An Inductive Bias for Emergent Communication in a Continuous Setting

John Isak Fjellvang Villanger*[1] and Troels Arnfred Bojesen[2]

[1]Department of Energy and Petroleum Engineering, University of Stavanger, Norway
[2]Department of Informatics, University of Bergen, Norway
john.i.villanger@uis.no, troels.bojesen@uib.no

Abstract

We study emergent communication in a multi-agent reinforcement learning setting, where the agents solve cooperative tasks and have access to a communication channel. The communication channel may consist of either discrete symbols or continuous variables. We introduce an inductive bias to aid with the emergence of good communication protocols for continuous messages, and we look at the effect this type of inductive bias has for continuous and discrete messages in itself or when used in combination with reinforcement learning. We demonstrate that this type of inductive bias has a beneficial effect on the communication protocols learnt in two toy environments, Negotiation and Sequence Guess.

1 Introduction

Although communication in multi-agent reinforcement learning (MARL) may emerge when the agents are given a communication channel that contains no predetermined communication protocol, efficiently doing so often turns out to be a difficult task. As is the case in MARL, the credit assignment problem together with the moving target problem leads to the existence of robust shadowed equilibria in many environments [1]. As a result, when it comes to communication; instead of benefiting from learning a "shared language", the agents may end up disregarding the communication channel altogether and only act upon their own observation.

In order to combat these issues and facilitate the emergence of meaningful communication, that is, communication which increases the expected return over a communication-free baseline, Eccles et al. [2] extended the measures of *positive signaling* and *positive listening* from Lowe et al. [3] to be used as inductive biases. The basic idea is to add additional terms to the loss function which encourage the following: In the case of positive signaling, a speaker is incentivized to produce different messages from different observations. While for positive listening a listener is incentivized to produce different actions from different messages. Overall, this ensures an improved use of the communication bandwidth and a reduced chance of not acting based upon messages. This scheme is explored by Eccles et al. [2] for when the messages passed between the agents are built from discrete symbols.

An orthogonal approach to improving the stability of learning to communicate is to allow for the gradient signal to flow through the communication channel. Doing so shifts the problem from a decentralized towards a centralized training paradigm, which helps at alleviating some of the issues plaguing the former [1]. The cost is that the gradient information needs to be available and passed between the agents while training, which may or may not be viable in a given setting.

In this work, we look at how MARL agents learn to form a "shared language" in order to solve cooperative tasks using two toy examples, a variant of Negotiation [4–7], and a new game we call Sequence Guess. We demonstrate how the positive signaling ideas can be extended to continuous communication protocols, where messages consist of real numbers rather than discrete symbols. We estimate the effect of positive signaling on differentiable communication protocols [8–13], reinforced communication protocols [14, 15], and a combination of both. The effect of a continuous communication protocol is also compared to the effect of a discrete one, in the otherwise discrete game of Sequence Guess.

2 Positive Signaling

We write the total loss function for a communicating MARL agent as $\mathcal{L} = \mathcal{L}_{\text{rest}} + \mathcal{L}_{\text{comm}}$, where the latter term is associated with communication and the former with other actions. The communication loss may be further subdivided into

$$\mathcal{L}_{\text{comm}} = \mathcal{L}_{\text{RC}} + \lambda_{\text{IB}}\mathcal{L}_{\text{IB}} \tag{1}$$

where $\mathcal{L}_{\text{RC}}$ is the loss associated with the communication policy, $\mathcal{L}_{\text{IB}}$ is an inductive bias, and $\lambda_{\text{IB}} > 0$ is a weighting factor. We will focus on $\mathcal{L}_{\text{comm}}$ from here on.

Positive signaling is equivalent to maximizing the entropy of the average message policy, while at the

*Corresponding Author.

Proceedings of the 5th Northern Lights Deep Learning Conference (NLDL), PMLR 233, 2024.
2024 John Isak Fjellvang Villanger & Troels Arnfred Bojesen. This is an open access article distributed under the terms and conditions of the Creative Commons Attribution license (http://creativecommons.org/licenses/by/4.0/).

same time minimizing the entropy of the message policy when conditioned upon a trajectory. We denote the policy for selecting message m given a trajectory x as $\pi(m|x)$, and the average message policy, weighted by frequency, as $\overline{\pi}(m) = \mathbb{E}_{x\sim\boldsymbol{\pi}}[\pi(m|x)]$, where $\boldsymbol{\pi}$ is the overall policy. $\overline{\pi}$ can be estimated from a mini-batch of B trajectories by

$$\overline{\pi}(m) \approx \frac{1}{B}\sum_{b}^{B}\pi(m|x_b), \tag{2}$$

where the subscript b labels the mini-batch members. Then, a natural inductive bias which encourages positive signaling would be

$$\begin{aligned}\mathcal{L}_{\mathrm{IB}}(\pi, x) &= -\mathcal{H}(\overline{\pi}) + \lambda_{\mathrm{PS}}\mathcal{H}(\pi(\cdot|x)) \\ &= \sum_{m\in M}\overline{\pi}(m)\ln(\overline{\pi}(m)) - \lambda_{\mathrm{PS}}\pi(m|x)\ln(\pi(m|x)),\end{aligned} \tag{3}$$

where M is the set of all possible messages and $\lambda_{\mathrm{PS}} > 0$ is a weighting factor.

