IFT6163 Image-Based Temporal Difference Models (An Attempt)

Alexandre Brown

April 28, 2024

Abstract

Model-free approaches offer a powerful and general approach to solving various tasks. That being said, contrarily to model-based approaches, they often lack great sample efficieny and they can only learn from an often sparse scalar reward. Model-based approaches are able to learn from every transition and can offer great sample efficiency but often result in lower asymptotic performance when compared to model-free approaches. Temporal difference models (TDMs) are a variant of goal-conditioned value functions, they offer an interpolation between model-free and model-based approaches allowing us to get the best of both worlds. Temporal difference models have not been extended to images yet and were only tested on rewards that usually rely on precise (x,y,z) location of the agent and the goal. Motivated by the success of TDMs and inspired by other methods that showed that using a distance in the latent space is generally a better reward for images, this project is an attempt at extending TDMs to images while relying purely on goal image and an observation image.

1 [4 pts] Project Introduction (2 paragraphs, with a figure)

The project explores the possibility of using Temporal Difference Models Pong et al. [2020] with images directly. This has not been explored yet but could allow leveraging ideas from Nasiriany et al. [2019] which introduced LEAP, a **model-free** planning framework that uses Temporal Difference Models (TDM) as implicit model for planning over long horizon tasks. The project also leverages similar ideas from Nair et al. [2018] which showed that using a reward based on the latent representation of images is easier to optimize than working in the pixels space directly. The objective of this project is to combine various ideas from previous work such that we get an approach that relies purely on images for the observation, goal and reward while optimizing in latent space instead of optimizing for pixel values directly. This is an important area for robot learning because if we can better leverage reinforcement learning methods that rely only on images or their latent representation, then perhaps this could mean that we wouldn't need to rely on task-specific reward shaping such as needing to know the (x,y,z) position of the robot and its goal in order to compute a distance to the final goal. Relying purely on images opens up a lot of possibilities and facilitates deployment to

the real world since all we'd need is images from a regular RGB camera and a goal provided by the end-user (eg: via a smartphone camera).

The project can be summarized as a combination of Temporal Difference Models (TDMs) Pong et al. [2020] with ideas from LEAP approach Nasiriany et al. [2019] which proposed to use a VAE to encode images into a latent space. Contrarily to Nasiriany et al. [2019], we do not extend our approach to subgoals generation and focus on simple tasks where there is only a final goal that is provided to the policy, this was done to restrict the scope of the project. In contrast to the original TDM approach exploited in LEAP which relied on being able to compute a distance to a goal using precise information from the environment such as computing a distance between the agent's (x,y,z) coordinates and the subgoals and goal (x,y,z) coordinates, we instead rely on the latent representation of the image (observation) and the goal as our reward. We also limit the inputs that is fed to the policy to include only image latent and goal latent representations. This removes the assumption that is often made in goal-conditioned reinforcement learning where one usually needs precise (x,y,z) location of the goal and the agent. For instance, in settings such as the Franka Kitchen environment, as discussed in Gupta et al. [2019] and as described by figure 1, additional details such as position, rotation and angle measurements of specific objects related to the goal are typically used when training goal-conditioned RL approaches. This includes knowing the exact orientation of the microwave door hinge in order to achieve the goal of opening the microwave door for instance. These information are not information we can realistically expect to know in the real world which is why this project is important for real world applications.

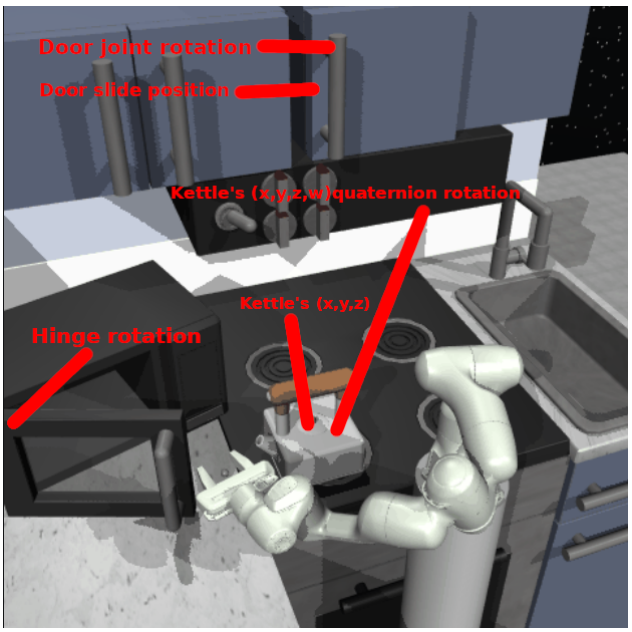


Figure 1: An example of a subset of detailed information that is usually used in goal-conditioned RL approaches for the Franka Kitchen environment. These information are not practical information we can obtain in the real world yet are often used when training in a simulated environment.

2 [2 pts] What is the related work? Provide references: (1-2 paragraph(s))

Reinforcement learning has shown great success in simulated environments Lu et al. [2023] however, most approaches still struggle with long-horizon visual tasks Ebert et al. [2018]. Traditional deep reinforcement learning usually aims at finding an optimal policy for a specific task François-Lavet et al. [2018]. This objective has been extended to a multi-goal objective via the framework of goal-conditioned reinforcement learning Plappert et al. [2018]. Approaches that fall under goal-conditioned reinforcement learning usually accept structured vector features as observations or unstructured data such as images Liu et al. [2022]. Acquiring cost functions that accurately reflect the distance between two images is a challenging problem Yu et al. [2019]. This is why goal-conditioned RL approaches usually require some type of distance to a goal that is available via a simulated environment such as precise (x,y,z) coordinates of the goal and the agent. Chane-Sane et al. [2021] generates a subgoal that is halfway between the state of the agent and the goal. It also requires estimation of a valid state distribution to generate subgoals that are valid. These limitations restrict the scope to which goal-conditioned reinforcement learning approaches can be applied. Some work like Ebert et al. [2018] Nair and Finn [2019] have tried to address this by utilizing a mixture of sel-supervised learning and visual subgoals generation to plan over a long horizon using image pixels directly. These approaches however also rely heavily on model-based approximation during planning which can be problematic as model-based methods can suffer from compounding errors which often leads to model-free methods outperforming them asymptotically Pong et al. [2020].

