Parameterizing Policies with Natural Language in Hierarchical Reinforcement Learning

Hengyuan Hu [*1] **Denis Yarats** [*2] **Qucheng Gong** [1] **Yuandong Tian** [1] **Mike Lewis** [3]

Abstract

We explore using natural language descriptions of complex actions as an expressive and compositional representation of policies in hierarchical reinforcement learning. We represent the control module as a two-level hierarchy, where the high-level policy (*instructor*) communicates a sub-goal in natural language to the low-level policy (*executor*), which directly interacts with the environment to achieve the specified goal. We introduce a challenging real-time strategy game, and gather a dataset of 75k pairs of instructions and executions from human play. We train instructor and executor policies to play the game, and then fine-tune the instructor policy using reinforcement learning. Experiments show that linguistic supervision enables learning diverse behavioral subpolicies, and outperforms unstructured policies significantly. The structure of language proves crucial to its effectiveness for action representation. We will release our code, models and data.

1. Introduction

Hierarchical reinforcement learning (HRL) enables complex decision making in environments with large action spaces, by learning composite actions. A key challenge in HRL is creating a suitable hierarchy of actions. Several solutions have been proposed to address this. Sutton et al. (1999) and Tessler et al. (2016) suggest specifying hierarchies manually – requiring significant domain expertise which limits its generality and scalability. However, learning composite actions from only end-task supervision can result in the hierarchy collapsing to a single action (Bacon et al., 2016).

We explore parameterizing policies with natural language instructions. Natural language can express an infinite range of sub-goals, while having a compositional structure that allows generalization across policies (Oh et al., 2017; Andreas et al., 2017a). We build on previous work on executing instructions, by combining it with a model for generating sequences of instructions to solve a complex task.

We propose representing the control module as a two-level hierarchy, where the high-level policy (*instructor*) communicates a sub-goal in natural language to the low-level policy (*executor*), which then tries to achieve the specified sub-goal through direct interaction with the environment (Fig. 1). Both models are first trained to imitate humans playing the roles, and then the instructor is fine-tuned using reinforcement learning (RL). Fixing the executor improves the stability of learning, by ensuring the language will be grounded similarly to how humans would, and allowing RL to tune strategic decision making.

We gather supervision by having two players work as a team in a complex game environment. Both players have access to the same partial information about the game state. One player on the team performs duties of the instructor, she periodically issues sub-goal instructions in natural language to the other player (the executor), but otherwise has no control over individual units. The executor player then needs to achieve the current sub-goal issued by the instructor directly manipulating the units of the game. This setup forces the instructor to focus on high-level planning, while the executor concentrates on low-level control.

As a test-bed for our approach, we use a real-time strategy (RTS) game. We develop an RTS environment based on MiniRTS (Tian et al., 2017). A key property of our game is the *rock-paper-scissors* unit attack dynamic, which emphasises strategic planning over micro control. It is a challenging environment for RL because of exponentially large state-action spaces, partial observability, and a diverse range of possible strategies. However, it is relatively intuitive for humans, easing data collection.

Using this framework, we gather a dataset of 5400 games, where two humans (the instructor and executor) play against rule-based AI opponents. The dataset contains over 75k pairs of human instructions and executions, spanning a wide

[*]Equal contribution [1]Facebook AI Research, Menlo Park, CA, USA [2]Facebook AI Research & New York University, New York, NY, USA [3]Facebook AI Research, Seattle, WA, USA. Correspondence to: Denis Yarats <denisyarats@cs.nyu.edu>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

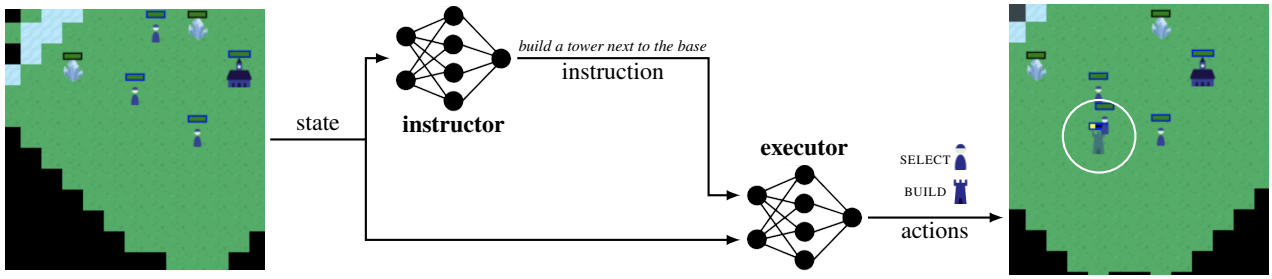


Figure 1. Two agents, designated *instructor* and *executor* collaboratively play a real-time strategy game (§2). The *instructor* formulates plans and issues instructions in natural language to the *executor*, who then executes them as a sequence of actions. We first gather a dataset of humans playing each role (§3). We then train models to imitate humans actions in each role (§4). Finally, we fine-tune the *instructor* model using reinforcement learning (§5.2).

range of strategies. We then use this dataset to supervise the executor and instructor models. Subsequently, we fine-tune the instructor policy using RL, while keeping the executor fixed. Extensive experiments show that parameterizing policies with natural language greatly improves performance over unstructured policies.

