Contrastive Learning Enables Pixel-Based Goal-Reaching

Aaron (Louie) Putterman
University of California, Berkeley
albp@berkeley.edu

***Abstract*—With recent advances in contrastive learning and data augmentation, pixel-based environments have become more approachable with traditional RL methods. However, one region of study which has notably lagged behind is goal-reaching in pixel-based environments with pixel goals. This refers to the setting where the agent only has access to a rendered image of the desired environment and the rendering of the current state of the environment and must decide on which actions to take in order to reach the goal observation. In this paper, we propose a method which solves this task by using contrastive learning to both label the reward and encode the observations to be used by the agent. Alongside Hindsight Experience Replay, this method is able to reach the majority of the goals proposed in the DMControl reacher and point_mass environments in very few training iterations. We also demonstrate the ability of the method to generalize to other goal-based environments.**

The method uses a MoCo-style encoder and contrastive loss along with a TD3 architecture to learn how to reach any goal-frame that is passed in. By either using a pretrained encoder or training an encoder online alongside the agent, we are able to use the latent space of the encoder to label the reward for an image based on the desired goal. More explicitly, we modify the existing CURL architecture and make a few key changes to extrapolate it to more general goal-conditioned environments. First, we make the policy goal-conditioned, meaning that the actor and critic both take in a goal-observations in addition to the current observation when deciding on a value or an action, respectively. Second, we modify the replay buffer so that it also stores the current goal for each step the policy takes. Third, we weave hindsight experience replay into our algorithm to prevent the model from succumbing to sparse rewards in our environments and encourage quicker learning by modifying the desired goals in our replay buffer. Lastly, we harness the contrastive encoders learned by the CURL module to calculate both the environment reward and the relabelled hindsight experience replay reward. More clearly, we encode both the current image and the goal image and then use either a step function based on their l2-distance in latent space, a step function based on their cosine similarity, or another way of measuring how close the two encodings are to label the respective reward for the observation and goal. Then, the actor loss and critic loss are calculated using this labelled reward, and these losses are backpropagated through the network (including the encoder). In order to ensure that the latent space is well-shaped and spread out, we continue to optimize with the contrastive loss, which is designed ensure that the encoder is capturing key identifying information in the environment like pose and orientation.

To ensure that our contrastive learning procedure is indeed creating a well-shaped and useful latent space, we first present some visualizations of the latent landscape. To do this, we create a heatmap of sorts, detailing the position of the points in the point_mass environment and what their respective l2-distance would be to a goal observation. Next, we present our results on the point_mass and reacher environments. Using this method, we are able to reach a greater than 90% evaluation success rate in the DM-Control reacher and point_mass environments, meaning that for 90% of random goals and random starting positions, the agent is able to reach the goal from the starting position. Additionally, the agent is able to learn very quickly, often reaching this high evaluation success rate within 20k updates. Furthermore, while the method requires some hyperparameter tuning to reach its best performance, the method is quite robust to most hyperparameter choices and can work well in both the point_mass and reacher environments using the same hyperparameter configuration. In order to compare the performance of this method, we also implemented an inverse-model as a baseline. Inverse-models tackle the problem of goal-reaching by using supervised learning to predict the action that is required to move from one observation to a subsequent observation. We found that the performance of inverse models was much worse, and often failed to reach any goals when compared with the performance of our method. This is likely because inverse-models lack the ability to plan for long-horizon tasks and do not have a Q-function which they can greedily exploit to decide which actions to take when far from the goal observation.

This kind of policy is best suited for pose copying or demonstration copying of real-life robots. Since the policy is reliant on a specific frame of the environment, as opposed to just x-y-z coordinates of where we want the robot to go, it is necessary for the task to be completed before the policy tries to reach the goal. This is different from some other goal-conditioned policies which are instead designed to reach specific locations or complete certain tasks, but are not given an image of what the desired task looks like. Additionally, this kind of policy can be used for pretraining of encoders or policies for other more complicated tasks since it requires minimal supervision or reward shaping, and instead is largely self-supervised (from its own encoder).

Another promising result from this work is that it proves the viability of more advanced curriculum learning frameworks in pixel-based settings. A natural next step is to harness the contrastive capabilities of the encoder to aid in generating a curriculum. Currently, the method relies on being able to randomly generate goals in the environment which is itself a difficult task. For this reason, we had to generate large offline datasets for various environments to ensure that the agent could condition itself on a variety of goals. Fortunately, as part of the contrastive learning loss calculation, a similarity score is calculated between all pairs of images in a specific update batch. This lends itself easily towards a greedy optimization of selecting the top-k subset of images in that specific batch that are most diverse and can thus be used for goals to condition on in the next round of training procedures. In this way, we can encourage the model to select new, unseen goals, which would solve the problem of goal selection and improve exploration speed in complex environments.

I. Introduction

In recent years, Reinforcement Learning (RL) has made tangible the possibility of training agents in various environments to accomplish specific tasks. However, one shortcoming is that standard reinforcement learning agents are typically taught to accomplish a single task in a single environment. Agents have been successfully taught to win games of chess [17], master old-fashioned Atari games [9], and perform feats of robotic manipulation like opening a door [14]. However, these agents typically have specifically crafted reward functions designed for the agent to perform one task (maximizing score, winning the game of chess, etc.)

Inspired by human agents who can perform a wide range of tasks, recent effort has gone into making agents that can generalize to multiple tasks within a single environment [16]. For example, many works detail robots that can try to match certain poses or rearrange a setting in a specific way [3]. This kind of environment is typically more difficult as the agent will have to learn more complex behaviors and adapt to more unfamiliar situations. Additionally, it is difficult to craft a reward function that allows for seamless transitions in performing different tasks. Inspired by human agents, who are very proficient at learning to reach multiple different goals with purely internal reward supervision, we present such a method for pixel-based goal-reaching in RL settings. Our method, much like human agents, is guided by strictly self-generated rewards, coming from our model's ability to discern how similar two states are. Additionally, we propose a method to take advantage of our model's internal similarity metric in order to propose a diverse set of goals for the agent to accomplish, much like humans do while playing.

