



Communication-based MARL

Seminar on Basic MARL, Session 3

Dan Qiao
Daoyuan Building, Room 423
August 23, 2022

- Abstract
- Problem Settings
- Methods
 - RIAL
 - DIAL
 - Model Architecture
- Experiments
 - Switch Riddles
 - MNIST Game
- Discussion
- Code Implementation (refer to Git)

Learning to Communicate with Deep Multi-Agent Reinforcement Learning

Jakob N. Foerster^{1,†}
jakob.foerster@cs.ox.ac.uk

Yannis M. Assael^{1,†}
yannis.assael@cs.ox.ac.uk

Nando de Freitas^{1,2,3}
nandodef Freitas@google.com

Shimon Whiteson¹
shimon.whiteson@cs.ox.ac.uk

¹University of Oxford, United Kingdom

²Canadian Institute for Advanced Research, CIFAR NCAP Program

³Google DeepMind

Abstract

We consider the problem of multiple agents sensing and acting in environments with the goal of maximising their shared utility. In these environments, agents must learn communication protocols in order to share information that is needed to solve the tasks. By embracing deep neural networks, we are able to demonstrate end-to-end learning of protocols in complex environments inspired by communication riddles and multi-agent computer vision problems with partial observability. We propose two approaches for learning in these domains: Reinforced Inter-Agent Learning (RIAL) and Differentiable Inter-Agent Learning (DIAL). The former uses deep Q-learning, while the latter exploits the fact that, during learning, agents can backpropagate error derivatives through (noisy) communication channels. Hence, this approach uses centralised learning but decentralised execution. Our experiments introduce new environments for studying the learning of communication protocols and present a set of engineering innovations that are essential for success in these domains.

- 完全合作
- 部分观测
- 端到端训练 protocol
- CTDE

Motivation:

- Why does language use discrete structures?
- What role does the environment play?
- What is innate and what is learned?
- How can agents use machine learning to automatically discover the communication protocols they need to coordinate their behaviour?

Setting:

fully cooperative,

partially observable,

share the goal of maximising the same discounted sum of rewards.

Two actions: **affect the environment** + **communication action** via a discrete limited-bandwidth channel.

Character:

It is only when multiple agents and partial observability coexist that agents have the incentive to communicate. As no communication protocol is given a priori, the agents must develop and agree upon such a protocol to solve the task.

CTDE:

Communication between agents is not restricted during learning, which is performed by a centralised algorithm; however, during execution of the learned policies, the agents can communicate only via the limited-bandwidth channel.

Contribution:

To our knowledge, this is the first time that either differentiable communication or reinforcement learning (RL) with deep neural networks have succeeded in learning communication protocols in complex environments.

The results also show that deep learning, by better exploiting the opportunities of centralised learning, is a uniquely powerful tool for learning communication protocols.

Finally, this study advances several engineering innovations, outlined in the experimental section, that are essential for learning communication protocols in our proposed benchmarks.

Protocols are mappings from action-observation histories to sequences of messages, the space of protocols is extremely high-dimensional.

Automatically discovering effective protocols in this space remains an **elusive challenge**, including coordinating the sending and interpreting of messages.

An Example:

If one agent sends a useful message to another agent, it will only receive a positive reward if the receiving agent correctly interprets and acts upon that message.

If it does not, the sender will be discouraged from sending that message again.

Hence, **positive rewards are sparse**, arising only when sending and interpreting are properly coordinated, which is hard to discover via random exploration.

Key Point: Combine DRQN + IQL

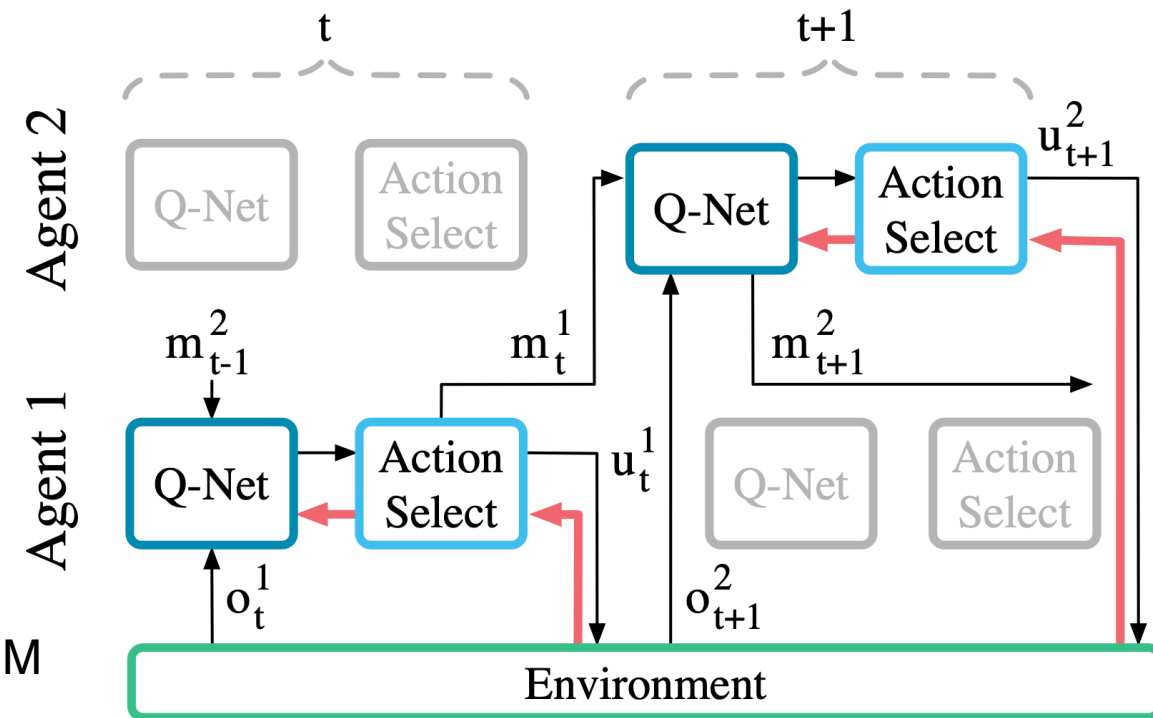
$$Q^a(o_t^a, m_{t-1}^{a'}, h_{t-1}^a, u^a)$$

environment action $u \in U$

communication action $m \in M$

Split the network into **Qu** and **Qm**, the action selector separately picks **ua** and **ma** from **Qu** and **Qm**, using an ϵ -greedy policy.

Action selection requires maximising over U and **then** over M



(a) RIAL - RL based communication

2 modifications:

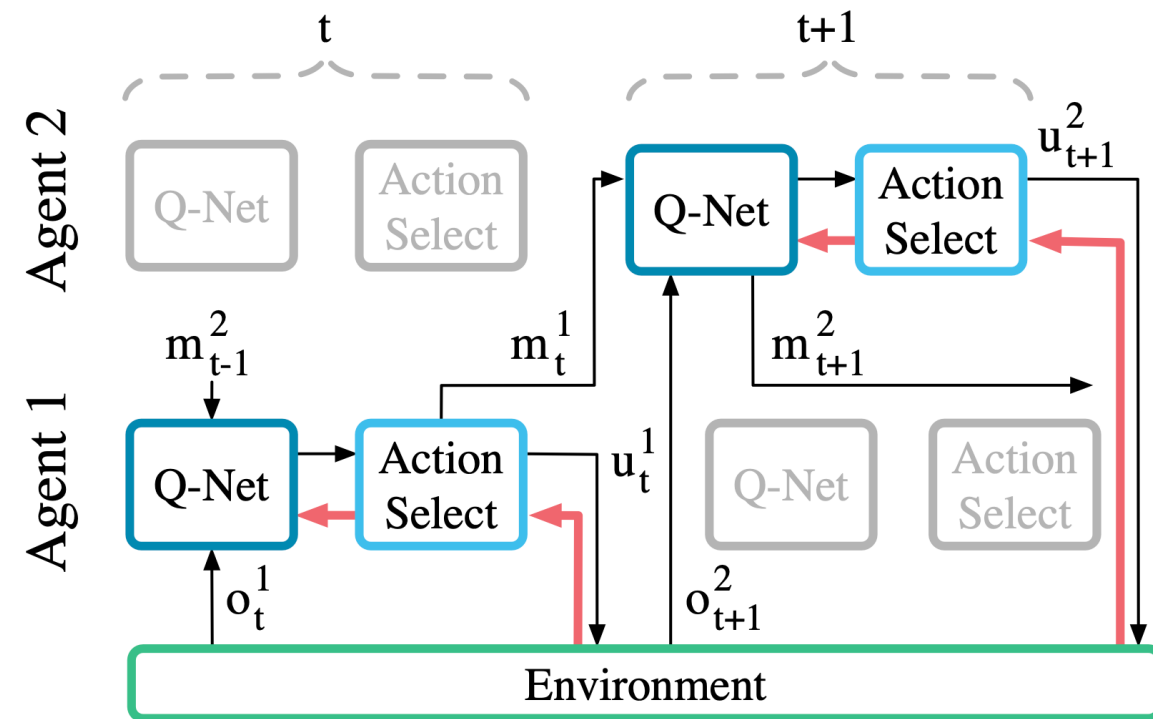
- Disable ER, avoid obsolete experience and misleading
- Solve POMDP, feed in the actions u and m taken by each agent as inputs on the next time-step.

Parameter Sharing

Only learn one network, used by all agents;
Receive different obs. and evolve different hidden states, behave differently.

In addition, receives own index a as input to specialize
Speed up training

$$Q_u(o_t^a, m_{t-1}^{a'}, h_{t-1}^a, \underline{u_{t-1}^a}, m_{t-1}^a, a, u_t^a) \quad Q_m(\cdot)$$

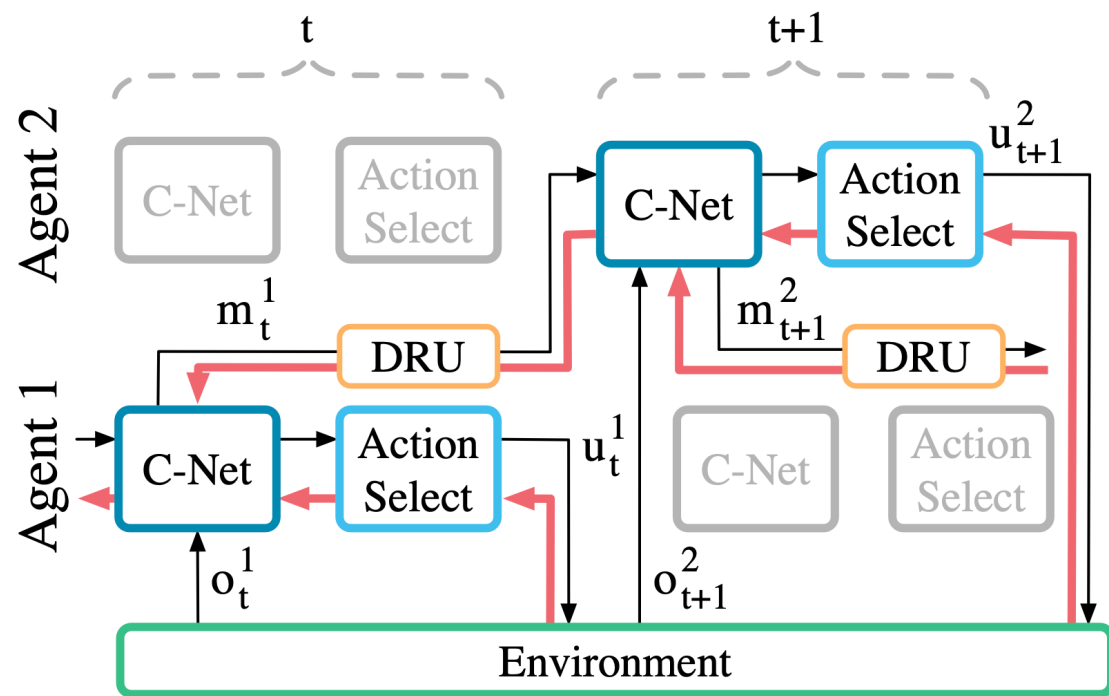


(a) RIAL - RL based communication

During decentralised execution, each agent uses its own copy of the learned network, evolving its own hidden state, selecting its own actions, and communicating with other agents only through the communication channel.

Key Point: Gradient across agents

- Limitation:
RIAL agents do not give each other feedback about their communication actions.
- Solving this limitation:
CTDE + Q-networks, push gradients from one agent to another through communication channel across the agents. Feedback reduces the required amount of learning by trial and error.



(b) DIAL - Differentiable communication

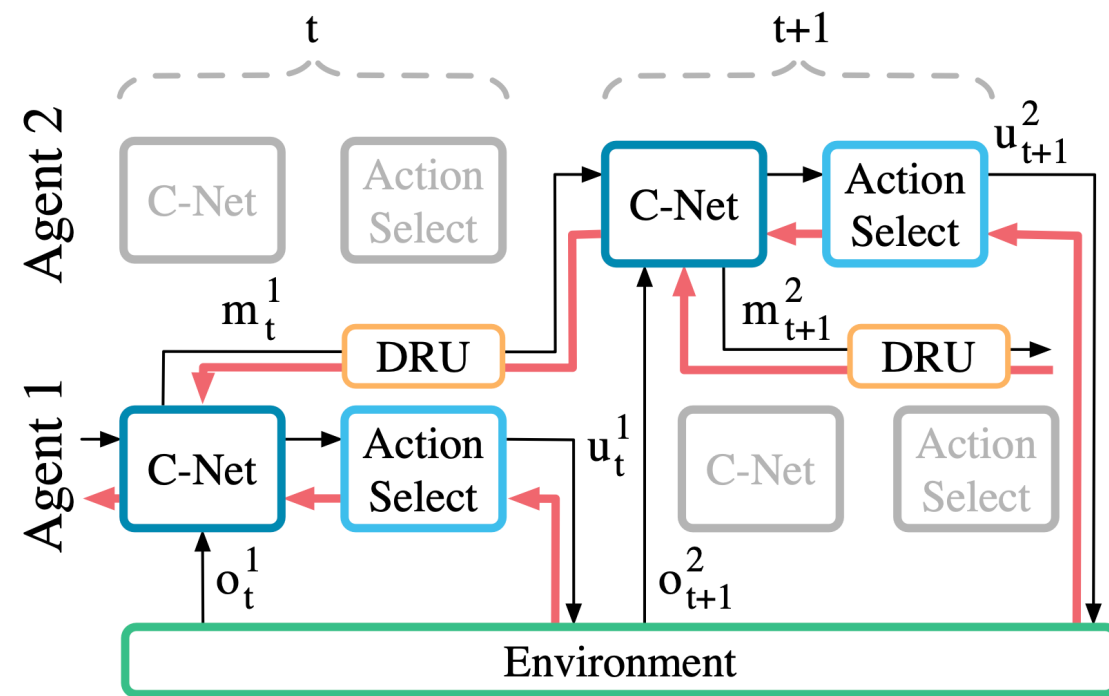
$$Q(\cdot), m_t^a = \text{C-Net} \left(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a; \theta_i \right).$$

Key Point: Gradient across agents

During centralised learning, communication actions are replaced with **direct connections** between the output of one agent's network and the input of another's.

C-Net outputs two distinct types of values:

1. $Q(\cdot)$, the Q-values for the environment actions, which are fed to the action selector;
2. Message $m(\cdot)$, the real-valued message to other agents, processed by the **discretise/regularise unit** ($\text{DRU}(m^a_t)$).



(b) DIAL - Differentiable communication

DRU regularizes it with logistic during learning, and discretises it during execution

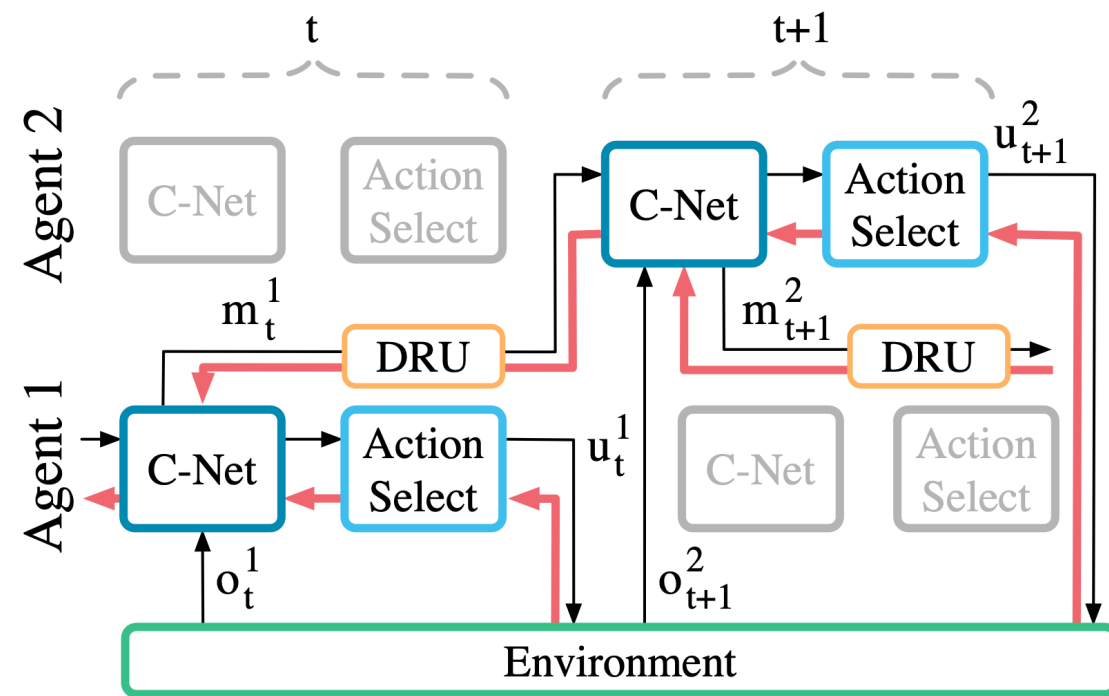
$$\text{DRU}(m_t^a) = \text{Logistic}(\mathcal{N}(m_t^a, \sigma)), \quad \text{DRU}(m_t^{\bar{a}}) = \mathbb{1}\{m_t^{\bar{a}} > 0\},$$

Richer training signal than DQN loss for Q_m in RIAL.

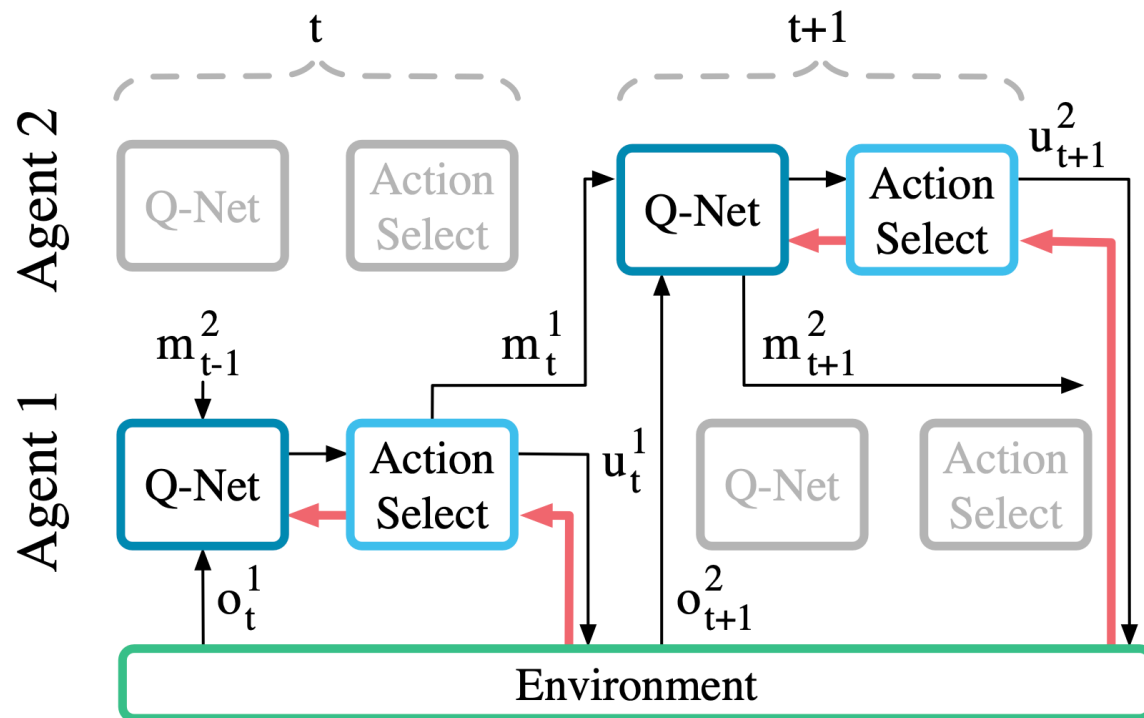
While the DQN error is nonzero only for the selected message, the incoming gradient is a **l_m -dimensional vector** that can contain more information, here l_m is the length of m .

It also allows the network to directly adjust messages in order to minimise the downstream DQN loss, **reducing the need for trial and error exploration** to learn good protocols.

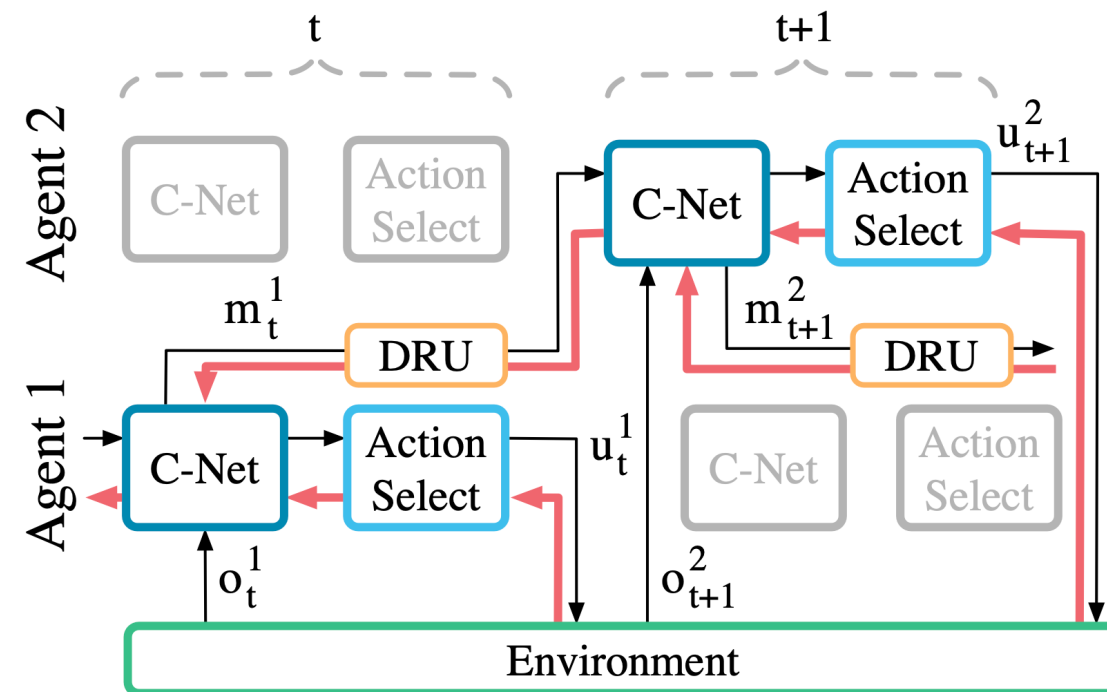
DIAL can also scale naturally to large discrete message spaces, since it learns binary encodings instead of the one-hot encoding in RIAL, $l_m = O(\log(|M|))$.



