ACTRCE: AUGMENTING EXPERIENCE VIA TEACHER'S ADVICE

Anonymous authors
Paper under double-blind review

ABSTRACT

Sparse reward is one of the most challenging problems in reinforcement learning (RL). Hindsight Experience Replay (HER) attempts to address this issue by converting a failure experience to a successful one by relabeling the goals. Despite its effectiveness, HER has limited applicability because it lacks a compact and universal goal representation. We present Augmenting experienCe via TeacheR's adviCE (ACTRCE), an efficient reinforcement learning technique that extends the HER framework using natural language as the goal representation. We first analyze the differences among goal representation, and show that ACTRCE can efficiently solve difficult reinforcement learning problems in challenging 3D navigation tasks, whereas HER with non-language goal representation failed to learn. We also show that with language goal representations, the agent can generalize to unseen instructions, and even generalize to instructions with *unseen lexicons*. We further demonstrate it is crucial to use hindsight advice to solve challenging tasks, but we also found that little amount of hindsight advice is sufficient for the learning to take off, showing the practical aspect of the method.

1 INTRODUCTION

Many impressive deep reinforcement learning (deep RL) applications rely on carefully-crafted reward functions to encourage the desired behavior. However, designing a good reward function is non-trivial (Ng et al., 1999), and requires a significant engineering effort. For example, even for the seemingly simple task of stacking Lego blocks, Popov et al. (2017) needed 5 complicated reward terms with different importance weights. Moreover, handcrafted reward shaping (Gullapalli & Barto, 1992) can lead to biased learning, which may cause the agent to learn unexpected and undesired behaviors (Clark & Amodei, 2016).

One approach to avoid defining a complicated reward function is to use a *sparse and binary* reward function, i.e., give only a positive or negative reward at the terminal state, depending on the success of the task. However, the sparse reward makes learning difficult.

Hindsight Experience Replay (HER) (Andrychowicz et al., 2017) attempts to address this issue. The main idea of HER is to utilize failed experiences by substituting with a fake goal in order to convert them to successful experiences. For their algorithm to work, Andrychowicz et al. made the non-trivial assumption that for every state in the environment, there exists a goal which is achieved in that state. As the authors pointed out, this assumption can be trivially satisfied by choosing the goal space to be the state space. However, representing the goal using the enormous state space is very inefficient and contains much redundant information. For example, if we want to ask an agent to avoid collisions while driving where the state is the raw pixel value from the camera, then there can be many states (i.e. frames) that achieve this goal. It is redundant to represent the goal using each state.

Therefore, we need a goal representation that is (1) expressive and flexible enough to satisfy the assumption in HER, while also being (2) compact and informative where similar goals are represented using similar features. Natural language representation of goals satisfies both requirements. First, language can flexibly describe goals across tasks and environments. Second, language representation is abstract, hence able to compress any redundant information in the states. Recall the previous driving example, for which we can simply describe "avoid collisions" to represent all states that satisfy this goal. Moreover, the compositional nature of language provides transferable features for generalizing across goals.

In this paper, we combine the HER framework with natural language goal representation, and propose an efficient technique called Augmenting experienCe via TeacheR's adviCE (ACTRCE; pronounced "actress") to a broad range of reinforcement learning problems. Our method works as follows. Whenever an agent finishes an episode, a teacher gives advice in natural language to the agent based on the episode. The agent takes the advice to form a new experience with a corresponding reward, alleviating the sparse reward problem. For example, a teacher can describe what the agent has achieved in the episode, and the agent can replace the original goal with the advice and a reward of 1. We show many benefits brought by language goal representation when combining with hindsight advice. The agent can efficiently solve reinforcement learning problems in challenging 2D and 3D environments; it can generalize to unseen instructions, and even generalize to instruction with *unseen lexicons*. We further demonstrate it is crucial to use hindsight advice to solve challenging tasks, but we also found that little amount of hindsight advice is sufficient for the learning to take off, showing the practical aspect of the method.

We note that our work is also interesting from a language learning perspective. Learning to achieve goals described in natural language is part of a class of problem called language grounding (Harnad, 1990), which has recently received increasing interest, as grounding is believed to be necessary for more general understanding of natural language. Early attempts to ground language in a simulated physical world (Winograd, 1972; Siskind, 1994) consisted of hard coded rules which could not scale beyond their original domain. Recent work has been using reinforcement learning techniques to address this problem (Hermann et al., 2017; Chaplot et al., 2017b). Our work combines reinforcement learning and rich language advice, providing an efficient technique for language grounding.

2 BACKGROUND

2.1 REINFORCEMENT LEARNING AND DEEP Q-NETWORKS

We first review the traditional reinforcement learning setting, where an agent interacts with an infinite-horizon, discounted Markov Decision Process $(\mathcal{S}, \mathcal{A}, \gamma, P, r)$. Here, $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, γ the discount factor, P the transition model and r is a reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. We consider a policy $\pi_\theta(a|s)$ parameterized by θ. At time t, the agent chooses an action $a_t \in \mathcal{A}$ according to its policy $\pi_\theta(a|s_t)$ where $s_t \in \mathcal{S}$ is the current state. The environment in turn produces a reward $r(s_t, a_t)$ and transitions to the next state s_{t+1} according to the transition probability $P(s_{t+1}|s_t, a_t)$. The goal of the agent is to maximize the expected γ-discounted cumulative return $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t)]$ with respect to the policy parameters θ. The action-value function is denoted as $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[\sum_{i=t}^{\infty} \gamma^{i-t} r(s_i, a_i)|s = s_t, a = a_t]$, which is the expected discounted sum of rewards obtained by performing an action a at state s and following the policy π thereafter. We denote π^* as the *optimal policy* such that $Q^{\pi^*}(s, a) \geq Q^\pi(s, a)$ for every $s \in \mathcal{S}$, $a \in \mathcal{A}$ and policy π. The *optimal Q-function* $Q^* = Q^{\pi^*}$ satisfies the *Bellman equation*:

$$Q^*(s, a) = \mathbb{E}_{s' \sim p(\cdot|s,a)} \left[r(s, a) + \gamma \max_{a' \in \mathcal{A}} Q^*(s', a') \right] \tag{1}$$

Q-learning (Sutton & Barto, 2018) is an off-policy, model-free RL algorithm that is based on the Bellman equation. The algorithm uses semi-gradient descent (Sutton & Barto, 2018) to minimize the squared Temporal Difference (TD) error: $\mathcal{L} = \mathbb{E}(Q(s_t, a_t) - y_t)^2$, where the $y_t = r_t + \gamma \max_{a' \in \mathcal{A}} Q(s_{t+1}, a')$ is the *TD target*. Deep Q-Network (Mnih et al., 2013) builds on the Q-learning algorithm and uses a neural network to approximate Q^*. It also uses a replay buffer as well as a target network to improve training stability.

2.2 GOAL ORIENTED REINFORCEMENT LEARNING

We review the goal-oriented reinforcement learning framework following Schaul et al. (2015). We augment the previously defined infinite-horizon, discounted Markov Decision Process with a goal space $\mathcal{G}$. A goal g is chosen from $\mathcal{G}$ and stays fixed for each episode. The goal induces a reward function $r_g : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, that assigns reward to a state conditioned on the given goal. At time step t, the agent $\pi(a_t|s_t, g)$ chooses an action conditioned on the current state s_t, as well as the current goal g. The agent's objective is to maximize the expected discounted cumulative return given the goal, i.e., $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, a_t, g)]$.

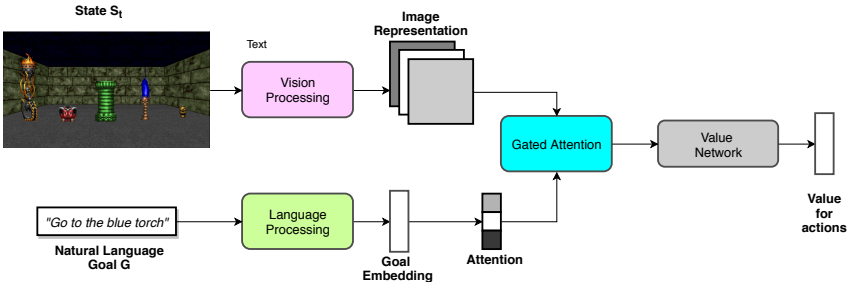


Figure 1: The diagram illustrates our model architecture.

2.3 HINDSIGHT EXPERIENCE REPLAY (HER)

We follow Andrychowicz et al. (2017) and consider a specific family of goal conditioned reward functions, where the reward is either 0 or 1, depending on the resulting state, $r_g(s_t, a_t) = f_g(s_{t+1})$, where $f_g : \mathcal{S} \to \{0, 1\}$. In many scenarios, the function f_g acts as a predicate that determines whether the agent is successful or not according to g, and only assigns positive reward at the terminal state. Hence the reward function is very sparse and makes it difficult for an agent to learn.

HER proposes a solution to this problem. The agent first collects an episode of experiences $s_0, a_0, r_1, ..., s_{T-1}, a_{T-1}, r_T, s_T$ under the goal g, where $r_T = r_g(s_{T-1}, a_{T-1})$ is the terminal reward. If r_T is zero, one can replace g with $g' \in \mathcal{G}$ such that $r_{g'}(s_{T-1}, a_{T-1}) = 1$. With an off-policy algorithm, one is able to learn from the goal transformed experience. In order to flexibly relabel with a desirable goal, the authors assume that there exists a representation map from the state space to the goal space $\mathbb{T} : \mathcal{S} \to \mathcal{G}$, so that for each $s \in \mathcal{S}$, the corresponding goal representation satisfies the predicate $f_g(\mathbb{T}(s)) = 1$. However, such mapping is not simple to construct. Although the author pointed out a trivial construction can be done by taking $\mathcal{G} = \mathcal{S}$ and $f_g(s) = [g = s]$, such goal representation is redundant and uninformative, and largely limits the applicability of the algorithm. Consider an example where the state is the first-person raw pixel observation in a 3-D environment, and the goal is to walk towards a particular object. There are many possible directions to approach the object from, which results in many different possible states that satisfy the goal. However, no subset of the raw pixel observation can abstractly represent the goal.

