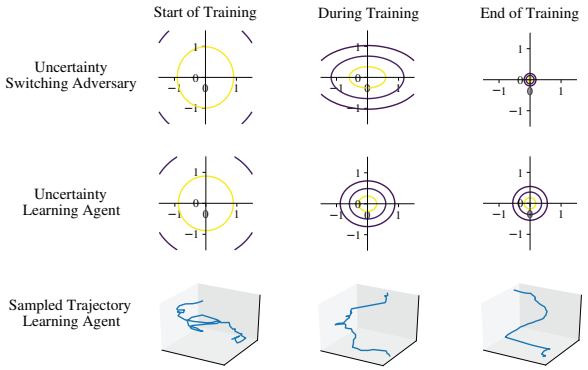

Figure 4: Predictive uncertainty of the opponent model trained on the switching adversary (top) and learning agent (middle) for the same state, along with the learning agent's trajectory (bottom). Columns represent the stage of training from start (left) to end (right).

The results of this are presented in Figure 3, whereby we visualise the running error of a learned opponent model for 100 switches, centering the agent's switch on timestep $t = 0$. On average, our method is able to detect a switch in the adversary's behaviour after 19.2 ± 4.4 timesteps as indicated by the dotted black line. In other words, on average we can detect a change in an agent's behaviour from observing $\approx$19 of their actions.

3.3 Experiment 2: Gradual Changes

In this experiment, we consider the setting of two agents simultaneously learning in the same environment. We train a SAM agent against a DDPG agent, evaluating them after training for 10,000 further timesteps with no exploration noise in their action selection.

Normalising performance metrics by the total achieved by both agents, we find that SAM manages to obtain a higher reward than DDPG (0.57 to 0.43), doing so by both stealing more (0.53 to 0.47) as well as collecting more apples (0.56 to 0.44). Along with our results from Experiment 1, this suggests that our methods help improve learning against a non-stationary agent which not only changes suddenly but also gradually as they learn.

3.4 Uncertainty Analysis

As we use our uncertainty estimations to determine when to switch policies, it is important that they are robust and have meaning across training. In Figure 4, we present our findings on this from the start, middle, and end of training from the previous two experiments.

At the top we visualise our model's predictive uncertainty in the switching adversary's actions along two dimensions, where each ring represents one standard deviation from the zero-centered mean. As can be seen, predictive uncertainty is high at the start of training and markedly low at the end. This is encouraging as it means that our model will return errors with high confidence when the switching adversary starts behaving anomalously compared to our current beliefs, causing a switch to occur. We saw this in practice in our first experiment (see Figure 3).

While it is intuitive that introducing uncertainty will help against a switching adversary, it is less intuitive when applied to an agent who is also learning. To investigate this, we repeat the previous analysis but instead consider a learning agent (middle row) and visualise their trajectory through time (bottom row).

From these rows, we can see that our model's predictive uncertainty decreases as the agent's actions become more consistent and meaningful. In comparison to the switching adversary's result, the predictive uncertainty is higher due to the learning agent selecting actions with some exploration noise. We conclude from this that an agent's trajectory (and therefore their actions) is indicative of the stage of training, and that our opponent models are able to capture this.

4 Conclusions & Future Work

In this work, we proposed the *Switching Agent Model* (SAM) as a way of learning in the presence of non-stationary agent behaviour. We achieved this through combining traditional deep reinforcement learning with opponent modelling, using uncertainty estimations from Monte Carlo dropout to robustly switch between opponent models and their associated response policies. We empirically demonstrated the benefits of our approach in a continuous-action environment against two types of agents and presented insights into the uses of uncertainty.

Future work will further investigate the applicability of opponent modelling in the presence of another deep learning agent. Additionally, we will also investigate the dynamics of our switching strategy in the presence of other types of non-stationary agents, such as those who are actively trying to exploit the switching mechanism, looking at how it compares to alternative strategies.

References

Foerster, J.; Assael, Y. M.; de Freitas, N.; and Whiteson, S. 2016. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, 2137–2145.

Foerster, J. N.; Chen, R. Y.; Al-Shedivat, M.; Whiteson, S.; Abbeel, P.; and Mordatch, I. 2017. Learning with opponent-learning awareness. *arXiv preprint arXiv:1709.04326*.

Gal, Y., and Ghahramani, Z. 2016. Dropout as a bayesian approximation: Representing model uncertainty in deep learning. In *international conference on machine learning*, 1050–1059.

Ganzfried, S., and Sandholm, T. 2011. Game theory-based opponent modeling in large imperfect-information games. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, 533–540. International Foundation for Autonomous Agents and Multiagent Systems.

He, H.; Boyd-Graber, J.; Kwok, K.; and Daumé III, H. 2016. Opponent modeling in deep reinforcement learning. In *International Conference on Machine Learning*, 1804–1813.

Hernandez-Leal, P.; Taylor, M. E.; Rosman, B.; Sucar, L. E.; and Munoz de Cote, E. 2016. Identifying and tracking switching, non-stationary opponents: a bayesian approach.

Hernandez-Leal, P.; Kaisers, M.; Baarslag, T.; and de Cote, E. M. 2017a. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*.

Hernandez-Leal, P.; Zhan, Y.; Taylor, M. E.; Sucar, L. E.; and de Cote, E. M. 2017b. Efficiently detecting switches against non-stationary opponents. *Autonomous Agents and Multi-Agent Systems* 31(4):767–789.

Kingma, D., and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Lewis, M.; Yarats, D.; Dauphin, Y. N.; Parikh, D.; and Batra, D. 2017. Deal or no deal? end-to-end learning for negotiation dialogues. *arXiv preprint arXiv:1706.05125*.

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

Lowe, R.; Wu, Y.; Tamar, A.; Harb, J.; Abbeel, P.; and Mordatch, I. 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. *arXiv preprint arXiv:1706.02275*.

Mnih, V.; Kavukcuoglu, K.; Silver, D.; Rusu, A. A.; Veness, J.; Bellemare, M. G.; Graves, A.; Riedmiller, M.; Fidjeland, A. K.; Ostrovski, G.; et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518(7540):529–533.

Rapoport, A., and Chammah, A. M. 1965. *Prisoner's dilemma: A study in conflict and cooperation*, volume 165. University of Michigan press.

Shoham, Y.; Powers, R.; and Grenager, T. 2007. If multi-agent learning is the answer, what is the question? *Artificial Intelligence* 171(7):365–377.

Strassmann, J. E.; Queller, D. C.; Avise, J. C.; and Ayala, F. J. 2011. In the light of evolution v: Cooperation and conflict.

Sutton, R. S., and Barto, A. G. 1998. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge.