Experiments on learning to communicate

Hugo Berard, Tom Bosc

April 23, 2017

1 Introduction

There is a recent surge of papers on the theme of learning to communicate [1] [2] [3] [4]. The common setting is a Partially Observable Markov Decision Process (POMDP) where several agents act at each timestep and share the same rewards. The environments are designed so that communication is necessary to for agents to get a reward: one agent has an information which another agent needs to pick the action that leads to the rewards. The emphasis is on learning the language, that is, tokens emitted by agents don't have a predetermined meaning. Rather, the meaning emerges from its use.

The goal of this project is to study how Policy Gradients (PG) methods and Deep Q-Learning (DQN) perform in this kind of POMDP with several agents. The evaluation of the method is done on a grid world where communication is necessary to maximize the returns.

2 Models

2.1 Recurrent Policy Gradients

Indirect reinforcement learning approaches to control require learning a state-action value function $Q(s, a)$. Policy are derived from Q by maximizing over actions (*greedifying*). On the contrary, with Policy Gradient (PG) methods, the policy is directly parametrized. There are several motivations for doing that[5]. Firstly, in control problems, the Q function is not the quantity of interest. Furthermore, computing this quantity can be a harder problem. Finally, by comparison with ϵ-greedy policy where the entropy or randomness of the policy is fixed using ϵ, here it is flexible and adjusted during the learning process. Here we consider learning stochastic policies. The policy is denoted by π_θ and the objective is then to find the parameters θ that maximize the value function of the policy evaluated in the start state:

$$argmax_\theta v_{\pi_\theta}(s_0)$$

PG methods require computing the gradient, or at least estimating it when the policy is not differentiable. The policy gradient theorem states that:

$$\nabla_\theta v_{\pi_\theta} = \sum_s d_{\pi_\theta} \sum_a \nabla \pi_\theta(a|s) q_{\pi_\theta}(s,a)$$

This sum over states and actions is actually an expectation under the policy π_θ and the environment's dynamics:

$$\nabla_\theta v_{\pi_\theta} = E_{\pi_\theta}[\gamma^t R_t \nabla log \pi_\theta(A_t|S_t)]$$

We can estimate the empirical return with Monte-Carlo and the log of the policy is known so this estimate is tractable.

In POMDPs, the agent has to infer the state its in. Indeed, it only has access to an observation which is a function of the state. This partial observability is generally dealt with by adding memory to the agent: at each timestep, the agent updates its memory which contains probabilities to be in the true states. Based on this probability that we call belief[1], the agent can decide on actions.

Recurrent Policy Gradients[6] (RPG) is a PG algorithm that can solve POMDPs. It combines REINFORCE with Recurrent Neural Networks (RNN). REINFORCE can be used jointly with backpropagation as it is in general a gradient estimator[7]. If the policy π is a probability mass or density function parametrized by parameters $y(\theta)$ where y is a deterministic function, we can decompose the expectation of the gradient using the chain rule:

$$E[\frac{\partial \pi}{\partial \theta_i}] = E[\frac{\partial \pi}{\partial y}] \frac{\partial y}{\partial \theta_i}$$

In the case of RPG, REINFORCE is computed on the stochastic outputs units of the RNN (first factor of RHS) whereas regular backpropagation is used on the deterministic part of the RNN (second factor of RHS). Here, $y_t = f(h_t, \theta)$ where h_t is the hidden state of the LSTM at timestep t. The hidden state can be seen as the belief and the updates are learned and represented as the transition matrix. The inputs of the LSTM are the observations of the agent, the last action that was performed by the agent (necessary because the policy is stochastic) and the reward. The outputs of the LSTM are the parameters of the distribution from which the actions will be sampled.

A Long Short-Term Memory[8] (LSTM) RNN is chosen for convenience and performance.

