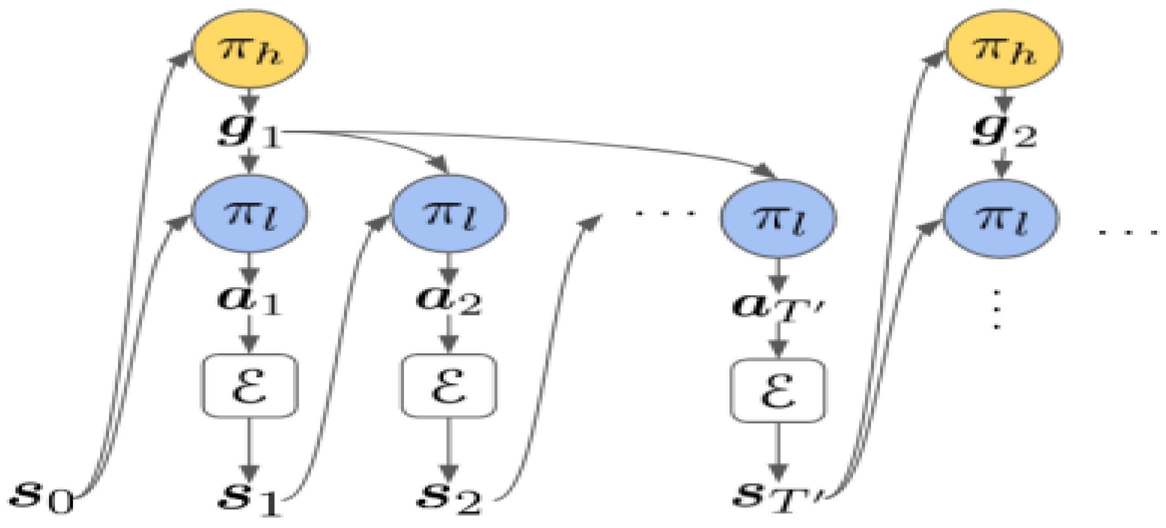
Language As an Abstraction for Hierarchical Deep Reinforcement Learning

Parth Kachhadia
1223619202

Introduction: Language is one of the basic forms of communication between humans, one key reason behind such feats of language is, its inherent structure which allows flow of information between users without many prior required assumptions. For example, Communicating simply “I am hungry now” delivers useful information to the listener on the current state of the speaker, without providing explicit representations for words used and different modes of information can be embedded together within a simple sentence. Which takes us further into the compositional nature of language. Authors of the paper, while working on the problem of abstractions required to solve long-horizon tasks using Hierarchical Reinforcement Learning explored language because of these reasons. Because Long-horizon Temporally extended tasks require solving many small tasks and reusing some of those learned sub-tasks again. For example, If an autonomous agent has to deliver an order to a customer, it cannot try to solve the whole task everytime from scratch. Instead it can formulate sub-tasks like crossing roads, waiting for signals, avoiding obstructions, opening a lock, etc and use those learned sub-tasks again and again to get to the original goal. But, Hierarchical Reinforcement Learning, which is one possible solution for these problems tries to learn these hierarchy of policies, task A can be accomplish by resolving task A1 and A2, faces challenges where agent cannot generalise well to unseen High-level tasks and hence cannot use these already learned sub-tasks as it should have. Reasons for these sorts of problems can be attributed to abstractions used by the learning agent to formulate such sub-tasks. In a way, because setting changes and abstractions are fixed then it won’t identify those. On the other hand if an agent formulates flexible abstractions totally reliant on primitive sensor data, we may encounter an alignment problem, where we (humans) cannot interpret, use ,correct or control agent behaviours. If language can be used between high-level decision making and low-level actions, all of these described problems seem solvable. Although, language itself is ambiguous sometimes, which can bring on additional challenges of its own. But, this work focuses on the use of language as an abstraction in an explorative view.