3 AUGMENTING EXPERIENCE VIA TEACHER'S ADVICE (ACTRCE)

For a goal oriented MDP, $\{\mathcal{S}, \mathcal{G}, \mathcal{A}, \gamma, P, r_g\}$, and a parameterized agent $\pi(a_t|s_t, g)$, our objective is to maximize the expected discounted cumulative return, i.e., $\mathbb{E}_\pi[\sum_{t=0}^{\infty} \gamma^t r_g(s_t, a_t)]$ for each $g \in \mathcal{G}$. Following HER, we assume there exists one or more more goal representation mappings $\mathbb{T} : \mathcal{S} \to \mathcal{G}$, which describe what goal is satisfied at each state. For example, in the original HER paper, the mapping $\mathbb{T}$ is simply a subspace projection from the state. Such representation can work well for their robotic tasks, where subset of the state (i.e., the coordinates of objects and body parts) can reasonably well represent the goal. However, in general, such mapping can not abstract meaningful information about the goal, and will cause redundancy.

Language provides an abstract representation for a goal and hence reduces redundancy in representation. This motivates our proposal to use *natural language* for representing the goal space. Concretely, for each state $s \in \mathcal{S}$, we define $\mathbb{T}$ as a teacher that gives an advice $\mathbb{T}(s)$, a natural language description of the goal that is achieved in the state s. To implement such a teacher, one can ask a human to look at a state and describe the goal in words, or generate a goal description automatically, as part of a narration or accessibility feature in a game. Given $\mathbb{T}$, we can convert a failure trajectory to a successful one by relabeling the original goal with the teacher advice. To illustrate, say the original goal for the episode is "Go to the armor", but at this particular time step, the agent has reached the blue torch instead. The teacher then tells the agent that the goal that reflects the current state is "Go to blue torch". The agent can hence take the advice and use it as a positive experience.

Besides positive reward signals, we also find that negative experience is necessary for training. Hence, in addition to a teacher who describes the achieved goal with a positive reward, we also introduce a teacher who gives negative reward with a description of a goal that has not been achieved. We further consider the scenario where there is more than one goal that can be satisfied in a state, e.g.,

Algorithm 1 Augmenting Experience via Teacher's Advice (ACTRCE)

Given:
- an off-policy RL algorithm $\mathbb{A}$, and replay buffer R. ▷ e.g. DQN, DDPG, NAF, SDQN
- A language goal space $\mathcal{G}$ and a desired goal space $\mathcal{G}_d$.
- A group of teachers $\{\mathbb{T}\}$

for episode = 1, M **do**
  Sample a desirable goal description $g \in \mathcal{G}_d$ and an initial state s_0.
  for $t = 0, T-1$ **do**
    Sample an action a_t using the behavioral policy from $\mathbb{A}$:
      $a_t \leftarrow \pi_b(s_t||g)$ ▷ $||$ denotes concatenation
  end for
  for every teacher $\mathbb{T}$ **do**
    Compute advice $g' = \mathbb{T}(s_T)$. ▷ Teacher's advice in natural language
    for $t = 0, T-1$ **do**
      $r = r_{g'}(s_t, a_t)$
      Store transitions $\{(s_t||g',\ a_t,\ r,\ s_{t+1}||g')\}$ in R ▷ HER
    end for
  end for
  Perform training with $\mathbb{A}$ and a minibatch B from the replay buffer.
end for

the state that is described as "Reach a blue torch" may also be described as "Reach the largest blue object". Each different description of the goal corresponds to a different teacher (see more discussions and experiments on variations of teachers in Appendix E). Therefore, in general, there is a group of teachers of different kinds, each giving different advice. We then relabel the original goal with each advice and corresponding positive or negative reward, and augment the replay buffer with these trajectories. Algorithm 1 describes our proposed approach formally.

Note that in the above formulation, we assume a MDP setting, and let the teacher give advice based solely on the terminal state $g' = \mathbb{T}(s_T)$. By contrast, in the POMDP setting, we ask the teacher to give advice based on the history of states and actions during the episode, i.e., $g' = \mathbb{T}(\{s_0, a_0, \ldots, s_T\})$.

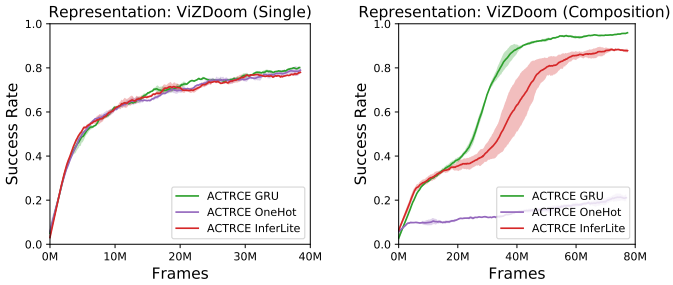
3.1 Language representation

There remains a question of how we deal with the natural language goal representation, a sequence of discrete symbols. In this paper, we explore two standard ways to convert a language goal into a continuous vector representation for neural network training. One way is to represent each word as a one-hot vector, and use a recurrent neural network (RNN) to sequentially embed each word into its hidden state. We can then obtain some function of its hidden states (e.g., last hidden state, an attention over all hidden states) as the representation of the goal. The other way is to use a pre-trained language component obtained by other approaches and dataset (e.g. Mikolov et al. (2013)) to represent the given language instructions. Since a pre-trained sentence embedding defines a reasonable similarity metric among sentences, one can expect the agent to understand *unseen lexicons* that are closely related to the language goals in training. This allows the agent to gain better natural language understanding, and potentially be more robust to the language advice it gets from teachers (advice from humans can be quite noisy).

To integrate language representation of goals into the model, we design the architecture as follows. The architecture consists of 3 modules. One language component that takes an instruction and convert it into a continuous vector, and we apply an attention vector. The second component processes the observation to obtain a image representation using convolution neural networks. We then use gated attention from Chaplot et al. (2017b) to fuse goal information and the observation. The third component then take the result of the fused representation and compute the output value. Figure.(1) shows the computational graph.

4 Experiments

In the following experiments, we demonstrate the effectiveness of the proposed method and discuss qualitative differences from the baseline. We first describe the experimental set up with our 2 environments, KrazyGrid World and ViZDoom. In the subsequent sections, we investigate:



(a) (b)

Figure 2: Performance in average success rate during training, comparing between different sentence embedding methods for the single target and composition task on ViZDoom.

1. *A comprehensive comparison between goal representations in hindsight advice, generalization, and semantic similarities.* We compare different goal representations: a naive one hot vector for each instruction, a Gated Recurrent Unit (GRU) (Chung et al., 2014) that embeds the discrete language tokens, and a pre-trained word embedding. We show that as the number of instructions increases, the one hot approach does not scale as well as the GRU and pre-trained embedding. From visualizing the representation, we can see that structures emerge from GRU and pre-trained embeddings. We also demonstrate that pre-trained embedding representation can generalize to goals containing out of training vocabulary words.

2. *Does the hindsight language advice help with learning? How much advice do we need?* We show significant improvement in sample efficiency by using advice from teachers. In many of the challenging tasks, the agent cannot learn at all without teachers' advice. We also show that significant improvement can be achieved even if we provide limited amount of teachers' advice, showing low burden of the method in practice.

4.1 Environments and Training

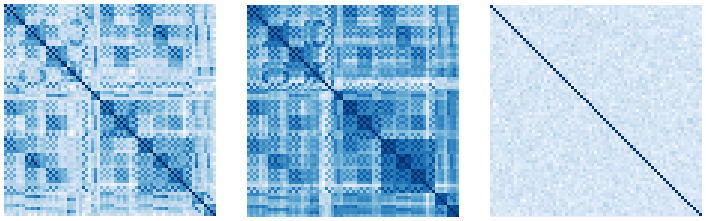
We tested our method in a 2D grid world (KrazyGrid World (Stadie et al., 2018)), as well as a 3D environment (ViZDoom (Chaplot et al., 2017b)). We describe the environment and our modifications below.

KrazyGrid World (KGW) (Stadie et al., 2018): KrazyGrid World is a 2D grid environment. For our experiments we chose to use 4 functionality of tiles: Goal, Lava, Normal, and Agent. We added an additional colour attribute in: Red, Blue, Green. The desired goals in this environment is to reach goals of different colours. Appendix A lists all language goals and additional environment details.

ViZDoom(Kempka et al., 2016; Chaplot et al., 2017b): The 3D learning environment is based on the first person shooter game Doom. The state is a raw pixel image from first person perspective of a room containing several random doom objects of various types, colours, and sizes. The goal for the episode is a natural language instruction in the form "Go to the [*target attribute*] [*object*]", such as "*Go to the green torch*". See Appendix B for list of instructions and more description of ViZDoom.

Compositions of goals. We refer language goals introduced in KGW and ViZDoom sections as singleton tasks. We further consider a set of more challenging tasks – tasks that are composed of singleton tasks. Given two singleton tasks A, B, we take composition function "and" and "or" to form two new tasks "A and B", and "A or B". The task "A and B" is considered as complete when the agent completes both A and B within an episode, and "A or B" is considered as complete when one of the task is achieved. In our experiments, we consider all combinations of singleton tasks with both compositions "and" and "or" for KGW, and "and" for ViZDoom.

Training details. We used the DQN algorithm as the base reinforcement learning algorithm in both environments. A detailed architecture description for all of our experienments can be found in Appendix C. We carried out multi environment training as follows. First we sample 16 different environments. We collected data from all 16 environments. We updated the agents with an average gradient. We then resampled 16 environments. Further training details can be found in Appendix D.



(a) GRU Embedding (b) InferLite (c)One-hot Embedding

Figure 3: Comparison among different sentence embedding methods showing the pairwise correlation between the sentence embedding vectors for each of the singleton instructions. The darker the colour, the higher the correlation.

4.2 Investigating different types of goal representation

In this section, we investigate the hypothesis that using language for goal representation helps learning in various aspects. We compare language-based goal representations and a non-language goal representation. In summary, we show that when the difficulty of the tasks increases, language goal representations became more effective in providing learning signals. We also show how one can use language goal representation to generalize to unseen goals in training, whereas non-language goal representation could not. With a pre-trained sentence embedding, we can even generalize to instructions consisting of *unseen lexicons*, showing the robustness of the language goal representation. Three goal representations are described as follows,

Language Sentence Representation with GRUs: For each instruction, we represent each word as a one hot vector, and sequentially embed words into a GRU. We use take the last hidden state representation of the GRU for representing the instruction. We use the same GRU architecture from Chaplot et al. (2017b), with hidden size 256.

Pre-trained Language Sentence Representation: We also consider models where the language component is pre-trained. We use *InferLite* (Kiros & Chan, 2018) as the pre-trained sentence representation and hold the parameters fixed during training. InferLite is a lightweight form of generic sentence encoder trained to perform natural language inference (Bowman et al., 2015; Williams et al., 2018) resembling InferSent (Conneau et al., 2017) but without the use of RNNs. The original sentence embedding vector is of dimension 4096, and we use a learned linear layer to project it down to 256 for keeping every other part of the model same as the other two.

One-Hot Representation: The non-language baseline represents each instruction as a one hot vector, which does not contain any language structure. We embed this one-hot goal representation to a vector with the same number of dimensions as the GRU representation. This embedding is learned independently for each goal encountered.

4.2.1 How does each goal representation perform?

We use all three goal representations with hindsight advice for learning both singleton tasks and compositional tasks of ViZDoom environment. The singleton tasks consists of 7 objects, and the compositional tasks consists of 5 objects. The result of comparisons is plotted in Figure 2. We see that in an easier benchmark—singleton—tasks, agents using one-hot goal representations are still able to learn as quickly as agents using the other two goal representations. However, in a much more challenging benchmark— compositional–tasks, the one-hot representations can only achieve a 24% success rates, whereas ACTRCE with language representations can achieve 97%. Since one hot representations represent each goal independently, the agent is unable to generalize to any new unseen instructions. With language representation, we can easily generalize to *unseen instructions* that consists of *seen lexicons*. We reported the generalization results in Table 1, under "ZSL" (zero-shot learning). Remarkably, with GRU language goal representation, the agent is able to generalize to unseen instructions 83% of the time, showing great generalization ability of language goal representations.

4.2.2 Visualization of learned embeddings

We carried out a visualization analysis to see the statistical relations between the learned embeddings of goals. We took the model trained in the singleton VizDoom benchmark, where the agent was able

Method	Single (7 objects)		Composition (5 objects)	
	MT	ZSL	MT	ZSL
DQN (GRU)	0.08 ± 0.01	0.065 ± 0.007	0.115 ± 0.007	0.07 ± 0.04
DQN (InferLite)	0.05 ± 0.07	0.04 ± 0.06	0.10 ± 0.02	0.13 ± 0.07
DQN (OneHot)	0.035 ± 0.007	-	0.02 ± 0.03	-
ACTRCE (GRU)	$\mathbf{0.80 \pm 0.06}$	$\mathbf{0.76 \pm 0.03}$	$\mathbf{0.97 \pm 0.04}$	$\mathbf{0.83 \pm 0.09}$
ACTRCE (InferLite)	$\mathbf{0.80 \pm 0.01}$	$\mathbf{0.76 \pm 0.02}$	0.88 ± 0.05	0.62 ± 0.01
ACTRCE (One Hot)	$\mathbf{0.80 \pm 0.03}$	-	0.24 ± 0.05	-

Table 1: The table shows the averaged success rates, calculated over 100 episodes, in multitask and zero-shot scenario for the model trained with DQN versus DQN with ACTRCE. The standard deviations are calculated across 2 seeds. In the Multitask (**MT**) scenario, the goal for the episode is sampled from the training set of instructions but with a random environment initialization. For Zero-Shot (**ZSL**), the instruction is sampled from the held out test instruction set and with a random environment initialization.

to learn with all three goal representations. However, we find there are huge differences in learned embeddings. For each goal instruction, we extract the corresponding learned embedding for all three goal representation, which is the continuous vector before the attention layer (recall Fig.(1)), all of size 256. We then calculated the correlation matrix for each goal representations. We plot all three correlation matrices in Figure.(3). The rows and columns are grouped by object type, then colour and size. We found that GRU and InferLite embeddings have very similar block-like structure, due to clustering of colours and shapes, while each one-hot goal embeddings share almost no correlation between each other. We further performed t-SNE (Maaten & Hinton, 2008) embeddings and observe meaningful clustering with language goal representations and sporadic embeddings for one-hot goals. See more details in Appendix K.1.

4.2.3 Generalizing to unseen lexicons with pre-trained word embedding

We now show how to use pre-trained embeddings to allow our model to generalize to unseen lexicons at test time through representation transfer, similar in spirit to DeViSE (Frome et al., 2013) that was used for image classification with unseen classes. We took the agent trained in singleton tasks with InferLite goal representations, and replaced one word in each original instruction with its synonym[1]. We evaluate the performance of the agent on these new instructions that contain unseen lexicons. The results are shown in Tab.(3). We find the agent is able to achieve tasks above 66% of time. The ability of understanding sentences of similar meanings become useful when one implements the method with advice comes from humans. Since humans can describe the same meaning in many different ways, understanding synonyms can improve the robustness of learning in general noisy settings.

Method	**MT + Synonym**	**ZSL + Synonym**
ACTRCE (InferLite)	$\mathbf{0.66 \pm 0.05}$	$\mathbf{0.62 \pm 0.02}$

Table 2: Multitask evaluation with synonym and Zero-shot with synonym using InferLite.

4.3 Do we need hindsight advice?

In the previous section, we demonstrate the effectiveness of language goal representation from various perspectives. In this section, we show how hindsight advice plays an important role in learning. We compared our method (denoted as "ACTRCE") to the algorithm DQN, the base algorithm without any hindsight language advice. We show that without hindsight advice, DQN is not able to learn many challenging tasks. However, we also found that little advice (1%) is sufficient for learning to take off, showing the practicality of our method. For language representation, we chose to use recurrent neural networks for embedding language goals, for both our method and the baseline method.

4.3.1 Comparison of ACTRCE to DQN

Singleton tasks. For experiments on KGW, we tried two different kind of grids, one with 3 lavas and the other with 6 lavas, and both with 3 goals of different colours. Fig. 4 (a) and (b) show the

[1]See Appendix B.4 for details on the synonyms used.

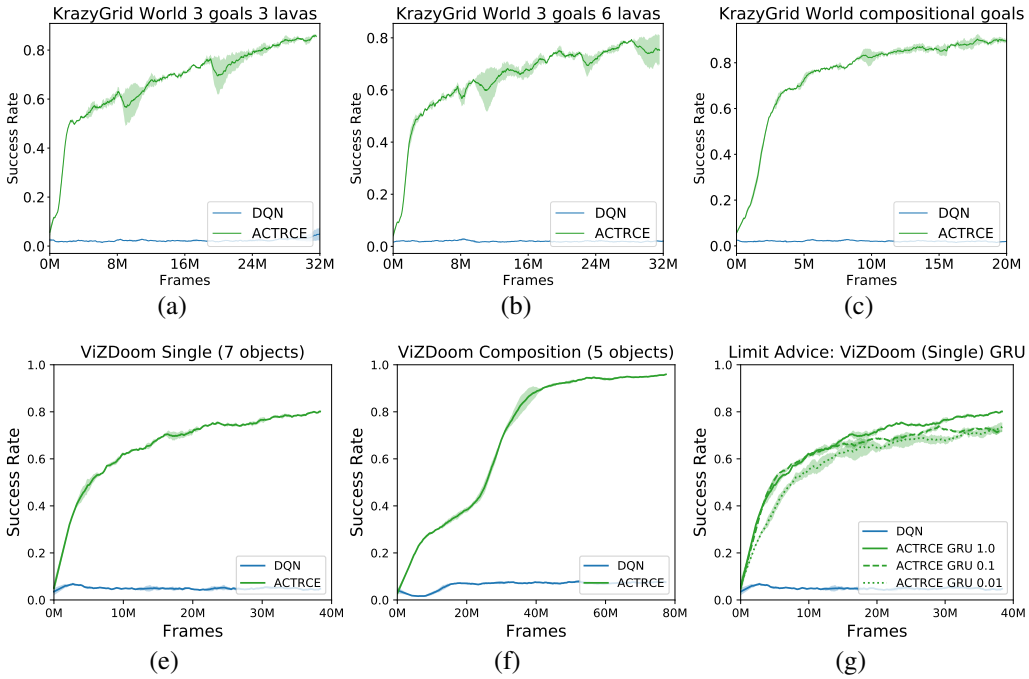


Figure 4: Performance comparisons on KGW and ViZDoom environments. The success rates are calculated over all desired goals and 16 different environments. Shaded area represents standard deviation 2 random seeds.

average success rate over all goals on 16-environments training. The shaded area represents the standard deviation over 2 random seeds. The baseline DQN is shown in blue curve, which failed to learn anything at all. Our method (shown in green) quickly learned and achieved good results on both environments (around 80% success rate). In ViZDoom experiments, we trained our agent in 3 configurations: 5 objects in (1) *easy* and (2) *hard* mode, and (3) 7-objects in the *hard* mode with 50% larger room size. While the DQN and A3C implementation from Chaplot et al. (2017a) were able to learn in the 5 objects task (see Appendix G for details), we found that in the more difficult 7-objects environment, only the agent trained with ACTRCE was able learn at all. Figure 4 (e) illustrates the training instruction average success rate (over 100 episode chunks) vs. the number of frames set, when using ACTRCE versus DQN baseline. Table 1 summarizes the agent's Multitask and Zero-Shot generalization performance.