Temporal Difference Models (TDMs) introduced in Pong et al. [2020] are a variant of goal-conditioned value functions and they offer an interpolation between model-free and model-based approaches. TDMs can be trained using model-free training, they can be used for model-based control and offer great sample efficiency. That being said, TDMs have not been extended to images so it is not clear if their advantages still apply to this type of modality. TDMs have been used for long-horizon model-free planning by Nasiriany et al. [2019]. Although LEAP can work with images and plans in the latent-space once the TDM model is trained, it still relies on training a TDM value function that itself requires knowing precise information like distance to a goal (x,y,z) position. In contrast, Nair et al. [2018] uses only images as observations and as goal to train a goal-conditioned policy. It computes a distance in the latent space to get a well-shaped reward function for images and shows that computing the distance in the latent space vastly outperforms distance in the pixel space. RIG, introduced in Nair et al. [2018], provides great insights on how to decouple our approach from task-specific environment details like goal position as (x,y,z) coordinates and instead rely purely on images. Our project aims at combining the best of both of these approaches (TDMs and RIG). We also rely on TD3 Fujimoto et al. [2018] as our base approach for model-free learning.

3 [2 pts] What background/math framework have you used? Provide references: (2-3 paragraph(s) + some math)

Generally speaking, the project uses the framework of goal-conditioned reinforcement learning Plappert et al. [2018] which includes a learning objective that can be formally described as follow :

$$J(\pi) = \mathbb{E}_{a_t \sim \pi(\cdot|s_t,g), g\sim p_g, s_{t+1}\sim p(\cdot|s_t,a_t)} \left[\sum_t \gamma^t r(s_t, a_t, g) \right] \tag{1}$$

The agent tries to maximize the expected discounted reward given an action from a goal-conditioned policy, a goal from a goal distribution and a state from the environment. We consider a finite-horizon, goal-conditioned MDP defined by a tuple $(S, G, A, p, R, H, \tau, \rho_0, \rho_g)$ where S is the set of states, G is the set of goals, A is the set of actions, $p(s_{t+1}|s_t, a_t)$ is the time-invariant and **unknown** dynamics function, R is the reward function, τ is the maximum planning horizon for the TDM models, H is the maximum number of steps we take in the environment during a rollout, ρ_0 is the initial state distribution and ρ_g is the goal distribution. We consider the case where the goals reside in the same space as the states.
An important aspect of goal-conditioned MDPs is the goal-conditioned value function. TDMs consist of a goal-conditioned policy and a goal-conditioned value function Pong et al. [2020]. The intuitive interpretation of the TDM value function is that it tells us how close the agent will get to a given goal state s_g after τ time steps, when it is attempting to reach that state in τ steps. **Temporal Difference Models do not use a γ discount factor**, they instead introduce a different mechanism for aggregating long-horizon rewards via an additional input τ. More formally we define the value function and reward as follow :

$$Q_\phi(s_t, a_t, s_g, \tau) = \mathbb{E}_{p(s_{t+1}|s_t,a_t)} \left[R_{\text{TDM}}(s_{t+1}, s_g)\mathbb{1}[\tau = 0] + Q_\phi(s_{t+1}, \mu_\theta(s_{t+1}), s_g, \tau - 1)\mathbb{1}[\tau \neq 0] \right] \tag{2}$$

$\mathbb{1}$ is the indicator function that returns 1 only when the predicate is true. So the left side is activated only when $\tau = 0$ for instance while the right side is activated in the equation when $\tau \neq 0$. μ_ϕ is the mean action network that is used in TD3 and while various time-varying reward functions can be used, we used the following for the temporal difference models (TDMs):

$$R_{\text{TDM}}(s_t, s_g) = -||q_{w_1}(s_t) - q_{w_1}(s_g)||_p \tag{3}$$

Where $q_{w_1}(x)$ is a VAE encoder model that returns a latent representation of an image and p is the order of the norm, we tested both $p = 1$ and $p = 2$.

We can recover a policy from our Q-function using TD3 but it could also be done using model predictive control (MPC) or via a simple argmax :

$$a_t = \arg\max_a Q_\phi(s_t, a, s_g, \tau) \tag{4}$$

In our case, we focus on continuous action and state space so we went with TD3. Also, to facilitate training, we defined our Q-function as follow :

$$Q_\phi(s_t, a, s_g, \tau) = ||f(s_t, a_t, s_g, \tau) - s_g||_p \tag{5}$$

Where f is a neural network predicting the latent representation that will be reached when trying to reach s_g in τ timesteps.

To train the VAE model that is used to encode the observation and goal images to a latent space, we used the following loss :

$$\mathcal{L}(w_1, w_2; \mathbf{x}, \beta) = \underbrace{\mathbb{E}_{q_{w_1}(\mathbf{z}|\mathbf{x})}[\log p_{w_2}(\mathbf{x}|\mathbf{z})]}_{\text{Reconstruction Loss}} + \beta \times \underbrace{D_{KL}(q_{w_1}(\mathbf{z}|\mathbf{x})||p(\mathbf{z}))}_{\text{KL Divergence Loss}} \tag{6}$$

Where w_1 are the parameters of the encoder, w_2 are the parameters of the decoder, $\mathbf{x}$ is the input data, β is a hyperparameter that balances the reconstruction accuracy and the KL divergence, $q_{w_1}(\mathbf{z}|\mathbf{x})$ is the variational approximation to the true posterior $p(\mathbf{z}|\mathbf{x})$, D_{KL} is the Kullback-Leibler divergence and $p(\mathbf{z})$ is the prior distribution over latent variables $\mathbf{z}$, here it is assumed to be a Gaussian distribution.