A. Problem Statement

In this paper, we are dealing with the problem setting of goal-reaching from pixels. Namely, the policy is given a goal observation o_g and a current observation o_t and is tasked with selecting an action a_t to help the agent reach the desired goal observation. In order to encourage the agent to select actions that move closer to the goal observation, for any pair of o_t, o_{t+1}, a_t, o_g, we label a reward r_t based on some similarity measure of the encodings of o_{t+1} and o_g. As an example, in the DM-Control [18] reacher environment, o_g would be the reacher arm in a specific position, o_t would also be the reacher arm in a specific position, and the agent would be tasked with finding the actions to move the arm to match the position of o_g. In order to do this, the agent must be able to decide how "close"two observations are and use that closeness as a way to label the rewards between a current observation and the goal observation.

II. Related Work

Previous approaches for pixel-based goal-reaching in reinforcement learning have often taken the form of using VAEs [10] or other similar latent variable models to encode the images and exploit their latent space. Most notably, SkewFit/RIG [15] [13] uses a VAE to encode pixel-based observations into a latent space. Since the VAE is a generative model, it can be used to generate specific goal-observations, yet also provides a prior probability of seeing each observation. Then, because the agent has access to the probability of each generated goal observation, it can skew the sampling of low-probability images and thus expedite the exploration of the environment. Additionally, the Gaussian shape of the probability distribution allows for a convenient reward-labelling scheme of exp(-l2) of the encoded images in latent space. Our method is similar in that it also uses the latent space of the encoder to label the reward, but uses a MoCo-style [7] contrastive learner and cosine similarity instead of a VAE with exp(-l2). One negative side effect of our approach is that it removes the ability of generating never-before seen observations. However, we present later a method of exploiting our contrastive encoders to select more diverse goals for curriculum training.

Another essential method for goal-based reinforcement learning is Hindsight Experience Replay [2]. HER cleverly stores achieved goals and desired goals for every observation seen during a trajectory, which allows for hindsight relabelling. The process of relabelling entails choosing a random observation (achieved goal) along with another random observation (desired goal) and calculating what the reward should be for that state. This can be difficult in pixel-based goal-reaching environments since there is no reward signal from the environment, and simply using an MSE between the current image and goal image will not capture important relationships in the environment. Hence, it is often necessary to train some sort of encoder which has a meaningful latent space (in this case a MoCo-style encoder). HER can then select current and goal images which are temporally close and may have larger rewards to overcome the sparse nature of goal-based environments. In fact, it has been shown that in a bit-flipping environment, HER allows agents to quickly pick up on trends and master tasks that would otherwise take an exponential amount of time for more traditional agents. HER has been successfully deployed on push, slide, and pick-and-place environments, significantly outperforming other agents.

HER can also be seen as a form of implicit curriculum learning [2], where the new proposed goals being assigned to the agent come from its previously visited states. Curriculum learning in general has been shown to be important in expediting the training process of an agent and encouraging the agent to completely explore an environment. Recent advances like GoalGAN [8] and Automatic Curriculum Learning Through Value Disagreement [20] propose curricula by choosing goals they believe to be closest to the frontier of knowledge and thus force the agents to adapt to a more diverse set of goals. Like these papers, we propose a method through which agents can use contrastive encoders to propose diverse goals to expedite training.

The use of contrastive learning in conjunction with RL is also not a novelty. In fact, recent papers like CURL [12] have demonstrated that using contrastive losses and data augmentations [11] is very helpful for achieving high-performance pixel-based agents. The CURL architecture relies on a MoCo style contrastive learning scheme alongside Soft Actor Critic [6] for reinforcement learning. The results achieved by the agent showed that on several environments in the DMControl suite, pixel-based agents could match the performance of state-based agents with relatively simple architectures and short training times. However, the paper RAD [11] published by the same group found that much of the performance improvement seen by the agents was not due to the contrastive encoders but instead was attributed to the vast number of data augmentations which increased the robustness of the agent. In contrast, our method requires a contrastive learner to enforce a latent space conducive to a cosine similarity metric for reward labelling, but this reward labelling does not rely on extensive data augmentations and hence serves to further motivate the use of contrastive learning.

Another similar vein of work is that on inverse models. This branch of models relies on a more supervised approach compared to traditional RL. Given a large dataset or replay buffer of states, next states, actions, and optionally goals, these models will try to predict the action to go from the given state to the next state and optionally conditioned on the goal. While these methods are simple, they provide a clear approach to the goal-reaching problem on pixel-based environments. However, they often lack the ability to predict farther into the future and will fail on more complicated tasks. We implement some of these methods and show how their performance compares to our method on basic environments in the DM-Control suite.

III. Our Method

A. Overview

The general structure of the model resembles that of a typical pixel-based reinforcement learning model. First, there is an encoding module with convolutional layers [1] and layer normalizations [4]. On top of these convolutional layers are both the actor and critic modules which are fully connected layers which output either the action or value of the state and action respectively. The reinforcement learning scheme that we are using is TD3 [5], meaning that there are two Q-networks, less frequent updates, and added noise to the actions as compared to a more typical DDQN [19]. To decide on the next action taken by the agent, we encode the current observation o_t and the goal observation o_g using our contrastive encoder and pass these latents into our actor module.

B. Training the Encoder

Training the encoder is one of the most important aspects of this method, as it lays the foundation for both the reward labelling as well as the latent variables that are passed into actor and critic. As such, much time was spent fine-tuning the encoder to make sure that the latent space was well-shaped, i.e. inducing a good reward under our similarity functions, while still providing enough information for the policy to act near optimally. To get the encoder in its final working shape, there were a few notable tricks that had to be applied to the MoCo [7] implemenation adopted from the CURL [12] framework. First, it was important to move away from stacks of 3 frames and instead focus on single frames at a time. This intuitively makes sense because if we are trying to teach a policy to reach goals, we are more focused on the position of the robot rather than the velocity or angular speeds of its arms. Since position is readily available from a single frame, using this approach allowed for faster and less noisy training and better shaped rewards, while still leaving the possibility of concatenating the encodings of multiple frames if the velocity information becomes necessary in the future. Next, it was important to change the random croppings of the renderings of the environment to instead be random translations (meaning the image itself is shifted left, right, up, or down within a bigger frame). This is because the croppings removed information from the image, so in extreme cases as in the point-mass environment, it occasionally happened that the point-mass itself was cropped out of the image. This in turn provided noisy inputs to the model and confounded training because there were no remaining distinguishing characteristics in the rendering.