To reduce variance, a baseline is used. In MDPs, baselines are taken to be v_π so that the log gradient term is weighted by the advantage function $q_\pi(s,a) - v_\pi(s) = A_\pi(s,a)$. Since we are in a POMDP, we can't estimate the state value function but instead learn an approximation based on the belief state. It is recommended to use another RNN as a baseline which takes the same inputs (except for the reward) and produce an estimate of the empirical

[1] Because of the equivalence between POMDPs and Belief MDPs, which are MDPs where the states live in the probability simplex on the states of the original POMDP.

returns. We can train this RNN supervisedly as we have access to the empirical returns.

The learning process is done offline. By storing histories like DQN's experience replay, we can use minibatches that further reduce the variance of our gradient estimates as their sizes grow.

2.2 Recurrent Deep-Q-Learning

When dealing with a large state space tabular representation becomes impractical to use. To remedy to this problem we can use function approximation and parametrize the Q-value function as a function of the state. Recently it has been shown [9] empirically that despite not having any convergence guarantee, neural networks performs well. The parameters are updated following the usual subgradient of the MSE between the predicted Q-value and the target:

$$\theta_{t+1} = \theta_t + \alpha(r + \max_a Q_{\hat{\theta}}(s_{t+1}, a) - Q_{\theta_t}(s_t, a_t))\nabla_{\theta_t} Q_{\theta_t}(s_t, a_t)$$

In order to work well, we need to introce several tricks:

Replay Memory: Instead of updating online, the transitions are stored in a replay memory, and at each time step a batch of transitions is sampled from the memory to train the model.

Target Network: When computing the target, the Q-value for the next-step, we use a copy of the network but with a set of parameters $\hat{\theta}$ only updated every n iterations.

Recurrent Network: To deal with the fact that the environement is partially observable, we add a recurrent layer so that the agent is able to remember the previous observation. To train the model, following [10] we sample random episodes from the replay memory and unroll the episode for n steps.

3 Gridworld experiment

This experiment is a simplified variant of [4]. The main difference is that our environment has discrete action space and state space. This allows for a fairer comparison between RDQN and RPG. Indeed, Q-learning updates are computed by maximizing the action-state value function $Q(s, a)$ over actions which is in general costly in continuous action space.

The environment is a POMDP. There are N agents and M landmarks which are defined by an integer i and their respective coordinates x_i on a squared grid. At each timestep, agents choose an action from $A = left, right, up, down$ which makes them move[2]. Agents know their own position as well as positions of the landmarks. In addition, each agent i observes its own goal g_i which is to direct another agent to one of the landmarks. All the agents are rewarded at once every first timestep one agent reach its assigned landmark. Because of

[2]For simplicity's sake, we allow several agents to be on the same landmark at the same time.

partial observability of goals, agents need to communicate the location of the goal landmark to the goal agent. For that purpose, we allow the agent to emit one token $c \in C = \{1..V\}$ at each timestep which will be visible at the next timestep to everyone. Technically, the action space is now $C \times A$.

To sum up, at each timestep, the observation of agent i is a vector $o_i(t) = [i, x_i, c_{1..N}, g_i]$

The fact that the rewards are shared is essential. Without this, agents would have no incentive to make other agents fulfill their goals. An episode terminate when all the agents have reached their goal landmark once.

4 Results

4.1 RPG

The simplest environment has 2 agents and 3 landmarks located on a 5×5 grid world. With as little as 2 agents, the goal of each agent concern the other agent. Thus, the only information each agent needs to transmit is the ID of the landmark. The vocabulary contains 5 different tokens, which contains more tokens than necessary to disambiguate between the 3 landmarks. When the episodes have not terminated after the 50th timestep, we cut them. It still manages to learn meaningful policies because the episodes that do terminate provide reinforcement to good actions, while not terminated episodes lower the probabilities of actions taken.

We ran a limited hyperparameter search one one trial per set of hyperparameters for computational reasons. Figure 4.1 shows the results. Unsurprisingly, the larger the batch size, the faster the length of episodes go down. One can also hope that it can go further down with more iterations.