In practice, however, minimizing the entropy when it is conditioned upon the current trajectory does not work well. One reason for this may be that for any $c < \log(2)$ the space of policies with entropy at most c is disconnected[2], in that the minimal possible entropy during a gradual message policy shift from an old to a new most likely message will at some point have to be at least $\log(2)$, if the entropy of the initial message policy was less than $\log(2)$. Because of this, Eccles et al. instead introduces a finite target entropy $H_{\mathrm{target}} > 0$ for $\mathcal{H}(\pi(\cdot|x))$ and write

$$\mathcal{L}_{\mathrm{IB}}(\pi, x) = -\mathcal{H}(\overline{\pi}) + \lambda_{\mathrm{PS}}(\mathcal{H}(\pi(\cdot|x)) - H_{\mathrm{target}})^2 \tag{4}$$

One can view H_{target} as an exploration parameter, with a larger H_{target} meaning a higher degree of exploration.

2.1 Positive Signaling for Continuous Communication

We take a different approach to positive signaling in the case of the messages consisting of continuous variables rather than discrete symbols. To make the computations viable, a compact support for the messages is needed. A particularly easy set is the n-dimensional torus, where each component of a message lies on an interval of the real line with the end points identified, i.e. a circle. Furthermore, by interpreting the messages as points on a n-torus, we observe that a more uniform average policy may be encouraged by making the messages mutually "repulsive". In practice, this can be achieved stochastically by introducing a pair-wise repulsive "potential" ℓ between the members of a mini-batch of B messages $\boldsymbol{M}$, leading to an inductive bias on the form

$$\mathcal{L}_{\mathrm{IB}}(\boldsymbol{M}) = \frac{1}{B^2}\sum_{\boldsymbol{m},\boldsymbol{m'}\in\boldsymbol{M}}\ell(\boldsymbol{m},\boldsymbol{m'}). \tag{5}$$

Since the messages are continuous, the gradient of the message policy with respect to its weights will be affected by this term as long as we choose an almost everywhere differentiable ℓ. In this work, we have used the simple functional form

$$\ell(\boldsymbol{m},\boldsymbol{m'}) = \max[-\lambda_1 d(\boldsymbol{m},\boldsymbol{m'}) + \lambda_2, 0], \tag{6}$$

where $\lambda_1 > 0$ and $\lambda_2 > 0$ are hyperparameters and d is a distance measure in message space. Assuming that $\boldsymbol{m},\boldsymbol{m'} \in (-1,1)^n$ with -1 and 1 identified as the same point, we define the distance between the two messages to be

$$d(\boldsymbol{m},\boldsymbol{m'}) = \sqrt{\sum_{i=1}^{n}\min(|m_i - m'_i|, 2 - |m_i - m'_i|)^2}, \tag{7}$$

where m_i is the i-th component of $\boldsymbol{m}$. See Fig. 1 for an illustration. The intuition behind Eq. (6) is to penalize two messages that are closer than λ_2/λ_1, and otherwise do nothing. Except for in pathological cases, this cutoff should not affect the inductive bias's push for a more uniform policy, while it does remove any constraining influence on the policy learning when messages are already far apart in message space. After all, what we are mainly interested in is to utilize the entire bandwidth of the message space, not to make the utilization perfectly uniform.

It should be noted that there are multiple ways of achieving a similar outcome as the one described here. Notably, an alternative to Eq. (5) could read

$$\mathcal{L}_{\mathrm{IB}}(\boldsymbol{M}) = \frac{2}{B}\sum_{\substack{\boldsymbol{m}\in\boldsymbol{M}_A\\ \boldsymbol{m'}\in\boldsymbol{M}_B}}\ell(\boldsymbol{m},\boldsymbol{m'}), \tag{8}$$

where $\boldsymbol{M}$ has been equipartitioned into $\boldsymbol{M}_A$ and $\boldsymbol{M}_B$. This reduces the computational complexity of the inductive bias, at the expense of a higher variance. Furthermore, while the functional form of Eq. (6) seems to work well in practice, it is straightforward to construct alternatives.

Since the effect of the latter term in Eq. (4) is to tune the degree of exploration for the message policy when conditioned upon a trajectory, we do not attempt to introduce such a term in the continuous case. Rather, we let the degree of exploration be controlled explicitly by the communication policy.