Algorithm 1 Random Translation

```
feed in imgs, size
n, c, h, w = imgs.shape
out = zeros((n, c, size, size))
h1s = randint(0, size - h + 1, size = n)
w1s = randint(0, size - w + 1, size = n)
for out, img, h1, w1 in zip(outs, imgs, h1s, w1s) do
    out[:, h1:h1+h, w1:w1+w] = img
end for
return out
```

There were two main approaches that we took for training the encoders. First, we tried pretraining the encoders on large offline replay buffers, collected from other policy deployments. Second, we tried training the encoders during the actual training of the agent. Both approaches led to success, but we found that the second method was more usable and faster since it takes away the requirement of pretraining a contrastive encoder. In fact, on the DMControl [18] reacher environment, we found that training the encoder during the run led to better performance when compared to a pretrained encoder. This is likely due to the fact that a pretrained encoder is frozen during the entire run, so any problems in its latent space, such as if two images produce similar encodings but are not actually similar, can lead to extra noise in the reward signal and cause the agent to fail. On the contrary, an encoder that is learning during the run will likely be as noisy in any given iteration, but it is less likely that one image will consistently have misleading rewards, so the agent can adapt. In the pretrained encoder case it is worth noting that we still learned an extra encoder for the actor and critic. The rationale behind this was that we only wanted the encoder to be frozen for labelling the rewards, to prevent a "moving target", but still wanted an encoder that could adapt and provide more meaningful information for the actor and critic. Since there may be some features important for the actor and critic but not for the contrastive loss, we made this extra encoder to avoid any shortcomings.

The actual optimization procedure for the encoder is pretty simple. We first calculate an actor loss using the undetached latents from the encoder and then backpropagate the loss through the network. Then, we repeat this procedure with the critic loss. Lastly, we implemented a "contrastive"loss, to encourage the latents for different images to be far apart from one another. In order to do this, we first select a batch of images. Then, we duplicate each image and perform random translations, but not the same random translation, to both copies. Next, we encode all the images to get latents z_1 and z_2, and perform a bi-linear product of the latents as follows: $z_1 \cdot W \cdot z_2^T$, to get a matrix that we interpret as the logits matrix. Then, we set our target for the first row to be 1, for the second row to be 2, for the jth row to be j. Lastly, we take the cross entropy loss of the logits matrix with these labels and backpropagate this loss only to the encoder to encourage the separation between latents that we desire. This happens because we are trying to force the logits matrix to become strictly ones on the diagonal and zeros everywhere else, which would mean that the encodings contain enough information even after a random augmentation to discern it from all the other encodings, ensuring diversity.

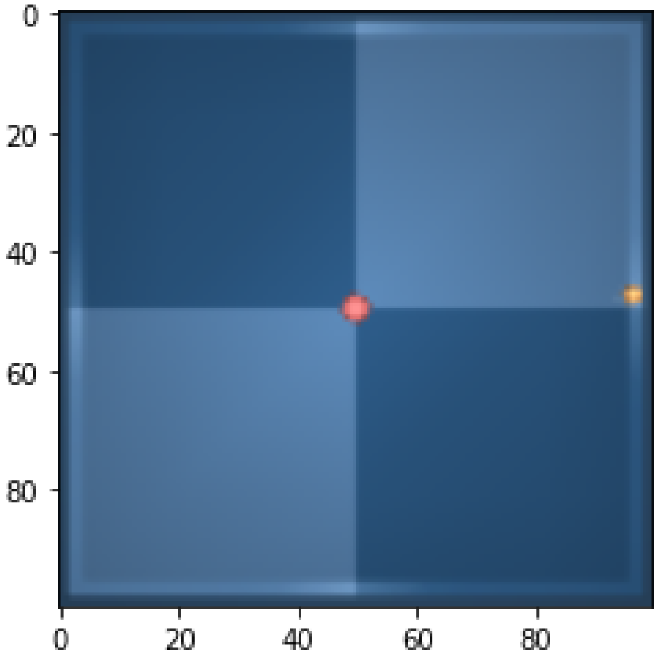
Algorithm 2 Contrastive Loss Calculation

```
obs_anchor, obs_pos = replay_buffer.sample()
z_a = self.contrastive_encoder.encode(obs_anchor)
z_pos = self.contrastive_encoder.encode(obs_pos, ema = True)
Wz = matmul(self.W, z_pos.T)
logits = matmul(z_a, Wz)
logits = logits - max(logits, 1)[0][:, None]
labels = arange(logits.shape[0])
loss = crossentropyloss(logits, labels)
minimize(loss)
```

C. HER

In order to prevent the sparsity of most goal-reaching tasks from hindering our performance, our model uses Hindsight Experience Replay [2] to provide more positive examples for the model to learn from. To achieve this, the model selects a random proportion of the examples sampled from the buffer to be modified. For these examples, we first start at a random time t_0. Next, a random time offset t is selected, and the desired goal is now relabelled to be the achieved goal from time step $t + t_0$. This way, when the generated t is small, there is a high likelihood that the desired goal and achieved goal are close when encoded into the latent space, since they are temporally close and hence the images will be similar, and they will act as a positive example for the policy to learn from.

However, there were some necessary modifications that had to be made to the code-base to get HER working well. Most importantly, HER's performance was much better when the calculated rewards were in the interval $[-1, 0]$ as opposed to $[0, 1]$. As such, many of the reward functions were changed to always output results in this interval, via subtracting 1. For example, the cosine similarity metric became the Boolean



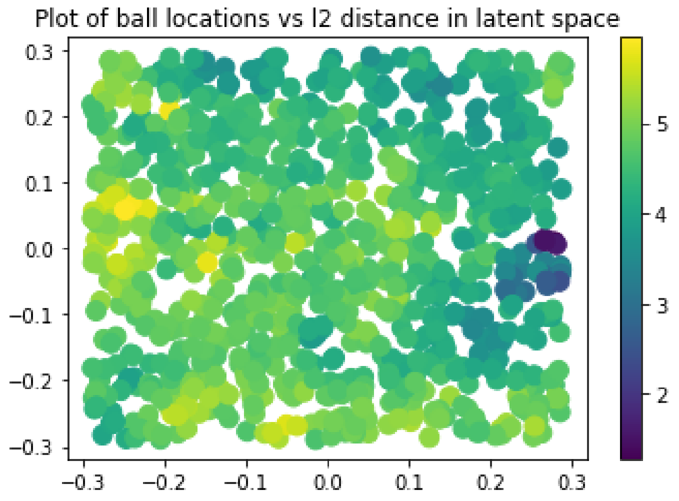


Figure 1. Top: The goal image. Bottom: An example of a heatmap that was generated when checking the shape of the latent space of the encoder. To generate the heatmap, we first select a goal image (which in this case was a rendering of the pointmass environment with the pointmass on the middle far right side) and get its encoding and underlying state. Next, we continually reset the environment, generating many new renderings of the environment with the pointmass in various different locations. For each of these resets, we keep track of the latent encoding and the actual underlying state (x, y) coordinates of the ball. Then, we compute the distance between the goal encoding and the current encoding for all the images that were generated and color-code the the distance. Then, a point is plotted on the heatmap of the specific color at the recorded (x, y) coordinates. As expected, the generated images whose pointmasses were near the goal image have a much smaller l2-distance in latent space compared to those that were farther away. This allows us to decide on a cutoff and accurately label trials as a success or failure without the underlying state.