In summary, we introduce a challenging RTS environment for RL and a corresponding dataset of instruction-execution mappings. We then develop a novel model with planning and control components, connected with a compositional and expressive natural language interface. Fine-tuning instruction generation with reinforcement learning dramatically outperforms approaches that attempt to learn low-level control directly. We will release our code, models and data.

2. Task Environment

We implement our approach in a RTS game environment, which has several attractive properties compared to traditional RL environments, such as Atari (Mnih et al., 2013) or grid worlds (Sukhbaatar et al., 2015). The very large action space is challenging for non-hierarchical RL algorithms, and planning at different levels of abstraction is beneficial for both humans and machines. However, manually designed macro-actions typically cannot match strong human performance, because of the unbounded space of possible strategies (Synnaeve et al., 2016; Vinyals et al., 2017). Even with simple rules, adversarial games have the scope for complex emergent behaviour.

We introduce a new RTS game environment, which distills the key features of more complex games while being faster to simulate and more tractable to learn. Current RTS environments, such as StarCraft, have dozens of unit types, adding large overheads for new players to learn the game.

Our new RTS environment is based on MiniRTS (Tian et al., 2017). It has a set of 7 unit types, designed with a *rock-paper-scissors* dynamic such that each has some units it is effective against and vulnerable too. Maps are randomly generated each game to force models to adapt to their environment as well as their opponent. The game is designed to be intuitive for new players (for example, catapults have long range and are effective against buildings). Numerous strategies are viable, and the game presents players with dilemmas such as whether to attack early or focus on resource gathering, or whether to commit to a strategy or to attempt to scout for the opponent's strategy first.

Overall, the environment is easy for humans to learn, but challenging for machines due to the large action space, imperfect information, and the fact that the effectiveness of a strategy depends on both the map and opponent.

3. Dataset

To learn to describe policies with natural language, we gather a dataset of human play on the game. Two humans play collaboratively against a rule-based AI opponent. Both players have access to the same information about the game state, but have different roles. One is designated the *instructor*, and is responsible for designing strategies and describing them in natural language, but has no direct control. The other player, the *executor*, must ground the instructions into low level control. The *executor*'s goal is to carry out commands, not to try to win the game. This setup naturally decomposes game playing into two roles of planning and execution, with a natural language interface between them.

We collect a dataset of 5392 games of human teams against our bots.[1] Qualitatively, we observe a wide variety of different strategies. An average game consists of 13 natural language instructions and lasts for about 16 minutes. Each instruction corresponds to roughly 7 low-level actions, giving a challenging grounding problem. See Table 1.

The dataset contains over 75K instructions, most of which are unique, and their executions. The diversity of instructions shows the wide-range of high level actions that are possible when parameterizing policies with natural language.

[1] Using ParlAI (Miller et al., 2017) to connect players.

Statistic	Value
Number of games	5392
Win rate	58.67%
Number of instructions	76045
Number of unique instructions	39598
Number of words	307162
Number of unique words	2851
Average number of words per instruction	7.76
Average number of instructions per game	13.09
Average number of actions per instruction	7.18

Table 1. We gather a large language dataset for instruction generation and following. Major challenges include the wide range of unique instructions and the large number of low-level actions required to execute each instruction.

Linguistic Phenomena	Example
Counting	*Build 3 dragons.*
Spatial Reference	*Send him to the choke point behind the tower.*
Composed Actions	*Attack archers, then peasants.*
Cross-instruction anaphora	*Use it as a lure to kill them.*

Table 2. Complex linguistic phenomena naturally emerge as humans instruct others how to play the game.

We find the instructions contain a number of challenging linguistic phenomena, particularly in terms of reference to locations and units in the game, which often requires pragmatic inference. Instruction execution is typically highly dependent on context. See Table 2 for some examples.

4. Model

We factorize our control module into the executor model (§4.2), which maps instructions and the game states into low-level actions of the environment, and the instructor model (§4.3), which generates language instructions given the game states. We train both models with human supervision (§5.1), and then fine-tune the instructor with RL (§5.2).

4.1. Global State Encoder

Our instructor and executor models both condition on a fixed-size representation of the game state. We concatenate fixed size representations of the currently visible units (§4.1.1), current instruction (§4.1.2) and amount of resources available to the player. See Fig. 2.

4.1.1. Map Encoder

We first encode the currently visible game map using a convolutional network. Specifically, we discretize the map and use multiple channels to represent spatial locations for different unit and object types on the map (for example, in a particular channel we store the number of agent's peasants at each location). We zero out parts of the map that are hidden under the fog of war. We preserve the spatial dimensions after each convolutional layer. We then extract representations for agent's units, opponent's units, and other objects on the map using their corresponding coordinates. We amend these representations with the encodings of various attributes (e.g. health balance, type, previous action). Consequently, we use element-wise sum to reduce the unit and object representations to achieve fixed size vectors.

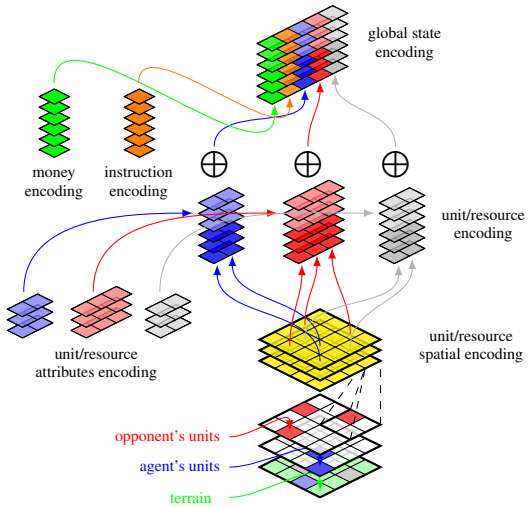
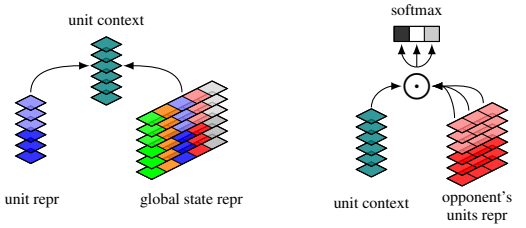


Figure 2. We encode the game map using a convolution network (§4.1.1). We then extract these encodings using spatial locations and concatenate them with embeddings of attributes, instruction and resources.

4.1.2. Current Instruction Encoder

The state also contains a fixed-size representation of the current instruction. We experiment with several models:

- An instruction-independent model (ExecutorOnly), that simply tries to mimic typical human play.
- A non-compositional encoder (OneHot) which maps each instruction to an action with no parameter sharing across instructions (with rare instructions being represented with an *unknown* embedding).
- A bag-of-words encoder (BoW), where an instruction encoding is the sum of its individual word embeddings. This model tests whether the compositionality of language can improve generalization.
- An RNN encoder (Rnn), which is order-aware. Unlike the bag-of-words encoder, this approach can differen-



(a) Per unit context

(b) Action classifier for *attack* action.

Figure 3. We combine an agent's unit and the global state representations (Fig. 3a) and use it as an input to each action classifier. For example, we can use this classifier to model a probability distribution over possible targets for the attack action (Fig. 3b).

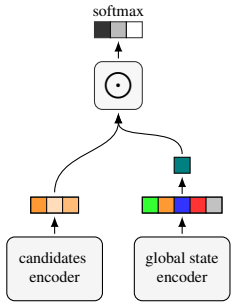


Figure 4. The discriminative instructor model generates a natural language instruction based on the game state and the previous instruction. We model a softmax distribution over a set of candidate instructions using the action classifier (Fig. 3b).

tiate instructions such as *attack the dragon with the archer* and *attack the archer with the dragon*.

4.2. Executor Model

We use the global state encoder (§4.1) to obtain a representation of the game state and the current instruction. Specifically, we extract encodings for the global state, agent's units, opponent's units, resources, and arbitrary cells of the map.

We use these representations to model agent's units actions. Initially, we create an unit context for each agent's unit by concatenating the unit's embedding with the global state representation, and feeding into an MLP (Fig. 3a).

For each unit, at each time step, we dynamically create a list of possible actions.

We first choose an action type (e.g. *move* or *attack*) with a classifier. We then choose a fine-grained action, such as moving to a particular location, or attacking a specific unit.

We define classifiers for all the available actions below. These classifiers input the contextualized representation for the actor unit and a list of targets (with corresponding representations), and then model a categorical probability distribution over the targets:

- ATTACK: takes the opponent's unit representations for the targets, and predicts which unit to attack.
- GATHER: looks similar to the ATTACK classifier, but inputs the resource representations instead.
- MOVE: models a softmax distribution over a target location represented as a corresponding cell from the convolutional feature map.
- BUILD UNIT: is only available for buildings, and chooses which unit type to build based on the unit types embeddings.
- BUILD BUILDING: is only available for worker units, and combines a MOVE classifier with a softmax over which building type to construct.