Hierarchical Reinforcement Learning:

Reinforcement Learning has expanded the scope of AI far ahead than ever realised before. Using simple algorithms like Q-learning an agent can determine actions to perform based on reward or exploration strategy. Such systems can solve many simple tasks. One setting generally used to train such RL agents is Markov Decision Processes. Using this, RL agents can learn to fulfil goals, with or without a model. But, when it comes to temporally extended tasks, MDPs cannot be applied in such settings. To get around these, HRL has been used, where these micro MDPs can be learned in a hierarchical structured way. In other words, High-level MDPs formulates its actions as small MDPs, which can be learned by Low-level. As described earlier, such an approach works for tasks but it faces certain problems where one possible solution is to use language in the middle of these hierarchies. Authors of the paper, formulates the HRL with language abstractions as, High-level policy learns to generate or select (in their case) language instructions or sub-tasks describe using language and Low-level policy is formulated in a way, its their objective to solve or learn those language instructions or sub-tasks. One possible question would be, how low-level policy would learn such an objective using simple RL. To answer this the authors use Goal Conditioned RL.

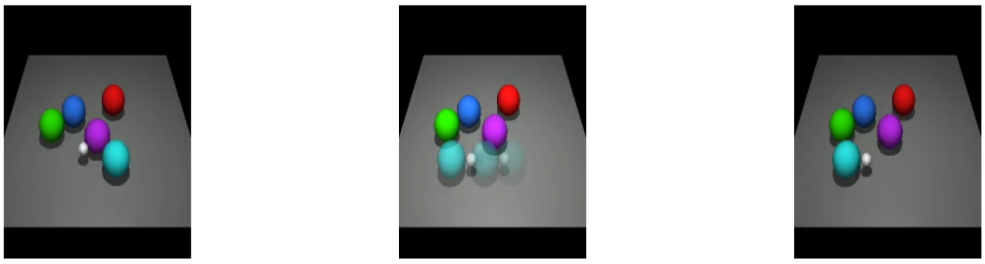


Goal Conditioned RL:

For simple RL, a goal is often described in the form of state, where it would get reward. Markov decision processes provide transitions required and reward signals for an agent to learn. As described earlier, language has combinatorial nature, where multiple possible states can be described with simple sentences. Goal conditioned RL is often used in such cases, where an RL agent learns a particular behaviour given the fixed goal. When supplied with similar functionalities within MDP, Augmented MDP allows an agent to learn transition functions and get rewards based on such goals. Since, an RL agent is now learning a behaviour based on language goal, it would learn to reach multiple possible goal states, because language goal represents a set of goal states. These features would allow such an agent to generalise to new domains given the same language interface and address the problem of generalisations to new tasks. Although language provides all the benefits, how would an agent get a reward based on learning to solve instructions? For that, we would require some way of verifying the language instruction against the agent's current state.

Environment:

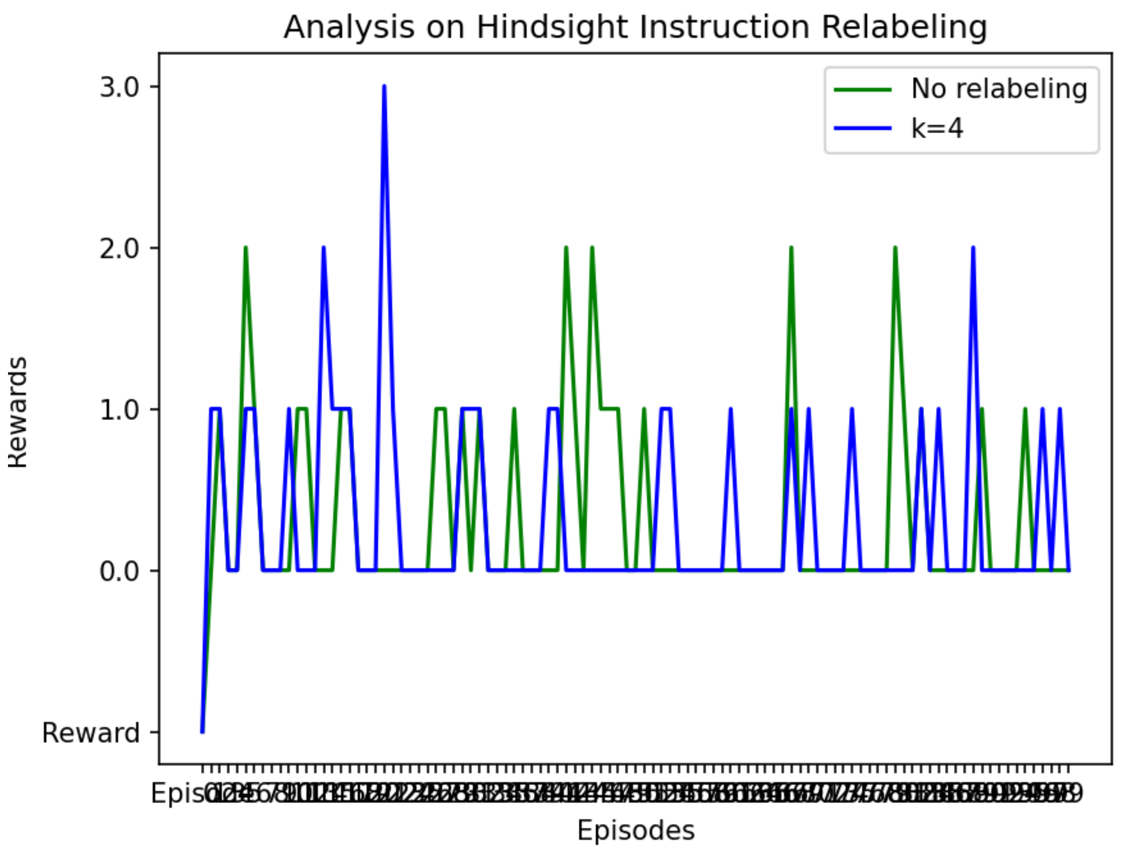
To embed language within the HRL, authors in the paper relied on the specialised environment called Clevr_robot_env built using CLEVER vision-language question answering dataset and mujoCo physics engine. In a way, the environment in this case