Overall, we have not managed to learn the optimal policy which should involve communication. The best set of hyperparameters reach an average of around 16 timesteps per episode that includes episodes cut off at the 50th timesteps. It is possible that the agents learn to visit the 2 landmarks that do not appear in their goals in a random order. We have computed Kendall's tau (a non-parametric test for evaluating correlations among categorial random variables) on the most sent token per episode and the goal landmark to transmit but have not found any significant correlation. This means that the agents are not communicating efficiently through the communication channel.

For simplicity, we have implemented the simplest kind of experiment but this is not enough. By comparison with another paper[2], we are in the setting of Reinforced Inter-Agent Learning (RIAL) which is the least successful learning method. By comparison, Differentiable Inter-Agent Learning (DIAL) allows to solve much harder tasks. The difference is that DIAL allows gradients to flow through one agent's parameters to another's.

Under the DIAL setting, gradients propagated across agents make it much easier to learn but it is sort of counter-intuitive as it implies that there is already a meta communication protocol available during learning: backpropagation. In

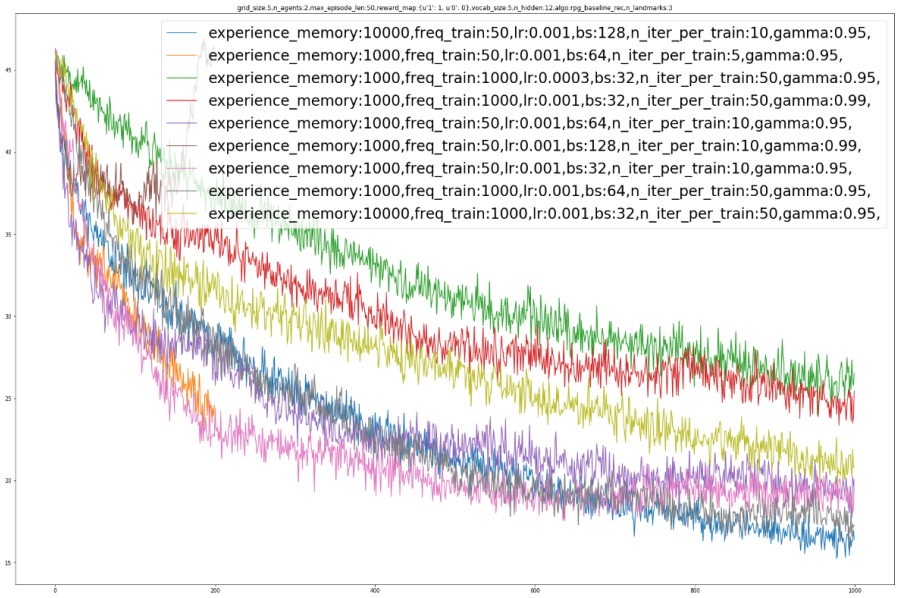

Figure 1: Every datapoint is the average episode length of the last thousand episodes. gamma is the discounting factor, lr is the learning rate, experience memory is the number of total episodes stored in the memory. Training was done every freq_train episode for n_iter_per_train iterations.

our case, on the contrary, an agent consider the other agent as part of the environment. Because the 2 agents concurrently learn, it makes the environment highly non-stationary.

One possibly related problem of our implementation is that the parameters of the two agents change at the same time every n iterations. It means that neither agent can adapt to a quick change in the policy of the other as the changes are done at once. It could help to shift the updates in time.

Another drawback of our code is that it does not allow easy parameter-sharing between agents. As shown in [2], parameter-sharing significantly improves result. This is due to the simple fact that agents can get as much more learning as there are agents interacting. A consequence of not using weight-sharing is problematic: the vocabulary can have different meaning during emission and reception of a message. For example, agent 1 can emit token 1 to mean landmark 2 and understand token 1 coming from agent 2 as meaning landmark 1.