We add an additional binary classifier (GLOBAL CONTINUE), that takes the global state as an input and predicts whether all the agent's units should continue working on their previous action.

4.3. Instructor Model

The instructor model assigns a distribution over instructions given a representation of the current game state (§4.1). We experiment with two types of models:

Discriminative Models These models operate on a fixed set of instructions. Each instruction is encoded as a fixed-size vector, and the dot product of this encoding and the game state encoding is fed into a softmax classifier over the set of instructions. As in §4.1.2, we consider non-compositional (ONEHOT), bag-of-words (BOW) and RNN DISCRIMINATIVE encoders.

Generative Model The discriminative models can only choose between a fixed set of instructions. We also train a generative model, RNN GENERATIVE, which generates instructions word-by-word from left-to-right. To compare likelihoods with the discriminative models, which consider a fixed set of instructions, we re-normalize the probability of an instruction over the space of instructions in the set.

The instructor model must also decide at each time-step whether to issue a new command, or leave the executor to continue following the previous instruction. We add a simple binary classifier that conditions on the global state, and only return a new instruction if the result is positive.

5. Training

5.1. Supervised Learning

Since one game may last for tens of thousands of frames, it is not feasible or necessary to use all frames for training. Instead, we take one frame every K frames to form the supervised learning dataset. To preserve unit level actions for executor training, we put all actions that happen in $[tK, (t+1)K)$ frames onto the tKth frame if possible. For actions that cannot happen on the tKth frame, such as actions for new units built after tKth frame, we simply discard them.

Humans players sometimes did not execute instructions immediately. To ensure our executor acts promptly, we filter out action-less frames between a new instruction and first new actions.

5.1.1. Executor Model

The executor is trained to minimized the following negative log-likelihood loss:

$$\mathcal{L} = -\log P_{\text{cont}}(c|s) - (1-c) \cdot \sum_{i=1}^{|u|} \log P_{\text{A}}(a_{u_i}|s)$$

where s represents game state and instruction, $P_{\text{cont}}(\cdot|s)$ is the executor Global Continue classifier (see §4.2), c is a binary label that is 1 if all units should continue their previous action, $P_A(a_{u_i}|s)$ is the likelihood of unit i doing the correct action a_{u_i}.

5.1.2. Instructor Model

The loss for instructor models is the sum of a loss for deciding whether to issue a new instruction, and the loss for issuing the correct instruction:

$$\mathcal{L} = -\log P_{\text{cont}}(c|s) - (1-c) \cdot \mathcal{L}_{\text{lang}}$$

where s represents game state and current instruction, $P_{\text{cont}}(\cdot|s)$ is the continue classifier, and c is a binary label with $c = 1$ indicating that no new instruction is issued. The language loss $\mathcal{L}_{\text{lang}}$ is the loss for choosing the correct instruction, and is defined separately for each model.

Specifically, for OneHot instructor, the language loss is simply negative log-likelihood of a categorical classifier over a pool of N instructions. If the true target is not in the candidate pool, we mask out the language loss and only use the continue loss.

Because BoW and Rnn Discriminative can compositionally encode any instruction (in contrast to OneHot), we can additionally train on instructions from outside the candidate pool. To do this, we encode the true instruction, and discriminate against the N instructions in the candidate pool and another M randomly sampled instructions. The true target is forced to appear in the $M + N$ candidates. We then use the NLL of the true target as language loss. This approach approximates the expensive softmax over all 40K unique instructions.

For Rnn Generative, the language loss is the typical autoregressive loss.

5.2. Reinforcement Learning

The pretrained instructor and executor models form a natural starting point for hierarchical reinforcement learning. The instructor provides high level strategic decisions while the executor performs low level control for each unit given instruction and game state. In reinforcement learning, we freeze the parameters of the executor as if it is part of the environment and fine-tune instructor model with proximal policy optimization (Schulman et al., 2017).

Fixing the executor on the imitation learning policy improves the stability of learning, because then the grounding of language does not change during RL training—the meaning of an instruction is fixed as however the the executor interprets it. Effective RL on the executor is also challenging, because of the very large action space.

We train our model against a pool of rule-based AIs, which implement a variety of strategies.

The pre-trained model rarely wins against these AIs, which is problematic for RL as rewards become very sparse. We create a curriculum by dynamically adjusting the resource collection efficiency of its opponent, such that the model's win-rate stays at roughly 50%.

We found training was prone to local optima where the agent would learn a fixed strategy that was effective against some of the rule-based AI opponents but not others. To counteract this problem, we adversarially sample the opponent in proportion to the exponential moving average of its win-rate, so more successful opponents are sampled more often.