3 Experiments

The main purpose of the experiments done here is to show how using continuous positive signaling

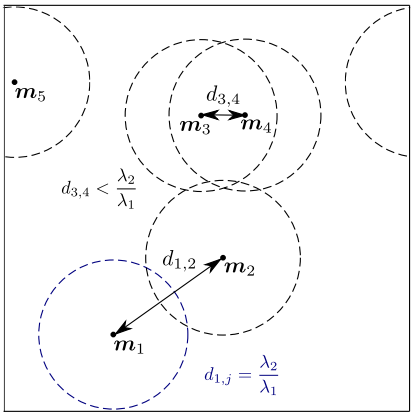

Figure 1. Example of continuous positive signaling where messages contain two components. The distance $d_{i,j} = d(\boldsymbol{m}_i, \boldsymbol{m}_j)$ is calculated between every message. The circles indicate where $d_{i,j} = \lambda_2/\lambda_1$, in other words where message $\boldsymbol{m}_i$ is considered too close to $\boldsymbol{m}_j$. $\boldsymbol{m}_3$ and $\boldsymbol{m}_4$ show a case where $d_{3,4} < \lambda_2/\lambda_1$. $\boldsymbol{m_5}$ illustrates that the space wraps around.

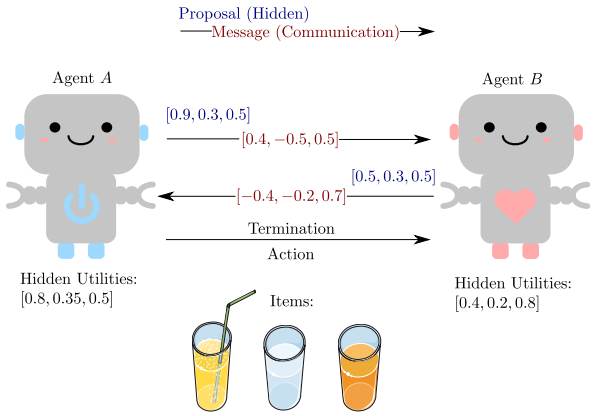

Figure 2. Negotiation. An example run of two agents negotiating over three different types of beverages. Messages have no predefined meaning. The hidden utilities indicate how each beverage is weighted when calculating the reward. A proposal of $[0.9, 0.3, 0.5]$ from agent A would mean agent A receives these proposed fractions of each beverage (here: soda, water and orange juice), while agent B receives the remainder. Agent B can either accept this proposal or come with a counter-proposal. In this example, where agent A accepts B's counter-proposal, the negotiation ends. The agents individual raw reward will be $0.5{\cdot}0.8{+}0.7{\cdot}0.35{+}0.5{\cdot}0.5 \approx 0.95$ and $0.5{\cdot}0.4{+}0.3{\cdot}0.2{+}0.5{\cdot}0.8 \approx 0.58$, which leads to a shared reward of $r_{\text{Neg}} \approx (0.95 + 0.58)/(0.8 + 0.35 + 0.8) \approx 0.78$. The robots have been taken from [19, 20]. The beverages have been taken from [21–23].

for the communication protocol results in improved performance. We demonstrate this using two toy examples, a variant of *Negotiation*, and *Sequence Guess*, which are described below. For the experiments the REINFORCE algorithm [16, 17] has been used, with a policy modeled as a recurrent neural network with a final softmax layer in case of discrete actions, and as a normal distribution with mean and standard deviation parameters given by a neural network when dealing with continuous actions. See appendices Appendices B and C for further details.

Pytorch has been used both to create the environments and the agents [18]. One experiment run on an NVIDIA GeForce RTX 3080 takes about one hour with roughly 40% GPU utilization, this results in an estimated use of $3.2 \cdot 10^{16}$ FLOP during an experiment run.

3.1 Negotiation

Negotiation is a game that consists of two agents, A and B, which, through negotiation, try to distribute a set of k different types of divisible items (e.g. beverages) among themselves. The game is fully cooperative and individual rewards are shared in order to produce a final reward for both agents.

At the beginning of a negotiation round, the agents are individually assigned utilities $\boldsymbol{u} \in (0, 1)^k$ for each of the items, drawn uniformly at random. They receive their own utilities as inputs, but are blind to the utility vector of their partner. The game proceeds in a turn-based manner by the agents alternately making a partitioning proposal $\boldsymbol{p} \in (0, 1)^k$ and passing a message $\boldsymbol{m} \in (-1, 1)^n$ to their partners, until either the partner agrees to the proposal, or a finite time limit T has been reached. Note that the proposals are not explicitly shared, but have to be communicated through the messages, which in turn have to acquire meaning through collaborative learning. In addition to messages and utilities, the agents receive the current turn index t as an input. A proposal agreement in the first turn, i.e. before the first agent has received a message, is ignored. The dynamics is illustrated in Fig. 2 for $k = n = 3$.

At the end of a negotiation round, if the agents have come to an agreement, an individual normalized reward is calculated as

$$r_x = r(\boldsymbol{p}_x; \boldsymbol{u}_x) = \sum_{i=1}^{k} p_{xi} u_{xi}, \tag{9}$$

for $x \in \{A, B\}$, where $p_{Ai} + p_{Bi} = 1$. Note that if agent A is the proposing agent, $\boldsymbol{p}_B$ is a function of the proposal $\boldsymbol{p}_A$, and not an independent proposal. The overall, shared reward is then defined as

$$r_{\text{Neg}} = \frac{r_A + r_B}{r_{\max}}. \tag{10}$$

The scaling factor $r_{\max} = \sum_{i=1}^{k} \max(u_{Ai}, u_{Bi})$ is to ensure that $\max_{\boldsymbol{p}} r_{\text{Neg}} = 1$, regardless of the randomly drawn utilities. If the agents do not come to an agreement before reaching the time limit T, they are punished by $r_{\text{Neg}} = -1$.