4 [4 pts] Project Method, How will the method work (1-2 pages + figure(s) + math + algorithm)

Our approach consists of multiple steps. First we need to augment the environments to generate goals as images.

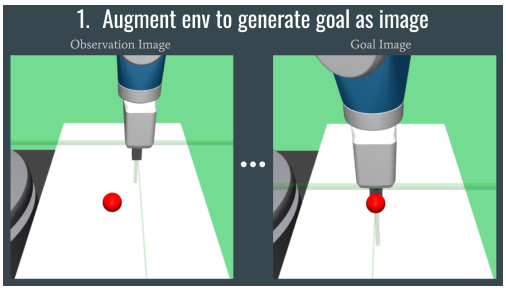


Figure 2: FetchReach environment augmented with goal as images.

Then we need to generate a VAE dataset for training our VAE model. To do this, we performed random actions in the environments while also saving the image goal associated with the steps perform in the environment. For each environment, we generated a dataset of 65 536 frames where 50% of it were goal frames and 50% of it were random observations images. This was done under the intuition that providing frames where the goal is present in the dataset might help the VAE model to better represent goals and frames that are similar to it which is crucial for an accurate reward computation in our case. The output latent dimension for the VAE model was 256. I tested using 128 and noticed that for some environments, that would result in the goal not being present in the reconstructed image. I also tested 512 and 1024 and saw no noticeable drop in performance when going to 256 but going to 256 made inference speed much faster and more memory efficient therefore I decided to go with 256 for all environments. The following figures illustrate some samples of random observations and goals for the various environments we tested :

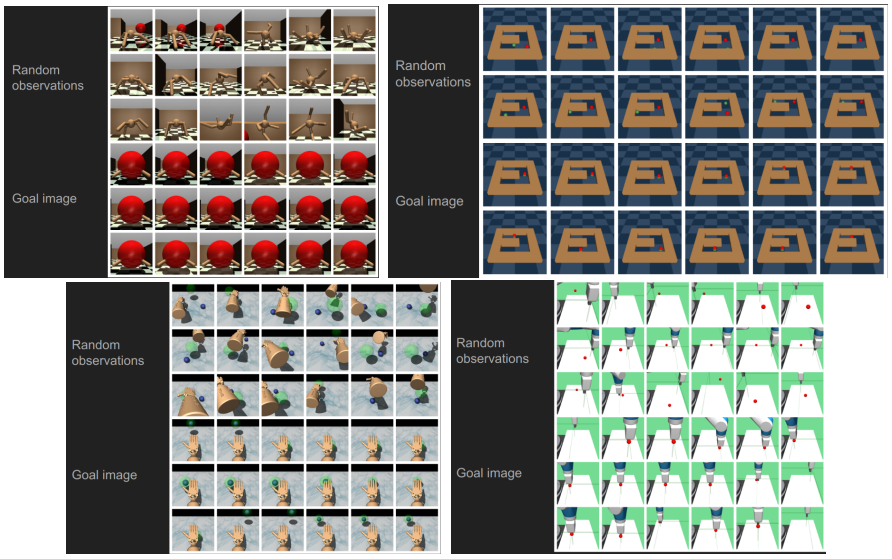


Figure 3: VAE dataset samples for the various environments. Top images are random observations and bottom images are goal images.

Once we have a VAE dataset, we need to train the VAE model Kingma and Welling [2019]. To do so, we split the vae dataset into a training and validation set using a (0.75/0.25) split. This means that the training set consisted of 49 152 frames and the validation set that was used for hyperparameters optimization and evaluation contained 16 384 frames. Motivated by Bowman et al. [2016], we used a linear β annealing strategy during training to stabilize the tradeoff between minimizing the KL divergence loss and the reconstruction loss for the VAE.

Once the VAE is trained, we need to train a TDM policy and value function Pong et al. [2020] as illustrated by algorithm 1. We used TD3 Fujimoto et al. [2018] for the goal-conditioned TDM policy instead of an argmax as this allowed us to use TDM in continuous action space. For the relabelling strategy we used the same one as presented in Pong et al. [2020], that is we relabel the time horizons as a random value between 0 and the maximum planning horizon. For the goal, we relabelled the goal at time t with a random observation (in latent space) at time $t + k$ where $k \geq 1$. In other words, the goal is relabelled as being a random future frame (encoded in the latent space).

Algorithm 1 TD3 Visual Temporal Difference Model Learning

Require: Random networks Q_{ϕ_1}, Q_{ϕ_2}, $Q_{\phi'_1}$, $Q_{\phi'_2}$, μ_θ, $\mu_{\theta'}$, replay buffer $\mathcal{B}$, trained q_{w_1} encoder