These two techniques greatly simplified learning.

6. Experiments

We first compare different supervised learning models for the *executor* (§6.1) and *instructor* (§6.2), and then with fine-tuning the *instructor* with reinforcement learning (§6.3).

6.1. Executor Model

The executor model solves a grounding problem of mapping pairs of states and instructions onto actions.

With over 75k examples, a very high dimensional action space, and multiple sentences of context, this problem in

Instruction Encoder	NLL
EXECUTORONLY	3.06
ONEHOT	3.02
BOW	2.88
RNN	**2.85**

Table 3. The negative log-likelihood of human actions under different models for encoding instructions. Modelling instructions compositionally gives the best performance, showing how the structure of natural language can help improve generalization.

isolation is one of the largest and most challenging tasks currently available for grounding language in action.

As explained in §4.1.2, we explore a series of models for encoding instructions, evaluating the likelihood they assign to human actions.

Results are shown in Table 3, and show executor models that condition on instructions language assign more likelihood to human actions than can be predicted by only considering the game state. Further, modelling instructions compositionally (BOW) allows significantly better generalization than a non-compositional encoder (ONEHOT). There is a relatively small additional gain from accounting for word-order in instructions (RNN), suggesting that the model may be able to infer the information from word-order using context.

6.2. Instructor Model

The instructor model maps game states onto instructions. In this section, we compare different supervised models.

As in §6.1, we experiment with non-compositional, bag-of-words and RNN models for instruction generation. For the RNNs, we train both a discriminative model (which maps complete instructions onto vectors, and then chooses between them) and a generative model that outputs words autoregressively. We also train a baseline TIME-STEP ONLY that cannot condition on the game state, to explore the effectiveness of a non-reactive sequence of actions.

Evaluating language generation quality is challenging, as many instructions may be reasonable in a given situation, and they may have little word overlap. We therefore compare the likelihood of the human instructions. Our models choose from a fixed set of instructions, so we measure the likelihood of choosing the correct instruction, normalized over all instructions in the set. We do not include examples with out-of-set instructions at test time, so likelihoods across different instructions sets are not comparable.

Table 4 shows results. As in the executor model experiments, we again find that more structured instruction models give better likelihoods. The gain increases with larger instruction sets, which are harder to model non-compositionally. The TIME-STEP ONLY baseline performs poorly, showing that instruction generation depends strongly on the game state.

We also compare the win-rate of different models against a baseline which directly imitates human actions (without intermediate language). Our best model strongly outperforms this baseline, with a win-rate of 58%, showing how latent language can improve even on supervised learning tasks.

Crucially, we find that the instruction set has to be large enough to give the model a sufficient range of high level actions, and that we do not outperform the non-linguistic model when using a set of only 50 instructions. Interestingly, performance is better with 250 instructions than 500 instructions, perhaps because rarer instructions are both harder to generate appropriately and to execute.

6.3. Reinforcement Learning Evaluation

Finally, we experiment with tuning the *instructor* models using RL. We explore whether RL is able to improve over pre-trained models of human actions, and whether RL to improve natural language instructions is more effective than RL operating directly on low level actions. During RL, we play against opponents sampled from a pool of rule-based systems that implement a variety of strategies.

Results are shown in Table 5. Most importantly, fine-tuning instruction generation models uniformly gives large improvements, whereas RL directly on low-level actions does not. This result shows that using intermediate actions can greatly simplify reinforcement learning, by reducing the action space. We also see that initializing the model based on human play is important for good performance.

In contrast to the pre-trained model, the best results are obtained using a larger instruction set of 500 instructions and an RNN instruction generator. This result suggests that RL is able to help models learn to effectively use instructions which are rarer in the supervised training data.

Learning curves (Fig. 5) show that compositional models of language converge faster and to a higher win-rate, likely due to being able to share parameters across instructions.

6.4. Qualitative Analysis

Observing a set of games played by our model, we find that most instructions are both generated and executed as humans plausibly would. The *executor* is often able to correctly count the number of units it should create in commands such as *build 3 dragons*.

However, there are a number of limitations. The *executor* is prone to sometimes acting without instructions. This behaviour is partly due to mimicking some humans in the training data, but may also indicate a failure to learn some dependencies between instructions and actions. The *instructor* sometimes issues commands which are not possible in a

Instructor Model (with N instructions)	NLL N=50	NLL N=250	NLL N=500	Win-rate N=50	Win-rate N=250	Win-rate N=500
TIME-STEP ONLY	0.756	0.937	1.015	38.37	44.23	43.43
ONEHOT	0.595	0.788	0.877	44.23	51.03	47.73
BOW	0.588	0.775	0.854	51.07	58.10	56.33
RNN DISCRIMINATIVE	0.580	0.757	0.829	44.17	56.27	52.97
RNN GENERATIVE	0.573	0.756	0.828	44.13	52.53	51.73

Table 4. Results for different pretrained *instructor* models, using different numbers of instructions. Win-rates are against a pre-trained executor model that does not condition on language. More compositional models improve likelihoods of human instructions, but a bag-of-words instruction encoder achieves the highest win-rate.