(b) DIAL - Differentiable communication



(a) RIAL - RL based communication



(b) DIAL - Differentiable communication

In our experiments, we use an ϵ -greedy policy with $\epsilon = 0.05$, the discount factor is $\gamma = 1$, and the target network is reset every 100 episodes. To stabilise learning, we execute parallel episodes in batches of 32. The parameters are optimised using RMSProp with momentum of 0.95 and a learning rate of 5×10^{-4} . The architecture makes use of *rectified linear units* (ReLU), and *gated recurrent units* (GRU) [20], which have similar performance to *long short-term memory* [21] (LSTM) [22, 23]. Unless stated otherwise we set $\sigma = 2$, which was found to be essential for good performance. We intent to published the source code online.

the Q -values and the messages m . The input for agent a is defined as a tuple of $(o_t^a, m_{t-1}^a, u_{t-1}^a, a)$. The inputs a and u_{t-1}^a are passed through lookup tables, and m_{t-1}^a through a 1-layer MLP, both producing embeddings of size 128. o_t^a is processed through a task-specific network that produces an additional embedding of the same size. The state

$$z_t^a = (\text{TaskMLP}(o_t^a) + \text{MLP}[|M|, 128](m_{t-1}^a) + \text{Lookup}(u_{t-1}^a) + \text{Lookup}(a))$$

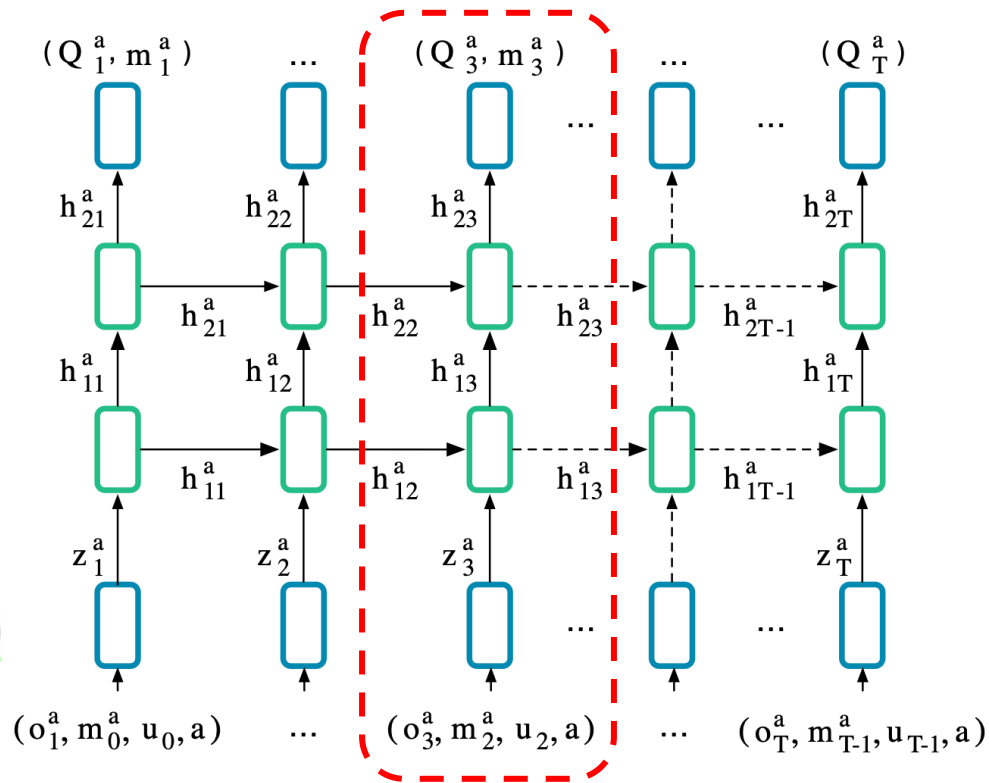


Figure 2: DIAL architecture.

We found that performance and stability improved when a batch normalisation layer [24] was used to preprocess m_{t-1} . z_t^a is processed through a 2-layer RNN with GRUs, $h_{1,t}^a = \text{GRU}[128, 128](z_t^a, h_{1,t-1}^a)$, which is used to approximate the agent's action-observation history. Finally, the output $h_{2,t}^a$ of the top GRU layer, is passed through a 2-layer MLP $Q_t^a, m_t^a = \text{MLP}[128, 128, (|U| + |M|)](h_{2,t}^a)$.

Experiment 1: Switch Riddle

100 prisoners, random selected into the room with one light, in room observe light.

Tell: all prisoners have visited this room, right free, wrong executed. **Fully cooperative**

Communication only today (training), can't from tomorrow(execution).

- Obs: in/out (0,1)
- Action: tell/none (in); none (out)
- Message: 1-bit switch position (on/off)
- Reward: right 1, wrong -1, none 0
- Time horizon: $4n - 6$

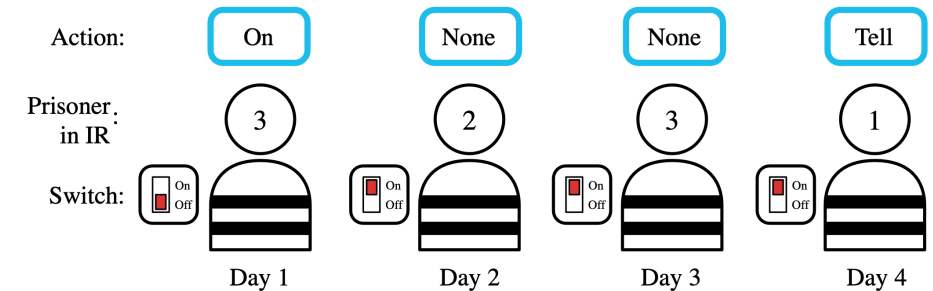
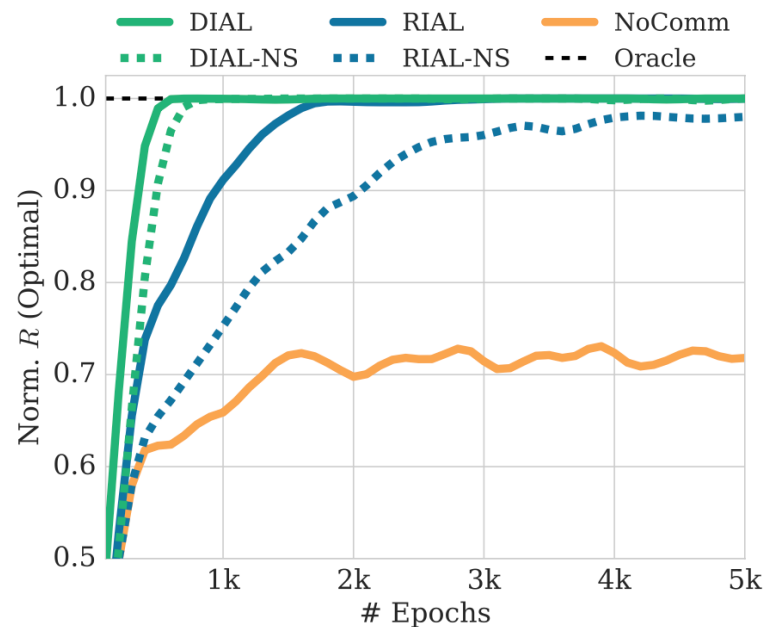
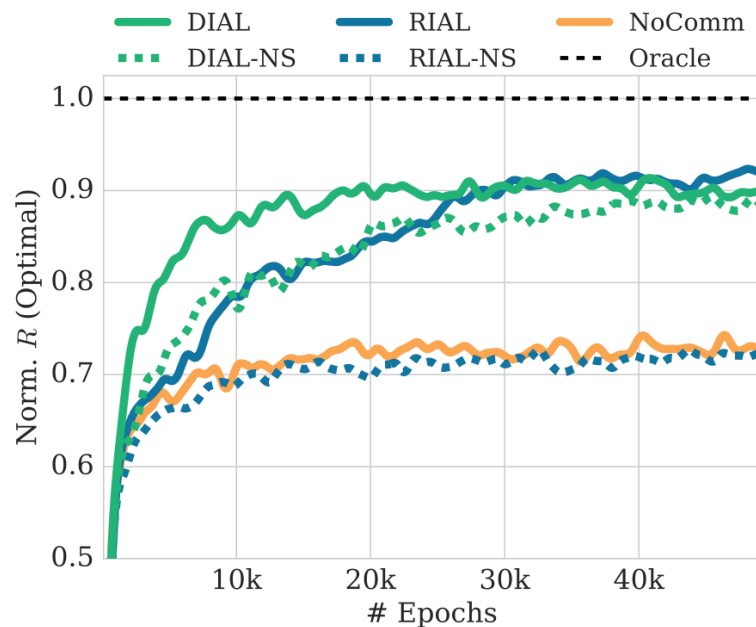