generates the state and a language description which is not the current state. This environment also hosts a question-answering module, which would resolve a language question on current state, and provide yes/no answers. Using these functionalities, the objective of learning language based goal conditioned RL can be formulated, where a scene generated by the environment acts as a current state of the environment. Observations for the agent and simultaneously generated language description works as a goal, because it's different from the current state. Environment supports different actions where objects in the scene can be moved around, sorted, filtering, etc. After each step, the current state in the scene can be verified with a language description giving whether the agent has reached the region of states which represents the goal or not. This formulation provides an agent a binary reward, which can be changed as per the objective of the problem at hand.

Hindsight Instruction Relabeling:

Environment used to explore their claims is a sparse reward setting. Learning in such scenarios is very computationally exhaustive for agents. Because the agent will get reward only when it will reach the language statement in the description. Heuristics works well for planning problems, by providing additional information related to the goal state. Similar approach often used in RL, is k-future strategy, where trajectory leading towards the goal state would be relabeled with discounted rewards, changes in state transitions as per their distance. Getting such distance is also very complex for goals with language. To overcome such shortcomings, authors mentioned Hindsight instruction relabeling algorithm in their work, which relies on greedy approach and applies these changes only when goal is reached. In such changes, their algorithm finds k prior states leading towards the goal state and provides discounted reward to these states. Although it's not an optimal setting, the learning agent gets a stronger reward signal towards the states it would need to consider.