Compositional tasks. In KGW, we carried out the experiments in a grid with 3 goals and 3 lavas. A comparison of average success rates over all goals of ACTRCE, and baseline DQN is shown in in Fig. 4 (C). The shaded area represents the standard deviation over 2 random seeds. We observed that the baseline still could not learn the task while ACTRCE learned efficiently and achieved good performance in this highly challenging environment. In ViZDoom, we chose to use 5 objects in *easy* mode for the compositional task, where all 5 objects appeared in front of the agent at the beginning of the episode. Figure 4 (b) shows the average success rate when using ACTRCE vs. baseline DQN. Likewise, our agent was able to learn very well with hindsight advice, whereas the baseline DQN failed to learn.

4.3.2 How much advice do we need?

We consider a variant of ACTRCE where the teacher provides hindsight advice only in the first $\{10\%, 1\%\}$ of frames during training, and stops giving any advice for the remaining portion of the training. We perform this experiment in the ViZDoom environment single target mode with 7 objects, using the GRU as the sentence embedding. Figure 4 illustrates the training average success rate over the frames, and Table 5 lists the final MT and ZSL performance on the trained agents. We observe that the agent is still able to learn comparably well even given very little (1%) advice, indicating of the method's robustness in practical settings.

5 Related Work

Several approaches have used natural language in the context of reinforcement learning. In pioneering work, Maclin & Shavlik (1994) translated language advice into a short program, implemented as a neural network. This advice network encouraged the policy to take the suggested action. Similarly, Kuhlmann et al. (2004) exploited natural language advice for a RL agent learning to play a (simulated) soccer game. In Ling & Fidler (2017), human feedback in the form of natural language was exploited to shape the reward of an image captioning RL agent. Kaplan et al. (2017) introduced an agent that learns to use English instructions to self-monitor its progress to beat Atari games. This was accomplished by implementing a multi-modal embedding of the pairs of game frames and instructions to determine if the instruction is satisfied, then provide additional reward to the agent. Andreas et al. (2017) proposed to use language as the latent parameter space for few-shot learning problems, including policy search.

The reverse has also been studied: using reinforcement learning to learn grounded language, referred to as task-oriented language grounding. Misra et al. (2017) mapped language instructions and visual observations to actions for manipulating blocks on a 2D plane. Yu et al. (2018) grounded language in 2D maze environment, via sentence-directed navigation and question answering task. They introduced the notion of extrapolation zero-shot sentence, where new words are transferred from other use cases and models. In their work, the unseen word in the navigation sentence was seen (transferred from) during the question answering task. In contrast, we transfer the unseen synonym words from a pre-trained sentence embedding InferLite model. Hermann et al. (2017) presented an agent that learns to execute written instructions in a 3D environment through reinforcement learning and unsupervised learning. Our work builds on Chaplot et al. (2017b) who proposed a gated-attention architecture for combining the language and image features to learn a policy that execute written instruction in a 3D environment.

Our work also relates to augmenting the experience using a experience replay buffer. Lin (1992) introduced experience replay (ER) to speed up the credit assignment process. The technique was later popularized by DQN (Mnih et al., 2013) for Atari games. Further works focus on sampling techniques from the replay buffer to accelerate learning. Hindsight Experience Replay (Andrychowicz et al., 2017) builds on ER by providing additional experiences in a goal conditional task. Our work extends this technique to use a natural language goal representation.

6 Conclusion

In this work, we propose the ACTRCE method that uses natural language as a goal representation for hindsight advice. The main point of the paper is to show that using language as goal representations can bring many benefits, when combined with hindsight advice. We analyzed the differences among goal representation, and show that ACTRCE can efficiently solve difficult reinforcement learning problems in challenging 3D navigation tasks, whereas HER with non-language goal representation failed to learn. We also show that with language goal representations, the agent can generalize to unseen instructions. With pre-trained language component, the agent can even generalize to instructions with *unseen lexicons*, demonstrating its potential to deal with noisy natural language advice from humans. Although ACTRCE algorithm crucially relies on hindsight advice, we showed that little amount of advice is sufficient for the learning to take off, showing its great practicality.

References

J. Andreas, D. Klein, and S. Levine. Learning with Latent Language. *ArXiv e-prints*, November 2017.

Marcin Andrychowicz, Dwight Crow, Alex K Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *NIPS*, 2017.

Samuel R Bowman, Gabor Angeli, Christopher Potts, and Christopher D Manning. A large annotated corpus for learning natural language inference. In *EMNLP*, 2015.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding (code). https://github.com/devendrachaplot/DeepRL-Grounding, 2017a.

Devendra Singh Chaplot, Kanthashree Mysore Sathyendra, Rama Kumar Pasumarthi, Dheeraj Rajagopal, and Ruslan Salakhutdinov. Gated-attention architectures for task-oriented language grounding. *arXiv preprint arXiv:1706.07230*, 2017b.

J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *ArXiv e-prints*, December 2014.

Jack Clark and Dario Amodei. Faulty reward functions in the wild, 2016. URL https://blog.openai.com/faulty-reward-functions/.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*, 2017.

Andrea Frome, Greg S Corrado, Jon Shlens, Samy Bengio, Jeff Dean, Tomas Mikolov, et al. Devise: A deep visual-semantic embedding model. In *NIPS*, 2013.

V. Gullapalli and A. G. Barto. Shaping as a method for accelerating reinforcement learning. In *Proceedings of the 1992 IEEE International Symposium on Intelligent Control*, pp. 554–559, Aug 1992. doi: 10.1109/ISIC.1992.225046.

Stevan Harnad. The symbol grounding problem. *Physica D: Nonlinear Phenomena*, 42(1):335 – 346, 1990. ISSN 0167-2789. doi: https://doi.org/10.1016/0167-2789(90)90087-6. URL http://www.sciencedirect.com/science/article/pii/0167278990900876.

Hado vang Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, pp. 2094–2100. AAAI Press, 2016. URL http://dl.acm.org/citation.cfm?id=3016100.3016191.

K. M. Hermann, F. Hill, S. Green, F. Wang, R. Faulkner, H. Soyer, D. Szepesvari, W. M. Czarnecki, M. Jaderberg, D. Teplyashin, M. Wainwright, C. Apps, D. Hassabis, and P. Blunsom. Grounded Language Learning in a Simulated 3D World. *ArXiv e-prints*, June 2017.

Russell Kaplan, Christopher Sauer, and Alexander Sosa. Beating atari with natural language guided reinforcement learning. *ArXiv e-prints*, 04 2017.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pp. 341–348, Santorini, Greece, Sep 2016. IEEE. URL http://arxiv.org/abs/1605.02097. The best paper award.

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

Jamie Ryan Kiros and William Chan. Inferlite: Simple universal sentence representations from natural language inference data. In *EMNLP*, 2018.

G. Kuhlmann, P. Stone, R. Mooney, and J. Shavlik. Guiding a reinforcement learner with natural language advice: Initial results in robocup soccer. In *AAAI Workshop on Supervisory Control of Learning and Adaptive Systems*, 2004.

Guillaume Lample. Arnold. https://github.com/glample/Arnold, 2017.

Guillaume Lample and Devendra Singh Chaplot. Playing fps games with deep reinforcement learning. In *Proceedings of AAAI*, 2017.

Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine Learning*, 8(3):293–321, May 1992. ISSN 1573-0565. doi: 10.1007/BF00992699. URL https://doi.org/10.1007/BF00992699.

Huan Ling and Sanja Fidler. Teaching machines to describe images via natural language feedback. In *NIPS*, 2017.

Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. In *EMNLP*, 2015.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne. In *JMLR*, 2008.

Richard Maclin and Jude W. Shavlik. Incorporating advice into agents that learn from reinforcements. In *National Conference on Artificial Intelligence*, pp. 694–699, 1994.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 26*, pp. 3111–3119. Curran Associates, Inc., 2013.

Dipendra Misra, John Langford, and Yoav Artzi. Mapping instructions and visual observations to actions with reinforcement learning. *EMNLP*, 2017.

V. Mnih, A. Puigdomènech Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu. Asynchronous Methods for Deep Reinforcement Learning. *ArXiv e-prints*, February 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

Andrew Y. Ng, Daishi Harada, and Stuart J. Russell. Policy invariance under reward transformations: Theory and application to reward shaping. In *Proceedings of the Sixteenth International Conference on Machine Learning*, ICML '99, pp. 278–287, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc. ISBN 1-55860-612-2. URL http://dl.acm.org/citation.cfm?id=645528.657613.

Ivaylo Popov, Nicolas Heess, Timothy P. Lillicrap, Roland Hafner, Gabriel Barth-Maron, Matej Vecerik, Thomas Lampe, Yuval Tassa, Tom Erez, and Martin A. Riedmiller. Data-efficient deep reinforcement learning for dexterous manipulation. *CoRR*, abs/1704.03073, 2017.

Tom Schaul, Dan Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, pp. 1312–1320. JMLR.org, 2015. URL http://dl.acm.org/citation.cfm?id=3045118.3045258.

Jeffrey Mark Siskind. Grounding language in perception. *Artificial Intelligence Review*, 8(5):371–391, Sep 1994. ISSN 1573-7462. doi: 10.1007/BF00849726. URL https://doi.org/10.1007/BF00849726.