Instructor Model (with N instructions)	w/o RL	with RL rand init	with RL pretrained
EXECUTOR ONLY	19.53	0.58	19.90
ONEHOT (50)	16.66	30.00	43.20
BOW (50)	19.00	28.40	40.70
RNN (50)	13.60	29.60	40.80
ONEHOT (250)	18.77	31.00	47.53
BOW (250)	24.30	32.40	45.70
RNN (250)	24.43	31.57	48.03
ONEHOT (500)	18.63	28.43	44.67
BOW (500)	23.33	31.97	49.80
RNN (500)	22.90	32.03	53.70

Table 5. Win-rates for RL, training and evaluating against a pool of rule-based opponents. RL does not improve a model that works directly in the space of low-level actions, but all models with latent instructions improve dramatically. The strongest model uses an RNN to sample from a large set of 500 instructions.

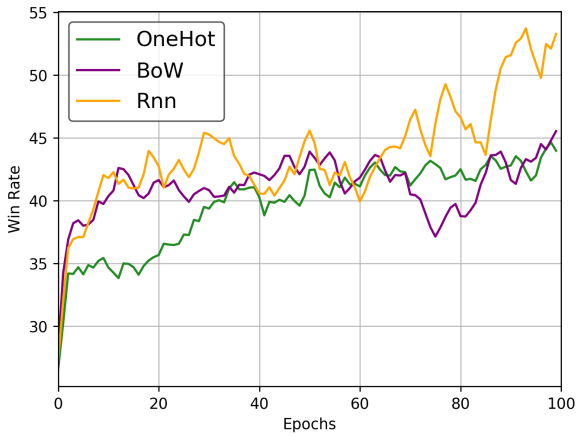


Figure 5. Reinforcement learning curves. We fix the executor model (BOW) and RL fine-tune three pre-trained instructor models: ONEHOT, BOW, and RNNusing 500 instructions. We observe better sample efficiency for compositional models.

given situation (e.g. to attack with a unit that the agent does not have). While RL learns a reasonably effective strategy, it does not make use of certain action types, such as scouting. Future work should explore solutions to these issues.

7. Related work

Although perceived as a natural way to do planning, HRL is nontrivial to train. Many previous works inject more supervision than a reward signal to accelerate training in a virtual environment. This includes reward shaping (Ng et al., 1999; Aytar et al., 2018), using auxiliary tasks (Jaderberg et al., 2016), curriculum training (Andreas et al., 2017b), imitation learning then RL fine-tuning (Wu et al., 2018), intrinsic rewards (Eysenbach et al., 2018; Tang et al., 2017; Pathak et al., 2017).

Previous works have used language as an efficient supervision for learning exponentially many policies (Oh et al., 2017; Andreas et al., 2017b), allowing zero-shot generalization across tasks. We develop this work by generating instructions as well as executing them, using hierarchical reinforcement learning. We also show how complex tasks can be decomposed into a series of instructions and executions.

Executing natural language instructions has seen much attention. The task of grounding language into an executable representation is sometimes called semantic parsing (Zettlemoyer & Collins, 2007), and has been applied to navigational instruction following, e.g. Artzi & Zettlemoyer (2013). More recently, neural models instruction following have been developed for a variety of domains, for example Mei et al. (2016) and Hermann et al. (2017)

Instruction generation has been studied as a separate task. Daniele et al. (2017) map navigational paths onto instructions. Fried et al. (2017) generate instructions for complex tasks that humans can follow Fried et al. (2018) train a model for instruction generation, which is used both for data augmentation and for pragmatic inference when following human-generated instructions. We build on this work by also generating instructions at test time, and showing that latent language improves performance.

Learning to play a complete real-time strategy game, includ-

ing unit building, resources gathering, defence, invasion, scouting, and expansion, remains a hard problem in RL (Ontanón et al., 2013), in particular due to the complexity and variations of commercially successful games (e.g., StarCraft I/II), and its demand of computational resources. Traditional approaches focus on sub-tasks with hand-crafted features and value functions (e.g., building orders (Churchill & Buro, 2011), spatial placement of building (Certicky, 2013), attack tactics between two groups of units (Churchill et al., 2012), etc). Inspired by the recent success of deep reinforcement learning, more works focus on training a neural network to finish sub-tasks (Usunier et al., 2017; Peng et al., 2017), some with strong computational requirement (Zambaldi et al., 2018). For full games, (Tian et al., 2017) shows that it is possible to train an end-to-end agent on a small-scaled RTS game with predefined macro actions, and TStarBot (Sun et al., 2018) applies this idea to StarCraft II and shows that the resulting agent can beat carefully-designed, and even cheating rule-based AI. By using human demonstrations, we hand crafting macro-actions.