Figure 3: *Switch*: Every day one prisoner gets sent to the interrogation room where he sees the switch and chooses from “On”, “Off”, “Tell” and “None”.

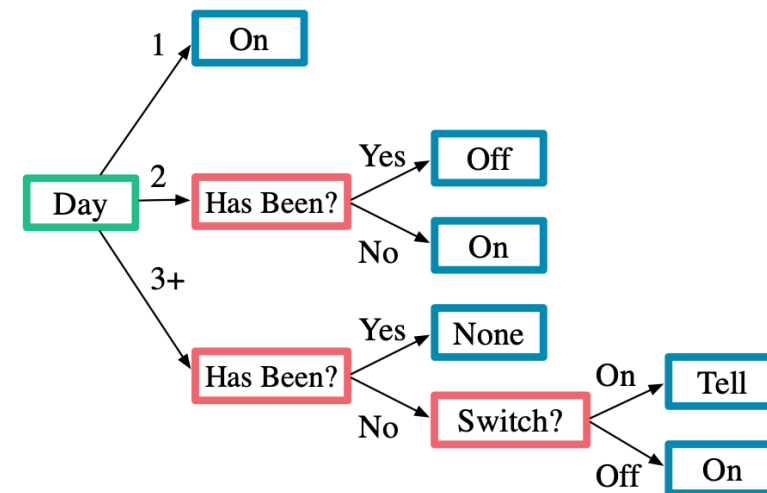
Experiment 1: Switch Riddle Results



(a) Evaluation of $n = 3$



(b) Evaluation of $n = 4$



(c) Protocol of $n = 3$

Figure 4: *Switch*: (a-b) Performance of DIAL and RIAL, with and without (-NS) parameter sharing, and NoComm-baseline, for $n = 3$ and $n = 4$ agents. (c) The decision tree extracted for $n = 3$ to interpret the communication protocol discovered by DIAL.

Complexity. The switch riddle poses significant protocol learning challenges. At any time-step t , there are $|o|^t$ possible observation histories for a given agent, with $|o| = 3$: the agent either is not in the interrogation room or receives one of two messages when he is. For each of these histories, an agent can choose between $4 = |U||M|$ different options, so at time-step t , the single-agent policy space is $(|U||M|)^{|o|^t} = 4^{3^t}$. The product of all policies for all time-steps defines the total policy space for an agent: $\prod 4^{3^t} = 4^{(3^{T+1}-3)/2}$, where T is the final time-step. The size of the multi-agent policy space grows exponentially in n , the number of agents: $4^{n(3^{T+1}-3)/2}$. We consider a setting where T is proportional to the number of agents, so the total policy space is $4^{n3^{O(n)}}$. For $n = 4$, the size is 4^{88572} . Our approach using DIAL is to model the switch as a continuous message, which is binarised during decentralised execution.

Color-digit

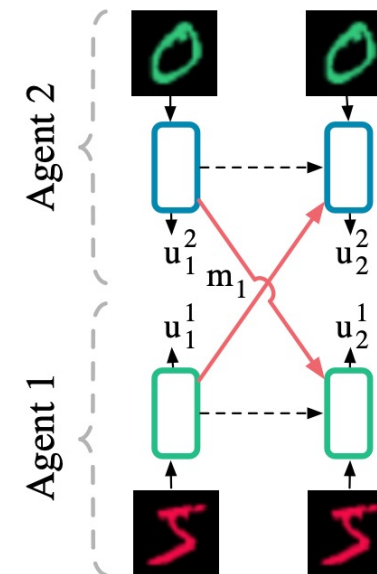
- Color label: red/green 0,1
- Digit value: 0~9
- Input: 2x28x28 size pixels
- Message: 1-bit
- Reward depends on action, color, parity (odd,/even), Total reward sums r1 and r2

$$r(a) = 2(-1)^{a_2^a + c^a + d^{a'}} + (-1)^{a_2^a + d^a + c^{a'}}$$

Step 1: Both send 1-bit message,

Step 2: Select binary action u_2^a

Agents need to agree to encode/decode color or parity, where parity has greater rewards.