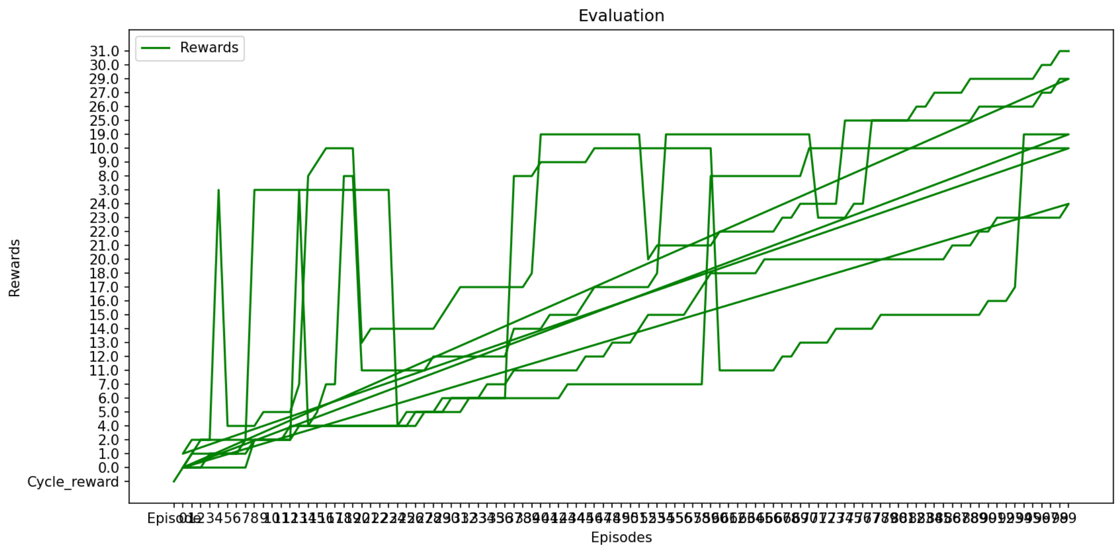
As we can see, HIR makes learning faster for the agent. However, there is a limit to this advantage gained. Since, each iteration also increases compute while training, selecting a K for our setting has trade-off with compute resources available.

<u>**Low-level policy:**</u>

Paper provides the implementation details on learning state-based language based low-level policy. As described earlier the environment contains a scene and language description. These scenes can contain upto 5 different objects with varied properties. The language description, which is the final objective of these policies, often contains comparisons of these objects on their properties. For example, “There is a blue rubber ball, are there any red metal spheres behind it?” The Low-level policy as described before will learn different actions to eventually reach a state where a red metal sphere is behind a blue rubber ball in the scene.
Environment used provides the state as a set of individual observations, each for an object present in the scene at the time. For that reason, the objects can be concatenated together to get combined representation, useful to train DDQN network used to do Q-learning and get an action for the low-level policy. These concatenations can be performed for all possible objects in any number of possible objects combined. To keep things simple, pairwise observations concatenated and language descriptions are also limited to generate questions for one-hop distances.

Implementing High-level policy

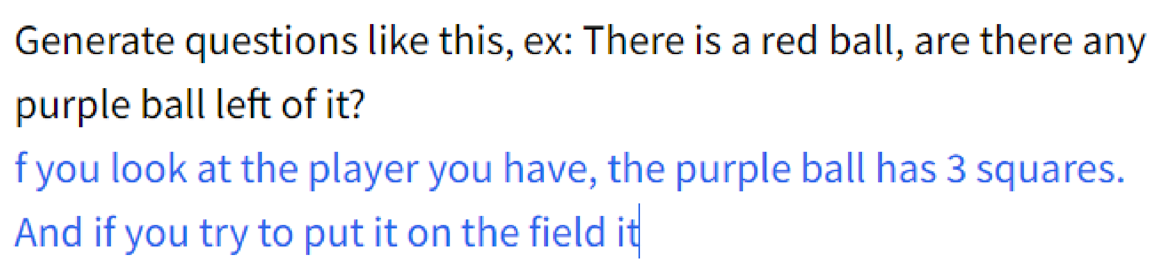
Same state based pari-wise concatenated observations were again used to train another DDQN, High-level policy generates or selects a language instruction as an action and applies a low-level policy to solve it. Number of low-level policies has been fixed to 5 in our implementation. Implementation can be made to make such high-level policy roll out optimum number of low-level policies to reach the goal state. However, that would require generating a language instruction and training it with this DDQN to reduce combined loss. In our setting, we have 75 possible language instructions, possible to get generated by our environment, and as an action, high-level policy selects one each time independent of the other four instructions selected. Implementation is provided in the Appendix.