$\frac{\mathbf{x}\cdot\mathbf{y}}{\|\mathbf{x}\|\|\mathbf{y}\|} > 0.95$, from which we then subtracted 1. Another change which we made to the model was that we increased the number of action-repeats that were performed in each step. We found that Hindsight Experience Replay worked better and that the model reached higher success rates when there were fewer steps in a given roll-out and larger changes in the environment between consecutive steps. As such, we repeated the actions between 5 and 20 times in order to have a stronger learning signal.

Algorithm 3 HER Update Procedure

```
feed in episode_batch, batch_size
T = actions.shape
rollout_batch_size = length(episode_batch)
episode_idxs = random_int(low = 0, high = rollout_batch_size, size = batch_size)
t_samples = random_int(low = 0, high = T, size = batch_size)

transitions = {key: episode_batch[key][episode_idxs, t_samples] for key in episode_batch.keys()}
her_idxs = where(random_uniform(size = batch_size) < self.future_p)
future_offset = random_int(low = 0, high = T - t_samples, size = batch_size)
future_t = (t_samples + 1 + future_offset)[her_idxs]
future_ag = episode_batch['achieved_goals'] [episode_idxs [her_idxs], future_t]
transitions['goal'][her_idxs] = future_ag
transitions['reward'] = relabel_rewards(transitions['achieved_goal_next'], transitions['goal'])
```

D. Evaluation of the Model

Since the model itself only has access to the renderings of the environment, it is important to keep track of the underlying state representations to make sure that the model is indeed reaching its desired goal. To this end, we modified the environment to return the current state representation of the positions of the environment as well as the state representation of the positions of desired goal environment. Then, during evaluation, we check in every iteration if these positions are within ϵ (where ϵ was a hyperparameter that could be changed) of each other, and if they are, it is considered to be a success. On the plots presented in the next section, we generate 10 random starting positions and 10 random goal positions and measure the percentage of the time that the goal positions are reached, which we call our evaluation success rate.

E. Environments

As stated previously, our model requires a random rendering of the environment to use as a goal observation. This is because the reward relabelling process uses the contrastive encoder which is trained on renderings of the environment. However, by default the goals in the DMControl point_mass and reacher environments are just target spheres in the environment that are supposed to be touched. So, in order to make these environments suitable for pixel-observation-based goal-reaching, we created wrappers for the environment that randomly generated states and rendered them to be the goal observation until the next reset. Then, a new random state is generated and rendered as the current observation. The underlying states of the environment are also stored, but are never accessed by the model and instead are used during the evaluation process.

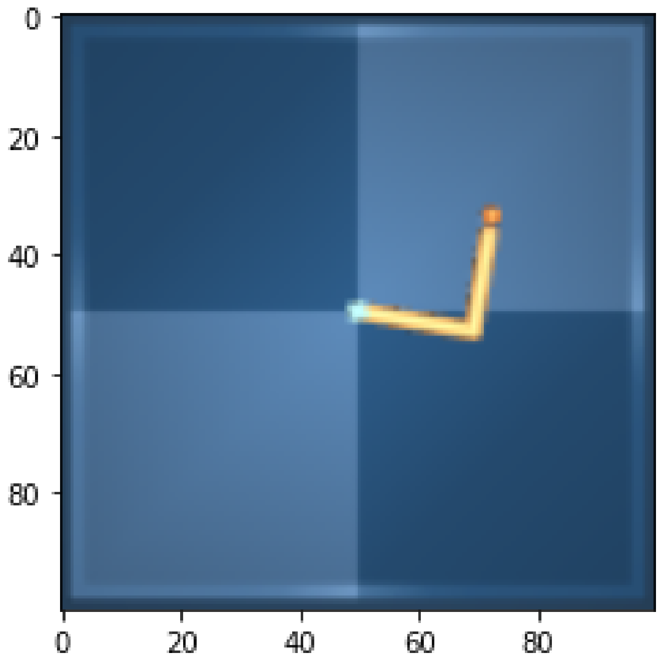


Figure 2. Above is an example of what we may see as a typical goal observation (o_g) or a typical current observation (o_t) in the modified reacher environment. It is then up to the policy to decide how to reach this frame (if it is a goal observation) or choose what action to get from this frame to the goal observation (if it is the current observation).

F. Baselines

For one baseline, we implemented a standard inverse model. In our case, the inverse model was trained on a large offline dataset, consisting of observations o_t, next observations o_{t+1}, and actions a_t that were taken to go from o_t to o_{t+1}. Then, for evaluation, we would take the current frame as the observation o_t, and use the desired goal in the place of o_{t+1} in hopes that the model would learn which actions to take to reach the desired goal. Since this method was training on pixel-based environments, we either included a contrastive loss in the model to help train the encoder or simply loaded in a previously trained encoder (pretrained with contrastive losses). Otherwise, the only loss that we used was a mean-squared error between the actual action and the predicted action during training. In our pixel-based environments, this method failed to reach any of the goals presented. Since the method is only trained on consecutive observations, it has no examples of long-term goal-reaching so intuitively we would expect it to perform worse.

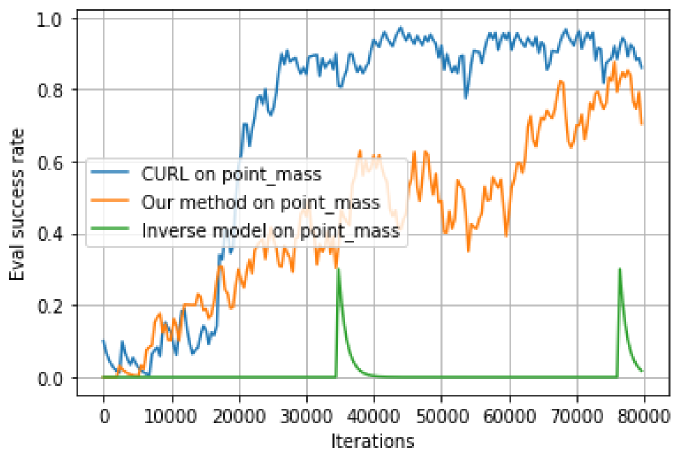


Figure 3. Above we can see a plot of the relative performance of the CURL method, inverse model baseline, and our method. It is important to note that the point_mass baseline is only taught to reach one goal at the center of the board and is thus not a typical goal-reaching algorithm.In fact it has a much easier task. Our method here significantly outperforms the inverse model method, but lags behind the CURL method on the unedited environment for the reasons mentioned before.