4.2 2 Agents Fully Observable

For all experiments we use Adam [11] with $alpha = 1.10^{-3}, \beta_1 = 0.9, \beta_2 = 0.999$ and minibatches of size 32. During training we follow an ϵ-greedy policy starting

with 1 and decaying lienarly to 0.1 over 5.10^4 steps. We used a replay memory of 25 000 transitions and the target network was updated every 10 000 updates. To test the environement and the approach we first train 2 agents, on a fully observable task with 2 landmarks. We tried both using a recurrent layer and without. The architecture used consist of one hidden layer of 32 units, followed by a layer that output one single value per action corresponding to the Q-value of that function. For the recurrent architecture, the hiddent layer is replaced by a lstm layer with the same number of units, and the episode was unrolled for 4 time steps.

For both models the training is quite unstable, and starts to diverge as soon as the optimal policy is almost reached. We're not sure why this behavior occur, we tried to anneal the learning rate but it doesn't seem to work better.

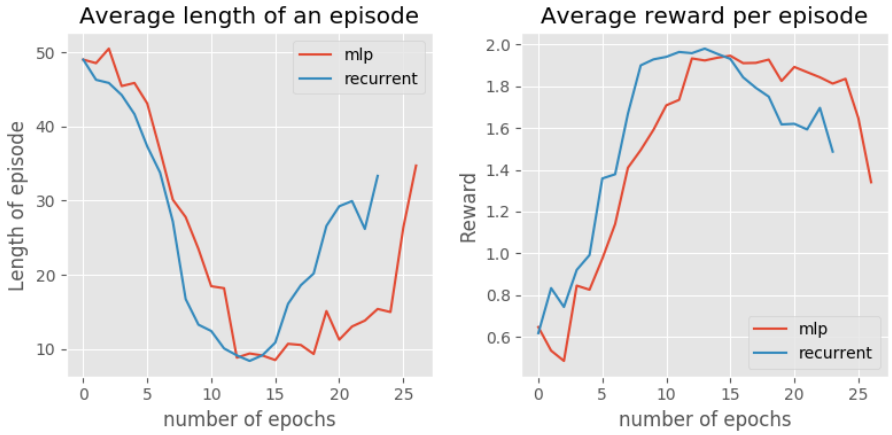

Figure 2: The average length and reward of the different models. The results are averaged over 100 episodes.

4.3 2 Agents Partially Observable

Unfortunately we were unable to make DQN works on that example.

5 Conclusion

As our experiments show dealing with partially observable environement and multiple agents is a hard problem. In our case Policy gradient methods seem to perform better, since we couldn't succeed to make DQN converge in the partially observable setting. Even tough RPG method seems to work, it doesn't seem to use the communication channel. There is still a lot to explore to make those methods work, in particular we didn't try to use parameter sharing between the

agents. We wish to continue to explore how to make DQN and RPG works on those tasks.

References

[1] Sainbayar Sukhbaatar, Rob Fergus, et al. Learning multiagent communication with backpropagation. In *Advances in Neural Information Processing Systems*, pages 2244–2252, 2016.

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

[3] Angeliki Lazaridou, Alexander Peysakhovich, and Marco Baroni. Multi-agent cooperation and the emergence of (natural) language. *arXiv preprint arXiv:1612.07182*, 2016.

[4] Igor Mordatch and Pieter Abbeel. Emergence of grounded compositional language in multi-agent populations. *arXiv preprint arXiv:1703.04908*, 2017.

[5] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*, volume 1.

[6] Daan Wierstra, Alexander Förster, Jan Peters, and Jürgen Schmidhuber. Recurrent policy gradients. *Logic Journal of IGPL*, 18(5):620–634, 2010.

[7] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.

[8] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

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

[10] Matthew Hausknecht and Peter Stone. Deep recurrent q-learning for partially observable mdps. *arXiv preprint arXiv:1507.06527*, 2015.

[11] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.