Learning an end-to-end agent that plays RTS games with unit-level actions is even harder. Progress is reported for MOBA games, a sub-genre of RTS games with fewer units—for example, OpenAI (2018) shows that achieving professional level of playing DoTA2 is possible with massive computation, and Wu et al. (2018) shows that with supervised pre-training on unit actions, and hierarchical macro strategies, a learned agent on Honor of Kings is on par with a top 1% human player.

Multi-agent frameworks have been used in RTS Games (Peng et al., 2017; Hagelbäck & Johansson, 2008; Foerster et al., 2017), mostly in finding tactics in local battles. For full-games, a large hierarchical command system is often built manually (Pérez & Villar, 2011; Synnaeve et al., 2016) to handle different situations in the game. In this paper, we choose to use a single-agent framework for our RTS AI with neural architecture.

8. Conclusion

We have introduced a framework for decomposing complex tasks into steps of planning and execution, connected with a natural language interface. We experimented with this approach on a new strategy game which is simple to learn but features challenging strategic decision making. We collected a large dataset of human instruction generations and executions, and trained models to imitate each role. Results show that exploiting the compositional structure of natural language improves generalization, and that RL can substantially improve instruction generation.

References

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, pp. 166–175, 2017a.

Andreas, J., Klein, D., and Levine, S. Modular multitask reinforcement learning with policy sketches. *ICML*, 2017b.

Artzi, Y. and Zettlemoyer, L. Weakly supervised learning of semantic parsers for mapping instructions to actions. *Transactions of the Association of Computational Linguistics*, 1:49–62, 2013.

Aytar, Y., Pfaff, T., Budden, D., Paine, T. L., Wang, Z., and de Freitas, N. Playing hard exploration games by watching youtube. *arXiv preprint arXiv:1805.11592*, 2018.

Bacon, P., Harb, J., and Precup, D. The option-critic architecture. *CoRR*, abs/1609.05140, 2016.

Certicky, M. Implementing a wall-in building placement in starcraft with declarative programming. *arXiv preprint arXiv:1306.4460*, 2013.

Churchill, D. and Buro, M. Build order optimization in starcraft. In *AIIDE*, pp. 14–19, 2011.

Churchill, D., Saffidine, A., and Buro, M. Fast heuristic search for rts game combat scenarios. In *AIIDE*, pp. 112–117, 2012.

Daniele, A. F., Bansal, M., and Walter, M. R. Navigational instruction generation as inverse reinforcement learning with neural machine translation. In *Proceedings of the 2017 ACM/IEEE International Conference on Human-Robot Interaction*, pp. 109–118. ACM, 2017.

Eysenbach, B., Gupta, A., Ibarz, J., and Levine, S. Diversity is all you need: Learning skills without a reward function. *arXiv preprint arXiv:1802.06070*, 2018.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H., Kohli, P., and Whiteson, S. Stabilising experience replay for deep multi-agent reinforcement learning. *International Conference on Machine Learning*, 2017.

Fried, D., Andreas, J., and Klein, D. Unified pragmatic models for generating and following instructions. *arXiv preprint arXiv:1711.04987*, 2017.

Fried, D., Hu, R., Cirik, V., Rohrbach, A., Andreas, J., Morency, L.-P., Berg-Kirkpatrick, T., Saenko, K., Klein, D., and Darrell, T. Speaker-follower models for vision-and-language navigation. *arXiv preprint arXiv:1806.02724*, 2018.

Hagelbäck, J. and Johansson, S. J. Using multi-agent potential fields in real-time strategy games. In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2*, pp. 631–638. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

Hermann, K. M., Hill, F., Green, S., Wang, F., Faulkner, R., Soyer, H., Szepesvari, D., Czarnecki, W. M., Jaderberg, M., Teplyashin, D., et al. Grounded language learning in a simulated 3d world. *arXiv preprint arXiv:1706.06551*, 2017.

Jaderberg, M., Mnih, V., Czarnecki, W. M., Schaul, T., Leibo, J. Z., Silver, D., and Kavukcuoglu, K. Reinforcement learning with unsupervised auxiliary tasks. *arXiv preprint arXiv:1611.05397*, 2016.