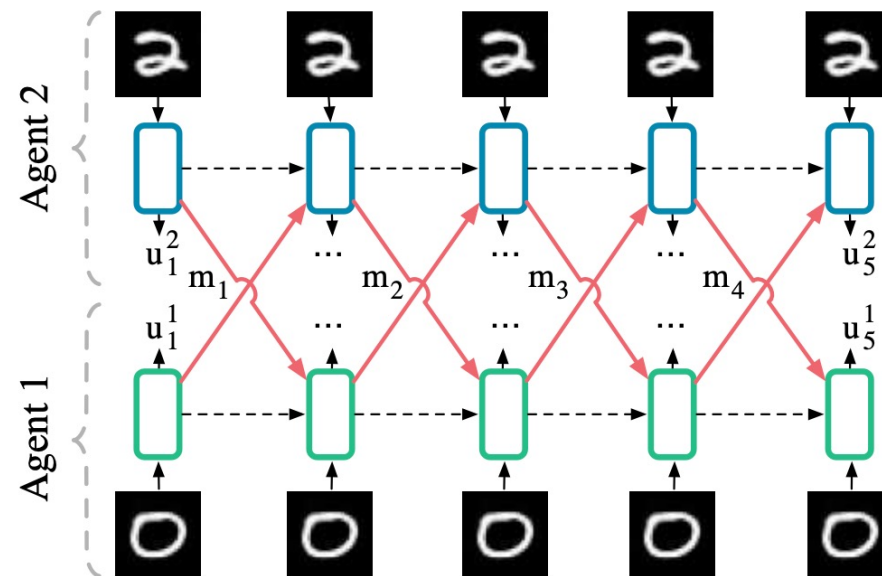
Experiment 2: MNIST Game

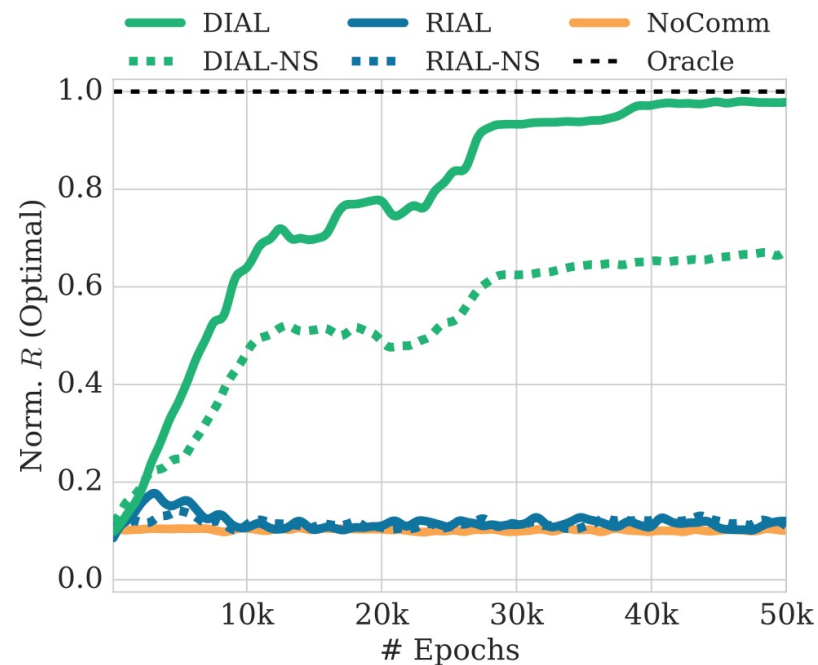
Multi-step

- action: 0~9
- Message: m_t^a 0~9
- Time horizon $t=5$
- Reward at $t=5$ given $r_5 = 0.5$ for each right guessed digit

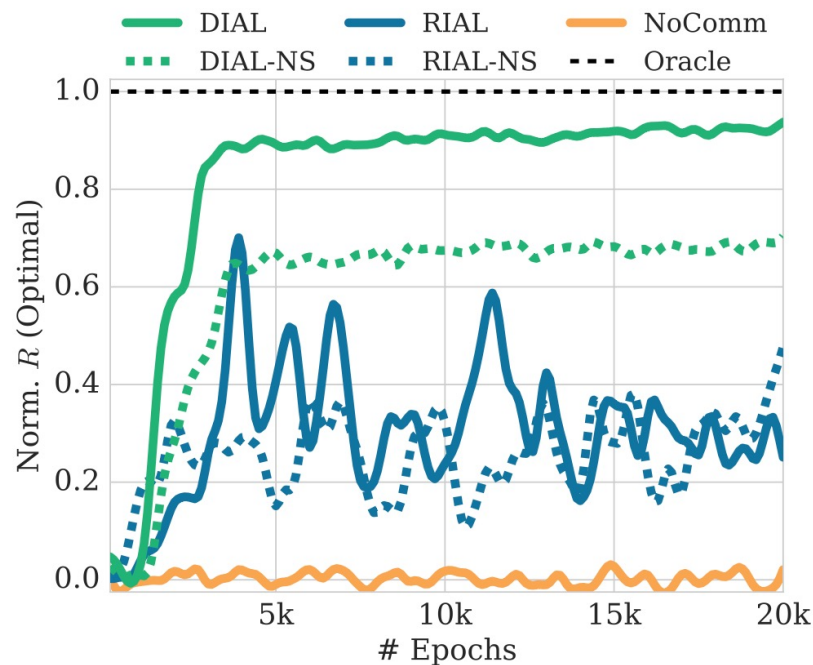
Agents must find a protocol that integrates information across the four messages they exchange.

Noise = 0.5





(a) Evaluation of Multi-Step



(b) Evaluation of Colour-Digit

Figure 6(c) is a table showing the extracted coding scheme for multi-step MNIST. The y-axis represents the True Digit (0-9) and the x-axis represents the Step (1-4). The table shows the relationship between the true digit and the step.

True Digit	Step 1	Step 2	Step 3	Step 4
9	0	1	0	0
8	0	0	0	0
7	0	1	1	1
6	1	1	0	0
5	1	0	1	1
4	0	0	1	0
3	1	0	0	1
2	0	0	1	1
1	1	1	1	1
0	1	0	0	0

(c) Protocol of Multi-Step

Figure 6: *MNIST Games*: (a,b) Performance of DIAL and RIAL, with and without (-NS) parameter sharing, and NoComm, for both MNIST games. (c) Extracted coding scheme for multi-step MNIST.

6.4 Effect of Channel Noise

The question of why language evolved to be discrete has been studied for centuries, see e.g., the overview in [27]. Since DIAL learns to communicate in a continuous channel, our results offer an illuminating perspective on this topic.

In particular, Figure 7 shows that, in the switch riddle, DIAL without noise in the communication channel learns centred activations. By contrast, the presence of noise forces messages into two different modes during learning. Similar observations have been made in relation to adding noise when training document models [12] and performing classification [11]. In our work, we found that adding noise was essential for successful training. More analysis on this is provided in Appendix C.

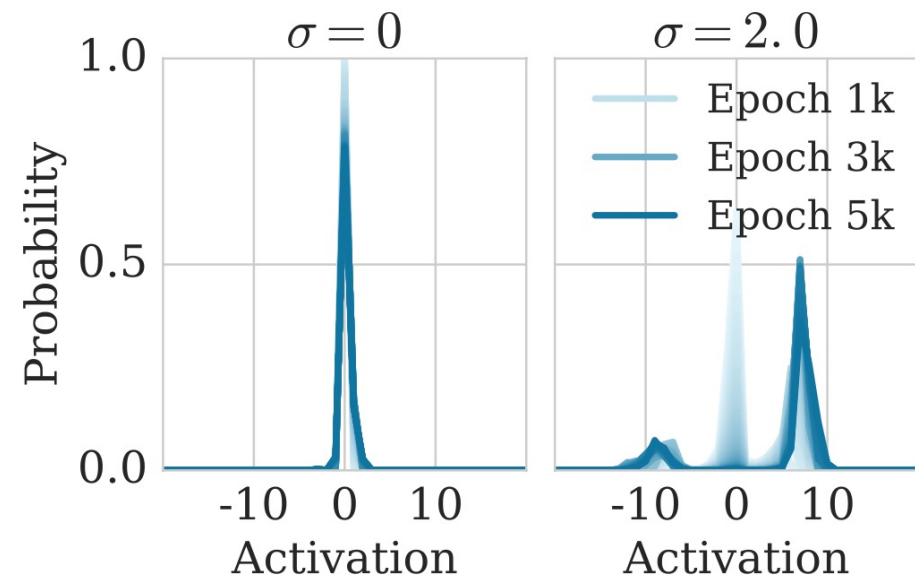


Figure 7: DIAL’s learned activations with and without noise in DRU.

Add large noise to increase the information ability in the channel. As we communication continuous message 0~1 in training, but only transmit 0 or 1 in execution. Without the noise disturbance, from the perspective of receiver, message 0/1 can only represent a small set of message such as (0~0.1)/(0.9~1). With the large noise, e.g., the green shadow area, 0/1 can represent the full set of message 0~1, leading to decodable values.

