AI to AI Communication

Author
Jordi Puig Rabat
jordi.puig.rabat@estudiantat.upc.edu

Advisor
Luis Velasco Esteban

Advisor
Marc Ruiz-Ramírez

Abstract

In this work, we present a study of a method for learning communication protocols in multi agent cooperative tasks. Differential Inter-Agent Learning (DIAL) is one powerful method for learning such communication protocols, here we study the performance of this method in the Multi-Step MNIST game. We evaluate the sensitivity of the discount factor parameter and the standard deviation of the noise added to the communication. Moreover, we extended the game to a version of three players and found that DIAL fails to learn with the provided setting. We also found contradictions in our results with respect to other works and extend the discussion of this method.

1. Introduction

In some Machine Learning tasks where more than one agent has to cooperate to achieve a shared goal among all agents, establishing a communication protocol is a key step towards solving such tasks.

In this paper, we study a reinforcement learning method proposed by Forster et al. [2] that can be used in settings where agents have to learn to communicate in order to solve a given task. We will focus on reproducing some of the results and extend the discussion on the methods proposed. Moreover, modifications and simplifications will be introduced.

In the original work, the method uses Deep Q-Learning [4] and Recurrent Neural Networks to learn both the communication protocol and the Q function. They also propose some baseline tasks or games to test the methods they present, namely: Switch Riddle, Colour-Digit MNIST game and Multi-Step MNIST game. They present different methods to learn communication protocols: Reinforcement Inter-Agent Learning (RIAL) and Differentiable Inter-Agent Learning (DIAL). Specifically, we will center our study on the DIAL method which has the higher potential as it takes advantage of the ability to compute gradients through the communications exchanged. In their work they study which strategy works better: sharing the parameters of the network among agents or not sharing them; here we will be working with shared parameters as this strategy obtains the best results.

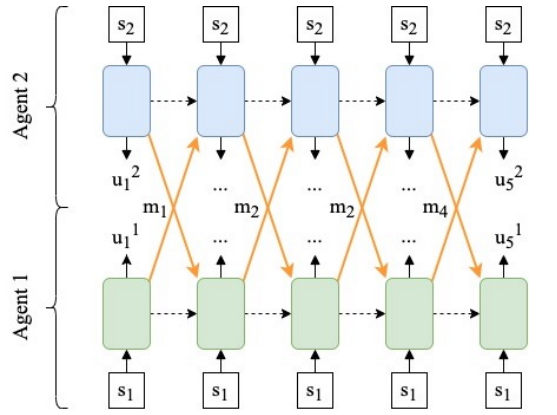
Aside from working with the mutli-agent setting with a single dimensional action space, as one modification will consist in adding a player to the game, we will extend the method to a higher dimensional space of actions while maintaining the communication methods. To this end, we will make use of some methods developed by Tavakoli et al. [6], specifically the loss of the network will be computed as the sum of square losses for each individual action.

All the code developed by Foerster et al. [1] was published in a single repository using Lua language; moreover, a version of the prisoners switch riddle implementation in Python can be found in a public repository [3]. The code we developed to execute the experiments was heavily inspired by this work although lots of modifications were needed to adapt the methods to the Multi-Step MNIST game.

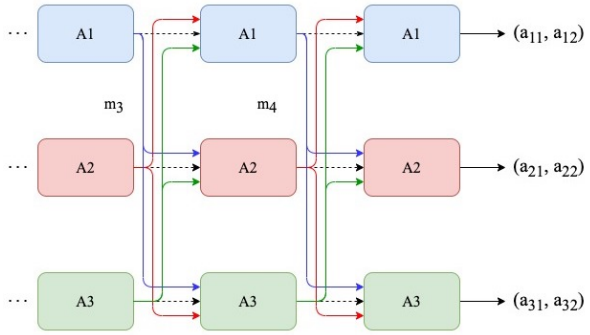
2. Use case

In this paper we will be working with the Multi-Step MNIST game, a two-player game of guessing numbers; more precisely, a simplified version of the game. While in the original game the agents received a MNIST image representing a digit (and then they had to recognize it), in our work we will give the agents a digit identifier which will be directly converted to an embedding. After that, each agent has to communicate this digit to the other agent in a sequence of four binary digits. This happens within a sequence of five steps where at each step, the agent can only pass one binary bit of information to the other and perform an action that does not have effect on the environment. At the fifth step, the action of the agents is considered as the guess of the agent and a reward is given for each digit well guessed. In figure 1a we have a diagram of the mechanics of the game.