We invite other groups to create more complex models in this setting, perhaps using larger time-offsets. However, we expect that using larger time offsets may also introduce more noise in training as an agent may not be moving directly towards o_{t+k} in action a_t. We do believe that using RNNs or transformers will likely improve the performance of the inverse model in this setting, as the sequence of actions is much more suited for a time and order-dependent approach.

For another baseline, we trained the existing CURL implementation on the DM Control point_mass (see figure 3) and reacher (see figure 4) environments. However, the comparison is not strictly fair as we used the out-of-the-box point_mass and reacher environments for the CURL run (i.e. with the reward coming from the environment), while our RL method relied on its own encoder for the reward. However, our method does have an advantage in the sense that the environments for CURL are only supplied with a dot for the ideal position that the reacher arm has to reach (i.e. the end effector must touch a specific sphere), while in our environment the agent is given a picture of an RL agent in a specific position that it must copy. In a sense, our agent is given a ’solved’ environment that it must learn on its own to copy, while the CURL agent is simply led along by environment-given rewards. Nevertheless, this baseline shows differences in how quickly and accurately the different agents learn to accomplish their respective tasks. Also worth noting is that in the point_mass environment, the agent must only learn how to reach the center of the board, which is to say, the goal is static in every run. Thus, we would expect the learning to be quicker for the CURL model in its own respective environment/reward setting because it is not a goal-based policy, but instead simply learning to reach the same goal from different positions.

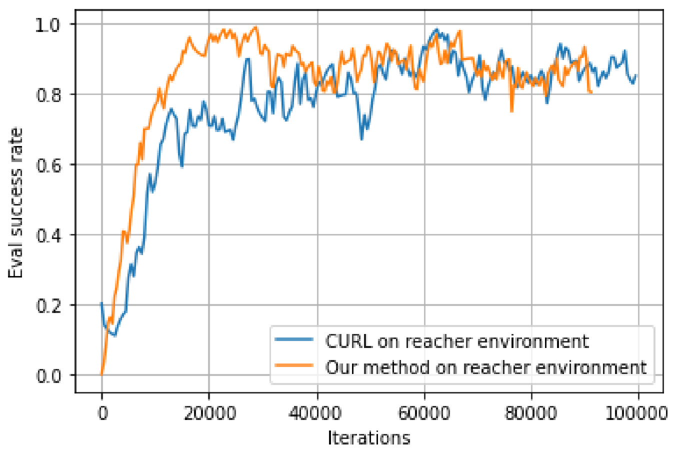


Figure 4. Above we can see a plot of the relative performance of the CURL method and our method. We can see that even though our method is responsible for labelling its own reward (i.e. the reward is not coming from the environment), our method is able to learn faster and reach a higher success rate than the CURL method.

IV. Experiments

In this section, we will present a detailed ablation study of our method as well as a discussion of failed experiments and likely explanations for their performance.

A. SAC

In our first attempt at creating a goal-reaching policy, we tried to use a basic SAC [6] implementation for teaching our actor and critics. Initially, this was successful for basic goal-reaching tasks on the point-mass environment. However, as we tried more experiments, we found this policy was very unstable and in many cases had exploding losses for both the actor and critic (see figure 5). We tried to pinpoint the cause of these exploding losses but were unsuccessful. Instead, after looking through the literature we found that many other pixel-based goal reaching [13] had used TD3 or other methods not including entropy regularization , so this shifted the focus of the project away from a SAC base.

Some important hyperparameters that we noticed when working with SAC were whether or not we backpropagated certain losses through the convolutional encoder (see figure 6). Since we were using the CURL[12] module, we were calculating a contrastive loss and could choose to backpropagate that through the encoder. However, we were also calculating critic losses and actor losses based on the outputs of our encoders. Depending on whether or not we detached these encodings before passing them through the actor and critic, we could control whether or not the actor loss and critic loss could also contributed towards shaping the latent space of the encoder.

B. TD3

After succumbing to the difficulties presented by training with SAC, we decided to change our approach to instead use TD3 [5], which does not include the entropy regularization of

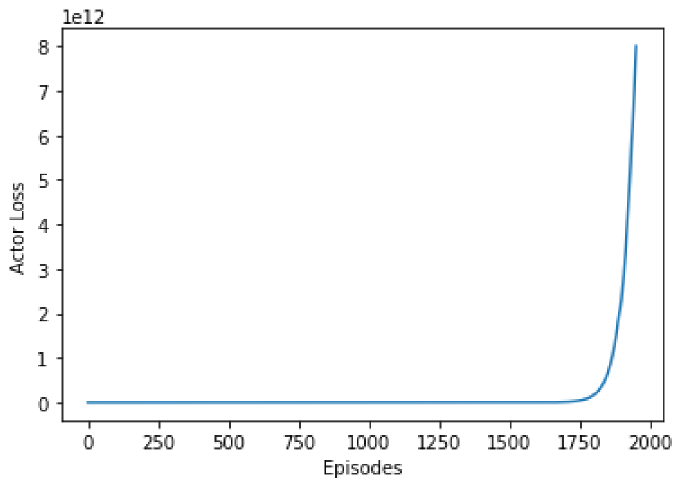


Figure 5. Above we can see an example of the exploding actor loss while training on the DM-Control reacher environment. After running for 2000 iterations, the actor loss approached $1x10^{13}$, indicating that the actor was not optimizing as it should. Indeed, after rendering the agent's behavior in the environment, we could see that it was not reaching the proposed goals.