Mei, H., Bansal, M., and Walter, M. R. Listen, attend, and walk: Neural mapping of navigational instructions to action sequences. In *AAAI*, volume 1, pp. 2, 2016.

Miller, A. H., Feng, W., Fisch, A., Lu, J., Batra, D., Bordes, A., Parikh, D., and Weston, J. Parlai: A dialog research software platform. *arXiv preprint arXiv:1705.06476*, 2017.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. Playing atari with deep reinforcement learning. *CoRR*, 2013.

Ng, A. Y., Harada, D., and Russell, S. Policy invariance under reward transformations: Theory and application to reward shaping. In *ICML*, volume 99, pp. 278–287, 1999.

Oh, J., Singh, S., Lee, H., and Kohli, P. Zero-shot task generalization with multi-task deep reinforcement learning. *ICML*, 2017.

Ontanón, S., Synnaeve, G., Uriarte, A., Richoux, F., Churchill, D., and Preuss, M. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

OpenAI. Openai five. https://blog.openai.com/openai-five/, 2018.

Pathak, D., Agrawal, P., Efros, A. A., and Darrell, T. Curiosity-driven exploration by self-supervised prediction. In *International Conference on Machine Learning (ICML)*, volume 2017, 2017.

Peng, P., Yuan, Q., Wen, Y., Yang, Y., Tang, Z., Long, H., and Wang, J. Multiagent bidirectionally-coordinated nets for learning to play starcraft combat games. *CoRR*, abs/1703.10069, 2017. URL http://arxiv.org/abs/1703.10069.

Pérez, A. U. and Villar, S. Multi-reactive planning for real-time strategy games. *M. Sc. Report. Universit Autònoma de Barcelona, Spain*, 2011.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *CoRR*, abs/1707.06347, 2017.

Sukhbaatar, S., Szlam, A., Synnaeve, G., Chintala, S., and Fergus, R. Mazebase: A sandbox for learning from games. *CoRR*, abs/1511.07401, 2015.

Sun, P., Sun, X., Han, L., Xiong, J., Wang, Q., Li, B., Zheng, Y., Liu, J., Liu, Y., Liu, H., et al. Tstarbots: Defeating the cheating level builtin ai in starcraft ii in the full game. *arXiv preprint arXiv:1809.07193*, 2018.

Sutton, R. S., Precup, D., and Singh, S. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artif. Intell.*, 112(1-2), August 1999.

Synnaeve, G., Nardelli, N., Auvolat, A., Chintala, S., Lacroix, T., Lin, Z., Richoux, F., and Usunier, N. Torchcraft: a library for machine learning research on real-time strategy games. *CoRR*, abs/1611.00625, 2016.

Tang, H., Houthooft, R., Foote, D., Stooke, A., Chen, O. X., Duan, Y., Schulman, J., DeTurck, F., and Abbeel, P. # exploration: A study of count-based exploration for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2753–2762, 2017.

Tessler, C., Givony, S., Zahavy, T., Mankowitz, D. J., and Mannor, S. A deep hierarchical approach to lifelong learning in minecraft. *CoRR*, abs/1604.07255, 2016.

Tian, Y., Gong, Q., Shang, W., Wu, Y., and Zitnick, C. L. Elf: An extensive, lightweight and flexible research platform for real-time strategy games. In *Advances in Neural Information Processing Systems*, pp. 2659–2669, 2017.

Usunier, N., Synnaeve, G., Lin, Z., and Chintala, S. Episodic exploration for deep deterministic policies: An application to starcraft micromanagement tasks. *ICLR*, 2017.

Vinyals, O., Ewalds, T., Bartunov, S., Georgiev, P., Vezhnevets, A. S., Yeo, M., Makhzani, A., Küttler, H., Agapiou, J., Schrittwieser, J., Quan, J., Gaffney, S., Petersen, S., Simonyan, K., Schaul, T., van Hasselt, H., Silver, D., Lillicrap, T. P., Calderone, K., Keet, P., Brunasso, A., Lawrence, D., Ekermo, A., Repp, J., and Tsing, R. Starcraft II: A new challenge for reinforcement learning. *CoRR*, abs/1708.04782, 2017.

Wu, B., Fu, Q., Liang, J., Qu, P., Li, X., Wang, L., Liu, W., Yang, W., and Liu, Y. Hierarchical macro strategy model for moba game ai. *arXiv preprint arXiv:1812.07887*, 2018.

Zambaldi, V., Raposo, D., Santoro, A., Bapst, V., Li, Y., Babuschkin, I., Tuyls, K., Reichert, D., Lillicrap, T., Lockhart, E., et al. Relational deep reinforcement learning. *arXiv preprint arXiv:1806.01830*, 2018.

Zettlemoyer, L. and Collins, M. Online learning of relaxed ccg grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, 2007.