Moreover, we introduce a modification to the original game aside from the simplification mentioned above. As we are interested in the communication among various agents, we will add a third player to the game. The mechanics will be very similar: all players receive a digit, during the first 4



(a) The diagram of the game with 2 agents.



(b) The diagram of the game with 3 agents; for simplicity reasons, we only show the last three steps and the node representing the state received is not drawn although it works the same way as in the version with two players: an agent receives the same digit during a single episode.

Figure 1: Diagrams of the game mechanics.

steps they can send a single bit of information to the other two agents (they send the same digit to each of the other agents) and in the final step they perform two independent actions to guess the digit the other agents had. For each digit well guessed they will receive a shared reward. It is relevant to note that the agents know which bit was sent by whom.

3. Architecture

In this section, the network architecture of the agents will be explained. The method for learning the communication protocol that will be analyzed is DIAL. Nevertheless, we made a couple of modifications with respect the original architecture shown in the original paper: first, as we simplified the state space and don't have images to represent the digits, we only use a simple Embedding layer that maps each identifier to a single embedding; the second modification is made on the recurrent modules, while Gated Recurrent Units were used originally, we use simple recurrent layers.

In figure 2 we display a diagram showing how the agent network architecture works. The input is composed by the sum of four embeddings: the digit, the incoming communication, the previous action, and the agent ID. All of this embeddings are obtained with an identifier except for the communication that is obtained with a fully connected Single Layer Perceptron, the architecture of which is also described in the original work and consist of a batch normalization layer, a linear layer and a ReLU activation function. This architectural trick (plus the DRU) enables the network to differentiate with respect to this weights and the communication received during training and actually behaves like a simple embedding during test. These embeddings with dimension 128 are then passed to a Recurrent Neural Network with two layers, the hidden size is again 128. The final step is a Feed Forward Network, in this work we used a fully connected network consisting of two linear layers and ReLU activation of hidden size 128 and output size $|Q| + 2$ (i.e. the number of actions plus a neuron for each possible bit to send).

The output of the network has two components: the Q values for choosing the action and the communication that the agent will emit. These values correspond to a specific step and a specific agent (note that in the diagram in figure 2 the output has the corresponding sub indices).

The key difference between DIAL and RIAL is the discretise/regularise unit (DRU) which allows to backpropagate the gradients through the communication the agents exchange. If we take a look at a lower level of implementation, the network of the agent has two neurons to decide the communication. During training, when the value of both neurons is passed to the DRU, it adds gaussian noise with a given standard deviation σ. The effect this hyperparameter has over the reward curve will be studied in this work. After adding this noise, the values are passed to the other agent (this allows for backpropagating through these values). On the other hand, during test, the output of the neurons is passed to the DRU and it discretizes them, meaning it selects both values and returns a one and a zero, being the one at the place where the higher value was.

This architecture is not altered in the case of more than three agents except for the communication embedding layer. The changes affect the input size (from 2 to 4) but the expected behaviour is the same.

4. Results

All the experiments were made in the simplified version of the game where digit identifiers are passed directly to the agents instead of the MNIST image. Thus, with 20.000 epochs, we could determine whether the agents succeeded at learning to communicate. This number of epochs is smaller than the one from the results presented in the original work (they execute 50.000 epochs) because as the set-