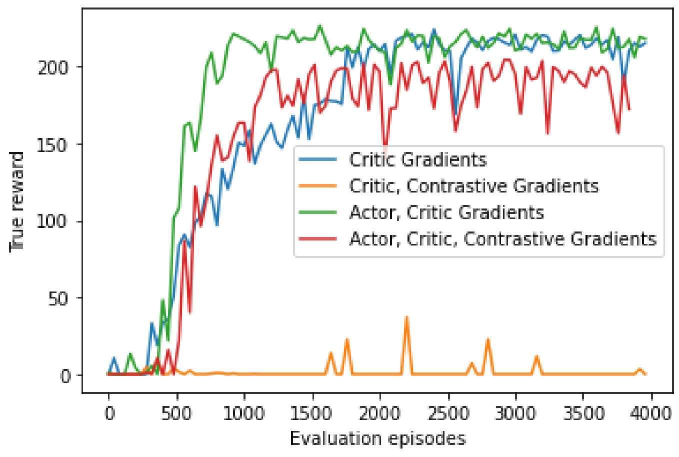


Figure 6. Above we can see the impact of varying which losses are backpropagated through the encoder on the point-mass environment. In this case, the four choices were 1. only critic, 2. critic and contrastive, 3. critic, contrastive, and actor, 4. critic and actor. The reason we can sometimes afford to not always backpropagate the contrastive losses is that in this experiment we were using a pretrained contrastive encoder to label the rewards. In every case, the critic gradients were included because the critic loss was very stable. On the y-axis is what we call the true reward. This reward is calculated using the state-space representation of the environment and gives 1 if $||s_{goal} - s_{current}||_2 < 0.03$, and otherwise 0. Since the maximum number of environment steps in this environment is 250, we would expect that the maximum possible true reward to be just under 250 (1 reward per timestep). In this experiment we saw that the best performance was with the actor and critic gradients, followed closely by only the critic gradients.

SAC. We also modified the evaluation procedure to instead be a measure of the success rate, since the aforementioned true reward was not as clear as would have liked. As a basic sanity check that the model is performing well, we first showed a comparison between the evaluation success rate of the inverse model and our RL agent (see 3). As mentioned before, our RL agent drastically outperformed the inverse agent. Within 100k iterations on the point-mass environment, the RL agent is able to reach an average 90% evaluation success rate on the point_mass environment, while the inverse model struggles to reach any goals.

Here we present an ablation for whether or not we use a pretrained encoder for relabelling rewards during our Hindsight experience relabelling. On the one hand, using a pretrained encoder reduces the noisiness of the labelled rewards because in successive iterations the same current observation and desired goal will have the same labelled reward. However, using a stationary reward encoder can also present a challenge because there are bound to be some inputs for which the labelled reward is not perfectly accurate. Then, this inaccuracy will remain the same for the entire training procedure and can possibly confound certain targets. On the other hand, an online, learning encoder during the training procedure will present more dynamic rewards that change from iteration to iteration, but on average will be more accurate than a pretrained encoder (assuming independent noise). Looking at the plots or this experiment, we see an interesting divide: On the simple point_mass environment, we find that the pretrained encoder actually performs better than using an online encoder (see figure 7). However, on the reacher environment we see that the online encoder significantly outperforms the pretrained encoder (see figure 8). This trend of encoders not working well on the point_mass environment has been a recurring theme in our studies. Indeed, when the environment is very simple, contrastive learning can struggle to learn meaningful encodings, as there are not many details in the environment that can be used to distinguish one state from another. Since point_mass states differ only in the position of the ball, in some cases the encoders will only learn trivial representations (i.e. the same representation for every state). Then, the hindsight-relabelled reward will always be the same for every state, so the policy's goal-reaching capabilities will collapse, both from the actor/critic and the labelled reward failing.

Another interesting ablation we performed is on the reward function used during the hindsight relabelling procedure. For this experiment, we used the reward 0 if $\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} > \epsilon$ and -1 else, for $\epsilon \in \{0.95, 0.97, 0.99\}$. We also used a dense reward, which was $\frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \|\mathbf{y}\|} - 1$ (using the -1 so that the reward is always less than or equal to 0, as is typical for HER methods). Interestingly, among the sparse rewards, we found that the lower cutoff of 0.95 was best, followed closely by 0.97 and 0.99 (see figure 9). Using these reward functions, we were able to reach a maximum evaluation success rate of about 0.9. More impressively, using the dense cosine-similarity reward function proved to be very successful. Within 20k iterations, we were already seeing consistent success rates around 99%,

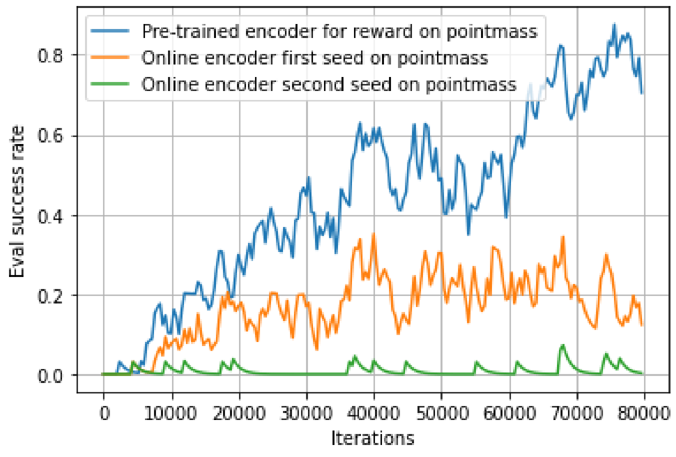


Figure 7. Above we can see the difference between online enocoders and pretrained encoders on the point-mass environment. As described, the pretrained encoders vastly outperform the online encoders.

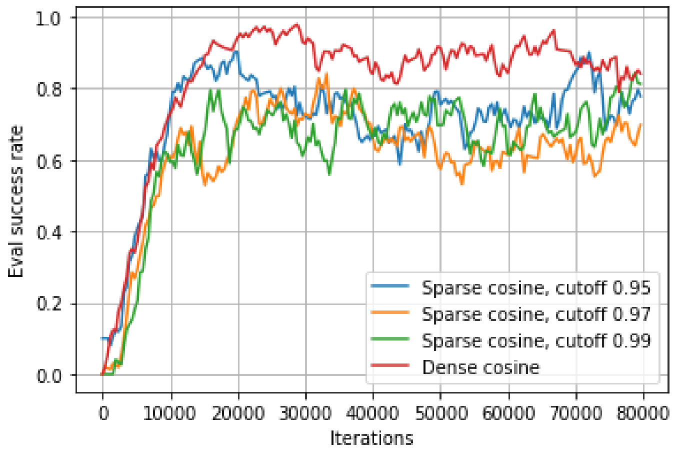


Figure 9. Above we can see the performance plot of the agent on the reacher environment as we vary the reward function that we use for relabelling. We see that the dense cosine function outperforms the others. At its best, it achieves about a 99% evaluation success rate.