Experiments with GPTs:

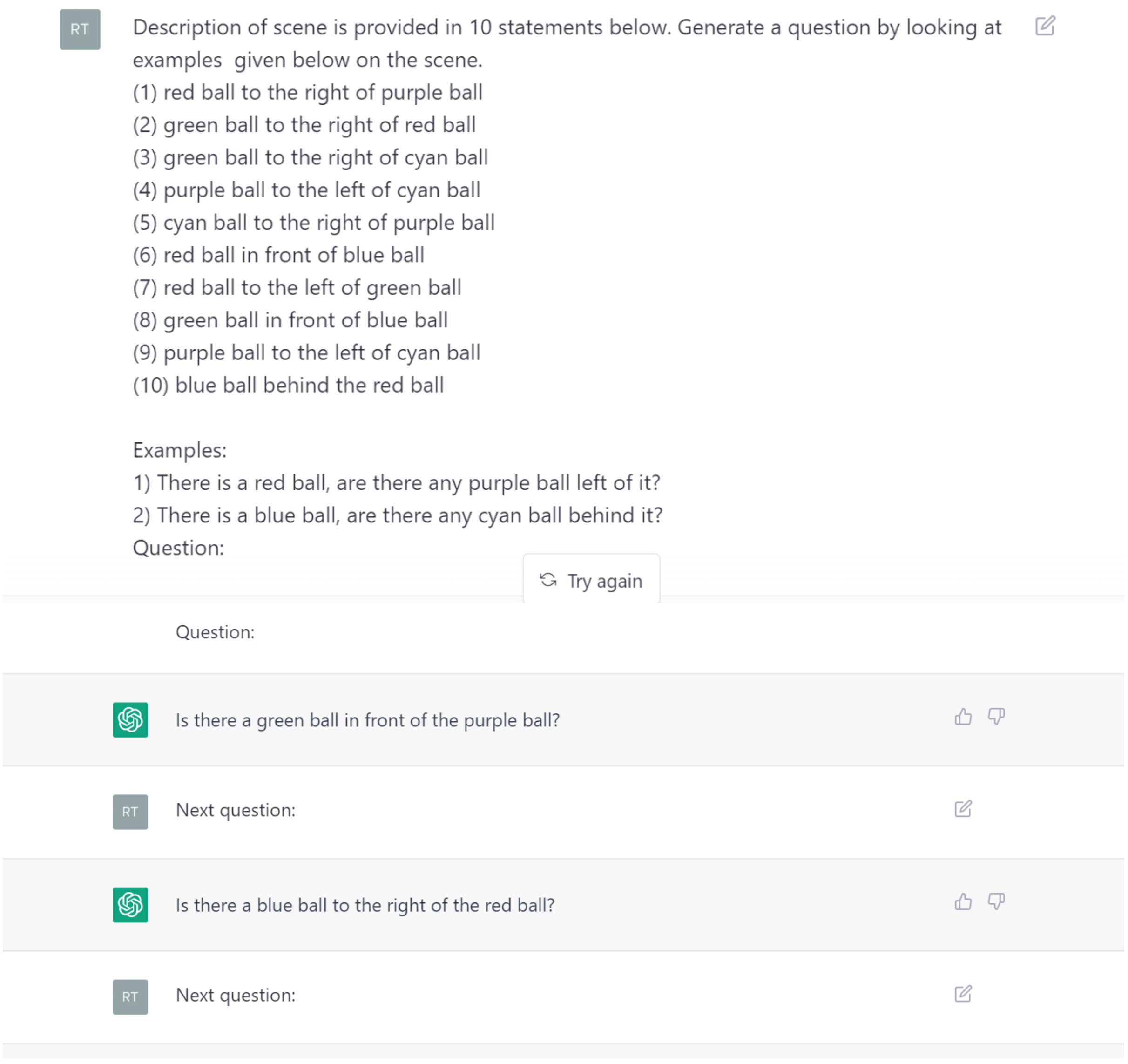
To provide generative structure to high-level policy, one possible way is to use GPT-2. Pretrained GPT-2 was used to generate next question given the previous random question, since GPT-2 has been trained to generate next most probable word from the dictionary, it couldn't seem to have understand the context we have set for it to do the generation and gives output which is not useful for our use. One limiting factor in the case of GPT-2 is it has smaller input capacity. So, similar experiments are performed with GPT-3 as well, since it has larger token input, we can set the context for it to generate at least relevant questions for our training objective. GPT-3 api is limited, so cannot be used for training, however chatGPT certainly solves that issue. Examples of these experiments are provided below. It seems once api access becomes available for chatGPT, it can certainly be used for this task.