Bradly Stadie, Ge Yang, Rein Houthooft, Xi Chen, Yan Duan, Yuhuai Wu, Pieter Abbeel, and Ilya Sutskever. Some considerations on learning to explore via meta-reinforcement learning. *ICLR*, 2018.

Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA, USA, 2nd edition, 2018. ISBN 9780262193986.

Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *NAACL*, 2018.

Terry Winograd. Understanding natural language. *Cognitive Psychology*, 3(1):1 – 191, 1972. ISSN 0010-0285. doi: https://doi.org/10.1016/0010-0285(72)90002-3. URL http://www.sciencedirect.com/science/article/pii/0010028572900023.

Haonan Yu, Haichao Zhang, and Wei Xu. Interactive grounded language acquisition and generalization in a 2d world. *ICLR*, 2018.

A KRAZYGRID WORLD LANGUAGE GOALS

A.1 MORE DETIALS ON THE ENVIRONMENT

KrazyGrid World is a 2D grid environment. For our experiments we chose to use 4 functionality of tiles: Goal, Lava, Normal, and Agent. We added an additional colour attribute ranging in 3 colours: Red, Blue, Green. There are 4 discrete actions, “up", “down", “right", “left". The desired goals in this environment is to reach goals of different colours. We use a global view of the grid state as the observation to the agent. Each tile in the grid is represented by a concatenation of its functionality and colour attribute, both in one hot vectors. The grid state also includes a one hot vector that represents the agent's position. In our experiments, we use an environment of grid size 9×9, with 3 goals in the environment, each has a distinct colour. We tried various number of lavas in the environment, and made sure there is at least 1 lava of each colour. In the simple task setting, the episode terminates when the agent reaches a lava or a goal, or the episode runs over a maximum time step $T = 25$. We automatically generates descriptions give any states of the environment as the teacher.

Compositional We enlarged the goal space by considering the task of compositions of goals. As the goal space changed, we also had to make several modifications to the environment. We let the episode terminate when the agent reaches a lava or reaches 2 different goals, or runs over the maximum time step of 50. Since we still included simple goals (e.g., “Reach blue goal") in the goal space, we added an extra action called “flag". This action can be used for the agent to terminate the episode when it thinks the given goal is accomplished. The environment will be terminated if the agent indeed has accomplished the given task, and will stay unchanged otherwise.

A.2 LIST OF LANGUAGE INSTURCTIONS

Single goal setting The entire goal space $\mathcal{G}$ consists of 8 goals: $\{$Reach red goal, Reach blue goal, Reach green goal, Reach red lava, Reach blue lava, Reach green lava, Avoid any lava, Avoid any goal$\}$. The desired goal space $\mathcal{G}_d$ consists of 3 goals $\{$Reach red goal, Reach blue goal, Reach green goal$\}$.

Compositional goal setting The desired goal space $\mathcal{G}_d$ consists of 21 goals $\{$Reach red goal, Reach blue goal, Reach green goal, Reach red goal and Reach blue goal, Reach blue goal and Reach red goal, Reach red goal or Reach blue goal, Reach blue goal or Reach red goal, Reach red goal and Reach green goal, Reach green goal and Reach red goal, Reach red goal or Reach green goal, Reach green goal or Reach red goal, Reach blue goal and Reach green goal, Reach green goal and Reach blue goal, Reach blue goal or Reach green goal, Reach green goal or Reach blue goal, Reach red goal and Avoid any lava, Avoid any lava and Reach red goal, Reach blue goal and Avoid any lava, Avoid any lava and Reach blue goal, Reach green goal and Avoid any lava, Avoid any lava and Reach green goal$\}$.

The entire goal space $\mathcal{G}$ has another of 17 goals: $\{$Reach red lava or Reach blue lava, Reach blue lava or Reach red lava, Reach red lava or Reach green lava, Reach green lava or Reach red lava, Reach blue lava or Reach green lava, Reach green lava or Reach blue lava, Reach red lava, Reach red lava and Avoid any goal, Avoid any goal and Reach red lava, Reach blue lava, Reach blue lava and Avoid any goal, Avoid any goal and Reach blue lava, Reach green lava, Reach green lava and Avoid any goal, Avoid any goal and Reach green lava, Avoid any lava and Avoid any goal, Avoid any goal and Avoid any lava$\}$.

B VIZDOOM ENVIRONMENT DETAIL

B.1 ENVIRONMENT DESCRIPTION

ViZDoom(Kempka et al., 2016; Chaplot et al., 2017b): The 3D learning environment is based on the first person shooter game Doom. At each time step, the state is a raw pixel image from first person perspective of the 3D environment. The environment consists of a room containing several random doom objects of various types, colours, and sizes. The agent can navigate the environment via 3 possible actions: turn left, turn right, and move forward. The goal for the episode is a natural language instruction in the form "Go to the [*target attribute*] [*object*]", such as "*Go to the green torch*". Only one target object is present in each episode. The episode terminates either when the

agent reaches an object (regardless of whether it is correct or not), or the maximum time step is reached ($T = 30$). A reward of 1 is given if the correct object is reached, otherwise a reward of 0 is given. The environment has several difficulty modes, depending on how the objects and the agent are distributed at the beginning of the episode. In the *easy* mode, the agent is spawned at a fixed location facing forward. The objects are placed evenly spaced apart along a horizontal row in front of the agent. In *hard*, both the objects and the agent are randomly spawned, and the objects are not necessarily in view. We focus on the easy and hard mode for our experiments.

B.2 ViZDoom Composition Instructions

The compositional instructions consist of two single object instructions from the original training set joined by the word "and", such as "*Go to the red torch and Go to the pillar*". We did not include any superlative instructions, such as "*Go to the largest object*".

We ensured that the desirable instructions were unambiguous–given two instructions, the set of objects satisfying the first instruction is mutually exclusive from the set of objects satisfied in the second instruction. For example, "Go to the torch and Go to the pillar" have mutually exclusive set of valid objects. An ambiguous compositional instruction may have objects that satisfy both instructions. For example, "Go to the blue object and Go to the torch" is ambiguous because a blue torch satisfies both instructions. This is illustrated in Figure 5.

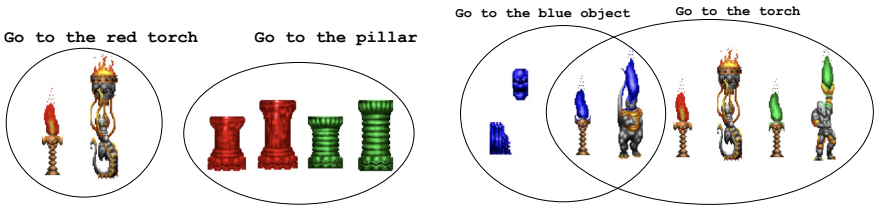


Figure 5: Example of (a) unambiguous composition of instructions and (b) ambiguous composition of instructions

B.3 ViZDoom Composition: Modification for State

For the ViZDoom composition task, we modify the raw pixel image of the environment to include a basic head-up display (HUD) consisting of black rectangle in the bottom left of the screen. When the agent reaches an object, a small thumbnail image of the reached object will appear inside the HUD. The HUD can show up to 2 objects reached, and the episode terminates once the again has reached a second object. Figure 6 illustrates what the agent sees as the input image throughout a composition task episode.

Figure 6: Example state input screens seen by the agent in a ViZDoom composition task episode. The composition instruction for this episode is "*Go to the green short torch and Go to the green short pillar*". (a) Shows the starting state. (b) is the state when the agent has reached the green short pillar. (c) is the final state once the agent reached the short green torch.

B.4 ViZDoom Synonym Instructions

We generate the synonym instructions by replacing an original word in the instruction with one of the synonyms listed in Table 3.

Original Word	Synonym
Go to	Reach, Approach, Get to
largest	biggest
smallest	littlest
small	tiny
large	big, huge
short	low, little
tall	big, towering
red	crimson
blue	indigo, teal
green	olive, lime
yellow	amber, gold, sunny
torch	lamp, beacon, lantern, flaming object
pillar	column, pedestal
skull	cranium, head
key	opener, unlocker
card	badge, pass
armor	shield
object	thing, item

Table 3: Original vocabulary and corresponding synonyms

B.5 VIZDOOM TEACHER ADVICE GENERATION

Singleton task. When the agent does reach an object (correct or incorrect), we generate a positive teacher advice by sampling from the training set of instructions that can describe the reached object. Additionally, we also generate a negative feedback by randomly picking one of the object in the current environment that was not reached, then sampling an instruction that described that unreached object, while ensuring that it will not describe the reached object.

When the agent did not reach any object, we simply do not give any positive teacher advice (i.e. the achieved goal). We had originally tried to give the advice of "*Reach no object*" when the agent did not reach any objects at the end of the episode, but it did not improve the performance. However, we still give the unachieved goal, from any of the objects in the current environment.

Compositional task. We follow a similar process for compositional task, depending on the number of objects reached. If the agent reached 0 or 1 object only, we generate a singleton advice in similar fashion as singleton task. We later found that not giving any advice when reaching 0/1 object also performed equally well and slightly faster learning. If the agent reached 2 objects during the episode, for positive advice we sample generate singleton instructions for each of the two objects, ensuring that they were not the same instructions, then merge them with "*and*" keyword. For negative advice, we generate singleton instructions for the unreached objects and combine two of them, ensuring that the instructions do not refer to the reached objects at all.