3.2 Sequence Guess

We propose a new game, *Sequence Guess*, inspired by the board game *Mastermind*[24]. Sequence Guess is a cooperative, asymmetrical two-player game consisting of a mastermind and a guesser. The goal is for the guesser to as quickly as possible guess a target sequence $\boldsymbol{a} = (a_1, a_2, \ldots, a_k)$ drawn from a finite alphabet Σ, while being guided by the mastermind through messages. Depending on the variant of the game, a message may consist of either a collection of discrete symbols or continuous variables. As with Negotiation, the meaning of a message is not predetermined, but has to be learned.

Each turn of Sequence Guess consists of one sequence guess $\hat{\boldsymbol{a}} \in \Sigma^k$ by the guesser and one reply message from the mastermind: The guesser receives the latest message from the mastermind (initially a constant) as well as the current turn number t, and returns a guess. Then the mastermind replies with a new message based on the correct sequence, the guess, and the current turn number. If the guess is the same as the target sequence, or $t = T$, the game terminates. Fig. 3 shows an excerpt of a possible realization of Sequence Guess.

When a game of Sequence Guess ends, the agents are rewarded according to the fraction of letters in the guess sequence being equal to the corresponding letters in the target sequence, as well as a time penalty to encourage fast solutions:

$$r_{\text{Seq}} = -0.1t + \frac{1}{k}\sum_{i=1}^{k} \mathbf{1}_{a_i = \hat{a}_i}. \tag{11}$$

Sequence Guess differs from Negotiation in a few key aspects. Negotiation is symmetric; each agent is given the same type of information, and the game is "dialogue focused". Sequence Guess, on the other hand, is "monologue focused", isolating the tasks of "formulating" and "understanding" messages to separate agents. In Negotiation it is possible for the agents to learn a policy significantly better than the non-repeated random policy even with the communication channel removed. This is not possible in Sequence Guess. Thus, Sequence Guess cannot suffer from shadowed equilibria.

3.3 Results

To demonstrate the effects of positive signaling, we set up a number of Negotiation and Sequence Guess ablation experiments, both when interagent gradients are allowed to flow and not. We focus on three cases: when the communication policy is learned using only a standard reinforcement learning approach without an inductive bias (RC), when it is obtained through minimizing the positive signaling inductive bias (PS), and when a combination of these two methods is used. The results of these experiments are shown in Fig. 4 and summarized in Table 1. Note that the average returns of the Sequence Guess experiments have been shifted by a factor of $1 - (1 \cdot 1/27 + 0.9 \cdot 26/27) \approx 0.0963$ [1] to ensure that the expected return under an optimal policy is 1.

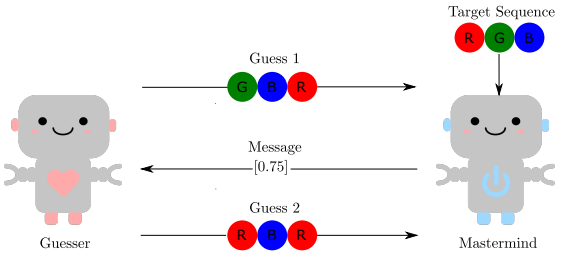

Figure 3. An excerpt of Sequence Guess. The guesser attempts to guess some target sequence, while the mastermind tries to provide information about the target sequence to the guesser. Here, the alphabet size is 3 and the target sequence length is 3, while messages consist of one real number. The robot figures are from [19, 20].

We observe that including a positive signaling learning bias typically improves the learning, as does interagent gradients. Moreover, continuous communication protocols outperform discrete ones. These points are not surprising, given that continuous messages and gradients carry more information compared to only communicating discrete messages.

One should, however, keep in mind that the positive signaling learning bias is different when using continuous and discrete messages.

Note that in Fig. 4(d) using only the positive signaling learning bias performs best. This may be due to the the fact that in this case all the information can be communicated in one message, making it possible for PS in and of itself to push the learning towards an optimal bandwidth usage. In this case, adding an extra RC component will mostly provide detrimental noise. When the information cannot be communicated in a single message, this may not apply; then, a combination of PS and RC may perform better, as RC may help prioritize the most important information. Why this does not happen in Fig. 4(c) is not entirely clear to us, but we suspect that without a gradient to couple the learning of sending and receiving messages, local minima may more easily be encountered. In this case, an RC contribution may provide a helpful learning signal.

Without communication, the maximum possible expected return for Sequence Guess is $1/9 \approx 0.11$, while for Negotiation simulation experiments show that it is ~ 0.92. This explains why the separation between the different Negotiation experiments (Fig. 4(a,b)) is smaller than for the corresponding

[1] The chance for the initial guess to be correct is 1/27 which gives a reward of 1, under an optimal policy the reward will be 0.9 if the initial guess is incorrect.