for $n = 0, \ldots, N-1$ episodes **do**
  $s_0 \sim \rho_0$ // Get initial observation
  $s_g \sim \rho_g$ // Get goal image
  for $t = 0, \ldots, H-1$ time steps **do**
    $a_t^* = \mu_\theta(q_{w_1}(s_t), q_{w_1}(s_g), \tau - t)$ // Using TD3 Policy
    $a_t = \text{AddNoise}(a_t^*)$ // Noisy exploration
    $s_{t+1} \sim p(s_t, a_t)$, and store $\{s_t, a_t, s_{t+1}\}$ in the replay buffer $\mathcal{B}$ // Step environment
    for $i = 0, I-1$ iterations **do**
      Sample M transitions $\{s_m, a_m, s'_m\}$ from the replay $\mathcal{B}$.
      Relabel time horizons and goal states $\tau_m, s_{g,m}$
      $z'_m = q_{w_1}(s'_m)$ // Encode next observation
      $z_{g,m} = q_{w_1}(s_{g,m})$ // Encode goal
      $a' = \mu_{\theta'}(z'_m, z_{g,m}, \tau_m - 1) + \epsilon, \epsilon \sim \text{clip}(\mathcal{N}(0, \sigma), -c, c)$ // Get target action
      $y_m = -\|z'_m - z_{g,m}\| \mathbb{1}[\tau_m = 0] + \min_{i=1,2} Q_{\phi'_i}(z'_m, a', z_{g,m}, \tau_m - 1) \mathbb{1}[\tau_m \neq 0]$
      $z_m = q_{w_1}(s_m)$ // Encode observation
      $\phi_i = \text{argmin}_{\phi_i} \sum_m (y_m - Q_{\phi_i}(z_m, a_m, z_{g,m}, \tau_m))^2 / M$ // Update critics
      if $t \mod d$ **then**
        Update μ_θ using TD3 (encode obs and goal using q_{w_1})
        Update target networks $Q_{\phi'_1}$, $Q_{\phi'_2}$, $\mu_{\theta'}$ using polyak average.
      end if
    end for
  end for
end for

5 [2 pts] What new skills are you(s) learning from this project?

Throughout this project, I learned how to more efficiently read RL papers, how to combine ideas from various papers and how to implement algorithms presented in various papers. This projects allowed me to learn more about TorchRL Bou et al. [2023] (PyTorch new RL library). I learned how to setup project dependencies and how to use Hydra to handle configurations. I also used CometML to log my metrics, images, videos during training, this was extremely valuable to debug and to get insights on the behaviour of my policy. I learned how to reliably generate a VAE dataset in a memory efficient format using h5py. This allowed me to generate datasets that would usually not fit in RAM memory, the file format I used allowed me to append new data without reading the entire dataset and allowed for efficient reading as well. I also learned how to efficiently train a VAE model. Initially, training a VAE was challenging as there can be some instabilities and exploding losses since the network starts off with a weak signal from the encoder and therefore would struggle to reconstruct the samples. I learned to use different scheduling strategy to smooth out the learning process and make it extremely stable and reproducible. I learned how to generate image observation from an environment and how to manipulate the environment to get a goal as an image. Once I figured a scalable way to do this, it allowed me to apply the same principle to multiple environments quite easily. I learned how to implement temporal difference models and how to train them using a model-free approach such as TD3.

Since my project was attempting something that had not been done with TDMs, I learned that it can be quite challenging to debug issues when nobody has laid the ground for you previously. I realized that perhaps my approach was too naive and maybe I should've started from a visual-based reinforcement learning approach instead of starting from TDMs and attempting to use ideas from various visual-based reinforcement learning approach on top of TDMs. By training policies and value functions purely on images/goals in the latent space, I learned that naively using an image as input or goal can be quite challenging as the agent can learn to cheat its way into a low latent space distance while getting further from the real objective. This project provided me with valuable insights on the various challenges of training image-based reinforcement learning approaches.

6 [6 pts] Experiments and Analysis, (1-2 pages + envs(s) + math + draft learning curves)

All the code for this project was made publicly available :
https://github.com/AlexandreBrown/VisualTDM.

6.1 VAE Training

For all environment, a linear learning scheduling of 1 cycle with ratio 0.5 was used meaning that the weight of the KL-Divergence loss starts at 0 then linearly increases to 1.0 at 50% of the training. A β value of 1 was used for the KL-Divergence term in the loss. For all environments, the VAE architecture was a CNN with an encoder that consists of hidden dims : [64,32,16], hidden kernels size : [4,4,4], hidden paddings : [1,1,1], hidden strides: [2,2,2], output kernel size : 4, output padding : 1, output stride: 2, using batch norm and a ReLU activation function. The decoder used a leaky ReLU activation function, hidden dims: [16,32,64,128], hidden kernels size : [4,4,4], hidden paddings: [1,1,1,1], hidden strides: [2,2,2,2], leaky ReLU negative slope : 0.2, using batch norm. The output latent dimension was 256 as this offered the best tradeoff between representation, computation demand and model size. The Adam optimizer was selected with a learning rate of 0.001. The MSE loss was used with a train batch size of 512 and a train/validation split of 0.25. The model was trained for 50-200 epochs depending on the environment but convergence happened very quickly therefore it is also possible to stop training even before that. The best validation model was saved during training. Here is a learning curve that shows the validation epoch loss with respect to the number of steps in various environments during training. For more details on all the environments and training hyperparameters, you can check the publicly available experiment data : https://www.comet.com/alexandrebrown/visual-tdm/view/new/panels.

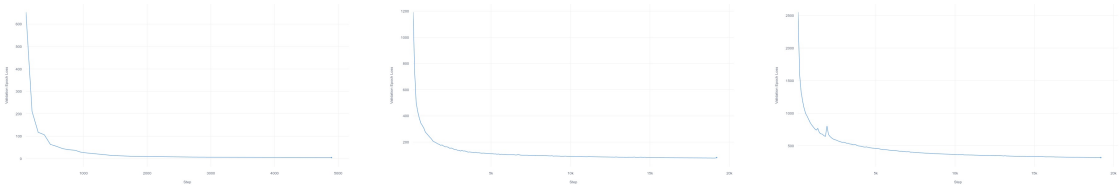


Figure 4: Validation epoch loss for InvertedPendulum (left), FetchReach (middle) and AntMaze (right).

Figure 4 shows that once a few tricks are used to properly train a VAE, training can be quite smooth and efficient. To keep the metric fair, these are plots of the epoch mean validation loss when using no annealing meaning that although the VAE is trained using annealing of the β term, the loss used for logging the best model and to assess the performance of the model was computed using a weight of 1.0 for the KL-Divergence loss part. This prevented seeing a lower loss simply because the KL-Divergence term in the loss was being annealed during training.