GPT-2



Generate questions like this, ex: There is a red ball, are there any purple ball left of it?
f you look at the player you have, the purple ball has 3 squares. And if you try to put it on the field it

ChatGPT:



Description of scene is provided in 10 statements below. Generate a question by looking at examples given below on the scene.
(1) red ball to the right of purple ball
(2) green ball to the right of red ball
(3) green ball to the right of cyan ball
(4) purple ball to the left of cyan ball
(5) cyan ball to the right of purple ball
(6) red ball in front of blue ball
(7) red ball to the left of green ball
(8) green ball in front of blue ball
(9) purple ball to the left of cyan ball
(10) blue ball behind the red ball

Examples:
1) There is a red ball, are there any purple ball left of it?
2) There is a blue ball, are there any cyan ball behind it?
Question:

Question:

Is there a green ball in front of the purple ball?

Next question:

Is there a blue ball to the right of the red ball?

Next question:

However, our environment setting keeps changing once an agent reaches a goal statement, getting those language descriptions of scenes given to chatGPT is challenging, since we don’t have a module of such kind within our setting. One way around this, is to use adversarial training where with help of a discriminator, GAN can be trained to generate language instructions out of noise. These discriminators and GAN can be trained jointly with High-level DDQN to optimise instruction generation tasks which makes agents getting closer to their goal one step at a time.

Future Extensions:

Language based instruction generation with language based goal conditioned reinforcement learning for high-level policy and goal conditioned, instruction learning low-level policy are certainly possible only because the environment used in the experiments provides the modules required for the tasks. To get out of the confines of such an environment and apply a similar approach on different temporally extended long-horizon tasks, pairs of pretrained models can be used with environment rendering. The models required for such an objective are a Visual Question Answering model and an Image to Text model. We need both of these functionalities to generate language statements required to represent current state in the form of language descriptions and a model which can resolve language based query using knowledge from the scene provided. These models are often hard to train, vision based deep learning requires huge amounts of compute power and large amounts of datasets. But, for the exploration purposes, huggingface framework provides a library of pretrained models, which can be used either off the shelf or fine-tuned with task specific natural or synthetic dataset to serve our purposes.

Conclusions

Use of language with Hierarchical Reinforcement learning opens up many opportunities. However, Due to the current state of language based generative models and vision and language based systems, our environment setting for the agent is of sparse reward. Hindsight Instruction Relabeling tries to solve this problem but still it requires more computation power to train these goal conditioned language based deep reinforcement learning systems. Depending on the objective of the task, these systems are trained for, if getting understandable, human aware and communicable policy is of priority, then such systems can provide significant benefits.

Appendix:

```
class HDQN(nn.Module):
    def __init__(self, observations_size, goal_embedding_size):
        super(HDQN, self).__init__()
        self.observations_size = observations_size

        self.goal_embedding_size = goal_embedding_size
              self.network  =  Network(self.obobservations_size[1]  *  2,
goal_embedding_size).to(DEVICE)
```

```
    def forward(self, obs):

        out = current_state_high_level(obs, self.network)
        return out
class Network(nn.Module):
    def __init__(self, in_size, out_size):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(in_size, out_size//2),
            nn.ReLU(),
            nn.Linear(out_size//2, out_size)
        ).to(DEVICE)

    def forward(self, observations):
        return self.layers(observations)
def current_state_high_level(observations_combined, network_module):


                                        observations_combined          =
torch.Tensor(observations_combined).to(DEVICE)



    list = []
    for i in range(observations_combined.shape[0]):
        for j in range(observations_combined.shape[1]):
            for k in range(observations_combined.shape[1]):
                 list.append(torch.cat((observations_combined[i, j, :],
observations_combined[i, k, :]), 0).to(DEVICE))
    output = network_module(torch.stack(list))

    return output

instruction_set = []
T = 5

agent.train():
for instruction in range(T):
      lower_goal, prog = instructor.get_instruction(state)
      env.set_goal(lower_goal, prog)
```

- https://github.com/pathu007/Language-as-abstraction

References:

- Yiding Jiang, Shixiang Gu, Kevin Murphy, and Chelsea Finn. Language as an Abstraction for Hierarchical Deep Reinforcement Learning. In Workshop on “Structure & Priors in Reinforcement Learning”at ICLR 2019, jun 2019. URL http://arxiv.org/abs/1906.07343.
- https://github.com/google-research/clevr_robot_env.git