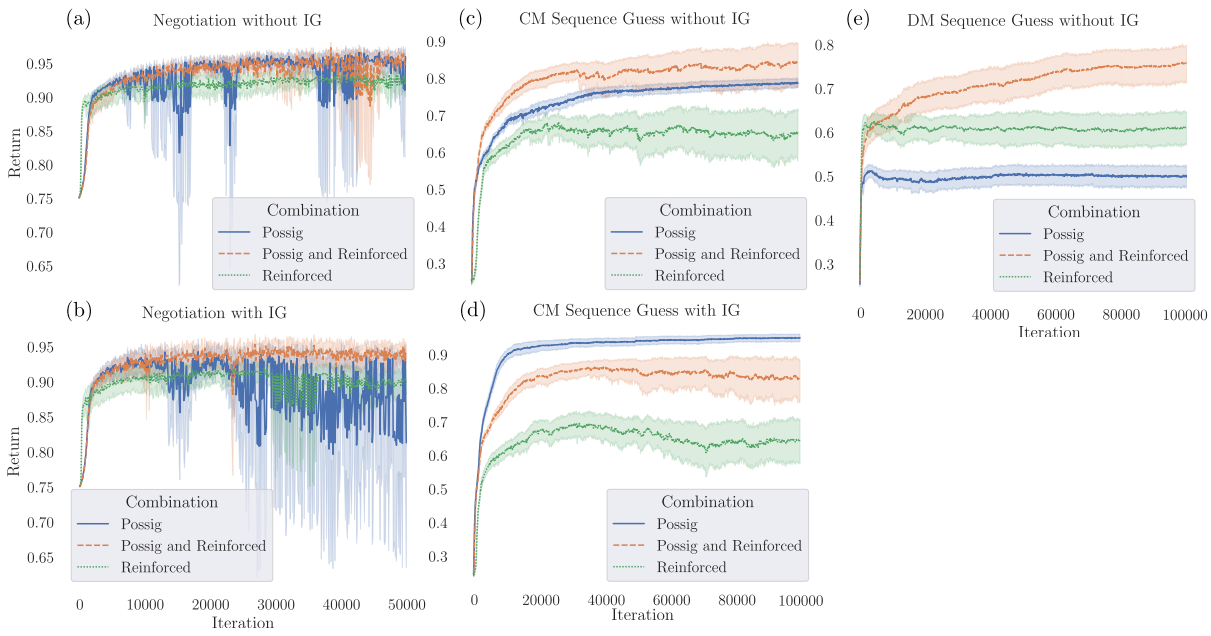

Figure 4. Summary of 30 independent experiment runs for each game and loss function combination investigated. Each sample is an average of a mini-batch of size 2048. The lines indicate the means, while the bands denote 95% confidence intervals. The columns display the different games: Negotiation, continuous message (CM) Sequence Guess, and discrete message (DM) Sequence Guess. The rows are organized with respect to whether interagent gradients (IG) are allowed to flow or not. The returns have been scaled to ensure that their maximal expectation values are one. See the text for more details.

Sequence Guess experiments (Fig. 4(c-e)), assuming that it is easier to learn a non-communicative policy,.

The discrepancy between Table 1 and Fig. 4 for Negotiation is due to averaged out oscillations; it seems that the learning is unstable with respect to whether the communication protocol is beneficial.

4 Discussion and Summary

In this work, we have shown that it is possible to generalize the idea of positive signaling as an inductive bias[2] for learning communication with continuous messages. The advantages of introducing such an inductive bias to the loss function has been demonstrated on two toy MARL environments, Negotiation and Sequence Guess.

Positive Signaling "encourages" the agents to learn to utilize the entire bandwidth of the communication protocol without getting stuck in local minima where two input states result in the same message. This is similar to what is done in variational autoencoders[11], where the average encoding is forced to follow a predefined distribution (typically a Gaussian), regularizing the learning space.

Continuous message protocols have some intrinsic advantages over discrete message protocols when it comes to learning: a smooth optimization landscape is usually preferable when dealing with policies modeled by neural networks. In addition, there is a practical limit on how large the communication alphabets can be, since their one-hot encoding grows linearly with the alphabet size. Furthermore, computing a positive signaling bias through "repelling" messages in a continuous message space is easier and more robust than estimating entropies based on mini-batches of discrete messages.

It is noteworthy that continuous message protocols seem to generally perform better than discrete message protocols, even when the desired information is discrete; compare Fig. 4(e) with Figs. 4(c, d).

If the set of possible states is smaller than the set of possible messages, loss-less communication is achievable and can be encouraged by using positive signaling. If the contrary holds, the sending agent can prioritize by adding RC. In practice, using only RC is likely not desirable, since the agents can get stuck in a local optimum where the same message is sent in two different states; most attempts at deviating from such a communication policy is likely to lead to a worse performance, driving the agents back to their original suboptimal policies.