6.2 Baseline Training

For the baseline, I chose TD3 Fujimoto et al. [2018] as this offered a solid model-free approach that can handle continuous action and state space. The baseline receives as input all observation information as described here :
https://www.gymlibrary.dev/environments/mujoco/inverted_pendulum/#observation-space
https://robotics.farama.org/envs/fetch/reach/#observation-space
https://robotics.farama.org/envs/maze/ant_maze/
https://robotics.farama.org/envs/maze/point_maze/
https://robotics.farama.org/envs/adroit_hand/adroit_relocate/
https://robotics.farama.org/envs/franka_kitchen/franka_kitchen/
For all TD3 trainings (baseline), a dense reward representing the euclidean distance between the agent (x,y,z) position and the goal (x,y,z) position were used besides for InvertedPendulum where the baseline was not a goal-conditioned implementation as it wasn't necessary. Here are some plots showing the performance of TD3 on some tasks :

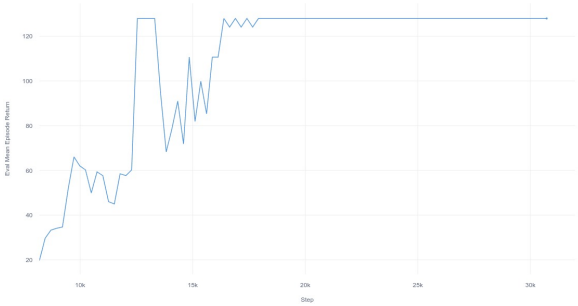


Figure 5: TD3 Eval Mean Episode Return for InvertedPendulum. We can observe a quick convergence at around 20k steps. Model architecture is hidden layers size [64,64], ReLU activation function, no batch norm. Tanh output activation function for the Q-function and identity function for the policy. The Q-function receives state, action and desired_goal (x,y,z positions) as inputs while the policy receives state and desired_goal as inputs.

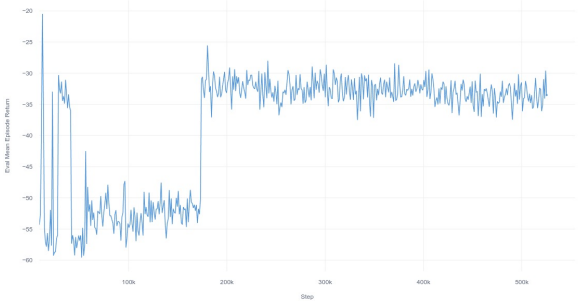


Figure 6: TD3 Eval Mean Episode Return for FetchReach. We can observe that TD3 learned a suboptimal policy. Model architecture is hidden layers size [256,256], ReLU activation function, with batch norm. Tanh output activation function for the Q-function and identity function for the policy. The Q-function receives state, action and desired_goal (x,y,z positions) as inputs while the policy receives state and desired_goal as inputs.

6.3 TD3 TDM Training

For all TDM trainings, the VAE encoder is used to encode the observation image and desired goal image to a latent representation. These latent representations are passed as inputs to the policy and value function along with the planning horizon τ. The policy and Q-function are MLP networks. For the Q-function it is predicting the latent representation that will be achieved when trying to reach the latent goal in τ steps starting from the observation latent representation. Various hidden layers sizes were tested ([64, 64], [64, 64, 64], [128, 128], [128, 128, 128], [256, 256], [400, 300]) but no configuration produced significantly different results. A ReLU activation function was used for the hidden layers of the MLP. A TanH output activation function was used for the policy and an identity function was used for the Q-function. For more details, you can check the experiments on CometML https://www.comet.com/alexandrebrown/visual-tdm/view/new/experiments.

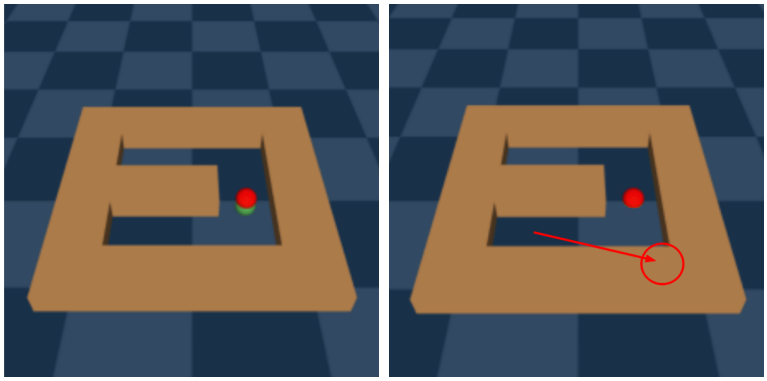


Figure 7: Left : Desired goal image received as input, Right : Achieved goal image from TD3 TDM model. The policy learned to cheat its way into a low latent representation distance by hiding itself in a corner.

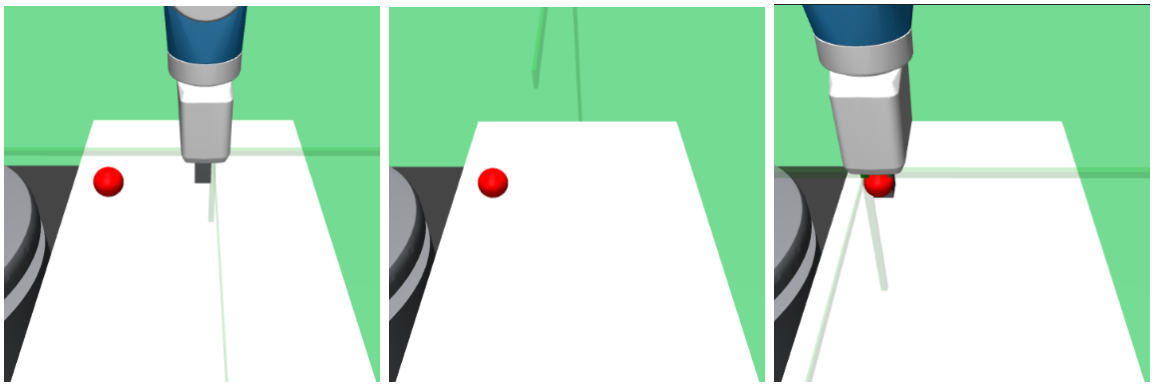


Figure 8: Left : First frame of the trajectory, Middle : Achieved goal image from TD3 TDM, Right : Desired goal image for this trajectory.