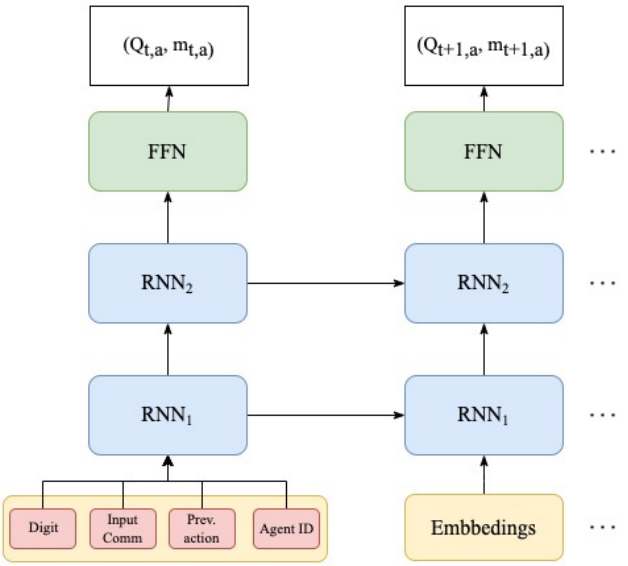


Figure 2: Block diagram of the architecture used. The diagram shows two steps of a single agent activity while in the game we will always have 5 steps.

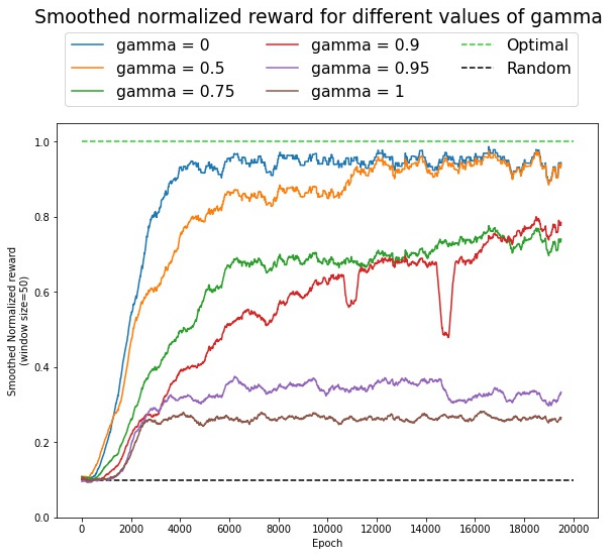


Figure 3: Normalized reward evolution averaged over 2 trials for each value of gamma. Smoothed with a window of size 50.

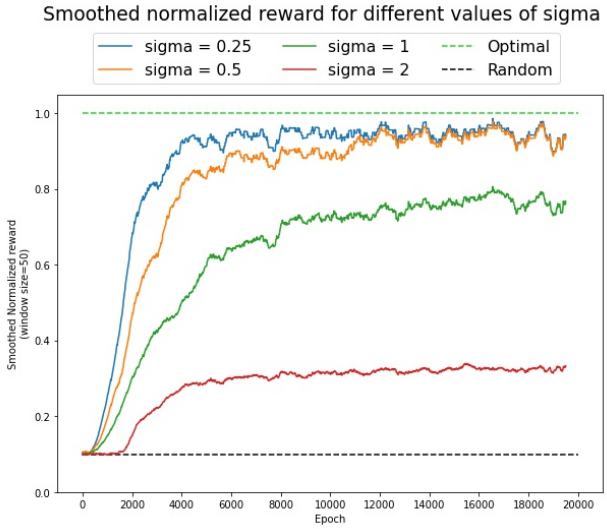


Figure 4: Normalized reward evolution averaged over 5 trials for different values of sigma. Smoothed with a window of size 50.

ting is simpler, the learning is faster. The tests were made with the DIAL architecture and simple Recurrent Neural Networks as stated in the previous section; moreover, the parameters of the network were shared among the agents. The value of other parameters like the optimizer and its learning rate, the epsilon for choosing random actions, and the target network actualization frequency are set to match the values used in the original work. The parameters that were studied by making the experiments for different values were the standard deviation of the noise added in the DRU σ and the discount factor γ. Here we present the evolution of the normalized reward function (i.e. the total reward in a batch divided by the size of the batch) for the different experiments made, which indicates whether the agents have learned to communicate, and therefore solved the task. A reward of 1 means they have guessed all numbers (this is forced via the magnitude of the reward).