C ARCHITECTURES

KrazyGrid World

- **Singleton** We use series of convolution layers follows with ReLU activation functions for prepossessing the grid observation: 32 of 3x3 kernel with stride 1, followed by 64 of 3x3 kernel size with stride of 2 and then 64 of 3x3 kernel with a stride of 1 and 128 of 3x3 kernel with a stride of 2. We input the language sentences as word level one hot vectors to a LSTM of hidden size 128. The LSTM's last hidden vector is passed into a fully connected layer with 128 output units with sigmoid activation. This acts as the attention vector that is multiplied channel-wise with the 64 feature maps from the convolutional later. The gated feature maps are then flattened and passed through a fully connected layer with ReLU activation with 256 units. We then pass into a linear layer to predict the 4 action values.
- **Compositional** We use series of convolution layers follows with ReLU activation functions for prepossessing the grid observation: 32 of 1x1 kernel with stride 1, followed by 32 of 3x1 kernel size with stride of 2. We input the language sentences as word level one hot vectors to a bidirection LSTM, of hidden size 40. After obtaining the preprocessed observation, we augment with the history vector provided by the environment as a query to attend to all the hidden states in the Bi-direction LSTMs via Luong attention Luong et al. (2015). We then keep processing the observation using two more convolution layers follows with ReLU activation functions: 64 of 3x3 kernel with stride 2, followed by 64 of 3x3 kernel size with stride of 2. At each of these two layers, we use the context vector formed by language attention to attend back the observation by gating on 64 feature maps in a channel-wise fashion. The final gated feature maps are then flattened and passed through a fully connected layer with ReLU activation with 256 units. We then pass into a linear layer to predict the 5 action values.

ViZDooms Our architecture is almost identical to Chaplot et al. (2017b), except that we uses a linear output layer for the action values, and we did not use dueling architecture. The state input to the network is RGB image of size $3 \times 300 \times 168$. A series of convolution layers follows with ReLU activation functions: 128 of 8x8 kernel with stride 4, followed by 64 of 4x4 kernel size with stride of 2 and then 64 of 4x4 kernel with a stride of 2. To process the language input, we first use an embedding matrix to embed the voculary to a vector of size 32. We use the Gated Recurrent Unit (GRU) Chung et al. (2014) of size 256 as the recurrent cell to process word embeddings. The GRU's last hidden vector is passed into a fully connected layer with 64 output units with sigmoid activation. This acts as the attention vector that is multiplied channel-wise with the 64 feature maps from the convolutional later. The gated feature maps are then flattened and passed through a fully connected layer with ReLU activation with 256 units. We then pass into the LSTM with 256 hidden units, and its hidden units go through a linear layer to predict the 3 action values.

D TRAINING DETAILS

KrazyGrid World For KrazyGrid World experiments, we tune the following hyperparameters within corresponding range: learning rate $\{0.0003, 0.001, 0.003\}$, replay buffer size of $\{5000, 10000\}$, training the network every $\{1, 2, 4\}$ frames. We generate episodes with ϵ-greedy policy starting at $\epsilon = 1.0$ then decaying linearly to 0.01 by 10000 frames and remain at 0.01. We use Double DQN Hasselt et al. (2016) to reduce the Q-value overestimation and Huber loss ($\delta = 1$) for stable gradients.

ViZDoom For ViZDoom environment, we use the same set of 56 training instructions to train the agent, and a set of 15 held out test instructions for zero-shot evaluation as in Chaplot et al. (2017b). We also used their code Chaplot et al. (2017a) for reproducing training using Asynchronous Advantage Actor Critic (A3C) Mnih et al. (2016). We implemented our version of DQN on top of the implementation of Arnold Lample & Chaplot (2017); Lample (2017). The cyclic buffer replay buffer contains the last 10000 and 20000 most recent transitions for the easy and hard mode, respectively, chosen from the range $\{1000, 10000, 100000\}$ and fine tuned. We generate episodes with ϵ-greedy policy starting at $\epsilon = 1.0$ then decaying linearly to 0.01 by 10000 frames and remain at 0.01. We found that this performed better than using a larger $epsilon = 0.1$. Only after the first 1000 frames have been collected that the training begins, selected from a range of $\{1000, 10000, 100000\}$. We use Double DQN Hasselt et al. (2016) to reduce the Q-value overestimation and Huber loss ($\delta = 1$)

for stable gradients. We use the Adam optimizer Kingma & Ba (2014) with a learning rate of 0.0001. The network is updated every 4 frames on easy and 16 frames on difficult mode. While updating less often reduces sample efficiency, in practice we found that this speeds up training wall clock time and converges to a better performance. When sampling from the replay buffer, we sample the start a random episode and select 32 consecutive frames from the replay buffer. This approach leads to more accurate estimates of the hidden state of the LSTM. In addition, the sequential mini-batch allows us to perform n-step Q learning as outlined in Mnih et al. (2016). The correlation between samples in the mini-batch is alleviated by running 16 parallel threads to send gradient updates to the shared network parameters. We synchronize the training thread model with the shared network each time before computing the training loss and gradient. The target network is synchronized with the current model once every 500 time steps. and use one additional thread to evaluate the multi-task success rate throughout the training process.

E DIFFERENT TYPES OF TEACHERS

In this section we described the details of teacher types. Firstly, note that not all of the language descriptions describe favorable behaviors. For example, in the KGW, a desirable goal is "reach a goal", but there are also undesirable states that corresponds to goal "reach a lava". Hence we consider a subset $\mathcal{G}_d \subseteq \mathcal{G}$ to denote all desired goals that the agent is expected to perform. At each episode, we first sample a goal $g \in \mathcal{G}_d$ from the set of all desired language goal spaces, and ask the agent to explore the environments conditioned on the goal. When the episode ends, we obtained an advice from a teacher $\mathbb{T}$ by observing the terminal state, $g^* = \mathbb{T}(s_T)$. Depending on the kind of teacher, we obtain a different kind of advice or no advice. We consider three kinds of teacher as follows:

- **Optimistic** A teacher who gives advice only when the agent achieved a desirable goal, i.e., when $g^* \in \mathcal{G}_d$. When the agent performs poorly, there is no advice from this teacher.
- **Knowledgeable** A teacher who describes what the agent has achieved in all scenarios, including the undesirable behaviors, as advice to the agent.
- **Discouraging** A teacher who describes a desired goal $g^* \in \mathcal{G}_d$ that the agent has not achieved as advice to the agent. In this case, the trajectory with goal replaced by the teacher's advice will receive a reward of 0.

E.1 HOW DOES EACH TEACHER PERFORM?

In this section, we investigated how different teachers help in giving advice. We compared our method to the DQN algorithm. We denote the method using only optimistic and discouraging teachers as "ACTRCE$^-$". We denote the method using *knowledgeable* teachers as well as discouraging teachers as ACTRCE (this is the version we use in the main paper). We evaluated three methods in KrazyGrid World and the results are as follows.

We tried two different kind of grids, one with 3 lavas and the other with 6 lavas, and both with 3 goals of different colours. Fig. 4 (a) and (b) show the average success rate over all goals on 16-environments training. The shaded area represents the standard deviation over 2 random seeds. The baseline DQN is shown in blue curve, which failed to learn anything at all. "ACTRCE$^-$" (shown in orange) quickly learned and achieved good results on both environments. However, we observed that when the number of lavas increased, the task became harder, and ACTRCE$^-$ performed worse after 32 millions frames of training (the performance dropped from 83% to 63%). On the other hand, we observed that having knowledgeable teachers always helped speed up learning. In particular, when the number of lavas increased, the task became harder for ACTRCE$^-$. However, with knowledgeable teachers, language advice was provided even when the agent reached lava, and hence the amount of advice per step was kept the same in the more difficult setting. Therefore, we observed ACTRCE learning at a similar rate in the more difficult setting, leaving a big gap from ACTRCE$^-$ after 32 millions frames of training (80% vs 63%).

F CAN LEARNING EASY GOALS HELP LEARNING DIFFICULT GOALS?

As we mentioned, there are desirable tasks as well as undesirable tasks. One would worry what if the desirable tasks are very difficult, would the agent only learns to accomplish easier tasks which are undesirable? Hence, we would like to ask whether learning the easier tasks from hindsight can

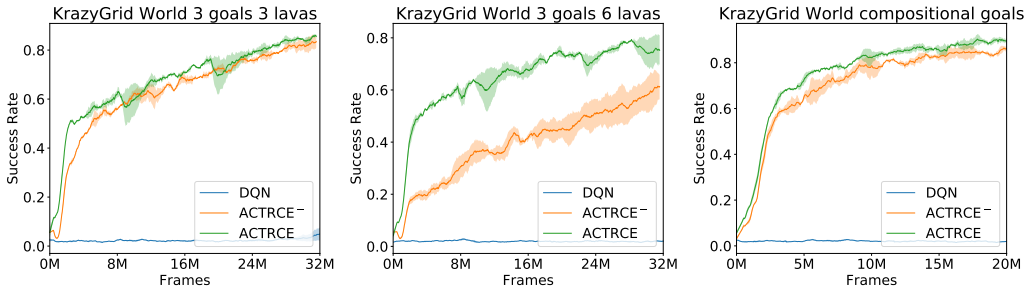


Figure 7: Performance comparisons on KrazyGrid World environments. The success rates are calculated over all desired goals and 16 different environments. Shaded area represents standard deviation 2 random seeds.

provide any learning signal for more difficult tasks. We hypothesize that it is possible in the KGW setting as learning to “reach lava" can aid learning controllers, which is used to “reach goals".

We designed the following transfer learning experiment to further our hypothesis. First, we artificially constructed a **pessimistic** teacher who only gives advice when the agent achieved undesirable goals, i.e., when $g^* \in \mathcal{G} \backslash \mathcal{G}_d$. We pretrained the agents with only the pessimistic teacher for 5 million frames. Now, we carried out training using ACTRCE$^-$ (with optimistic and discouraging teachers) with the pretrained agent and compared to unpretrained ones. Results are shown in Figure 8. We found that by pretraining the agent using pessimistic agents, even though the language goals in the pretraining phase have no overlapping with the actual training goals, the agent learned much faster than the unpretrained ones. In particular, in the environment with 3 lavas, the pretrained agent learned the fastest. In the environment with 6 lavas, pretrained agents learned as fast as ACTRCE, leaving a big gap from ACTRCE$^-$, even though it was given the same amount of advice during training as ACTRCE$^-$. This provides an evidence that learning easier goals can sometimes provide learning signals for harder goals, especially when both tasks require similar modules.