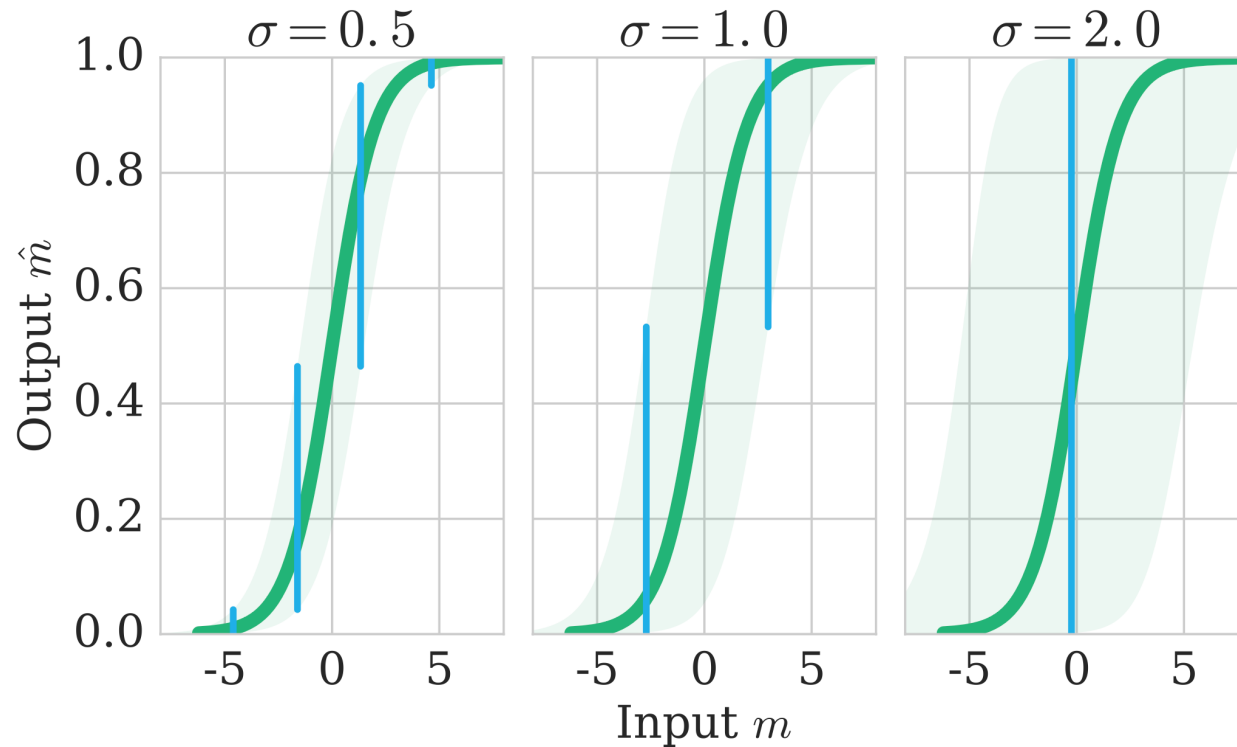


Figure 8: Distribution of regularised messages, $P(\hat{m}|m)$ for different noise levels. Shading indicates $P(\hat{m}|m) > 0.1$. Blue bars show a division of the x -range into intervals s.t. the resulting y -values have a small probability of overlap, leading to decodable values.

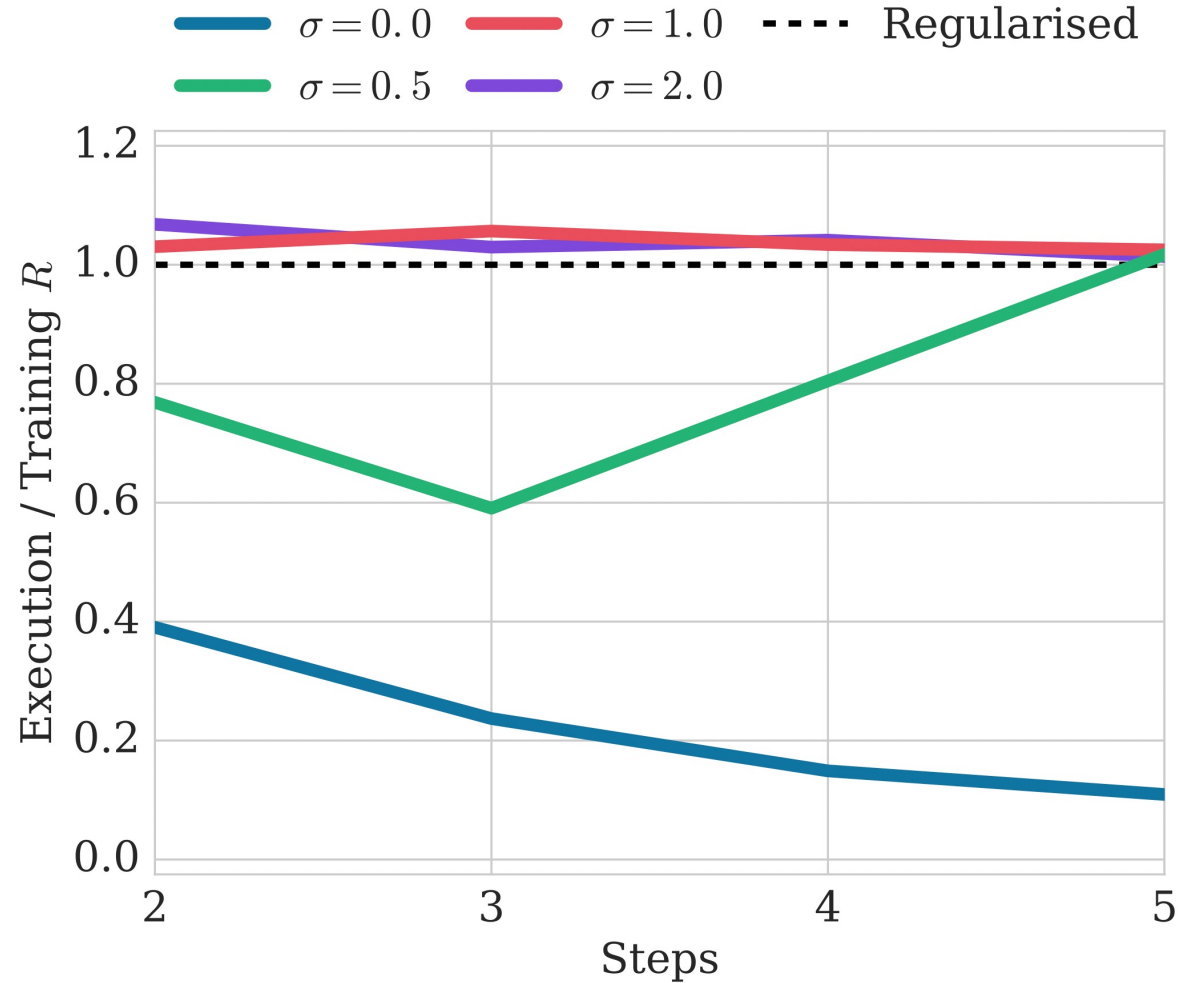


Figure 9: Final evaluation performance on multi-step MNIST of DIAL normalised by training performance after 50K epochs, under different noise regularisation levels $\sigma \in \{0, 0.5, 1, 1.5, 2\}$, and different numbers of steps $step \in [2, \dots, 5]$.

Algorithm 1 Differentiable Communication (DIAL)

 Initialise θ_1 and θ_1^-
for each episode e **do**
 $s_1 =$ initial state, $t = 0$, $h_0^a = \mathbf{0}$ for each agent a
while $s_t \neq$ terminal **and** $t < T$ **do**
 $t = t + 1$
for each agent a **do**

 Get messages $\hat{m}_{t-1}^{a'}$ of previous time-steps from agents m' and evaluate C-Net:

$$Q(\cdot), m_t^a = \text{C-Net} \left(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a; \theta_i \right)$$

 With probability ϵ pick random u_t^a , else $u_t^a = \max_a Q \left(o_t^a, \hat{m}_{t-1}^{a'}, h_{t-1}^a, u_{t-1}^a, a, u; \theta_i \right)$

 Set message $\hat{m}_t^a = \text{DRU}(m)$, where $\text{DRU}(m) = \begin{cases} \text{Logistic}(\mathcal{N}(m_t^a, \sigma)), & \text{if training, else} \\ \mathbb{1}\{m_t^a > 0\} \end{cases}$

 Get reward r_t and next state s_{t+1}

 Reset gradients $\nabla\theta = 0$
for $t = T$ **to** 1, -1 **do**
for each agent a **do**

$$y_t^a = \begin{cases} r_t, & \text{if } s_t \text{ terminal, else} \\ r_t + \gamma \max_u Q \left(o_{t+1}^a, \hat{m}_t^{a'}, h_t^a, u_t^a, a, u; \theta_i^- \right) \end{cases}$$