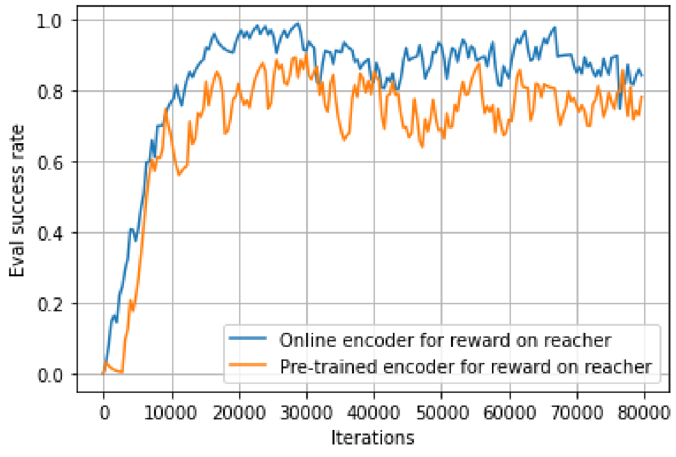


Figure 8. Above we can see the difference between online enocoders and pretrained encoders on the reacher environment. As described, online encoders achieved superior performance in this environment.

however the performance began to drop off after more training iterations. We hypothesize that the reason for this behavior is that 1) the dense reward encodes more information than the sparse reward, and 2) as the contrastive encoder gets more and more optimized, it is better at telling images apart which makes the reward sparser. Our first point suggests that using a dense reward should always be better, as the agent will have more information about how "closeä current observation and goal observation really are. Our second point explains the drop-off in performance as the agent continues training. Indeed, as the contrastive encoder further optimizes its own loss, the "dense"reward becomes more and more sparse, since the encoder will be able to distinguish between more and more similar-looking states. This suggests that perhaps some further form of regularization should be applied to the reward labelling to ensure that the contrastive encoder never becomes overly optimized for distinguishing one state from the others.

Another reward-function study that we performed dealt with how the agent performed with l2-distance functions compared to cosine reward functions. For the l2-distance reward functions, we encoded the next observation o_{t+1} to get x_{t+1} and the goal observation o_g to get x_g. Then, we calculated either the sparse reward as 0 if $\|x_{t+1} - x_g\|_2 < \epsilon$, -1 else, or a dense reward as $\exp\left(-\|x_{t+1} - x_g\|_2\right) - 1$, where both functions are constructed such that the reward is always non-positive. One drawback from this method is that it will likely not generalize as well to other environments, as it depends heavily on the magnitude of the encodings generated by the encoder. This is unlike the cosine similarity function, which depends only on the orientations of the encodings, rather than the magnitudes. For this experiment, we ran the agent on the reacher environment with the only variable changing being the type of reward function that we use. As expected, the best performance came from the dense cosine similarity reward function (see figure 10). This was followed closely by the sparse cosine similarity reward function, and with roughly equal performance in last place were the sparse and dense exponential reward functions. While we could likely have improved the performance of the sparse exponential reward function by fine-tuning the value of epsilon, we wanted to cut down on the number of hyperparameters to make the method more usable for other groups and settings.

As a test of the robustness of this method, we also tried running it on other environments like the OpenAI FetchReach-v1 environment. Without modifying any hyperparameters or pretraining any encoders, we were able to train a fairly successful agent on the FetchReach environment. Our best agent reached a smoothed 95% evaluation success_rate (see figure 11), but the training was quite unstable. Interestingly, the best performance of the agent came in the sparse exp(-l2) setting, which contradicts our previous logic that dense rewards would tend to be better and that using the cutoffs for the exp(-l2) reward function would require lots of tuning for

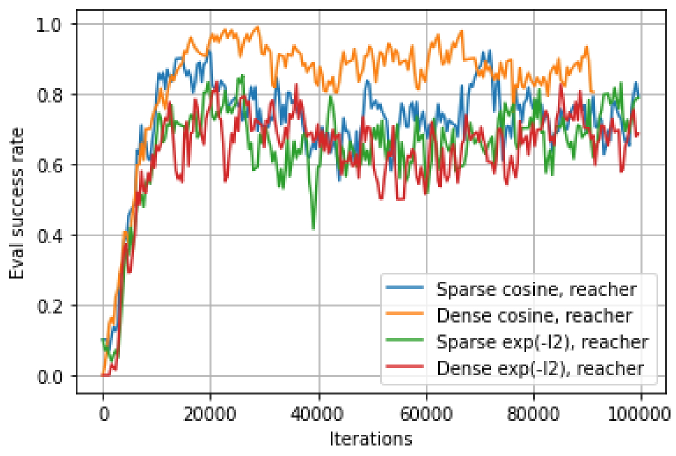


Figure 10. Above we can see the performance plot of the agent on the reacher environment as we vary the reward function that we use for relabelling. We see that the dense cosine performs better than all the other methods, including those with exp(-l2) reward.

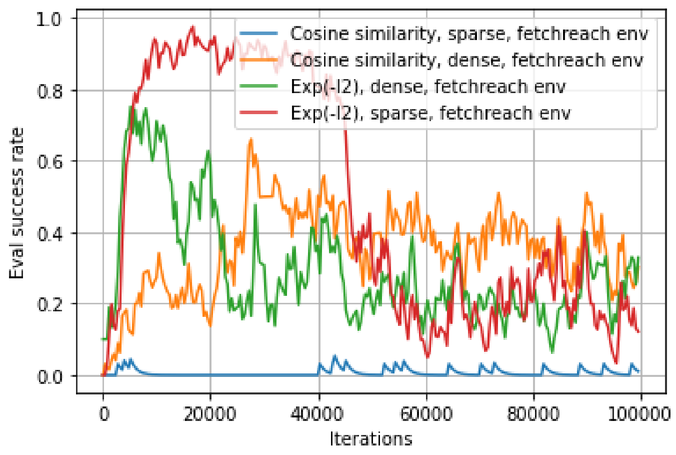


Figure 11. Above we can see the performance plot of the agent on the fetch-reach environment as we vary the reward function that we use for relabelling. In this setting, the best reward function was the sparse exp(-l2) distance reward function. We also see that performance decreases after the initial rise, which is an issue we are still trying to fix, but is present in most of our experimental runs. As mentioned before, we conjecture that it is due to the contrastive encoder becoming too well optimized at differentiating between states, meaning the rewards become too sparse.

new environments. However, we still saw that the dense cosine reward outperformed the sparse cosine similarity reward, as our intuition had suggested. This result is promising, as it shows that the method can be generalized to new environments with relative ease.