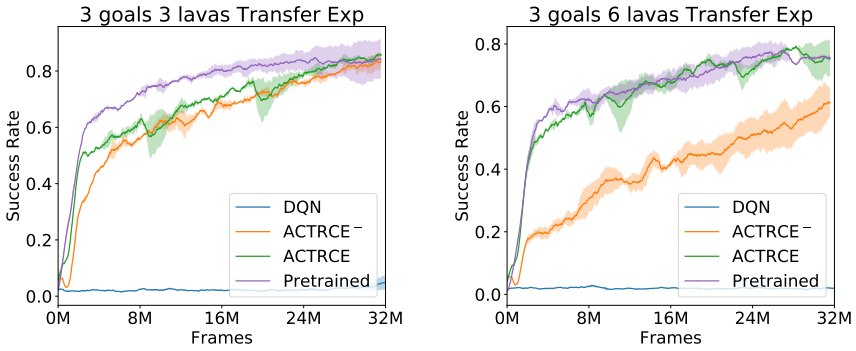


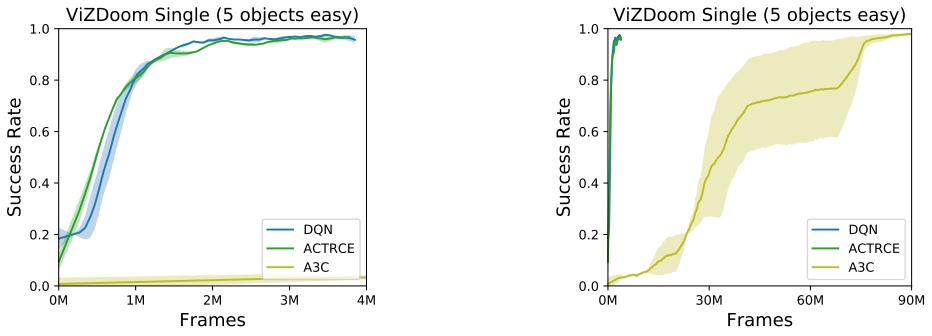
Figure 8: Transfer experiments that demonstrate pre-training with easier goals can aid learning more difficult goals.

G VIZDOOM 5 OBJECTS SINGLE TARGET EXPERIMENTS

In this section we show experiments done on ViZDoom with single target for 5 objects on easy and hard mode. On the easy task, the baseline DQN is able to consistently learn to solve the task, and with ACTRCE the performance was similar. However, as the task difficulty increased to the 5 objects on hard mode, we found that DQN's learning was less consistent across seeds, in contrast with ACTRCE's low variance across seeds. We also ran the A3C baseline from Chaplot et al. (2017b), from their implementation available online (Chaplot et al. (2017a)) using the hyperparameter settings stated in their paper. On the easy task, we were able to solve the task with A3C, but with an order of magnitude less sample efficiency compared to our DQN/ACTRCE implementation. On the hard task, we were unable to reproduce the results with our limited computational budget. A similar trend of an order of magnitude difference in sample efficiency is observed between A3C and DQN/ACTRCE on the hard task.

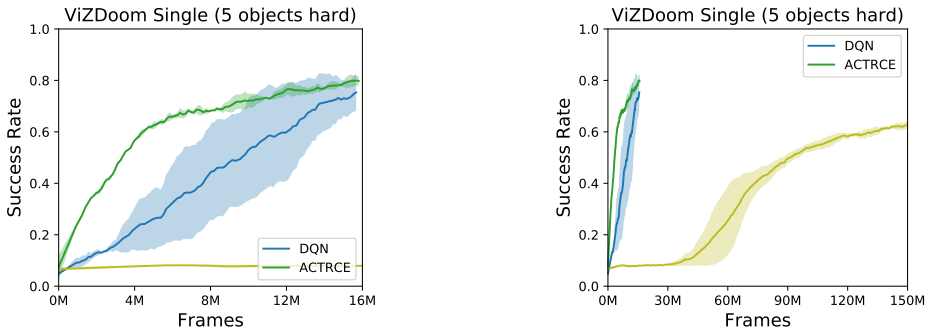
	MT				**ZSL**		
# of frames	8M	16M	150M	N/A	16M	150M	N/A
A3C [1]	-	-	-	0.83	-	-	0.73
A3C (Reprod.)	0.10 ± 0.01	0.09 ± 0.04	0.73 ± 0.01	-	-	0.71 ± 0.02	-
DQN	0.4 ± 0.2	0.73 ± 0.08	-	-	0.75 ± 0.05	-	-
ACTRCE	$\mathbf{0.69 \pm 0.04}$	$\mathbf{0.83 \pm 0.02}$	-	-	$\mathbf{0.77 \pm 0.02}$	-	-

Table 4: The table shows the averaged success rates in multitask and zero-shot scenario for the model trained with DQN versus DQN with ACTRCE, compared to A3C published and reproduced result, on ViZDoom single target 5 objects hard mode. The standard deviations are calculated across 2 seeds. MT was evaluated throughout the training process, while ZSL was evaluated at the end of training.



(a) (b)

Figure 9: ViZDoom experiment with 5 objects in easy mode for single target case.

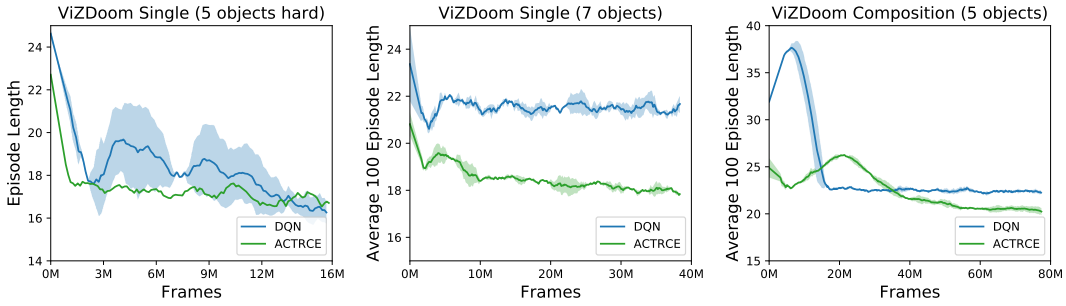


(a) (b)

Figure 10: ViZDoom experiment with 5 objects in hard mode for single target case.

H VIZDOOM AVERAGE EPISODE LENGTH

We report on the average episode length (averaged over 100 episodes) during training for 3 ViZDoom scenarios: single target 5 and 7 objects in hard mode, and composition target 5 objects in easy mode. The GRU hidden state representation of the sentence were used in the plots in Figure 11. We observe that the average episode length is slowly decreasing when using ACTRCE, while the baseline DQN remains fairly flat for the harder 7 objects (single target) and 5 objects (compositional).



(a) (b) (c)

Figure 11: Average training episode lengths in VizDoom environments. (a) 5 objects in hard mode, in single target case. (b) 7 objects in hard mode, in single target case. (c) 5 objects in easy mode, in composition case.

I VIZDOOM CUMULATIVE SUCCESS RATE VS. EPISODE LENGTH CURVE

We take the final trained model and run 100 episodes, noting whether each run was successful or not (binary value). We construct a *cumulative success rate* (CSR) versus episode length curve, where the x-axis is the episode length, and the y-axis is the fraction of total number of episodes that were successful and had episode length *less than or equal* to the x-axis value:

$$\text{CSR}(x) = \frac{\text{Number of successful episodes with episode length} \leq x}{\text{Total number of episodes}} \tag{2}$$

Therefore, the curve is monotonically increasing, with the y-axis being the overall success rate when the x-axis value is the maximum episode length. The better the model, the larger the area under this curve will be—similar in spirit to the precision-recall curve—because it will be able to have more of successful trajectories that are short early on.

Figure 12 shows the Multi-task (MT) cumulative success rate for the 3 ViZDoom environment tasks, using GRU hidden state language encoding: single target 5 and 7 objects in hard mode, and composition target 5 objects in easy mode. In the 5 objects hard mode (Figure 12a), we observe that all 3 training algorithms had similar performance until around episode length of 20, where ACTRCE has more successful trajectories which are longer. In the 7 objects hard mode (Figure 12b), ACTRCE maintains a similar behaviour while the baseline DQN essentially was only able to get success on very short episodes (i.e. when the target object was very close by). Lastly, in the 5 objects composition task (Figure 12c), the curve for ACTRCE indicates that there were 2 groups of trajectories: one requiring less than 10 time-steps, while another requiring over 20 time-steps. The former group occurs when the two target objects are adjacent to one another, making it easy to reach the second object after the first in only a few time steps. The latter group occurs when the target objects are not adjacent to each other, and thus requires the agent to more carefully turn around and avoid hitting other objects when trying to reach the second target.

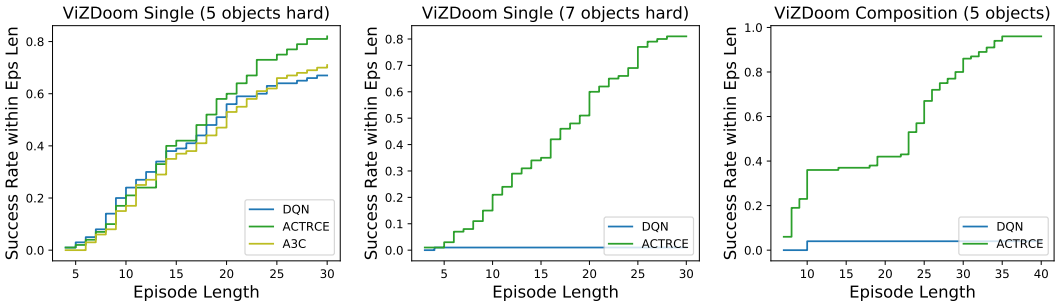


Figure 12: Multi-task (MT) cumulative success rate versus episode in ViZDoom environments, using GRU hidden state sentence representation. (a) 5 objects in hard mode, in single target case. (b) 7 objects in hard mode, in single target case. (c) 5 objects in easy mode, in composition case.