Regarding future work, we believe from a linguistic or cognitive perspective it could be interesting to look more into visualizing the message policies that are being learnt. Another interesting avenue is looking at environments for more than two agents with no reward sharing, and see if some form of communication or cooperation can more easily arise in such games with the use of positive signaling.

Table 1. Summary of 30 independent experiment runs, each with a batch size of 2048. The table shows the average returns of the best mini-batches with 95% confidence intervals, i.e. the expected highest return. *IG*, *CM*, and *DM* refers to *Interagent Gradients*, *Continuous Messages*, and *Discrete Messages*, respectively.

		RC	PS	RC and PS
Negotiation	IG	0.942 ± 0.009	0.976 ± 0.010	0.979 ± 0.008
	No IG	0.951 ± 0.008	0.987 ± 0.002	0.986 ± 0.003
CM Sequence Guess	IG	0.780 ± 0.034	0.958 ± 0.010	0.921 ± 0.020
	No IG	0.778 ± 0.029	0.803 ± 0.011	0.897 ± 0.024
DM Sequence Guess	No IG	0.693 ± 0.020	0.551 ± 0.014	0.782 ± 0.035

5 Acknowledgements

We would like to thank the Department of Informatics, University of Bergen, for financial support while completing this work.

References

[1] S. Gronauer and K. Diepold. "Multi-agent deep reinforcement learning: a survey". In: *Artificial Intelligence Review* 55.2 (2022), pp. 895–943.

[2] T. Eccles, Y. Bachrach, G. Lever, A. Lazaridou, and T. Graepel. "Biases for emergent communication in multi-agent reinforcement learning". In: *Advances in neural information processing systems* 32 (2019).

[3] R. Lowe, J. Foerster, Y.-L. Boureau, J. Pineau, and Y. Dauphin. "On the pitfalls of measuring emergent communication". In: *arXiv preprint arXiv:1903.05168* (2019).

[4] D. Devault, J. Mell, and J. Gratch. "Toward Natural Turn-Taking in a Virtual Human Negotiation Agent". In: Mar. 2015.

[5] M. Lewis, D. Yarats, Y. Dauphin, D. Parikh, and D. Batra. "Deal or No Deal? End-to-End Learning of Negotiation Dialogues". In: *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sept. 2017, pp. 2443–2453. DOI: 10.18653/v1/D17-1259. URL: https://aclanthology.org/D17-1259.

[6] K. Cao, A. Lazaridou, M. Lanctot, J. Z. Leibo, K. Tuyls, and S. Clark. "Emergent communication through negotiation". In: *arXiv preprint arXiv:1804.03980* (2018).

[7] H. He, D. Chen, A. Balakrishnan, and P. Liang. "Decoupling strategy and generation in negotiation dialogues". In: *arXiv preprint arXiv:1808.09637* (2018).

[8] S. Sukhbaatar, R. Fergus, et al. "Learning multiagent communication with backpropagation". In: *Advances in neural information processing systems* 29 (2016).

[9] P. Peng, Y. Wen, Y. Yang, Q. Yuan, Z. Tang, H. Long, and J. Wang. "Multiagent bidirectionally-coordinated nets: Emergence of human-level coordination in learning to play starcraft combat games". In: *arXiv preprint arXiv:1703.10069* (2017).

[10] J. Jiang, C. Dun, T. Huang, and Z. Lu. "Graph convolutional reinforcement learning". In: *arXiv preprint arXiv:1810.09202* (2018).

[11] T. Lin, J. Huh, C. Stauffer, S. N. Lim, and P. Isola. "Learning to ground multi-agent communication with autoencoders". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 15230–15242.

[12] T. Lin, J. Huh, C. Stauffer, S. N. Lim, and P. Isola. "Learning to ground multi-agent communication with autoencoders". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 15230–15242.

[13] C. Zhu, M. Dastani, and S. Wang. "A Survey of Multi-Agent Reinforcement Learning with Communication". In: *arXiv preprint arXiv:2203.08975* (2022).

[14] J. Foerster, I. A. Assael, N. de Freitas, and S. Whiteson. "Learning to Communicate with Deep Multi-Agent Reinforcement Learning". In: *Advances in Neural Information Processing Systems*. Ed. by D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett. Vol. 29. Curran Associates, Inc., 2016. URL: https://proceedings.neurips.cc/paper/2016/file/c7635bfd99248a2cdef8249ef7bfbef4-Paper.pdf.

[15] A. Das, T. Gervet, J. Romoff, D. Batra, D. Parikh, M. Rabbat, and J. Pineau. "Tarmac: Targeted multi-agent communication". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 1538–1546.

[16] R. J. Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Reinforcement learning* (1992), pp. 5–32.

[17] R. S. Sutton and A. G. Barto. "Reinforcement Learning: An Introduction". In: Second. The MIT Press, 2018, pp. 324–332, 115. URL: http://incompleteideas.net/book/the-book-2nd.html.

[18] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. "Pytorch: An imperative style, high-performance deep learning library". In: *Advances in neural information processing systems* 32 (2019).