Figures 7 and 8 illustrates the results we obtained from TD3 TDM when using only the observation image and the desired goal image along with the planning horizon τ as inputs.

We can see that the policy learned to cheat its way into a low latent space distance with the desired goal image latent representation. In the case of the PointMaze environment (figure 7), we can see that the agent learned to go hide in a corner that is not visible by the camera and this actually produces an achieved goal latent representation that has a very low distance with the desired goal latent representation. The same phenomenon was observed on the FetchReach environment, again we see that the agent learned to hide its robotic arm away from the camera. It seems like the agent would rather not take any risk and have a latent representation that is only representing the goal and not the robotic arm instead of trying to position the robotic arm somewhere that might not be close to the goal.

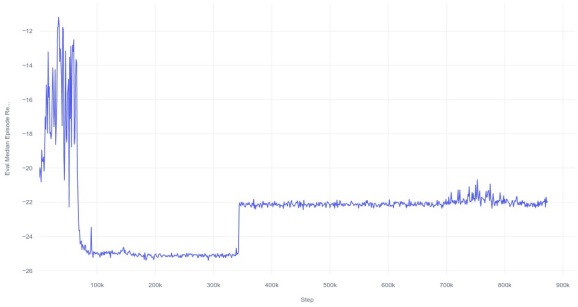


Figure 9: TD3 TDM Eval median episode return (based on latent representation distance reward) for the InvertedPendulum environment.

Figure 9 shows that even for environments where TD3 converged quickly, TD3 TDM using only images still struggles to find an optimal solution and instead converges to a suboptimal solution.

6.3.1 Why?

The following are some hypotheses on why TD3 TDM (using images only) did not converge and found this suboptimal behaviour instead.

Figure 10: Eval mean episode return. Left : Using (x,y,z) position to compute the reward for logging purposes only, Right : Using latent representation distance reward.

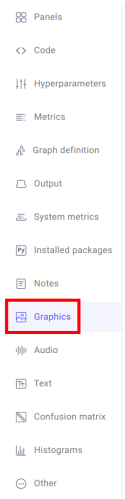
Figure 10 shows on the left that the mean episode return was getting worse on the actual task of getting closer to the goal while the mean episode return when using the latent representation as reward (right image) was actually improving. This means that the agent is getting better at getting a lower latent representation distance and it thinks it is doing

better at the original task but it is not, in fact it is getting worse. This could be because the latent representation outputted by the VAE encoder does not capture enough semantic information for the policy to get closer to the actual goal.

A VAE is known to be not the best at encoding and not the best at decoding (it's ok at both but not the best at either). For instance, if we are only interested in a good representation and do not need to decode the representation, then recent progress in self-supervised learning approaches such as SimCLR (Chen et al. [2020]), MoCo (He et al. [2020]) or BYOL (Grill et al. [2020]) have shown success in offering a great representation that can be used for classification even when only a linear classifier is added on top of the representation highlighting the strength of the representation that is obtained from these approaches. For decoding-only purposes, it has been shown that diffusion models Yang et al. [2024] tend to outperform other methods. Therefore it's possible that using a different mechanism than a VAE to obtain a good latent representation might yield better results. It's also possible that the issue comes from the fact that we're using an MLP to predict the latent representation in our Q-function and that using something like a Conditional Variational Auto Encoder Harvey et al. [2022] might result in much better performance as we would be using a method that can more easily take advantage of the goal-conditioning aspect of our Q-Function. It's also very important to note that we did not try to add any regularization term in the Q-function loss that would force the generated latent representation to be close the the VAE prior $p(z)$. As a result, it means that nothing guarantees us that the predicted latent representation from our Q-function will still be in the same space as our VAE encoder. This could explain why the goal latent representation might not match our predicted Q-function latent representation. Approaches like Nasiriany et al. [2019] have incorporated such regularization and perhaps this could've helped us achieve better results as well.

7 [2 pts] Video Results

Images and videos of the various experiments can be found here : https://www.comet.com/alexandrebrown/visual-tdm/view/new/experiments, simply select an experiment, then navigate to the "Graphics" tab.



Here are some images that represent reconstructed images from the VAE during trainig :
Ant Maze:
https://www.comet.com/alexandrebrown/visual-tdm/e9f987d10f8449569d58f21cc1a74bd3?experiment-tab=images
Hand Relocate:
https://www.comet.com/alexandrebrown/visual-tdm/b8dbfacffae9488aa209108bcbfb5aff?experiment-tab=images
Fetch Reach:
https://www.comet.com/alexandrebrown/visual-tdm/dcb7f2bc195a409ab854d6303f28f6d1?experiment-tab=images
Inverted Pendulum:
https://www.comet.com/alexandrebrown/visual-tdm/5a3ca05bc3424ecb973f99c171845d22?experiment-tab=images
Franka Kitchen:
https://www.comet.com/alexandrebrown/visual-tdm/ed54d8262ed5452986c51751d4722ca7?experiment-tab=images

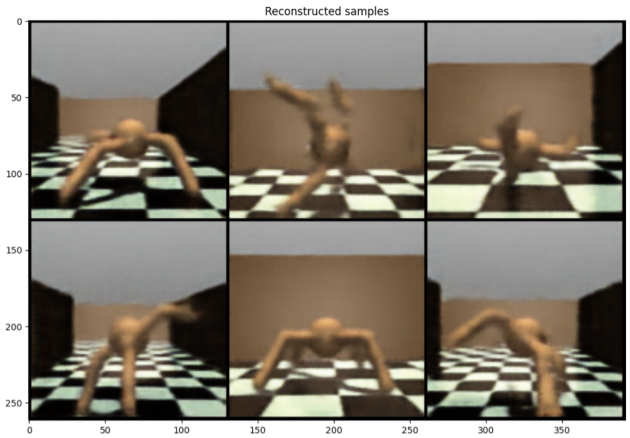


Figure 11: VAE reconstructed samples (validation set) for the Ant Maze environment.