J Additional details on Hindsight Advice Annealing

We report the results for the hindsight advice annealing when using InferLite and One-Hot representation when providing hindsight advice on the first 10% and 1% of training:

Method	**MT**	**ZSL**
ACTRCE (GRU) 1.0	$\mathbf{0.80 \pm 0.06}$	$\mathbf{0.76 \pm 0.03}$
ACTRCE (GRU) 0.1	0.75 ± 0.01	0.74 ± 0.06
ACTRCE (GRU) 0.01	0.73 ± 0.04	0.66 ± 0.05
ACTRCE (InferLite) 1.0	$\mathbf{0.80 \pm 0.01}$	$\mathbf{0.76 \pm 0.02}$
ACTRCE (InferLite) 0.1	0.78 ± 0.04	0.68 ± 0.01
ACTRCE (InferLite) 0.01	0.76 ± 0.01	0.68 ± 0.01
ACTRCE (OneHot) 1.0	$\mathbf{0.80 \pm 0.03}$	–
ACTRCE (OneHot) 0.1	0.74 ± 0.06	–
ACTRCE (OneHot) 0.01	0.70 ± 0.06	–

Table 5: Comparing the Multitask and Zero-shot Generalization when the agent has limited hindsight advice to only the beginning of training.

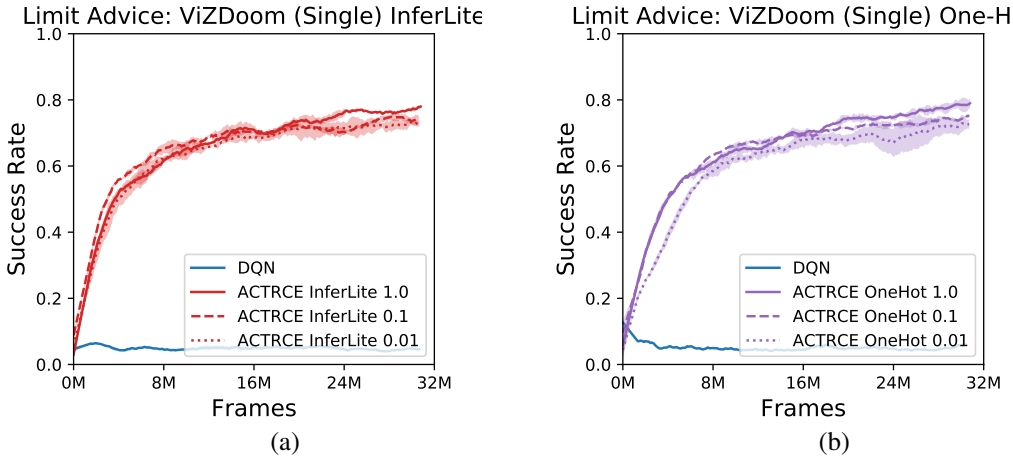


Figure 13: Performance comparisons on VizDoom environment.

K VISUALIZATION

For each single-target instruction in the training and test set (70 instructions), we obtain an embedding vector of dimension 256 using either GRU, InferLite, or One Hot (storing entire embedding). From this embedding matrix, we can visualize the learned instruction embedding in several ways.

K.1 EMBEDDING CORRELATION COMPARISON

For each pair of instructions i and j, we obtain their embeddings vectors v_i and v_j, and compute their *correlation distance*, $\text{cdist}(v_i, v_j)$. We define the *correlation distance* between two vectors u and v as:

$$\text{cdist}(u, v) = 1 - \frac{(u - \bar{u}) \cdot (v - \bar{v})}{||(u - \bar{u})||_2 ||(v - \bar{v})||} \tag{3}$$

We divide each matrix by the maximum entry as to scale the results to $[0, 1]$.

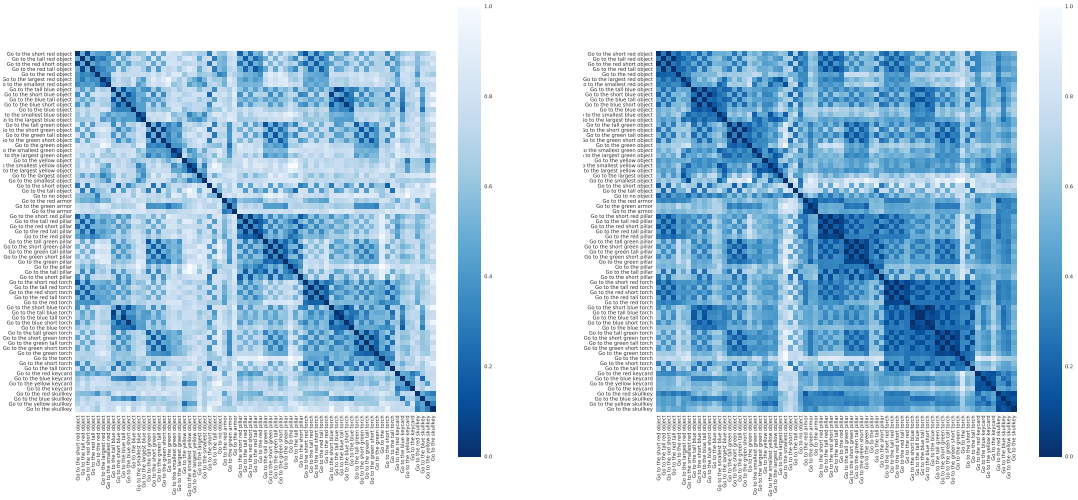


Figure 14: Correlation matrix for GRU (left) and InferLite (right). Best seen in electronic form.

K.2 T-SNE PLOT COMPARISON

We use t-SNE to visualize the space of instruction embeddings. In the following figure, the shape of the datapoint indicates the type of object, such as triangle for armor, diamond for pillar, etc., and the colour indicate the object's colour, with black being used for when no colour is specified. Finally the size of the data point corresponds to the size indicated in the object, from 'smallest', 'small', none specified, 'large', and 'largest'. For the instructions with synonym word replacement, we simply leave the colour as black and the default size.

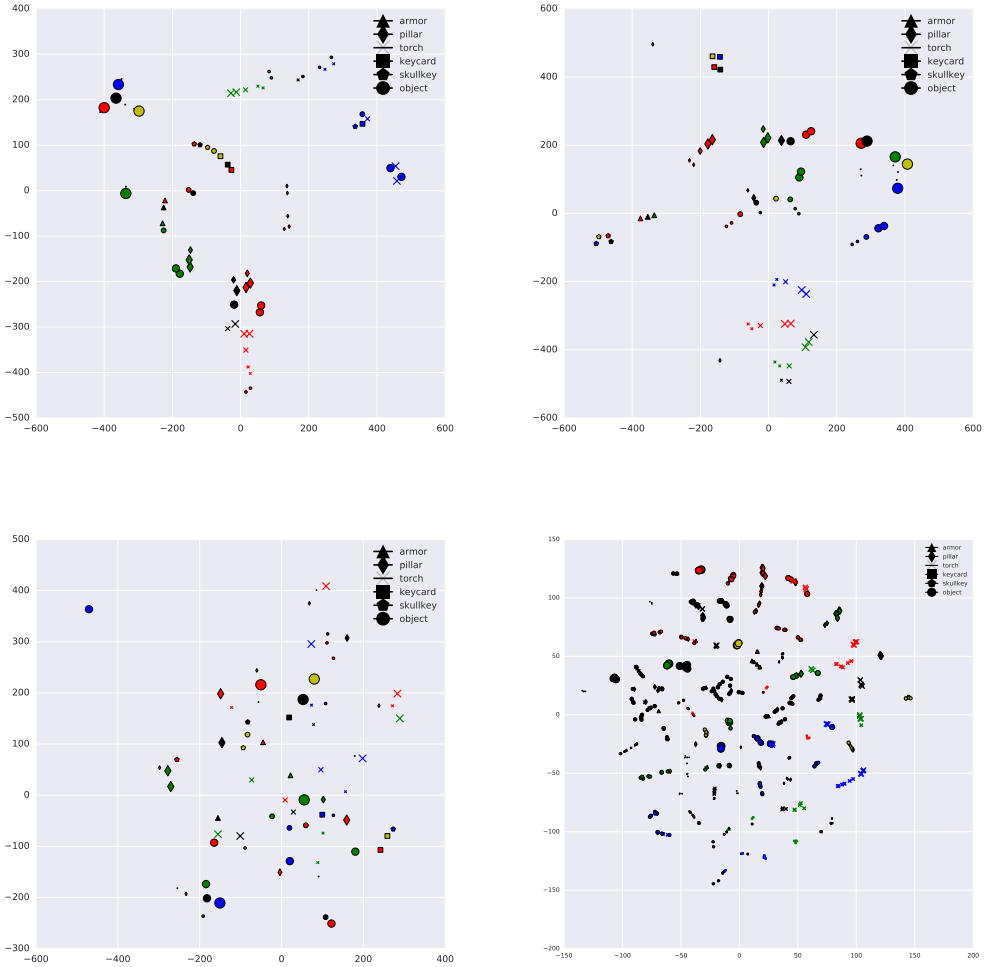


Figure 15: t-SNE plots of instruction embeddings for GRU (top left), InferLite (top right), One-hot (bottom left) and Inferlite synonyms (bottom right). Best seen in electronic form.