[19] Mvolz. *Kawaii robot power clipart*. https://commons.wikimedia.org/wiki/File:Kawaii_robot_power_clipart.svg. Seen: 05/08/2022. License: Creative Commons CC0 1.0 Universal Public Domain Dedication. 2019.

[20] Mvolz. *Kawaii robot with heart clipart*. https://commons.wikimedia.org/wiki/File:Kawaii_robot_power_clipart.svg. Seen: 05/08/2022. License: Creative Commons CC0 1.0 Universal Public Domain Dedication. 2019.

[21] smart.servier.com. *ORANGE JUICE*. https://smart.servier.com/smart_image/orange-juice/. Seen: 05/08/2022. License: Creative Commons Attribution 3.0 Unported. 2016.

[22] smart.servier.com. *WATER*. https://smart.servier.com/smart_image/water/. Seen: 05/08/2022. License: Creative Commons Attribution 3.0 Unported. 2016.

[23] smart.servier.com. *SODA*. https://smart.servier.com/smart_image/soda-2/. Seen: 05/08/2022. License: Creative Commons Attribution 3.0 Unported. 2016.

[24] E. Weisstein. *Mastermind*. From MathWorld–A Wolfram Web Resource. https://mathworld.wolfram.com/Mastermind.html. Seen: 03/02/2023. 2023.

[25] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[26] J. Jiang and Z. Lu. "Learning attentional communication for multi-agent cooperation". In: *Advances in neural information processing systems* 31 (2018).

A Example PyTorch Implementation of Continuous Positive Signaling

This example code illustrates an example implementation of continuous positive signaling where the the distances between all neighbours are used for each message to calculate the loss.

```
def positive_signalling_loss(self, means):
    means = means.repeat(len(means), 1, 1)
    # means has dimensions of length [Batch
    ↪  Size, Batch Size, Sequence Length]
    means_2 =
    ↪  torch.transpose(means.clone().detach(),
    ↪  0, 1)
    distances = torch.abs(means - means_2)
    delta = torch.pow(torch.minimum(distances, 2
    ↪  - distances) + self.epsilon, 2)
    delta = torch.sum(delta, dim=2)
    delta = torch.triu(delta, diagonal=1)
    delta[delta == 0] = torch.inf
    delta = torch.sqrt(delta)
    zeroes = torch.zeros(delta.shape,
    ↪  device=device)
    loss = torch.mean(torch.maximum(-lambda_1 *
    ↪  delta + lambda_2, zeroes))
    return loss
```

B Network Architecture, Algorithm and Hyperparameters for Negotiation

B.1 Algorithm

For Negotiation the REINFORCE algorithm is used with a parameterized baseline. The ADAM optimizer[25] is used for both the baseline and the policy. The baseline is centralized and receives the same input as the agent whose turn it is, learning was observed to be unstable without the use of a parameterized baseline.

B.2 Network Architecture

The network architecture for the agents and the baseline is the same unless otherwise stated:

1. Input: Hidden utilities, current message, current turn/max turns
2. LSTM layer with output size 100, Hidden state persists across one game of Negotiation.
3. Fully Connected layer with output size 100 and Leaky ReLU activation
4. Fully Connected layer with output size 1 and a *tanh* activation for the baseline in order to produce the return prediction, for the policy the output size is 13.

Hyperparameter	Value
Batch Size	2048
Iterations	50 000
LSTM Layer Size	100
Hidden Layer Size	100
Max Turns	6
λ_1	250
λ_2	10
Gradient Clip Value	1
β_1 (Decay Rate, Adam)	0.9
β_2 (Decay Rate, Adam)	0.999
Weight Decay	0.0001

Table B.1. Hyperparameters used in Negotiation, the same layer sizes are used for the agents and the parameterized baseline. We selected values for λ_1 and λ_2 which caused the magnitude of the PS loss the be similar to that of the REINFORCE loss.

5. For the policy 6 output variables are used as means $\boldsymbol{\mu}$, 6 are used as standard deviations $\sigma(\boldsymbol{s})$, where σ is the sigmoid activation function. $\boldsymbol{\mu}$ and $\sigma(\boldsymbol{s})$ are then used to initialize six normal distributions. Three normal distributions are used to sample the proposal $\sigma(\hat{\boldsymbol{y_1}})$. Three normal distributions are used to create the message $\tanh(\hat{\boldsymbol{y_2}})$. The remaining output is used to determine probability of termination $\sigma(\hat{y_3})$ from which the termination action is sampled.

The learning rate α begins at 0.001 for both the baseline and the policies. When an iteration with $r \geq 0.9$ is reached, the learning rate is reduced to 0.0001 for both the baseline and the policies. $r \geq 0.9$ indicates a close to optimal joint policy if the agents only know their own hidden utilities. The reason for the learning rate adjustment is that the learning can very easily become unstable with a learning rate of 0.001, while learning would be very slow if it began with a learning rate of 0.0001. Table B.1 shows the hyperparameters used. The hidden layer sizes, message sizes and number of item categories are the same ones that Cao et al. [6] used for their implementation. The choice of a ¨large batch" size is based upon Jiang and Lu [26] who noted that a large batch size helped accelerate their learning process in a MARL setting.