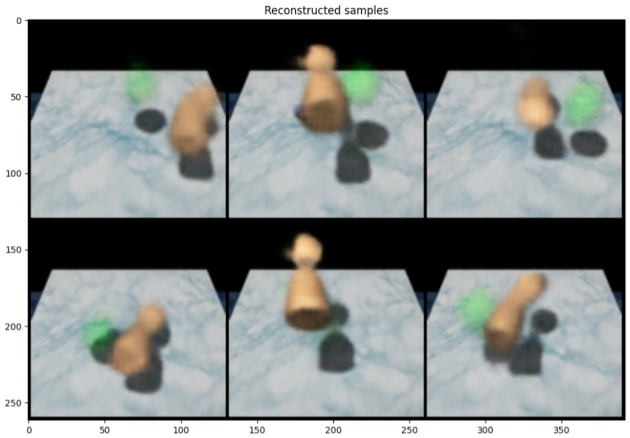


Figure 12: VAE reconstructed samples (validation set) for the Hand Relocate environment.

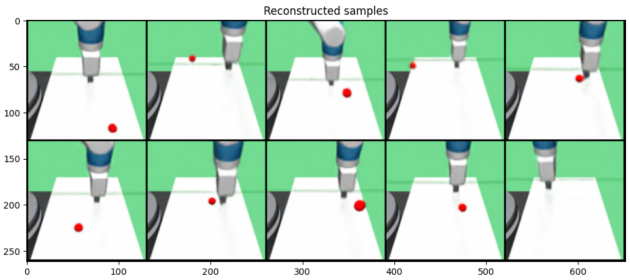


Figure 13: VAE reconstructed samples (validation set) for the Fetch Reach environment.

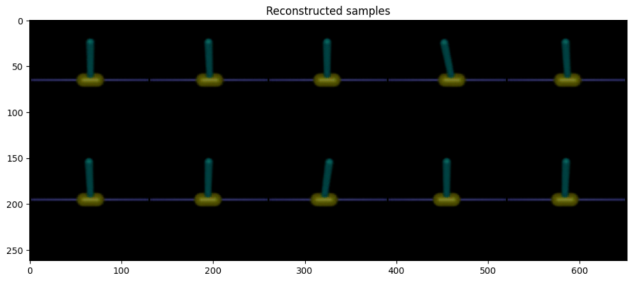


Figure 14: VAE reconstructed samples (validation set) for the Inverted Pendulum environment.

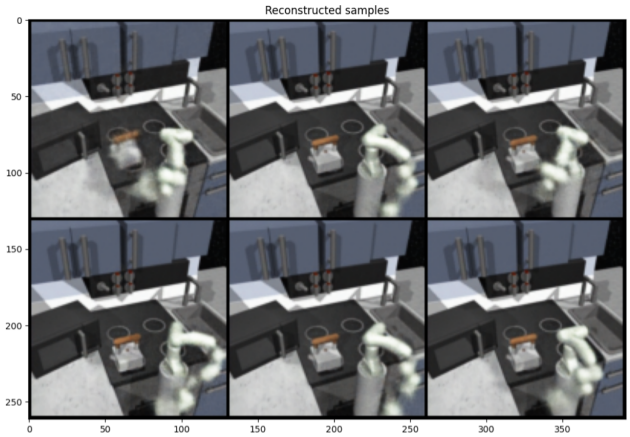


Figure 15: VAE reconstructed samples (validation set) for the Franka Kitchen environment.

Here is a link of videos and images for the TDM policy on Fetch Reach environment : https://www.comet.com/alexandrebrown/visual-tdm/9d0bfff9926a406e9ae9b8cbc4312e6e?experiment-tab=images.
Here is a link to the videos for the TD3 policy on Inverted Pendulum : https://www.comet.com/alexandrebrown/visual-tdm/efb12388e07d4dea9ff08a23a55a66fb?experiment-tab=images

8 [2 pts] Conclusions

I have learned a lot about VAEs, temporal difference models, the tradeoffs between model-based and model-free deep RL, planning and visual goal-conditioned reinforcement learning. A great literature overview provided me with valuable insights for the rest of the project and to guide which aspects of which approaches I wanted to combine and try out. I also learned about TorchRL, how to quickly setup a deep RL environment, add logging using Comet ML and video logging that is environment agnostic using TorchRL. Overall TorchRL helped me explore the environments quickly and implement training code without re-implementing from scratch core components such as memory-efficient replay buffers, data loaders and environment loading. I also learned how to define custom losses for training, load environments from various third parties like gym-robotics while keeping my code agnostic to the environment details and I also learned how to define RL training loops and how to use different type of replay buffer (on disk, in-memory, lazy, tensor-based etc). With hindsight, I think that I should have started with a visual reinforcement learning approach or framework that is known to work for images and then I should've applied TDMs on top of it. Instead I started from TDMs and tried to combine some ideas from various papers to make it work for images. The approach I took was naive and probably starting from a good visual RL base framework would've helped here. I also realized that, it was hard to compare progress because my baseline was vanilla TD3 and even TD3 would not converge on some environments. I should've started from a visual RL framework and use it as a baseline instead.
Overall the scope of the project was still reasonable but it would have helped to start earlier. Unfortunately my teammates dropped the course so this slowed me down as I had to completely re-think about a new project, this gave me 1.5-2 months to do the project alone. But it also allowed me to dive deeper into goal-conditioned RL and I learned a lot from it. I am grateful for the lessons I learned along the way and this provided me with great insights for future research directions.