We found that the effect the gamma parameter has over the learning curve is relevant. It can be observed in figure 3 that lower values for this parameter yield optimal performance while values close to 1 cannot achieve the optimal performance. This result contrasts with the ones presented in the original work, where they achieve optimal performance with a value of 1.

In figure 4 we observe that for lower values of sigma, the agents can almost achieve optimal performance in the task while with higher values, the maximum mean reward achieved is around 0.3. In this case, nonetheless, the agents do learn something because the reward obtained is higher than the random baseline.

4.1. Tests with 3 agents

The experiments done with 3 agents were performed just like the experiments with 2, although only the gamma parameter was tested in this case. The sigma was fixed at $\sigma = 0.5$.

We noticed that just like in the Switch Riddle game,

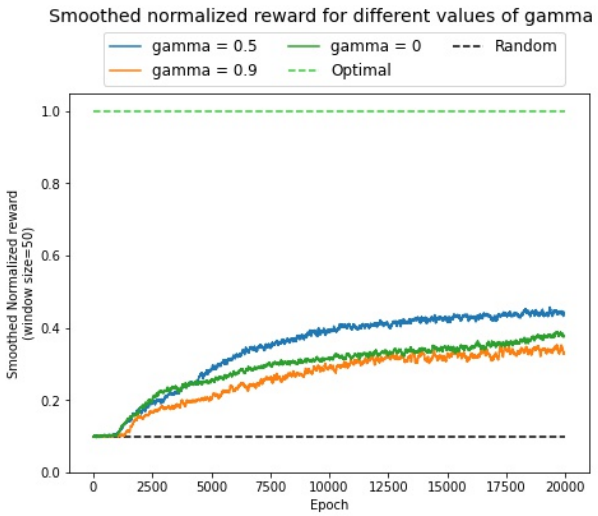


Figure 5: Normalized reward evolution averaged over 5 trials for each value of gamma. Smoothed with a window of size 50.

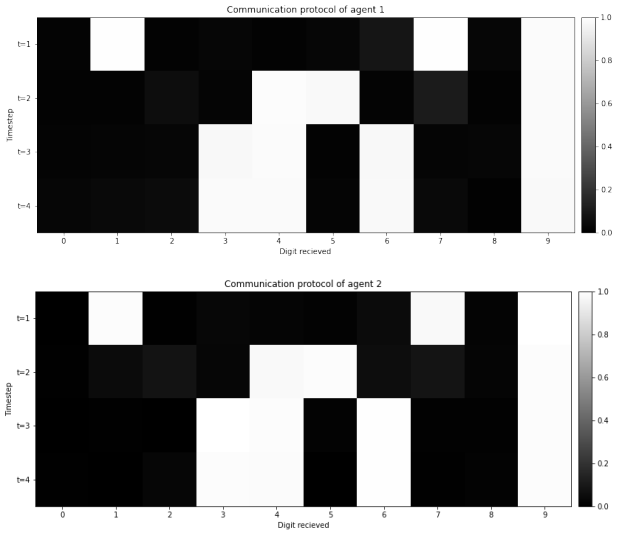


Figure 6: Average value of the i-th bit passed by each agent at each timestep. The results were obtained when the agents shared the parameters.

adding more agents makes the task harder, thus our methods perform worse. In this scenario, the agents fail to learn the optimal behaviour. Nevertheless, they seem to learn something; the effect of gamma is the same as in the case with 2 agents: as gamma decreases, the performance gets better although it does not reach the optimal.

4.2. Interpretability of the communication protocol

In order to understand how the agents are communicating with each other, we executed 3200 test episodes after training and recorded the communications of the agents. Then we averaged the bit sent at each step, grouped by the digit received and the agent. This experiment was made with the same setting as explained above, and the gamma value was 0 and the sigma was $\sigma = 0.5$. This average can be considered the representation of the digit in their protocol. In figure 6 we observe that both agents follow the same protocol in average. The reason could be easily linked to the fact that they share the parameters. Another insightful observation is that the representation for each digit is not unique for all digits. In figure 6 it can be checked that there are three groups that share a representation. Having this result and an accuracy close to 1, we are led to think that the communication protocol for this pairs also depends on what the other agent is sending at the time; else wise, there is no explanation to why they achieve almost perfect performance with a protocol with so many repeated values.