The network also outputs the probability of termination from a sigmoid activation, and the termination action is sampled from this probability.

C Network Architecture, Algorithm and Hyperparameters for Sequence Guess

C.1 Algorithm

The REINFORCE algorithm with a baseline and the ADAM[25] optimizer is used. A moving average is used as the baseline. This is how the baseline is calculated at iteration $t+1$:

$$b_{t+1} = 0.7b_t + 0.3G_t \tag{12}$$

Where G_t is the mean return over the entire minibatch for the current iteration. This is the same baseline that Cao et al. [6] used. Since the stability issues observed in *Continuous Negotiation* has not been observed in this game, a more simple baseline should suffice.

C.2 Network Architecture

C.2.1 Discrete messages

Both the guesser and mastermind:

1. Input Guesser: Current Turn, Last Message. Input Mastermind: Current turn, Last Guess, Target Sequence.

2. An encoder-decoder using LSTMs, a one hot encoding of the current turn is appended to the hidden state. Refer to Fig. C.1 for more details.

3. Fully Connected layer of input size 100 and output size 3, one for each letter, with softmax activation over final dimension.

4. Initialize $B \cdot k$ categorical distributions with $|\Sigma|$ categories, where B is the batch size, k is the sequence length and $|\Sigma|$ is the alphabet size. We then sample the guesses or messages for the Guesser and Mastermind respectively.

C.2.2 Continuous messages

For continuous messages the architecture is the same except for that mastermind employs only the encoder part, where the final hidden state is used in a fully connected layer of size 100, then a ReLU activation and another fully connected layer, the output from this layer is used in the same manner as in Negotiation in order to generate the message. The guesser employs only the decoder part for continuous messages, as messages are of the same type as those used for Negotiation.

Table C.1 shows the hyperparameters used in Sequence Guess.

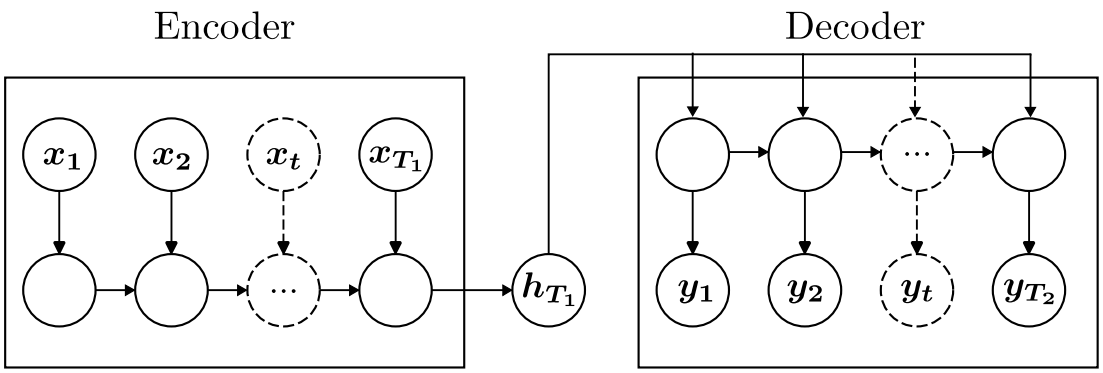

Figure C.1. The encoder-decoder architecture used for DM Sequence Guess. *In the case of the mastermind*, input x_t will contain symbol number t from the guess sequence and target sequence, T_1 will be the length of the target sequence and T_2 will be the length of the message sequence. The output y_t is used in a fully connected layer with a Softmax activation function in order to find message symbol number t. *In the case of the guesser*, input x_t will contain symbol number t from the message, T_1 will be the length of the message sequence and T_2 will be the length of the target sequence. y_t is used in a fully connected layer with a Softmax activation function in order to find guess symbol number t. In both cases, a one-hot encoding of the current turn is appended to the final hidden state of the Encoder in order to produce the context vector.

Hyperparameter	Value
Batch Size	2048
Learning Rate	0.001
Iterations	100 000
Encoder Size	100
Decoder Size	100 + n turns
Hidden Layer Size	100
λ_1	100
λ_2	10
λ_{PS}	1
λ_{IB}	1
β_1 (Decay Rate, Adam)	0.9
β_2 (Decay Rate, Adam)	0.999
weight decay	0.0001 if CM 0 if DM

Table C.1. Hyperparameters used in Sequence Guess. The same hyperparameters are used for the guesser and the mastermind. The reasoning behind the choice of hyperparameters is they are of the same magnitude as the ones used by Cao et al. [6]. Note that weight decay with DM fails to produce any convergence, the most likely reason beings that the gradient signal in the initial correlations giving rise to communication is too weak compared to the gradient signal of weight decay. We selected values for λ_1 and λ_2 which caused the magnitude of the PS loss the be similar to that of the REINFORCE loss.