V. Future Work

One notable flaw in our method is that it relies on humans to predefine what is meant by a "goal". In the reacher and point_mass environments, we defined a goal to be a random rendering of the environment based on a random, valid, initial starting state. However, the assumption that the model can come up with its own goals does not always hold. Indeed, in real-world tasks, goal-selection is non-trivial and often involves human intervention to define what is desired of the agent. Here, we propose a convenient trick that can be used to ensure randomness and diversity in goal-selection. Since our model has already learned a contrastive encoder which can quantify the similarity between any pairs of images, we can use this to remove goals which are already similar to a previously selected goal. First, we sample a random batch of goal observations, from which we then plan on picking a diverse subset. Now, for this batch, we choose a random goal as our starting point. Next, we compute the similarity between this goal and all the other goals we sampled. Then, we choose the goal that corresponds with the least similar image to the already chosen goal by taking the argmin of the computed similarity_scores. We then add this next goal to our selected_goals and repeat this process with the only change being that in future iterations we take the max over the similarity scores along the ßelected_goals"dimension, so we can greedily select the goal whose maximum similarity to any previously chosen image is as small as possible. Using this

Algorithm 4 Goal-Selection Algorithm

```
goals = replay_buffer.sample()
starting_idx = randint(0, len(goals))
selected_goals = [goals[starting_idx]]
i = 1
while i < num_desired_goals do
    similarity_scores = CURL.compute_similarity(goals, selected_goals)
    similarity_scores = max(similarity_scores, dim = 1)
    next_idx = argmin(similarity_scores)
    next_goal = goals[next_idx]
    i += 1
    selected_goals.append(next_goal)
end while
return selected_goals
```

method will allow the model to generate its own curriculum assuming that it sees some diverse goals during its exploration phase. This removes the requirement for humans to specifically define what is meant by diverse goals as now the agent is capable of its own goal selection. We conjecture that this type of self-generated goal reaching will be useful for pretraining large models, as the actor, critic, and encoders will be learning important features about the environment.

We also urge other groups to use our method (whose code will be available on github shortly) in a wider range of environments, including real-world robotics systems. This is an important step in proving the generalizability and usefulness of the proposals set forth in this paper.

VI. Conclusions

We introduced a novel method that takes advantage of contrastive encoders to label its own reward in pixel-based goal-conditioned settings. Our technique outperforms an inverse model on the DMControl point_mass environment and outperforms the CURL method in the reacher environment. We also demonstrate the ability of our model to generalize to other environments with more complex dynamics without the need for tuning hyperparameters. Lastly, we discuss modifications that can be made to the method that will allow for goal-selection and hence enable its own curriculum generation.

VII. Acknowledgements

I would like to thank Aravind Srinivas, Professor Lerrel Pinto, and Professor Pieter Abbeel for their mentoring, guidance, and advice in this project. I would also like to thank the course staff and my fellow students for their feedback throughout the year.

References

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In *2017 International Conference on Engineering and Technology (ICET)*, pages 1–6. IEEE, 2017.

[2] Marcin Andrychowicz, Filip Wolski, Alex Ray, Jonas Schneider, Rachel Fong, Peter Welinder, Bob McGrew, Josh Tobin, OpenAI Pieter Abbeel, and Wojciech Zaremba. Hindsight experience replay. In *Advances in neural information processing systems*, pages 5048–5058, 2017.

[3] OpenAI: Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Jozefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, et al. Learning dexterous in-hand manipulation. *The International Journal of Robotics Research*, 39(1):3–20, 2020.

[4] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[5] Scott Fujimoto, Herke Van Hoof, and David Meger. Addressing function approximation error in actor-critic methods. *arXiv preprint arXiv:1802.09477*, 2018.

[6] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint arXiv:1801.01290*, 2018.

[7] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9729–9738, 2020.

[8] David Held, Xinyang Geng, Carlos Florensa, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. 2018.

[9] Matteo Hessel, Joseph Modayil, Hado Van Hasselt, Tom Schaul, Georg Ostrovski, Will Dabney, Dan Horgan, Bilal Piot, Mohammad Azar, and David Silver. Rainbow: Combining improvements in deep reinforcement learning. *arXiv preprint arXiv:1710.02298*, 2017.

[10] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[11] Michael Laskin, Kimin Lee, Adam Stooke, Lerrel Pinto, Pieter Abbeel, and Aravind Srinivas. Reinforcement learning with augmented data. *arXiv preprint arXiv:2004.14990*, 2020.

[12] Michael Laskin, Aravind Srinivas, and Pieter Abbeel. Curl: Contrastive unsupervised representations for reinforcement learning. In *International Conference on Machine Learning*, pages 5639–5650. PMLR, 2020.

[13] Ashvin V Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals. *Advances in Neural Information Processing Systems*, 31:9191–9200, 2018.

[14] Bojan Nemec, Leon Žlajpah, and Aleš Ude. Door opening by joining reinforcement learning and intelligent control. In *2017 18th International Conference on Advanced Robotics (ICAR)*, pages 222–228. IEEE, 2017.

[15] Vitchyr H Pong, Murtaza Dalal, Steven Lin, Ashvin Nair, Shikhar Bahl, and Sergey Levine. Skew-fit: State-covering self-supervised reinforcement learning. *arXiv preprint arXiv:1903.03698*, 2019.

[16] Ozan Sener and Vladlen Koltun. Multi-task learning as multi-objective optimization. *Advances in Neural Information Processing Systems*, 31:527–538, 2018.

[17] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

[18] Yuval Tassa, Yotam Doron, Alistair Muldal, Tom Erez, Yazhe Li, Diego de Las Casas, David Budden, Abbas Abdolmaleki, Josh Merel, Andrew Lefrancq, et al. Deepmind control suite. *arXiv preprint arXiv:1801.00690*, 2018.

[19] Hado van Hasselt, Arthur Guez, and David Silver. Deep reinforcement learning with double q-learning, 2015.

[20] Yunzhi Zhang, Pieter Abbeel, and Lerrel Pinto. Automatic curriculum learning through value disagreement. *Advances in Neural Information Processing Systems*, 33, 2020.