Accumulate gradients for action:

$$\Delta Q_t^a = y_t^a - Q \left(o_j^a, h_{t-1}^a, \hat{m}_{t-1}^{a'}, u_{t-1}^a, a, u_t^a; \theta_i \right)$$

$$\nabla\theta = \nabla\theta + \frac{\partial}{\partial\theta} (\Delta Q_t^a)^2$$

Update gradient chain for differentiable communication:

$$\mu_j^a = \mathbb{1}\{t < T - 1\} \sum_{m' \neq m} \frac{\partial}{\partial \hat{m}_t^a} \left(\Delta Q_{t+1}^{a'} \right)^2 + \mu_{t+1}^{a'} \frac{\partial \hat{m}_{t+1}^{a'}}{\partial \hat{m}_t^a}$$

Accumulate gradients for differentiable communication:

$$\nabla\theta = \nabla\theta + \mu_t^a \frac{\partial}{\partial m_t^a} \text{DRU}(m_t^a) \frac{\partial m_t^a}{\partial\theta}$$

$$\theta_{i+1} = \theta_i + \alpha \nabla\theta$$

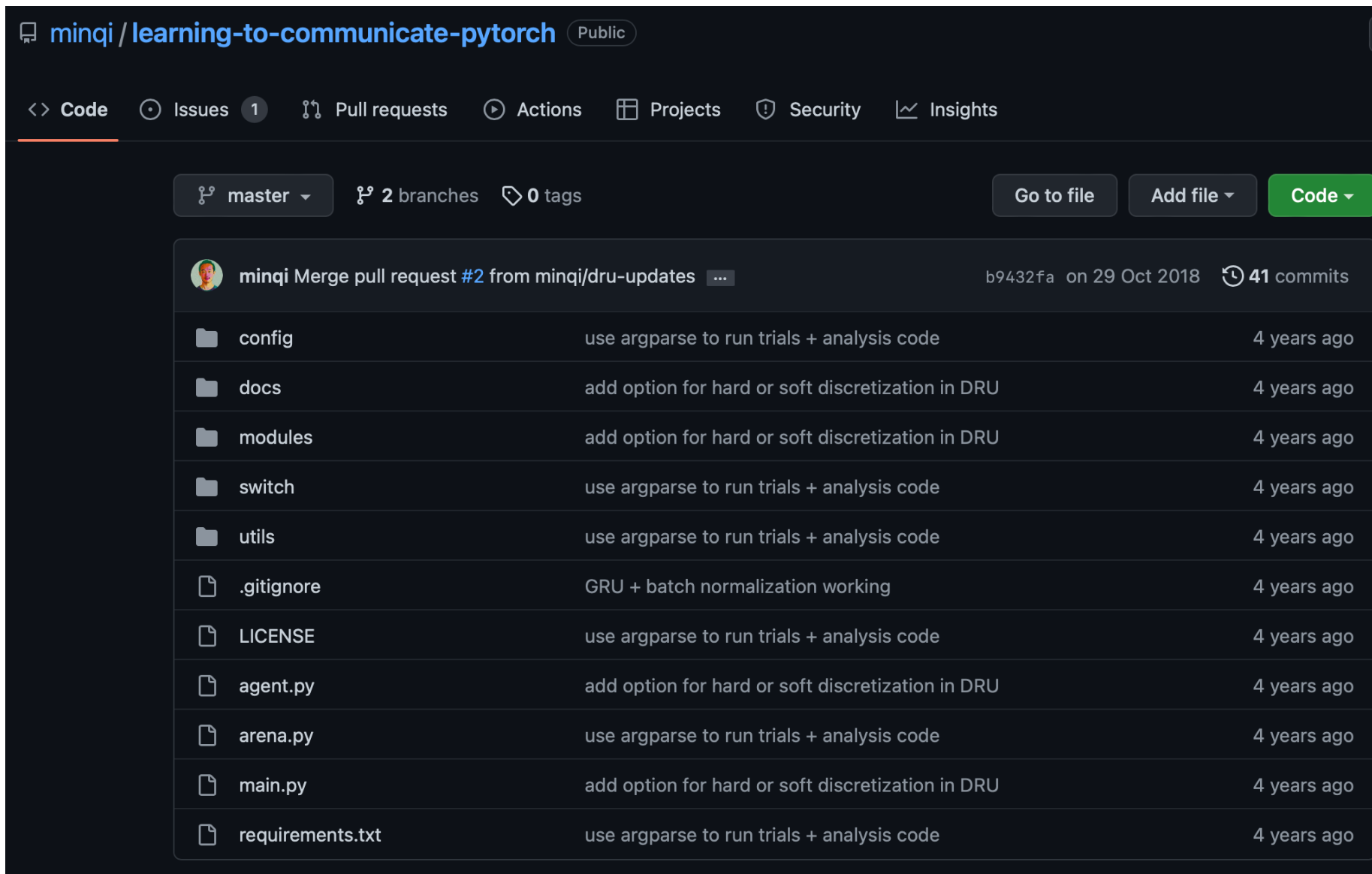
 Every C steps reset $\theta_i^- = \theta_i$

against the target Q-value, y_t^a , for the action chosen, u_t^a . We also update the message gradient chain μ_t^a which contains the derivative of the downstream bootstrap error $\sum_{m, t' > t} \left(\Delta Q_{t+1}^{a'} \right)^2$ with respect to the outgoing message m_t^a .

To allow for efficient calculation, this sum can be broken out into two parts: The first part, $\sum_{m' \neq m} \frac{\partial}{\partial \hat{m}_t^a} \left(\Delta Q_{t+1}^{a'} \right)^2$, captures the impact of the message on the total estimation error of the next step. The impact of the message m_t^a on all other future rewards $t' > t + 1$ can be calculated using the partial derivative of the outgoing messages from the agents at time $t + 1$ with respect to the incoming message m_t^a , multiplied with their message gradients, $\mu_{t+1}^{a'}$. Using the message gradient, we can calculate the derivative with respect to the parameters, $\mu_t^a \frac{\partial \hat{m}_t^a}{\partial \theta}$.

Having accumulated all gradients, we conduct two parameter updates, first θ_i in the direction of the accumulated gradients, $\nabla \theta$, and then every C steps $\theta_i^- = \theta_i$. During decentralised execution, the outgoing activations in the channel are mapped into a binary vector, $\hat{m} = \mathbb{1}\{m_t^a > 0\}$. This ensures that discrete messages are exchanged, as required by the task.

<https://github.com/minqi/learning-to-communicate-pytorch>



The screenshot shows the GitHub interface for the repository 'minqi / learning-to-communicate-pytorch'. The repository is public and has 1 issue, 2 branches, and 0 tags. The current branch is 'master'. The repository contains a list of files and folders, including 'config', 'docs', 'modules', 'switch', 'utils', '.gitignore', 'LICENSE', 'agent.py', 'arena.py', 'main.py', and 'requirements.txt'. The repository was last updated 4 years ago.

minqi / **learning-to-communicate-pytorch** Public

<> Code Issues 1 Pull requests Actions Projects Security Insights

master 2 branches 0 tags Go to file Add file Code

minqi Merge pull request #2 from minqi/dru-updates b9432fa on 29 Oct 2018 41 commits

config	use argparse to run trials + analysis code	4 years ago
docs	add option for hard or soft discretization in DRU	4 years ago
modules	add option for hard or soft discretization in DRU	4 years ago
switch	use argparse to run trials + analysis code	4 years ago
utils	use argparse to run trials + analysis code	4 years ago
.gitignore	GRU + batch normalization working	4 years ago
LICENSE	use argparse to run trials + analysis code	4 years ago
agent.py	add option for hard or soft discretization in DRU	4 years ago
arena.py	use argparse to run trials + analysis code	4 years ago
main.py	add option for hard or soft discretization in DRU	4 years ago
requirements.txt	use argparse to run trials + analysis code	4 years ago



Thanks