Although we only present the interpretation of one protocol obtained in a particular learning environment, we have obtained different protocols with different configurations of the parameters or different seed for the randomness. The other protocols were very similar to the one presented as there were some digits with the same exact representation. This results also contrast with the ones obtained in the original work, where they show that each different digit has a different representation.

The code developed to obtain these results can be found in a GitHub public repository [5].

5. Discussion

The aim of this paper is to extend the discussion of the methods studied and compare the results of the original paper with the ones obtained in our experiments. The first thing we address is the simplifications made in both the setting and the architecture. As for the state space (giving IDs instead of images), we consider that the results and the conclusions we provide to be valid as long as the agent is able to embed the state information in a stable way (similar states are mapped to similar embeddings). In real applications, this task could be detached from the communication learning using Auto Encoders or other techniques to learn a representation of the state in a smaller dimensional space. This pre-trained module can be then added to the agents architecture freezing or not the weights.

An explanation to why simpler recurrent units work better could simply be that the task does not require that much power, having too many weights to tune results in a harder task that the agents cannot solve. Nevertheless, this aspect hasn't been deeply explored; although some tests with different hidden sizes were performed, the results are not presented.

The main focus of this discussion will be in the sensitivity of the discount factor (gamma) and the standard deviation of the noise (sigma). While in the original paper they use a gamma value of $\gamma = 1$ for all their tests, resulting in optimal performance in some cases, in this work we have seen the opposite: smaller discounted factors lead to higher rewards. The explanation behind this could be related to the fact that the state does not change along one episode, this implies that the four first action doesn't have a clear meaning nor utility, and in the cases with high values of gamma, the loss function carries a lot of noise that comes from the terms of the first actions loss. Nevertheless, an explanation to why this difference arises could not be found.

The behaviour of the sigma parameter was found similar to the one presented in the original work. While they solve the task of the switch riddle (and the color-digit MNIST game) using a sigma $\sigma = 2$, the multi-step MNIST game is solved with a sigma value of $\sigma = 0.5$. The reason they add the noise in the first place is to lead the activations of the neurons to have a discrete behaviour; thus, as we increase the standard deviation of the noise, we will be forcing this discretization of the activation values[1]. Nevertheless, we found out that smaller values can have slightly faster learning curves; the interpretation of this result couldn't be provided due to the lack of time, and many interpretations are possible.

The last insight we want to provide is the effect of adding more agents to the game. As we have seen, when we play the game with three agents, reaching the optimum reward in 20.000 epochs is not possible. This behaviour was also present in the original work where they test the switch riddle with 3 and 4 agents; while the experiments with 3 agents lead to optimal behaviour, the case with 4 agents does not reach the optimal. Future work could be aimed to determine whether the DIAL learning method is suited for tasks with a greater number of agents (more than three). Nevertheless, the task in which a great number of agents have to cooperate should have different characteristics from the MNIST games. One line that has not been explored is to combine more agents but maintain a task where the information of all agents has to be combined in order to achieve the higher reward. One example could be a game where the agents have to sum all their numbers in order to obtain the reward; although this task shouldn't be solved with Q learning, it serves as an example of a task where combining information from different agents is key to solve such task.

[1] The reasoning behind can be found in the work from Foerster et al. [2]

References

[1] Jakob N. Foerster and Yannis M. Assael. Learning to communicate with deep multi-agent reinforcement learning. `https://github.com/iassael/learning-to-communicate`, 2016.

[2] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, page 2137–2145, 2016.

[3] Jiang Minqi. Learning to communicate with deep multi-agent reinforcement learning. `https://github.com/minqi/learning-to-communicate-pytorch`, 2018.

[4] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, Feb. 2015.

[5] Jordi Puig Rabat. Mnist games. `https://github.com/jordipuig37/mnist-games`, 2022.

[6] Arash Tavakoli, Fabio Pardo, and Petar Kormushev. Action branching architectures for deep reinforcement learning, 2019.