9 [2 pts] How was the work divided?

Student Name:Alexandre Brown (Did 100% of the work)

1. Paper literature review (Goal-Conditioned RL, latent representation, planning).
2. Setup github repository
3. Setup dependencies and reproducible environment
4. Code exploration script to easily explore an environment and generate videos, this allowed me to understand the environment from random actions.
5. Code environment wrappers to augment the environment with observation and goal as image.
6. Code VAE dataset generation code. This was an important part of the code that runs an environment, collects data, logs images and videos of the results and saves the dataset.
7. Code VAE training. This consisted of loading the vae dataset and train the VAE model as well as log the metrics on CometML and save the trained model on CometML.
8. Code TD3 training. This was done to obtain a baseline performance on various environments. It includes coding TD3 from scratch, the training loop and logging of the metrics, images and videos.
9. Code TD3 TDM training. This is the code for the attempt at combining TD3 and TDMs with various approaches to extend it to images. It automatically downloads the VAE model from CometML and loads it to be used during training to encode the observation and goal images into a latent representation. The code also does the end-to-end training for the temporal difference models (value functions and policy).
10. Use Hydra configuration to modularize training code.
11. Analyze results and graphs.
12. Generate report.

References

Albert Bou, Matteo Bettini, Sebastian Dittert, Vikash Kumar, Shagun Sodhani, Xiaomeng Yang, Gianni De Fabritiis, and Vincent Moens. Torchrl: A data-driven decision-making library for pytorch, 2023.

Samuel R. Bowman, Luke Vilnis, Oriol Vinyals, Andrew M. Dai, Rafal Jozefowicz, and Samy Bengio. Generating sentences from a continuous space, 2016.

Elliot Chane-Sane, Cordelia Schmid, and Ivan Laptev. Goal-conditioned reinforcement learning with imagined subgoals, 2021.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations, 2020.

Frederik Ebert, Chelsea Finn, Sudeep Dasari, Annie Xie, Alex Lee, and Sergey Levine. Visual foresight: Model-based deep reinforcement learning for vision-based robotic control, 2018.

Vincent François-Lavet, Peter Henderson, Riashat Islam, Marc G. Bellemare, and Joelle Pineau. An introduction to deep reinforcement learning. *Foundations and Trends® in Machine Learning*, 11(3–4):219–354, 2018. ISSN 1935-8245. doi: 10.1561/2200000071. URL http://dx.doi.org/10.1561/2200000071.

Scott Fujimoto, Herke van Hoof, and David Meger. Addressing function approximation error in actor-critic methods, 2018.

Jean-Bastien Grill, Florian Strub, Florent Altché, Corentin Tallec, Pierre H. Richemond, Elena Buchatskaya, Carl Doersch, Bernardo Avila Pires, Zhaohan Daniel Guo, Mohammad Gheshlaghi Azar, Bilal Piot, Koray Kavukcuoglu, Rémi Munos, and Michal Valko. Bootstrap your own latent: A new approach to self-supervised learning, 2020.

Abhishek Gupta, Vikash Kumar, Corey Lynch, Sergey Levine, and Karol Hausman. Relay policy learning: Solving long-horizon tasks via imitation and reinforcement learning, 2019.

William Harvey, Saeid Naderiparizi, and Frank Wood. Conditional image generation by conditioning variational auto-encoders, 2022.

Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning, 2020.

Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning*, 12(4):307–392, 2019. ISSN 1935-8245. doi: 10.1561/2200000056. URL http://dx.doi.org/10.1561/2200000056.

Minghuan Liu, Menghui Zhu, and Weinan Zhang. Goal-conditioned reinforcement learning: Problems and solutions. *CoRR*, abs/2201.08299, 2022. URL https://arxiv.org/abs/2201.08299.

Xiuyuan Lu, Benjamin Van Roy, Vikranth Dwaracherla, Morteza Ibrahimi, Ian Osband, and Zheng Wen. Reinforcement learning, bit by bit, 2023.

Ashvin Nair, Vitchyr Pong, Murtaza Dalal, Shikhar Bahl, Steven Lin, and Sergey Levine. Visual reinforcement learning with imagined goals, 2018.

Suraj Nair and Chelsea Finn. Hierarchical foresight: Self-supervised learning of long-horizon tasks via visual subgoal generation, 2019.

Soroush Nasiriany, Vitchyr H. Pong, Steven Lin, and Sergey Levine. Planning with goal-conditioned policies, 2019.

Matthias Plappert, Marcin Andrychowicz, Alex Ray, Bob McGrew, Bowen Baker, Glenn Powell, Jonas Schneider, Josh Tobin, Maciek Chociej, Peter Welinder, Vikash Kumar, and Wojciech Zaremba. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

Vitchyr Pong, Shixiang Gu, Murtaza Dalal, and Sergey Levine. Temporal difference models: Model-free deep rl for model-based control, 2020.

Ling Yang, Zhilong Zhang, Yang Song, Shenda Hong, Runsheng Xu, Yue Zhao, Wentao Zhang, Bin Cui, and Ming-Hsuan Yang. Diffusion models: A comprehensive survey of methods and applications, 2024.

Tianhe Yu, Gleb Shevchuk, Dorsa Sadigh, and Chelsea Finn. Unsupervised visuomotor control through distributional planning networks, 2019.